Proceedings of the Third International Workshop on Sustainable Ultrascale Computing Systems (NESUS 2016)
Sofia, Bulgaria

Jesus Carretero, Javier Garcia Blas, Svetozar Margenov
(Editors)

October, 6-7, 2016

Volume Editors

Jesus Carretero
University Carlos III
Computer Architecture and Technology Area
Computer Science Department
Avda Universidad 30, 28911, Leganes, Spain
E-mail: jesus.carretero@uc3m.es

Javier Garcia Blas
University Carlos III
Computer Architecture and Technology Area
Computer Science Department
Avda Universidad 30, 28911, Leganes, Spain
E-mail: fjblas@inf.uc3m.es

Svetozar Margenov
Institute of Information and Communication Technologies
Bulgarian Academy of Sciences
Acad. G. Bontchev Str., Bl. 25A, 1113 Sofia, Bulgaria
E-mail: margenov@parallel.bas.bg

Published by:
Computer Architecture,Communications, and Systems Group (ARCOS)
University Carlos III
Madrid, Spain
http://www.nesus.eu

ISBN: 978-84-617-7450-0

This document also is supported by:

Printed in Madrid — December 2016

Preface

Network for Sustainable Ultrascale Computing (NESUS)

We are very excited to present the proceedings of the Third International Workshop on Sustainable Ultrascale Computing Systems (NESUS 2016), a workshop created to reflect the research and cooperation activities made in the NESUS COST Action (IC1035) (www.nesus.eu), but open to all the research community working in large/ultra-scale computing systems. It was held in Sofia (Bulgaria) on October 6-7, 2016.

The goal in scalable and sustainable technology today is to have on the one hand large parallel supercomputers, named Exascale computers, and on the other hand, to have very large data centers with hundreds of thousands of computers coordinating with distributed memory systems. Ultimately, NESUS idea is to have both architectures converge to solve problems in what we call ultrascale. Ultrascale systems combine the advantages of distributed and parallel computing systems. The former is a type of computing in which many tasks are executed at the same time coordinately to solve one problem, based on the principle that a big problem can be divided into many smaller ones that are simultaneously solved. The latter system, in both grid and cloud computing, uses a large number of computers organized into clusters in a distributed infrastructure, and can execute millions of tasks at the same time usually working on independent problems and big data. The applications of these systems and the benefits they can yield for society are enormous, according to the researchers, who note that this type of computing will help conduct studies about genomics, new materials, simulations of fluid dynamics used for atmospheric analysis and weather forecasts, and even the human brain and its behavior.

The goal of the NESUS Action is to establish an open European research network targeting sustainable solutions for ultrascale computing aiming at cross fertilization among HPC, large scale distributed systems, and big data management. Ultrascale systems are envisioned in NESUS as large-scale complex systems joining parallel and distributed computing systems that will be two to three orders of magnitude larger that today's systems. The EU is already funding large scale computing systems research, but it is not coordinated across researchers, leading to duplications and inefficiencies. The network will contribute to glue disparate researchers working across different areas and provide a meeting ground for researchers in these separate areas to exchange ideas, to identify synergies, and to pursue common activities in research topics such as sustainable software solutions (applications and system software stack), data management, energy efficiency, and resilience. Some of the most active research groups of the world in this area are members of this NESUS Action. This Action will increase the value of these groups at the European-level by reducing duplication of efforts and providing a more holistic view to all researchers, it will promote the leadership of Europe, and it will increase their impact on science, economy, and society.

The scientific objective of NESUS is to study the challenges presented by the next generation of ultrascale computing systems to enhance their sustainability. These systems, which will be characterized by their large size and great complexity, present significant challenges, from their construction to their exploitation and use. We try to analyze all the challenges there are and see how they can be studied holistically and integrated, to be able to provide a more sustainable system. The challenges that this type of computing poses affect aspects such as scalability, the programming models used, resilience to failures, energy management, the handling of large volume of data, etc. One of the NESUS goals is to find the way that all solutions that are proposed can be transmitted to user applications with the minimum possible redesign and reprogramming effort.

The project began last March with 29 European countries, but at present consists of 39 European countries and six countries from other continents. It now involves nearly 200 scientists, almost 40% of whom are young researchers, because

one essential goal of these Actions is to promote and create an ecosystem of scientists who can work on these matters in the European Union in the future.
This Action, which concludes in 2018, aims to produce a catalogue of open source applications that are being developed by the members of the Action and which will serve to demonstrate new ultrascale systems and take on their main challenges. In this way, anyone will be able to use these applications to test them in their systems and demonstrate their level of sustainability.

Prof. Jesus Carretero
University Carlos III of Madrid
NESUS Chair

December 2016

NESUS

Network for Sustainable Ultrascale Computing

COST Action IC1305 2014 | 2018

Aim

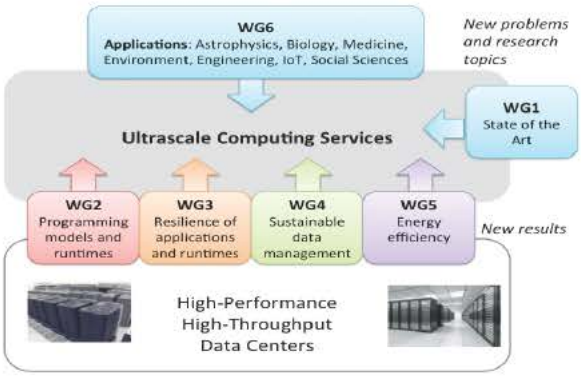
- Coordinate European efforts for proposing realistic solutions addressing major challenges of building sustainable Ultrascale Computing Systems (UCS) with a holistic approach.

To:

1. Increase EU research in the field of sustainable ultrascale computing.
2. Give coherence to the European ICT research agenda related to sustainability.
3. Build a multi-disciplinary forum for cross-fertilization of ideas for sustainable ultrascale computing.

Information and Communication Technologies (ICT)

Scientific Workplan



Topics

- WG1: New techniques to enhance sustainability holistically.
- WG2: Promoting new sustainable programming and execution models in the context of rapidly changing underlying computing architecture.
- WG3: Innovative techniques to deal with hardware and system software failures or intentional changes within the complex system environment.
- WG4: Study data management lifecycle on scalable architectures in a synergistic approach to pave the way towards sustainable UCS.
- WG5: Explore the design of metrics, analysis, frameworks and tools for putting energy awareness and energy efficiency at the next stage.
- WG6: Identify algorithms, applications, and services amenable to ultrascale systems and to study the impact of application requirements on the sustainable ultrascale system design.

Activities

- Research activities though WGs
- Set up collaborations through STSM and internships
- Training schools and PhD forum
- Meetings for WGs and MC
- Dissemination and cooperation with industry and stakeholders.
- Publications, conference organization, industry seminars, …

Participating countries: 45

EU COST countries: 33

AT, BA, BE, BG, BO, CH, CY, DE, DK, EE, EL, ES, FI, FR, HR, HU, IE, IL, IT, LT, LU, MK, MT, NL, NO, PL, PT, RO, SI, SK, SE, TR, UK

NNC countries: 6

AL, AM, MD, MO, RU, UA

Global Collaboration: 6

AU, CA, CO, IN, MX, US

Contact details

Chair of the Action
Jesus Carretero
jesus.carretero@uc3m.es

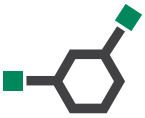
Website
www.nesus.eu

COST is supported by the EU RTD Framework Programme

TABLE OF CONTENTS

Third NESUS Workshop (NESUS 2016)

91 List of Authors



cost

Elastic Cloud Services Compliance with Gustafson's and Amdahl's Laws

Sasko Ristov, Radu Prodan

University of Innsbruck, Austria
sasko@dps.uibk.ac.at, radu@dps.uibk.ac.at

Marjan Gusev

Ss. Cyril and Methodius University, Skopje, Macedonia
marjan.gushev@finki.ukim.mk

Dana Petcu

West University of Timisoara, Romania
petcu@info.uvt.ro

Jorge Barbosa

University of Porto, Portugal
jbarbosa@fe.up.pt

Abstract

The speedup that can be achieved with parallel and distributed architectures is limited at least by two laws: the Amdahl's and Gustafson's laws. The former limits the speedup to a constant value when a fixed size problem is executed on a multiprocessor, while the latter limits the speedup up to its linear value for the fixed time problems, which means that it is limited by the number of used processors. However, a superlinear speedup can be achieved (speedup greater than the number of used processors) due to insufficient memory, while, parallel and, especially distributed systems can even slowdown the execution due to the communication overhead, when compared to the sequential one. Since the cloud performance is uncertain and it can be influenced by available memory and networks, in this paper we investigate if it follows the same speedup pattern as the other traditional distributed systems. The focus is to determine how the elastic cloud services behave in the different scaled environments. We define several scaled systems *and we model the corresponding performance indicators. The analysis shows that both laws limit the speedup for a specific range of the input parameters and type of scaling. Even more, the speedup in cloud systems follows the Gustafson's extreme cases, i.e. insufficient memory and communication bound domains.*

Keywords Load, Distributed systems, Performance, Superlinear speedup.

I. Introduction

Cloud computing has introduced a rapid change in the way of designing the architecture of today's services from license-based to as-a-service-based services [1]. The main driver was influenced by its multitenancy, on demand elastic resources and underlined virtualisation technology. Customers do not buy the license to own the software service, but instead they pay only for the period of its usage. In order to satisfy the customers' demands, cloud providers offer various types of resources, usually represented as virtual machine (VM) instances, each with specific computing, memory and storage capacity. The customers expectation is that the performance will follow the price.

Due to its elasticity and the linear pay-as-you-go model, the cloud is preferred platform both for the granular and scalable algorithms, especially if they are low communication-intensive, such as scientific applications [2, 3]. Still, many applications are data-intensive, and provide a high throughput. This is a huge challenge in the cloud because the data

transfer between the cloud compute nodes and storage is a bottleneck [4]. Despite the additional virtualisation layer, the superlinear speedup is also reported, both for granular [5], and scalable application types [6].

However, despite all these benefits, the main challenge for the customers is whether they will get the performance proportionally to the cost. That is, whether the cloud elastic resources comply with the Amdahl's Law [7] for the fixed size problems and with Gustafson's Law [8] for the fixed time problems. In this paper, we model several performance indicators, to determine if both laws hold for the cloud elastic services, each in a specific region. Although one can argue that the web services are scalable and therefore will comply with the Gustafson's law only, our analysis and taxonomy show in which scaled systems the Amdahl's law limits the speedup.

The rest of the paper is organised in several sections as follows. The speedup definitions and limits in parallel and distributed systems are described in Section II. Section III defines a taxonomy for scaled systems in cloud, in order to adapt the existing Amdahl's and Gustafon's laws for elastic services. According to the taxonomy, Section IV models the speeds and speedups for each scaled system for various load regions. Despite the virtualisation layer, the cloud environment can achieve even a superlinear speedup, as discussed in Section V. Section VI discusses further challenges. Finally, we conclude the paper in Section VII.

II. Background

Parallel and distributed systems offer a powerful environment that can be utilised for two main purposes: to speed up some algorithm's execution or to execute some big data problems. The former is useful in order to finish with execution in proper time; for example, we need today a weather forecast for tomorrow, and it is unusable to have it tomorrow. Distributed systems are used to solve a problem that cannot be even started on a single machine due to hardware limitation. Both parallel and distributed systems have more computing resources than a nominal single-machine or a single-processor system. In this paper, we will denote these systems as *scaled systems*.

Two main laws exist in the computer architecture, or more broader in the parallel and distributed systems, which limit the speedup that can be achieved, according the algorithm's type: Amdahl's and Gustafson's laws. Both laws target the speedup, but analyse it from different perspectives.

Let's analyze a scaled system with a scaling factor p. The metric for measuring the performance of a scaled computing system is the *speed* $V(p)$, which defines the amount of work $W(p)$ performed for a period of time $T(p)$, as presented in (1). Another important metric is the *normalised speed* $NV(p)$, which measures the amount of work per processor per time period, as defined in (2).

$$V(p) = \frac{W(p)}{T(p)} \tag{1}$$

$$NV(p) = \frac{V(p)}{p} = \frac{W(p)}{T(p) \cdot p} \tag{2}$$

To compare the scaled with a non-scaled system, one should evaluate the *speedup* $S(p)$, which is defined as a ratio of speeds of the scaled system and the best speed in the non-scaled system, as presented in (3).

$$S(p) = \frac{V(p)}{V(1)} = \frac{W(p)/T(p)}{W(1)/T(1)} \tag{3}$$

The amount of work is constant for fixed-time algorithms, which transfers (3) to (4), where $T(1)$ denotes the execution times of the best sequential algorithm, while $T(p)$ the execution time of the algorithm on scaled system with scaling factor p.

$$S(p) = \frac{T(1)}{T(p)} \tag{4}$$

Amdahl's Law limits the size of the problem and limits the speedup to the value $S_{max}(p) = 1/s$, where s is the serial part of the algorithm. As one can observe, the maximal theoretical value for the speedup is limited and does not depend on the number of processors. On the other side, Gustafson reevaluated the Amdahl's Law by showing that a *linear* speedup $S_{max}(p) = p$ can be achieved if a problem is executed within a fixed time. He achieved a near linear speedup of impressive 1000, when running a problem on 1024 cores [9].

III. A taxonomy of scaled systems

Usually both Gustafson's and Amdahl's laws are intended for granular algorithms, which can be divided into many independent sub-tasks and then scattered to a scaled system for execution. This section presents a taxonomy that we define for scaled systems in order to adapt both laws to be appropriate for cloud elastic services. We are using a similar approach for scalable algorithms, such as web services, with an exception that in this case, the parallelisation is usually not conducted by some API, but on the web server level.

III.1 Definition and classification of scaled systems

Let a *nominal system* be a cloud system that possesses R cloud resources and is loaded with L requests, as presented in Fig. 1 a). One would expect the nominal system can handle L amount of work in a specific time period using R resources. For example, the load can be represented as the number of requests for some service which is hosted in a group of VMs that have a total of R cloud computing resources.

Our classification is based on scaling both the requirements of a cloud computing system and cloud resources. Therefore, we define two scaling types in cloud computing: scaling the load (requirements) and scaling the (cloud computing) resources. A different scaling factor can be used for requirements and resources. Without loosing generality, we assume that the resources can scale $p > 1$ times, while the load, N times.

We will use the notation $xRyL$ to define the taxonomy of scaling the cloud systems where $x, y \in \{n, s\}$ are the indicators in front of each scaling parameter. The s presence indicates that the corresponding parameter is scaled and n if it is not scaled. According to this notation, the nominal system is defined as a *non-scaled Resources non-scaled Load*, and denoted as ($nRnL$) system.

If the customers want to improve the performance of a service hosted in a cloud system, they need to scale the cloud resources. In case of scaling the load, there are two possibilities for the customer: either to retain the cost (keep the same cloud resources), but degrade the performance, or to scale the cloud resources and to retain the same performance. Consequently, we will define three different scaled systems when only one or both parameters are scaled with Definitions 1, 2, and 3. All three scaled systems are presented in Fig. 1 b), c) and d).

Definition 1 ($sRnL$ scaled system) *The $sRnL$ scaled cloud system denotes a cloud system with* scaled Resources non-scaled Load, *that is, a system with p times more cloud resources.*

Definition 2 ($nRsL$ scaled system) *The $nRsL$ scaled cloud system denotes a cloud system with* non-scaled Resources scaled Load, *that is, a cloud system with N times more load.*

Definition 3 ($sRsL$ scaled system) *The $sRsL$ scaled cloud system denotes a system with* scaled Resources scaled Load, *that is, a system with p times more cloud resources and N times more load.*

The next examples explain these types of scaled systems. Assume that a web server hosted in a cloud instance with one CPU core ($R = 1$) can handle 100 client requests ($L = 100$) in acceptable response time. According to the Gustafson's Law one would expect that the performance would be doubled when the same 100 requests are executed on a server using resources with double the capacity ($sRnL$).

Another example is when both the load and resources are scaled, that is, the expected response time of 200 requests to be executed on a server with doubled resources should be the same as the nominal case - 100 requests executed on one CPU core ($sRsL$). And, for $nRsL$, the response time should be doubled if the load is increased to 200 requests.

III.2 Expected performance of scaled cloud systems

Let PF be a function (5) that returns the performance P of a system with specific resources R and loaded with a load L. Then, (6) defines the expected performance for all three scaled systems.

$$P = PF(R, L) \tag{5}$$

$$\begin{aligned} \textbf{sRnL}: & p \cdot P = PF(p \cdot R, L); \\ \textbf{nRsL}: & \frac{1}{p} \cdot P = PF(R, p \cdot L); \\ \textbf{sRsL}: & P = PF(p \cdot R, p \cdot L). \end{aligned} \tag{6}$$

This classification of scaling the system can help in determination of performance limits of a system.

IV. Theoretical analysis of scaled systems compliance with Amdahl's and Gustafson's laws

In order to adapt both laws for elastic services, this section introduces the work per resource and models the speedup for scaled systems compared to nominal (non-scaled) systems.

IV.1 Modeling the resource utilization

In order to find the *resource utilization* W we determine how much average work (load) L is sent to a particular resource R and calculate it according to (7) as a ratio of the load and the number of resources. This parameter shows the average "speed" of performing a particular work per resource. To simplify the notation, in the remaining text we will use abbreviations omitting the R and L identifications, such as nn for the $nRnL$ system.

$$W_{nn} = \left\lceil \frac{L}{R} \right\rceil; \tag{7}$$

Next, distribute all L requests in groups, such that each group has R requests to map each request to a specific computing resource. Then, in each time period, R requests will

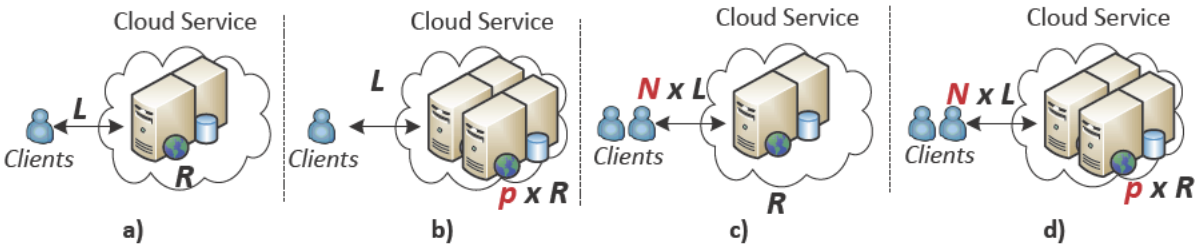


Figure 1: Nominal (non-scaled) system nRnL a) and three possible scalings b) sRnL, c) nRsL and d) sRsL

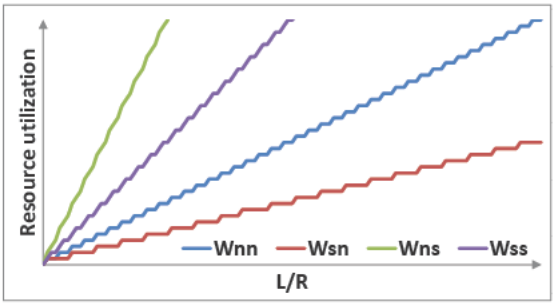


Figure 2: Resource utilization of the scaled systems

be scattered among available R resources, which yields to (7). Note that the last group will have $L \mod R$ requests, and not all the resources will be loaded with requests.

By applying the corresponding parameters for the three scaled systems into (7), then (8) defines the resource utilizations (expressed as work per resource) W_{sn}, W_{ns}, W_{ss}, corresponding to the $sRnL$, $nRsL$ and $sRsL$ systems.

$$W_{sn} = \left\lceil \frac{L}{p \cdot R} \right\rceil; \quad W_{ns} = \left\lceil \frac{N \cdot L}{R} \right\rceil; \quad W_{ss} = \left\lceil \frac{N \cdot L}{p \cdot R} \right\rceil; \tag{8}$$

Observing the definitions for all four work per resource, one can conclude that all of them depend on the ratio L/R, representing a nominal value of a work per resource. Therefore, we will continue with an analysis that determine the impact of the L/R ratio over the work per resource and the speedup.

Fig. 2 presents the resource utilization (work per resource) of the nominal and the three scaled systems. Observe that all resource utilizations are in a shape of stairs, with a linear trendline. The stairs' effect appears due to roundup function in (7) and (8). Obviously, the resources of $sRnL$ scaled system have a smaller amount of work, while the other two scaled systems have more, compared to the nominal system. Fig. 2 is obtained for a value of $p/N = 2/4 = 1/2$. We must note that W_{ss} and W_{ns} can change their place in Fig. 2, depending whether the p/N ratio is greater of smaller than 1.

IV.2 Modeling the speedup

In order to measure the impact of scaling, (9) defines the *speedup* S_{sn}, S_{ns}, S_{ss} for all scaled systems, when compared to the nominal one.

$$S_{sn} = \frac{\left\lceil \frac{L}{R} \right\rceil}{\left\lceil \frac{L}{p \cdot R} \right\rceil}; \quad S_{ns} = \frac{\left\lceil \frac{L}{R} \right\rceil}{\left\lceil \frac{N \cdot L}{R} \right\rceil}; \quad S_{ss} = \frac{\left\lceil \frac{L}{R} \right\rceil}{\left\lceil \frac{N \cdot L}{p \cdot R} \right\rceil}; \tag{9}$$

Fig. 3 visually presents all three speedups as a function of the L/R ratio, along with their trendlines. The speedup S_{sn} shows an increasing trend starting from 1, and saturates up to the scaling factor p when $\frac{L}{R} \to \infty$. Whenever p is a divisor of L/R, the speedup achieves its maximal value of p, regardless of the L/R ratio value, as depicted with point $A(i \cdot p, p)$. Although, seemingly, it looks like that the $sRnL$ scaled system relies on the Gustafson's Law, it is true only when the L/R ratio is huge. For smaller L/R ratio, scaling the resources will not provide a greater speedup, which is exactly the Amdahl's Law.

The speedup S_{ns} starts from $S_{ns} = 1$ and saturates its value to the point $1/N$ for greater ratio L/R. Obviously, although $S_{ns} < p$, and seemingly it is a sublinear speedup, in fact this is a slowdown. This is expected since the load is increased compared to the nominal system. According to Fig. 2, the work per resource is increased, which will reduce the performance. The tradeoff for this performance suffering is the constant cost that the customer should pay for renting the resources.This is the resource underprovisioning and overutilisation.

Similar behavior for speedup is present for the the $sRsL$ scaled system. The only difference is that the trendline saturates to the value $S_{ss} = p/N$. Let us discuss about the

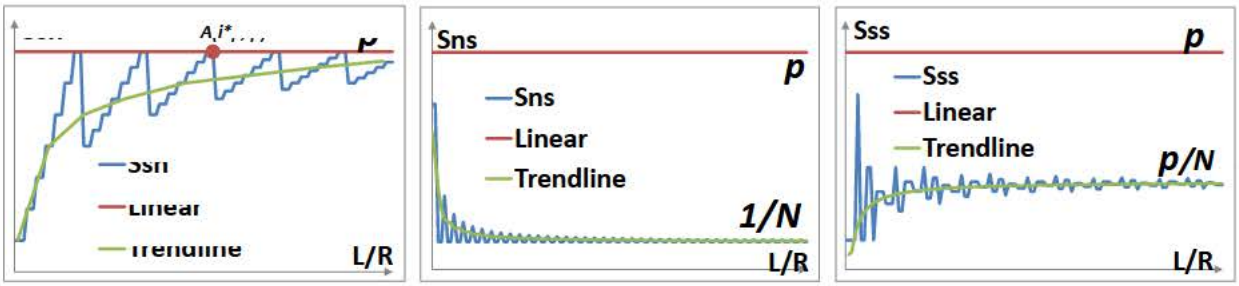


Figure 3: Speedup for various scaled systems as a function of L/R

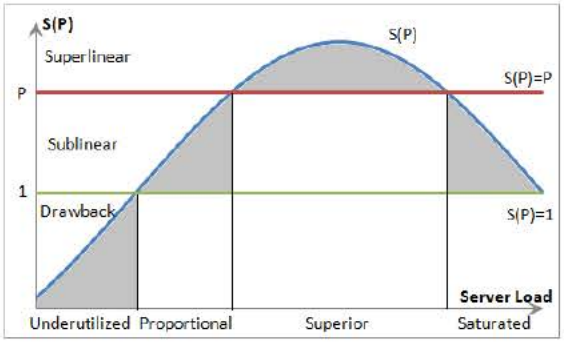


Figure 4: Expected speedup of a scaled system

expected performance of this scales system (6) and the calculated speedup (9). If the resource scaling follows the scaling of the work, then the speedup will saturate to 1, which means that this scaled system scales ideally. However, if $p > N$, which means overprovisioning and underutilisation, we are getting closer to the $sRnL$ scaled system, in the hands of Amdahl's Law.

IV.3 Going beyond the speedup limits

Although previous subsection presents the theoretical limits of the speedup in various scaled systems, several examples are reported where the speedup went beyond the limits, that is, a *superlinear* speedup is achieved.

Ristov et al. [10] have modeled the performance behavior of services classifying five sub-domains of speedup:

- *Drawback* $0 < S(p) < 1$ - worse performance for the scaled system;
- *No Speedup* $S(p) = 1$ - the new scaled system reports the same performance as unscaled;
- *Sublinear* $1 < S(p) < p$ - similar to Gustafson's scaled speedup;
- *Linear* $S(p) = p$ - maximum limited speedup according to the Gustafson's scaled speedup;
- *Superlinear* $S(p) > p$ - greater performance than the limited speedup.

The expected sub-domains, along with four regions of server load (underutilised, proportional, superior and saturated) are presented in Fig. 4 [10]. The first three regions are already expressed in our theoretical analysis. The superior region is of interest in this paper, and it appears because the web server with one core will enter in its saturation mode, while the scaled system is still in its normal mode. The superior region ends when the scaled system enters the saturation mode. This means that theoretical speedups of all scaled systems do not saturate to the constant value, but they will start to falling down when the ratio L/R will increase up to some level when even the scaled resources cannot handle the load in the appropriate time.

However, the reported results show that this model works only for both computation-intensive and memory-demanding web services, while the computation-intensive only web services achieve a sublinear speedup, that is, those systems have four regions.

V. Analysis of a superlinear speedup in cloud environment

Nowadays, cloud computing is being increasingly used for high-performance and high throughput applications. It allows the customers to rent, for example, 1000 processors and execute a certain task at peak times, instead of building their own data center. Since the cloud's pricing strategy is linear, and expected speedup is also linear, it seems that customers will be charged fairly. However, there are several

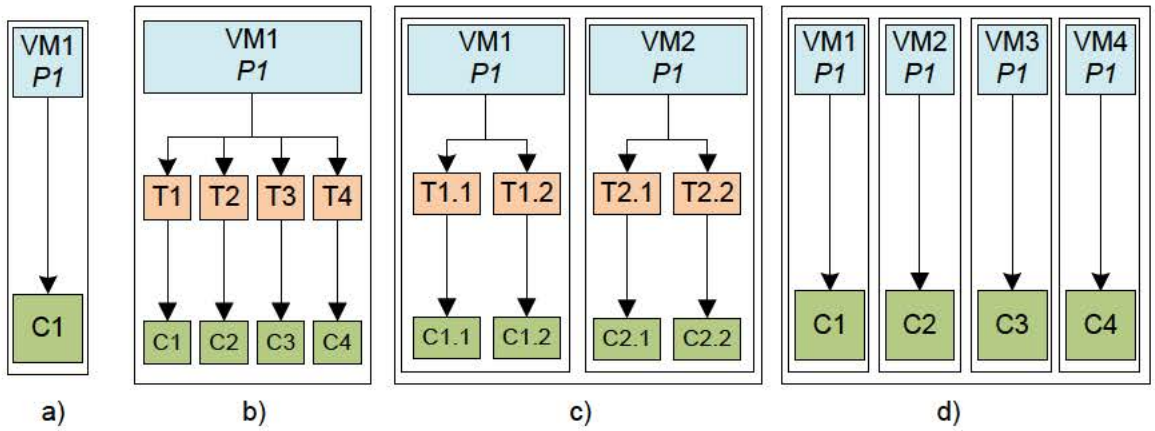


Figure 5: Example of b) Vertical, c) Diagonal, and d) Horizontal scaling of nominal resources a) for granular algorithms

cases where the superlinear speedup is achieved, despite the virtualisation layer.

V.1 Granular algorithms

Customers can scale their rented resources horizontally, vertically or diagonally in the cloud for the *sRnL* or *sRsL* scaled systems. If the original configuration maps one process to a VM instance hosted on a processor with one CPU core, as presented in Fig. 5 a), then Fig. 5 b), c) and d) present the three possible cloud scalings. The horizontal scaling presented in Fig. 5 d) increases the number of same VM instances and maps a separate process (with a single thread) to a different VM instance. The vertical scaling presented in Fig. 5 b) increases the number of CPU cores per VM (resized VM) and maps separate threads of a single process to a different core on the same VM instance. A combination of the both scaling types yields a diagonal scaling presented in Fig. 5 c). To realise the vertical and diagonal scaling, the customer should use some API for parallelization, such as OpenMP, which will create parallel threads.

Few papers are reporting a superlinear speedup in both the horizontal and vertical scaling. A superlinear speedup is reported for cache-intensive algorithms in [5] for the case of vertical scaling. Although sequential execution utilises cache in the sequential execution more, the superlinear speedup can be achieved also for horizontal scaling in the cloud, according [11]. The authors of [12] have determined that the cloud environment can handle the cases when the problem size can be fitted in the last level cache memory better leading to a superlinear speedup.

V.2 Scalable algorithms

Fig. 6 presents three possible ways how to scale from nominal system to the *sRnL* scaled system. Similarly, there is a horizontal, vertical and diagonal scaling. The main difference here is the necessity of cloud load balancer that will schedule the load among many end-point VM instances for horizontal and diagonal scaling.

Ristov et al. [6] proposed a scalable architecture for e-ordering system hosted in the cloud. Their experiments reported a significant superlinear speedup of 20 for the *SRNL* system with a scaling factor $p = 4$ analyzing the response time. The superlinearity also appeared for the throughput, i.e. the percentage of responses for given number of requests per second.

V.3 Superlinear speedup of a load balancer

Distributing the load to several end-point servers is much easier in the case of a load balancer, which will forward the load to the servers by using a particular algorithm. Since it is a new layer, it adds a small amount of delay, which usually depends on the load. However, Ristov et al. [13] developed a balancer with a region that achieves a superlinear speedup when using more end-point servers.

Even more, when it is used with only one end-point web server, the results are still better than the case without it. This appears because of the connections that are opened to the end-point web servers are maintained without opening a new connection for each client request. This reduces the number of resources for creating a new session compensating the delay produced due to the additional layer.

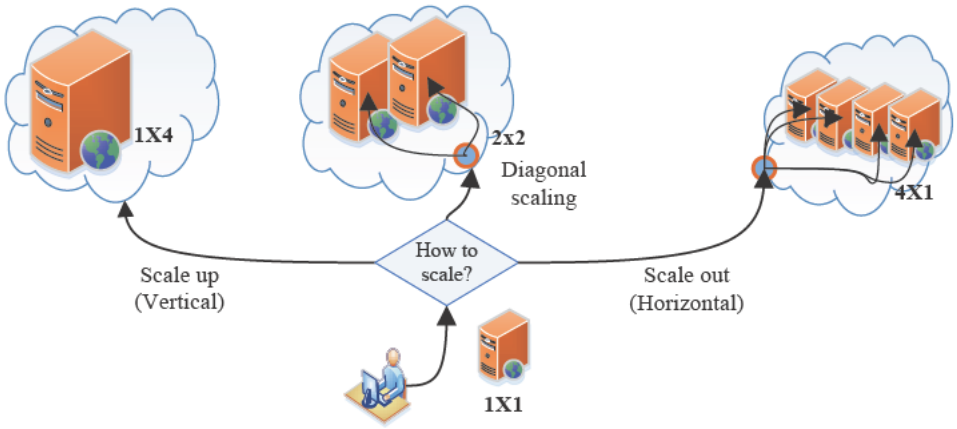


Figure 6: Example of vertical, diagonal and horizontal scaling of nominal system for elastic cloud services

VI. Discussion

This section discusses further challenges and issues connected to the performance of cloud elastic services.

VI.1 Granular and scaling algorithms' similarity

Although granular and scaling algorithms seem to be totally different, as they are executed in different environment, they still have several similarities. In case of persistent algorithms, more memory is needed for both algorithm types. That is why a sublinear speedup is achieved for computation-intensive web services [10].

We can observe another similarity. The granular algorithms executed on tightly coupled processors correspond to vertical scaling for scaling algorithms, while executed on loosely coupled processors to horizontal scaling.

These similarities can be used to utilise the pros of one algorithm type for the other, which could also lead to a superlinear speedup. In this context, we can follow the idea of Trang et al. [14] who introduced a model for parallel execution of web services promising that both approaches can be used together.

VI.2 New challenge: How to scale?

Cache-intensive granular algorithms, whose data reuse complexity is similar with the problem size, will benefit from a bigger cache. Many Intel's multiprocessors use a marketing trick based on a huge L3 smart cache. However, one can easily check that it is not shared among all cores, but only among part of them. For example, 6MB of total 12MB cache is shared between each group of two cores. In this case, the vertical scaling will utilise more the last level cache. AMD multiprocessors usually use a smaller L3 cache, but it is shared among all cores of the multiprocessor. Therefore, depending on the algorithm, appropriate processor and scaling type should be chosen in order to achieve the best speedup, potentially superlinear.

On the other hand, today's cloud elastic resources can also be scaled in different ways: horizontally, vertically or diagonally, each of which can offer various performance and possibility for achieving a superlinear speedup. The vertical scaling provides a better speedup, but the horizontal offers more flexible scaling of resources, which can minimise the cost. Although, using a load balancer in front of the siblings, a superlinear speedup can be achieved due to reduced number of opened connections.

VI.3 Further challenges

Achieving a superlinear speedup does not necessarily mean that customers will obtain the maximum achievement. In the workflow executions in parallel and distributed systems, customers usually use bi-objective optimizations to minimise the makespan and cost. These two parameters are opposite one to another. Minimising the makespan produces a greater cost and vice versa.

Cloud computing customers can set a deadline for the execution requiring a minimal cost, rather than a minimal makespan [15]. In these cases, budget constraints and reducing the race for the speedup can yield the reduced cost for the

execution. For example, although a superlinear speedup can be achieved in a Windows Azure cloud for matrix multiplication when VM instances with Windows operating system are used, Linux VM instances achieved better performance cost trade-off because they are cheaper.

On the other side, there is a risk of cloud resources performance variation, different setup time [16], instance failure over the time [17] and difficulty to predict the performance, which will harden the resource provisioning [18]. Additional problem in modeling the elastic cloud services' behavior is the uncertainty in cloud provisioning and VM instability. For example, Dejun et al. [19] reported a performance uncertainty of up to 8% in Amazon EC2.

Increasing the budget by duplicating the tasks on more than one instance could mitigate those risks, in order to meet the deadline [20]. Sometimes, using a bigger instance executes the task faster, rather than waiting several minutes for the deployment time to start another smaller, but an appropriate instance, which reduces the turnaround time of an activity [21].

Not all offered pricing models are linear. For example, some providers charge the customers on hourly based policy, while others charge some amount at the beginning plus charge then per smaller time unit. For example, Google charges the usage for the first 10 minutes, and then per minute. Also, Google have recently introduced the non-linear model by including the VM usage sustainability. All these issues impact on choosing the appropriate scaled system for a specific cloud elastic service.

VII. Conclusion

Cloud services are scalable and can be executed in both the parallel and distributed systems by load balancing among the scaled resources. This balancing reduces the amount of work per resource, which speedups the average execution time. Predicting and measuring the performance of such services is very difficult because the real cloud elastic service receives client requests with an unknown distribution probability function. Also, they are hosted on an unpredictable resource provisioning, which makes their modeling almost impossible. Still, by using the upper and lower limits of the speedup, one can compare the fairness of the pricing model.

The Amdahl's and Gustafson's laws set limits on the speedup that a scaled system achieves, but usually for granular algorithms. However, even in the traditional parallel and distributed systems, there are many cases when these laws are disproved due to the nature of the algorithms, hardware and software architecture. The uncertainty of the VM provisioning and performance, along with many differences between the scaling and granular algorithms, questions their compliance with both laws. However, our modeling and theoretical analysis showed that cloud elastic services are compliant with both laws. Such general laws are push drivers to enable the technologies and pull drivers that lead toward technical innovations. This chain of push and pull drivers makes the positive feedback that enables the overall technology continual development.

Acknowledgment

This work is partially supported by the European Union's Horizon 2020 research and innovation programme under the grant agreements 644179 ENTICE: dEcentralized repositories for traNsparent and efficienT vIrtual maChine opErations (first two authors) and 643946, CloudLightning: Self-organizing, self-managing Heterogeneous Clouds (fourth author).

The authors would like to acknowledge networking support by the COST programme Action IC1305, Network for Sustainable Ultrascale Computing (NESUS).

References

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al., A view of cloud computing, Communications of the ACM 53 (4) (2010) 50–58.

[2] A. Gupta, D. Milojicic, Evaluation of hpc applications on cloud, in: Open Cirrus Summit (OCS), 2011 Sixth, 2011, pp. 22–26. doi:10.1109/OCS.2011.10.

[3] S. A. Tsaftaris, A scientist's guide to cloud computing, Computing in Science Engineering 16 (1) (2014) 70–76. doi:10.1109/MCSE.2014.12.

[4] L. Liu, M. Zhang, Y. Lin, L. Qin, A survey on workflow management and scheduling in cloud computing, in: Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on, 2014, pp. 837–846. doi:10.1109/CCGrid.2014.83.

[5] M. Gusev, S. Ristov, Superlinear speedup in Windows Azure cloud, in: Cloud Networking (IEEE CLOUDNET), 2012 IEEE 1st International Conference on, Paris, France, 2012, pp. 173–175.

[6] S. Ristov, F. Dimitrievski, M. Gusev, G. Armenski, Scalable system for e-orders as a service in cloud, in: International Conference on Computer as a Tool (IEEE EUROCON 2015), Salamanca, Spain, 2015, pp. 1–6.

[7] G. M. Amdahl, Validity of the single-processor approach to achieving large scale computing capabilities, in: AFIPS Conference Proceedings, Vol. 30, AFIPS Press, Reston. Va., Atlantic City, N.J., 1967, pp. 483–485.

[8] J. L. Gustafson, Reevaluating Amdahl's law, Communication of ACM 31 (5) (1988) 532–533.

[9] J. Gustafson, G. Montry, R. Benner, Development of parallel methods for a 1024-processor hypercube, SIAM Journal on Scientific and Statistical Computing 9 (4) (1988) 532–533.

[10] S. Ristov, M. Gusev, G. Velkoski, Modeling the speedup for scalable web services, in: A. M. Bogdanova, D. Gjorgjevikj (Eds.), ICT Innovations 2014, Vol. 311 of Advances in Intelligent Systems and Computing, Springer International Publishing, 2015, pp. 177–186.

[11] M. Gusev, S. Ristov, Resource scaling performance for cache intensive algorithms in Windows Azure, in: F. Zavoral, J. J. Jung, C. Badica (Eds.), Intelligent Distributed Computing VII, Vol. 511 of SCI, Springer International Publishing, 2014, pp. 77–86.

[12] M. Gusev, S. Ristov, The optimal resource allocation among virtual machines in cloud computing, in: Proceedings of The 3rd International Conference on Cloud Computing, GRIDs, and Virtualization (CLOUD COMPUTING 2012), Nice, France, 2012, pp. 36–42.

[13] S. Ristov, K. Cvetkov, M. Gusev, Implementation of a scalable L3B balancer, Scalable Computing: Practice and Experience 17 (2) (2016) 79–90. doi:10.1109/TE.2014.2327007.

[14] M. X. Trang, Y. Murakami, T. Ishida, Cloud Computing: 6th International Conference, CloudComp 2015, South Korea, Springer International Publishing, 2016, Ch. Modeling Parallel Execution Policies of Web Services, pp. 244–254.

[15] M. A. Rodriguez, R. Buyya, Deadline based resource provisioning and scheduling algorithm for scientific workflows on clouds, IEEE Transactions on Cloud Computing 2 (2) (2014) 222–235. doi:10.1109/TCC.2014.2314655.

[16] M. Mao, M. Humphrey, A performance study on the vm startup time in the cloud, in: Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on, 2012, pp. 423–430.

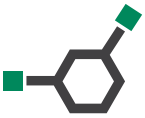
[17] F. Wu, Q. Wu, Y. Tan, Workflow scheduling in cloud: a survey, The Journal of Supercomputing 71 (9) (2015) 3373–3418. doi:10.1007/s11227-015-1438-4.

[18] A. Tchernykh, U. Schwiegelsohn, V. Alexandrov, E.-G. Talbi, Towards understanding uncertainty in cloud computing resource provisioning, Procedia Computer Science 51 (2015) 1772 – 1781.

[19] J. Dejun, G. Pierre, C.-H. Chi, Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops: International Workshops, ICSOC/ServiceWave 2009, Stockholm, Sweden, November 23-27, 2009, Revised Selected Papers, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, Ch. EC2 Performance Analysis for Resource Provisioning of Service-Oriented Applications, pp. 197–207.

[20] R. N. Calheiros, R. Buyya, Meeting deadlines of scientific workflows in public clouds with tasks replication, IEEE Transactions on Parallel and Distributed Systems 25 (7) (2014) 1787–1796. doi:10.1109/TPDS.2013.238.

[21] M. Mao, M. Humphrey, Scaling and scheduling to maximize application performance within budget constraints in cloud workflows, in: Parallel Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on, 2013, pp. 67–78. doi:10.1109/IPDPS.2013.61.



cost

Exploring OpenMP Accelerator Model in a real-life scientific application using hybrid CPU-MIC platforms

Kamil Halbiniak*, Lukasz Szustak*, Alexey Lastovetsky** and Roman Wyrzykowski*

* Czestochowa University of Technology, Poland
{khalbiniak, lszustak, roman}@icis.pcz.pl

** University College Dublin, Ireland
alexey.lastovetsky@ucd.ie

Abstract

The main goal of this paper is the suitability assessment of the OpenMP Accelerator Model (OMPAM) for porting a real-life scientific application to heterogeneous platforms containing a single Intel Xeon Phi coprocessor. This OpenMP extension is supported from version 4.0 of the standard, offering an unified directive-based programming model dedicated for massively parallel accelerators. In our study, we focus on applying the OMPAM extension together with the OpenMP tasks for a parallel application which implements the numerical model of alloy solidification. To map the application efficiently on target hybrid platforms using such constructs as omp target, omp target data and omp target update, we propose a decomposition of main tasks belonging to the computational core of the studied application. In consequence, the coprocessor is used to execute the major parallel workloads, while CPUs are responsible for executing a part of the application that do not require massively parallel resources. Effective overlapping computations with data transfers is another goal achieved in this way. The proposed approach allows us to execute the whole application 3.5 times faster than the original parallel version running on two CPUs.

Keywords Intel MIC, hybrid architecture, numerical modeling of solidification, heterogeneous programming, OpenMP Accelerator Model, task and data parallelism

I. Introduction

Heterogeneous platforms combining general-purpose processors with specialized computing accelerators (e.g., GPU or Intel Xeon Phi) offer ample opportunities for accelerating a wide range of applications [1]. However, realizing these performance potentials remains a challenging issue.

A promising way to exploit capabilities of heterogeneous platforms is the OpenMP Accelerator Model [2] offered by the OpenMP standard, starting with version 4.0. It provides an unified directive-based programming model encompassing both CPUs and accelerators. The major advantage of this extension is applying the same programming model for the whole application, that allows decreasing the code complexity and increasing its portability.

The main goal of this paper is evaluation of the OpenMP Accelerator Model for porting a real-life scientific application to platforms equipped with a single Intel Xeon Phi coprocessor. In this study, we focus on the effective utilization of new mechanisms provided by the OpenMP 4.0 standard for parallelization of the computational core of the studied application. The proposed approach allows us to execute computations 3.49x faster than the original parallel code that uses two CPUs. This application was already studied in our previous work [3], where we developed a methodology that utilized the dedicated Intel Offload interface.

This paper is organized as follows. Section 2 gives an overview of the OpenMP Accelerator Model, while Section 3 introduces the numerical model of solidification, which is based on the generalized finite difference method. The next section describes the idea of parallelizing the solidification application on hybrid platforms with OpenMP 4.0 mechanisms, while Section 5 shows performance results achieved by the proposed approach. Section 6 concludes the paper.

II. Overview of OpenMP Accelerator Model

OpenMP is the directive-based programming standard designed for programming shared-memory systems. [2]. Starting with version 4.0, OpenMP provides a mechanism called

OpenMP Accelerator Model (OMPAM in short). It aims at simplifying the issue of programming heterogeneous computing platforms with many-core accelerators such as Intel MIC or GPU. This model assumes that a computing platform is equipped with multiple target devices connected to the host device.

The execution model of OMPAM is based on a host-centric view, where the host device transfers (offloads) data and computations to target devices before execution, using **target** construct. By default, code regions offloaded to accelerators are executed using a single thread, that can spawn multiple threads after encountering an appropriate parallel construct.

Using accelerators requires usually to perform data transfers. To reduce the total amount of allocations and deallocations of device memory, OMPOA provides **target data** construct, which creates the data region for a device. This gives the possibility for sharing the same data between multiple target regions. OMPAM allows defining the data movements between the host and the device before and after the execution of the offloaded region by using **map** clause. The transfers of data are possible using the following attributes: **to**, **from** and **tofrom**. These attributes allows the implicit initialization of device buffers and determination of the direction of data copying [2]. At the same time, **map** clause with **alloc** attribute is used when the explicit allocation of device memory is required.

Another important directive of OMPAM is **target udpate**. It allows the synchronization of buffers between the host and device environments. This construct can be used only inside the device data region. The direction of update is specified using two clauses: **to** and **from**, which provide the list of synchronized buffers consistent with variables in the device data region. Another new directive, **declare target**, is used to determine regions of the source code mapped to the device, with the resulting binaries called from the **target** region.

An example of source code written using the OpenMP Accelerator Model is shown in Listing 1.

```
#pragma omp target data map(to: n, B[0:n]) \
    map(alloc: A[0:n], C[0:n])
for(int t=0; t<num_steps; ++t) {
    #pragma omp target map(to: n, B[0:n]) \
        map(to: C[0:n]) map(from: A[0:n])
    #pragma omp parallel for
    for (int i=1; i<n-1; ++i ) {
        A[i]=C[i]*(B[i-1] + B[i] + B[i+1]);
    }
    //rest of code
}
```

Listing 1: Offloading computations in OpenMP Accelerator Model

Comparing to alternative tools that allow for programming accelerators, OMPAM provides a reasonable support for multiple heterogeneous platforms, through a growing amount of compilers. This increases the interest of developers in using OpenMP as a promising way to achieve the code portability between platforms.

III. Application: Modeling Solidification

The phase-field method is a powerful tool for solving interfacial problems in materials science. It has mainly been applied to solidification dynamics, but it has also been used for other phenomena such as viscous fingering, and fracture dynamics. The number of scientific papers related to the phase-field method grows since the 90 years of XX century, reaching for the last 7 years more than 400 positions (according to the SCOPUS database) [4].

In the numerical example studied in this paper, a binary alloy of Ni-Cu is considered as a system of the ideal metal mixture in the liquid and solid phases. The numerical model refers to the dendritic solidification process in the isothermal conditions with constant diffusivity coefficients for both phases. In the model, the growth of microstructure during the solidification is determined by solving a system of two PDEs which define the phase content ϕ (Fig. 1) and concentration c of the alloy dopant. The solutions of these PDEs are obtained on the basis of the generalized finite difference method and explicit scheme of calculations, so the resulting numerical algorithm [3] belongs to the group of forward-in-time iterative algorithms. In the model studied in the paper, values of ϕ and c are calculated for grid nodes uniformly distributed across a square domain. However, this model can be also used for irregular grids.

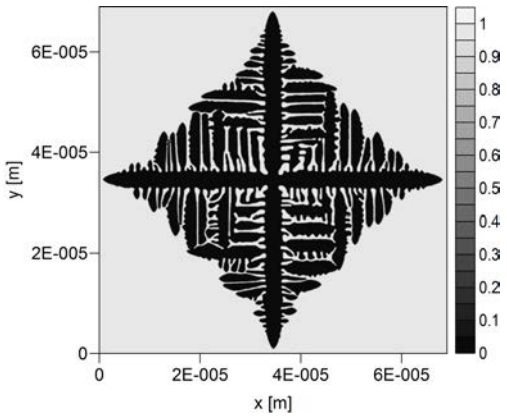


Figure 1: Phase content for the simulated time $t = 2.75 \times 10^{-3}s$

IV. Parallelization of the Application on Hybrid CPU-MIC Platforms

IV.1 Task Parallelization with OpenMP 4.0

In the studied application, computation are interleaved with writing partial results to a file. In the basic version (Fig.2a), parallel computations are executed for subsequent time steps, and writing results to the file is performed after the first time step, and then after every package of $R = 2000$ time steps.

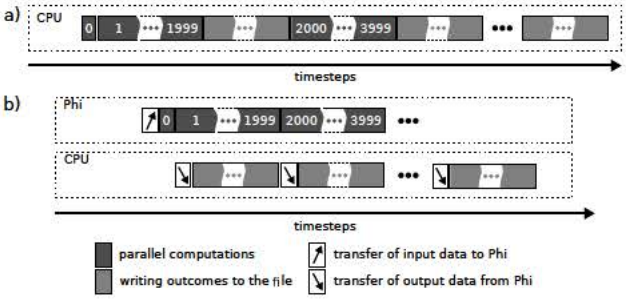


Figure 2: Adapting the application to platforms with Intel MIC [3]

In the proposed approach (Fig. 2b), the Intel Xeon Phi coprocessor is utilized to perform parallel computations, while the host processor is responsible for executing the rest of the application that not required massively parallel resources. As a result, writing outcomes to the file is assigned to the CPU, while the coprocessor is utilized for parallel computations in subsequent time steps. At the beginning, all the input data are transferred from the CPU to the coprocessor, which then starts computations for the first time step. After finishing it, all the results are transferred back to the CPU. During this transfer, the coprocessor starts computations for the next package of R time steps. At the same time, CPU begins writing results to the file, immediately after receiving outcomes from the coprocessor. Such a scheme is repeated for every package of R time steps. A critical performance challenge here is to overlap workload performed by the coprocessor with data movements. To reach this goal, data transfers between the CPU and coprocessor, writing data to the file, as well as computations have to be performed simultaneously.

To offload data and computations to the coprocessor, we use two major constructs of OMPAM: **omp target data** and **omp target**. By default, OpenMP 4.0 does not provide a mechanism for the asynchronous execution of **omp target** region. In consequence, the thread calling this pragma is stopped before completing the execution by the accelerator. Therefore to ensure overlapping computations with writing outcomes to the file, we propose to use the OpenMP task parallelism. This mechanism can be successfully applied to parallelize these operations. As a result, two tasks are distinguished in the proposed approach: (i) running parallel computations on the coprocessor, and (ii) writing results to the file. These tasks are spawned inside the parallel region by the master thread using **omp task** construct.

When applying the proposed idea of adapting the solidification application to heterogeneous platforms (Fig. 2b), the usage of task parallelism requires to provide an adequate task synchronisation, since results cannot be written to the file before completing computations for the previous package of R time steps. Therefore, the synchronization points occur after every package. To ensure the effective synchronization of tasks, we use **omp taskwait** pragma.

Overlapping data transfers with computations requires also to apply the double-buffering technique. The first buffer is utilized for performing parallel computations, while the second one is for providing data movements and writing outcomes to the file. To transfer data from the coprocessor to CPU during computations, **omp target update** construct is adopted.

IV.2 Data Parallelization

The original CPU version of the application uses the OpenMP standard to utilize cores/threads, based on the OpenMP construction `#pragma omp parallel for`. Since the Intel Xeon Phi coprocessors supports OpenMP, the application code can be rather easily ported to this platform. To ensure the best overall performance without significant modifications in the source code, we use several compiler-friendly optimizations, empirically determine the best OpenMP setup for the loop scheduling and set appropriate affinity.

The utilization of vector processing is crucial for ensuring the best performance on Intel Xeon Phi. The quickest way to achieve this goal is the compiler-based automatic vectorization. However, in the studied case the innermost loop cannot be vectorized safely, mainly because of data dependencies. To solve this problem, we propose to change slightly the code by adding temporary vectors responsible for loading data from the irregular memory region, and than providing SIMD computations (see our previous works [3, 5]).

V. Performance Results

In this work, we use a platform equipped with two Intel Xeon E5-2699 v3 CPUs (Haswell-EP), and Intel Xeon Phi 7120P coprocessor (Knight Corner). All the tests are performed for the double-precision floating-point format with 110 000 time steps, and 2D grid containing 4 000 000

Table 1: Performance results for different versions of the application

Version	Tasks		Time	Speedup
	Data writing	Parallel computing		
original	CPU	CPU	641 min 32 sec	-
offload	CPU	MIC	183 min 41 sec	3.49x
OpenMP	CPU	MIC	187 min 43 sec	3.42x

nodes (2000 nodes along each dimension), using the Intel icpc compiler (v.15.0.2) with optimization flag -O3.

Table 1 presents the comparison of the performance for: (i) original CPU parallel version of the application running on two CPUs with 18 cores each, (ii) offload-based code for hybrid CPU-MIC platforms, developed in our previous work [3], and (iii) the proposed version developed in this work using OpenMP 4.0 mechanisms. Both the second and third versions implement the proposed scheme of adapting the solidification application to platforms with a single Intel Xeon Phi coprocessor.

The total execution time of the original code is the sum of the execution times necessary for performing parallel computations and writing partial results to the file. The proposed approach allows us to hide more than 99% of data movements behind computations, for both the offload- and OpenMP-based versions, and finally accelerate the whole application of about 3.5x. Comparing the execution times for the OpenMP- and offload-based codes, we can see a very low difference of 2.2% in favour of the offload interface.

VI. Conclusion and Future Works

This paper shows that the OpenMP Accelerator Model is the promising tool for porting a real-life scientific application to heterogeneous platforms with many-core accelerators such as Intel Xeon Phi. The performance results obtained for the offload-based and OpenMP-based versions, executed on the platform with a single coprocessor, confirms that the OpenMP Accelerator Model allows achieving a quite similar performance as the Intel Offload Model dedicated directly for Intel MIC architectures. It is expected that the potential of using OpenMP for current and future architectures will be manifested for a wide range of applications.

The primary direction of our future work is to take advantage of all the computing resources (multiple CPUs and multiple MICs) of heterogeneous platforms, for executing the application. The OpenMP Accelerator Model will be compared against the offload-based [5] and hStreams-based [6, 7] solutions, taking into account both the performance and productivity. We plan to explore new features available in version 4.5 of OpenMP [8], since version 4.0 does not provides the asynchronous offload mechanisms, which are necessary for the efficient utilization of all the resources available in such multi-device heterogeneous platforms.

Acknowledgments

This research was conducted with the support of COST Action IC1305 (NESUS), as well as the National Science Centre (Poland) under grant no. UMO-2011/03/B/ST6/03500. The authors are grateful to the Czestochowa University of Technology for granting access to Intel Xeon Phi coprocessors provided by the MICLAB project no. POIG.02.03.00.24-093/13 (http://miclab.pl).

References

[1] L. Szustak, K. Rojek, R. Wyrzykowski, and P. Gepner. Toward efficient distribution of MPDATA stencil computation on Intel MIC architecture. In *Proc. 1st Int. Workshop on High-Performance Stencil Computations (HiStencils' 14)*, pages 51–56, 2014.

[2] OpenMP Application Programming Interface. http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf.

[3] L. Szustak, K. Halbiniak, A. Kulawik, J. Wrobel, and P. Gepner. Toward parallel modeling of solidification based on the generalized finite difference method using Intel Xeon Phi. *LNCS*, 9573:411–412, 2016.

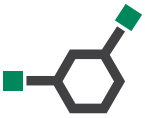
[4] T. Takaki. Phase-field Modeling and Simulations of Dendrite Growth. *ISIJ International*, 54 (2):437–444, 2014.

[5] L. Szustak et al. Porting and optimization of solidification application for CPU-MIC hybrid platforms. *Int. Journal of High Performance Comp. Applications*, (accepted to print).

[6] Chris J. Newburn et al. Heterogeneous streaming. *IPDPSW, AsHES*, 2016.

[7] L. Szustak et al. Using hStreams Programming Library for Accelerating a Real-Life Application on Intel MIC. In *ICA3PP 2016 Conference*, (accepted to print).

[8] M. Klemm. Heterogeneous Programming with OpenMP 4.5. https://www.scc.kit.edu/downloads/sca /Heterogeneous%20Programming%20with%20OpenMP%204.5.pdf.



cost

Geographical Competitiveness for Powering Datacenters with Renewable Energy

Simon Holmbacka*, Enida Sheme†, Sébastien Lafond*, Neki Frasheri†

*Factulty of Science and Engineering, Åbo Akademi University Turku, Finland
firstname.lastname@abo.fi
†Polytechnic University of Tirana Tirana, Albania
firstname.lastname@fti.edu.al

Abstract

In this paper we analyze the feasibility of using renewable energy for powering a data center located on the 60th parallel north. We analyze the workload energy consumption and the cost-energy trade-off related to available wind and solar energy sources. A wind and solar power model is built based on real weather data for three different geographical locations, and The available monthly and annual renewable energy is analyzed for different scenarios and compared with the energy consumption of a simulated data center. We show the impact different data center sizes have on the coverage percentage of renewables, and we discuss the competitiveness of constructing datacenters in different geographical location based on the results.

Keywords Green energy, datacenter, simulation, geographical locations

I. Introduction

The global energy price and tighter restrictions on energy production has led to a higher utilization of green energy, which is produced from completely carbon neutral sources. One of the latest trends in reducing the carbon footprint of data centers is powering the datacenters with renewable sources of energy. This course is being encouraged by the advances of renewable technologies and continuously decreasing renewable energy costs. Renewable energy sources have become an interesting option for large scale server farms, and initiatives such as Google Green[1] and Facebook Sustainability[2] have been taken to decrease the carbon footprint both for ecological and monetary reasons. Recently, the location of large scale server farms has shifted to the nordic countries above the 60th parallel because of a cooler climate, which in turn reduces the cooling costs for such datacenters. The energy required for computation and the infrastructure must, however, be delivered from the electric grid, preferably generated by renewable energy. This poses a challenges for northern countries because of the large variation in available solar energy throughout the year. While the summer period provides from 18 to 20 hours of sunlight, the winter period provides merely a few hours – this from a very shallow angle of reflection. The lack solar energy can be compensated with other sources such as wind energy, but the total cost of powering the data center must be sufficiently low in order to stay competitive to other geographical locations.

We present in this paper a thorough analysis of the feasibility of powering large scale datacenters in geographical locations above the 60th parallel north with renewable energy. The analysis contains simulations of different datacenters executing various workloads and the requirements in green energy production for different geographical locations. In contrast to previous work we compare different geographical locations in terms of both available renewable energy and required datacenter capacity for satisfying the end user. We also provide an a competitiveness factor between different geographical locations for the feasibility of powering a datacenter with renewable energy sources.

II. Related Work

Real implementations of green data centers. Researchers at Rutgers University [9] present Parasol and GreenSwitch, a research platform for a green data center prototype. It consists of GreenSwitch software running over a real hardware data center, Parasol. Its aim is reducing the total data center cost by properly managing workloads and available energy sources for maximum benefits. It also studies the space requirements and capital costs of self-generation with wind and solar energy. Similarly, [18] presents Blink, a physical implementation of using intermittent power to supply a cluster of 10 laptops by two micro wind turbines and two solar

[1]https://www.google.com/green/
[2]https://sustainability.fb.com/en/

panels, supported by small 5-minute energy buffer batteries. HP Labs has built a 4 servers data center partially powered by solar panels [6]. The data center is powered by the grid when no solar energy is available. In contrast to these real implementations, we simulate different scenarios to adapt different data center sizes and workload, with thousands of physical and virtual machines. Thus, we have a broader view of the impact they have on the amount of required renewable energy.

Simulators for green data centers. Michael Brown and Jose Renau present ReRack [2], an extensible simulation infrastructure that can be used to evaluate the energy cost of a data center using renewable energy sources. It also includes an optimization module to find the best combination of renewable sources that minimize cost. Yanwei Zhang et al [24] have developed GreenWare, a middleware system that conducts dynamic request dispatching to maximize the percentage of renewable energy used to power a network of distributed data centers, based on the time-varying electricity prices and availabilities of renewable energy in their geographical locations. It also considers different prices per kWh solar and wind energy have in different geographical data center locations, distributing the workload accordingly for lowest overall cost possible. In our study, instead, we do not develop a simulator but focus on studying the relation between quantity of renewable energy sources and data center energy consumption for a certain coverage with renewable. We take into consideration different workload scenarios.

Managing the workload in green data centers. Rutgers University proposes GreenSlot [8], a parallel batch job scheduler for a data center powered by a solar panel and the electrical grid (as a backup). It can predict the amount of solar energy that will be available in the near future, and schedules the workload to maximize the renewable energy consumption up to 117% while meeting the jobs' deadlines. Likewise, GreenHadoop [10], a GreenSlot successor, represents a MapReduce framework seeking to maximize the renewable energy consumption within the jobs' time bounds. Ghamkhari et. al. [7] offer an optimization-based workload distribution framework for Internet and cloud computing data centers with behind-the-meter renewable generators in order to save energy. This is achieved by better resource utilization taking into account several impacting factors like computer servers' power consumption profiles, data center's power usage effectiveness, availability of renewable power at different locations, price of electricity at different locations. Aksanli et al. [1] design a new data center job scheduling methodology that effectively leverages green energy prediction, which enables the scaling of the number of jobs to the expected energy availability. They develop a discrete event-based simulation platform for applying this methodology in a data center consisting of hundreds of servers. Liu et al. [14] evaluate the impact of geographical load balancing and the role of storage in decreasing the brown energy costs. The authors also suggest the optimal mix of renewables to power Internet-scale systems using (nearly) entirely renewable energy. They use homogeneous servers and 1 week HP Labs workload traces, while we base our simulations on heterogeneous sets of servers and a more generalized workload trace, which is automatically generated by uniformly distributed time, duration and type of the user requests. Beside this, their selected data center countries represent locations with high solar energy production, but we give contribute in studying the renewable energy capacity on the 60th parallel north where sun intermittent nature is more significant. Workload management is not part of our analysis in this paper but we plan to address it in our future work. Furthermore, our simulation input parameters are intended to resemble real data centers as closely as possible in terms of size and power, and provides clear guidelines for green data centers' designers.

Managing energy sources for green data centers. Researchers at University of Florida, IDEAL Lab, propose iSwitch [13], a novel dynamic load power tuning scheme for managing intermittent renewable energy sources. The study introduces a renewable energy utilization (REU) metric, defined as (PL / PR) x 100%, where PL is the amount of renewable power utilized by the load and PR is the total renewable power generation. Instead, we study another parameter called Minimal Percentage Supply (MPS) which is the percentage of total energy consumption that can be driven by available renewable energy, given as renewable energy divided by energy consumption converted in percentage.

Studies on battery usage in data centers have also been conducted by [23],[11], [22] to optimize the energy management and minimize the energy cost. This study does not include the usage battery as an energy storage, but initiates a discussion on the impact of energy storage to the cost model and how to integrate such a factor when modelling a data center.

III. Available renewable energy

In our study, we consider renewable energy produced by wind turbines and solar panels. To simulate the system and analyze the results, we must first model both the consumption and production rate of our datacenter and energy sources. Since the weather and the season directly influences the production of renewable energy, we must utilize a

weather model to predict the production rate. We must also use a simulation environment realistic enough to accurately model the energy consumption of a datacenter.

For this we have chosen three geographically distributed locations for investigating the feasibility study of using renewable energy. Firstly we chose Turku, Finland at 60° latitude as our reference because of the increased interest in constructing datacenters in northern countries. Secondly we chose Crete, Greece at 35° latitude because of its typically solar intense southern European climate. And thirdly we selected, Ilorin, Nigeria at 8.5° latitude to cover the equatorial extreme point. For each of these locations we are going to analyze the generation of renewable energy using solar- and wind power. In this section we describe the total amount of renewable energy produced in one year for our chosen geographical location, and in Section V we compare the production of energy to the consumption.

Data collection We collected the weather data from different sources. The weather data for Finland was collected from a weather station located at Åbo Akademi University in Turku, Finland [3]. Sensors in this weather station [12] measure a variety of meteorological data, including wind speed and direction, temperature, humidity, barometric pressure, rain and solar radiation.

For the non-local geographical locations, we collected the solar radiation data from `http://solrad-net.gsfc.nasa.gov/`. The website contains freely available data from solar radiation such as various forms of radiation data and the energy intensity measured by pyranometers which are compatible with the weather data from the Finnish location. All data is sampled by at least the granularity of one hour. For describing the production rate of a solar panel, we acquire the data containing the solar power radiance on a horizontal 1 m^2 solar panel, and we calculated the produced power in Watts describe later in our power model. We acquired the local wind speed data from the same weather station in Turku [3], and from `https://mesonet.agron.iastate.edu/` for the non-local data. The wind speed data was converted to meter/seconds $[m/s]$ from the non-local weather data in order to match with the local weather data. All data is sampled by at least the granularity of one hour.

III.1 Solar power model

The solar power model is constructed by analyzing the solar radiation obtained from the weather data, and by considering the following trigonometrical aspects of the radiation angle and practical aspects of the solar panel:

- *Angle tilt:* The power incident on a solar panel depends not only on the power contained in the sunlight, but also on the angle between the module and the sun. Referring to [4], we calculate the optimal angle at which a solar array should be tilted in order to achieve maximum energy through the year. Different geographical locations with different latitude are operating optimally using different angle tilt with respect to the horizontal plane. In all cases we assumed that the angle tilt is fixed throughout the year for all geographical locations, but we assume that the solar array tracks the sun on the vertical axis (east to west). Equation 1 shows the power generation of a 1 m^2 solar panel as:

$$P_{solar} = P_{solar\_h} \times sin(\alpha + \beta)/sin(\alpha) \quad (1)$$

  where P_{solar_h} is the solar radiance in the horizontal plane we already have from weather data, α is the sun elevation angle through the year and β is the tilt angle of the module measured from horizontal plane, 45°. The value for α is calculated according to Equation 2:

$$\alpha = 90 - \phi + \delta \quad (2)$$

  where ϕ is the latitude (60°) and δ is the declination angle computed in Equation 3 as:

$$\delta = 23.45° \times sin[360 \times (284 + d)/365] \quad (3)$$

  where d is the day of the year.

- *Solar panel efficiency:* is the percentage of the sunlight energy that is actually transformed into electricity because of limitations in the solar panel cells. Today's solar panel technology (multi-crystalline silicon) efficiency value varies from 15% up to 18% – which is the record of 2015 [19]. Therefore, we multiply all hourly solar energy values with the coefficient 0.18 in order to achieve realistic data.

- *Solar inverter efficiency:* is the efficiency of the inverter connected between the solar panel cells and the AC grid. According to [20], the average coefficient of the DC-AC power converting today is 95%. Thus, we take this value into account to assure accurate and realistic power values.

III.2 Wind power model

The wind power model describes the power generation from the wind turbines in the system. To produce the wind energy we have chosen a HY 1000 [21], 5 blade wind turbine generating a peak output power of 1200 W. We chose this model because of its availability on the market and because

of its suitable size for our datacenter. The wind power model is constructed by taking into consideration the following key features:

- *Wind turbine power curve:* According to the power profile in the technical specifications, we constructed the mathematical model of power as a function of wind speed. Equation 4 describes the power production of a wind turbine as follows:

$$P_{wind} = 1151 \times exp(-((wind_{speed} - 14.28)/6.103)^2) \quad (4)$$

  where $wind_{speed}$ is the wind speed in $[m/s]$. The parameters in Equation 4 were obtained by using curve fitting tools in Matlab.

- *Wind inverter efficiency:* according to [5], wind turbine power converters typically reach an efficiency of 95%. Thus, we multiply this value with the prediction of the power model to provide a more accurate and realistic model.

Finally, the total renewable power model is given as:

$$P_{renewable} = P_{solar} + P_{wind} \quad (5)$$

which is simply the sum of the total solar and total wind production. As a result of the above processing and calculations, we have available total renewable (solar and wind) energy information in hourly granularity for the whole year.

IV. Renewable Energy Analysis

To illustrate the impact of the weather conditions on the renewable energy production, we used the previously defined power models for the wind turbine and solar panels to calculate the total sum of the produced energy for each month of the year. The weather data was collected at the following points in time:

Finland: January 1, 2012 – December 31 2012
Greece: January 1, 2006 – December 31 2006
Nigeria: January 1, 2011 – December 31 2011
Even though the data origins from different years, we assume that the average over one year will provide a sufficiently accurate and comparable result. Figure 1 shows the energy production of a 1 m^2 solar panel in Finland, Greece and Nigeria Figure 2 shows the energy production of one 1200W wind turbine, and Figure 3 shows the total sum of both energy sources throughout one year in each location.

The very predictable weather in Ilorin, Nigeria shows an almost constant solar energy production in Figure 1. Since all days throughout the year is approximately 12h, there is only a slight difference between winter and summer months. The low point is in July due to weather conditions such as rainy seasons with an extensive cloud coverage. In Crete, Greece, the 35°latitude and solar intensity provides a large but varying energy production. The winter months in Greece provide far less sunlight than the summer months, and have therefore a lower energy production than Nigeria. However, the days in the summer months are longer, and the solar energy produced in one day exceeds the energy productions of Nigeria even if the intensity of the solar radiation is larger in Nigeria. The most varying results are measured in the Finnish location. The winter months produce almost no solar energy because of a very short time of sunlight during the day. On the other hand, during the summer the solar energy production can exceed both Greece and Nigeria; in this case during May and June because of the long duration of sunlight during the day. Table 1 finally shows the energy values in kWh for each of these extreme points for total, solar and wind energy.

Also the wind speed is relatively constant in Nigeria throughout the year as seen in Figure 2. The wind speed is relatively low in most months with the exception of a slight increase during August and September. The wind speed in Greece is, on the other hand, very strong in the early months

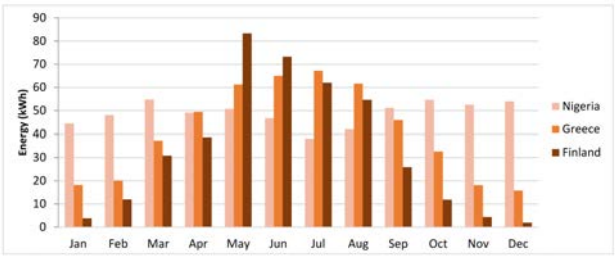


Figure 1: Solar energy produced by 1 m^2 solar panel in three geographical locations during one year

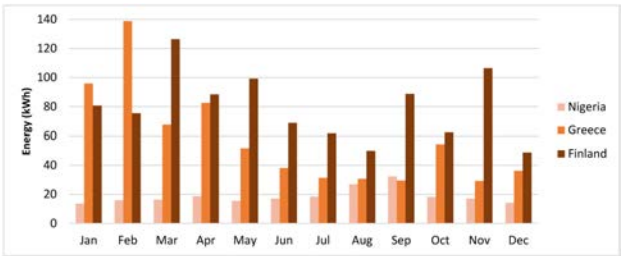


Figure 2: Wind energy produced by a 1200W wind turbine in three geographical locations during one year

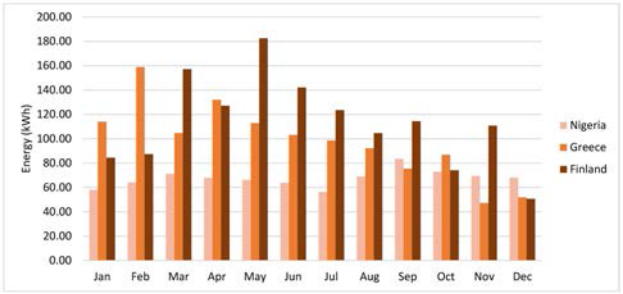


Figure 3: Combination of Solar- and wind energy produced in three geographical locations during one year

Table 1: Total renewable, solar- and wind energy (kWh) extreme months values, from 1 m^2 solar panel and a 1200W wind turbine

	Min month	Min energy	Max month	Max energy
Finland				
Solar	Dec.	1.89	May	83.34
Wind	Dec.	48.71	Mar.	126.36
Total	Dec.	50.6	May	182.52
Greece				
Solar	Dec.	15.75	Jul	67.12
Wind	Nov.	29.12	Feb.	138.80
Total	Nov.	47.21	Feb	158.83
Nigeria				
Solar	Jul.	37.98	Mar.	54.78
Wind	Jan.	18.90	Sep.	44.85
Total	Jul.	56.38	Sep	83.45

of the year, also seen in Figure 2, with a maximum in February. This high wind speed causes almost a possible 140 kWh of energy to be produced with the aforementioned wind turbine. It is about 7x higher than Nigeria and almost twice as high as the related wind production in Finland. During the summer month, the wind production in Greece is lower and hits the minimum about 4x lower than the wind production in February. The wind speed in Finland is typically more randomized, with a slight decrease during the summer months as seen in Figure 2. Overall though the year, the wind energy generation is highest in Finland compared to the other locations, even during summer months.

With this data, we will analyze the extreme months of maximum and minimum wind and solar energy as we intend to investigate the feasibility of using renewable energy sources during one year. Furthermore, the data used to build this model can be applied to other locations by considering different input values of latitude and weather characteristics for the selected area without modifying the core method. All data is freely available at:

https://doi.org/10.5281/zenodo.154401

V. Energy consumption

To account for all the attributes included in causing energy consumption in a datacenter, we used an already made simulation environment,

System simulation We performed the simulations using the Philharmonic simulator developed by Vienna University of Technology, freely available at [15]. It is an open source cloud simulator used to calculate energy consumption and electricity costs for datacenters. The simulator allows the user to input configuration parameters such as the number of physical machines (PM), virtual machines (VM) and internal specification parameters such as clock speed, RAM size etc. Virtual Machines are virtual entities running over the physical machines and performing workload tasks. The cloud control algorithm decides on scheduling the workload using VM migrations and frequency scaling of the physical machines to control the power dissipation[3]. The workload is modelled with user requests uniformly distributed in time and duration[17]. Figure 4 illustrates the overview of the Philharmonic simulator. A given workload and cloud server settings are taken as input after which the tool simulates the scheduling of the workload on the defined server cloud.

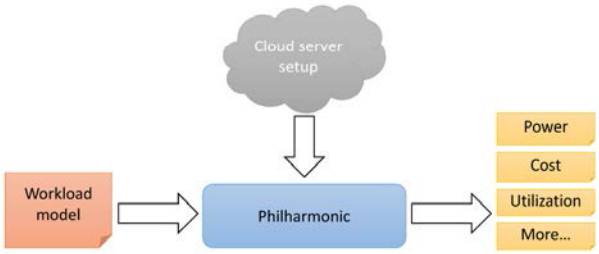


Figure 4: The Philharmonic simulator uses a cloud server setup and a defined workload to calculate power dissipation, cost, utilization and other parameters as a function of time

We used 3 different data center sizes to observe the proportion by which they impact energy consumption. The PMs are configures with 1-4 CPU cores and 16-32 GB RAM, to model a heterogeneous infrastructure. Each VM is configured to have one CPU core and 4-16 GB amount of RAM to vary resource utilization over time. The workload consists of user requests to be handled by VMs. The user requests are

[3]The power model of the Philharmonic simulator was developed during a NESUS STSM at TU Wien May 2015 and is to appear in IEEE Transactions 2016

generated randomly by uniformly distributing the creation time and their duration. Each of the requests can either ask for a new VM to be booted or an existing one to be deleted. The specifications of the requested VMs were modelled by normally distributing each resource type, i.e 4-16 GB of RAM. Further details on the simulator can be found in [16] and [17].

The total duration of the simulation was set to 1 week, with 1 hour step size in order to be compatible with the energy production data. The simulation step size was selected based on the available weather data input of solar and wind energy, so that we can compare hourly available renewable and consumed energy. Finally, the cloud control algorithm decides on the suitable VM migrations and frequency scaling of the physical machines to make the scenario as realistic as possible. The best cost fit frequency scaling (BCFFS) cloud controller described in [17] was used.

Consumed energy We defined input scenarios of 500 to 2500 PMs in the Philharmonic simulator, with a step size of 1000 PMs. The number of virtual machines was chosen 2 fold the number of physical machines for each simulation in order to replicate a realistic scenario. We replicate results of one week for every week of the year, assuming that the workload weekly pattern is homogeneously distributed over the year. As a result, Table 2 shows the total energy consumption for 3 server scenarios during one week and one year.

Table 2: Weekly and annual energy consumption (kWh) of different data center configurations

Nr.	nr. PMs	nr. VMs	weekly energy	annual energy
1	500	1000	2425	126500
2	1500	3000	7285	380200
3	2500	5000	12146	634000

VI. Minimal Percentage Supply

With a model of both energy production (in Section III) and energy consumption (in Section V), we evaluate different scenarios to investigate the feasibility of using renewable energy sources in different geographical locations. We give the notion of a new metric *Minimal Percentage Supply* (MPS), used to determine the data center energy coverage provided from 1 single turbine and 1 m^2 solar panel. Furthermore, we build a quantity model describing the number of wind turbines and solar panels needed to obtain a certain energy coverage in a certain location. MPS is calculated as:

$$MPS = \frac{RenewableEnergyProduction(kWh)}{TotalEnergyConsumption(kWh)} \times 100\% \quad (6)$$

Table 3: MPS annual, maximal and minimal months values in percentage

Scenario Nr.	1	2	3
Finland			
Annual MPS(%)	1.17	0.39	0.23
May MPS(%)	1.88	0.62	0.38
December MPS(%)	0.52	0.17	0.10
Greece			
Annual MPS(%)	1.01	0.34	0.20
May MPS(%)	1.16	0.39	0.23
December MPS(%)	0.54	0.18	0.11
Nigeria			
Annual MPS(%)	0.70	0.23	0.14
May MPS(%)	0.68	0.23	0.14
December MPS(%)	0.70	0.23	0.14

When comparing the energy production with the energy consumption, we determine the MPS value for each data center setting. Table 3 presents the annual, Maximum and Minimum MPS values when applying the respective energy values to Equation 6. The results from Table 3 indicate that the order of magnitude for powering such a datacenter is roughly between 10^2 and 10^3.

We further analyze Scenario 2 datacenter with different MPS values. The MPS of 100%, 75% and 50% for a datacenter of size according to Scenario 2 is illustrated in Figures 5 and 6. The figures illustrate the requirements in both solar and wind power, and various combinations for all three geographical locations. Figure 5 shows the results from May month, since it is the best case scenario for our reference location: Finland. As seen in Figure 5, the least amount of solar or wind power sources are required in Finland to meet the MPS constraints compared to Greece and Nigeria. For example, for an equal distribution of solar- and wind energy a MPS of 75% can be achieved in Finland, while the same configuration only provides 50% MPS in Greece. This is due to the long duration of sunlight in Finland during the summer months in combination with moderate wind production throughout the year.

Figure 6 shows the same MPS configurations as in Figure 5 but for the worst-case month in Finland: December. Since the duration of sunlight during the day is very limited, a very large amount of solar panels are needed to cover the MPS of the Scenario 2 datacenter. For MPS values over 75%, more than 10^4 m^2 of solar panels are needed, which is orders of magnitude more than both Nigeria and Greece. Combining solar power with wind power decreases the number of

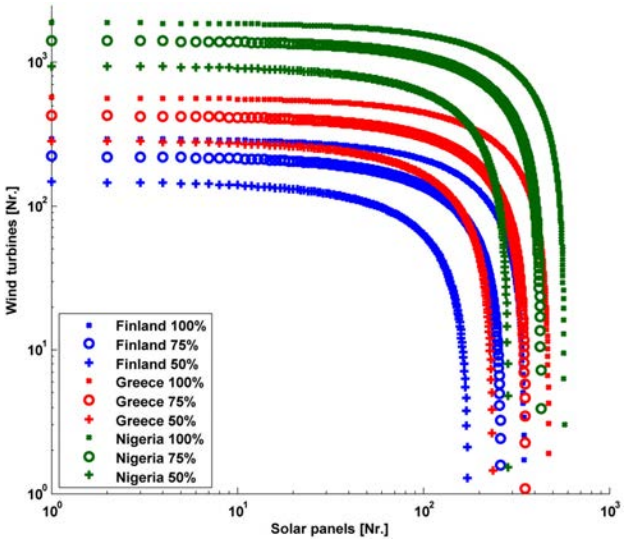


Figure 5: MPS of 100%, 75% and 50% for three geographical locations in May month

required panels, and half an order of magnitude is decreased for a 50/50 configuration. However, with the limited amount of sunlight in December, Finland is only competitive with Greece and Nigeria when using a significantly larger amount of wind turbines.

Figure 7 finally shows the MPS for the annual average

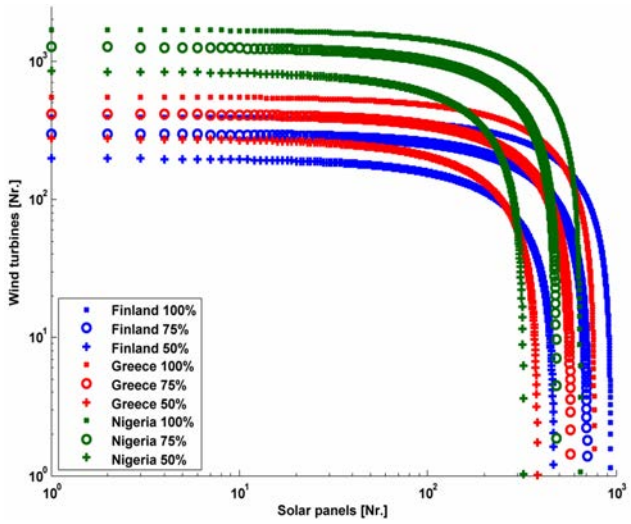


Figure 6: MPS of 100%, 75% and 50% for three geographical locations annually

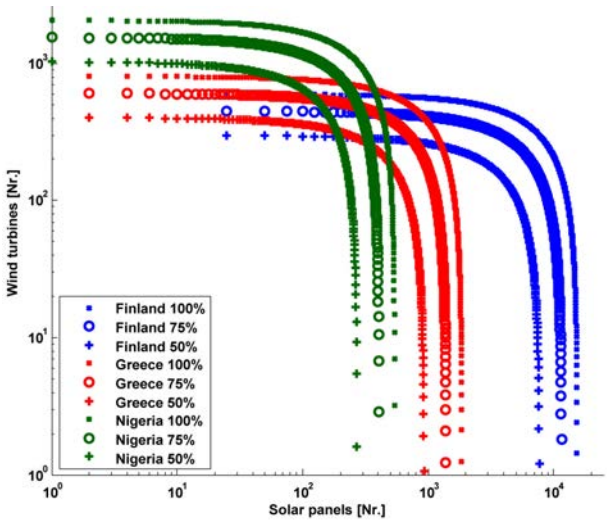


Figure 7: MPS of 100%, 75% and 50% for three geographical locations in December month

energy productions from solar- and wind power. Similarly to the previous figures, the MPS values for 100%, 75% and 50% coverage is shown for all three geographical locations. On an annual average all three locations have the same order of magnitude in energy production, but a few details differ. With the predictable and high intensity sunlight in Nigeria, the annual average energy production from solar power is higher than the wind power. Greece has a more balanced annual energy generation from solar- and wind power. For example using 150 solar panels and 450 wind turbines reaches 100% MPS in Greece while the same configuration in Nigeria results in only 50% MPS. Lastly, Figure 7 shows that Finland reaches the MPS coverage faster than the other locations on an annual basis only if the ratio solar-to-wind is about 1:2.

As seen in the table, there is a 3 fold difference between minimal and maximal MPS values, which clearly indicates different operational costs for producing the same amount of renewable energy during different times of the year. Obviously, we need more physical resources, i.e wind turbines and solar panels, in December to produce same amount of energy compared to May.

VII. Conclusions

In this paper we analyzed the feasibility on competitiveness of powering datacenters with renewable energy at 60°latitude. The energy production on different geographical locations was determined by an energy model based on

real weather data from three geographical different locations, and the energy consumption of different datacenters was simulated on a hourly basis for one year. In order to measure the renewable coverage over the energy consumption of a datacenter a new metric is introduced, called Minimal Percentage Supply (MPS). We built a model for relating the quantity between solar- and wind energy sources in order to achieve a certain MPS coverage with renewables.

Results indicate that the geographical location influences heavily the utilization of renewable energy; for northern latitudes, energy produced from only solar energy is feasible during the summer months, but probably insufficient during the winter months because of the low amount of sunlight during the day. To achieve competitive MPS on a 60°northern latitude on an annual basis, the ratio of solar-to-wind energy must be about 1:2. However during the summer months, competitive (or higher) MPS is achieved on 60 °latitude location independent of the solar-to-wind ratio and using 30-40% less energy generators. During the winter months in Finland, the lack of sunlight naturally deems solar power highly inefficient, and a competitive MPS value is only achieved with a solar-to-wind ratio of roughly 1:1.5. Also, during the winter months in Finland 1.3x the amount power generators must be installed in order to reach the same power generation as the summer months in Finland.

Using this information datacenter designers can determine the feasibility and cost efficiency of constructing data centers powered by renewable energy on a northern latitude.

Acknowledgment

The work presented in this paper has been partially supported by EU under the COST programme Action IC1305: Network for Sustainable Ultrascale Computing (NESUS) and under the Erasmus Mundus programme Euroweb+.

References

[1] Baris Aksanli, Jagannathan Venkatesh, Liuyi Zhang, and Tajana Rosing. Utilizing green energy prediction to schedule mixed batch and service jobs in data centers. In *Proceedings of the 4th Workshop on Power-Aware Computing and Systems*, HotPower '11, pages 5:1–5:5, New York, NY, USA, 2011. ACM.

[2] Michael Brown and Jose Renau. Rerack: Power simulation for data centers with renewable energy generation. In *In Workshop of GreenMetrics*, 2011.

[3] Process Design and Åbo Akademi University Systems Engineering Laboratory. The weather station at the process design and systems engineering laboratory. `http://at8 abo fi/cgi-bin/get_weather`.

[4] PV Education. Solar radiation on a tilted surface. `http://www pveducation org/pvcdrom/properties-of-sunlight/solar-radiation-on-tilted-surface`.

[5] Bob Erickson. Future directions in wind power conversion electronics. 2013.

[6] Martin Arlitt et al. Towards the design and operation of net-zero energy data centers. *ITherm*, 2012.

[7] M. Ghamkhari and H. Mohsenian-Rad. Optimal integration of renewable energy resources in data centers with behind-the-meter renewable generator. In *Communications (ICC), 2012 IEEE International Conference on*, pages 3340–3344, June 2012.

[8] I. Goiri, Kien Le, M.E. Haque, R. Beauchea, T.D. Nguyen, J. Guitart, J. Torres, and R. Bianchini. Greenslot: Scheduling energy consumption in green datacenters. In *High Performance Computing, Networking, Storage and Analysis (SC), 2011 International Conference for*, pages 1–11, Nov 2011.

[9] Íñigo Goiri, William Katsak, Kien Le, Thu D. Nguyen, and Ricardo Bianchini. Parasol and greenswitch: Managing datacenters powered by renewable energy. *SIGARCH Comput. Archit. News*, 41(1):51–64, March 2013.

[10] Íñigo Goiri, Kien Le, Thu D. Nguyen, Jordi Guitart, Jordi Torres, and Ricardo Bianchini. Greenhadoop: Leveraging green energy in data-processing frameworks. In *Proceedings of the 7th ACM European Conference on Computer Systems*, EuroSys '12, pages 57–70, New York, NY, USA, 2012. ACM.

[11] Sriram Govindan, Anand Sivasubramaniam, and Bhuvan Urgaonkar. Benefits and limitations of tapping into stored energy for datacenters. In *Proceedings of the 38th Annual International Symposium on Computer Architecture*, ISCA '11, pages 341–352, New York, NY, USA, 2011. ACM.

[12] Texas Weather Instruments Inc. Texas weather instruments inc.

[13] Chao Li, Amer Qouneh, and Tao Li. Characterizing and analyzing renewable energy driven data centers. In *Proceedings of the ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '11, pages 131–132, New York, NY, USA, 2011. ACM.

[14] Zhenhua Liu, Minghong Lin, Adam Wierman, Steven H. Low, and Lachlan L.H. Andrew. Geographical load balancing with renewables. *SIGMETRICS Perform. Eval. Rev.*, 39(3):62–66, December 2011.

[15] D. Lucanin. A geo-distributed cloud simulator. `https://github com/philharmonic/philharmonic`, 2014.

[16] D. Lucanin and I. Brandic. Pervasive cloud controller for geotemporal inputs. *Cloud Computing, IEEE Transactions on*, PP(99), 2015.

[17] D. Lucanin, I. Pietri, I. Brandic, and R. Sakellariou. A cloud controller for performance-based pricing. In *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*, pages 155–162, June 2015.

[18] Navin Sharma, Sean Barker, David Irwin, and Prashant Shenoy. Blink: Managing server clusters on intermittent power. *SIGARCH Comput. Archit. News*, 39(1):185–198, March 2011.

[19] First Solar. First solar achieves world record 18.percent thin film module conversion efficiency.

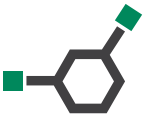
[20] ENERGY STAR. Market and industry scoping report: Solar pv inverters.

[21] Guangzhou HY Energy Technology. Hy small wind turbine special features.

[22] Rahul Urgaonkar, Bhuvan Urgaonkar, Michael J. Neely, and Anand Sivasubramaniam. Optimal power cost management using stored energy in data centers. In *Proceedings of the ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS '11, pages 221–232, New York, NY, USA, 2011. ACM.

[23] Di Wang, Chuangang Ren, Anand Sivasubramaniam, Bhuvan Urgaonkar, and Hosam Fathy. Energy storage in datacenters: What, where, and how much? *SIGMETRICS Perform. Eval. Rev.*, 40(1):187–198, June 2012.

[24] Yanwei Zhang, Yefu Wang, and Xiaorui Wang. Greenware: Greening cloud-scale data centers to maximize the use of renewable energy. In *Middleware 2011*, pages 143–164. Springer, 2011.



cost

Resource Management Optimization in Multi-Processor Platforms

Atanas Hristov

University of Information Science and Technology, Ohrid, Macedonia
atanas.hristov@uist.edu.mk

Iva Nikolova, Georgi Zapryanov

Technical University of Sofia, Bulgaria
inni@tu-sofia.bg, gszap@tu-sofia.bg

Dragi Kimovski

University of Innsbruck, Austria
dragi@dps.uibk.ac.at

Vesna Kumbaroska

University of Information Science and Technology, Ohrid, Macedonia
vesna.gega@uist.edu.mk

Abstract

The modern high-performance computing systems (HPCS) are composed of hundreds of thousand computational nodes. An effective resource allocation in HPCS is a subject for many scientific research investigations. Many programming models for effective resources allocation have been proposed. The main purpose of those models is to increase the parallel performance of the HPCS. This paper investigates the efficiency of parallel algorithm for resource management optimization based on Artificial Bee Colony (ABC) metaheuristic while solving a package of NP-complete problems on multi-processor platform.In order to achieve minimal parallelization overhead in each cluster node, a multi-level hybrid programming model is proposed that combines coarse-grain and fine-grain parallelism. Coarse-grain parallelism is achieved through domain decomposition by message passing among computational nodes using Message Passing Interface (MPI) and fine-grain parallelism is obtained by loop-level parallelism inside each computation node by compiler-based thread parallelization via Intel TBB. Parallel communications profiling is made and parallel performance parameters are evaluated on the basis of experimental results.

Keywords High-Performance Computing, Parallel Programming Model, Parallel Performance, Parallel Algorithm

I. Introduction

There are many open research problems in the field of high-performance computing systems (HPCS) studied extensively in many scientific research investigations. These systems are composed of hundreds or thousands of computational nodes and combine several technologies - hardware, software, networking and programming to solve advanced problems and performing experimental research work.

Most often the HPCS are used for high-throughput computing in time-sharing mode as well as for running complex parallel applications in space-sharing mode. One of the main challenges in HPC is to achieve highest possible system performance for a given application at optimal load balance and utilization of the available computational resources on the HPC platform. This causes the problem of effective resource management.

The resource management system is responsible for allocation of computing resources for extraordinary use and also to determine an optimal job or task scheduling for a

given system topology. An effective resource allocation and scheduling in HPCS is a subject of many scientific research investigations. The problem is well known as NP-complete [1], [2] and a number of approaches to different aspects of this problem can be found in the research literature. Also, many programming models have been proposed during the years. The main purpose of these models is to increase the parallel performance of HPCS. Currently, most of the HPC systems are based on conventional sequential programming languages as C, C++, FORTRAN. In order to achieve better parallel performance, the flat parallel programing model with message passing in distributed memory systems, supported by the MPI standard [3] and parallel programming model with multithreading in shared memory systems using the OpenMP programming interface [4] have been included as template libraries. The main disadvantages of the parallel programing based on conventional programming language are: process synchronization, deadlocks, workload balancing, and thread concurrency. In order to achieve better parallel performance, a parallel programming model must combine the distributed memory parallelization on the node interconnect with the shared memory parallelization inside of each node.

In order to improve this situation, Intel provides a range of tools specifically designed to help developers in parallelizing their applications. Three sets of complementary models for multithreading programming in shared memory systems are supported by Intel: Intel Cilk Plus, Intel Threading Building Blocks (Intel TBB) and Intel Array Building Blocks (Intel ArBB). The main purpose of those models is to increase the reliability, portability, scalability and the parallel performance of the application during the multithreading execution [5], [6].

The complexity class of decision problems NP-complete can be used as a pattern for benchmarking and parallel performance evaluation of multi-core and multi-machine architectures. Parallel versions of several NP-complete problems, such as N-Queens Problem, Travelling Salesman Problem, Sam-Loyd Puzzle etc., will be proposed in order to determinate the overall parallel performance of the system.

The paper investigates the efficiency of parallel algorithm for resource management optimization. It is proposed a metaheuristic approach based on swarm optimization with Artificial Bee Colony (ABC) [7] to solve the resource allocation problem for multi-core platform. The experimental work is based on the efficiency analysis of the proposed resource allocation scheme when solving of a package of three well known NP-complete problems - N-Queen, Travelling Salesman and Sam-Loyd Puzzle on homogeneous multi-processor platforms. Programmatically, the proposed scheme is implemented on the basis of multi-level hybrid parallel computational model using Intel TBB [8] and MPI [3] libraries. This model combines coarse-grain and fine-grain parallelism. Coarse-grain parallelism is achieved through domain decomposition by message passing among computational nodes using Message Passing Interface (MPI) and fine-grain parallelism is obtained by loop-level parallelism inside each computation node by compiler-based thread parallelization via Intel TBB.

The rest of this paper is organized as follows. An overview of the resource scheduling problem is presented in Section II with discussing some related works. In Section III, the proposed resource allocation scheme, based on ABC metaheuristics and its parallel implementation is presented. An experimental results and summary are offered in Section IV.

II. Related work

The resource management optimization problem has been studied extensively in the parallel and distributed computing literature for more than two decades. Many studies have been done in the field in order to effectively utilize the costly high performance computing platforms. Most of the advanced resource management systems are vendor-specific, but often they do not comply with specific features of a particular computing platform. Thus, they are not well optimized to provide efficient management to reach the required for a given parallel application performance of the implementation.

A variety of policies, strategies, schemes and algorithms have been proposed, developed, analyzed and implemented in a number of studies. These works investigate the problem in terms of diverse target HPC platforms. The most common researches are done in the field of high performance distributed computing with cluster, grid and cloud computing systems. Regardless of the conceptual closeness of these systems, the strategies for an optimal reserving of computing resources, effective load balance and resource utilization are different. Also, the resources that each parallel application for distributed processing requires can be very different from one to other and this raised the problem of finding an optimal job and tasks schedule for a given set of parallel resources. Taking into account the specific architectural, system and communication characteristics of a given parallel computer platform, finding an optimal solution of resource management task is further complicated.

The parallel resource scheduling problem is known to be NP-hard [1], [2]. It is usually solved by various heuristic and meta-heuristics algorithmic schemes [9], [10] depending on the homogeneity of the parallel system and the scheduling

method applied - static (off-line) or dynamic (on-line) one. Also, there are several exact algorithms with the goal to solve small to medium size problems to optimality [11], [12].

In the schemes with static scheduling it assumes that the total number of parallel executing tasks, as well the duration of each task is known in advance. The decision concerning computational resource allocation and task assignment is made at the start of the job execution. Because the execution time for a task is dependent on the input data, static scheduling carries some degree of uncertainty. This leads to unbalanced load, which leads to longer parallel execution time and low system resource utilization. With dynamic scheduling, the number of computational resources allocated to a job may vary during the execution. Also, the task assignment to the allocated resources takes place during the execution of a job. As pointed out in [13], dynamic scheduling policies are complementary to static policies in both their advantages and drawbacks. Because they are implemented at the execution time, dynamic policies usually incur in high run-time overhead, which may lead to a degradation of performance. Since decisions are made during job execution, scheduling should be based on simple and constant time heuristics. On the other hand, dynamic scheduling mechanisms exhibit an adaptive behavior, which leads to a high degree of load balancing.

In [14] the resource scheduling problem is explored in terms of real-time jobs executing on heterogeneous clusters. Heterogeneity in the parallel systems introduces an additional degree of complexity because, in addition to the problem of deciding when and how many computing resources to allocate, scheduling policies have also to deal with the choice among processor nodes of different speeds and also with reliability issues and tasks independency in parallel jobs. The proposed heuristic dynamic scheduling scheme (reliability-driven algorithm (DRCD)) for a various cluster sizes (between 4 and 18 machines) has been experimentally tested on a real world application DSP [15] as well as synthetic workloads, based on binary trees [16], lattices [17] and random graphs [18].

The problem of resource management in large many-core systems is addressed in [19], where a novel resource-management scheme that supports so-called malleable applications is proposed. These applications can adopt their level of parallelism to the assigned resources. [19] design a scalable decentralized scheme that copes with the computational complexity by focusing on local decision-making. The proposed algorithm is tested via simulation experiments on different system sizes ranging from 5x5 to 32x32 cores and synthetically generated workload consisting of 16, 32 and 64 parallel applications that is generated using the widely used Downey model [20].

Many papers have been published to address the problem of resource allocation in Grid computing environments. Some of the proposed algorithms are modifications or extensions to the traditional distributed systems resource allocation algorithms. A survey of job scheduling and resource management algorithms in Grid computing can be found in [21] where various algorithms are compared on various parameters like distributed, hierarchical, centralized, response time, load balancing, and resource utilization. The experiments were conducted via simulations with help of GridSim for number of jobs varied from 50 to 300.

In [22] resource-aware hybrid scheduling algorithm for different type of application: batch jobs and workflows are proposed. The performance tests are conducted in a realistic setting of CloudSim tool [23] with respect to load-balancing, cost savings, dependency assurance for workflows and computational efficiency. Multimedia applications that consist in both independent tasks and tasks with dependencies (workflows), and are both CPU intensive (they process a large amount of data) and I/O intensive (they access remote data) are used as test case scenarios. The experiments are performed with a 1000 tasks, 1000 Processing Elements and 10 Virtual Machines.

III. Parallel Implementation of Resource Management Optimization Algorithm

An effective resource utilization of the modern high performance computing (HPC) platforms is a subject for many scientific research investigations. The resource management optimization for those platforms is an essential part for optimal resource allocation while solving NP hard problems. An effective resource management algorithm strongly determines the overall parallel performance of the high-performance computing system. The proposed algorithm for resource management optimization in multi-core and multi-machine platforms is based on Artificial Bee Colony (ABC) metaheuristic. The ABC simulates the collective behavior of the honeybees in nature. The basic approach during implementation process is building a computer model which will simulate the collective behavior of the bees while collecting nectar.

In the proposed algorithm, the bees are divided in two beehives, beehive of the scout bees (beehive 1) and beehive of the onlooker and worker bees (beehive 2). When the algorithm is started, beehive 1 generates N number of scout bees, where N represents the number of processors in the system. Each scout bee checks whether a processor is free or busy by execution of specific task on it. If a free resource

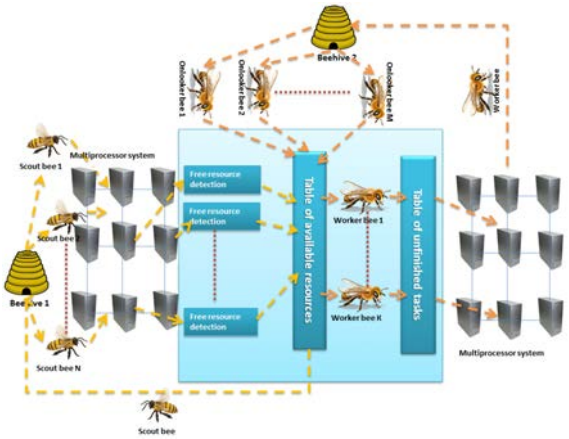


Figure 1: Parallel computing model for resource management optimization based on Artificial Bee Colony metaheuristic

is found the scout bee record the ID of the processor into the table of available resources and returns to the beehive 1, where it is terminates. The main purpose of the beehive 2 is to generate M number of onlooker bees, where M is the optimal number of parallel threads. After generation, the onlooker bees search in to table of available resources. If the onlooker bee finds a free resource, it takes the ID of the processor and removes it from the table. If the onlooker bee do not find a free resource in the table, the bee will return to the beehive 2 and will be terminate. Once the onlooker bee takes the available resource it starts to behave as a worker bee. Thus obtained K number of worker bees initially turned to the table of outstanding tasks where they taking certain sub-problem, remove it from the table and submit it for the performance by the processor which ID has been taken from the table of available resources. After the processor solves a sub-problem, it provides the solution to a worker bee. The worker bee with the current solution returns to beehive 2, where it is terminated.

In Figure 1, parallel computing model for resource management optimization based on artificial bee colony metaheuristic is presented. Parallel implementation of the algorithm was realized by using MPICH-2 message passing model and Intel TBB programming model built in Intel Parallel Studio 2010. For virtualization of resources a virtual machine of Intel ArBB, built-in Intel Parallel Studio 2010 was used.

IV. Experimental Evaluation

The experimental results were conducted by using multiprocessor platform. The platform is represented by a homogenous cluster composed of twelve Blade servers, HS21, Xeon Quad Core E405 80w 2.00GHz/1333MHz/12MB L2 and hard disk drive subsystems IBM 750GB Dual Port HS SATA HDD and Windows Server 2008 operating system.

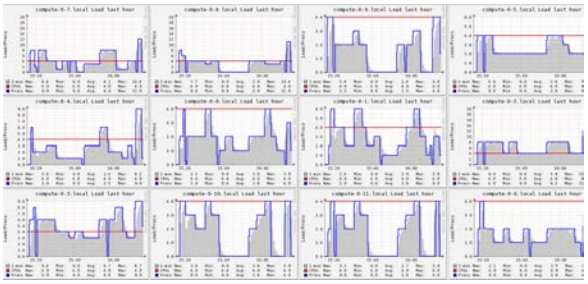


Figure 2: CPU load while solving package of three NP-Complete problems without the algorithm for resource management optimization

The load of the computational resources while solving a package of three NP-Complete problems - Traveling Salesman Problem, the N-queens problem, and the Sam-Loyd puzzle are presented in Figure 2. The package is started without algorithm for resource management optimization. From the charts shown on the figure, it is clear that the load of the processors is not well balanced, because only at a certain point of the time the processors have good load balance i.e. there are only few processors where the number of cores corresponds with the number of active processes. On the other hand, during the most of the time some of the processors have less number of active processes than cores, while some processors are overloaded i.e. the number of active processes exceeds number of cores in the processor.

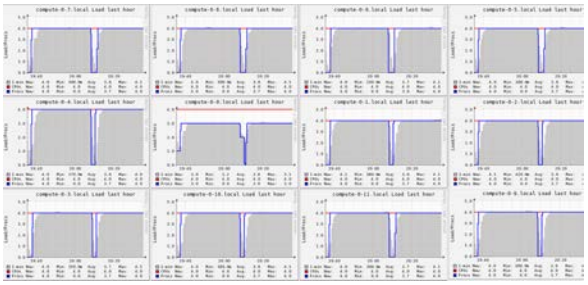


Figure 3: CPU load while solving package of three NP-Complete problems by using the algorithm for resource management optimization

In order to improve this situation, the proposed algorithm for resource management optimization was implemented on

the target platform. In Figure 3, the load of the processors while solving a package of three NP-Complete problems by using the algorithm for resource management optimization based on ABC metaheuristic is shown.

According to the figure above, it is clear that after the implementation of the optimization algorithm, the load of the processors is almost optimal as the overloading of the processors is avoided i.e. starting a bigger number of threads than cores on single processor, while dissatisfied load is shown only during the timeslots reserved of implementation of the algorithm for resource planning.

Figures 4 and 5 presents the load of the cluster during the execution of the tested package.

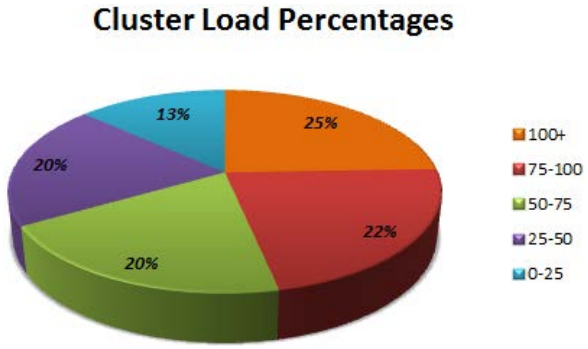


Figure 4: Cluster load during the execution of a package with three NP-Complete problems without the optimization algorithm

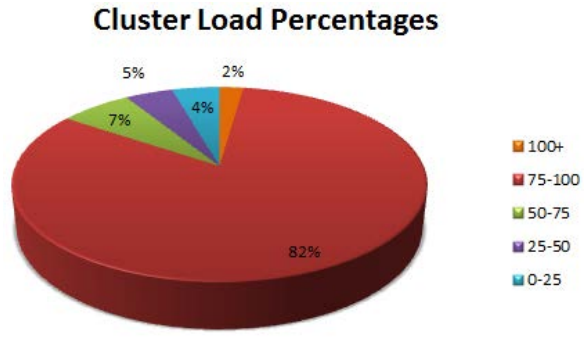


Figure 5: Cluster load during the execution of a package with three NP-Complete problems by using the proposed optimization algorithm

During the execution of package with three NP-Complete problems without the proposed algorithm for resource management optimization, only 22,22% of the resources of the cluster have optimal load balancing with 75-100%, while the remaining resources are overloaded with 100%+ or not good loaded i.e. below 75%. On the other hand, during the execution of the package by using the algorithm for resource management optimization, 82.22% of the resources of the cluster have optimal load balancing and only 17.77% of the resources have poor load balance. These 17% of the resources with poor load balance appears mainly due to the time required for implementation of the algorithm for resource planning as well as other system costs of the platform.

V. Conclusion and future work

An effective resource utilization of the modern high performance computing (HPC) systems is a subject for many scientific research investigations. The resource management for those platforms is an essential part for optimal resource allocation while solving NP complete problems. An effective resource management algorithm strongly determines the overall parallel performance of the high-performance computing system.

This paper suggests an innovative algorithm for effective resource management in multi-processor platforms based on parallel metaheuristic "Artificial Bee Colony" (ABC) optimization. The efficiency of the proposed algorithm for resource management in multi-processor platforms was evaluated on the basis of the software tools of Intel Array Building Blocks build-in Intel Parallel Studio.

Moreover, parallel programming implementations of three NP-Complete problems: the N-Queens problem, the Sam-Loyd puzzle, and the Traveling Salesman Problem (TSP) have been proposed in order to evaluate the overall parallel performance of the platform. The proposed parallel implementations were developed on the basis of Message Passing Interface (MPI) and Intel Threading Building Blocks (TBB) programming models.

Finally, we applied the proposed algorithm a homogenous cluster composed of twelve Blade servers HS21. This allows us to observe the behavior of the cluster while simultaneously is started a package of three NP-Complete problems. From the experimental results we conclude that in the cases when the proposed algorithm is run on the target platform, the cluster has very good load balance, which leads to increasing of the overall parallel performance of the system.

Future objectives of this research include implementation of our algorithm on very large-scale systems and on the new generation of ExaScale machines.

Acknowledgment

This work is partially supported by EU under the COST Program Action IC1305: Network for Sustainable Ultrascale Computing (NESUS).

References

[1] Ullman, J.D., 1975. NP-complete scheduling problems. Journal of Computer and System sciences, 10(3), pp.384-393.

[2] Hall, N.G. and Sriskandarajah, C., 1996. A survey of machine scheduling problems with blocking and no-wait in process. Operations research, 44(3), pp.510-525.

[3] Gropp, W., Lusk, E., Doss, N. and Skjellum, A., 1996. A high-performance, portable implementation of the MPI message passing interface standard. Parallel computing, 22(6), pp.789-828.

[4] Sato, M., 2002, October. OpenMP: parallel programming API for shared memory multiprocessors and on-chip multiprocessors. In Proceedings of the 15th international symposium on System Synthesis (pp. 109-111). ACM.

[5] Wooyoung Kim, Voss M., 2011. Multicore Desktop Programming with Intel Threading Building Blocks. IEEE Software journal, Page(s): 23 âĂŞ 31.

[6] Newburn C.J., Byoungro So, Zhenying Liu, McCool M., Ghuloum A., Toit S.D., 2011. Intel's Array Building Blocks: A retargetable, dynamic compiler and embedded language. In Proceedings of the 9th Annual IEEE/ACM International Symposium on Code Generation and Optimization, Page(s): 224 âĂŞ 235.

[7] Karaboga, D. and Basturk, B., 2007. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. Journal of global optimization, 39(3), pp.459-471.

[8] Reinders, J., 2007. Intel threading building blocks: outfitting C++ for multi-core processor parallelism. " O'Reilly Media, Inc."

[9] Haouari, M., Gharbi, A. and Jemmali, M., 2006. Tight bounds for the identical parallel machine scheduling problem. International Transactions in Operational Research, 13(6), pp.529-548.

[10] Talbi, E.G., 2009. Metaheuristics: from design to implementation (Vol. 74). John Wiley & Sons.

[11] Darbha, S. and Agrawal, D.P., 1998. Optimal scheduling algorithm for distributed-memory machines. IEEE transactions on parallel and distributed systems, 9(1), pp.87-95.

[12] Dell'Amico, M., Iori, M., Martello, S. and Monaci, M., 2008. Heuristic and exact algorithms for the identical parallel machine scheduling problem. INFORMS Journal on Computing, 20(3), pp.333-344.

[13] Saha, D., Menasce, D. and Porto, S., 1995. Static and dynamic processor scheduling disciplines in heterogeneous parallel architectures. Journal of Parallel and Distributed Computing, 28(1), pp.1-18.

[14] Qin, X. and Jiang, H., 2005. A dynamic and reliability-driven scheduling algorithm for parallel real-time jobs executing on heterogeneous clusters. Journal of Parallel and Distributed Computing, 65(8), pp.885-900.

[15] Woodside, C.M. and Monforton, G.G., 1993. Fast allocation of processes in distributed and parallel systems. IEEE Transactions on parallel and distributed systems, 4(2), pp.164-174.

[16] Srinivasan, S. and Jha, N.K., 1999. Safety and reliability driven task allocation in distributed systems. IEEE Transactions on Parallel and Distributed Systems, 10(3), pp.238-251.

[17] Qin, X. and Jiang, H., 2001, September. Dynamic, reliability-driven scheduling of parallel real-time jobs in heterogeneous systems. In Parallel Processing, 2001. International Conference on (pp. 113-122). IEEE.

[18] Ahmad, I. and Kwok, Y.K., 1999. On parallelizing the multiprocessor scheduling problem. IEEE Transactions on Parallel and Distributed systems, 10(4), pp.414-431.

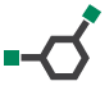
[19] Kobbe, S., Bauer, L., Lohmann, D., SchrÃűder-Preikschat, W. and Henkel, J., 2011, October. DistRM: distributed resource management for on-chip many-core systems. In Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis (pp. 119-128).

[20] Downey, A.B., 1997. A model for speedup of parallel programs. University of California, Berkeley, Computer Science Division.

[21] Buyya, R. and Murshed, M., 2002. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. Concurrency and computation: practice and experience, 14(13âĂŘ15), pp.1175-1220.

[22] Vasile, M.A., Pop, F., Tutueanu, R.I. and Cristea, V., 2013, December. HySARC2: hybrid scheduling algorithm based on resource clustering in cloud environments. In International Conference on Algorithms and Architectures for Parallel Processing (pp. 416-425). Springer International Publishing.

[23] Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A. and Buyya, R., 2011. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Software: Practice and Experience, 41(1), pp.23-50.



cost

Analysis of fiber-reinforced concrete: micromechanics, parameter identification, fast solvers

R. Blaheta[a], I. Georgiev[b], K. Georgiev[b], O. Jakl[a], R. Kohut[a], S. Margenov[b], J. Starý[a]

[a] Institute of Geonics, Czech Academy of Sciences, Ostrava, CR

[b] Institute of Information and Communication Technologies, Bulgarian Academy of Sciences, Sofia, BG

[blaheta, jakl, kohut, stary]@ugn.cas.cz, [ivan.georgiev, georgiev, margenov]@parallel.bas.bg

Abstract

Ultrascale computing is required for many important applications in chemistry, computational fluid dynamics etc., see an overview in the paper Applications for Ultrascale Computing by M. Mihajlovic et al. published in the International Journal Supercomputing Frontiers and Innovations, Vol 2 (2015). In this abstract we shortly describe an application that involves many aspects described in the above paper - the multiscale material design problem. The problem of interest is analysis of the fiber reinforced concrete and we focus on modelling of stiffness through numerical homogenization and computing local material properties by inverse analysis. Both problems require a repeated solution of large-scale finite element problems up to 200 million degrees of freedom and therefore the importance of HPC and ultrascale computing is evident.

Keywords Analysis of fiber-reinforced concrete, homogenization, identification of parameters, parallelizable solver, additive Schwarz method, two-level parallelization

I. Introduction

This paper is a continuation of paper [1] presented at the NESUS workshop in Cracow, Poland 2015. While [1] focused on linear micromechanics exploiting CT scans for determination of microstructure and numerical homogenization, this paper is driven by a specific application - analysis of fiber-reinforced concrete. This analysis includes an identification problem and stochastic uncertainty, which brings new dimension and enhances the need for fast solvers and ultrascale computations.

Fiber-reinforced concrete with steel fibers has a lot of applications in civil and geotechnical engineering. It is less expensive than hand-tied rebar, while still increasing the tensile strength many times. The shape, dimension, and length (standard 1 mm diameter, 45 mm length) of the fiber together with fiber volume amount and distribution are important parameters influencing the tensile strength of concrete.

The analysis includes assessment of tensile stiffness for several samples of fiber-reinforced concrete which differ in amount and distribution of fibers. These samples are scanned by CT and analysed with provided elastic parameters for steel fibers and concrete matrix. The detailed scan of a sample leads to solving of elastic problems with about 200 million degrees of freedom.

As the global response of the samples can be tested on a loading frame, then the output allows to solve an inverse identification procedure to determine the elastic properties of the concrete matrix. In this way we can both determine the properties of concrete matrix, which can also be variable to some extent, as well as assess whether some discrepancy can be explained by imperfect bonding of fibers.

It is also possible not only to investigate selected physical samples of the fiber-reinforced concrete but to do stochastic analysis with a repeated generation of stochastic microstructure, see e.g. [5, 6].

II. Homogenization and identification of parameters

The numerical homogenization starts with solving the elasticity problem on the domain Ω with given microstructure. The solution is possibly repeated for different loadings by imposed boundary conditions. In an abstract way, we denote the loading conditions by L or in the case of multiple loading by $L^{(k)}$. The stress and strain tensors $\sigma^{(k)}$ and $\varepsilon^{(k)}$ are averaged over Ω and the homogenized elasticity tensor $\bar{C} \in R^{6\times6\times6\times6}_{sym}$, $C = \left[c_{ijkl}\right]$, $c_{ijkl} = c_{jikl} = c_{klij}$ is determined

as a (generalized) solution of the system

$$C\bar{\varepsilon}^{(k)} = \bar{\sigma}^{(k)}, \quad \bar{\sigma}^{(k)} = |\Omega|^{-1} \int_\Omega \sigma^{(k)} d\Omega, \quad \bar{\varepsilon}^{(k)} = |\Omega|^{-1} \int_\Omega \varepsilon^{(k)} d\Omega.$$

Assuming isotropy of the homogenized elasticity tensor, one loading is sufficient for getting elasticity constants. If $\xi = \xi_{vol} + \xi_{dev}$, is the decomposition of $\xi \in R^{6\times 6}_{sym}$ into the volumetric and deviatoric parts and $\|\cdot\|$ is the Frobenius norm, then the bulk and shear moduli can be determined as

$$K = \frac{1}{3} \|\bar{\sigma}_{vol}\| \,/\, \|\bar{\varepsilon}_{vol}\|, \quad G = \frac{1}{2} \|\bar{\sigma}_{dev}\| \,/\, \|\bar{\varepsilon}_{dev}\|.$$

For parameter identification, we assume that some local material properties are unknown, e.g. that the concrete matrix is described by unknown parameters $p = (K_c, G_c)$, where K_c and G_c are unknown bulk and shear parameters of the concrete. More generally, Ω can be split into subdomains with different unknown elastic moduli of concrete. Then the parameters are found by minimization of a proper objective function J over a set of admissible parameters, see e.g. [7].

The construction of the objective function can be as follows

$$J(p) = \sum_k \left[w_{1k} \left\| \bar{\varepsilon}^{(k)}(p) - \bar{\varepsilon}^{(k)}_{test} \right\|^2 + w_{2k} \left\| \bar{\sigma}^{(k)}(p) - \bar{\sigma}^{(k)}_{test} \right\|^2 \right],$$

where $\bar{\sigma}^{(k)}(p)$ and $\bar{\varepsilon}^{(k)}(p)$ are averaged stresses and strains computed by solving the boundary value problem in Ω with given microstructure, local material properties involving the parameters from p and the loading $L^{(k)}$. This boundary value problem represents a physical test on the specimen Ω. The test configuration is such that in the case of homogeneity of Ω, the problem has a solution with unique and constant stress $\bar{\sigma}^{(k)}_{test}$ and strain $\bar{\varepsilon}^{(k)}_{test}$, which can be determined from measurements. The weights w_{ik} can be determined by numerical experiments or simply set to be equal $w_{ik} = 1$.

The optimization is performed by a suitable method, we already successfully tested the Nelder-Mead and Gauss-Newton methods.

More details on the exploited homogenization and identification methods can be found in [3, 4].

III. Additive Schwarz solver with two-level parallelization

A crucial component of the homogenization and identification procedures is the solver for boundary value problems of elasticity. We assume finite element discretization leading to algebraic systems of the type of $Au = b$ or $A(p)u(p) = b$, where later indicates dependence on some local material parameters. The system can be solved by the preconditioned conjugate gradient (PCG) method with one level additive Schwarz (AS) preconditioner B_{AS1} and mostly its extended two-level version B_{AS2},

$$B_{AS1} = \sum_{k=1}^{N} R_k^T \tilde{A}_k^{-1} R_k, \quad B_{AS2} = B_{AS1} + R_0^T \tilde{A}_0^{-1} R_0.$$

Here R_k is a restriction defined by subdomain Ω_k or algebraically by overlapping decomposition of the solution vector $u \in R^n$, $\tilde{A}_k$ is an approximation to $A_k = R_k A R_k^T$. In our case $\tilde{A}_k$ is a displacement decomposition - incomplete factorization of A_k. The one level AS preconditioner is not scalable, the number of iterations increases with N, although this grow is a bit compensated by the fact that $\tilde{A}_k$ becomes a better approximation to A_k. It fits the algebraic form of the Schwarz methods if $R_0 \in R^{n_0 \times n}$ is a Boolean matrix, which defines aggregation of degrees of freedom, i.e. each row of R_0 defines one aggregate by unities in this row. On the other hand, each degree of freedom corresponds to just one aggregate, i.e. there is precisely one unity in each column of R_0. More details about this setting can be found e.g. in [2].

In the case of computing at a massively parallel computer like Salomon [8], it is possible to exploit hundreds of processors, which makes the local problems A_k small even for large scale matrices A. It makes difficult to keep balance of times for solving the local problems A_k and the coarse global one A_0. For this reason, parallel inner CG iterations for the solution of problem A_0 were suggested and the algorithm become with two levels of parallelization.

IV. Numerical experiments

Our numerical experiments present five real samples of fiber-reinforced concrete, each of cubic shape and size 35 mm.

Variant	Steel fibers [kg/m^3]	Volume Steel [%]	Volume Voids [%]
0	0	0.00	1.55
2	50	0.92	1.22
3	100	1.82	0.75
4	150	2.57	0.71
5	200	2.11	1.83

Table 1: Characteristics of REV for each sample of reinforced concrete. Variants differ in the volumes of steel fibers as well as voids. The size of fibers: length 6 mm, diameter 0.12 mm.

Their microstructure is taken from industrial CT scanning performed at the CT lab of the Institute of Geonics. Digital

models arose from meshes of approx. 1400×1400×1400 voxels, which were further trimmed to 1000×1000×1000 voxels due to surface damage or irregular sides of the samples.

Consequent computational models use smaller representative volumes (REV) and standard linear tetrahedral finite elements. The size of each REV is 400×400×400 for homogenization experiments or 100×100×100 voxels for tests related to material identification, respectively. Accordingly the model leads to a (repeated) solution of the resulting linear system in size of about 193 millions or 3 millions degrees of freedom. Main characteristics of each REV are summarized in Tab. 1.

Material	E [GPa]	ν
concrete	19	0.2
steel	200	0.3
voids	0.01	0.1

Table 2: List of involved materials and their properties (Young's modulus E and Poisson's ratio ν).

The properties of the materials involve in mathematical modelling are listed in Tab. 2. Voids (air bubbles in the microstructure) bring a kind of singularity caused by the finite elements weekly hanged in the void space. They are replaced with a very week elastic material. The convergence of the applied PCG method is then smoother and faster.

The arising large-scale systems of linear equations are processed by parallel solvers based on the PCG method, with stabilization in the singular case [10]. The computations are performed on SGI cluster Salomon [8] run by the IT4Innovations National Supercomputing Center in Ostrava. The cluster, currently on 55. place in Top500, consists of 24192 cores and 129 TB of memory in total and with the theoretical peak performance over 2 Pflop/s. The most of its compute nodes is equipped by two 12-core processors Intel Xeon E5-2680 v3 and 128 GB of memory.

Tab. 3 gives the results of numerical homogenization applying pure Dirichlet and pure Neumann boundary conditions (BC). The choice of BC sets a configuration of homogenization procedure, which simulates an appropriate laboratory test under uniaxial loading. Dirichlet BC prescribe some non-zero displacement on the top side in the direction of uniaxial loading, the other sides have zero normal displacements. Neumann BC enter opposite non-zero forces on the top and bottom sides in the direction of uniaxial loading, the other sides have zero normal forces. The use of pure Dirichlet and pure Neumann BC allows us to get upper and lower bounds for the upscaled elasticity tensor, see e.g. [3].

Due to irregular placement of steel fibers as well as voids

Dirichlet BC						
Variant	E [GPa]			ν		
	18.365	18.370	18.407	0.199	0.199	0.199
0		18.381			0.199	
	19.050	18.960	19.063	0.200	0.201	0.200
2		19.024			0.200	
	20.015	19.621	19.768	0.200	0.202	0.201
3		19.801			0.201	
	20.865	19.977	19.960	0.198	0.203	0.203
4		20.267			0.201	
	19.345	19.508	19.715	0.202	0.202	0.201
5		19.523			0.202	

Neumann BC						
Variant	E [GPa]			ν		
	18.307	18.305	18.216	0.199	0.199	0.197
0		18.276			0.198	
	18.692	18.822	18.798	0.197	0.199	0.199
2		18.771			0.198	
	19.912	19.599	19.716	0.203	0.199	0.201
3		19.742			0.201	
	20.613	19.948	19.435	0.204	0.199	0.195
4		19.999			0.199	
	18.297	17.193	19.213	0.190	0.178	0.199
5		18.234			0.189	

Table 3: Results obtained by numerical homogenization applying Dirichlet and Neumann BC. Values of material parameters for different directions (X Y Z) of uniaxial loading and averaged (below).

in the microstructure, the results documents the anisotrophy of tested material, when the values of material properties strongly vary for different directions of loading, e.g. the Young's modulus E (the sample 4, Neumann BC) in Tab. 3 varies about more than 1 GPa. However as expected and consistent with theory, their averaged values follow the increase of volume of steel fibers in concrete.

The corresponding values for pure Dirichlet and pure Neumann BC give quite close bounds for real material properties. However we observe that these bounds grow away with the increasing volume of voids in the microstructure, moreover when the voids are closer to the border of the studied domain and pure Neumann BC are applied, see the values for the sample 5. Comparing with the others, the sample 5 contains also another abnormality. Although this sample should contain the most of steel fibers according to Tab. 1, the real volume of steels in REV is not the biggest. Moreover,

REV of this sample overcomes the others in the volume of the void space in its microstructure.

The previous tests were related to the direct problem denoting a computation of stiffness of the fiber reinforced concrete based on known material distribution and local material properties. The next numerical experiments describe one of the possible inverse problems, an identification of the material properties (Young's modulus E and Poisson ratio ν) of the concrete matrix from known material distribution, elastic properties of fibers and response of the sample (REV) to uniaxial or triaxial loading tests. This inverse problem exploits the objective function (the cost functional) $J(p)$, $p = (E, \nu)$, $w_{1k} = w_{2k} = 1$, introduced in the section II. For more details see [3].

Dirichlet BC			
Variant	Steps	E [GPa]	ν
0	135	19.020	0.199
2	141	19.000	0.200
3	141	19.005	0.200
4	141	19.029	0.200
5	141	19.007	0.200

Neumann–Dirichlet BC			
Variant	Steps	E [GPa]	ν
0	138	18.996	0.200
2	135	19.004	0.200
3	135	19.006	0.200
4	162	19.034	0.200
5	129	19.007	0.200

Table 4: Results of material identification applying Dirichlet and Neumann-Dirichlet BC. The number of transformation steps of the applied Nelder-Mead method and the identified averaged material properties of the concrete matrix for each REV.

The optimization is performed by the non-gradient Nelder-Mead (NM) method with starting values (E, ν) provided by three pairs (17.000, 0.26), (21.000, 0.17), (18.000, 0.23). In each step of the NM method, three direct problems (three computation of local stresses and strains), corresponding to simulation of uniaxial loading tests for each direction X, Y and Z, are solved. Dirichlet BC describe the same loading as in case of homogenization tests. Neumann-Dirichlet BC enter a combination of pure Dirichlet and pure Neumann BC introduced earlier. It means the prescribed non-zero displacement on the top side in the direction of loading, zero displacement on the bottom side in the direction of loading and zero normal forces on the other sides. The NM iterations are stopped if the decrease of the cost functional and differences in the identified parameters are sufficiently small.

The numbers of transformation steps performed by the NM optimizaton procedure and the averaged values of the identified material properties are summarized in Tab. 4. Dirichlet BC on the whole sample boundary are used for comparison purposes. They are applicable if the loading response is computed artificially. The obtained results show a good accordance with the values for the concrete matrix presented in Tab. 2. Considering the number of NM steps and a need to repeat the FEM calculation several times in each step, the results document also a substantially increased requirements on the computational power of the used computer.

V. Tuning of parallel solvers

Nowadays powerful parallel computers for HPC have hundreads or thousands of cores. Therefore we decided to reimplement our original parallel solver for large-scale systems of linear equations arising from 3D boundary problems of elasticity. The solver dates back to the times of Beowulf type clusters and small multiprocessors with up to 20 processors.

The original solver is based on the PCG method, uses the one-directional domain decomposition for parallelization of iterative process as well as the construction of efficient one-level and two-level AS preconditioners (AS1, AS2), see their definition through B_{AS1} and B_{AS2} in III. Parallel processes communicate through message passing (MPI standard).

Figure 1: Traces of one PCG iteration processing 4 subdomains. From above, records for the original solver with AS1 and AS2, and new solver with AS1 only. States of parallel processes: work (blue), wait or idle (red).

Fig. 1 shows traces of the runs of parallel solvers produced by the Intel Trace Analyzer. The implementation of the

original solver follows the master-slave design, when the first process (from above) is the master, almost idle, just controlling the iterative process and computing two global scalar products. Each of the four slave processes (below the master) works on its portion of data, especially during the dominating operations matrix by vector multiplication (MXV) and preconditioning (PREC).

The second trace adds a coarse grid computation to AS2. This computation is performed by a separate process, idle for more than a half of the iteration execution time. Nevertheless this process is very important because a coarse grid computation strongly improves the efficiency of the preconditioner and speeds up the convergence of the PCG iterations.

The third trace documents a run of the new version of the parallel solver, surpassing the original one in the execution time and a better utilization of processes. New solver works internally with data in double precision and dynamic allocation of memory, uses a modified domain decomposition (with an overlapping of subdomains) leading to a better load balancing of processes, has optimized (mainly global) communication of processes and also calculations in loops (during MXV and PREC operations). The new solver abandons master-slave design, the negligible amount of work performed by the master process was taken over by the other processes.

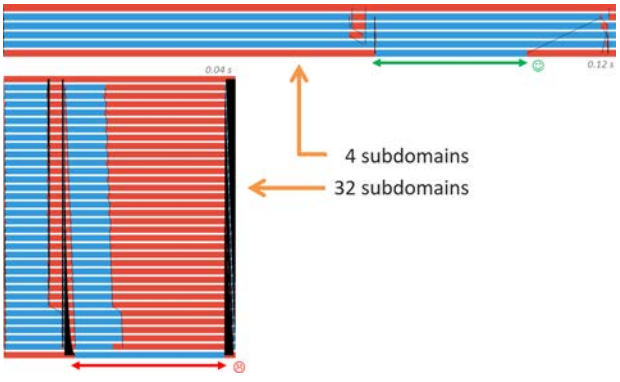


Figure 2: A coarse grid computation bottleneck in the original solver. Traces of one PCG iteration processing 4 and 32 subdomains.

The next step in the parallel solver optimization is indicated by Fig. 2. With the increase of processes, the execution time of the most demanding MXV and PREC operations performed by worker processes scales down correspondingly, whilst the execution time of a coarse grid computation stays constant. In the example shown in Fig. 2, the described effect limits the possible speed-up of the solver only to 3, instead of expected 8, which corresponds to the increase of the number of processes.

Such a negative effect can be eliminated by a coarse grid parallelization in a hybrid way, when all processes do not perform the same calculations. On hundreads of computing elements (processors or cores), such hybrid parallelization includes the most of processes solving the subproblems corresponding to subdomains and only a few (units or tens) of processes performing coarse grid computations in parallel. It should not substantially decrease convergence properties of the applied AS2 preconditioner, but dramatically increase the efficiency of the resulting PCG iterations. However, the described hybrid paralellization can bring difficulties how to treat optimal load balancing of processes.

VI. Conclusions

The paper demonstrates the need for high performance computing by focusing on one engineering application - investigation of the fiber reinforced concrete. The primary analysis solves a microscale problem for homogenization within the range of linear material behaviour. This basic problem can be modified (extended) in several directions and any of them substantially increases the computational demands. One extension, roughly described in this abstract, is the solution of the inverse problem of identification of the local material parameters or some level of debonding of the matrix and fibers. This problem is solved by the optimization methods which require repeated solution of the basic problem. The increase in computational demands can be about hundred times. Another extension is based not only on the solution of selected and scanned samples of the concrete, but also on the stochastic generation of a set of such samples and evaluation of the mean properties by Monte Carlo or multi-level Monte Carlo methods, see e.g. [9]. The last extension is to consider the strengths and non-linear post peak behaviour, which involves the usage of damage mechanics techniques, see e.g. [5] and the references therein.

Acknowledgement: The work is supported by COST Action IC1305 project Network for Sustainable Ultrascale Computing and a bilateral project of collaboration between the Institute of Geonics CAS and IICT BAS. Further support is through the projects LD15105 Ultrascale computing in geosciences and LQ1602 IT4Innovations excellence in science supported by the Ministry of Education, Youth and Sports of the Czech Republic.

References

[1] R. Blaheta, A. Kolcun, O. Jakl, K. Souček, J. Starý and I. Georgiev: *HPC in Computational Micromechanics of Composite Materials.* NESUS workshop, Cracow, Poland, 2015.

[2] R. Blaheta, O. Jakl, R. Kohut and J. Starý: *GEM - A Platform for Advanced Mathematical Geosimulations.* In: R. Wyrzykowski et al. (eds.): PPAM 2009, Part I, LNCS 6067, 2010, pp. 266–275.

[3] R. Blaheta, R. Kohut, A. Kolcun, K. Souček, L. Staš and L. Vavro: *Digital image based numerical micromechanics of geocomposites with application to chemical grouting.* International Journal of Rock Mechanics and Mining Sciences. Vol. 77 (2015), pp. 77-88.

[4] R. Blaheta, R. Kohut and J. Starý: *Computational and reliability aspects of micro-geomechanics.* In: Oka, Murakami, Uzuoka and Kimoto (eds.): Computer Methods and Recent Advances in Geomechanics, Taylor & Francis Group, London, 2015.

[5] M.A. Hickman and P.K. Basu: *Stochastic Multiscale Characterization of Short-Fiber Reinforced Composites.* Technische Mechanik, 36, Vol. 1-2 (2016), pp. 13–31.

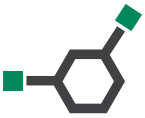
[6] X. Guan, X. Liu, X. Jia, Y. Yuan, J. Cui and H.A. Mang: *A stochastic multiscale model for predicting mechanical properties of fiber reinforced concrete.* International Journal of Solids and Structures, Vol. 56–57 (2015), pp. 280–289.

[7] J. Haslinger, R. Blaheta and R. Hrtus: *Identification problems with given material interfaces.* Journal of Computational and Applied Mathematics, early view June 2016.

[8] *Salomon Cluster Documentation*, https://docs.it4i.cz/salomon

[9] R. Blaheta, S. Domesová and M. Béreš: *A study of stochastic FEM method for porous media flow problem.* Proceedings of PhD workshop, Institute of Geonics CAS, December 2015.

[10] R. Blaheta, O. Jakl, J. Starý and E. Turan: *Parallel solvers for numerical upscaling.* In: PARA 2012, LNCS 7782, Springer-Verlag, 2013, pp. 375–386.



cost

A Data-Aware Scheduling Strategy for DMCF workflows over Hercules

Fabrizio Marozzo∗, Francisco Rodrigo Duro†, Javier Garcia Blas†

fmarozzo@dimes.unical.it, frodrigo@arcos.inf.uc3m.es, fjblas@arcos.inf.uc3m.es

Jesus Carretero†, Domenico Talia∗, Paolo Trunfio∗

jesus.carretero@uc3m.es, talia@dimes.unical.it, trunfio@dimes.unical.it

∗ DIMES, University of Calabria, Italy
† ARCOS, University Carlos III, Spain

Abstract

As data-intensive scientific prevalence arises, there is a necessity of simplifying the development, deployment, and execution of complex data analysis applications. The Data Mining Cloud Framework is a service-oriented system for allowing users to design and execute data analysis applications, defined as workflows, on cloud platforms, relying on cloud-provided storage services for I/O operations. Hercules is an in-memory I/O solution that can be deployed as an alternative to cloud storage services, providing additional performance and flexibility features. This work extends the DMCF-Hercules cooperation by applying novel data placement and task scheduling techniques for exposing and exploiting data locality in data-intensive workflows.

Keywords DMCF, Hercules, workflows, in-memory storage, data cache, Microsoft Azure, data locality

I. Introduction

Scientific computing applications and platforms are evolving from CPU-intensive tasks executed over strongly coupled infrastructures, i.e. complex simulations running on supercomputers, to data-intensive problems requiring flexible computing resources depending on the requirements and budget of the user. This evolution paves the future of Ultrascale systems, which will blur the differences of existing scientific computing infrastructures, such as HPC systems and cloud computing platforms. In current approaches, the interfaces and management of the different infrastructures are too different, requiring different programming models, even for the same application. In contrast, the future Ultrascale systems should take advantage of every possible resource available, in a transparent way for the user.

Workflow engines are the leading approach for executing data-intensive applications in different computing infrastructures. Scientific workflows consist of interdependent tasks, connected in a DAG style, which communicate through intermediate storage abstractions, typically files. There is a main tradeoff that should be taken into account when the user relies on workflow engines for data-intensive applications. While portability and flexibility offers a broader support of the existing computing resources, the achieved performance is usually limited in contrast with native applications (classical HPC applications running on HPC clusters or supercomputers).

The increasing availability of data generated by high-fidelity simulations and high-resolution scientific instruments in domains as diverse as climate, experimental physics, bioinformatics, and astronomy, has shown the underlying I/O subsystem to be a substantial performance bottleneck. While typical high-performance computing (HPC) systems rely on monolithic parallel file systems, data-intensive workflow implementations must borrow techniques from the Big Data computing (BDC) space, such as exposing data storage locations and scheduling work to reduce data movement. This lack of performance is the result of a sub-optimal exploitation of the available resources, based on two main reasons: task schedulers unable to select the best nodes depending on the characteristics of the task and under-performing I/O solutions.

Our previous works have targeted these disadvantages in a real-world scenario by combining two existing solutions: the Data Mining Cloud Framework (DMCF) and the in-memory

I/O accelerator known as Hercules. The present work deepens in this combination providing locality-aware features, both in the DMCF task scheduler and in the Hercules data placement algorithms. By running the workflow workers in the same VM instances as Hercules I/O nodes, data locality can be exposed and exploited, executing the task in the node where the data are stored in-memory.

This paper proposes the application of locality-aware data placement and data discovery techniques into the DMCF-Hercules integration. Additionally, this work proposes a novel task scheduler integrated in DMCF for the co-location of tasks and data, relying on the locality-aware functionality offered by Hercules. The evaluation carried on shows how data-locality exploitation is especially critical in cloud platforms, where virtualized network interfaces provide limited bandwidth in contrast with the state-of-the-art high-performance network infrastructures present in HPC systems.

The remainder of the paper is structured as follows. Section II describes the main features of DMCF. Section III introduces Hercules architecture and capabilities. Section IV emphasizes the advantages of integrating DMCF and Hercules and outlines how this integration will work. Section IV.3 details the novel locality-aware techniques proposed in this work. Section V presents preliminary results of the performance improvements achieved by the application of the locality-aware techniques in a Microsoft Azure cloud infrastructure. Finally, section VI concludes the work and give some future research related to the presented work.

II. Data Mining Cloud Framework overview

The Data Mining Cloud Framework (DMCF) [1] is a software system designed for designing and executing data analysis workflows on Clouds. A Web-based user interface allows users to compose their applications and to submit them for execution to the Cloud platform, following a Software-as-a-Service (SaaS) approach.

The architecture of DMCF includes different components that can be grouped into storage and compute components (see Figure 2).

The DMCF architecture has been designed to be implemented on top of different Cloud systems. The implementation used in this work is based on Microsoft Azure[1].

DMCF allows to program data analysis workflows using two languages: VL4Cloud (Visual Language for Cloud) and JS4Cloud (JavaScript for Cloud).

Both languages use two key programming abstractions:

- Data elements, denoting input files or storage elements (e.g., a dataset to be analyzed) or output files or stored elements (e.g., a data mining model).
- Tool elements, denoting algorithms, software tools or complex applications performing any kind of operation that can be applied to a data element (data mining, filtering, partitioning, etc.).

Another common element is the Task concept, which represents the unit of parallelism in our model. A task is a Tool invoked in the workflow, which is intended to run in parallel with other tasks on a set of Cloud resources. According to this approach, VL4Cloud and JS4Cloud implement a data-driven task parallelism.

III. Hercules overview

Hercules [2] is a distributed in-memory storage system based on the key/value Memcached database [3]. The distributed memory space can be used by the applications as a virtual storage device for I/O operations and has been especially adapted in this work for being used as an in-memory shared storage for cloud infrastructures. Our solution relies on an improved version of Memcached servers, for offering an alternative storage solution to the default cloud storage service provided by Azure.

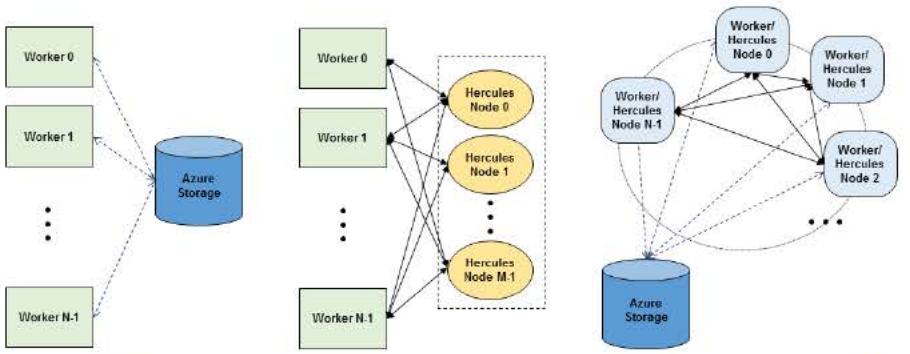
Figure 3 shows how Hercules architecture has two main layers: front-end (Hercules client library) and back-end (server layer). The worker user-level library is based on a layered design, while back-end components are based on the Memcached server, extending its functionality with persistence and tweaks. Main advantages offered by Hercules are: scalability, easy deployment, flexibility, and performance.

Scalability is achieved by fully distributing data and metadata information among all the nodes, avoiding the bottlenecks produced by centralized metadata servers. Data and metadata placement is completely calculated in the worker-side by a hash algorithm. The servers, on the other hand, are completely stateless.

Easy deployment and flexibility at worker-side are tackled using a POSIX-like user-level interface (open, read, write, close, etc.) in addition to classic put/get approach existing in current NoSQL databases. Existing software requires minimum changes to run using Hercules. Servers can be deployed without requiring any special privileges

Finally, performance and flexibility at server-side are targeted by exploiting the parallel I/O capabilities of Memcached servers. Flexibility is achieved by Hercules due to its easiness to be deployed dynamically on as many nodes as

[1] http://azure.microsoft.com



(a) Scenario 1: Azure storage. (b) Scenario 2: Hercules. (c) Scenario 3: Azure storage and Hercules .

Figure 1: Integration scenarios between DMCF and Hercules.

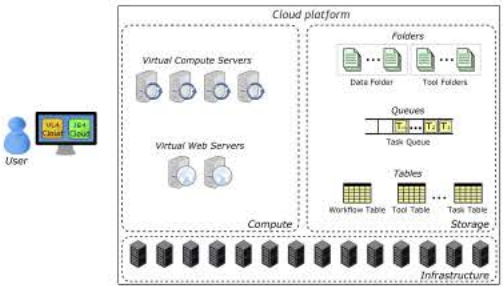


Figure 2: DMCF architecture.

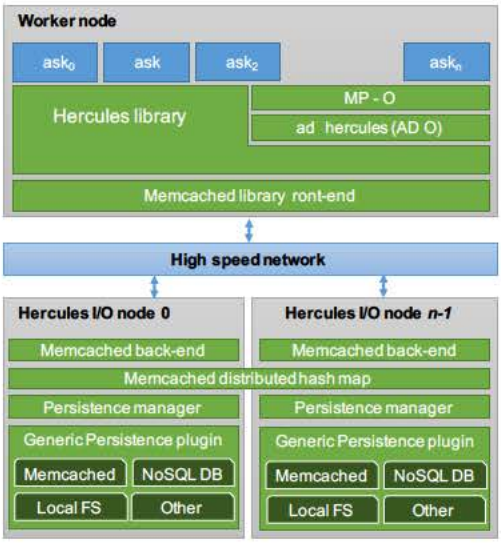


Figure 3: Hercules architecture.

necessary. Each node can be accessed independently, multiplying the total throughput peak performance.

IV. Integration between DMCF and Hercules

The integration between DMCF and Hercules is an ongoing work where different scenarios have been studied in previous works. As can be seen in Figure 1, Hercules and DMCF can be configured according with different deployment scenarios to achieve different levels of integration.

Figure 1(a) shows the original approach of DMCF, where every I/O operation is performed against the cloud storage service offered by the cloud provider (Azure Storage). There are, at least, four disadvantages about this approach: proprietary interfaces, I/O contention in the service, lack of configuration options, and persistence-related costs unnecessary for temporary data.

Figure 1(b) shows a second scenario with the use of Hercules as the default storage for temporary generated data [4]. Hercules I/O nodes can be deployed on as many VM instances as needed by the user depending on the required performance and the characteristics of data.

Figure 1(c) shows a third scenario with a tighter integration of DMCF and Hercules infrastructures. In this scenario, initial input and final output are stored on persistent Azure storage, while intermediate data are stored on Hercules in-memory nodes. Hercules I/O nodes share virtual instances with the DMCF workers.

Previous work [5] explored the third scenario outlined above. However, in order to simplify the implementation of the solution, some workarounds were explored: each time

that one worker needed to access data (read/write operations over a file), it copied the whole file from Hercules servers to the worker local storage. This approach may greatly penalize the potential performance gain in I/O operations for two main reasons:

- *Data placement strategy*. The original Hercules data placement policy distributes every partition of a specific file among all the available servers. This strategy has two main benefits: avoids hot spots and improve parallel accesses. In an improved DMCF-Hercules integration, whole files can be stored on the same Hercules server.
- *Data locality agnosticism*. Data-locality will not be fully exploited until the DMCF scheduler is tweaked for running tasks on the node that contains the necessary data and/or the data is placed where the computation will be realized.

IV.1 Improved integration strategy

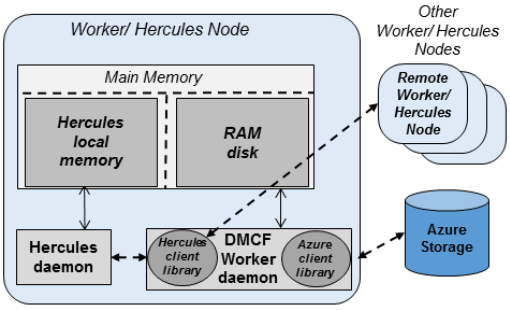


Figure 4: DMCF and Hercules daemons.

Figure 4 describes the proposed improvement to the third scenario of integration between DMCF and Hercules. Four main components are present: DMCF Worker daemon, Hercules daemon, Hercules client library, and Azure client library. The DMCF workers are in charge of executing the tasks of the workflow (data analysis tools/applications), Hercules daemons act as I/O nodes (storing data in-memory and managing data accesses), the Hercules client library is intended to be used by the applications to access to the data stored in Hercules (query Hercules daemons), and the Azure client library is used to read/write data from/to the Azure storage.

To exploit the potential of the data-aware DMCF-Hercules integration, we propose the use of a RAM disk as generic storage buffer for I/O operations performed by workflow tasks. The objective of this approach is the support of DMCF to any existing tool, supporting even binaries independently of the language used for their implementation, while offering in-memory performance for local accesses.

The logic used for managing this RAM disk buffer is based on the full information about the workflow possessed by the DMCF workers. When every dependency of an specific task is fulfilled (every input file is ready to be accessed) the DMCF worker brings the necessary data to the node from the storage (Azure Storage in the first scenario or Hercules in the second scenario). Instead of storing the data in the default file system, as in previous works, we propose storing the data in a RAM disk.

IV.2 Resource optimization challenge and possible solutions

If this solution shows potential performance gains compared with the existing solution, the need of duplicated memory regions (RAM disk and Hercules local memory) can be avoided. We propose three different approaches for solving this challenge:

- Modify Hercules daemons to use the RAM disk memory region as default storage, avoiding the necessity of the *Hercules local memory* region. Hercules daemon can store data in the RAM disk instead of using the Hercules local memory. Any data stored in the RAM disk can be transparently accessed by workflow tasks.
- Modify the code of the workflow tasks (tools/applications) to use the Hercules client library for performing every data access directly over Hercules I/O nodes, avoiding the use of a RAM disk. The main disadvantage is the limited support of existing applications, requiring the modification and re-compilation of every application/tool executed as workflow task.
- Offer the memory managed by Hercules as a storage device, accessed transparently by the workflow tasks. If the Hercules memory subsystem can be mounted as a storage device in every DMCF worker, the applications/tools can access data stored in Hercules in the same way as data stored in any other file system. This approach can be implemented as a FUSE interface or as on-the-fly patching of POSIX I/O operations.

The implementation and evaluation of this approaches is out of the scope of this work, but will be studied in the future.

IV.3 DMCF execution mechanisms and data-aware scheduling

We propose novel workflow-aware task and data placement mechanisms that combine DMCF load-balancing capabilities and Hercules data and metadata distribution functionality for implementing various locality-aware and load-balancing policies. Data placement mechanisms focus in grouping data related to the same task, while the locality-aware scheduler policy targets the co-location of compute task in the nodes where the data can be found in-memory.

In the new execution mechanism proposed the DMCF Worker cyclically checks whether there are tasks ready to be executed in the Task Queue. If so, a task is removed from the Task Queue and its status is changed to 'running'. To take advantage of data locality, the task removed from the queue is the one having the highest number of inputs locally. This differs from the original data-locality agnostic scheduling policy adopted in DMCF, as described in [6], in which each Worker picks and executes the task from the queue following a FIFO policy.

Then, the transfer of all the needed input resources (files, executables and libraries) is performed from their location (Hercules local or remote node) to two local folders and the Worker locally executes the task and waits for its completion.

V. Evaluating the integration between DMCF and Hercules

In this section we show the evaluation results of the integration between DMCF and Hercules. For this evaluation, we have emulated the execution of a data analysis workflow using three alternatives:

- Azure-only scenario: every I/O operation of the workflow is performed by DMCF using the Azure storage service.
- Locality-agnostic Hercules scenario: a full integration between DMCF and Hercules is exploited, where each intermediate data is stored in Hercules, while initial input and final output are stored on Azure. DMCF workers and Hercules I/O nodes share resources (they are deployed in the same VM instance), however, every I/O operations is performed over remote Hercules I/O nodes through the network.
- Locality-aware Hercules scenario: based on the same deployment as the previous case, this scenario simulates a full knowledge of the data location, and executes every task in the same node as the data are stored, leading to fully local accesses over temporary data. Based on this locality exploitation, every I/O operation is performed in-memory instead of through the network.

The goal of this evaluation is to better understand the potential performance improvements in different scenarios where the Hercules I/O accelerator is combined with the DMCF scheduler.

The evaluation is based on a data mining workflow that analyzes n partitions of the training set using k classification algorithms so as to generate kn classification models. The kn models generated are then evaluated against a test set by a model selector to identify the best model. Then, n predictors use the best model to produce in parallel n classified datasets. The k classification algorithms used in the workflow are C4.5 [7], Support Vector Machine (SVM) [8] and Naive Bayes [9], that are three of the main classification algorithms [10]. The training set, test set and unlabeled dataset, which represent the input of the workflow, have been generated from the *KDD Cup 1999*'s dataset[2], which contains a wide variety of simulated intrusion records in a military network environment.

The workflow is composed of $3 + kn + 2m$ tasks. In the specific example, where $n = 20$, $k = 3$, $m = 80$, the number of generated tasks is equal to 223.

Figure 5 shows the VL4Cloud version of the data mining workflow. The visual formalism clearly highlight the level of parallelism of the workflow, expressed by the number of parallel paths and the cardinality of tool array nodes.

Once the workflow is submitted to DMCF using either JS4Cloud or VL4Cloud, DMCF generates a JSON descriptor of the workflow, specifying which are the tasks to be executed and the dependency relationships among them. Thus, DMCF creates a set of tasks that will be executed by workers.

Table 1 lists all the read/write operations performed during the execution of the workflow on each data array. Each row of the table describes: i) the number of files included in the data array node; ii) the total size of the data array; iii) the total number of read operations performed on the files included in the data array; and iv) the total number of write operations performed on the files included in the data array. As can be noted, all the inputs of the workflow (i.e., *Train*, *Test*, *UnLab*) are never written on persistent storage, and the output of the workflow (i.e., *ClassDataset*) is never read.

The simulation results are based on synthetic bandwidth measurements performed over the Azure infrastructure. The benchmark application performs write and read operations over a 256 MB file with a 4 MB chunk size. We have deployed the application on Azure D2_v2 VM instances. The results can be found in Table 3 and represent the expected I/O

[2] http://kdd.ics.uci.edu/databases/kddcup99/kddcup99

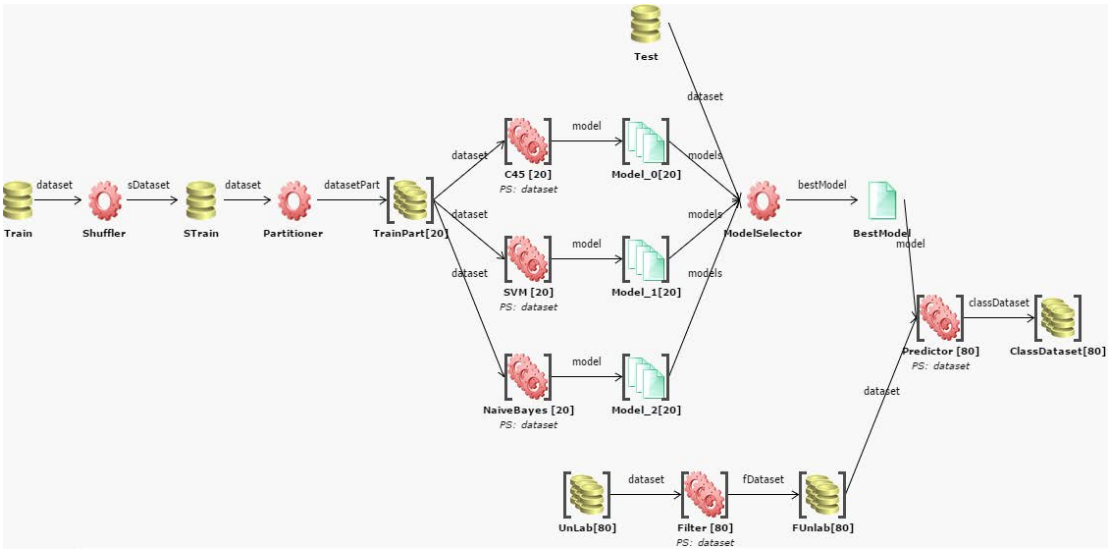


Figure 5: Classification VL4Cloud workflow.

Table 1: Read/write operations performed during the execution of the workflow.

Data node	N. of files	Total size	Number of read operations	Number of write operations
Train	1	100MB	1	-
Strain	1	100MB	1	1
TrainPart	20	100MB	60	20
Model	60	≈20MB	60	60
Test	1	50MB	1	-
BestModel	1	300KB	80	1
UnLab	80	8GB	80	-
FUnLab	80	≈8GB	80	80
ClassDataset	80	≈6GB	-	80

Table 2: Read/write operations performed during the execution of the workflow.

Task Node	N. of istances	Execution times in secs
Shuffler	1	1
Partitioner	1	1
C45	20	288
SVM	20	600
NaiveBayes	20	791
Filter	80	104
ModelSelector	1	9
Predictor	80	2,321

performance of the application when deployed over each evaluated scenario.

Table 3: Synthetic bandwidth measurements performed over the Azure IaaS platform.

Solution	Write op.	Read op.
Azure storage	30 MB/s	60 MB/s
Hercules remote	180 MB/s	175 MB/s
Hercules local	1,000 MB/s	800 MB/s

Figures 6, 7, and 8 show different details of the same experiment where the previously introduced workflow is simulated in different infrastructures. The configurations of the infrastructures range from 1 to 32 DMCF workers. Every DMCF is deployed over a different VM instance, and one Hercules I/O node is deployed on each VM, sharing resources with the DMCF worker. Three different scenarios are studied, as previously presented: Azure-only (labeled as *Azure*), Locality-agnostic Hercules (labeled as *Hercules remote*), and Locality-aware Hercules (labeled as *Hercules local*).

Figure 6 presents an estimation of the total execution time scaling the available resources. The figure shows how the differences in total execution time estimated for every case are narrow, but the cases where the Hercules I/O accelerator is applied are always in front of the Azure-only solution, resulting in up to 8% improvements in total execution times for the best scenarios where the data locality is fully-exploited

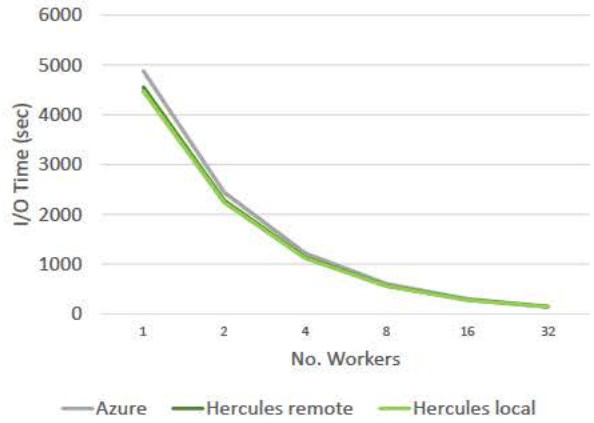


Figure 6: Estimated execution time of the workflow deployed over different scenarios, using up to 32 DMCF workers.

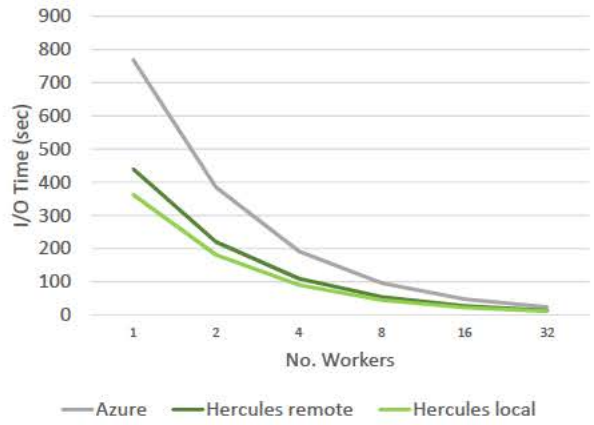


Figure 7: Estimated time required by the application to perform every I/O operation. Different scenarios have been evaluated, using up to 32 DMCF workers.

for temporary files, while 6% reductions in execution time are obtained in locality-agnostic scenarios.

Figure 7 presents an estimation of the time required by the application to perform every I/O operation of the application, and Figure 8 increases the level of detail, showing only the operations affected by the deployment of the Hercules I/O accelerator: I/O operations performed strictly over temporary files. The deployment of Hercules is translated in up to 52% reductions in the time spent in I/O operations when fully exploiting data locality (95% over temporary files) and up to 42% in locality-agnostic scenarios (77% over temporary

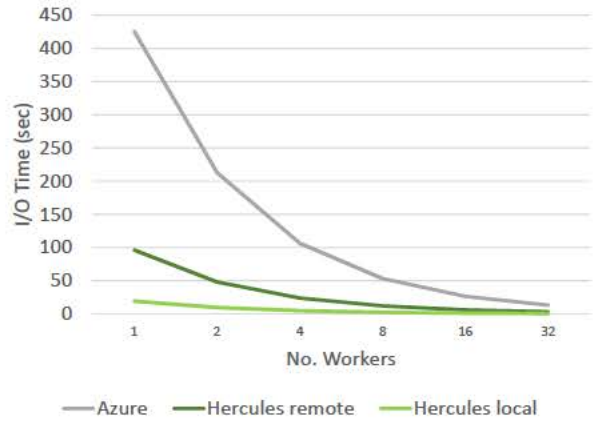


Figure 8: Estimated time required by the application to perform I/O operations strictly over temporary files (I/O operations affected by Hercules). Different scenarios have been evaluated, using up to 32 DMCF workers.

files).

In order to better show the impact of the Hercules I/O accelerator, Figure 9 presents a breakdown of the total execution time, detailing the time spent on each of the tasks executed by the workflow application: computation tasks, I/O tasks over input/results files stored in Azure Storage, and I/O operations performed over temporary files, stored in Hercules when available. This figure clearly shows how the time required for I/O operations over temporary files, the only operations affected by the Hercules accelerator, are reduced to be almost negligible during the execution of the workflow, showing a great potential for increasing the I/O performance in data-intensive applications with large amounts of temporary data. Should be noted how the axis in Figure 9 starts in the second 230, in order to zoom in the top part of the figure, where the I/O times are depicted. The time excluded from the figure (seconds 0 to 230) are spent on CPU operations.

Based on these estimations, we can conclude that the deployment of the Hercules I/O accelerator can greatly benefit the execution of data-intensive applications when a large amount of temporary data is present. The evaluation also shows that, with proper locality-aware mechanisms, the I/O performance can be further improved, exploiting data locality through in-memory computation.

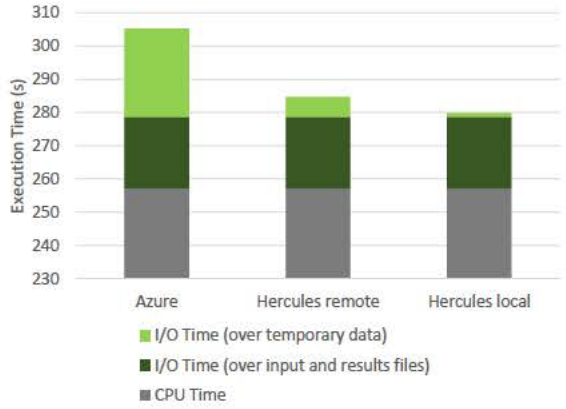


Figure 9: Breakdown of the estimated total execution time deployed over 16 worker nodes sharing resources with 16 Hercules I/O nodes. The breakdown shows the time required by the application to perform compute tasks, I/O tasks over input/result files, and I/O tasks over temporary files (affected by Hercules).

VI. Conclusions

This work presents an evolution of the integration of the Data Mining Cloud Framework (DMCF) with the Hercules in-memory I/O accelerator. The DMCF task scheduler has been improved in combination with Hercules modifications in order to expose and exploit data locality for data-intensive applications.

The evaluation shows an estimation of the potential improvements in I/O performance when data locality is fully exploited during the execution of a data-intensive workflow application deployed over the DMCF-Hercules solution.

Future work should focus on the execution of a real data-intensive application and the evaluation of the improvements achieved by the proposed locality-aware mechanisms. Additionally, an evaluation of the cost of deploying Hercules should be performed, in contrast with the Azure storage-only approach.

Acknowledgment

This work is partially supported by EU under the COST Program Action IC1305: Network for Sustainable Ultrascale Computing (NESUS).

References

[1] Fabrizio Marozzo, Domenico Talia, and Paolo Trunfio. Js4cloud: Script-based workflow programming for scalable data analysis on cloud platforms. *Concurrency and Computation: Practice and Experience*, 27(17):5214–5237, 2015.

[2] Francisco Rodrigo Duro, Javier Garcia Blas, and Jesus Carretero. A hierarchical parallel storage system based on distributed memory for large scale systems. In *Proceedings of the 20th European MPI Users' Group Meeting*, EuroMPI '13, pages 139–140, New York, NY, USA, 2013. ACM.

[3] Brad Fitzpatrick. Distributed caching with memcached. *Linux J.*, 2004(124):5–, August 2004.

[4] Francisco Rodrigo Duro, Fabrizio Marozzo, Javier Garcia Blas, Jesus Carretero, Domenico Talia, and Paolo Trunfio. Evaluating data caching techniques in dmcf workflows using hercules. In *In Proceedings of the Second International Workshop on Sustainable Ultrascale Computing Systems (NESUS 2015)*, pages 95–106, Krakow, Poland, 2015.

[5] Francisco Rodrigo Duro, Fabrizio Marozzo, Javier Garcia Blas, Domenico Talia, and Paolo Trunfio. Exploiting in-memory storage for improving workflow executions in cloud platforms. *The Journal of Supercomputing*, pages 1–20, 2016.

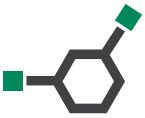
[6] Fabrizio Marozzo, Domenico Talia, and Paolo Trunfio. A workflow management system for scalable data mining on clouds. *IEEE Transactions On Services Computing (IEEE TSC)*, 2016.

[7] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[8] S.S. Keerthi, S.K. Shevade, C. Bhattacharyya, and K.R.K. Murthy. Improvements to platt's smo algorithm for svm classifier design. *Neural Computation*, 13(3):637–649, 2001.

[9] George H. John and Pat Langley. Estimating continuous distributions in bayesian classifiers. In *Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 338–345, San Mateo, 1995. Morgan Kaufmann.

[10] Xindong Wu, Vipin Kumar, J. Ross Quinlan, Joydeep Ghosh, Qiang Yang, Hiroshi Motoda, Geoffrey J. McLachlan, Angus Ng, Bing Liu, Philip S. Yu, Zhi-Hua Zhou, Michael Steinbach, David J. Hand, and Dan Steinberg. Top 10 algorithms in data mining. *Knowl. Inf. Syst.*, 14(1):1–37, December 2007.



cost

Efficient Energy Sources Scheduling in Green Powered Datacenters: A Cloudsim Implementation

Enida Sheme† Patricia Stolf* Georges Da Costa* Jean-Marc Pierson* Neki Frashëri†

Polytechnic University of Tirana, Albania†
esheme@fti.edu.al, nfrasheri@fti.edu.al
University of Toulouse, France*
patricia.stolf@irit.fr, Georges.Da-Costa@irit.fr, jean-marc.pierson@irit.fr

Abstract

In this paper we address the issue of managing different energy sources which supply green powered datacenters. The sources are scheduled based on a priority scheme, aiming to maximize the renewable energy utilization, minimize the energy used from the grid and optimize battery usage. Dynamic power capping technique is used to put a threshold on the drawn energy from the grid. The algorithm is implemented and tested in CloudSim simulator. Renewable energy is considered as solar energy. A workload scheduling algorithm is already implemented for higher renewable energy utilization. The results show that the proposed scheme is efficient and it is a promising direction in the field of the optimization in datacenters using renewable energy.

Keywords green datacenter, renewable energy, grid energy, battery, priority scheme, dynamic power capping, simulator, CloudSim

I. Introduction

Recent studies have addressed the topic of using different sources of energy, mainly renewable one, to supply datacenters. As such, studying the energy sources engagement with resource scheduling has become one of the research directives of this field. Some of the existing techniques take in consideration renewable energy only with batteries and/or grid as a backup. This approach has two disadvantages: first it is not realistic for the current conditions when grid is still the main source of energy in most of datacenters. Second, it requires high capacity of batteries to compensate energy needs in time periods when renewable energy is lacking. This means higher costs, longer charging time and higher environmental risks from battery pollution.

In our paper we propose a new prototype scheme for managing three sources of energy: renewable, grid and battery, following this priority level. The aim is to maximize renewable energy utilization, minimize energy taken from the grid and optimize battery usage. Dynamic power capping technique is used in order to limit the grid power used to supply the datacenter. The proposed algorithm is evaluated through simulations.

The paper is organized as follows. Section II describes the energy context where the new sources scheduling algorithm is studied: the energy consumption of a chosen datacenter and the renewable energy used to supply it. At section III the proposed scheduling scheme is introduced and its implementation in CloudSim simulator is illustrated. Another workload scheduling algorithm is integrated in the simulator, being described at section III.2 . Section IV illustrates the conducted experiments and the results of implementing the proposed priority scheme. The paper finalizes with conclusions at section V .

II. Energy Consumption and Renewable Energy

In this section we evaluate the energy consumption of a datacenter running a given workload, describing datacenter parameters and workload characteristics. The available renewable energy is explored and presented as well, based on real weather data in Tirana, Albania.

II.1 Energy Consumption

The energy consumption is evaluated in a simulator environment, running a chosen workload over a specific datacenter. The datacenter size is chosen based on similar experimental studies in the field of energy efficiency in datacenters. Datacenter represents the processing entity in our system. It runs

the workload and consumes energy, which we track during 24 hours of simulations. Datacenter parameters are chosen based on similar experimental studies in the field of energy consumption in datacenters and typical datacenter size in Albania. To run the simulation we configured the number of hosts equal to 100 and the number of virtual machines running over hosts equal to 200. This means, 2 virtual machines run in every host. The host model is HP ProLiant ML110 G5, Xeon 3075 , processing capacity 2660 MHz, 2 cores and RAM of 4GB. The workload chosen to run the experiments represent a synthetic reproduction of a Google workload, scaled over our own simulating datacenter parameters. A Google trace file was published in 2011, giving detailed information of 12.000 Google servers traffic over 29 days, processing various types of applications. This workload data are studied in order to know its characteristics. The main findings of studies [1], [2], are used in our workload in order to produce patterns that resemble to the Google workload.

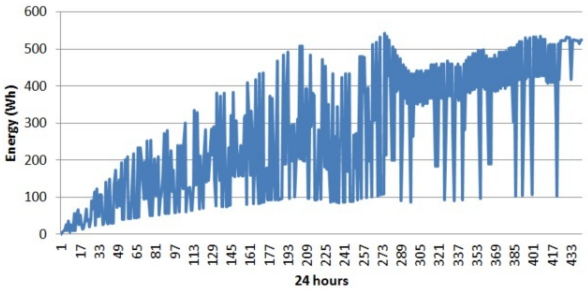


Figure 1: Energy consumption over 24 hours of simulation.

As such, we configure the following workload parameters for our study: total number of jobs, their length, deadline, resource requirements and inter-arrival time. The chosen number of jobs is 400, where 200 of them are short, 150 are medium and 50 are long. The length of short jobs varies from 5 to 7 minutes, medium jobs from 25 to 50 minutes and long jobs from 100 to 300 minutes. The jobs length is generated through Poisson distribution. Deadline is another parameter we set, which is the limit of time it can pass till the job is fully completed. Based on bibliography [1], [2], we categorize jobs into three types of deadline: loose, medium and urgent. 130 of short jobs have loose deadline, which means they are tolerant to be postponed for running in a later moment, 50 of short jobs have medium deadline and 20 are urgent. Out of 150 medium length jobs, 100 of them have loose deadline and 50 have medium deadline. Meanwhile, all long jobs have loose deadline. Loose, medium and urgent deadline is set in proportion to the length of jobs. Regarding resource requirements, half of short jobs require an average of 25% of CPU usage and other half requires 50% of CPU. 50 out of 150 medium jobs require 25% CPU and 100 of them need an average of 50% CPU. While long jobs need to use an average of 80% CPU. The inter-arrival time is set to every 7 minutes for short jobs, every 10 minutes for medium length jobs and every 30 minutes for long jobs.

The energy consumed by the datacenter running the described workload over 24 hours time of simulation is evaluated through CloudSim simulator. The total value of energy consumption is 120 kWh and its distribution through time is shown in figure 1 .

II.2 Renewable Energy

In our study, we used solar energy to represent renewable energy. A. Maraj presents a study regarding solar energy in Tirana [3]. We acquire the solar energy data from the results of this study. The parameters are provided from the database built through the utilization of a data collecting system, which is installed on behalf of the Department of Energy, Faculty of Mechanical Engineering, Polytechnic University of Tirana. Solar power irradiance on a 45 degrees tilted 1 squared meter solar panel, installed over the terrace of the central building of this University, has been collected, providing data for every 5 minutes of its daily operation. We consider a typical clear summer day as input for renewable energy in our experiments. The specific date is July 16, 2010. Further details on solar panel specifications and results of the study are explained in the article [3]. The solar irradiance over 24 hours of a typical summer day in Tirana is shown in figure 2 .

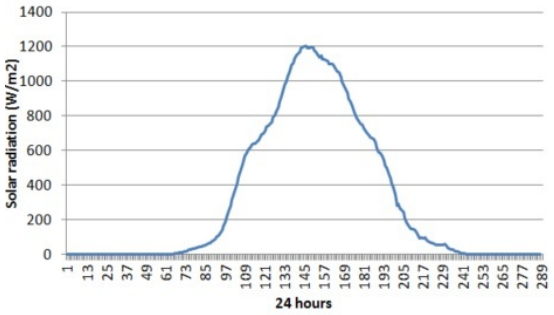


Figure 2: Solar irradiance over 24 hours.

III. Energy Sources Scheduling Scheme

In this section, we describe the working platform, CloudSim simulator, and a workload scheduling algorithm already implemented aiming to maximize the utilization of available renewable energy. We further present the analysis and implementation of the energy sources scheduling algorithm, which is the new prototype scheme we propose in this paper. A detailed explanation of how this algorithm works is described at section III.3 .

III.1 CloudSim Simulator

CloudSim is an extensible simulation toolkit that enables modeling and simulation of Cloud computing systems and application provisioning environments [4]. The CloudSim toolkit supports both system and behavior modeling of Cloud system components such as datacenters, virtual machines (VMs) and resource provisioning policies. Its main functional entities include:

- Hosts: physical machines where the jobs are to be executed.
- Virtual machines: virtual entities running over real physical entities.
- Cloudlets: representing the workload or the jobs to be executed in the datacenter.
- Broker: a scheduler which allocates virtual machines to hosts and cloudlets to virtual machines.

CloudSim is chosen as a simulator because of its high rate in reviews of the energy efficiency in datacenter field, 7 years among researchers and still being widely used, open source code and a rich forum of programmers and researchers.

III.2 Workload Scheduling Algorithm

The algorithm already implemented in CloudSim regarding efficient workload scheduling aims higher leveraging of available renewable energy. The main idea is to postpone non urgent jobs towards periods of time when renewable energy availability is higher. Equivalently, some jobs might be run urgently though they are not urgent in order to exploit current available renewable energy if this level is predicted to be decreasing in the near future. The steps of this algorithm are presented at figure 3 .

Basically, the code is divided in two sections, testing if the available renewable energy is increasing or decreasing. In each case, the behaviour will be different. After testing the urgency of the arrived job, the algorithm decides to run it if it is urgent or postpone it if it is not urgent. The amount of time it will be postponed depends on renewable energy prediction and length of the job. If it is an increasing period, than short jobs are postponed with a time period equal to their length, medium jobs are postponed to medium time between arrival and start deadline time, while long jobs are postponed at their maximum allowed time, as long as it does not violate the desired quality of service. Otherwise, if it is a decreasing period, the behaviour will be contrary to the mentioned approach. Short jobs will be postponed at their maximum, as they require less processing resources, while the long jobs are immediately run in order to use the available solar energy, as figure 3 presents.

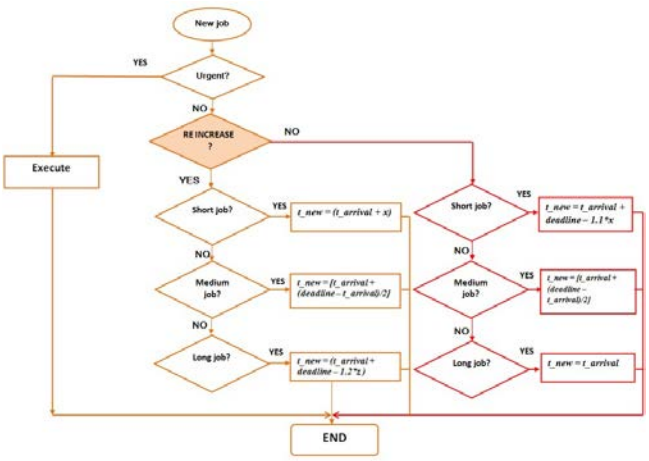


Figure 3: Workload scheduling algorithm.

The result of implementing this algorithm is illustrated in figure 4 . The energy consumption through 24 hours without using the workload scheduling algorithm is compared to the energy consumption after implementing this algorithm. The workload is intentionally modelled in higher intensity in the morning and in the evening to show the benefits of the algorithm. The results show 21% higher utilization of solar energy. More details on this algorithm can be found at article [5].

III.3 Energy Sources Scheduling Algorithm

The energy sources scheduling algorithm works in cooperation with the jobs scheduling algorithm described at section III.2 . Both algorithms are implemented in CloudSim simulator.

Based on prior studies on the field [6], [7], [8], [9] only two sources of energy are used, according to a priority scheme

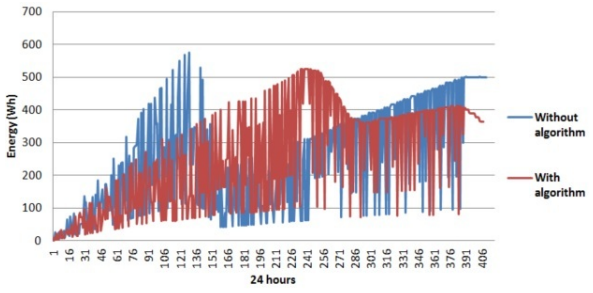


Figure 4: Energy consumption without and with implemented workload scheduling algorithm.

as given in Equation 1 .

$$Consum = RE + battery(+grid) \quad (1)$$

where *Consum* is the energy consumption of the datacenter, *RE* represents renewable energy, *battery* is the energy drawn from battery and *grid* represents energy taken from the grid. First priority is given to renewable energy source, and battery is mainly used as the second source, alternatively combining with grid energy. We evaluate this scheme as not yet realistic and not optimal. The main reason is because: if only grid is used as a backup, high amount of grid energy is needed during times when renewable energy is lacking. If only battery is used a backup, we cannot yet switch to no grid energy systems when nowadays grid energy serves as main source of energy supplier in almost 100% of electric and electronic equippments. Furthermore, we mention 4 drawbacks of batteries to argument why using battery as a second source is not a good choice:

- High capacity battery is needed to compensate required energy during periods when renewable energy is lacking. This means higher costs of using batteries.
- Up to 30% of its produced energy is wasted due to AC/DC convertion.
- Batteries self - discharge.
- Batteries are toxic for the environment.

We propose a new priority scheduling scheme, where first priority is given to renewable energy, second priority to the grid energy, power capped dynamically, and third priority to the battery, as given in Equation 2 .

$$Consum = RE + grid_{\text{capped}} + battery \quad (2)$$

The proposed scheme uses three priority levels for the different sources of energy. The aim is to prioritize renewable energy usage, in order to maximize its utilization which is equivalent to minimizing its waste. First priority is given to available renewable energy. Second, power from the grid is used if energy consumption is greater than the available quantity of renewable energy. However, dynamic power capping, a well-known technique on energy efficiency [10], is applied to the grid energy aiming to limit the power taken from the grid. The dynamic power capping factor is based on the difference between energy consumption and renewable energy. Table 1 describes how the power capping factor is set based on combinations between 3 levels of renewable energy and 4 levels of energy consumption.

Consumption / Renewable	1	2	3
1	2	1	1
2	3	2	1
3	4	3	2
4	4	4	3

Table 1: Power capping factor value based on combinations of energy consumption and renewable energy levels.

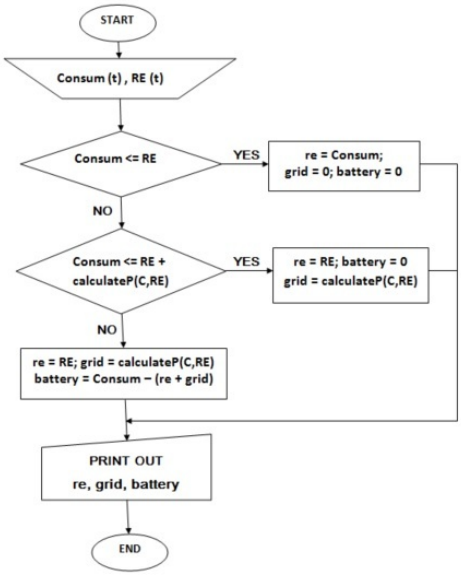


Figure 5: Energy sources scheduling algorithm.

Available renewable energy, which we assume to be known, is divided into 3 levels: 0-33% of its maximum generated

value belongs to level 1, 34%-66% of the maximum belongs to level 2 and 67% - 100% of the maximum belongs to level 3. Likewise, energy consumption, which we suppose it is known, is categorized into 4 levels: 1 means low energy consumption and 4 means very high energy consumption. The levels are set in segments of 0-25%, 26%-50%, 51%-75% and 76%-100% of the maximum value of daily energy consumption. The power capping factor is assigned a value from 1 to 4, accordingly, as illustrated at table 1 : higher is the gap between required energy and available renewable energy higher it is this value and vice versa. Higher power capping value means more power will be drawn from the grid. Dynamic power capping is already used for energy savings purpose in datacenters [10], [11], but it is never used, to our knowledge, in green powered datacenters for energy sources scheduling.

The energy sources scheduling algorithm is explaind in details by the depicted flowchart given at figure 5 . *Consum* represents the energy consumption at a given time and *RE* is the available renewable energy at the same moment of time. Variables *re*, *grid*, and *battery* represent the amount of energy that is drawn from each of the energy sources to supply the datacenter. Given a certain amount of energy consumption and available renewable energy in a given moment of time *t*, first conducted test is to know whether the available renewable energy is enough to meet datacenter energy need. If yes, then all needed energy is taken from renewable source and grid energy and battery is assigned a value equal to 0. If energy consumption is higher than the available renewable energy, then power from grid will be taken. The amount to be drawn from it is calculated from a function named calculateP(C,RE) which assigns the value of power capping factor, according to the logic described at table 1 . If the energy drawn from both sources is not enough for the datacenter energy requirements then the battery will be used. The amount that will be drawn from it is defined by the difference between energy consumption and energy taken from renewable and grid sources. All the three sources are printed out to be used for illustrating and further ellaboration purpose.

IV. Experiments and Results

The experiments aim to assess the result of applying the proposed scheduling algorithm to the energy quantity driven from each of the energy sources used to fulfill the energy requirements of the datacenter. Figure 6 represents the renewable energy utilization over total energy consumption through 24 hours of simulation. The horizontal axis represents 288 5-minutes time intervals of energy consumption in blue and renewable energy shown in red, expressed in Wh unit. The total energy consumption is 120 kWh and the total renewable energy used is 90kWh. This figure is interconnected to figure 7 which illustrates the energy drawn from the grid and from the battery considering the same energy required for the datacenter. We can notice that renewable energy is fully used when it is available, during the day, while grid and battery are used during the night, when solar energy is lacking. It can be clearly seen from the graph that the grid power is capped at 4 levels and battery is used exactly during these periods when renewable energy and limited energy from the grid do not satisfy the need for energy.

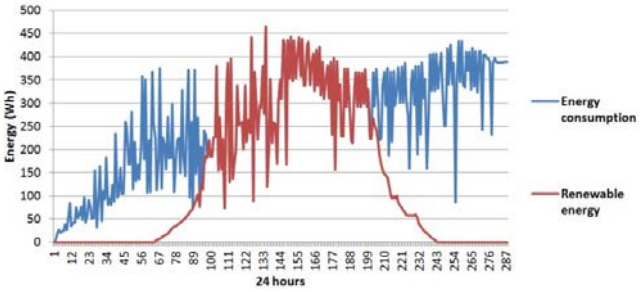


Figure 6: Renewable energy usage and datacenter energy consumption over 24 hours.

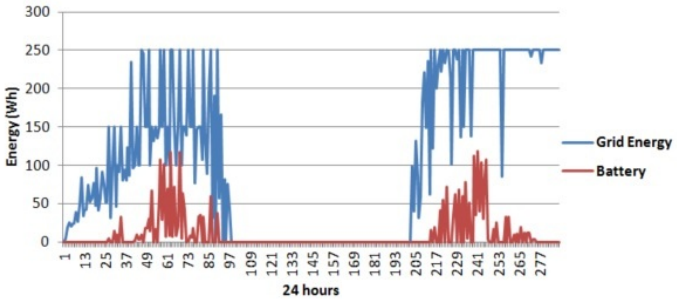


Figure 7: Grid energy and battery usage over 24 hours.

Table 2 presents experiment results regarding the value and percentage of using these three energy sources. We intentionally chose a 75% renewable energy supply scenario, to explore the quantity that would be used from two other sources. The results show that out of 100% of energy consumption, 75% is drawn from the renewable energy source, 20% is taken from the grid and 5% is taken from battery.

Energy source	Renewable	Grid	Battery
$Value(kWh)$	90	25	5
$Percentage(\%)$	75	20	5

Table 2: Value and percentage of energy sources utilization.

V. Conclusions

In this paper we present a new energy sources scheduling scheme in order to maximize the renewable energy utilization, and reduce energy drawn from the grid and batteries. First priority is given to renewable energy, second priority is given to grid energy and third one to batteries. A dynamic power technique is used for capping the energy used from the grid. In a 75% renewable energy coverage scenario, the results of the experiments show that 20% of the required energy is supplied by the grid and 5% is drawn from batteries. The advantage of this algorithm is that it is enough realistic to consider supplying the datacenter with energy from the grid, which is limited by implementing a dynamic power capping technique. On the other hand, we optimize the battery usage by encouraging lower capacity batteries.

The proposed algorithm has higher efficiency if it is implemented over a platform where renewable energy is efficiently and maximally exploited, e.g using a workload scheduling algorithm. Also, a precondition of this new algorithm to function is it assumes energy consumption and renewable energy are already known through prediction.

To conclude, we point out that the proposed scheme is a first prototype. Other elements should be taken in consideration in the future, like: the lifetime of the battery (number of cycles) and other power characteristics. Furthermore, the experiments will be extended to 2 days of simulation in order to monitor jobs that are postponed to the consecutive day.

Acknowledgment

The work presented in this paper has been supported by EU under the COST programme Action IC1305: Network for Sustainable Ultrascale Computing (NESUS).

References

[1] M. Alam, K. Shakil, Sh. Sethi, Analysis and Clustering of Workload in Google Cluster Trace based on Resource Usage, *CoRR Journal*, Volume 1501.01426, 2015.A.

[2] F. Gbaguidi, S. Boumerdassi, E. Renault, E. Ezin, Characterizing servers workload in Cloud Datacenters, In *Proceedings of 3rd International Conference on Future Internet of Things and Cloud (FiCloud)*, 2015

[3] A. Maraj A. Londo, C. Firat, A. Dorri, M. Alcani. Energy investigation of the flat plate solar collector during its daily operation in clear days of summer and winter. *Journal of Natural and Technical Sciences*, Vol XIX (1), Academy of Sciences, Albania, 2014.

[4] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, Cesar A. F. De Rose, Rajkumar Buyya. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Published in *Journal of Software, Practice and Experience*, Vol. 41, Issue 1, New York, USA, January 2011

[5] E. Sheme, N. Frasheri. Implementing workload postponing in CloudSim to maximize renewable energy utilization. In *International Journal of Engineering Research and Application (IJERA)*, Vol. 6, Issue 8, August 2016

[6] Raymond Carroll, Sasitharan Balasubramaniam, Dmitri Botvich, William Donnelly. Application of Genetic Algorithm to Maximise Clean Energy usage for Data Centres. Volume 87 of the series *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, Pages 565-580, 2012.

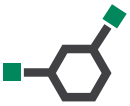
[7] Inigo Goiri, William Katsak, Kien Le, Thu D. Nguyen, Ricardo Bianchini. Parasol and GreenSwitch: Managing Datacenters Powered by Renewable Energy, In *ASPLOS'13*, 2013

[8] M. A. et al., Towards the design and operation of net-zero energy data centers. In *ITherm'12*, 2012.

[9] Sonja Klingert, Florian Niedermeier, Corentin Dupont, Giovanni Giuliani, Thomas Schulze, and Hermann de Meer. Renewable Energy-Aware Data Centre Operations for Smart Cities, the DC4Cities Approach, *International Conference on Smart Cities and Green ICT Systems (SMARTGREENS)*, 2015, Pages 1-9

[10] Hao Chen, Can Hankendi, Michael C. Caramanis and Ayse K. Coskun. Dynamic Server Power Capping for Enabling Data Center Participation in Power Markets. In *Proceedings of the International Conference on Computer-Aided Design ICCAD '13*, San Jose, California, 2013

[11] Simon Seagrave. HP Dynamic Power Capping. Available online through URL http://techhead.co/hp-dynamic-power-capping/, 2008



cost

Heterogeneous computation of matrix products

Pedro Alonso
Universitat Politècnica de València, Spain
palonso@upv.es

Ravi Reddy Manumachu
University College of Dublin, Ireland
ravi.manumachu@ucd.ie

Alexey Lastovetsky
University College of Dublin, Ireland
alexey.lastovetsky@ucd.ie

Abstract

The work presented here is an experimental study of performance in execution time and energy consumption of matrix multiplications on a heterogeneous server. The server features three different devices: a multicore CPU, an NVIDIA Tesla GPU, and an Intel Xeon Phi coprocessor. Matrix multiplication is one of the most used linear algebra kernels and, consequently, applications that make an intensive use of this operation can greatly benefit from efficient implementations. This is the case of the evaluation of matrix polynomials, a core operation used to calculate many matrix functions, which involve a very large number of products of square matrices. Although there exist many proposals for efficient implementations of matrix multiplications in heterogeneous environments, it is still difficult to find packages providing a matrix multiplication routine that is so easy to use, efficient, and versatile as its homogeneous counterparts. Our approach here is based on a simple implementation using OpenMP sections. We have also devised a functional model for the execution time that has been successfully applied to the evaluation of matrix polynomials of large degree so that it allows to balance the workload and minimizes the runtime cost.

Keywords Matrix multiplication, heterogeneous system, energy consumption, matrix polynomials

I. Introduction

Matrix multiplication is one of the most essential computational kernels used in the core of scientific applications. This operation has been highly studied in the past in order to improve the efficiency of its computation in both sequential and parallel computer architectures. It has also received full attention in parallel heterogeneous environments. Many contributions in this context basically propose irregular partitions of the factor matrices that can efficiently be mapped on the computing resources; see for instance [10, 16, 12].

It is difficult to find actual implementations of the matrix multiplication on heterogeneous nodes that feature very different devices. The MAGMA project, for instance, aims to develop a dense linear algebra library similar to LAPACK but for heterogeneous/hybrid architectures; it is one of the most active projects that implement BLAS routines for nodes featuring accelerators [22]. Currently, MAGMA implements a version for NVIDIA GPUs in which the matrix multiplication is carried out only by the GPUs, i.e. the CPU does not intervene. The MAGMA project also provides with a version, MAGMA MIC, which provides hybrid algorithms that involve the host CPU and one or more Intel Xeon Phi processors. However, this project does not use both NVIDIA GPUs and MICs processor all together in the same host. Authors of [13] propose a programming model for heterogeneous computers featuring CPU, a GPU and a Xeon Phi with the aim to incorporate it to MAGMA library. However, they have not shown its proposal with matrix multiplication. Hence and as a starting point, we propose here a simple implementation to carry out parallel heterogeneous matrix multiplications in a node composed by CPU cores, one NVIDIA GPU, and one Intel Xeon Phi.

As it is explained in the next section, in this paper we are interested in evaluating matrix polynomials of only square matrices. Section III shows the application implemented to carry out a square matrix multiplication on these three different devices. The following section shows experimental results both in time and energy consumption of our application. In Section V we propose a model to implement a heterogeneous matrix multiplication routine that can exploit easily the underlying hardware. We finish the paper with some conclusions and proposals for future research.

II. Matrix polynomials

An application for matrix multiplications is, for instance, the calculus of matrix polynomials. Matrix polynomials are used, e.g. for the computation of functions of matrices [9] by the Taylor method. A matrix function is the exponential of a matrix [21]. This function appears in the solution of many engineering and physics phenomena which are governed by systems of linear first-order ordinary differential equations with constant coefficients [15]. Also, the matrix exponential appears in other scientific contexts like, e.g. control theory [14] or theory of multimode electric power lines [24]. Some other engineering processes are described by second order differential equations, whose exact solution is given in terms of the trigonometric matrix function sine and cosine [11, 17].

There are different techniques for computing or approximating matrix functions. Some of them are very general but others are specialized to particular functions. Two techniques are widely used to approximate a matrix function, one is based on polynomial approximations and the other is based on rational approximations. The one based on polynomial approximations makes intensive use of matrix multiplications. For example, the matrix exponential can be calculated efficiently by using Taylor series [21], which is in turn formulated as a matrix polynomial. Other trigonometric matrix function is the cosine of a matrix. This function has been tackled in [20] to show that it is possible to perform its computation in a

Algorithm 1 Algorithm for the evaluation of a matrix polynomial.

```
1: function EVALUATE( n, X, d, α ) return P
2:     P ← α_0 I
3:     P ← P + α_1 X
4:     B ← X
5:     for i ← 2, d do
6:         A ← B
7:         B ← X · A
8:         P ← P + α_i B
9:     end for
10: end function
```

very efficient way by using also a Taylor series approximation. As it has been shown in [19] it is possible to obtain more accuracy with polynomial approximations than with rational approximations with similar or even lower computational cost. Another advantage of using this technique is due to the fact that the most expensive operations are all matrix products and there exist many libraries that provide efficient implementations of this operation in different environments. For instance, one can find very optimized implementations for multicore processors in Intel MKL, OpenBLAS, or BLIS. Another example is CUBLAS, a library that includes a very efficient routine to perform matrix multiplications in NVIDIA GPUs. This library was recently used in [9] to implement the algorithm proposed in [20] that computes the cosine of a matrix using one or two GPUs.

A matrix polynomial P of degree d can be defined as

$$\begin{aligned} P &= \sum_{i=0}^{d} \alpha_{d-i} X^{d-i} \qquad (1) \\ &= \alpha_d X^d + \alpha_{d-1} X^{d-1} + \cdots + \alpha_1 X + \alpha_0 I \,, \end{aligned}$$

where $X, I \in \mathbb{R}^{n \times n}$, being I the identity matrix. The polynomial matrix X can be arbitrary large, e.g. when appears in the solution of PDEs related to fluid dynamics. Also the polynomial degree d can be very large, e.g. 30 is a common number in the calculus of a trigonometric matrix function [19].

In theory, the evaluation of a matrix polynomial with the form (1) is quite straightforward by using, for instance, Algorithm 1. There exist algorithms that allow to reduce the total number of matrix products by means of the so called Paterson–Stockmeyer method [9]. However, these methods also need an efficient implementation of the matrix product. In any case, the efficiency of the evaluation of a matrix polynomial depends on how efficient the underlying matrix multiplication routine is. When the computational resources are the cores of a multicore, we rely on *threaded* routines (e.g. Intel MKL) that exploit all the CPU cores concurrently to perform this computation transparently to the user. But things are more complicated in a heterogeneous environment, where the computational resources are different among them and, in turn, are "far away" from the main memory where data initially reside.

III. A Hybrid Matrix Multiplication Application

In order to solve efficiently problems like the evaluation of matrix polynomials that intensively use matrix multiplications on a heterogeneous server, we propose an implementation for a matrix multiplication application and present an experimental study of its performance in both execution time and energy consumption.

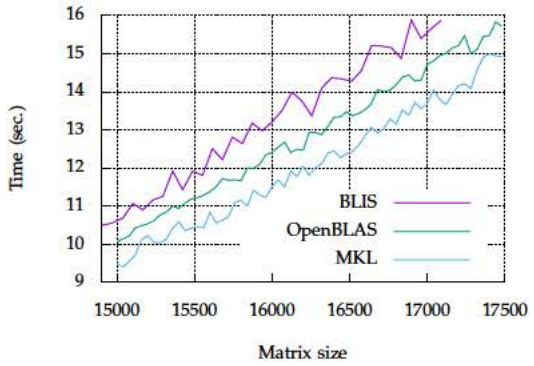


Figure 1: Execution time for a square matrix multiplication in one core using different libraries.

III.1 The Hardware and Software Used

The heterogeneous server we have been working with features the following devices:

- **CPU**: Two sockets with an Intel Xeon CPU E5-2670 v3 at 2.30 GHz each. This processor has 12 cores so the server contains a total of 24 threads. The main memory of the host is 64 GB.
- **GPU**: NVIDIA Tesla K40c with 2880 cores and 12 GB of device memory.
- **PHI**: An Intel Xeon Phi 3120A coprocessor with 57 processors (228 cores) and 6 GB of device memory.

Although the term "device" is usually assigned to accelerators only, i.e. GPU and PHI, in the next and for the sake of simplicity, we will use it to denote the three of them.

On the software side, we have within reach different implementations of BLAS [1] to perform the matrix multiplication:

MKL: Intel Math Kernel Library is an optimized implementation of linear algebra routines contained in BLAS and LAPACK, and other mathematical functions like the FFT. This library is available for multicore x86 processors, and also for the Intel Xeon Phi coprocessor [3]. There exist "threaded" routines, e.g. the matrix multiplication routine `GEMM`, for both devices.

OpenBLAS: OpenBLAS is an optimized BLAS library based on GotoBLAS2 1.13 BSD version [4]. Used in the CPU.

BLIS: This library is self-described as "a portable software framework for instantiating high-performance BLAS-like dense linear algebra libraries". In addition the "framework was designed to isolate essential kernels of computation that, when optimized, immediately enable optimized implementations of most of its

commonly used and computationally intensive operations" [23]. The library is accessible in [2]. It has been used in the CPU.

CUBLAS: BLAS implementation for NVIDIA GPUs [8].

We performed a simple experimental analysis of the speed of the matrix multiplication (`GEMM`) in the CPU (Figure 1). For this test we used the maximum available CPU cores, i.e. 24. (We ignored the fact that Hyper-Threading (HT) can be enabled to give a total of 48 logical processors. We observed that using just one thread per core is enough to fully exploit the execution resources of the core and not increase in performance can be achieve by activating HT.) It must be said that the performance of BLIS could be probably better by selecting the best parallel configuration. Contrary to the other two packages, BLIS is tuned by setting the value of up to four environment variables. That value corresponds to the number of threads that will be used to parallelize a given loop among the five nested loops in which the matrix multiplication is implemented in order to exploit the hierarchical set of intermediate memories of the most current architectures. In this test, only the outer loop was parallelized. A more suitable combination of values are likely to produce a better performance of BLIS, however, we decided not to test the large set of different combinations with the idea that barely the performance would outperform MKL in this machine. Consequently, we consider the performance of Intel MKL to be the best and, therefore, it is the only library used on the CPU side.

III.2 Implementation option

To proceed towards a heterogeneous matrix product, we started by implementing an application that partitions the problem into three concurrent pieces so that the three devices can cooperate in the solution. There exist different options to implement such an application. However, all the options can be gathered into two main classes standing for the use of light processes (threads), or heavy processes. The last option can be implemented e.g. by using MPI [7]. Here, we decided to use a simple approach based on threads, which are spawned by means of OpenMP sections.

The application has been implemented with OpenMP sections, so that each device code is included in a given section (Listing 1). The code for the Intel Xeon Phi, in lines 31–32, is implemented in a different source file (Listing 2) and compiled separately. This is because it is necessary to compile this code with the Intel C compiler (`icc`). For the compilation of the rest of the C code of the application we used the GNU compiler (`gcc`) since there exists incompatibility between the available versions for the NVIDIA compiler (`nvcc`, version 7.5) and for the Intel compiler (`icc`, version 16.0).

The basics of the heterogeneous multiplication are easy. To perform the multiplication $C = AB$, matrix A is completely broadcast to the two accelerators from the Host computer. Matrix B, however, is partitioned into three blocks of consecutive columns. The second block is uploaded to the GPU, the third one is uploaded to the PHI, and the first one remains into the host memory. The amount of columns of each block is denoted in Listing 1 by the values of variables `gpu_n`, `phi_n`, and `cpu_n` for the GPU, the PHI, and the CPU, respectively. Currently, the application receives these values as arguments by command line, in particular, the user sets the percentages for the GPU and for the PHI in the range $[0, 1]$, the rest is computed by the CPU. Upon termination of the execution,

```
1  int gpu_n  = (int) (gpu_weight * n);
2  int phi_n  = (int) (phi_weight * n);
3  int cpu_n  = n-gpu_n-phi_n;
4  #pragma omp parallel sections num_threads(3)
5  {
6  #pragma omp section
7  { // GPU
8    if( gpu_n ) {
9    cublasHandle_t handle;
10   CUBLAS_SAFE_CALL( cublasCreate(&handle) );
11   double *gpu_A, *gpu_B, *gpu_C;
12   CUDA_SAFE_CALL( cudaMalloc((void **) &gpu_A, n*n*sizeof(double) ) );
13   CUDA_SAFE_CALL( cudaMalloc((void **) &gpu_B, n*gpu_n*sizeof(double) ) );
14   CUDA_SAFE_CALL( cudaMalloc((void **) &gpu_C, n*gpu_n*sizeof(double) ) );
15   CUBLAS_SAFE_CALL(cublasSetMatrix(n, n, sizeof(double), A, n, gpu_A, n ));
16   CUBLAS_SAFE_CALL( cublasSetMatrix( n, gpu_n, sizeof(double),
17                                      &B[n*cpu_n], n, gpu_B, n ) );
18   CUBLAS_SAFE_CALL( cublasDgemm(handle, CUBLAS_OP_N, CUBLAS_OP_N, n, gpu_n,
19                         n, &alpha, gpu_A, n, gpu_B, n, &beta, gpu_C, n ) );
20   CUBLAS_SAFE_CALL( cublasGetMatrix( n, gpu_n, sizeof(double), gpu_C, n,
21                                      &C[n*cpu_n], n ) );
22   CUDA_SAFE_CALL( cudaFree(gpu_A) );
23   CUDA_SAFE_CALL( cudaFree(gpu_B) );
24   CUDA_SAFE_CALL( cudaFree(gpu_C) );
25   CUBLAS_SAFE_CALL( cublasDestroy(handle) );
26   }
27 }
28 #pragma omp section
29 { // PHI
30   if( phi_n ) {
31   gemmPHI( n, phi_n, n, alpha, A, n, beta, &B[n*(cpu_n+gpu_n)], n,
32            &C[n*(cpu_n+gpu_n)], n );
33   }
34 }
35 #pragma omp section
36 { // CPU
37   if( cpu_n ) {
38   dgemm( &transa, &transb, &n, &cpu_n, &n, &alpha, A, &n, B, &n,
39                                                    &beta, C, &n );
40   }
41 }
42 }
```

Listing 1: Code for the heterogeneous matrix multiplication.

the resulting matrix C appears partitioned and distributed among the three devices. We include in the application, and in the time measurement, the operation of gathering the result in the memory location allocated into the host to store the resulting matrix.

The code for the execution in the GPU is quite regular (Lines 7–27). It includes creation of the CUBLAS context, allocating memory for the three matrix factors, uploading matrices, executing the matrix product, downloading the result, and freeing the resources involved in the computation.

For the Xeon Phi, we used the "offload mode" of computation, that is, data is explicitly uploaded to the device and the operation is also explicitly executed there. Thus, the programmer have control of what exactly is executing the coprocessor. Arguments `in`, `out`, and `inout` specify clearly the direction of variables characterized by those words. The operation is actually performed by calling to the BLAS matrix multiplication routine using the MKL version.

Finally, the code executed by the CPU only includes a call to the `gemm` routine (lines 38–39) for the matrix computation using MKL as well. We used the `fortran` interface instead of the C one used for the PHI for no specific reason but the application is oblivious of this.

Attention must be paid to the way in which the application is executed in our heterogeneous server. As it has been implemented, only three OpenMP threads are created so that each one will execute a different section. There will be, thus, one thread bound to each accelerator for data transference and control purposes. For the CPU case, however, the execution of the MKL routine will use only

```
void gemmPHI( int m, int n, int o, double alpha, double *A, int lda,
              double beta, double *B, int ldb, double *C, int ldc ) {
#pragma offload target(mic) in(m,n,o,alpha,beta,lda,ldb,ldc) \
              in(A:length(m*o)) in(B:length(o*n)) inout(C:length(m*n))
{
  cblas_dgemm(CblasColMajor, CblasNoTrans, CblasNoTrans, m, n, o,
                             alpha, A, lda, B, ldb, beta, C, ldc );
}
}
```

Listing 2: Code for the heterogeneous matrix multiplication in the Xeon Phi (offload mode).

one thread. To use more threads (cores) collaborating in the matrix multiplication on the CPU side, the "nested parallelism" ability must be explicitly set. In addition, there are more environment variables that control the behaviour of the application (Table 1).

This is an example of execution:

```
shell_$ MKL_NUM_THREADS=22 OMP_NESTED=TRUE MKL_DYNAMIC=FALSE
        MKL_MIC_ENABLE=0 MIC_ENV_PREFIX=MIC
        MIC_KMP_AFFINITY=balanced,granularity=fine
        MIC_OMP_NUM_THREADS=228 MIC_USE_2MB_BUFFERS=64K
        numactl --physcpubind=+0-11,12-23 program 10000 0.48 .15
```

The example executes the program `program` which generates two random matrices of order $n = 10000$. The GPU performs 48% of the computation, 15% is carried out by the PHI, and the rest, 37%, is computed by the CPU.

It should also be noted that the server has the hyperthreading enabled, but we decided not to use all the 48 threads and always use 24 as a maximum number of threads instead. For instance, when operating with the three devices, two threads are bound to one accelerator each, leaving the other 22 for the execution of the matrix multiplication in the CPU.

In addition, we have always used core affinity. This is to prevent threads from leaping amongst the cores at runtime, so as to reduce the variability of the execution times and also to improve the performance of all the devices attached to the host. Concretely speaking, we use the tool `numactl` to bind threads to cores.

The following is an example of the output of the application:

```
n = 10000 (CPU = 38.00% GPU = 50.00% PHI = 12.00%) [4 reps]
          (cpu = 1.17 sec. gpu = 1.14 sec. phi = 1.19 sec.)
          (1.30 sec. 1541.47 gflops)
```

for a random matrix of size $n = 10000$. The weight used for each device in this example results in a workload rather well balanced.

III.3 Energy consumption

We are also interested on evaluating the energy consumption of the devices participating in the matrix multiplication with the aim at, first, understanding the power trace of each device and, second, exploring a workload distribution which can result in energy savings.

For the energy measurement, we have used a tool called `powerrun` [5]. This tool is a wrapper to other tools for measuring the power draw of the CPU (uses PAPI and Intel PCM), of the GPU (uses NVML [18]), and of the PHI (uses Intel's MPSS [6]). The tool gathers the power samples of all the devices under operation and dumps a power trace to a file to compute the energy consumed during the execution time. This tool provides a library to instrument the code under test with simple calls that frame the part of the code to be measured.

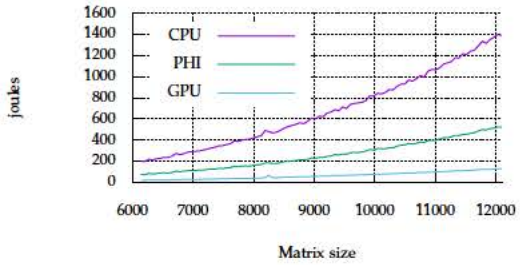


Figure 2: Energy consumption when executing a matrix multiplication in the CPU and the other two devices remain idle.

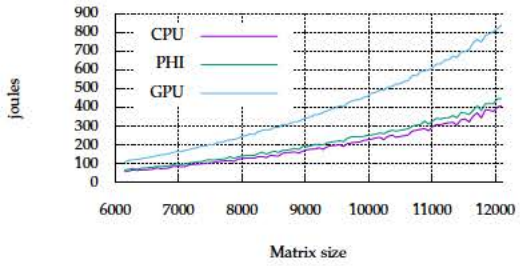


Figure 3: Energy consumption when executing a matrix multiplication in the GPU and the other two devices remain idle.

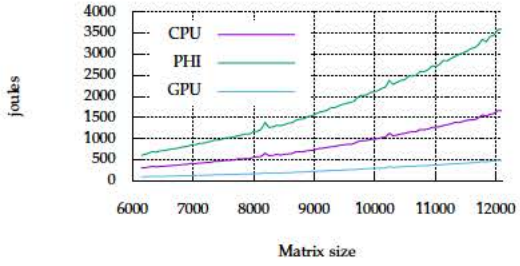


Figure 4: Energy consumption when executing a matrix multiplication in the PHI and the other two devices remain idle.

We provide here three tests that show the energy consumption (in joules) of the three devices, respectively. In each test, the three devices are operating concurrently. Only one of them is working on a matrix multiplication while the other two remain idle. The test samples the energy of the three devices.

Figure 2 shows the energy consumed by the system when only the CPU is "working" and is rather easy to interpret. The CPU is the most consuming device since it is the only one that performs useful work, while the other two consume the energy in idle state. It is also quite clear the difference in energy consumption when idle between the two accelerators, being very low in the case of the NVIDIA GPU

Variable name	Meaning
`OMP_NESTED`:	Set to `TRUE` to ensure that MKL uses more than one thread when called inside an OpenMP section.
`MKL_NUM_THREADS`:	Number of threads used by MKL (CPU).
`MKL_DYNAMIC`:	Set to `FALSE` to avoid MKL automatically selects the number of threads (CPU).
`MKL_MIC_ENABLE`:	Set to 0 to avoid the Xeon Phi is used to accelerate the CPU computation.
`MIC_ENV_PREFIX`:	Specifies the environment variables with prefix `MIC` will address only the PHI.
`MIC_OMP_NUM_THREADS`:	Number of threads used by the PHI to execute MKL routines.
`MIC_KMP_AFFINITY, MIC_USE_2MB_BUFFERS`:	These variables control the efficiency of the Xeon Phi in the execution of the matrix multiplication routine. They have been set to such values according to the advice of Intel documentation.

Table 1: Meaning of shell variables used to execute the heterogeneous matrix multiplication application.

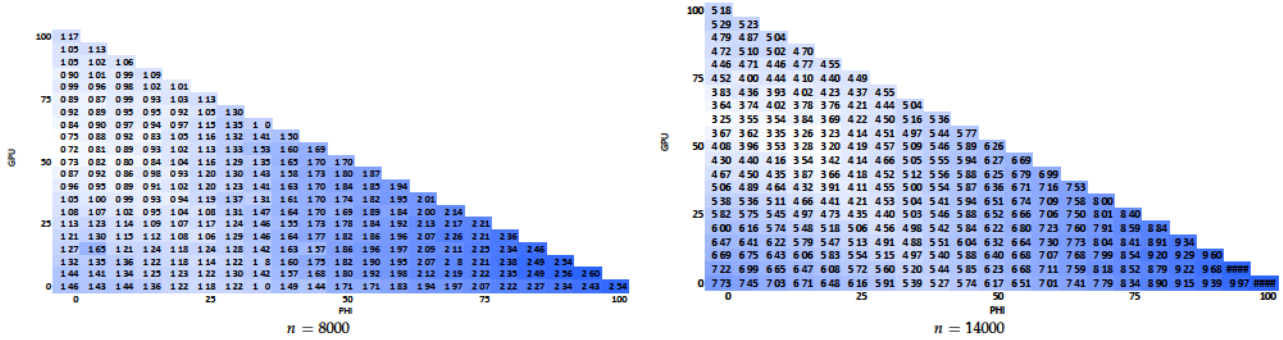


Figure 5: Execution time in seconds for a matrix product varying the weight of workload on each device.

compared with the Intel Xeon Phi.

Figure 3 shows the energy consumption when the GPU is the only device operating on a matrix multiplication. Note that one of the cores of the CPU is also working since it is in charge of sending the two matrices to be multiplied and receiving the resulting one. The consumption of the Intel Xeon Phi is very large in idle state when compared with the CPU.

As expected, the consumption of the Intel Xeon Phi is quite large when executing the matrix multiplication (Figure 4). Also in this case one of the cores of the CPU is working to feed the coprocessor with the two factor matrices and to receive the solution matrix.

IV. Experimental Results of the Matrix Multiplication Application

Figure 5 shows the execution time in seconds spent by the application to perform a matrix multiplication of two square matrices of sizes $n = 8000$ and $n = 14000$. The two graphics show times for different weight combinations. The percentage of computation carried out by the GPU is shown on the y-axis, while the work done by the PHI is shown on the x-axis. These two values are selected by the user. The rest of the computation is performed by the CPU. The figure shows less execution times (clearer cells) within the region between $\approx 25\%$ and $\approx 50\%$ for the GPU, and $\lessapprox 20\%$ for the PHI in the case of the problem sizes selected. There exists more opportunity for the PHI to participate as long as the problem size increases.

Figure 6 shows the percentages of the minimum values obtained for the problem sizes $n = 8000, 10000, 12000, 14000$, which are 0.72 sec., 1.30 sec., 2.08 sec., and 3.20 sec., respectively. For large problems

	$n \in [2000, 10000]$	$n \in [10000, 14000]$
$w_{\text{phi}} =$	0	$\frac{n}{200} - 50$
$w_{\text{gpu}} =$	$\frac{n}{800} + 47.5$	$-\frac{n}{400} + 85$
$w_{\text{cpu}} =$	$100 - w_{\text{gpu}}$	$100 - (w_{\text{phi}} + w_{\text{gpu}})$

Table 2: Functions of the weight for each device for the execution time of the matrix multiplication.

both the CPU and the GPU reduce their weight to make room for the PHI, which does not contribute to the task with any size smaller than $n = 12000$. We can approximate the weight of each device, i.e. w_{cpu}, w_{gpu}, and w_{phi}[1], by the two linear functions shown in Table 2 for two intervals. By means of a larger experimental setup we could easily devise a functional model that allows to predict the best percentage of workload to be mapped on each device. However, we must take into account that there exist a problem size not very much smaller than $n = 2000$ for which it is not worthwhile to use the GPU. Also, for problem sizes $n > 14000$, the weight to be assigned to each device stabilizes around a fix value ($w_{\text{phi}} \approx 15\%$ and $w_{\text{ghi}} \approx 55\%$). However, as the problem size increases a little more, out-of-core algorithms are required and these functional models can significantly change.

Things are slightly different when we observe the total energy consumed by the matrix multiplication application. The minimum values of energy (in joules) are 379, 700, 1177, and 1783, for the problem sizes 8000, 10000, 12000, and 14000, respectively. Figure 7

[1]Note that the number of matrix columns assigned to a device d is $n_{\text{d}} = n \cdot w_{\text{d}}$, where d = cpu, gpu, phi.

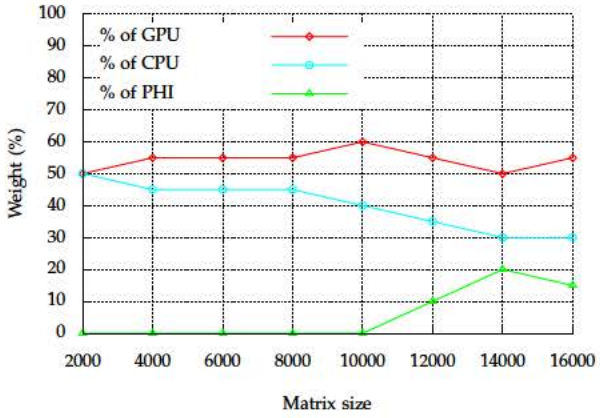


Figure 6: Functional model in graphics for the execution time of the matrix multiplication.

shows, as an example, the energy consumption with problem sizes $n = 8000$ and $n = 14000$. The corresponding weights of w_{gpu} in which we can find these minimum values are 55%, 60%, 60%, and 60%, for each problem size, respectively, and 0% for w_{phi}. These numbers show that while Intel Xeon Phi can contribute a little to reduce the execution time, it can contribute nothing towards reducing the energy consumption in any case, as it was expected according to Figures 2–4. The NVIDIA GPU is, currently, a more efficient device for HPC. In this particular server and for large problem sizes ($n > 10000$), the Intel Xeon Phi is used as a trade-off between execution time and total energy consumption.

We also figured out the dynamic energy of the application, i.e. the energy due to the execution of the application and that we obtain after taking away the energy consumed by each device in idle state. The results showed that we can find the minimum value for the dynamic energy for all problem sizes when none of the accelerators are used. This is due, on one hand, to the high energy consumption of the PHI and, on the other hand, to that the NVIDIA GPUs has two different performance states (when idle) that are difficult to control and disturb the actual energy measurement when the device is idle.

V. A Heterogeneous Matrix Multiplication System for Evaluating Matrix Polynomials

For this part of the work we have got a representative application of matrix polynomials, i.e. that intensively uses matrix multiplications. This application has been recently developed and it allows to compute the cosine of a matrix [9]. Implementing this application on the top of a heterogeneous matrix multiplication routine allows to get the most out of a heterogeneous computer.

The task of developing a program that solves this problem poses a big challenge from the performance point of view, as we showed before, but also from the programmability point of view. Thus, in order to make as easy as possible the programming task we propose a system based on three different kind of objects which

Algorithm 2 Heterogeneous algorithm for the evaluation of a matrix polynomial.

```
1:  function EVALUATE( n, X, d, α ) return P
2:      P ← α_0 I
3:      P ← P + α_1 X
4:      P_D ← P
5:      X_R ← X
6:      B_D ← X_D
7:      for i ← 2, d do
8:          A_D ← B_D
9:          B_D ← X_R · A_D
10:         P_D ← P_D + α_i B_D
11:     end for
12: end function
```

represent matrices. Let M be a square matrix stored into the host main memory, then we report the following definitions:

- *Regular matrices*: these matrices are uniquely stored into the CPU main memory, e.g. the matrix M itself.
- *Replicated matrices*: these matrices, denoted by subscript R (e.g. M_R), are replicated into all the three devices.
- *Distributed matrices*: these matrices, denoted by subscript D (e.g. M_D), are partitioned in column blocks and scattered into all the three devices.

We use these objects to rewrite Algorithm 1 into its heterogeneous counterpart, Algorithm 2 . In the heterogeneous algorithm, each matrix object is characterized by its condition according to where the entries of this matrix are stored, i.e. *regular*, *replicated* or *distributed*.

We also describe the communication operation that takes place between each pair of matrix-types as follows:

- $M \rightarrow M_R$: This is a *Broadcast* communication.
- $M \rightarrow M_D$: This is *Scatter* communication.
- $M_R \rightarrow M$: This is a dummy operation since the destination matrix is already into CPU memory and can be implemented through a local copy.
- $M_R \rightarrow M_D$: This is a local copy of the right data partition.
- $M_D \rightarrow M$: This is a *Gather* communication.
- $M_D \rightarrow M_R$: This is an *Allgather* communication.

We make the assumption that there exists just one distribution for all the distributed matrices involved, and this distribution, represented by the tuple $(w_{\text{cpu}}, w_{\text{gpu}}, w_{\text{phi}})$, has been previously calculated and it is known before executing the algorithm. There are conversion operations between matrix types in steps 4 and 5. Steps 6 and 8 are local copies of the proper data objects, and Step 10 can be readily implemented calling to the BLAS `saxpy` routine. Step 9 is the matrix multiplication of the replicated matrix X_R by the distributed matrix A_D, and this is just the multiplication tackled in Section III.

Figure 8 represents the evolution of runtime with regard to the polynomial degree for the evaluation of two polynomials of size

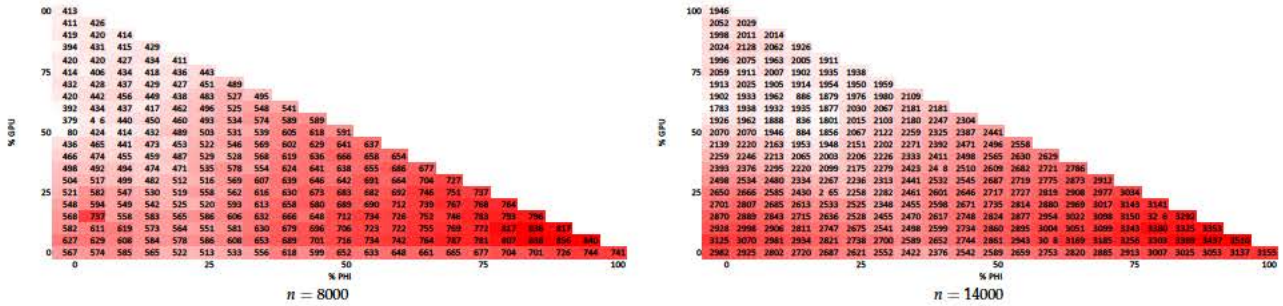


Figure 7: Energy consumption in joules for a matrix product varying the weight of workload on each device.

$n = 10000$ and $n = 14000$, respectively, of random coefficients. The figure shows the execution times using only the CPU versus using the three devices. For the second case, we selected the distribution tuple suggested by Figure 6 and Table 2 for each problem size. The figure demonstrates that the evaluation of matrix polynomials can be speeded up significantly by using all the devices in the heterogeneous platform.

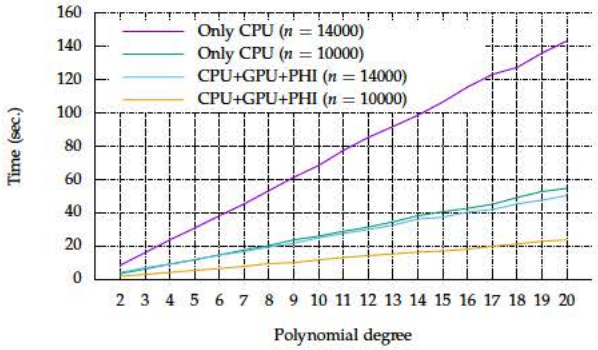


Figure 8: Execution time for the evaluation of matrix polynomials.

VI. Conclusions and future work

This work has presented an application for matrix multiplications in a heterogeneous node composed by a multicore CPU and two very different accelerators. We have shown that it is not difficult to implement the application using OpenMP sections. However, the incompatibility among compiler versions can make this task a bit cumbersome, and in addition, selecting the exact suitable value for the large number of environment variables is an arduous task that highly affects the performance of the application.

We have reduced the study of our application to a particular case in which all matrices involved are square. This case is motivated by our aim to evaluate efficiently matrix polynomials, which is the core operation to obtain matrix functions using the Taylor method. However, the study can be extended to rectangular matrices with little effort. We have developed a functional model for the runtime so that we can select the proper amount of work to do by each device. In our node, the K40 is the most speedy device, far more than the Xeon Phi, which only has opportunity to contribute to the computation on matrices larger than $n = 10000$. Furthermore, the Xeon Phi is currently the most expensive device in terms of energy consumption, and the K40 is the most energy efficient. Our study on the energy consumption resulted in a quite simple behaviour, i.e. the lowest total energy consumption is achieved when the GPU is used in a similar proportion as that selected to achieve the lowest execution time, provided the Xeon Phi is not used at all. It was impossible to obtain an accurate measure of dynamic energy due to specific behaviour of the GPU, which changes between two different performance states (when idle) in an unpredictable way.

Finally, we proposed a heterogeneous matrix multiplication system to make easy the programmability of algorithms based on a heterogeneous matrix multiplication. The system was successfully applied to obtain rapidly an application for the evaluation of matrix polynomials. We plan for the future to generalize this system so that we can perform products of the form

$$C_X \leftarrow \beta C_X + \alpha A_Y B_Z,$$

being $X, Y, Z \in \{\text{none}, \text{R}, \text{D}\}$, i.e. products that involve any type of matrix distribution.

Finally, our aim is to extend everything carried out in this work as fast and easy as possible to host the new upcoming FPGA device.

Acknowledgment

This research has been supported by EU under the COST programme Action IC1305, 'Network for Sustainable Ultrascale Computing (NESUS)'.

References

[1] BLAS: Basic linear algebra subprograms library. `http://www.netlib.org/blas`. Accessed: 2016-04-12.

[2] Blis: BLAS-like library instantiation software framework. `https://github.com/flame/blis`. Accessed: 2016-04-12.

[3] Intel Math Kernel Library (Intel MKL). `https://software.intel.com/en-us/intel-mkl`. Accessed: 2016-04-12.

[4] OpenBLAS: An optimized BLAS library. `http://www.openblas.net`. Accessed: 2016-04-12.

[5] `powerrun`: A tool to measure energy. Accessed: 2016-04-12.

[6] Intel Manycore Platform Software Stack (Intel MPSS). `https://software.intel.com/en-us/articles/intel-manycore-platform-software-stack-mpss`, 2016.

[7] Message Passing Iterface. `https://www.mpi-forum.org`, 2016.

[8] NVIDIA CUDA Basic Linear Algebra Subroutines (cublas) library. `https://developer.nvidia.com/cublas`, 2016.

[9] Pedro Alonso, Javier Ibáñez, Jorge Sastre, Jesús Peinado, and Emilio Defez. Efficient and accurate algorithms for computing matrix trigonometric functions. *Journal of Computational and Applied Mathematics*, 309:325–332, January 2017.

[10] Olivier Beaumont, Vincent Boudet, Arnaud Legrand, Fabrice Rastello, and Yves Robert. Heterogeneous matrix-matrix multiplication, or partitioning a square into rectangles: NP-completeness and approximation algorithms. In *EuroMicro Workshop on Parallel and Distributed Computing (EuroMicro'2001)*, pages 298–305. IEEE Computer Society Press, 2001.

[11] S. M. Cox and P. C. Matthews. Exponential time differencing for stiff systems. *J. of Comput. Physics, Elsevier*, 176:430–455, 2002.

[12] Ashley DeFlumere, Alexey Lastovetsky, Brett Becker, et al. Partitioning for parallel matrix-matrix multiplication with heterogeneous processors: The optimal solution. In *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), IEEE 26th International*, pages 125–139. IEEE, 2012.

[13] Azzam Haidar, Jack Dongarra, Khairul Kabir, Mark Gates, Piotr Luszczek, Stanimire Tomov, and Yulu Jia. Hpc programming on intel many-integrated-core hardware with magma port to xeon phi. *Scientific Programming*, 23, 01-2015 2015.

[14] Nicholas J. Higham. *Functions of Matrices: Theory and Computation*. SIAM, Philadelphia, PA, USA, 2008.

[15] Marlis Hochbruck, Christian Lubich, and Hubert Selhofer. Exponential integrators for large systems of differential equations. *The SIAM Journal on Scientific Computing*, 19(5):1552–1574, September 1998.

[16] A. Kalinov and A. Lastovetsky. Heterogeneous distribution of computations solving linear algebra problems on networks of heterogeneous computers. *Journal of Parallel and Distributed Computing*, 61:520–535, 2001.

[17] Aly Khan Kassam and Lloyd N. Trefethen. Fourth-order time-stepping for stiff PDEs. *The SIAM J. on Scientific Comp.*, 26(4):1214–1233, 2005.

[18] NVIDIA. NVML Reference Manual. `https://developer.nvidia.com/nvidia-management-library-nvml`, 2013.

[19] J. Sastre, Javier J. Ibáñez, E. Defez, and Pedro A. Ruiz. Accurate matrix exponential computation to solve coupled differential. *Mathematical and Computer Modelling*, 54:1835–1840, 2011.

[20] J. Sastre, Javier J. Ibáñez, E. Defez, and Pedro A. Ruiz. Computing matrix functions arising in engineering models with orthogonal matrix polynomials. *Mathematical and Computer Modelling*, 57:1738–1743, 2013.

[21] J. Sastre, Javier J. Ibáñez, E. Defez, and Pedro A. Ruiz. Efficient scaling-squaring Taylor method for computing matrix exponential. *SIAM J. on Scientific Comput.*, 37(1):A439–455, 2015.

[22] Stanimire Tomov, Jack Dongarra, and Marc Baboulin. Towards dense linear algebra for hybrid GPU accelerated manycore systems. *Parallel Computing*, 36(5-6):232–240, June 2010.

[23] Field G. Van Zee and Robert A. van de Geijn. BLIS: A framework for rapidly instantiating BLAS functionality. *ACM Transactions on Mathematical Software*, 41(3):14:1–14:33, 2015.

[24] D. F. Williams, L. A. Hayden, and R. B. Marks. A complete multimode equivalent-circuit theory for electrical design. *SIAM J. on Scientific Comput.*, 102(4):405–423, 1997.

NESUS cost IC1305
Network for Sustainable Ultrascale Computing

An Array API for FDM

EVA BURROWS† HELMER ANDRÉ FRIIS* MAGNE HAVERAAEN†‡

† Bergen Language Design Laboratory
Department of Informatics, University of Bergen, Norway
https://bldl.ii.uib.no/

∗ IRIS
Stavanger, Norway
http://www.iris.no/

Abstract

As we move towards ultrascale computing, computer architecture is bound to see dramatic changes. Multiple nodes, with or without shared memory, multicore and accelerators (GPUs, FPGAs) will be the norm. For many problems, such as finite difference numerical simulations, the array used to represent a perfect match between the user level code and the hardware architecture's uniform memory access. Arrays, and to some extent multiarrays, are well supported by most programming languages. A standard compiler maps the array for uniform memory. Some programming models, such as partitioned global address space, allows mapping an array across distributed, yet for each partition, uniform memory. For ultrascale architectures, the simple mapping between user level (multi)array and distributed, non-uniform memory, will disappear. Here we propose an API for arrays, empowering the software developer to implement their own array-memory layout. Application code written towards the API will be independent of underlying architecture changes, thus easily ported to new architectures as they evolve.

Keywords Scientific Computing, Ultrascale Computing, Multiarray API, Array API for Finite Difference Methods

I. INTRODUCTION

The array has been a central concept for software development, especially in the high performance domain. For instance, multiarrays are key to explicit finite difference solvers for partial differential equations (PDE). Languages for programming in computational science have direct language support for arrays. Many also directly support multidimensional array manipulations (e.g. MATLAB, Fortran, F), or introduce libraries or packages to support this (e.g. Boost.MultiArray Library for C++, NumPy for Python). In the problem space, the array provides an abstraction for indexable data collections. In the hardware space, the array represents linear addressable memory.

Compilers exploit that traditionally computer memory has been uniformly accessible. A linear function is sufficient to map from an array (multi)index to a memory address, giving efficient access to a memory location. The move towards ultrascale computing is breaking this mapping. Currently we see architectures with collections of manycore processors, connected on fast networks, often with GPUs and other accelerators connected to each processor. The combined memory of such an architecture is no longer linearly addressable, but possibly hierarchical: indexed by processor in the network, then split into core local memory, accelerator local memory, shared core memory and shared accelerator/processor memory. The access time for a memory location varies, depending on where the memory is located, which core/accelerator is accessing the location, and the local, global and collective data access patterns (cache lines, network contention, etc.). On future ultrascale architectures we should expect the data access functions and memory access costs to be more complex.

Programming models to deal with the situation are slow to emerge. The two dominant approaches are explicit processes with message passing (e.g., MPI [15]), and variations of partitioned global address space (PGAS). Both of these models currently assume that memory is distributed across nodes (or cores), but lack support for hierarchical memory and accelerators. The message passing approach is a dramatic change from sequential programming, since a computation here is an ensemble of explicitly communicating programs. Verifying the correctness of concurrent processes is hard. Using a pragmatic *single program multiple data* (SPMD) approach the code transformation to message passing form becomes manageable. The PGAS model is closer to standard programming and is thus easier to reason about. A PGAS compiler may use message passing processes as the target code [8]. Accelerators are mostly supported by specialised models (e.g., Cuda). Hybrid models, e.g., mixing MPI and multicore programming or MPI and accelerators, are being used for hybrid architectures. Porting an application from one programming model to another requires considerable changes to the code. This causes severe challenges to the portability of application between current architectures, and thus may be a severe hindrance for the uptake of efficient future ultrascale architectures.

The emerging gap between problem space arrays and computer memory addressing should be bridged by tools such as the compiler and its support libraries. Keeping tools up to date is a continuous effort as new architectures are being continuously introduced. Unfortunately, tool and compiler vendors are not catching up to the pace of change. For instance, Fortran's take on the PGAS model

was standardised in 2008 [13], almost a decade ago. Yet few Fortran compilers in 2016 support the coarray feature.

This paper promotes the idea that for the hardware space we need a standardised, linear array API encapsulating the heterogeneous memory structure. This can be considered a variation of the PGAS model for distributed memory, and is a refinement of our earlier suggestion [9]. Such an API will empower the software developer to provide their own mapping of the linear indices onto the hierarchical memory structure, in case a relevant one does not already exist. This liberates the developer from relying on compilers and other tools that may never materialise. It also liberates the hardware manufacturer from providing a full fledged tool chain to support every new architecture. A good implementation of the relatively simple API for the new architecture is sufficient. For the problem space we need various adapters mapping, e.g., multiarray or tree structures, to the linear array API. Again, the software developer is empowered to provide relevant mappings in case none exist. Such mappings are obviously reusable across problems with similar needs. The mapping from linear array to hardware structure will be reusable for every application running on that hardware. The mapping from problem space to linear array will be reusable for all applications in the problem space, across all applicable hardware architectures. In order to make such a software architecture to become an efficient tool, it is (1) important to tune the APIs carefully for generic reuse, and (2) to develop applications focussed on using collective operations on the user space abstractions. The former requires careful domain engineering coupled with domain experience. The latter require a change with software developers, who normally are drilled to work with individual elements of arrays and other structures. The proposed approach requires a focus on collective operations on entire data sets. The APIs need to be stable across old and new architectures ensuing portability of application code. The basic linear algebra subprograms (BLAS [12]) is an example of what can be achieved by an API approach. With many of the similar approaches (PGAS, Global Arrays, Coarrays), new notation creeps in to handle new hardware. This causes application portability costs. New notation causes portability issues with existing applications.

The contribution of the paper is to show that an API approach is viable by presenting an API suitable for explicit finite difference solvers. We use collective multiarray operations to develop a solver for Burgers' equation. The multiarray API is mapped to a linear array API. The linear array API is mapped to a plain CPU with linear memory, a GPU local memory, and a GPU local memory with explicit administration of data allocation and deallocation. Since these three distinct hardware mappings all provide the same linear array API, no change is needed in neither the multiarray mapping nor the solver itself. We present the APIs as *concepts* in the sense of [19], i.e., with declarations of types and operations, and axioms describing their properties. Such concepts provide precise description of intended semantics. They work very well with generic implementations (reusable code), and provide verifiable/testable requirements [1].

The paper is organised as follows. In the next section, we introduce the mathematics of our running example, the Burgers' equation, and show how a PDE normally is massaged for implementing a solver. In Section III, we propose our multiarray API, followed by a presentation of the linear array API in Section IV. These two APIs are tied together in Section V. Then we present some experimental results of using our approach to target the Burgers' solver for CPU and GPU implementations. Finally, Section VII discusses some related work before we conclude in Section VIII.

II. Finite difference numerical software

When writing numerical software, the HPC engineer typically starts from a partial differential equation which is then manipulated into a form suitable for programming. We will use Burgers' equation [3] as an illustration. Burgers' equation is an important nonlinear prototype equation, used for instance in the mathematical modelling of gas dynamics and traffic flow. It is similar to the incompressible Navier-Stokes equation, without the pressure term and external forces like, e.g., gravity. In coordinate free form it reads

$$\frac{\partial \vec{u}}{\partial t} + \vec{u} \cdot \nabla \vec{u} = \nu \nabla^2 \vec{u}, \tag{1}$$

where $\vec{u}$ denotes velocity, t is time, and ν is a viscosity coefficient. In one spatial dimension, putting $\vec{u} = (u)$, we get

$$\frac{\partial (u)}{\partial t} + (u) \cdot \nabla (u) = \nu \nabla^2 (u), \tag{2}$$

Choosing Cartesian coordinates, we can elaborate the gradient, the laplacian and the dot product, giving

$$\frac{\partial u}{\partial t} + u \frac{\partial u}{\partial x} = \nu \frac{\partial^2 u}{\partial x^2}. \tag{3}$$

We implement the initial value problem $u(x,0) = u_0(x)$, for periodic boundary conditions on an interval of length L, $u(x+L,t) = u(x,t)$.

To solve the problem numerically, the standard approach is to discretise the domain. For this we introduce the grid values $u_i^n = u(i\frac{L}{N}, t_n)$ for $i = 0, \ldots, N-1$, and $t_n = n\Delta t$, where Δt is the time step size. In the finite difference method (FDM), we compute a partial derivative by a weighted sum of neighbouring grid points. The weights are formed from two components: (i) a list of factors called the stencil, and (ii) a factor computed from the data resolution (the number of gridpoints). The stencil is carefully decided by a numerical expert. The choice is based on the kind of problem being solved, the initial value being used, accuracy versus speed, etc. For example, in this paper we use the numerical stencils $(-0.5, 0, 0.5)$ and $(1, -2, 1)$ for $\frac{\partial}{\partial x}$ and $\frac{\partial^2}{\partial x^2}$, with factors $\Delta x = \frac{L}{N}$ and $(\Delta x)^2 =$

$\frac{L^2}{N^2}$, respectively. The standard explicit finite difference numerical approximation for equation (3) then becomes,

$$\begin{aligned} u_i^{n+1} &= u_i^n - \frac{\Delta t}{2\Delta x} u_i^n (u_{i+1}^n - u_{i-1}^n) \\ &+ \frac{\nu \Delta t}{(\Delta x)^2} (u_{i+1}^n + u_{i-1}^n - 2u_i^n). \end{aligned} \tag{4}$$

The above approximation is accurate to $\mathcal{O}((\Delta x)^2, \Delta t)$.

The formulation in equation (4) is easy to write up as traditional code: The data for u^{n+1} and u^n is stored in two arrays, one for each, and a single loop, for all element indices 0 through $N-1$, computes u^{n+1} from u^n.

Now consider equation (1) in higher spatial dimensions. Using a Cartesian coordinate system, we can write Burgers' equation in three spatial dimensions (3D) as

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + w\frac{\partial u}{\partial z} = \nu\frac{\partial^2 u}{\partial x^2} + \nu\frac{\partial^2 u}{\partial y^2} + \nu\frac{\partial^2 u}{\partial z^2} \tag{5}$$

$$\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + w\frac{\partial v}{\partial z} = \nu\frac{\partial^2 v}{\partial x^2} + \nu\frac{\partial^2 v}{\partial y^2} + \nu\frac{\partial^2 v}{\partial z^2} \tag{6}$$

$$\frac{\partial w}{\partial t} + u\frac{\partial w}{\partial x} + v\frac{\partial w}{\partial y} + w\frac{\partial w}{\partial z} = \nu\frac{\partial^2 w}{\partial x^2} + \nu\frac{\partial^2 w}{\partial y^2} + \nu\frac{\partial^2 w}{\partial z^2} \tag{7}$$

where we have put $\vec{u} = (u, v, w)$. Here we also choose periodic boundary conditions in a 3D domain of length L_x, L_y and L_z in the x, y, and z coordinate directions, respectively. Introducing appropriate grid values, we denote for the first velocity component u that $u^n_{x,i,j,k} = u_x(i\frac{L_x}{N}, j\frac{L_y}{N}, k\frac{L_z}{N}, t_n)$ for $i = 0, \ldots, N-1$, $j = 0, \ldots, M-1$ and $k = 0, \ldots, P-1$. Using analogous approximations as in equation (4), equation (5) can then be discretised as follows.

$$\begin{aligned} u_{i,j,k}^{n+1} &= u_{i,j,k}^n - \frac{\Delta t}{2\Delta x} u_{i,j,k}^n (u_{i+1,j,k}^n - u_{i-1,j,k}^n) \\ &+ \frac{\nu \Delta t}{(\Delta x)^2} (u_{i+1,j,k}^n + u_{i-1,j,k}^n - 2u_{i,j,k}^n) \\ &- \frac{\Delta t}{2\Delta y} v_{i,j,k}^n (u_{i,j+1,k}^n - u_{i,j-1,k}^n) \\ &+ \frac{\nu \Delta t}{(\Delta y)^2} (u_{i,j+1,k}^n + u_{i,j-1,k}^n - 2u_{i,j,k}^n) \\ &- \frac{\Delta t}{2\Delta z} w_{i,j,k}^n (u_{i,j,k+1}^n - u_{i,j,k-1}^n) \\ &+ \frac{\nu \Delta t}{(\Delta z)^2} (u_{i,j,k+1}^n + u_{i,j,k-1}^n - 2u_{i,j,k}^n), \end{aligned} \tag{8}$$

where $\Delta x = \frac{L_x}{N}$, $\Delta y = \frac{L_y}{M}$ and $\Delta z = \frac{L_z}{P}$. Clearly, the two remaining equations (6-7) can be discretised in a similar manner.

Writing traditional style code for the 3D version is more involved than for the 1D case. The data for each of u, v, w is a multiarray with three indices (i, j, k). We will need two sets of multiarrays, one for timestep n and one for the next timestep $n+1$. A triply nested loop is then used to compute timestep $n+1$ from n.

```
for (int i = 0; i < N; i++) {
  for (int j = 0; j < M; j++) {
   for (int k = 0; k < P; k++) {
     up(i,j,k) = un(i,j,k)
       - deltat * un(i,j,k)
         * ( un(i+1,j,k) - un(i-1,j,k) )
         / (2 * deltax)
       + ... ;
     vp(i,j,k) = ...;
     wp(i,j,k) = ...;
   }
  }
}
```

Here the suffix `p` is used for variables at timestep $n+1$, the suffix `n` for variables at timestep n.

In each elaboration step above, abstractions from the problem domain are unfolded and removed from the exposition. In the end, it is difficult to directly relate the nested loops to the original problem. First, the coordinate-free operators ∇, $\cdot$ and ∇^2 were instantiated with the number of dimensions and Cartesian coordinate system, yielding equation (3) and equations (5-7), for 1 and 3 dimensions, respectively. Then the spatial representation as a finite difference method was chosen, and the continuous operators $\frac{\partial}{dx}, \frac{\partial^2}{dx^2}, \ldots$ were instantiated with the corresponding stencils. Thus the forms equation (3) and equations (5-7) bear little resemblance to each other, nor to the problem formulation equation (1). That the resulting code in fact is related to the original problem is non-trivial to validate, and a separate documentation trail needs to be maintained in order to relate the instantiations to the original problem.

Above we have sketched the sequential implementations, almost taking the code directly from the elaborated version of the equations.

III. A Burgers solver and multiarray API

As initially motivated, we want to reformulate the solver using *collective* operations, i.e., operations that work on the entire array rather than looping through the individual elements. The abstraction level in equation (3) consists of the continuous operators: partial differentiation, addition, multiplication, etc. Considering the FDM discretisation, addition, multiplication, etc, are simple pointwise operations on the array, while partial differentiation relies on neighbouring data.

First we assume an indexing function which returns the element given by the multiindex (i, j, k).

```
1 function get (a:MA, i:int, j:int, k:int) : E;
```

The type `MA` is the multiarray, `int` is an integer type used for indexing data, and `E` is the element type (floating point number).

Next we investigate *mapped* elemental operations, like $+, *, -$. Mapped functions can be defined as the following concept.

```
/** Elemental addition, multiplication and subtraction. */
  function _+_ ( a:E, b:E ) : E;
  function _*_ ( a:E, b:E ) : E;
  function _-_ ( a:E, b:E ) : E;
  function -_ ( a:E ) : E;
/** Mapped addition, multiplication and subtraction. */
  function _+_ ( a:MA, b:MA ) : MA;
  function _*_ ( a:MA, b:MA ) : MA;
  function _-_ ( a:MA, b:MA ) : MA;
  function -_ ( a:MA ) : MA;

/** Relating the mapped and elemental operations. */
  axiom binaryMap ( a:MA, b:MA, i,j,k:int ) {
    assert get( a+b, i,j,k )
      == get(a,i,j,k) + get(b,i,j,k);
    assert get( a*b, i,j,k )
      == get(a,i,j,k) * get(b,i,j,k);
    assert get( a-b, i,j,k )
      == get(a,i,j,k) - get(b,i,j,k);
  }
  axiom unaryMap ( a:MA, i,j,k:int ) {
    assert get( -a, i,j,k ) == - get(a,i,j,k);
  }
```

The assertions in the axiom must hold for all combinations of input data, the parameters, to the axiom. An axiom is like a procedure, whose intended effect is to validate the assertions on the input data. This can be used to test the correctness of the code, though testing on floating point data seldom works as intended.

To provide the partial difference operators we will need a *shift* function on the multiarrays.

```
  function shift (a:MA, dir:int, d:int) : MA;
```

Here the parameter `dir` instructs which direction we will be shifting (1 for x direction or index i, 2 for y direction or index j, 3 for z direction or index k), and `d` gives the shift distance (± 1 for one step as needed in the example).

```
  axiom multiarrayShiftAxiom
    ( a:MA, d:int, i,j,k:int ) {
    assert get( shift(a,1,d), i,j,k)
      == get(a, (Lx+i+d)%Lx,j,k);
    assert get( shift(a,2,d), i,j,k)
      == get(a, i,(Ly+j+d)%Ly,k);
    assert get( shift(a,3,d), i,j,k)
      == get(a, i,j,(Lz+k+d)%Lz);
  };
```

Using the modulus operator % for index manipulation above, we define a circular shift, as needed for circular boundary conditions.

With this sketch of the multiarray API, the indexing, map and shift operations, in place, we can for any stencil define the partial derivatives as collective operations on a multiarray. The function `partial1` implements a 1st order partial derivative using a $(-0.5, 0, 0.5)$ stencil, and the function `partial2` implements a 2nd order partial derivative using a $(1, -2, 1)$ stencil.

```
  function partial1 ( a:MA, dir:int ) : MA {
    return (shift(a,dir,1) - shift(a,dir,-1))
      / (2 * deltax );
  };
  function partial2 ( a:MA, dir:int ) : MA {
    return
      (shift(a,dir,-1) - 2*a + shift(a,dir,1))
      / (deltax * deltax );
  };
```

The `dir` argument encodes the direction, `dir==1` for x-direction $\frac{\partial a}{\partial x}$ and $\frac{\partial^2 a}{(\partial x)^2}$, `dir==2` for y-direction $\frac{\partial a}{\partial y}$ and $\frac{\partial^2 a}{(\partial y)^2}$, and `dir==3` for z-direction $\frac{\partial a}{\partial z}$ and $\frac{\partial^2 a}{(\partial z)^2}$.

The solver step for equations (5-7) can now be coded using these operations.

```
  up = un
       + nu*deltat*
       ( partial2(un,1)
       + partial2(un,2)
       + partial2(un,3))
     - deltat*un*partial(un,1)
     - deltat*vn*partial(un,2)
     - deltat*wn*partial(un,3);
  vp = ...;
  wp = ...;
```

Notice how we easily may change the stencil for this computation: it is encapsulated in the partial derivative functions, so replacing these with functions for another stencil is all it takes. The stencil is no longer embedded all over in the formulation of the solver, as it was in equation (4)

IV. Linear array API for abstracting hardware

Instead of implementing the multiarray API directly in the hardware, we propose a linear API for the mapping onto the hardware. The linear API is slightly more convoluted than the multiarray API, but is often more straight forward to implement on a target hardware architecture.

For this exposition, the linear array needs the following primitive operation to access an element based on an integer index.

```
  /** Get the element at the index position i. */
  function get ( a:A, i:int ) : E;
```

The type `A` is an array of elements and `E` is the element type, typically floating point numbers.

We have a similar mapping of elemental functions for the linear

```
/** Shifts the grp−sized groups of data d positions circularly to the left within each seg−sized segment. */
function shiftSegmentGroups ( a:A, seg:int, grp:int, d:int) : A
  guard seg % grp == 0 && getSize(a) % seg == 0 && abs(d) <= seg;

axiom shiftSegmentGroupsDefinitionAxiom ( a:A, seg:int, grp:int, d:int, j:int ) {
  var size = getSize(a);
  assert size % seg == 0 && seg % grp == 0 && abs(d) <= seg && 0 <= j && j < size;
    // local index within a segment
    var si = j % seg;
    //normalize actual shift value to perform within a segment
    var sh = (grp * (seg + d)) % seg;
    // new index within segment after the local shift
    var ni = ( seg+si−sh ) % seg;
    // obtain the global position of ni within the whole array
    var ind = idiv(j, seg)*seg + ni;
  assert get(shiftSegmentGroups(a,seg,grp,d),ind) == get(a,j);
};
```

Figure 1: Definition of `shiftSegmentGroups` *operation for the linear array.*

array as we did for the multiarray.

```
/** Elemental addition, multiplication and subtraction. */
  function _+_ ( a:E, b:E ) : E;
  function _*_ ( a:E, b:E ) : E;
  function _−_ ( a:E, b:E ) : E;
  function −_ ( a:E ) : E;
/** Mapped addition, multiplication and subtraction. */
  function _+_ ( a:A, b:A ) : A;
  function _*_ ( a:A, b:A ) : A;
  function _−_ ( a:A, b:A ) : A;
  function −_ ( a:A ) : A;

/** Relating the mapped and elemental operations. */
axiom binaryMap ( a:A, b:A, i:int ) {
  assert get( a+b, i ) == get(a,i) + get(b,i);
  assert get( a*b, i ) == get(a,i) * get(b,i);
  assert get( a−b, i ) == get(a,i) − get(b,i);
}
axiom unaryMap ( a:A, i:int ) {
  assert get( −a, i ) == − get(a,i);
}
```

We also need to rearrange (permute) the data of the array in various ways for different purposes. Here we provide a fairly general shift operation, see figure 1. It shifts groups of data within segments of the array. The group size must divide the segment size, the segment size must divide the actual array size, and the shift distance must at most be equal to the segment size. (this is written in the guard phrase, which captures the precondition for the shift function). The axiom similarly asserts the relevance of its input data, then nails down the behaviour of this shift function.

These are the linear array operations we need to define and implement for explicit finite difference solvers for PDEs. For other application domains the linear array API may need to contain further operations. Typically a linear API will also provide collective operations like the prefix scan and fold/reduce. These are not covered here.

In section VI.1 we sketch some hardware oriented implementations of this API.

V. Multiarray library

In section III we defined a multiarray API, and in the previous section we defined a linear array API to mask hardware. Here we explain how to provide a multiarray library on top of the linear array API.

First we define how to retrieve an element from the linear array using a multiindex. This is a bijective, simple linear mapping from a multilinear array with size L_x by L_y by L_z to a linear array of size $L_xL_yL_z$.

```
function get( a:MA, i,j,k:int ) : E {
  return get(a, i*Ly*Lz + j*Lz + k );
}
```

The map functions are straight forward to reuse from the linear array. The maps are pointwise, and thus irrespective of indexing, the result will be at the correct position.

The multiarray shift similarly needs to match both the multiar-

```
assert get( shift(a,1,d), i,j,k) == get(a, (Lx+i+d)%Lx,j,k);
assert get( shiftSegmentGroups(a,Lx*Ly*Lz,Ly*Lz,d), i*Ly*Lz + j*Lz + k )
 == get(a, ((Lx+i+d)%Lx)*Ly*Lz + j*Lz + k );
assert get( shiftSegmentGroups(a,Lx*Ly*Lz,Ly*Lz,d), i*Ly*Lz + j*Lz + k )
 == get( shiftSegmentGroups(a,Lx*Ly*Lz,Ly*Lz,d),
        ((Lx+i+d)%Lx)*Ly*Lz + j*Lz + k + (Lx*Ly*Lz - d*Ly*Lz) % (Lx*Ly*Lz) );
assert get( shiftSegmentGroups(a,Lx*Ly*Lz,Ly*Lz,d), i*Ly*Lz + j*Lz + k )
 == get( shiftSegmentGroups(a,Lx*Ly*Lz,Ly*Lz,d),
        ((Lx+i+d)%Lx)*Ly*Lz + j*Lz + k + ((Lx-d) % Lx)*Ly*Lz );
assert get( shiftSegmentGroups(a,Lx*Ly*Lz,Ly*Lz,d), i*Ly*Lz + j*Lz + k )
 == get( shiftSegmentGroups(a,Lx*Ly*Lz,Ly*Lz,d),
        ((Lx+i+d+ Lx-d)%Lx)*Ly*Lz + j*Lz + k );
```

Figure 2: Proof for the correctness of the multiarray shift.

ray's indexing structure and the linear array's shift behaviour, see figure 1. The following defines an appropriate function.

```
function shift ( a:MA, dir:int, d:int ) : MA {
  var seg =
    if dir == 1 then Lx * Ly * Lz
    else if dir == 2 then Ly * Lz
    else /* dir == 3 */       Lz;
    end end;
  var grp = seg /
    if dir == 1 then Ly * Lz
    else if dir == 2 then Lz
    else /* dir == 3 */    1;
    end end;
  return shiftSegmentGroups( a, seg, grp, d );
}
```

We sketch a proof of correctness for the x direction in figure 2. The first assert is from `multiarrayShiftAxiom`. In the next assert we have inserted the multiarray `get` and `shift` algorithms above. The third assert uses `shiftSegmentGroupsDefinitionAxiom` to replace the right hand side with an expression involving `shiftSegmentGroups`. The remaining lines simplify the right hand side until it is clear it matches the left hand side expression. The proof for the y and z directions follow a similar pattern.

The necessary abstractions to code FDM solvers at the continuous level requires a multiarray shift and mapped $+,-,*,/$ on the multiarray. This now boils down to providing `shiftSegmentGroups` and the mapped functions $+,-,*,/$ on an ordinary linear array. This API is quite simple with a few recurring patterns: (i) the map operations, representing a local, per element computation, (ii) the shift operation, representing data reorganisation and communication. Compared to rewriting the entire application code for each architecture, implementing this limited set of functions will be rather trivial—empowering the user to implement hardware specific array libraries if the hardware vendor does not provide it.

VI. Runtime experiments

We have done several runtime experiments with the developed 3D Burgers' solver. It uses the form from equations (5-7). The runtime experiments target the following two issues.

- Does the suggested approach support easy porting of code between architectures?
  To answer this question we provide implementations of the proposed array API for several architectures, and validate that the application using the API, without source code modification, will run on the relevant hardware.
- Does the application code scale as expected on the various architectures?
  To answer this question we run the application on varying data sizes. For our application example, a 3D FDM Burgers solver, we should see linear scaling with respect to data set size, modulo any effects of caching and virtual memory.

VI.1 Linear array implementations

Currently we have targeted two hardware architectures for the linear array abstraction.

CPU C++ A plain sequential implementation for a single CPU and uniform memory. This uses C++ arrays for the linear array API. The mapped operations are each wrapping a loop sequentially performing the lifted operation element by element. The `shiftSegmentGroups` operation makes a temporary copy of the current data, then overwrites the argument array with the shifted data.

Cuda An Nvidia GPU version implemented in Cuda. The array is represented as a linear structure in device (GPU) memory, avoiding transfer of data between CPU and GPU memory during the computation. The 5 lifted functions, $+, -, *, /$, are implemented as device loops in Cuda, then called to be executed in multithreading GPU kernel mode. This implies no internal synchronisation, but each mapped operation must be complete before the next operation is started. `shiftSegmentGroups` can be implemented by either synchronising the shifted data between the GPU thread blocks using multiple kernel invocations, or by obtaining the result of the shift operation in a fresh array across the GPU device, eliminating the need for explicit synchronisation within the function. We have chosen the latter, keeping the shift function as a single kernel call. This causes a larger memory use than strictly needed, but is not detrimental to efficiency if the application still fits into GPU memory. If this is not the case, other approaches may be beneficial.

During a computation temporary data is continuously created as subexpressions are evaluated, and subsequently released when the result of the expression is assigned to a variable. On the GPU allocating and deallocating data takes a significant amount of time, so yet another version of the linear array library was created.

CudaBuffer An Nvidia GPU version implemented in Cuda as above, but where a buffer large enough to store all temporary device data is created at the start. This is then managed explicitly under the hood by the linear array implementing code, possibly giving more efficient reuse of GPU memory when temporary variables are created and deleted.

This provides an affirmative answer to our first research question: we have achieved portability at the application level by a problem specific API.

These implementations do not attempt any clever optimisations. For instance, map fusion (loop merging) could give significant speedups. This entails rearranging the expressions of the PDE solver, such that local data is only iterated once on the cores, not once per operation. For instance, mapping $A = f(A,B,C)$ for an elemental function $f(a,b,c) = a*b+c$ typically is faster than mapping each of the operations $+, *$ in $A = A*B+C$. Such rewriting should be tool supported, otherwise the clarity, and possibly the portability, of the code will be sacrificed for efficiency.

Even for languages that natively support lifted operations, the efficiency of mapped operations is an important aspect. For instance, in early Fortran 90 compilers, executing $A = A + B$ was much slower for the multiarray version than the corresponding nested loop version.

VI.2 Runtime results

We configured the software for varying data sizes, each chosen data size doubling the memory requirement for the program. The data, 10 waves of sine functions in the z direction, was generated in the appropriate resolution. Since we are working with 3D data, we double the size of the data set (the size of the linear array), whenever the number of elements in each direction in increased by a factor of $\sqrt[3]{2} \approx 1.26$. We used data set sizes 78MB (50^3 elements), 156MB (60^3), 307MB (79^3), 624MB (100^3), 1 248MB (126^3), 2 508MB (159^3), 4 992MB (200^3), and 9 985MB (252^3). Each problem size was executed for $1, 10, 100, 1000$ timesteps in each of the three versions (CPU, GPU, GPU buffered), yielding a total of $8*4*3 = 96$ runs. The applications were run on the department's compute server `lyng`. It has Intel Xeon CPU E5-2699 v3 at 2.30GHz cores and Nvidia Tesla K40m with 2880 CUDA Cores at 745 MHz. The runtime is wall clock time. The clock was started immediately before the time iteration of the solver, and stopped immediately after the time iteration. This eliminates unpredictable overhead in starting especially the GPU (Cuda, CudaBuffer) applications. The overhead includes a Cuda just-in-time compilation of GPU code and initialisation of device data, which together may take several seconds even for small datasets. The CPU does not exhibit similar disparity between total execution times and the solver's timestepping loop times.

The CPU timings are tabulated below. The row captions show the linear data sizes, the column captions show the number of iteration timesteps, and the table data is the software's runtime in seconds.

Cpu C++	1	10	100	1000
50	0.268	2.900	26.198	264.116
63	0.535	5.794	53.620	530.304
79	1.058	11.516	103.989	1032.465
100	2.176	21.374	220.488	2181.002
126	4.671	51.581	482.747	4642.034
159	10.156	128.887	1176.986	10124.613
200	28.517	958.223	6532.079	32186.134
252	776.532	2387.521	13636.333	120525.580

These runs were concurrent with other loads on the computer, leaving about 10GB of free memory for our application. The results scale well for the smaller tests: it roughly doubles with data set size for the 4 smaller data sizes, and scales linearly with the number of timesteps for the 6 smaller data sizes. The two largest data set sizes behave somewhat erratic, see figure 3, possibly due to swap behaviour when memory ran low.

The Cuda timings are tabulated below. The row captions show the linear data sizes, the column captions show the number of iteration timesteps, and the table data is the software's runtime in seconds.

Cuda	1	10	100	1000
50	0.290	3.151	29.479	286.206
63	0.310	3.408	31.190	311.993
79	0.370	3.939	36.294	363.476
100	0.487	4.848	49.076	492.145
126	0.747	7.557	75.908	749.141
159	1.325	14.401	132.114	1327.804
200	3.227	31.472	310.946	3098.445
252	6.448	62.974	638.149	6358.311

These runs had exclusive access to the GPU. We see the runtimes grow linearly with the number of timesteps for all data set sizes, see figure 4. For the 4 larger data set sizes the runtime roughly doubles when the data set size doubles. However, the 4 smaller data set sizes do not double in runtime as the data set sizes double. This indicates overheads on the GPU for these smaller data set sizes, possibly related to allocation/deallocation of temporary variables.

The Cuda buffered timings are tabulated below. The row captions show the linear data sizes, the column captions show the number of iteration timesteps, and the table data is the software's runtime in seconds.

CudaBuffer	1	10	100	1000
50	0.052	0.577	5.057	50.619
63	0.095	1.045	9.552	94.447
79	0.162	1.777	16.178	162.277
100	0.290	2.934	29.526	292.908
126	0.563	5.694	57.514	567.513
159	1.133	12.439	114.399	1134.132
200	2.346	26.430	233.351	2477.200
252	5.108	49.518	500.261	5032.021

These runs show very good scaling in both dimensions, see figure 5. For each data set size, the runtime scales linearly with the number of timesteps, and the runtime roughly doubles when data set size is doubled.

Comparing the two Cuda versions against each other, we notice that for the smaller data set sizes, the unbuffered versions are 5-6 times slower than the buffered version. For the larger data set sizes, this difference is down to 25%. This is still a noticeable speedup for the buffered over the non-buffered GPU version.

Comparing the Cuda version with the CPU version, we see that the CPU does well for the smallest data set size. As the data set sizes grow, the CPU slows significantly down compared to the GPU executions, at times becoming a factor of 20 slower. Comparing the CPU to the buffered Cuda version, we see a slowdown factor close to 30 for some instances. In the figures 3-5, both Cuda versions (the two lower diagrams), are on the same scale, while the CPU version (the upper diagram) has an extra line for 100 000 seconds. This just indicates the common observation that in the PDE domain significant speedups can be achieved using parallel GPU computing over standard single-threaded CPU computing.

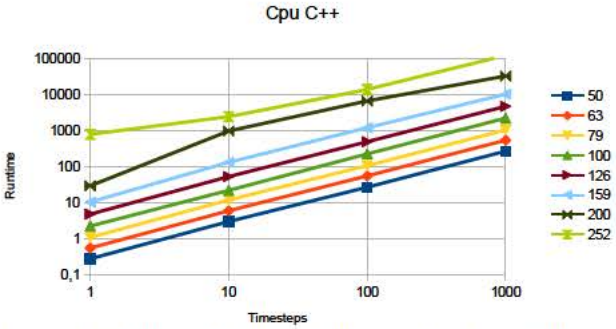


Figure 3: Runtimes for Burgers 3D solver on main CPU using collective operations implemented in plain C++.

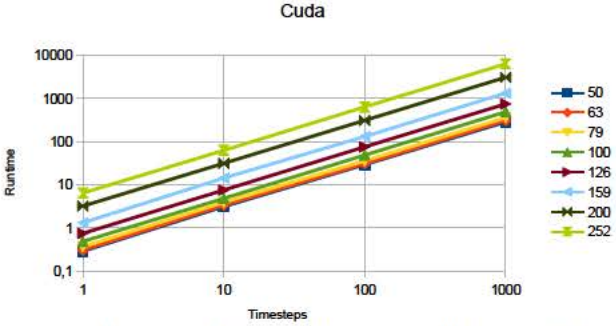


Figure 4: Runtimes for Burgers 3D solver on GPU using collective operations implemented in Cuda.

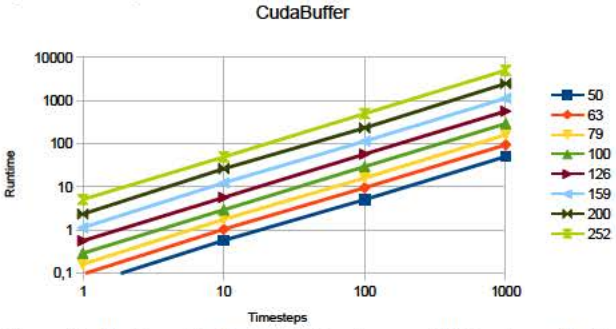


Figure 5: Runtimes for Burgers 3D solver on GPU using collective operations with explicit buffer management implemented in Cuda.

We have previously promoted similar ideas in [9] where we demonstrated an implementation using MPI of a multiarray abstraction. Again the data showed similar scaling.

The experiments confirm that the proposed abstraction layered approach to utilising heterogeneous architectures works. At least on the chosen kind of example (explicit finite difference solver), the measured runtimes roughly scale as expected for each type of backend.

VII. Related Work and Discussion

Abstracting away parts of numerical computations has long been recognised as the path to increase numerical software development productivity and flexibility. As a result, either language extensions or reusable software libraries have been proposed to raise the abstraction level. In the past decade, high-level parallelisation aspects of numerical code has also emerged as an active research field and most proprietary and open source software used in computational science tackle this issue to some extent. We will briefly summarise three approaches: directives, language extensions and libraries.

Common directives-based languages are OpenMP [4] and OpenACC [17]. Directives provide meta-information about the code, enabling the compiler to parallelise and distribute it across cores on a parallel architecture. These are fully dependent on compiler support, and the user cannot adapt the tools to deal with new hardware architectures. Directives are not compatible with our proposed API approach.

Fortran 2008 [13] is a programming language standard with explicit support for parallelism in the form of coarrays. Using coarrays require changing the sequential code, a change that may influence the structure of the entire program [10]. However, a coarray adapted program may also execute on sequential architectures, in principle making the code portable. The coarray feature has been designed for the PGAS model, but provides only rudimentary support for the feature. Work is being done to provide a reusable, open source support for coarrays [8]. The initiative builds on the MPI library, see discussion below. Some authors have proposed extensions to Fortran to handle accellerators [16].

There is a wide range of libraries for supporting parallel and distributed programming. We mention a few here.

C++ [2] has no native support for parallelism, but there are many libraries supporting multiarrays and parallelism (boost.org, Blitz++[20, 7]). It is easy to implement our proposed API structure as a C++ library.

Cuda [5] is an extension to C, C++ and Fortran providing facilities for using Nvidia GPUs. It makes GPGPU programming straightforward, but the code is not portable to other parallel architectures or competing GPU vendors. We use Cuda in our GPU implementation of Burgers' equation.

OpenCL [14] is an extension to C providing interfaces to many different hardware backends, e.g., GPUs and FPGAs. The parallel programming features are low level, but should be well suited for writing the lower level parallel libraries in our proposed API structure.

MPI [15] is a widespread library for explicit communication of data. The library is available for most programming languages, and is adapted to almost all current parallel architectures. Using the library directly is intrusive and forces significant rewrite of source code. It is used as the standard low level communication library, and can easily be used for implementing our low level linear array abstraction.

Diffpack [11] is a proprietary C++ library based on object-oriented numerical code widely used in CSE applications and simulations. Diffpack has become successful due to the powerful abstractions imposed on numerial code offering productivity and efficient code. This provides a domain oriented API as proposed in our approach, but Diffpack does not empower their user to provide their own architecture mappings as we suggest.

Mathworks has an extensive parallel Computing Toolbox for Matlab [6]. These are based on the multiarray abstraction, and provides backends for many parallel architectures. It is possible to implement our proposed API structure in Matlab, but the user is dependent on the vendor for adaption to new architectures.

VIII. Conclusion

Software structure is very important for the versatility of software, specifically the ability of re-targeting a numerical solver for new HPC architectures. We argue that carefully creating a system of APIs for computational software is a way of organising software achieving this. With object-oriented numerics programming styles becoming embraced also in HPC [18], abstraction oriented approaches are now part of the HPC toolbox.

A well designed library API will embody the application domain's concepts, in such a way that a clean and natural separation occurs between application code and, in our case, the underlying hardware architectures. A message passing library, e.g., MPI, does not have such a property wrt PDE solvers, while an array based library does.

We have proposed using simple array based APIs as a means of abstracting over hardware and providing the applications with a stable abstraction layer. The approach empowers the user to provide their own mappings to heterogenous architectures. Empowering the user to easily re-target a code for new architectures is important to prepare for ultrascale computing.

Compiler vendors seem to a limited extent be able to support this fast changing landscape, hence leaving compiler dependent software support in the dark. Many language extensions for parallelism also fail in portability, requiring more or less intrusive rewrites of code when porting between architectures.

The technical results show that our approach is feasible and delivers on two important issues: (I) the approach makes applications portable across varying hardware architectures without modifications in the application source code, and (II) the approach achieves the expected runtime scalability to be useful for HPC. We have thus converted a portability problem into a much simpler library implementation problem.

Future work includes building further benchmarks for more complex hardware architectures, and comparing our results to those

achieved by the more labour intensive standard approaches. Further we want to expand the ideas to other problem domains.

Acknowledgment

This research has in part been financed by The Research Council of Norway through the project Design of a Mouldable Programming Language (DMPL), and has received support from EU under the COST Program Action IC1305, Network for Sustainable Ultrascale Computing (NESUS).

References

[1] Anya Helene Bagge, Valentin David, and Magne Haveraaen. Testing with axioms in C++ 2011. *Journal of Object Technology*, 10:10:1–32, 2011.

[2] Pete Becker et al. ISO/IEC 14882:2011: Programming languages – C++ (final draft international standard). Technical Report N3290, JTC1/SC22/WG21 – The C++ Standards Committee, April 2011.

[3] J.M. Burgers. A mathematical model illustrating the theory of turbulence. In Richard Von Mises and Theodore Von Kármán, editors, *Advances in Applied Mechanics*, volume 1, pages 171 – 199. Elsevier, 1948.

[4] Barbara Chapman, Gabriele Jost, and Ruud van der Pas. *Using OpenMP: Portable Shared Memory Parallel Programming (Scientific and Engineering Computation)*. The MIT Press, 2007.

[5] Shane Cook. *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2013.

[6] Timothy A. Davis. *MATLAB Primer, Eighth Edition*. CRC Press, Inc., Boca Raton, FL, USA, 8th edition, 2010.

[7] Denis Demidov, Karsten Ahnert, Karl Rupp, and Peter Gottschling. Programming CUDA and opencl: A case study using modern C++ libraries. *SIAM J. Scientific Computing*, 35(5), 2013.

[8] Alessandro Fanfarillo, Tobias Burnus, Valeria Cardellini, Salvatore Filippone, Dan Nagle, and Damian W. I. Rouson. OpenCoarrays: Open-source transport layers supporting coarray Fortran compilers. In Allen D. Malony and Jeff R. Hammond, editors, *Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models, PGAS 2014, Eugene, OR, USA, October 6-10, 2014*, pages 4:1–4:11. ACM, 2014.

[9] Magne Haveraaen. Machine and collection abstractions for user-implemented data-parallel programming. *Scientific Programming*, 8(4):231–246, 2000.

[10] Magne Haveraaen, Karla Morris, Damian W. I. Rouson, Hari Radhakrishnan, and Clayton Carson. High-performance design patterns for modern Fortran. *Scientific Programming*, 2015:942059:1–942059:14, 2015.

[11] Hans Petter Langtangen. *Computational Partial Differential Equations - Numerical Methods and Diffpack Programming*, volume 1 of *Texts in Computational Science and Engineering*. Springer, 2003.

[12] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. Basic linear algebra subprograms for fortran usage. *ACM Trans. Math. Softw.*, 5(3):308–323, September 1979.

[13] M. Metcalf, J. Reid, and M. Cohen. *Modern Fortran Explained*. Oxford University Press, 2011.

[14] Aaftab Munshi, Benedict Gaster, Timothy G. Mattson, James Fung, and Dan Ginsburg. *OpenCL Programming Guide*. Addison-Wesley Professional, 1st edition, 2011.

[15] Peter S Pacheco. *Parallel programming with MPI*. Morgan Kaufmann, 1997.

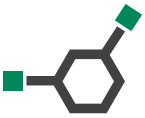
[16] Craig Rasmussen, Matthew Sottile, Soren Rasmussen, Dan Nagle, and William Dumas. Cafe: Coarray Fortran extensions for heterogeneous computing. In *Proceedings IPDPS Workshops*, page to appear, 2016.

[17] Ruymán Reyes, Iván López, Juan J. Fumero, and Francisco Sande. A preliminary evaluation of openacc implementations. *J. Supercomput.*, 65(3):1063–1075, September 2013.

[18] Damian W.I. Rouson, Jim Xia, and Xiaofeng Xu. *Scientific Software Design: The Object-Oriented Way*. Cambridge University Press, 2011.

[19] Alexander Stepanov and Paul McJones. *Elements of Programming*. Addison-Wesley Professional, 1st edition, 2009.

[20] Todd L. Veldhuizen. Blitz++: The library that thinks it is a compiler. In Hans Petter Langtangen, Are Magnus Bruaset, and Ewald Quak, editors, *Advances in Software Tools for Scientific Computing*, volume 10 of *Lecture Notes in Computational Science and Engineering*, pages 57–87. Springer Berlin Heidelberg, 2000.



cost

Highly Tuned Small Matrix Multiplications Applied to Spectral Element Code Nek5000

Berk Hess, Jing Gong, Szilárd Páll

KTH Royal Institute of Technology, Sweden
hess,gongjing,pszilard@kth.se

Philipp Schlatter, Adam Peplinski

Department of Mechanics, KTH Royal Institute of Technology, Sweden
pschlatt,adam@mech.kth.se

Abstract

Nek5000 is an open-source code for simulating incompressible flows using MPI for parallel communication. In the Nek5000 code, the tensor-product-based operator evaluation can be implemented as small dense matrix-matrix multiplications. It is clear that the routines for calculating the matrix-matrix product dominate the execution time of Nek5000. In this paper, we conduct the optimization of matrix-matrix multiplication using SIMD intrinsics and the LIBXSMM package. The evaluation of the computational cost and optimization of these subroutines is not only applied to the CFD code Nek5000, but also to the NekCEM and NekLEM software, which share same data structures with Nek5000.

Keywords Spectral Element Method (SEM), Nek5000, Nekbone, Single instruction multiple data (SIMD), LIBXSMM

I. Introduction

Nek5000 [1] is an open-source code for simulating incompressible flows using MPI for parallel communication. The code is widely used in a broad range of applications. The Nek5000 discretization scheme is based on the spectral-element method [2]. In this approach, the incompressible Navier-Stokes equations are discretized in space by using high-order weighted residual techniques employing tensor-product polynomial bases. The tensor-product-based operator evaluation can be implemented as small matrix-matrix multiplication, The main part of the program Nek5000 consists in small matrix-matrix multiplication routines, in which the program spends most of its time (more than 60% in a 2D version) [3].

Currently, the routines are basic FORTRAN routines with nested loops to compute the matrix multiplications in Nek5000. The aim of the work is to enhance the routines using vectorization techniques like SIMD (Single Instruction Multiple Data) instructions [4] and the high performance library for small matrix multiplications LIBXSMM [5].

The remainder of this paper is organized as follow. Section 2 describes the algorithms and the SIMD implementations. Section 3 presents the main performance results. Finally the conclusions and further works are discussed in Section 4.

II. The algorithms and the SIMD implementation

In Nek5000, the small dense matrix multiplication is written as

$$\mathbf{C}_{n_1 \times n_3} = \mathbf{A}_{n_1 \times n_2} \mathbf{B}_{n_2 \times n_3}$$

where the size of n_1, n_2, and n_3 can be N or N^2 with typical $N \in (4-16)$. In the routine written as below we use the "C" ordering wherein columns of **B** are assumed stored consecutively and that successive rows of **A** are stored n_1 floating point words apart in memory, see [7]).

```
int i, j, k;
for (i = 0; i < n1; i++) {
   for (k = 0; k < n3; k++) {
      c[i][k] = 0.0;
      for (j = 0; j < n2; j++) {
```

```
          c[i][k] += a[i][j] * b[j][k];
      }
    }
  }
```

However this implementation is very time consuming since the compilers have a hard time optimizing and vectorizing it. Also there is no hint given for the values of the loop parameters, and the compiler would not take full advantage of the underlying SIMD architecture.

The principle of SIMD instruction is to apply an instruction to multiple operands at once instead of on one operand and thus considerable improving code performance. Recent processors, e.g. Intel Haswell, have support for 256-bit SIMD instructions that operate on 256-bit registers [6] (512-bit for the next generation), thus processing 4 double precision numbers simultaneously. With 2 fused multiply-add operations per cycle per core, this results in a peak throughput of 16 FLOPs per cycle per core. However, with standard code one has to rely on the compiler to extract sufficient SIMD vectorization. Except for triial cases, such as operations on large vectors, this is a difficult task. Furthermore, the throughput is often limited by speed with which the operands can be loaded from memory or L2/L3 cache into SIMD registers.

```
int i, j, k;
for(k = 0; k + 1 < n3; k += 1) {
   simd_db bs0[n2];
   for (int j = 0; j < n2; j++) {
       bs0[j] = simd_broadcast_sd(b + j + k * n2);
   }

   i = 0;
   while(i + SIMD_WIDTH <= n1) {
       simd_db as = simd_loadu_pd(a + i);
       simd_db c0 = simd_mul_pd(as, *bs0);

       for (int j = 1; j < n2; j++) {
           as = simd_loadu_pd(a + i + j*n1);
           c0 = simd_fmadd_pd(as, bs0[j], c0);
       }
        simd_storeu_pd(c + i + k * n1, c0);
        i += SIMD_WIDTH;
   }
   if (i < n1) {
     simd_si mm = simd_castpd_si(simd_loadu_pd(
                  (const double*)mask[n1-i]));
     simd_db as = simd_maskload_pd(a + i, mm);
     simd_db c0 = simd_mul_pd(as, *bs0);
     for (j = 1; j < n2; j++) {
        as = simd_maskload_pd(a + i + j*n1, mm);
        c0 = simd_fmadd_pd(as, bs0[j], c0);
     }
     simd_maskstore_pd(c + i + k * n1, mm, c0);
   }
}
```

To optimize the matrix-matrix multiplication routines in the program we firstly take maximum advantage of the underlying architecture by using SIMD intrinsics, supported by several compilers, and to help the compiler by unrolling the different loops that are involved in the routine [8]. The fact that we have stride-1 access within the j-loops and not necessarily within the i-loops at the same time makes this idea less appealing. Thus, we could aim at SIMD vectorizing to ensure that all j-loops will SIMD vectorize well in the matrix-matrix multiplication and this requires that the compiler does indeed recognize the j-loops as stride-1 loop. By using the instruction set provided for AVX2-compatible architecture (`_mm256_*`). One instruction has been replicated 4 times in $\mathbf{B}$ and 4 rows in $\mathbf{A}$ are computed simultaneously.

$$\begin{bmatrix} |\mathbf{B}_{1,1}| & \cdots & \mathbf{B}_{1,p} \\ \mathbf{B}_{2,1} & \cdots & \mathbf{B}_{2,p} \\ \mathbf{B}_{3,1} & \cdots & \mathbf{B}_{3,p} \\ \mathbf{B}_{4,1} & \cdots & \mathbf{B}_{4,p} \\ \vdots & \ddots & \vdots \\ \mathbf{B}_{n,1} & \cdots & \mathbf{B}_{p,p} \end{bmatrix} \quad \begin{bmatrix} |\mathbf{A}_{1,1}| & \cdots & \mathbf{A}_{1,n} \\ |\mathbf{A}_{2,1}| & \cdots & \mathbf{A}_{2,n} \\ |\mathbf{A}_{3,1}| & \cdots & \mathbf{A}_{3,n} \\ |\mathbf{A}_{4,1}| & \cdots & \mathbf{A}_{4,n} \\ \vdots & \ddots & \vdots \\ \mathbf{A}_{n,1} & \cdots & \mathbf{A}_{n,n} \end{bmatrix}$$

Using assembly code can further optimize loops and memory fetching wherever possible and manually unroll. In Algorithm 1 we shown the core of assembly code for the routine. Most of the loops go downwards instead of the natural upward scheme. We keep in register everything that is often need (loop indexes) to avoid redundancy when possible.

The Intel LIBXSMM is designed in a very flexible way, that is, separated into a frontend (routine selection) and backend (specific xGEMM code generation). As a result, LIBXSMM can achieve its high application level performance for Intel processors. LIBXSMM offers an auto dispatcher which decides which backend should be executed for the given parameter set [5]. Finally we call through the interface of LIBXSMM, which implements the matrix multiplication shown in Algorithm 2 [9].

III. Performance results

To understand the performance implications of SIMD optimization, this paper presents case studies of porting and optimization of kernel benchmarks for a spectral element code Nekbone, which is a simplified version of a computational

Algorithm 1 Assembly SIMD code

```
for_j_loop:
        subq    %r9, %r14
        load_bs0_array

        #Initialisation of i-loop
        movl    %r8d, %r11d
        subq    %r8, %rdi
        subl    $32, %r11d
        jle     while_i_loop_end
while_i_loop:
        loop_mult       for_k_loop
        subl            $32, %r11d
        jg              while_i_loop
while_i_loop_end:
        loop_mult       for_k_loop_in_n1, 1
        decl            %r10d
        jge             for_j_loop
end_of_function:
        popq    %r15
        popq    %r14
```

Algorithm 2 LIBXSMM Interface

```
CALL libxsmm_init()
CALL libxsmm_dispatch(xmm,            &
             n1, n3, n2, alpha=alpha, beta=beta)
IF (libxsmm_available(xmm)) then
   CALL libxsmm_call(xmm, C_LOC(ap),  &
             C_LOC(bp), C_LOC(cp))
ENDIF
CALL libxsmm_finalize()
```

fluid dynamics (CFD) code Nek5000. Nekbone focuses on the Poisson operator evaluation that is a central computational kernel in Nek5000. As kernel benchmarks, we focus on highly tuned matrix multiplications for fine-grained parallelism of matrix-vector multiplications.

An initial performance profiling of Nek5000 application on a single Haswell node was carried out using the Cray Performance Analysis Tools (CrayPAT) profiler. The goal of this profiling work was to identify which subroutines are the most time consuming and can provide enough workload to exploit the SIMD instructions. The profiling table above shows the profiling results. The subroutine `mx2mf2` for the matrix multiplication takes around 42.3% total executive time.

Table 1: Profile by Function

Samp%	Samp	Group / Function
100.0%	2811.0	Total
95.7%	2689.0	USER
42.3%	1190.0	mxmf2_
14.5%	408.0	cg_
12.3%	347.0	glsc3_
11.3%	319.0	add2s2_
4.0%	113.0	add2s1_
3.1%	86.0	jl_gs_gather
2.0%	55.0	jl_gs_scatter
1.7%	47.0	add2_
1.7%	47.0	jl_sortp_ull
1.3%	37.0	jl_sortp_ui
4.3%	120.0	ETC

We carry out the performance tests on Beskow which is a Cray XC40 system, based on Intel Haswell processors and Cray Aries interconnect technology. This system has Intel Xeon E5-2698v3 (Haswell) CPUs with processor frequency of 2.3 GHz.

Figures 1 and 2 show the performance results with number of elements $E = 10000$ and $E = 20000$, respectively. From these figures, we find that better performance can be obtained using the SIMD intrinsics and LIBXSMM. Also SIMD intrinsic code can lead high performance with lower orders of polynomial ($N = 4, 6, 8$).

IV. Conclusion and Future Work

We have studied the performance implications of several optimization of small matrix-matrix multiplication. Specifically an originally optimized version is adapted to Nek5000. Through the SIMD vector instructions and the Intel library LIBXSMM, the results show that the performance significantly improved on the matrix multiplication. The overall performance of Nek5000 has also been improved due to the use of SIMD instructions.

References

[1] P. F. Fischer, J. W. Lottes, and S. G. Kerkemeier, Nek5000 web page, Web page: http://nek5000.mcs.anl.gov.

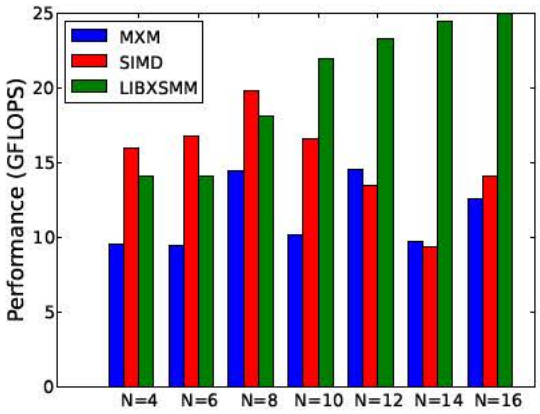


Figure 1: Performance results on a Haswell node with number of elements $E = 10000$

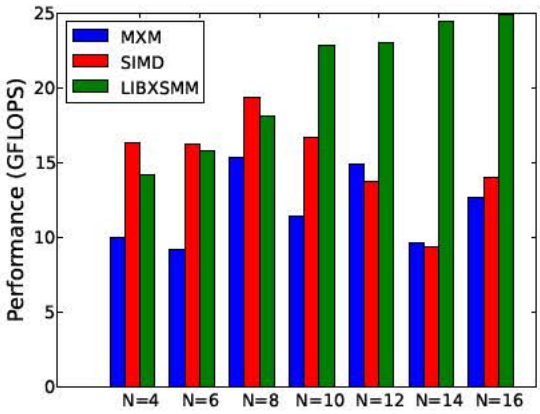


Figure 2: Performance results on a Haswell node with number of elements $E = 20000$

[2] M. Deville, P. Fischer, and E. Mund, *High-order methods for incompressible fluid flow*, Cambridge University Press, 2002.

[3] P. Fischer, J. Lottes, W. D. Pointer, and A. Siegel, "Petascale Algorithms for Reactor Hydrodynamics", *Journal of Physics: Conference Series*, vol. 125, 012076, 2008.

[4] Intel Architecture Instruction Set Extensions Programming Reference, www.naic.edu/ phil/software/intel/319433-014.pdf

[5] A. Heinecke, H. Pabst and G. Henry, "LIBXSMM: A High Performance Library for Small Matrix Multiplications," in *the Proceedings of SC15*, Austin, USA, November 15-20, 2015.

[6] G. Mitra, B. Johnston, A.P. Rendell, E. Mccreath, and J. Zhou, "Use of SIMD vector operations to accelerate application code performance on low-powered ARM and Intel platforms. in *IEEE 27th International Parallel and Distributed Processing Symposium Workshops and PhD Forum (IPDPSW)*, pp 1107-1116, 2013.

[7] W. P. Petersen and P. Arbenz, *Introduction to Parallel Computing, A Practical Guide with Examples in C*, Oxford University Press, 2004.

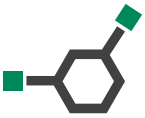
[8] S. Páll and B. Hess "A flexible algorithm for calculating pair interactions on SIMD architectures", *Computer Physics Communications*, vol. 184, no. 12, pp. 2641–2650, 2013.

[9] M. Hutchinson, A. Heinecke, H. Pabst, G. Henry, M. Parsani and D. Keyes, "Efficiency of High Order Spectral Element Methods on Petascale Architectures," in *ISC High Performance 2016 LNCS 9697, J.M. Kunkel et al. (Eds)*, pp. 449–466, 2016.

Acknowledgment

This work is partially supported by EU under the COST Program Action IC1305: Network for Sustainable Ultrascale Computing (NESUS) and the Swedish e-Science Research Center (SeRC).

We would also like to thank Erik Lindahl and Ismael Bouya for help with the SIMD code as well as Alexander Heinecke, Hans Pabst, and Greg Henry from Intel for the LIBXSMM used in the paper.



cost

Automatic Cache Aware Roofline Model Building and Validation Using Topology Detection

Nicolas Denoyelle & Aleksandar Ilic & Brice Goglin & Leonel Sousa & Emmanuel Jeannot

Inria - France – INESC-ID – Portugal

nicolas.denoyelle@inria.fr ilic@sips.inesc-id.pt brice.goglin@inria.fr las@sips.inesc-id.pt emmanuel.jeannot@inria.fr

Abstract

The ever growing complexity of high performance computing systems imposes significant challenges to exploit as much as possible their computational and memory resources. Recently, the Cache-aware Roofline Model has gained popularity due to its simplicity when modeling multi-cores with complex memory hierarchy, characterizing applications bottlenecks, and quantifying achieved or remaining improvements. In this short paper we involve hardware locality topology detection to build the Cache Aware Roofline Model for modern processors in an open-source locality-aware tool. The proposed tool also includes a set of specific micro-benchmarks to assess the micro-architecture performance upper-bounds. The experimental results show that by relying on the proposed tool, it was possible to reach near-theoretical bounds of an Intel 3770K processor, thus proving the effectiveness of the modeling methodology.

Keywords Roofline Model, DRAM, Cache, Tool, Cache Aware Roofline Model, hwloc

I. Introduction

Since the advent of multi-core era, computer systems tend to incorporate an increasing number of cores, while the relative memory bandwidth and memory space per core is decreasing [11]. In order to address application requirement and improve the overall performance, current computing platforms rely on memory hierarchies of increasing complexity. Reshaping applications data layout to take full advantage of those architectures can significantly improve the overall performance at the cost of tremendous development efforts. The Cache Aware Roofline Model (CARM) [5] is able to aggregate this complexity in a single insightful model, and guide application optimization to fit the micro-architecture performance upper-bounds. Its effectiveness motivated us to bring it to non expert developer a robust tool equipped with deep benchmarking of multi-core platforms with complex memory hierarchy, which automatically builds the model and provides the application optimization insights.

To conduct a thorough evaluation of memory and compute capabilities of a given platform, the proposed tool also includes the necessary software support to identify both micro-architecture instruction set and cache topology. The former can be found with compiler support [1], whereas the latter has only been mastered in a portable way by hwloc (hardware locality) library [3]. By relying on this run-time detection of compute and memory resources, the proposed tool automatically instantiates a set of custom platform-specific micro-benchmarks for deep evaluation of platform capabilities, upon which the Cache-aware Roofline Model is generated. Furthermore, the proposed tool also includes a lightweight library to provide access to the hardware counters and extract, at runtime, the application features to be mapped in the model. To the best of our knowledge, there are no existing cross-platform and open-source tools that allow automating this process (i.e building the CARM and mapping applications in it).

The remainder of this paper is organized as follow: Section II describes the original Roofline Model and the Cache Aware Roofline Model. Section III details our tool features, design choices to model the cache hierarchy, and take full advantage of the architecture, and provides preliminary results. Section IV concludes the paper.

II. The Roofline Model Then and Now

The Roofline modeling, in general, is an insightful approach to represent the performance upper-bounds of a processor micro-architecture. Since computations and memory transfers can be simultaneously performed, the Roofline modeling is based on the assumption that the overall execution time can be limited either by the time to perform

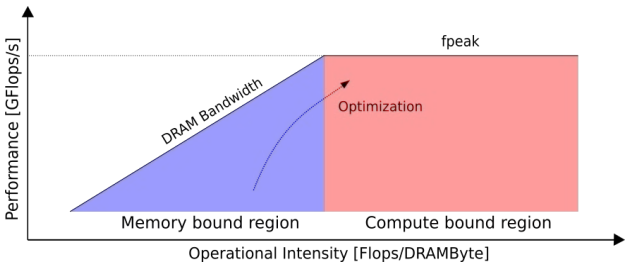


Figure 1: ORM chart

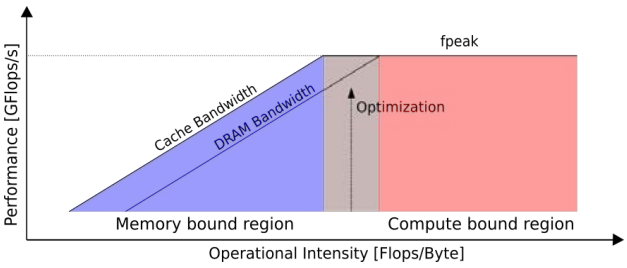


Figure 2: CARM chart

computations or by the time to transfer data. Hence, from the micro-architecture perspective, the overall performance (typically expressed in flops/s) can be limited by the peak performance of computational units or by the capabilities of memory system (i.e., memory bandwidth). To this date, there are two main approaches for Roofline modeling, namely: the Original Roofline Model (ORM) [13] and the Cache-aware Roofline Model (CARM) [5]. These two approaches provide different perspectives when describing the micro-architecture upper-bounds, and they are also differently constructed, validated, and used for application characterization and optimization.

The ORM targets the systems with a processing element (PE) connected to a single (slow) memory (usually, the DRAM). The ORM's PE encapsulates computational units and a set of fast memories (i.e., caches). As such, the ORM mainly considers the memory transfers between the last level cache and the DRAM (commonly referred as DRAM-Bytes). Hence, it denotes the theoretical DRAM bandwidth as one of the potential execution bottlenecks. Depending on the "operational intensity", i.e., the ratio of compute operations (flops) over the quantity of DRAM data (DRAMBytes), the applications can be characterized as compute-bound or memory-bound. The model was used in several works for application optimization [6] [10] [12], as well as to model other.

Figure 1 represents the ORM for a hypothetical computing platform. The axes of the chart are presented in log-log scale, where the "operational intensity" (in flops/DRAMByte) stands on abscissa and the performance (in flops/s) stands in ordinate.

In contrast, the CARM perceives the memory transfers from a consistent micro-architecture point of view, i.e., a core, where the memory transactions are issued. As such, the CARM targets contemporary systems where the PE encloses only compute units and registers, while all other memory levels are separately and explicitly considered. For this purpose, the CARM includes several memory lines in the same plot, each corresponding to the realistically achievable bandwidth of a specific memory level to the core, i.e., cache levels and DRAM. When characterizing the applications, the CARM relies on the true "arithmetic intensity", i.e., the ratio of performed compute operations (flops) over the total volume of requested data (in bytes) by taking into account the complete memory hierarchy (i.e., caches and DRAM).

Fig. 2 shows the CARM general layout for a hypothetical micro-architecture with a single cache level and DRAM. The CARM axes are presented in the log-log scale, where the x-axis refers to the arithmetic intensity (in flops/byte) and the y-axis to the performance (in flops/s). As presented in Fig. 2 (see dashed line), the CARM allows visualizing whether an application with a given arithmetic intensity is memory-bound or compute-bound by observing if a straight vertical line hits a peak (FP) roof or a bandwidth roof.

For these reasons, we base our methodology on the Cache Aware Roofline Model. As explained above, the CARM differs from the original model, it is usually capable of providing deeper insights when analyzing the applications execution bottlenecks, and it also has potential to be adapted to future memory designs. Moreover, the ORM has already a dedicated tool [7] for a similar purpose as ours, but the approach adopted in the herein proposed tool significantly differs and it targets a more consistent and concrete analysis.

III. Locality-Aware Roofline Tool

Our main contribution consists in the development of the open-source tool named Locality Aware Roofline Tool (LART)[1], which exploits hwloc topology detection to automatically build the Cache Aware Roofline Model (CARM).

[1]available at: https://github.com/NicolasDenoyelle/LARM-Locality-Aware-Roofline-Model-

Main tool features.

The proposed LART is composed of 3 main components, namely:

- A set of micro-benchmarks for automatic CARM construction on a given micro-architecture;
- The library for counter-based extraction of CARM metrics from a user application (i.e., the number of performed flops and transferred bytes, as well as the overall execution time);
- A visualization tool to present the model with architecture bounds and applications metrics extraction.

The first component consists in a program that automatically builds the CARM for the specific processor micro-architecture where the tool is run. By relying on a set of hwloc features, the proposed tool automatically detects the memory hierarchy and processor compute capabilities, based on which specific micro-benchmarks are instantiated to deeply evaluate the bandwidth of each memory level, as well as the peak floating point (FP) performance according to the CARM methodology. In addition, the proposed tools also permits to perform the CARM validation tests, by running a set of micro-benchmarks with variable arithmetic intensity. The second component of the tool represents a library with a set of API calls. These API calls are aimed at performing the automatic CARM characterization of a given user application, by instrumenting the application source code. To provide a wider cross-platform portability, this component relies on PAPI [9] features to collect all necessary CARM metrics via hardware performance counters, i.e., to determine the application arithmetic intensity and performance. The third component of the proposed tool is a command-line generating a visual plot of the CARM using platform analysis results. It enables a user to plot application metrics extracted with the above-referred library in the CARM chart. The model validation and bandwidth deviation can also be seen and provide a straightforward evaluation of the confidence one can grant to the model.

Building the model from a hierarchical topology

Discovering all the computing and memory resources in a computing platform can be performed with tools such as hwloc [4]. Prior to hwloc, the similar approaches were often less portable or they were not capable of exposing as many details about cache sharing etc. The hwloc framework models the machine topology as a tree and suits particularly well the caches structure. As presented in Figure 3, the view returned by the hwloc represents, express this structure with

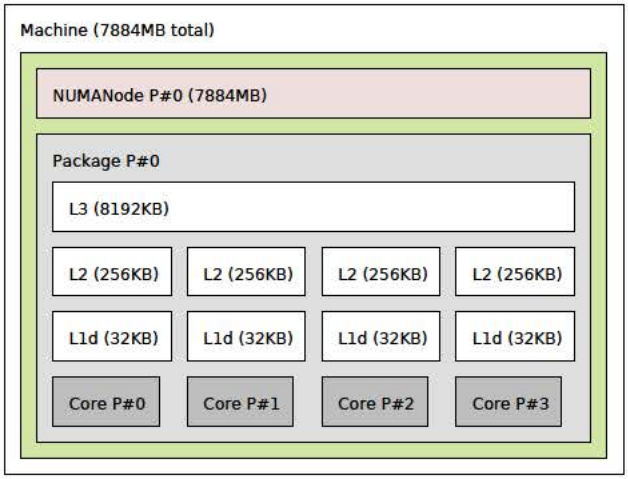


Figure 3: Topology of Intel Ivy Bridge processor model i7 3770k as seen by hwloc

nested boxes. Each core has a stack of 2 private caches, while all cores share the last level cache and the main memory (DRAM). This model where each Core sees the cache hierarchy as a cache stack of increasing size[2], perfectly suits the way how the CARM perceives the memory hierarchy. In addition, the hwloc library also allows a straightforward identification of the cache and memory sizes via the attributes of the Core parent nodes. These parameters are further used in the proposed tool in order to instantiate the appropriate micro-benchmarks for different memory subsystem levels by using the state of the art technique (i.e. buffer streaming of increasing sizes). The floating point peak performance is determined by executing a set of flop instructions in parallel on each core detected by hwloc. However, determining the bandwidth for different levels of memory hierarchy is more challenging, since it is required detecting the cache hierarchy structure with hwloc. This "topology aware" benchmarking technique is detailed in algorithm 1. For each cache level and memory, the proposed tool automatically determines an upper bound and lower bound size, which are subsequently used to build buffers of varying size fitting only the target cache. Afterwards, a specifically developed bandwidth benchmark is performed several times, the median value of all benchmarked sizes is reported as the experimentally determined bandwidth for the target memory level.

[2] The processors use a cache replacement policy where old data from closer caches is evicted in favor of more frequently used ones. The replacement policy defines the method how data is moved from bottom caches to top ones(see in Figure 3). Since the size of the caches closer to cores is smaller than the one for the farther memory levels, the cache stack as seen by each core has an increasing size from bottom to top.

```
Data: topology, repeat
n_threads = hwloc_get_nbobjs_by_type(topology,
 HWLOC_OBJ_CORE);
Core0 = hwloc_get_obj_by_type(topology,
 HWLOC_OBJ_CORE, 0);
/* See subsection III.3, figure 4 for benchmark
   details                                      */
fpeak = median(parallel_flop_uops(repeat));
/* Cache here is a memory subsystem.            */
foreach cache in ancestors(topology, Core0) do
    min_size = cache.size *
     hwloc_get_nbobj_inside_cpuset_by_type(topology,
     cache.cpuset, HWLOC_OBJ_CORE);
    max_size = ancestor_cache(topology, cache).size;
    for size in min_size:max_size do
        buffer = array_of_size(size/n_threads);
        /* See subsection III.3, figure 5 for
           benchmark details                      */
        time = parallel_mem_uops(copy(buffer));
        bandwidths[size] = buffer.size*n_threads/time;
    end
    cache.bandwidth = median(bandwidths);
end
```

Algorithm 1: Memory subsystem benchmark algorithm

Reaching the architecture upper-bounds

Nowadays, general purpose processors usually implement a variety of vector operations, also named as Single Instruction Multiple Data (SIMD) operations. Depending on the target micro-architecture, the tool proposed herein is able to automatically detect the operation type that allows to fully exploit the micro-architecture capabilities (typically, the widest vector instructions). These instructions refer to both compute operations and memory transactions, where the performance upper-bound of each involved unit is expressed as a function of the register size (i.e the number of floating point elements it contains) and the achievable throughput. By compiling the benchmarks on target architecture, we ensure that the largest vector size is used for the benchmarks by interpreting the compiler macros. For instance, figure 4 presents a set of instruction for MUL roof measure on architecture supporting AVX SIMD instructions. Each MUL instruction (vmulpd) is performed using a single register (ymm) for both MUL operands, i.e., it is equivalent to squared value. By ensuring the use of a single register per FP operation, the register dependencies among different instructions are avoided, which allows exercising the full potential of FP units in terms of the achievable throughput.

It is worth to emphasize that typically there are several types of memory/compute instructions on modern processors, and separate hardware units capable of performing different operations simultaneously. For instance, a core may perform a multiplication (MUL) and an addition (ADD) on separate FPUs, which can also be performed in parallel when there are no dependencies between them. Hence, a core can provide significantly higher performance for the codes that fully interleave ADD and MUL operations. This principle also applies to the memory subsystem, where several ports can be dedicated in modern processors to simultaneously serve different number of load (LD) and store (ST) operations, e.g., two LD and one ST 128-bit ports in the Intel Ivy Bridge micro-architecture. Hence, in order to exercise the full compute and memory capabilities of the target architecture, the proposed tool relies on several types of operations to benchmark the platform and it selects by default the one used by the CARM, e.g.for the Intel Ivy Bridge, it interleaves 2 LD and 1 ST instruction when assessing the peak memory bandwidth, while one ADD and one MUL are interleaved for peak FP performance. Figure 5 shows a 2 LD and 1 ST instruction set as used in our bandwidth benchmarks for architecture supporting AVX SIMD instructions.

```
loop:
 vmulpd %%ymm0, %%ymm0, %%ymm0
 vmulpd %%ymm1, %%ymm1, %%ymm1
 ...
 vmulpd %%ymm15, %%ymm15, %%ymm15
sub $1, (%[n_times])
jnz loop
```

Figure 4: assembly sample for MUL fpeak benchmark. The run time, the register size, and the number of instructions, determine the floating point peak performance of the unit running the benchmark.

LART Reproducing CARM Experimental Results on Intel Ivy Bridge

Figure 6, shows an output of the CARM plot generated by the herein proposed tool for an Intel i7 3770k (Ivy Bridge) processor, which topology is previously displayed in Figure 3. The black, red, green and blue oblique lines distinguish several regions of the attainable performance upper-bounds for AVX instructions, which are limited by the bandwidth of different memory levels , i.e., L1, L2, L3 and DRAM, respectively. The two horizontal lines represent the peak FP performance for MUL/ADD and multiplication with addition (MAD).

It is worth to note that the proposed tool was capable of reaching the near-theoretical upper-bounds of the tested

```
loop:
vmovapd (%[buf]), %%ymm0
vmovapd 32(%[buf]), %%ymm1
vmovapd %%ymm2, 64(%[buf])
vmovapd 96(%[buf]), %%ymm3
vmovapd 128(%[buf]), %%ymm4
vmovapd %%ymm2, 150(%[buf])
add $182, %[buf]
sub $182, %[buf_size]
jnz loop
```

Figure 5: assembly sample for 2LD 1ST bandwidth benchmark. The loop is run several times until the whole buffer is walked. The run time, the register size, and the number of instructions determine the bandwidth of the unit running the benchmark.

micro-architecture both for the the L1 bandwidth and peak FP performance. In particular, by relying on the CARM testing methodology, the throughput of 1.49 instructions per cycle (IPC) was achieved for the L1 AVX-256 accesses. In addition, the IPC of 1.98 was achieved for FP performance, which closely match the theoretical throughput of AVX FP instructions when overlapping ADD and MUL operations.

The colored points matching the CARM lines represent the results of the validation benchmarks provided within the proposed tool, i.e., a set of synthetic benchmarks tailored to hit the performance upper-bounds of the micro-architecture for different arithmetic intensities.

As presented in Figure 6, legend in the bottom right corner, includes first the memory subsystem, then the micro-operation type(i.e. 2ld1st - interleaving of 2 LD and 1 ST) and the experimentally obtained bandwidth. On the top right corner in Figure 6, the legend refers to the tested applications for which the CARM metrics were extracted with our library. Those applications express different arithmetic intensity and are well suited to be analyzed with this model. In particular they represent application potential hot spot and come from well known benchmarks named as HPCCG(from Mantevo [2] mini-applications) and STREAM [8]. Although deep performance evaluation of those applications is out of the scope of the paper, it is worth to note that the proposed LART tool is capable of providing the facilities visually analyze the behaviour even for real-world applications.

IV. Conclusion and future work

On the path of extreme scale computing, computer systems complexity is increasing to address hardware and software constraints. The CARM is able to aggregate this complexity and by relying on hwloc topology detection capability we developped a robust tool to build this model and characterize applications. The LART tool is capable of performing deep platform analysis, as well as model validation with automatic detection of micro architecture capabilities and topology. In order to further ease the burden of platform-specific benchmarking for non expert developers the proposed tool also provides a library to project and visualize applications in the model. The efficiency of the proposed tool was verified on a computing platform with Intel Ivy Bridge micro-architecture, where the obtained experimental results show that the proposed tool was capable of reaching near-theoretical performance.

In a close future, we plan to extend the tool and the model to cover heterogeneous memory systems and show their usefulness to improve data spatial locality in Non-uniform memory access (NUMA) systems, while the current model is manly used to improve data temporal locality with cache usage optimization.

V. Acknowledgements

We would like to acknowledge Action IC1305 (NESUS) for funding this work.

References

[1] GCC documentation on platform specific macros. https://gcc.gnu.org/onlinedocs/gcc-6.2.0/gcc/x86-Built-in-Functions.html#x86-Built-in-Functions.

[2] Richard F Barrett, Paul S Crozier, DW Doerfler, Michael A Heroux, Paul T Lin, HK Thornquist, TG Trucano, and Courtenay T Vaughan. Assessing the role of mini-applications in predicting key performance characteristics of scientific and engineering applications. *Journal of Parallel and Distributed Computing*, 75:107–122, 2015.

[3] François Broquedis, Jérôme Clet-Ortega, Stéphanie Moreaud, Nathalie Furmento, Brice Goglin, Guillaume Mercier, Samuel Thibault, and Raymond Namyst. hwloc: a Generic Framework for Managing Hardware Affinities in HPC Applications. In IEEE, editor, *PDP 2010 - The 18th Euromicro International Conference on Parallel, Distributed and Network-Based Computing*, Pisa, Italy, February 2010.

[4] Brice Goglin. Exposing the Locality of Heterogeneous Memory Architectures to HPC Applications. In *1st*

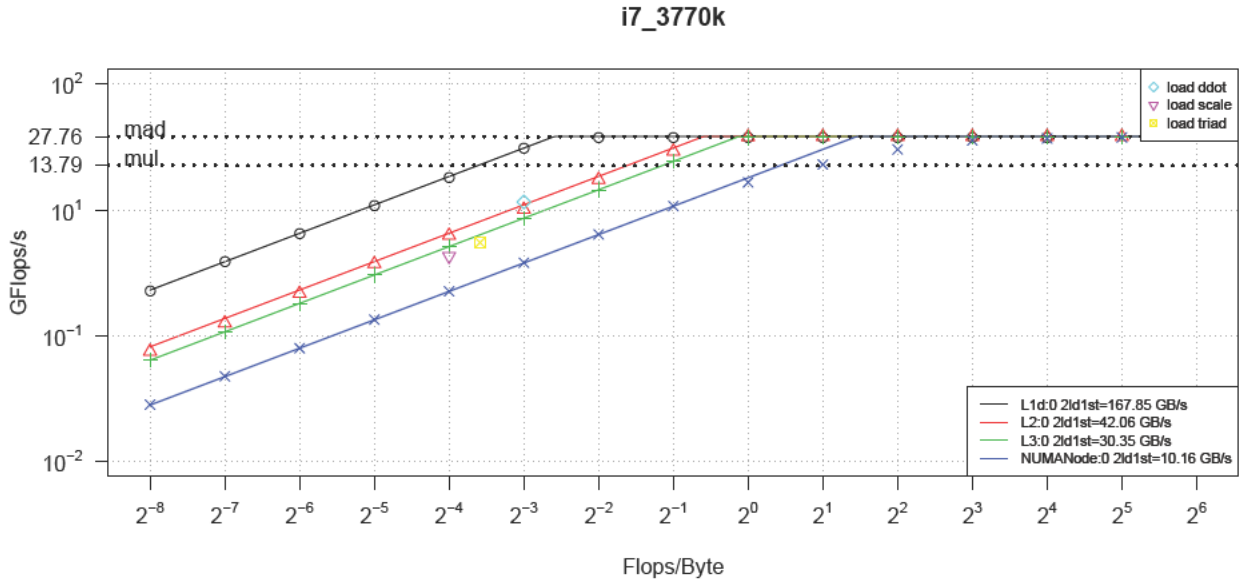


Figure 6: CARM on i7 3770k with LART tool.

ACM International Symposium on Memory Systems (MEMSYS16), Washington, DC, United States, October 2016. ACM.

[5] Aleksandar Ilic, Frederico Pratas, and Leonel Sousa. Cache-aware roofline model: Upgrading the loft. *IEEE Computer Architecture Letters*, 13(1):21–24, 2014.

[6] Ki-Hwan Kim, KyoungHo Kim, and Q-Han Park. Performance analysis and optimization of three-dimensional {FDTD} on {GPU} using roofline model. *Computer Physics Communications*, 182(6):1201 – 1207, 2011.

[7] Yu Jung Lo, Samuel Williams, Brian Van Straalen, Terry J. Ligocki, Matthew J. Cordery, Nicholas J. Wright, Mary W. Hall, and Leonid Oliker. *Roofline Model Toolkit: A Practical Tool for Architectural and Program Analysis*, pages 129–148. Springer International Publishing, Cham, 2015.

[8] John D McCalpin. Stream benchmark. *Link: www. cs. virginia. edu/stream/ref. html# what*, 22, 1995.

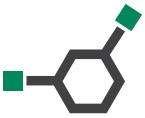
[9] Philip J Mucci, Shirley Browne, Christine Deane, and George Ho. Papi: A portable interface to hardware performance counters. In *Proceedings of the department of defense HPCMP users group conference*, pages 7–10, 1999.

[10] Diego Rossinelli, Christian Conti, and Petros Koumoutsakos. Mesh–particle interpolations on graphics processing units and multicore central processing units. *Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 369(1944):2164–2175, 2011.

[11] Avinash Sodan. Multi Core Trends in High Performance Computing. `https://www.sics.se/sites/default/files/pub/sics.se/avinash_final_sweden_many_core_day_keynote_-_avinash_final_-_clean.pdf`.

[12] Rob V. van Nieuwpoort and John W. Romein. Using many-core hardware to correlate radio astronomy signals. In *Proceedings of the 23rd International Conference on Supercomputing*, ICS '09, pages 440–449, New York, NY, USA, 2009. ACM.

[13] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: An insightful visual performance model for multicore architectures. *Commun. ACM*, 52(4):65–76, April 2009.



cost

Energy-efficient Assignment of Applications to Servers by Taking into Account the Influence of Processes on Each Other

Mateusz Jarus[†], Ariel Oleksiak[†‡], Wahi Narsisian[*], and Hrachya Astsatryan[*]

[†]Poznań Supercomputing and Networking Center, ul. Jana Pawla II 10, 61-139 Poznań
[‡]Poznań University of Technology, Pl. Marii Skłodowskiej-Curie 5, 60-965 Poznań
[*]Institute for Informatics and Automation Problems of the National Academy of Sciences of the Republic of Armenia, P. Sevak 1, Yerevan 0014, Armenia

Abstract

The power consumption of data centers is becoming a crucial challenge in the context of the steadily increasing demand for computation. In this regard finding a way to improve energy efficiency of running applications in data centers is becoming a crucial trend. One method to improve the processor utilization is the consolidation of applications on physical servers. It is possible to run multiple jobs in parallel on the same machine, especially when their requirements regarding computation are smaller than the maximum processor performance. It reduces the number of servers in the data center required to handle multiple requests and therefore leads to energy usage reductions. In this paper, we introduce a realistic model of applications with deadlines executed in parallel on a server and competing for the shared resources and present an energy-aware algorithm which may be used to minimize the overall energy consumption of the servers.

Keywords Data centers, Energy efficiency, Processor utilization, Applications scheduling

I. Introduction

Data centers are under pressure to transform their infrastructure to reduce energy cost, increase reliability and efficiency. Increasing the data volumes and network traffic in data centers is a worldwide trend. At the same time, the number of applications running in these data centers is becoming bigger and bigger over time. The types of the executed applications differ and include databases, file servers, middleware and various others. The difference between such data centers and typical HPC supercomputers is that it is natural in such places to co-locate dozens of tasks on a single physical node. It is a method for improving resource utilization. The relocation of the applications on servers is playing an important role to decrease the number of physical servers in data centers and to reduce the energy consumption.

In the case of data centers particularly important is the Service Layer Agreement which needs to fulfilled. In our model, it is introduced in the form of deadlines for the tasks.

In this paper, we create a realistic model of applications with deadlines executed in parallel on a server, which compete for the shared resources, such as memory or disk. We explain the observations from the experiments that create the basis for the model. We describe how the processor time quantum is shared between the applications and how their performance degrades through the use of the shared resources. We also present a Branch and Bound algorithm which may be used to minimize the overall energy consumption of the servers.

The reminder of this paper is divided into the following sections: Section 2 presents related work; Section 3 describes the model; Section 4 shows the performed experiments and their results; Section 5 concludes the paper.

II. Related work

As virtualization [1] has become the most widespread used technology in modern data centers, and due to the advances in virtualization technologies it is much easier to manage the allocation of tasks to the available resources. The live migration technique allows moving a running virtual machine from one physical server to another with no impact on virtual machine availability. Increasingly popular becomes the Docker platform, which allows starting up its containers even ten times faster than a standard virtual machine. The management of tasks is therefore very fast and efficient. However, the allocation of tasks to servers to maximize the utilization of resources

remains a challenge.

In [2] the authors illustrate the workload sensitivity to the machine on which it executed and the type of co-running applications. They analyzed co-running different applications on various processors and proved that it resulted in various levels of performance degradation of these jobs. The authors observed significant performance variability from the heterogeneity of the datacentre and from the co-allocation of applications. It is, therefore visible that in order to efficiently utilize available resources it is required to take into account the type of applications that are executed in parallel.

Multiple researchers have aimed at creating an algorithm to increase the utilization of machines in datacentres. In [3] the authors propose a Bubble-Up characterization methodology that enables the accurate prediction of the performance degradation that results from the contention for shared resources in the memory subsystem. Using this methodology they can improve the utilization of a 500-machines cluster by 50% to 90%.

In [4] the authors propose a performance model that considers the interferences in the shared last-level cache and memory bus. They also present a virtual machines consolidation method which is based on their interference model.

In [5] the authors propose a new resource management model for the collocation of different tasks that share a single physical machine. The model uses two parameters of a task – its size and its type – to characterize how a task influences the performance of other tasks allocated on the same machine.

However, all of the above methods are simplified. They take into account only one parameter of the application (such as memory accesses) or model the interference between applications by using one artificial parameter specified by the user. Experiments on real hardware prove that the dependencies between jobs are more complicated.

III. Model of tasks executed in parallel on a single machine

We propose a mathematical model that simulates the complex dependencies between co-running applications and hardware. It based on the observation that each of the executed benchmarks affects the underlying hardware by utilizing its resources (processor, memory, hard drive, etc.). The load exerted on these subcomponents influences in turn other co-running applications – their performance degrades due to the need to compete for shared resources. The model does not try to simulate the interactions between applications per se, but rather captures the relationships between applications that appear when sharing the available resources.

In the model both the processor performance and the application, size is represented as Instructions Per Second (IPS). When all of the applications exert load that is equal to or smaller than 100%, the server has enough performance to efficiently execute all of them. The situation becomes more complicated when the total requirements from applications are higher, for example, if each of the two applications requires 60% of the CPU load. In such case, they exceed the maximum processor performance. It is possible to execute them sequentially with the expected performance. They may also run in parallel but slower due to: a) the competition for shared resources, b) not satisfied CPU performance requirements. In both cases, the overall energy consumption and the duration of the execution may be analyzed.

To explain this mechanism in more detail, consider an application X that executes on a given server in T1 seconds and exerts the L1 load on the CPU. Another application with L2 load on the CPU may be executed in parallel, where L1 + L2 <= 100%. The execution time of X will increase slightly due to the interference effect, as they will compete for shared resources, such as memory or disk. Starting additional applications will further extend the execution time of application X. As long as the aggregated load from all applications will be smaller than 100%, the performance degradation of application X will only result from the increasing load on the shared resources. However, when CPU load exceeds 100%, another factor of performance degradation becomes visible. The execution of the application is affected by periods of inactivity when it needs to wait for the processor time quantum.

We have performed a few experiments on Intel Core i5 6200U with three different applications, each exerting different load on the CPU: *pi* (30%), *siege* (50%) and *openssl* (65%). We tested the execution time of *pi* application while running additional applications in parallel. When the aggregated load was lower than 100%, the execution of *Pi* increased slightly. However, it increased significantly more after exceeding 100% CPU load, when all applications were executed in parallel.

The situation changes when the power consumption is considered. Starting additional applications increases the power consumption of the processor proportionally to the load that they make.

Similar experiment was performed with the same three applications, but this time to calculate the power consumption of the processor. The CPU power increased only until the CPU load was below 100%. After this point it stabilized. Since it is not possible to exceed the maximum processor speed, after reaching the point of 100% CPU load the power consumption did not change. However, the execution time of the applications increased significantly, having an impact on the whole energy consumption

When realistic energy efficient job scheduling is considered, two challenges need therefore to be analyzed.

- the interference of applications on each other when they compete for shared resources,
- the calculation of the execution time of applications when their aggregated performance requirements exceed maximum processor performance.

III.1 Interference of applications

Our model is based on the observation that the applications do not affect directly each other but rather influence the underlying hardware, which in turn has an impact on the other executed applications. For example, accessing the memory by one application may cause a delay in accesses by another application.

In the model for each application different parameters regarding hardware may be specified, such as the number of memory accesses or disk usage. The more parameters are defined, the more accurate the results, but at the same time, the more data needs to be collected to run the experiments. For each application, there also needs to be defined a function of execution slowdown due to the aggregated load of a given subcomponent. It may be calculated using the Bubble-Up methodology, presented in [3]. It enables the accurate prediction of the performance degradation using a tunable amount of "pressure" to the subcomponent – memory in the case of this paper. "Bubble" is an artificial benchmark which is only used to stress the server memory. For different values of this pressure, the performance degradation of the original application is analyzed.

III.2 The extension of the execution time due to higher processor performance requirements

The model is based on the fact that the processor time quantum is consistently shared between the executed applications. This situation is presented in Figure 1. In this example, the maximum processor performance is 10 IPS and is named here "an execution window". This run window is moved down in each second and shared between neighboring applications. Linux Completely Fair Scheduler is based on the same assumption that each application receives a fair amount of time quantum – according to its needs.

Figure 2 presents the new speed of execution of an application when the aggregated requirements of all applications are higher than the maximum processor performance. The size of the execution window is equal to the maximum processor performance – 10 IPS in this example. For the sake of clarity the applications are analyzed for a time which is equal to the time window – though the calculations are general and independent of the length of execution of any application. Variable x represents the exceeded processor performance. In this example there are four applications, x is calculated as $x = (s1 + s2 + s3 + s4) - perf$, where $perf$ is the maximum processor performance and $s1$, $s2$, $s3$ and $s4$ are the execution speeds or the CPU loads exerted by the four consecutive applications. New speed of any application may be calculated as $IPS_{new} = \frac{s*c}{c} - \frac{s*x}{c}$. For example, the original speed of the second application in Figure 2 was 4 IPS, while when running with three other applications in parallel it slows down to $\frac{40}{13}$ IPS $\approx$ 3,08 IPS (interference effect due to the competition for shared resources is not considered in these calculations yet).

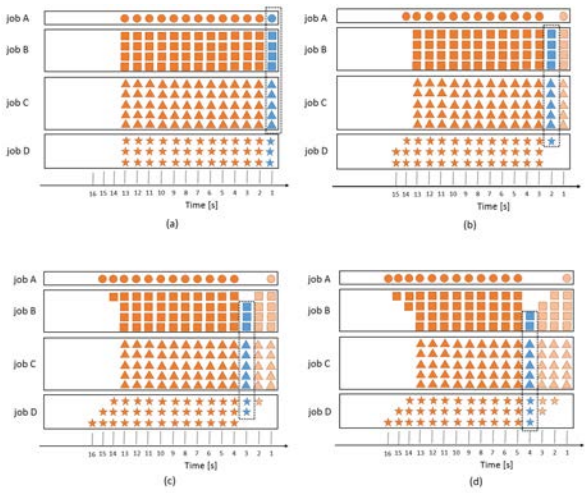


Figure 1: Model of the execution of the applications that exceed 100% of the processor load

IV. The energy-aware job scheduling algorithm

The input parameters for the algorithm are:

- the maximum processor performance (in IPS),
- the maximum power consumption of the processor,
- the number of all instructions for each application,
- the initial requirements for each application regarding processor performance (in IPS),
- the requirements for each application regarding its hardware usage (such as memory or disk usage),
- the function of performance degradation for each application due to the load exerted on different subcomponents of the server (such as memory or disk),
- the deadline for each application.

To calculate the optimal solution for a given processor and a number of various applications we implemented a Branch and Bound algorithm. It analyzes all correct instances of the problem. It starts with an array of a size N x N, where N is the number of applications. Each analyzed job may allocated to one of the N x N cells in the array. Columns represent the sequential execution of applications, while rows allow them to run in parallel. More generally – X axis represents passing time, while Y axis is the load of the CPU. An example instance of the problem presented in Figure 3 a). All of the jobs are allocated to the first column. Therefore all of them

should be executed at time 0 in parallel. Figure 3 b) shows their final execution on the processor. It is important to underline here that the array in Figure 3 a) does not take into account the length of the execution of any job and its requirements regarding processor performance. At this stage these values are not calculated, only their relative position against each other considered here.

Figure 4 a) presents another example of scheduling the tasks. In this case, there is a blank space between an orange and a green task. Figure 4 b) shows how these applications will executed on the server. It represents a situation where the green task should not be executed in parallel with the blue task.

Please note that the position of the green task on the Y axis has no meaning, since there is no other job running in parallel. In this case only the height of the green task is significant as it represents the CPU load. In Figure 4 b) the green task may be therefore depicted at the same level as the blue task.

Please also note that if the green application would be scheduled in the same last column but in the lower row (the same row as the blue task), this allocation would not be correct. It would represent a situation in which there should be a delay of execution between the blue and the green task. However, since the orange application is shorter than the blue one, there is no other application that could separate them. Artificial delays of any length are not considered by the algorithm since they are useless. They do not improve the energy consumption and do not prevent from exceeding the deadlines. Such a schedule would be correct if the orange application would be longer than the blue one. At this stage this information is not available yet – the correctness of the instance validated at a later stage.

The algorithm creates all possible instances of the problem using a Branch and Bound technique. A few different instances of the problem are presented in Figure 5.

For each instance of the problem in the first step the algorithm calculates the time when each application finishes its execution. An example is presented in Figure 6. Vertical borders that represent these times create different phases of execution. If the length of the execution of each application is different, there are always as many stages as the number of applications, no matter what is their relative order. Please note that in each phase the same application might have a different speed of execution. In this example the maximum processor performance is not exceeded, therefore it does not contribute to a slowdown of any application. However, if that would be the case here, the green application in phase 2 would have a higher speed (higher height in the figure) because it would no longer share the processor time quantum with the blue application. For this reason, every phase needs to be analyzed separately.

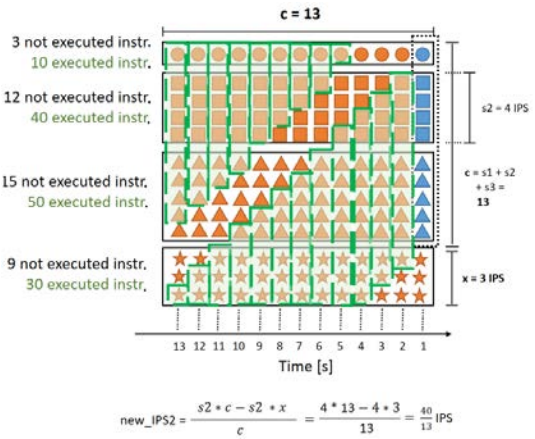


Figure 2: New speed of execution of an application after exceeding maximum processor performance

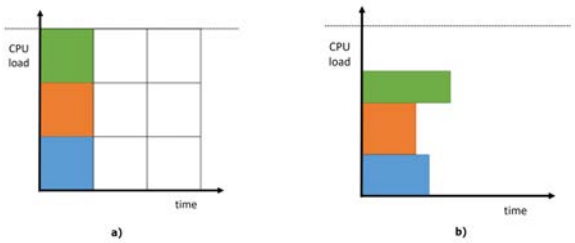


Figure 3: a) An example scheduling of the tasks and b) their final execution on the processor.

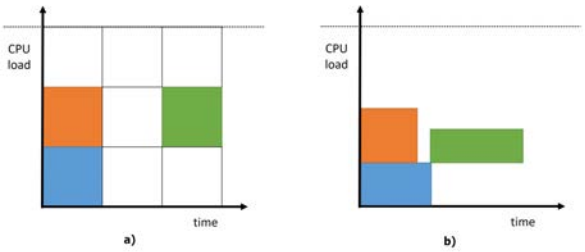


Figure 4: a) Another example of scheduling the tasks with a delay between an orange and a green task and b) their final execution on the processor.

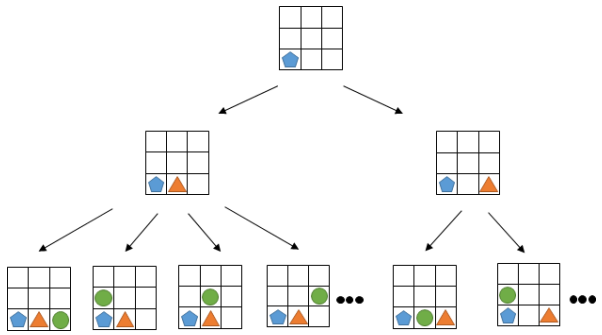


Figure 5: A few different instances of the problem created by the Branch and Bound algorithm.

In the second step for each row a bidirectional list is created (see Figure 6 c). Each item on these lists represents either a given job or an empty space between them. Each of these elements will hold the time when its phase finishes.

The algorithm iterates over every phase. For each list is saves the pointer to the currently analyzed item. For each phase it does two rounds – in the first one it analyzes items that are jobs, it the second round it analyzes blank items.

It starts with the first item on every list. If the first item on a given list represents a job (in our example on both lists the first items are jobs), it saves in it the execution time of this job, which is calculated as $time = instructions/speed$.

This value is added to the list *borders*, which holds information about the times of consecutive borders between phases. It moves the pointer of the currently analyzed item to the next one. While the next item on the given list is also a job, it repeats the same procedure – it calculates the time of the execution of this job. It adds to it the time of the previous item on the list and saves this value inside the currently analyzed item. It also adds it to the list *borders*. If the next item on a given list is a blank space, the algorithm moves to the next list and repeats the same procedure.

When all first jobs on each list are analyzed, the algorithm moves to the second phase – it examines blank spaces for each list. The algorithm checks whether the first item on the list *borders* is higher than the value saved for the previous item on the analyzed list. If yes, it saves it inside the item and moves the pointer to the currently analyzed item to the next one. If not, it leaves the item untouched. When all blank spaces in this phase for every list are checked, it removes the first item on the *borders* list.

The algorithm then moves to the next phase and repeats the whole procedure until all items on all lists are checked.

This step calculates the execution times for each job and the phases in which they are run. For instance, in the analyzed example, it shows that the green application is executed in phase 1 and 2, the blue one only in phase 1, while the orange one only in phase 3. It also shows which applications are run in parallel with others in every phase.

Since the allocations to different stages are now known, the algorithm may calculate for each phase the processor load and the aggregated loads exerted on the subcomponents, such as memory. For example, in phase 1 it sums up the memory requirements of the green and the blue application. Then it analyzes the speed degradation for each of them under this aggregated memory load. Based on that information it updates the time of each phase, already saved in the previously mentioned lists. In the next step, it analyzes the new speed of each application according to the calculations presented in section III.2. Based on that information it again updates the time of each phase, already saved in the previously mentioned lists.

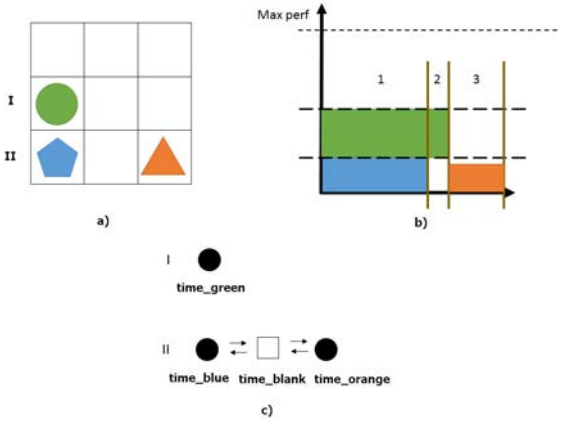


Figure 6: a) An example instance of the problem, b) marked the end of execution for each application (borders between phases) and c) the lists for each row that present the dependencies between tasks

The final result is the time of the end of every phase. It allows the algorithm to calculate the whole time required to run all of the applications. It also verifies the deadlines – if they are exceeded, the solution is treated as unacceptable. The energy consumption may be calculated as the time multiplied by the power (both of these values are known).

Since this is a Branch and Bound algorithm, the currently analyzed solution is compared to the previously saved. If it is better than the previous one, it will consider the best option. Finally, the algorithm returns the best instance from all analyzed.

V. Experiment

To test the algorithm we have started it with five applications. The applications compete for one shared resource, which is a memory.

The number of instructions, initial CPU and memory requirements and deadlines for each application is specified in Table 1. Applications three and four have high memory requirements. The first and the last applications have many instructions to execute (at the same time their execution would be the longest without taking into account the interference effects). All of them have specified deadlines.

Table 2 presents speed degradation of all applications in function of the memory load. For example, application one slows down by 5% when the memory load is 10 (e.g. Mb/s).

Name	Instructions	CPU	Memory	Deadline
one	1000	40	10	50
two	200	15	1	40
three	400	45	55	30
four	400	60	55	12
five	1000	30	10	40

Table 1: Parameters of five applications used to test the algorithm

	Application slowdown				
Memory	one	two	three	four	five
10	5,00%	5,00%	10,0%	15,0%	5,00%
20	6,00%	6,00%	18,0%	20,0%	6,00%
30	7,00%	6,30%	21,0%	25,0%	7,00%
40	7,30%	6,80%	23,0%	30,0%	7,30%
50	7,60%	7,10%	30,0%	35,0%	7,60%
60	8,00%	7,20%	35,0%	40,0%	8,00%
70	8,10%	7,20%	38,0%	45,0%	8,10%

Table 2: Speed degradation of applications in function of the memory load

Figure 7 presents the most energy-efficient scheduling for the proposed parameters. The time of calculations is 47,48 seconds and the energy consumed is 1751 Ws. In this solution, no deadlines are exceeded. It is also visible that the maximum processor performance is not exceeded in any phase. All of the applications are executed with the initially required speed.

VI. Conclusion

In this paper, we presented a model of applications with deadlines executed in parallel on a server, which compete for the shared resources, such as memory or disk. This model realistically represents the real execution of applications on physical servers, taking into account their speed, hardware requirements and performance degradation due to loaded subcomponents of the server. We presented a Branch and Bound algorithm to calculate the most energy-efficient scheduling of jobs. The algorithm was verified by five applications with specified hardware requirements and deadlines.

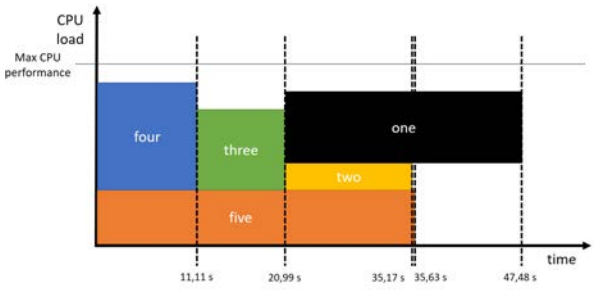


Figure 7: The most energy-efficient scheduling for the selected applications.

Acknowledgment

This work is partially supported by EU under the COST Program Action 1305: Network for Sustainable Ultrascale Computing (NESUS). The research presented in this paper is partially funded by a grant from Polish National Science Center under award number 2013/08/A/ST6/00296. This research was supported by the EU Seventh Framework Programme FP7/2007–2013 under grant agreement no. FP7-ICT-2013-10 (609757).

References

[1] M. Wang, X. Meng, L. Zhang, "Consolidating virtual machines with dynamic bandwidth demand in data centers" in: IEEE INFOCOM 2011 Proceedings, Shanghai, China, June 2011, pp. 71-75.

[2] J. Mars, L. Tang, R. Hundt, "Heterogeneity in 'Homogeneous' Warehouse-Scale Computers: A Performance Opportunity", in: IEEE Computer Architecture Letters, vol. 10, No. 2, pp. 29-32, 2011.

[3] J. Mars, L. Tang, K. Skadron, M. L. Soffa, R. Hundt, "Increasing Utilization in Modern Warehouse-Scale Computers Using Bubble-Up", in: IEEE Micro, vol. 32, pp. 88-99, 2012.

[4] S. Kim, H. Eom, H. Y. Yeom, "Virtual machine consolidation based on interference modeling", in: The Journal of Supercomputing, vol. 66, pp. 1489-1506, 2013.

[5] F. Pascual, K. Rzadca, "Partition with side effects", in: IEEE 22nd International Conference on High Performance Computing, vol. 66, pp. 1489-1506, 2013.

cost

On Parallel Numerical Algorithms for Fractional Diffusion Problems

Raimondas Čiegis†, Vadimas Starikovičius†, Svetozar Margenov*

†Vilnius Gediminas Technical University, Lithuania
raimondas.ciegis@vgtu.lt,
vadimas.starikovicius@vgtu.lt

*IICT-BAS, Bulgaria
margenov@parallel.bas.bg

Abstract

In this work, we consider a parallel numerical solution of problems depending on fractional powers of elliptic operator. Three different state of the art approaches are used to transform the original non-local problem into well-known local PDE problems. Parallel numerical algorithms for all three approaches are developed and discussed. Results of their parallel performance tests are presented and analysed.

Keywords Fractional Diffusion, Parallel Numerical Algorithms, Multigrid, Strong Scalability

I. Introduction

The permanent development of the large scale computational systems with all their diversity requires a constant attention, which must be paid to a development and selection of proper parallel algorithms for the solution of various problems. In this paper, we consider a parallel numerical solution of problems involving fractional powers of elliptic operators. Such problems arise in a wide range of areas, including image processing, porous media flow, material sciences (see, e.g., [1] and the references therein.

In this paper, we investigate the scalability and efficiency of parallelization of three state of the art discrete algorithms used for numerical solution of fractional power elliptic problems. These algorithms are reducing the given non-local diffusion problem to some local classical differential problems formulated in spaces of higher dimension. It is important to note, that despite using a common embedding technique, these three approaches lead to very different challenges in construction of efficient parallel algorithms.

The rest of this paper is organized as follows. In Section II, we describe the problem under consideration with fractional power of elliptic diffusion operator. In Section III, three partial differential equations (PDEs) models are formulated and applied to construct efficient numerical solution techniques for considered problems. They are based on different types of PDEs and all of them define local operators but embedded into higher dimension space. The finite volume method is used to approximate the formulated PDEs by the discrete schemes. The parallelization issues of these three numerical solution algorithms are discussed in Section IV. Parallel performance results of the developed parallel algorithms are presented and analysed. The strong scalability is considered. Some final conclusions are given in Section V.

II. Problem Formulation

Let Ω be a bounded domain in $\mathbb{R}^n$, $n \geq 2$ with boundary $\partial\Omega$. Given a function f, we seek u such that

$$L^\beta u = f, \quad X \in \Omega \tag{1}$$

with some boundary conditions on $\partial\Omega$, $0 < \beta < 1$ and

$$Lu = -\sum_{j=1}^{n} \frac{\partial}{\partial x_j}\left(k(X)\frac{\partial u}{\partial x_j}\right).$$

Let us denote by $\{\phi_k\}$, $k = 1, 2, \ldots, N$ the orthonormal basis (for convenience, here we restrict to the case of

finite number of modes typical for discrete approximations)

$$L\phi_k = \lambda_k \phi_k.$$

Then the fractional powers of the diffusion operator are defined by

$$L^\beta u = \sum_{k=1}^{N} \lambda_k^\beta w_k \phi_k, \tag{2}$$

where $w_k = (u, \phi_k)$.

Note, that the direct implementation of this approach is very expensive. It requires the computation of all eigenvectors and eigenvalues of large matrices. This algorithm can be used for practical computations if the fractional power of Laplace operator is solved in rectangular domain, when FFT techniques can be applied.

III. PDE APPROACH FOR THE FRACTIONAL NON-LOCAL MODEL

In this section we formulate three PDE models to approximate problems involving fractional powers of elliptic operators. These approximations allow us to construct efficient solution techniques for the original problem. The formulated PDEs are approximated by the finite volume schemes.

III.1 Extension to the mixed boundary value problem in the semi-infinite cylinder $C = \Omega \times (0, \infty) \subset \mathbb{R}^{n+1}$

Non-local problem (1) is equivalent to the following classical local linear problem in the extended space $\mathbb{R}^{n+1}$ [2, 3]:

$$-\frac{\partial}{\partial y}\left(y^\alpha \frac{\partial V}{\partial y}\right) + y^\alpha L V = 0, \quad (X, y) \in C, \ \alpha = 1 - 2\beta, \tag{3}$$

$$-y^\alpha \frac{\partial V}{\partial y} = d_\beta f, \quad X \in \bar{\Omega} \times \{0\},$$

$$V = 0, \quad (X, y) \in C_B = \partial C \setminus \bar{\Omega} \times \{0\},$$

where d_β is a positive normalization constant that depends only on β. Then $u(X) = V(X, 0)$.

In order to construct a finite volume approximation of (3), the semi-infinite cylinder is approximated by the truncated cylinder $C_Y = \Omega \times \{0, Y\}$ with a sufficiently large Y. A uniform mesh Ω_h is introduced in Ω and anisotropic mesh $\omega_h = \{y_j = (j/M)^\gamma Y, j = 0, \dots, M\}$ is used to compensate the singular behaviour of the solution as $y \to 0$, where $\gamma > 3/(2\beta)$ [2, 3].

By using the finite volume method and standard notations of the finite differences we define the discrete problem, which approximates (3):

$$-\left(y^\alpha_{j+1/2}\frac{V_{h,j+1} - V_{h,j}}{H_{j+1/2}} - y^\alpha_{j-1/2}\frac{V_{h,j} - V_{h,j-1}}{H_{j-1/2}}\right) + \frac{y^{\alpha+1}_{j+1/2} - y^{\alpha+1}_{j-1/2}}{\alpha + 1} L_h V_h = 0, \quad (X_h, y_j) \in C_{Y_h}, \tag{4}$$

$$-y^\alpha_{1/2}\frac{V_{h,1} - V_{h,0}}{H_{1/2}} + \frac{y^{\alpha+1}_{1/2}}{\alpha + 1} L_h V_h = d_\beta f_h,$$

$$X_h \in \bar{\Omega}_h \times \{0\},$$

$$V_h = 0, \quad (X_h, y_j) \in \partial C_{Yh} \setminus \bar{\Omega}_h \times \{0\},$$

where

$$L_h V_h = -\sum_{k=1}^{n} \partial_{x_k}(k(X_h)\partial_{\bar{x}_k} V_h),$$

$$y_{j+1/2} = \frac{y_j + y_{j+1}}{2}, \ H_{j+1/2} = y_{j+1} - y_j.$$

III.2 Integral representation of the solution of initial problem (1)

The algorithm is based on the integral representation of the non-local operator using the classical local operators [4]:

$$L^{-\beta} = \frac{2\sin(\pi\beta)}{\pi}\Big[\int_0^1 y^{2\beta-1}(I + y^2 L)^{-1} dy + \int_0^1 y^{1-2\beta}(y^2 I + L)^{-1} dy\Big]. \tag{5}$$

Different quadrature schemes can be used to approximate these singular integrals. In this paper, we have applied a graded partition of integration interval $[0, 1]$ to resolve the singular behaviour of $y^{2\beta-1}$:

$$y_{1,j} = \begin{cases} (j/M)^{\frac{1}{2\beta}} & \text{if } 2\beta - 1 < 0, \\ j/M & \text{if } 2\beta - 1 \geq 0, \end{cases}, \quad j = 0, \dots, M.$$

A similar partition is used to resolve the singularity of $y^{1-2\beta}$. Then the following approximation of integrals (5) is applied

$$L_h^{-\beta} f_h = \frac{2\sin(\pi\beta)}{\pi} \times \left[\sum_{j=1}^{M} \frac{y_{1,j}^{2\beta} - y_{1,j-1}^{2\beta}}{2\beta} \left(I_h + y_{1,j-1/2}^2 L_h\right)^{-1} f_h \right. \quad (6)$$
$$\left. + \sum_{j=1}^{M} \frac{y_{2,j}^{2-2\beta} - y_{2,j-1}^{2-2\beta}}{2-2\beta} \left(y_{2,j-1/2}^2 I_h + L_h\right)^{-1} f_h \right].$$

One or two level parallelization strategies can be applied to solve the multiple independent local linear sub-problems $(I_h + y_j^2 L_h)^{-1} f$ and $(y_j^2 I_h + L_h)^{-1} f$.

III.3 Reduction to a pseudo-parabolic PDE problem

The solution of non-local problem (1) is sought as a mapping [5]:

$$V(X,t) = \left(t(L-\delta I) + \delta I\right)^{-\beta} f,$$

where $L \geq \delta_0 I$, $\delta = \gamma\delta_0$, $0 < \gamma < 1$.

Thus it follows that $V(X,1) = L^{-\beta} f$. The function V satisfies the evolutionary pseudo-parabolic problem

$$(tG + \delta I)\frac{\partial V}{\partial t} + \beta G V = 0, \quad 0 < t \leq 1, \quad (7)$$
$$V(0) = \delta^{-\beta} f, \quad t = 0,$$

where $G = L - \delta I$.

Again, instead of the non-local problem (1) we solve a non-stationary local pseudo-parabolic problem (formally in $\mathbb{R}^{n+1}$ space). In order to solve (7), we use the following finite volume scheme [6]:

$$(t^{n-1/2} G_h + \delta I_h)\frac{V_h^n - V_h^{n-1}}{\tau} + \beta G_h V_h^{n-1/2} = 0, \quad (8)$$
$$0 < n \leq M,$$
$$V_h^0 = \delta^{-\beta} f_h,$$

where $G_h = L_h - \delta I_h$, $V_h^{n-1/2} = (V_h^n + V_h^{n-1})/2$ and $t^{n-1/2} = (t^{n-1} + t^n)/2$.

IV. Parallel algorithms

In this section we are considering and discussing the parallelization of all three numerical solution algorithms presented in Section III. Our analysis is restricted to the strong scalability, when the size of discrete problems is fixed and different numbers of processors are used in the computations. Such an information is very important when a medium size problem should be solved as fast as possible (consider optimization algorithms when computation of the value of the objective function reduces to numerical solution of fractional power of elliptic problem).

All parallel numerical tests in this work were performed on the computer cluster "HPC Sauletekis" (`http://www.supercomputing.ff.vu.lt`) at the High Performance Computing Centre of Vilnius University, Faculty of Physics. We have used up to 10 nodes with Intel® Xeon® processors E5-2670 with 16 cores (2.60 GHz) and 128 GB of RAM per node. Computational nodes are interconnected via the InfiniBand network.

IV.1 Discrete elliptic problem

The approximate PDE model (3) transforms the non-local fractional diffusion problem (1) into well-studied case of PDEs problems with elliptic operators. The selected finite volume scheme (4) means that our first numerical algorithm essentially deals with a solution of one large system of linear equations. In case, when the problem domain Ω is two-dimensional, one needs to solve a system with 7 point stencil of size $N = N_{x_1} \times N_{x_2} \times M$.

A standard approach for the parallel solution of such problems is the domain decomposition method [7]. The discrete mesh of the problem domain and its associated fields are partitioned into sub-domains, which are allocated to different processes. Note that in our case, the discrete mesh C_{Y_h} of the truncated cylinder $C_Y = \Omega \times \{0, Y\}$ needs to be partitioned. In this work, we use a simple one-dimensional partitioning in y direction.

It is well known that the parallel performance of PDE problem solver essentially depends on the quality of the parallel linear solver. In this work, we have used the parallel multigrid solver from AGMG package [8, 9].

To test the parallel performance of the developed algorithm, we have considered the problem (3) in the 2D unit square domain Ω using the discrete mesh of the size $N_{x_1} = N_{x_2} = 1000$ and $M = 250$. The tolerance of multigrid solver was set to 10^{-6} in all tests. Obviously, the computational complexity of this problem also depends on the fractional power β. The available numerical tests in the literature mostly concern the cases $\beta \in \{0.25, 0.5, 0.75\}$. In this article, we present numerical tests only for the most complicated case $\beta = 0.75$. Here we restrict to the analysis of 1D domain decomposition and the y coordinate is divided into M/p size blocs and distributed among p processes. From a scalability analysis it is known that for the larger problems and larger number of processors the 2D and 3D partitionings are more efficient decomposition strategies and this topic will be investigated in a separate paper.

Parallel performance results are presented in Table 1. The total wall time T_p is given in seconds. Here $p = n_d \times n_c$ is the number of used parallel processes computing with n_d nodes and n_c cores per node. In Table 1, we present the obtained values of parallel algorithmic speed-up $S_p = T_1/T_p$ and efficiency $E_p = S_p/p$.

p	1=1x1	2=1x2	4=1x4	8=1x8
T_p	1020	575.6	308.6	170.4
S_p	1	1.77	3.31	5.99
E_p	1	0.89	0.83	0.75
p	16=1x16	32=2x16	32=8x4	48=3x16
T_p	127.4	94.3	75.6	158.8
S_p	8.01	10.82	13.50	6.43
E_p	0.50	0.34	0.42	0.13

Table 1: The total wall time T_p, speed-up S_p and efficiency E_p solving problem (3) with $N_{x_1} = N_{x_2} = 1000$, $M = 250$, $\beta = 0.75$.

The obtained speed-up and efficiency values are not very good. The efficiency of the parallel algorithm is much better when a weak scalability analysis is done and the size of the discrete problem is increased proportionally to the increased number of processes. However, the presented results of strong scalability analysis show potential drawbacks of the first approach for the parallel solution with a larger number of processors.

IV.2 Integral evaluation problem

Using the second approach described in Section III.2, the non-local fractional diffusion problem (1) is transformed into a computation of two integrals (5). Each term in both sums of numerical approximation (6) can be computed independently, what is very convenient for the parallelization.

In our second parallel solver, we employ the well-known Master-Slave parallel model [10, 11]. Master process generates and distributes tasks (a block of consecutive y_j values) between the slave processes. For each received y_j value a slave process solves the local elliptic problem $(I_h + y_j^2 L_h)^{-1} f$ or $(y_j^2 I_h + L_h)^{-1} f$ in domain Ω.

Differently from the usual Master-Slave model, in our solver, slave processes do not return to the master results of each task immediately after its solution. The slave processes accumulate the obtained results - compute partial sums of the solution u for each mesh point. These big data vectors of the size $N_{x_1} \times N_{x_2}$ are sent only once, after the solution of the last task. The problem solution u is collected from the partial sums at the master process by MPI reduction operation [12].

To test the parallel performance of the developed algorithm, we have considered the problem (5) in the 2D unit square domain Ω using the discrete mesh of the size $N_{x_1} = N_{x_2} = 1000$ and $M = 3000$ in (6). A single task was defined as a block of 10 consecutive y_j values. For the local elliptic problems the tolerance of multigrid solver was set to 10^{-6}. The fractional power β was set to 0.75.

Parallel performance results of our second parallel solver are presented in Table 2. The total wall time $T_{s,n_d \times n_c}$ is given in seconds. Here $p = n_d \times n_c$ is the total number of used parallel processes computing with n_d nodes and n_c cores per node, $s = p - 1$ is the number of slave processes, which are solving computational tasks. In Table 2, we also present the obtained values of parallel algorithmic speed-up $S_s = T_1/T_{s,n_d \times n_c}$ and efficiency $E_s = S_s/s$.

	1, 1x2	2, 1x3	4, 1x5	8, 1x9	15, 1x16
T_s	11862	6192	3098	1605	1047
S_s	1	1.92	3.83	7.39	11.33
E_s	1	0.96	0.96	0.92	0.76
	31, 2x16	47, 3x16	63, 4x16	127, 8x16	159, 10x16
T_s	521.5	354.0	268.0	140.2	113.6
S_s	22.75	33.51	44.26	84.6	104.4
E_s	0.73	0.71	0.70	0.67	0.66

Table 2: The total wall time $T_{s,n_d\times n_c}$, speed-up S_s and efficiency E_s solving problem (5) with $N_{x_1} = N_{x_2} = 1000$, $M = 3000$, block size - 10, $\beta = 0.25$.

A slight degradation of the performance of our second parallel solver is caused by the load imbalance of the slave processes. The computational complexity of the local elliptic problems is different for the different y_j values. The number of tasks assigned to the single slave process is decreasing as the number of processes increases. This causes an increasing influence of the load imbalance on the total solution time.

The reduction of the single task (i.e. y_j block size) should reduce this drawback. However, this will cause an increasing communication between the master and slave processes. At some point, this can cause an idling of slave processes, waiting for the tasks from busy master.

IV.3 Discrete pseudo-parabolic problem

Using the third approach described in Section III.3, the non-local fractional diffusion problem (1) is transformed into another well-studied case of pseudo-parabolic PDE problem (7).

The constructed finite volume scheme (8) implies that our third numerical algorithm will advance in pseudo-time solving one system of linear equations at each of M iterations. In case, when the problem domain Ω is two-dimensional, the linear system will have 5 point stencil matrix of size $N = N_{x_1} \times N_{x_2}$.

One can easily see the similarities and differences with the first approach. One of the important practical implications is the significantly smaller amount of memory required to fit the system matrix, solution, and other data.

Again, a standard domain decomposition method is used for the parallel solution of pseudo-parabolic PDE problem. The discrete mesh of problem domain Ω and its associated fields are partitioned into subdomains, which are allocated to different processes. As in the previous tests, a simple one-dimensional block partitioning is used.

To test the parallel performance of the developed algorithm, we have considered the problem (7) in the 2D unit square domain Ω using the discrete mesh of the size $N_{x_1} = N_{x_2} = 1000$ and $M = 1000$. The tolerance of AGMG multigrid solver was set to 10^{-6} in all tests. Obviously, the computational complexity of problem (7) also depends on the fractional power β and parameter δ. In this case, we have performed numerical tests for $\beta = 0.25$ and $\delta = 10$.

Parallel performance results are presented in Table 3. The total wall time T_p is given in seconds. Here $p = n_d \times n_c$ is the number of used parallel processes computing with n_d nodes and n_c cores per node. In Table 3, we present the obtained values of parallel algorithmic speed-up $S_p = T_1/T_p$ and efficiency $E_p = S_p/p$.

p	1=1x1	2=1x2	4=1x4	8=1x8
T_p	2481.1	1562.7	813.6	421.7
S_p	1	1.59	3.05	5.88
E_p	1	0.79	0.76	0.74
p	16=1x16	32=2x16	32=8x4	48=3x16
T_p	320.9	376.6	345.3	610.3
S_p	7.73	6.59	7.18	4.07
E_p	0.48	0.21	0.22	0.08

Table 3: The total wall time T_p, speed-up S_p and efficiency E_p solving problem (7) with $N_{x_1} = N_{x_2} = 1000$, $M = 1000$, $\beta = 0.25$, $\delta = 10$.

Again, as it was with the first solver, the obtained speed-up and efficiency values are not very good. Since the size of 2D problem is even smaller, in this case the parallel scalability of AGMG multigrid solver is even

more critical, than in the case of the first solver.

V. Conclusions

Three different parallel numerical algorithms were developed for fractional diffusion problems. All of them rely on transformations of the original non-local problem to well-known local PDE problems.

The advantage of this approach is that due to the common use of these PDEs models their numerical solution methods are well developed. The software packages for their numerical solution (including parallel) are subject to a long-time development and permanent improvements.

The first and third algorithms strongly depend on the parallel scalability of the available multigrid solvers. The third algorithm has the significantly smaller demand on the amount of the required memory compared to the first one.

The performance results of second parallel algorithm are very promising. The issue of load balancing needs a special attention and further research. Possibility of employing a multilevel parallelism makes this approach even more attractive.

The weak scalability of these parallel algorithms will be studied in a following paper.

Acknowledgment

The work presented in this paper has been partially supported by EU under the COST programme Action IC1305, 'Network for Sustainable Ultrascale Computing (NESUS)'. This research was also funded by a grant (No. MIP-074/2015) from the Research Council of Lithuania.

Computations were performed on resources at the High Performance Computing Centre "HPC Sauletekis" in Vilnius University Faculty of Physics.

References

[1] S. Harizanov, S. Margenov, P. Marinov and Y. Vutov, "Volume constrained 2-phase segmentation method utilizing a linear system solver based on the best uniform polynomial approximation of $x^{1/2}$ ", *Journal of Computational and Applied Mathematics*, vol. 310, no. 1, 115–28, 2017.

[2] R. H. Nochetto, E. Otarola, and A. J. Salgado, "A PDE approach to fractional diffusion in general domains: a priori error analysis", *Foundations of Computational mathematics*, vol. 15, no. 3, pp. 733-791, 2015.

[3] R. H. Nochetto, E. Otarola, and A. J. Salgado, "A PDE approach to numerical fractional diffusion", in *Proceedings of the 8th ICIAM*, Beijing, China, 2015, pp. 211-236.

[4] A. Bonitto and J. E. Pasciak, "Numerical approximation of fractional powers of elliptic operators", *Math. Comp.*, vol. 84, pp. 2083-2110, 2015.

[5] P. N. Vabishchevich, "Numerical solving unsteady space-fractional problems with the square root of an elliptic operator", *Math. Model. and Anal.*, vol. 21, no.2, pp. 220-238, 2016.

[6] R. Čiegis and N. Tumanova, "Stability analysis of finite difference schemes for pseudo-parabolic problems with non-local boundary conditions", *Math. Model. and Anal.*, vol. 19, no.2, pp. 285-297, 2014.

[7] A. Quarteroni and A. Valli, *A Domain Decomposition Methods for Partial Differential Equations*, Oxford Science Publications, 1999.

[8] Y. Notay, "AGMG software and documentation", `http://homepages.ulb.ac.be/~ynotay/AGMG`. Last access: September 2016.

[9] A. Napov and Y. Notay, "An algebraic multigrid method with guaranteed convergence rate", *SIAM J. Sci. Comput.*, vol. 34, pp. 1079-1109, 2012.

[10] V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to Parallel Computing: Design and Analysis of Algorithms*, Benjamin/Cummings, 1994.

[11] R. Čiegis, V. Starikovičius, N. Tumanova, and M. Ragulskis, "Application of distributed parallel computing for dynamic visual cryptography", *Journal of Supercomputing*, doi:10.1007/s11227-016-1733-8.

[12] MPI Forum, "MPI: A Message Passing Interface Standard", `www.mpi-forum.org`, 1995.

List of Authors