Proceedings of the Second International Workshop on Sustainable Ultrascale Computing Systems (NESUS 2015)
Krakow, Poland

Jesus Carretero, Javier Garcia Blas
Roman Wyrzykowski, Emmanuel Jeannot.
(Editors)

September 10-11, 2015

Distributed Parallel Computing for Visual Cryptography Algorithms

Raimondas Čiegis†, Vadimas Starikovičius†, Natalija Tumanova†,
Minvydas Ragulskis*, Rita Palivonaitė*

† Vilnius Gediminas Technical University, Lithuania, * Kaunas Technological University, Lithuania
rc@vgtu.lt

Abstract

The recent activities to construct exascale and ultrascale distributed computational systems are opening a possibility to apply parallel and distributed computing techniques for applied problems which previously were considered as not solvable with the standard computational resources. In this paper we consider one global optimization problem where a set of feasible solutions is discrete and very large. There is no possibility to apply some apriori estimation techniques to exclude an essential part of these elements from the computational analysis, e.g. applying branch and bound type methods. Thus a full search is required in order to solve such global optimization problems. The considered problem describes visual cryptography algorithms. The main goal is to find optimal perfect gratings, which can guarantee high quality and security of the visual cryptography method. The full search parallel algorithm is based on master-slave paradigm. We present a library of C++ templates that allow the developer to implement parallel master-slave algorithms for his application without any parallel programming and knowledge of parallel programming API. These templates automatically give parallel solvers tailored for clusters of computers using MPI API and distributed computing applications using BOINC API. Results of some computational experiments are presented.

Keywords Visual Cryptography, Parallel Algorithm, BOINC, Parallel Templates

I. Introduction

The recent activities to construct exascale and ultrascale distributed computational systems are opening a possibility to apply parallel and distributed computing techniques for applied problems which previously were considered as not solvable with standard computational resources. In this paper we consider a global optimization problem, where a set of feasible solutions is discrete and very large. There is no possibility to apply some apriori estimation techniques to exclude an essential part of these elements from the computational analysis, e.g. applying branch and bound type methods [1]. Thus a full search is required in order to solve such global optimization problems.

The given problem describes visual cryptography algorithms [2]. The main goal of this paper is to find optimal perfect gratings, which can guarantee high quality and security of the visual cryptography method. The full search parallel algorithm is based on master-slave paradigm [3]. We present a library of C++ templates that allow the developer to implement parallel master-slave algorithms for his application without any parallel programming and knowledge of parallel programming API. These templates automatically give parallel solvers tailored for clusters of computers using MPI API and distributed computing applications using BOINC API [4]. For application of such MPI templates see [5].

The rest of this paper is organized as follows. In Section II, the discrete global optimization problem is formulated. The optimality criterion for finding the optimal perfect grating function is defined and the set of feasible solutions is described. The paral-

lel master-slave type algorithm is presented in Section III. Here we also describe a genetic evolutionary algorithm. Heuristics as an alternative for full search algorithms are recommended for such type of deterministic global optimization problems. Our aim is to investigate the efficiency of such algorithms in the case when the set of feasible solutions is described by quite complicated non-local requirements. In Section IV, templates for master-slave type algorithms are described. They allow developers to implement parallel master-slave algorithms for their applications without any parallel programming on clusters of computers using MPI and distributed computing systems using BOINC technology. Results of computational experiments are given in Section V. They illustrate the theoretical scalability results. Some final conclusions are presented in Section VI.

II. Problem Formulation

Here we define the most important details on advanced dynamic visual cryptography algorithms. The image hiding method is based on time-averaging moire gratings [6]. The method generates only one picture, which is used as a plaintext. The secret image can be seen by the human visual system only when the original encoded image is oscillated in a predefined direction at a strictly defined amplitude.

Function $F(x)$ defines a greyscale grating if the following requirements are satisfied

(i) The grating is a periodic function $F(x+\lambda) = F(x)$, here λ is the pitch of grating, and $0 \le F(x) \le 1$;

(ii) m-pixels n-level greyscale function $F_{mn}(x)$ is defined as:

$$F_{mn}(x) = \frac{y_k}{n}, \quad \frac{(k-1)\lambda}{m} \le x \le \frac{k\lambda}{m}, \quad k = 1, \dots, m,$$

where $k = 1, \dots, m$, $0 \le y_k \le n$.

We will consider a subset P of perfect greyscale step functions, they satisfy the following additional requirements:

(1) The grating spans through the whole greyscale interval

$$\min y_k = 0, \quad \max y_k = n.$$

(2) The average greyscale level in a pitch of the grating equals

$$\gamma := \frac{1}{m}\sum_{k=1}^{m} y_k = \frac{n}{2}.$$

(3) The "norm" of the greyscale grating function must be at least equal to the half of the norm of the harmonic grating

$$\|F\| \ge \|\widetilde{F}\| = \frac{1}{2\pi}, \quad \|F\| := \frac{1}{\lambda}\int_0^{\lambda} \left| F(x) - \frac{1}{2} \right| dx.$$

(4) The pitch of the grating λ must be strongly identifiable, the main peak of the discrete Fourier amplitude must be at least two times larger compared to all other Fourier modes

$$\sqrt{a_1^2 + b_1^2} \ge 2\sqrt{a_j^2 + b_j^2}, \quad j = 2, 3, \dots, m-1,$$

where the function F is expanded into the Fourier truncated series

$$F(x) = \frac{a_0}{2} + \sum_{j=1}^{m-1} \left(a_j \cos\frac{2\pi k x}{\lambda} + b_j \sin\frac{2\pi k x}{\lambda} \right).$$

Now we formulate the optimality criterion for finding the optimal perfect grating function

$$\delta(F_{mn}^0) = \max_{F_{mn} \in P} \min_{s \in S_1} \left(\sigma\big(H_s(F_{mn}, \widetilde{\xi}_s)\big) \right), \qquad (1)$$

where the standard deviation of a grayscale step grating function oscillated harmonically is given by (s is the oscillation amplitude):

$$\sigma\big(H_s(F_{mn}, \widetilde{\xi}_s)\big) = \frac{\sqrt{2}}{2}\sqrt{\sum_{j=1}^{m-1} (a_j^2 + b_j^2) J_0^2\,(2\pi j s/\lambda)},$$

where J_0 is the Bessel function of the first type.

III. Parallel Algorithm

The determination of the optimal perfect grating (1) requires to test a full set D of gratings in two steps. For each given grating the following algorithm is applied:

1. First, the testing of grating is done to check if all conditions of perfect gratings are satisfied.

2. Second, for a perfect grating the value of the standard deviation of the grating function is computed and the optimal value is updated.

This algorithm is fully parallel and it can be implemented by using the master and slaves paradigm [5]. Thus it is well suited for application of distributed computing technologies, including BOINC technology.

The complexity of the full search algorithm is of order

$$W = \mathcal{O}(m^2 n^m).$$

Here the factor m^2 arises due to application of simple Fourier summing algorithm instead of FFT algorithm. For small values of m this approach is more robust and flexible.

For industrial applications gratings with $12 \leq m \leq 25$ and $15 \leq n \leq 127$ are considered. In order to reduce the size of set D two specific modifications are applied.

1. Due to the first condition of the perfect gratings we can fix $y_m = 0$.

2. The periodicity condition and the mirror transformation are applied to exclude the gratings, which were tested in earlier stages of the full search algorithm.

We note, that such modifications still not change the asymptotic of the complexity of the full search algorithm.

The presented parallel full search algorithm requires very big computational resources. As an alternative heuristic methods can be considered. A natural selection is to use genetic evolutionary algorithms. We have applied a modification of the standard genetic method [7].

- Every chromosome represents one period of a grayscale function $F_{mn}(x)$. The initial population comprises of N randomly generated chromosomes (perfect gratings) with values of genes uniformly distributed over the interval $[0, n-1]$. We note, that only perfect gratings are included into the population.

- The crossover between two chromosomes is done by using the random roullete method. The chance that the chromosome will be selected to the mating is proportional to its fitness value. The main difficulty of this step is that after crossover between two perfect gratings in most cases we obtain non-perfect new grating. Thus this step is continued while the specified number M of new chromosomes are obtained. It is allowed to include into the new population more than one copy of the same chromosome.

- A mutation procedure is used with a slight modification that if the value of one gene is reduced by δ_k, then the value of the other randomly selected gene is increased by the same amount. Again, only perfect new chromosomes are included into the updated population.

In computational experiments we have tested the quality of solutions obtained by using this heuristic based on the genetic evolutionary algorithm.

IV. MPI and BOINC Templates

We obtain a parallel solver for the considered problem using our C++ templates for distributed computing applications. These templates were designed for easy and quick development of parallel applications based on master-slave parallelization paradigm [3]. In master-slave parallel algorithm, master process reads the problem input, generates and distributes jobs to the slave processes. Slave processes receive jobs from the master, solve them and return back the obtained results. Finally, master process receives the results from the slaves and generates a new set of jobs if necessary.

Our C++ templates allow the developer to implement the parallel master-slave algorithm for his application without any parallel programming and knowledge of parallel programing API. The developer needs only to implement the application-specific parts of the code for the reading of the problem input, consecutive generation of single jobs, solving of the single job, processing or merging of obtained results. These application-specific tasks need to be implemented and placed in appropriate virtual functions of our C++

templates. The workflow of the whole master-slave algorithm (including communication between the master and slave processes) is provided by the basic classes of our C++ templates.

Let us now formulate the special features of our C++ templates for distributed computing applications:

- The templates are built as a hierarchy of C++ classes. Basic classes implement the basic functionality of master-slave algorithm and specify the pure virtual functions, which need to be implemented in descendant classes to obtain application-specific parallel solvers.
- Input parameters and results of the job are exchanged between the master and slave using input and output files.
- Design of the templates allows to build a parallel application using MPI API [8] or distributed computing application using BOINC API [4] applying the same C/C++ code with implementation of application-specific tasks.

The usage of technology based on input and output files is not as efficient as a direct message passing between processes. However, the performance overhead is negligible for the coarse grained jobs. This is it the case for our problem. In turn, such an approach significantly simplifies the template and allows the communication of input and output data between the master and slaves without application-specific parallel programming.

Such an approach also allows the implementation of distributed computing applications. Currently, our programing tool allows easy and quick development of distributed application for volunteer computing project based on the Berkeley Open Infrastructure for Network Computing (BOINC) [4], which is the most popular middleware for volunteer computing. Using our C++ templates, application for BOINC project can be developed without any knowledge of BOINC API. Moreover, MPI version of application solver is very useful in testing and debugging implementations of application-specific tasks.

Application-specific tasks are separated and implemented in different classes:

- *WorkGenerator* class. It reads the problem input in the constructor, generates and writes to the file input for the next job by calling application-specific function *GenerateInputForNewJob*(FILE *jobInputFile), which must be provided by the application developer.
- *ClientApplication* class. It reads the input file, solves the job, and writes the results to output file by calling application-specific function *SolveSingleJob*(const char* inputFileName, const char* outputFileName), which must be provided by the application developer.
- *ResultsAssimilator* class. It is processing results of the single job and merging them with previous results by calling application-specific function *AssimilateResults*(FILE *jobResultsFile), which must be provided by the application developer.

For our problem we don't need it, but for BOINC project application, developer needs also to provide a separate class for validation of the obtained results.

V. Computational Experiments

Parallel numerical tests were performed on the computer cluster "Vilkas" at the Laboratory of Parallel Computing of Vilnius Gediminas technical university. We have used up to eight nodes with Intel® Core™ i7-860 processors with 4 cores (2.80 GHz) and 4 GB of RAM per node. computational nodes are interconnected via Gigabit Smart Switch.

V.1 Parallel search algorithm

First, we have solved a small benchmark problem in order to show a very good scalability of such type of parallel algorithms. They can be implemented efficiently on very large distributed heterogeneous systems, including BOINC technology.

We have solved the optimization problem for $m = 10$, $n = 10$. In table 1, we present the total wall time $T_{s,p\times c}$ in seconds, when parallel computations are done on a cluster with p nodes and c cores per node, and s slaves have solved computational tasks.

The master is responsible for generation and distribution of job set and accumulation of results from slaves, a separate core is used to run this part of the parallel algorithm. Also, we present the values of parallel algorithmic speed-up

$$S_s = \frac{T_{s,p\times c}}{s}.$$

	1, 2x1	2, 3x1	3, 4x1	7, 2x4	11, 3x4
$T_{s,p\times c}$	478.0	242.2	160.2	78.2	50.1
S_s	1	1.97	2.98	6.11	9.54

Table 1: The total wall time $T_{s,p\times c}$ and speed-up S_s values for solving the grading optimization problem with $m = 10$, $n = 10$.

The degradation of the efficiency of the parallel algorithm for $s = 7$ and $s = 11$ slaves is explained by the well-known fact, that in the case of more cores per node the shared-memory structure becomes a bottleneck when too many cores try to access the global memory of a node simultaneously [9]. This conclusion is confirmed also by results of more computational experiments with different configurations of nodes and clusters:

$$T_{2,1\times 3} = 275.5, \quad T_{3,2\times 2} = 166.6, \quad T_{3,1\times 4} = 183.0.$$

The presented estimate of the complexity of the parallel search algorithm gives quite accurate estimate from above for the total computation time. For example, using results of previous computational experiments we get prediction that a problem with $m = 11$, $n = 12$ on 4×4 cluster will be solved in $T = 1195$ seconds. The result of computational experiments is $T_{15,4\times 4} = 939$ seconds. Again, we can note that the scalability of the parallel search algorithm is very good, the same problem is solved on 5×4 cluster in $T_{15,4\times 4} = 746$ seconds, this CPU time is very close to the prediction from the scalability analysis.

V.2 Genetic search algorithm

The full search algorithm requires very big computational resources and leads to a big challenge even for ultrascale distributed computational systems. Thus alternatives based on heuristic global optimization algorithms also should be investigated. Next we present results of computational experiments for the heuristic search algorithm which is based on genetic evolutionary algorithm.

In table 2, we present the standard deviation values for optimal gratings and gratings computed by using the genetic algorithm.

m	n	δ_{optim}	$\delta_{genetic}$
8	13	0.06178	0.06178
9	13	0.06310	0.06267
10	13	0.05984	0.05717
11	13	0.06162	0.05808

Table 2: The standard deviation values for optimal gratings and gratings computed by using the genetic algorithm.

The presented results of computational experiments show that the classical genetic algorithm is not efficient for this type of problems. Such a behaviour of the given heuristic is connected to the fact that for perfect gratings the mutation of two high-quality gratings mostly will not produce a new perfect grating. But exactly this step is most important for obtaining efficient genetic algorithms for solving discrete global optimization problems.

VI. Conclusions

In this paper we have described a library of templates for implementation of parallel master-slave type algorithms. These C++ templates allow to build a parallel solver automatically from the sequential version of the algorithm. The parallel solvers for clusters using MPI API or distributed computing applications using BOINC API are generated using the same C/C++ code. Only application-specific tasks must be provided by users. These templates are used to generate a parallel solver for applied problem of visual cryptography. The provided results of computational experiments have confirmed theoretical scalability estimates of the parallel algorithm.

The complexity of the given discrete global op-

timization is very big even for modern distributed computational systems. Thus as an alternative some heuristics can be considered. Results of application of classical genetic heuristic algorithms are showing that such standard algorithms are not efficient for this type of problems. In the future paper we will investigate hybrid genetic algorithm. In these memetic algorithms the approximations obtained by genetic method are also subject to local improvement phases. For such local optimization the modifications of the full-search algorithm described above can be used.

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)'.

References

[1] R. Horst, P.M. Pardalos and N.V. Thoai, *Introduction to Global Optimization, Second Edition*. Kluwer Academic Publishers, 2000.

[2] P.S. Revenkar, A. Anjum and W.Z. Gandhare. "Survey of visual cryptography schemes," *Intern. Journal of Security and Its Applications*, vol. 4, no. 2, pp. 56-70, 2010.

[3] V. Kumar, A. Grama, A. Gupta and G. Karypis, *Introduction to Parallel Computing: Design and Analysis of Algorithms*. Benjamin/Cummings, Redwood City, 1994.

[4] D. P. Anderson, "Boinc: a system for public resource computing and storage," in *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, 2004, pp. 1-7.

[5] M. Baravykaitė and R. Čiegis, "An implementation of a parallel generalized branch and bound template", *Mathematical Modelling and Analysis*, vol. 12, no. 3, pp. 277–289, 2007.

[6] M. Ragulskis and A. Aleksa, "Image hiding based on time-averaging moire", *Optics Communications*, vol. 282, no. 14, 2752-2759, 2009.

[7] D. Goldberg, *The Design of Innovation: Lessons from and for Competent Genetic Algorithms*. Norwell, MA: Kluwer Academic Publishers, 2002.

[8] Message Passing Interface Forum, "MPI: A Message Passing Interface Standard," www.mpi-forum.org, Version 1.1, 1995.

[9] N. Tumanova and R. Čiegis, "Parallel algorithms for parabolic problems on graphs", in *High-Performance Computing on Complex Environments*, Chapter 4, pp. 51-71, John Wiley & Sons, Inc, 2014.