PROYECTO FIN DE GRADO

GRADO EN INGENIERÍA INFORMÁTICA

Implementación, paralelización y evaluación de una aplicación de tomografía 3D basada en C++ e Intel TBB

Autor: Diego Calvo Picado

Tutores: José Daniel García

Francisco Javier García Blas

Colmenarejo, Septiembre de 2012

Universidad Carlos III de Madrid
www.uc3m.es

AGRADECIMIENTOS

Mis más sinceros agradecimientos a:

Mi tutor José Daniel García y a mi co-tutor Francisco Javier García Blas por haber estado tan atentos a mi trabajo y por haberme dado todo su apoyo y depositado su confianza en mí.

A todos los profesores por todos los conocimientos que me han proporcionado, tanto en el aspecto académico como en el personal.

A mi familia por todo el apoyo y el ánimo que me han dado durante toda la carrera.

A mis compañeros de clase por haber mostrado interés e ilusión por mi proyecto.

A mis amigos por aguantar la sarta de tecnicismo cada vez que hablaba del proyecto y de aguantar mi estrés.

Universidad Carlos III de Madrid
www.uc3m.es

RESUMEN

En el presente Proyecto de Fin de Grado se aborda la necesidad de paralelización y optimización tanto de la memoria como de los recursos disponibles para reducir al mínimo posible el tiempo de procesamiento en la ejecución de una aplicación de imagen médica.

En concreto, la aplicación trata la tomografía computarizada (TC) mediante el uso de un haz cónico. Es una tecnología sanitaria de basada en el uso de rayos X obteniendo imágenes detalladas del cuerpo a estudiar. En vez de obtener una única imagen como se extrae de cualquier radiografía convencional, la TC obtiene un amplio número de imágenes al ir rotando alrededor del cuerpo, extrayendo así imágenes de cada uno de los ángulos disponibles.

Posteriormente, la computadora recoge todas estas imágenes y las combina en un volumen final que representa la reconstrucción digital del cuerpo en 3D, permitiendo su volteo y giro para poder percibir el objeto desde diversas vistas.

Para lograr el máximo rendimiento se dispone de un código escrito en lenguaje C con las optimizaciones ya hechas y se quiere comprobar si el mismo código escrito en lenguaje C++ se consigue una disminución notoria del tiempo de ejecución con respecto al código en C.

Contenido

Tabla de ilustraciones

Tablas

1. INTRODUCCIÓN

En este capítulo inicial se expone la visión general de este Proyecto Fin de Grado, cuál ha sido la motivación para realizarlo, qué necesidad tiene, por qué puede resultar útil y por último qué objetivos pretenden alcanzarse.

1.1 Motivación

La tomografía axial computarizada de rayos X (CT) es uno de los procedimientos de diagnóstico y evaluación por imagen médica más utilizados. Conforme ha ido evolucionando la tecnología, los tiempos de adquisición han ido disminuyendo. Por otra parte, la evolución de los paneles detectores ha supuesto un aumento de la densidad de elementos detectores, resultando en una mayor cantidad de datos a procesar. A este aumento del volumen de datos se le añade el requisito de reconstrucciones más rápidas impuesto por los usos actuales del CT. En este sentido, la planificación y monitorización en radioterapia, la cirugía asistida por imagen y otras modalidades clínicas requieren una respuesta lo más rápida posible. Por contra, los desarrollos en la algorítmica de reconstrucción no han traído avances equivalentes en lo que a tiempos se refiere, convirtiéndose en una barrera para la ampliación del uso de esta tecnología. Esta problemática motiva la necesidad de buscar procedimientos de aceleración que se adecuen a la creciente complejidad y exigencias de la reconstrucción.

Este trabajo presenta una solución al problema citado en el marco del CT de geometría de haz cónico y trayectoria circular por aplicación del algoritmo de Feldkamp, Davis y Kress (FDK). Este algoritmo es la extensión de la retroproyección filtrada para geometría de haz cónico incorporando unos factores de corrección de la longitud de los rayos. Así, los dos componentes principales del algoritmo son las fases de filtrado y retroproyección. Por su mayor complejidad algorítmica, es la segunda fase la que consume la mayor parte del tiempo total de procesamiento.

Una posibilidad para acelerar la reconstrucción consiste en emplear algoritmos alternativos. Éstos se pueden clasificar en tres grupos: el primer grupo interpola una retícula polar en otra rectangular en el espacio de Fourier para poder hacer uso de la familia de transformadas rápidas de Fourier (FFT); el segundo grupo acelera la retroproyección mediante procesos recursivos de sumas parciales y tratando todas las proyecciones simultáneamente; el tercer grupo utiliza un enfoque de divide y vencerás, y divide la imagen reconstruida en imágenes menores, en algún caso utilizando para la división la transformada de Fourier de la imagen. Según sus autores, algunos de estos algoritmos llegan a multiplicar por 40 la velocidad de reconstrucción, aunque

se debate sobre la calidad de las reconstrucciones y su falta de generalidad al depender de las propiedades de la imagen.

Otro enfoque es la aplicación de técnicas de computación paralela. Para ello existen multitud de enfoques. Algunos de ellos son rígidos y de coste elevado, como el uso de circuitos integrados específicos de la aplicación (ASIC) o de dispositivos FPGA.

Aquí presenta una implementación basada en multi-CPU del algoritmo FDK, partiendo de la implementación en C, Mangoose. El algoritmo permite aprovechar características de multiprocesador usando la librería de Intel TBB y más concretamente la librería de *parallel_for* para la paralelización del problema, además de traducir el código a C++.

1.2 Objetivos

El objetivo base del proyecto es desarrollar e implementar una versión mejorada de Mangoose que permita realizar la misma reconstrucción que en su versión inicial pero a un tiempo mucho menor mediante la optimización en la utilización de recursos, se puede desglosar en:

- Traducir el código del lenguaje C al lenguaje C++, haciendo uso de estructuras de datos complejas.

- Emplear sistemas de paralelización para el reparto de la carga de trabajo entre múltiples hilos con independencia de la cantidad de procesadores que tenga la máquina, haciendo uso de la biblioteca de Intel TBB.

- Lograr el ratio de aceleración más elevado posible manteniendo siempre la integridad de los datos y garantizando que el volumen obtenido sea idéntico al de su versión inicial.

1.3 Estructura del documento

En este apartado se detallan los diferentes capítulos que contiene este documento y en qué consisten:

- Capítulo 1, Introducción: Aporta la idea general del proyecto, con la motivación para realizarlo y los objetivos fundamentales que pretenden alcanzarse
- Capítulo 2, Estado del Arte: Se explica una visión general de la tecnología utilizada para la realización del proyecto.
- Capítulo 3, Mangoose: Concepto inicial del reconstructor tomográfico a optimizar, las etapas que lo componen y su finalidad.
- Capítulo 4, Análisis de requisitos: Necesidades básicas que el sistema debe satisfacer para desempeñar correctamente las funciones.
- Capítulo 5, Optimizaciones: Muestra los pasos realizados y las mejoras efectuadas sobre la versión inicial de Mangoose para lograr un mejor rendimiento.
- Capítulo 6, Evaluación y Resultados: Expone las pruebas realizadas para obtener los ratios de los tiempos frente a la versión inicial junto a los impactos en el rendimiento por la implementación de las mismas y los diferentes grados de paralelismo.

- Capítulo 7, Entorno de pruebas: Detalla el software utilizado para realizar las pruebas y comprobar el resultado correcto de las reconstrucciones digitales junto con los casos de estudio probados.
- Capítulo 8, Conclusiones y líneas futuras: Breve resolución del proyecto, las ideas extraídas de él y posibles vías de expansión.
- Capítulo 9, Presupuesto: Coste del desempeño y ejecución del proyecto.
- Apéndice: Información adicional complementaria a Mangoose y a sus evaluaciones.
- Bibliografía.

1.4 Glosario de términos

Término	Descripción
CUDA	Herramientas para programar en la GPU utilizadas para el desarrollo del proyecto
FDK	Algoritmo de reconstrucción digital que obtiene su nombre de Feldkamp, Davis y Kress
API	Interfaz de Programación de Aplicaciones, funciones y procesos de un servicio.
Vóxel	Porción del volumen.
AIO	Entrada/Salida Asíncrona.
Ratio de aceleración	Número de veces más rápido frente al más lento, obtenido de dividir el tiempo inicial entre el actual.
GPU	Unidad de Procesamiento Gráfico.
TILE	Sub-matriz en la memoria compartida de la GPU para agilizar los cálculos.
Hounsfield	Unidad de medida de densidad ósea.
OpenMP	Herramientas de programación en paralelo con memoria compartida basada en hilos.
Kernel	Proceso ejecutado por múltiples hilos dentro de una GPGPU.
Mangoose	Nombre del reconstructor tomográfico implementado en el lenguaje C que se toma como referencia en este proyecto.
SDK	Software Developement Kit
Pixel	

Tabla 1 : Glosario de términos

2. ESTADO DEL ARTE

En este apartado se aportará una visión global de las diversas tecnologías usadas para el desarrollo de este Proyecto Fin de Grado, así como posibles alternativas. Para ello se describirán detalladamente las tecnologías actuales disponibles que permitan extraer información y utilidad para lograr la eficiencia del reconstructor.

Inicialmente se detalla las diferencias entre los dos lenguajes de programación: C y C++, explicando las diferencias que existen entre ellos y las razones para usar C++ en este trabajo.

En sistemas de paralelización en C++ destaca la librería de Intel TBB, ya que ofrece una gran cantidad de funciones con la virtud de abstraer al programador de la creación de los hilos necesarios y la propia paralelización del trabajo.

2.1 Lenguaje de programación C

C es un lenguaje de programación estructurada inicialmente desarrollado por Dennis Ritchie entre 1969 y 1973 en los laboratorios Bell [2]. Su diseño proporciona construcciones que se asignan eficientemente a las instrucciones de máquina típicas, y por lo tanto ha tenido un uso duradero en aplicaciones que anteriormente habían sido codificados en código ensamblador, en particular en software de sistema como en el sistema operativo Unix [3].

C es uno de los lenguajes más usado mundialmente de todos los tiempos [4] [5] y hay muy pocas arquitecturas de computadores para las cuales no exista un compilador de C.

Muchos lenguajes posteriores han heredado de manera directa o indirecta del lenguaje C, incluyendo: C# (C sharp), D, Go, Java, JavaScript, Limbo, LPC, Perl, PHP, Python y Unix Shell C. La influencia más importante de C ha sido sintáctica y tienden combinar la expresión reconocible y la declaración de la sintaxis de C con sistemas de tipos subyacentes y modelos de datos que pueden ser radicalmente diferentes. C++ empezó como un preprocesador de C y actualmente es casi un super conjunto de C [6].

Antes de que existiese un estándar oficial de C, muchos usuarios y desarrolladores se basaron en una especificación informal de un libro escrito por Ritchie y Brian Kernighan; a esta versión se refiere generalmente como K&R C. En 1989 el Instituto Nacional de Estandarización Americano publicó el estándar de C (generalmente llamado ANSI C ó C89). El año siguiente, la misma especificación fue aprobada por la Organización Internacional de Estandarización como

un estándar internacional (llamado generalmente C90). ISO publicó más tarde una extensión para el soporte de la internacionalización en 1995 y revisó el estándar (conocido como "C99") en 1999. La versión actual del estándar (conocido como C11) fue aprobado en diciembre de 2011.

```c
// necesario para utilizar printf()

# include <stdio.h>

int main(void)
{
        printf("Hola Mundo\n");
        return 0;
}
```

Ilustración 1: Ejemplo código C

2.1.1 Diseño

C es un lenguaje imperativo o procesal. Fue diseñado para ser compilado usando un compilador relativamente sencillo, para permitir acceso de bajo nivel a la memoria, para proporcionar construcciones del lenguaje que se pudieran asignar eficientemente a las instrucciones máquina y que requiriese un mínimo de soporte en tiempo de ejecución. C fue, por lo tanto, útil en aplicaciones que habían sido codificadas en lenguaje ensamblador, como en la programación de sistemas.

A pesar de sus capacidades de bajo nivel, el lenguaje fue diseñado para fomentar la programación multi-plataforma. Compatible con los estándares y portable, un programa escrito en C puede ser compilado para una gran variedad plataformas y sistemas operativos con muy pocos cambios en el código fuente. El lenguaje ha sido usado en una amplia gama de plataformas, desde micro controladores empotrados hasta super ordenadores.

2.1.2 Características

Como en la mayoría de los lenguajes en la tradición de los ALGOL (diminutivo de lenguaje algorítmico), C cuenta con comodidades para programación estructurada y permite un alcance variable lexical, mientras que un sistema de tipos estático previene muchas operaciones no deseadas. En C, todo código ejecutable esta contenido por subrutinas, llamadas funciones (aunque no en el sentido estricto de programación funcional). Los parámetros de las funciones siempre se pasan por valor. El paso por referencia es simulado en C por el paso explicito de valores puntero. El texto fuente de un programa en C es de formato libre, utilizando el punto y coma como terminación de una declaración y las llaves para agrupar bloques de instrucciones.

El lenguaje C también tiene las siguientes características específicas:

- Hay un número pequeño y fijo de palabras claves o reservadas, incluyendo las primitivas del conjunto de control de flujo: for, if, while, switch y do…while. Sólo hay

un espacio de nombres y los nombres de usuario no se distinguen de las palabras claves por ningún tipo de dato.

- Hay una gran cantidad de operadores aritméticos lógicos, como +, +=, ++, &, ~, etc.
- Los valores de retorno de las funciones pueden ser ignorados cuando no son necesarios.
- Todo dato tiene un tipo, pero se puede hacer conversiones implícitas, como por ejemplo, caracteres pueden ser usado con integers.
- La sintaxis de declaración imita el uso del contexto. C no tiene la palabra clave difine, en su lugar, toda sentencia que empiece con el nombre de un tipo se considera declaración. Tampoco hay una palabra reservada function, sin embargo, una función viene indicada por paréntesis y una lista de parámetros.
- Tipos definidos por el usuario y tipos compuestos son posibles.
    - Agregación de tipos de datos heterogéneos (struct) permitiendo que los datos sean accedidos, como por ejemplo en una asignación, como una unidad.
    - Los arrays son una noción secundaria, definida en términos de aritmética de punteros. A diferencia de las estructuras, los arrays no son objetos de primera clase y, por consiguiente, no pueden ser asignados o comparados usando los operadores simples integrados. No existe la palabra reservada array, ni en uso ni en definición, sin embargo, los corchetes son un indicador sintáctico de arrays.
    - Los tipos enumerados son posibles con la palabra reservada enum. No esta etiquetados y pueden ser fácilmente transformados a integers.
    - Los strings no es un tipo de dato distinto, pero convencionalmente se implementa como arrays de caracteres terminados en cero.
- El acceso de bajo nivel a la memoria del ordenador es posible convirtiendo las direcciones a tipo de taos punteros.
- Los procedimientos (subrutinas que no devuelven valores) son un caso especial de función, con un tipo de dato devuelto void.
- Las funciones no pueden ser definidas dentro del ámbito léxico de otras funciones.
- Funciones y datos punteros permiten ad hock polimorfismo en tiempo real.
- Un preprocesado realiza una macro de definición, añadir archivos al código fuente y compilación condicional.
- Hay una forma básica de modularidad: los ficheros pueden ser compilados separadamente y pegados juntos, con un control sobre las funciones y los objetos de datos son visibles para otros ficheros vía los atributos statik y extern.
- Funcionalidades complejas como E/S, manipulación de strings y funciones matemáticas se delegan constantemente a las rutinas de bibliotecas.

C no incluye algunas características nuevas que se encuentran en los nuevos y más modernos lenguajes de alto nivel, incluyendo la orientación a objetos y el recolector de basura.

2.1.3 Historia

Comienzos del desarrollo

El desarrollo inicial de C se produjo en los laboratorios de AT&T Bell entre 1969 y 1973 [1]; y según Ritchie, el periodo más creativo fue en el año 1972. Fue llamado C porque muchas de sus características derivan de un lenguaje anterior llamado B, que de acuerdo con Ken Thompson era una versión reducida del lenguaje de programación BCPL.

El origen de C esta estrechamente relacionado con el desarrollo del sistema operativo Unix, originalmente implementado en lenguaje ensamblador en un PDP-7 por Ritchie y Thompson, incorporando muchas ideas de sus compañeros. Eventualmente decidieron portar el sistema operativo al PDP-11. La incapacidad de B para tomar ventaja de lagunas características del PDP-11, sobre todo por el direccionamiento de byte, impulso al desarrollo de la versión temprana de C.

La versión original de sistema Unix para el PDP-11 estaba desarrollada en lenguaje ensamblador. En 1973, con la incorporación de los tipos struck, el lenguaje C se había convertido en lo suficientemente potente que la mayor parte del kernle de Unix se rescribió a C. Este fue el primer sistema operativo que implementaba un kernel en otro lenguaje que no fuese ensamblador.

K&R C

En 1978, Brian Kernighan y Dennis Ritchie publicaron la primera edición de El lenguaje de programación C [7]. Este libro, conocido por los programadores de C como K&R, sirvió durante muchos años como una especificación informal del lenguaje. La versión de C que describe se refiere a ella comúnmente como K&R C. La segunda edición del libro [8] cubre el estándar ANSI C.

K&R introducen varias características al lenguaje:

- Biblioteca estándar de E/S
- El tipo de dato long int
- El tipo de dato unsigned int
- Operadores de asignación compuesta de la forma =op(como =-) fueron cambiados a la forma op= para evitar ambigüedad semántica creada por construcciones del tipo i=-10, que podía ser interpretada como i =- 10 en vez de la posibilidad de que fuese i = -10.

Incluso después de la publicación del estándar de C en 1989, por muchos años K&R C todavía era considerado el "mínimo común denominador" para que los programadores de C se limitasen si se quería conseguir la máxima portabilidad, desde que muchos compiladores antiguos todavía se usaban y porque el código escrito cuidadosamente de K&R C era también seguía el estándar de C.

En versiones tempranas de C, solo las funciones que no devolvían un valor int necesitaba ser declarado explícitamente si se usaban antes de la definición de la función; una función usada sin previa declaración se asumía que el valor de retorno era de tipo int, si el valor era usado.

Dado que en K&R la declaración de las funciones no incluía ninguna información sobre los argumentos de la función, la funcionalidad de comprobación de tipos de los parámetros no se había realizado, aunque algunos compiladores lanzaban mensajes de avisos si una función local era llamada con un número erróneo de argumentos, o si múltiples llamadas a funciones externas usaban un número diferente de argumentos. Herramientas independientes como Unix´s lint fueron creadas, entre otras cosas, para poder comprobar la consistencia del uso e funciones a través de múltiples ficheros.

En los años que siguieron a la publicación de K&R C, varias características no oficiales fueron añadidas al lenguaje, soportadas por compiladores procedentes de AT&T y algún que otro vendedor más. Estas incluían:

- Funciones void
- Funciones que devolvían struct o union
- Asignaciones para el tipo de dato struct
- Tipos enumerados

El gran número de extensiones y el poco consenso en la biblioteca estándar, junto con la popularidad del lenguaje y el hecho de que ni si quiera el compilador Unix implementaba de manera precisa la especificación de K&T, llevaron a la necesidad de la estandarización.

ANSI C e ISO C

Durante el final de la década de 1970 y la década de 1980, las versiones de C se llevaron a una gran variedad de ordenadores centrales, miniordenadores y microrordenadores, incluyendo el IBM PC, ya que su popularidad comenzó a aumentar de manera significativa.

En 1983, el American National Standards Institute (ANSI) formó un comité, el X3J11, para establecer una especificación del estándar de C. X3J11 basa el estándar de C en la implementación de Unix; sin embargo, la porción no portable de la biblioteca de Unix C fue tratada por el grupo de trabajo IEEE 1003 convirtiéndose en la base del estándar de POSIX en 1988. En 1989, el estándar de C fue ratificado como ANSI X3. 159-1989 "Lenguaje de programación C". Esta versión del lenguaje es referida habitualmente como ANSI C, Estándar de C o algunas veces como C89.

En 1990, el estándar ANSI C (con cambios de formato) fue adoptado por International Organization for Standardization (ISO) como ISO/IEC 9899:1990, que algunas veces se refieren a el como C90. Por ende, los términos C89 y C90 se refieren al mismo lenguaje de programación.

ANSI, como otros organismos de estandarización nacionales, no se encarga de desarrollar el estándar de C independientemente, pero se remite al estándar internacional de C, mantenido por el grupo de trabajo ISO/IEC JTC1/SC22/WG14. La adopción nacional de una actualización del estándar internacional típicamente ocurre del año de la publicación de ISO.

Uno de los objetivos del proceso de la estandarización de C era producir un super conjunto de K&R C, incorporando muchas de las características no oficiales introducidas posteriormente. El comité de estandarización también incluyó características adicionales como los prototipos de funciones (tomada de C++), punteros void, soporte para conjuntos de caracteres internacionales y entornos locales y mejoras de pre procesado. Aunque la sintaxis para la declaración de parámetros se ha ampliado para incluir el estilo usa en C++, la interfaz de K&R siguió estando permitida, para la compatibilidad con el código fuente existente.

C89 es soportado por los actuales compiladores de C y la mayor parte del código que se escribe actualmente en C hoy en día está basado en el. Cualquier programa escrito siguiendo sólo el estándar de C y sin ninguna dependencia de hardware funcionará correctamente en cualquier plataforma acorde con la implementación de C, dentro de los límites de sus recursos. Sin estas

precauciones, los programas podrían compilar solo en ciertas plataformas o con un compilador particular, debido, por ejemplo, por el uso de bibliotecas no estándares, tales como las bibliotecas GUI o en la confianza en el compilador o atributos específicos de la plataforma, tales como el tamaño exacto de los tipos de datos.

En caso de que el código deba ser compatible tanto con el estándar como con K&R C, la macro _STDC_ puede ser usada para dividir el código en estándar y secciones K&R para prevenir el uso de características disponibles solo en el estándar de C en compiladores basados en K&R C.

C99

Después del proceso de estandarización ANSI/ISO, la especificación del lenguaje C se mantuvo relativamente estática durante algunos años. En 1995 se publicó la enmienda al estándar C de 1990, para corregir algunos detalles y añadir un soporte más extenso para los conjuntos de caracteres internacionales. El estándar fue fuertemente revisado a finales de la década de 1990, llevando a la publicación de ISO/IEC 9899:1999 en 1999, al cual es referido comúnmente como C99. Desde entonces ha sido corregido 3 veces por errores técnicos.

C99 introduce varias características nuevas, incluyendo las funciones en línea, nuevos tipos de datos (incluyendo long long int y un tipo complejo para representar los números complejos), matrices de longitud variable, soporte mejorado para IEEE 754 de números de coma flotante y soporte para comentarios de una línea empezando por //, al igual que en BCPL o C++. Muchas de estas mejoras ya habían sido implementadas como extensiones en varios compiladores de C.

C99 es compatible en su mayor parte con C90, pero es más estricta en algunos aspectos, en particular, una declaración que no tenga un tipo específico deja de asumirse implícitamente como int. El macro del estándar _STDX_VERSION_ se define con el valor 19901L para indicar que el soporte con C99 esta disponible. GCC y otros compiladores de C ahora soportan muchas de las nuevas características de C99.

C11

En 2007 se empezó a trabajar en otra revisión del estándar de C, informalmente llamado C1X hasta que se hizo oficial su publicación en 2011. El comité de estandarización de C adoptó las directrices para limitar la adopción de nuevas características que no habían sido probadas por implementaciones existentes.

El estándar C11 añade numerosas nuevas características a C y la biblioteca, incluyendo un tipo de macros genérica, estructuras anónimas, mejora en el soporte Unicode, operaciones atómicas, multi hilo, etc. Incluso hace que algunas porciones de la biblioteca C99 sean opcionales y mejora la compatibilidad con C++.

2.2 Lenguaje de programación C++

C++ es un lenguaje de programación estáticamente tipificado, de libre formato, multi paradigma, compilado y de propósito general. Es considerado un lenguaje de nivel medio, ya

que comprende características de un lenguaje de alto nivel y de un lenguaje de bajo nivel [11]. Desarrollado por Bjarne Stroustrup a partir de 1979 en los laboratorios de Bell, añadiendo características de la orientación a objetos, como las clases y otras mejoras en el lenguaje de programación C. Originalmente llamado C con clases, el lenguaje fue renombrado a C++ en 1983 [12] como un juego de palabras que implica al operador incremento.

C++ es uno de los lenguajes de programación más populares [4] [5] y es implementado en una gran variedad de hardware y sistemas operativos. Como compilador eficiente de código nativo, sus dominios de aplicación incluyen software de sistemas, aplicaciones software, controladores de dispositivos, software integrado, servidores de gran rendimiento y aplicaciones cliente, y software de entretenimiento como video juegos [13]. Varios grupos proporcionan compiladores de C++ tanto libre como propietario, incluyendo el proyecto GNU, Microsoft e Intel entre otros. C++ ha sido una gran influencia para muchos otro lenguajes de programación populares, más notable en C# [10] y Java. Otros lenguajes exitosos como Objective-C utilizan una sintaxis muy diferente aproximándose a añadir clases al lenguaje C.

C++ es usado también para el diseño de hardware, donde el diseño es inicialmente descrito en C++, luego se analiza y programado para crear un registro de transferencia a nivel hardware a través de síntesis de alto nivel [14].

El lenguaje empezó como mejoras para C, primero añadiendo clases, seguidamente funciones virtuales, sobrecarga de operadores, herencia múltiple, plantillas y manejo de excepciones entre otras características. Después de años de desarrollo, el lenguaje de programación C++fue ratificado con un estándar en 1998 como ISO/IEC 14882:1998. El estándar fue modificado en 2003, pasando a ser ISO/IEC 14882:2003. El actual estándar de C++ se extiende con nuevas mejoras fue corroborado y publicado por ISO en 2011 como ISO/IEC 14882:2011 (Informalmente conocido como C++11) [15].

```
class Punto
{
//por omisión los miembros son 'private' para que sólo se puedan modificar
desde la propia clase.
private:
        // Variable miembro privada
        int id;
protected:
        // Variables miembro protegidas
        int x;
        int y;
public:
        // Constructor
        Punto();
        // Destructor
        ~Punto();
        // Funciones miembro o métodos
        int ObtenerX();
        int ObtenerY();
};
```

Ilustración 2: Ejemplo código C++

2.2.1 C++0x

C++0x es una actualización del lenguaje C++ aprobada a finales del 2011 (ver [39]). La nueva norma contiene mejoras y ampliaciones que pasan a formar parte del lenguaje C++ desde el núcleo del lenguaje, la biblioteca estándar de C++ y las funciones lambda. De esta manera, el lenguaje de programación se adhiere a la tendencia actual hacia la paralelización de software, impulsado por el paso de aumentar cada vez más la velocidad del reloj en arquitecturas de procesadores multi-cores.

Una de las ventajas de esta actualización utilizada en la solución del trabajo son las funciones lambda que le permiten a los programadores explotar de forma explícita el paralelismo disponible convirtiéndose en parte del lenguaje C++ básico. Los threads también se añaden a la biblioteca estándar. Estas extensiones serán las instalaciones por defecto para los programadores que le permitirán obtener un rendimiento en paralelo de los procesadores multi-core y many-core.

2.2.2 Historia

Bjarne Stroustrup empezó trabajando con C con Clases en 1979 [12]. La idea de crear un nuevo lenguaje vino del propio Stroustrup por su experiencia programando para su tesis doctoral. Stroustrup observó que Simula tenía características que fueron muy útiles para el desarrollo de grandes proyectos software, pero el lenguaje era demasiado lento para un uso práctico, mientras que BCPL era muy rápido pero de bajo nivel para que fuese adecuado para desarrollos grandes de software. Stroustrup empezó a trabajar en los laboratorios AT&T Bell, tenía el problema para analizar el kernel de Unix con respecto a la computación distribuida. Recodando su experiencia con la tesis, Stroustrup se propuso mejorar C con características similares de Simula. C se eligió porque era de propósito general, rápido, portable y usado mundialmente. Además de C y Simula, otros lenguajes de programación le inspiraron, como ALGOL68, Ada, CLU y ML. Al principio, las clases, las clases derivadas, la fuerte comprobación de tipos y las características predeterminadas de argumentos fueron añadidas a C por Stroustrup al compilador de C. La primera implementación comercial de C++ fue publicada en 1985.

En 1983, el nombre del lenguaje fue cambiado de C con clases a C++ (++ es el operador de incremento en C). Se añadieron nuevas características incluyendo las funciones virtuales, sobrecarga de los nombre de funciones y operadores, referencias, constantes, control de memoria libre controlada por el usuario, mejorada la comprobación de tipos y el estilo de comentarios de una sola línea de BCPL con doble barra (//). En 1985 salió la primera edición de El lenguaje de programación C++, ofreciendo una importante referencia para el lenguaje, ya que en ese momento no había un estándar oficial. [16] La versión 2.0 de C++ llegó en 1989 y la edición actualizada de El lenguaje de programación C++ se publicó en 1991. [17] Nuevas características incluyendo herencia múltiple, clases abstractas, miembros estáticos en las funciones, miembros constantes en las funciones y miembros protegidos. En 1990 The Annotated C++ Reference Manual fue publicado. Este libro sirvió de base para la futura estandarización. Más tarde fueron añadidas nuevas características incluyendo plantillas, excepciones, espacios de nombres, nuevos cast y el tipo Boolean.

A medida que el lenguaje C++ iba evolucionando, así lo hacía también la biblioteca estándar. La primera incorporación a la biblioteca estándar de C++ fue la biblioteca stream I/O la cual proporciona facilidades para remplazar las funciones tradicionales C como printf y scanf. Más tarde, entre las novedades más significativas añadidas a la biblioteca estándar, fue la gran cantidad de bibliotecas de plantillas estándar.

Es posible programar tanto en orientación a objetos como en procesal en el mismo programa en C++. Esto ha causado cierta preocupación porque algunos programadores de C++ siguen codificando en procesal, pero tienen la impresión de estar programando orientado a objetos, simplemente porque están usando C++. A menudo es una mezcla de ambas. Esto causa mayores problemas cuando el código es revisado por otro desarrollador [19]. Por estas razones, C++ es a veces llamado un lenguaje hibrido [18].

C++ se sigue usando y es uno de los lenguajes de programación preferidos para desarrollar aplicaciones profesionales. [20]

Etimología

De acuerdo con Bjarne Stroustrup: "el nombre significa la naturaleza evolutiva de los cambios de C" [21]. Durante el periodo de desarrollo de C++, el lenguaje ha sido nombrado como nuevo C, más tarde como C con clases. El nombre final se atribuye a Rick Mascitti (mediados de 1983) y fue usado por primera vez a finales de 1983. Cuando preguntaron de manera informal a Mascitti en 1992 sobre el nombre e indicó que se lo dio con espirito de ser un juego de palabras. Se deriva del operador de C ++ (el cual incrementa el valor de la variable a la que acompaña) y a un convenio de nomenclatura común de utilizar + para indique que es un programa de ordenador mejorado.

Estandarización

En 1998, el comité de estandarización de C++ (el grupo de trabajo ISO/IEC JTC1/SC22/WG21) estandarizó C++ y publico el estándar internacional ISO/IEC14882:1998 [25] (informalmente conocido como C++98). Durante algunas años después de la publicación oficial del estándar, el comité procesó informes sobre defectos y publicó una correcta versión del estándar de C++, el ISO/IEC 14882:2003 [24], en 2003. En 2005, un informe técnico, llamado Reporte técnico de biblioteca, fue publicado. Aun que no es una parte del estándar oficial, especifica un número de extensiones de la biblioteca estándar, las cuales se esperan que sean incluidas en la siguiente versión de C++ [23].

La última revisión del estándar de C++, C++11 (conocida comúnmente como C++0x) fue aprobada por ISO/IEC en el 2011 [26]. Fue publicada como 14882:2011[22] [27].

Filosofía

En El diseño y la evolución de C++ (1994), Bjarne Stroustrup describe algunas reglas que él mismo uso para el diseño de C++:

- C++ esta diseñado para ser estáticamente tipificado y un lenguaje de propósito general tan eficiente y portable como C

- C++ esta diseñado para soportar múltiples estilos de programación (programación procesal, abstracción de datos, programación orientada a objetos y programación genérica)
- C++ esta diseñado para dar al programación poder de elección, aunque esto hace que sea posible que el programador elija incorrectamente.
- C++ esta diseñado para ser tan compatible como sea posible con C, por lo tanto, proporciona una transición suave desde C
- C++ evita características de plataformas especificas o no de propósito general
- C++ no cae en gastos generales para las funciones que no se utilizan
- C++ está diseñado para funcionar sin un sofisticado entorno de programación

Inside the C++ Object Model (Lippman, 1996) describe como los compiladores pueden convertir declaraciones de C++ en un diseño de memoria. Sin embargo, los autores de los compiladores tienen libertad de aplicar el estándar a su manera.

Estándar de la biblioteca

En 1998, el estándar ANSI/ISO C++consta de dos partes: el núcleo del lenguaje y la biblioteca estándar de C++, este último incluye la mayor parte de la biblioteca estándar de plantillas (STL) y una versión ligeramente modificada de la biblioteca estándar de C. Muchas de las bibliotecas de C++que existen no forman parte del estándar y, con especificaciones de enlace, incluso pueden ser escritas en otras lenguajes de programación como BASIC, C, Fortran o Pascal.

La biblioteca estándar de C++ incorpora la biblioteca estándar de C con algunas pequeñas modificaciones para optimizarla con el lenguaje C++. Otra gran parte de la biblioteca estándar de C++ esta basada en STL. Esta ofrece herramientas tan útiles como contenedores (como por ejemplo vector y list) iteradores para ofrecer un acceso parecido para los arrays y algoritmos para realizar operaciones como búsqueda o reordenación. Además permite arrays asociativos y multi conjuntos, todos ellos exportados con interfaces compatibles. Con lo cual, es posible, usando plantillas, codificar un algoritmo genérico que trabaje con cualquier contenedor o cualquier secuencia definida por un iterador. Como en C, las características de la biblioteca son accesibles usando la directiva para incluir un encabezado estándar #include. C++ proporciona 105 cabeceras estándar, de las cuales 27 están en desuso.

La biblioteca STL fue originariamente de terceros de HP y más tarde de SGI, antes de su incorporación al estándar de C++. El principal arquitecto detrás del STL es Alexander Stepanov, el cual experimentó con algoritmos genéricos y contendores durante muchos años. Cuando empezó con C++, encontró finalmente un lenguaje donde era posible crear algoritmos genéricos con un rendimiento incluso mejor que en la biblioteca estándar de C, gracias a las características de C++ como el uso de inlining en vez del uso de funciones puntero. El estándar no se refiere a la biblioteca estándar de C++ como STL, ya que esta es una parte de la biblioteca estándar, pero mucha gente sigue usando ese nombre para distinguirla del resto de bibliotecas.

La mayor parte de los compiladores de C++ ofrecen una implementación de la biblioteca estándar de C++, incluido el STL [28].

2.3 PROGRAMACIÓN PARALELA

La programación paralela representa el punto de inflexión en cómo los ingenieros de software escriben software. ro cesadores multicore se pueden encontrar hoy en los superordenadores, computadoras de escritorio y portátiles. En consecuencia, las aplicaciones tendrán que ponerse en paralelo para aprovechar al m a imo los procesadores multicore, lo que permitirá obtener ganancias de rendimiento que ofrecen los modelos de alto nivel de programación en paralelo. Hay varias alternativas para conseguir la paralelización aunque se descartan aquellas que se centren en lenguajes específicos como por ejemplo Cilk, MPI, Pthreads, este trabajo se centra en Intel TBB (Threading Building Blocks de Intel) y OpenMP (API de paralelización sobre memoria compartida).

2.4 INTEL TBB

Intel Threading Building Blocks es una biblioteca plantilla desarrollada por la Corporación Intel para la codificación de programas para sacar ventaja de los procesadores multi núcleo. La biblioteca consiste en estructuras de datos y algoritmos que permiten al programador evitar algunas de las complicaciones que aparecen al usar los paquetes nativos de hilos como el POSIX threads o Windows threads en los cuales cada hilo individual para la ejecución es creado, sincronizado y terminado de manera manual. En vez de eso, la biblioteca abstrae el acceso a los múltiples procesadores permitiendo que las operaciones sean tratadas como “tareas”, las cuales son asignadas a los núcleos de manera dinámica por el motor de tiempo en ejecución de la biblioteca, y automatizando el uso eficiente de la caché de las CPU. Un programa con TBB crea, sincroniza y destruye grafos de tareas dependientes de acuerdo con los algoritmos, es decir, uso de paradigmas de programación paralela de alto nivel. Las tareas que se ejecutan han de respetar las dependencias del grafo. El enfoque de las soluciones para la programación paralela de la familia TBB es el de desvincular la programación de los detalles de la máquina subyacente.

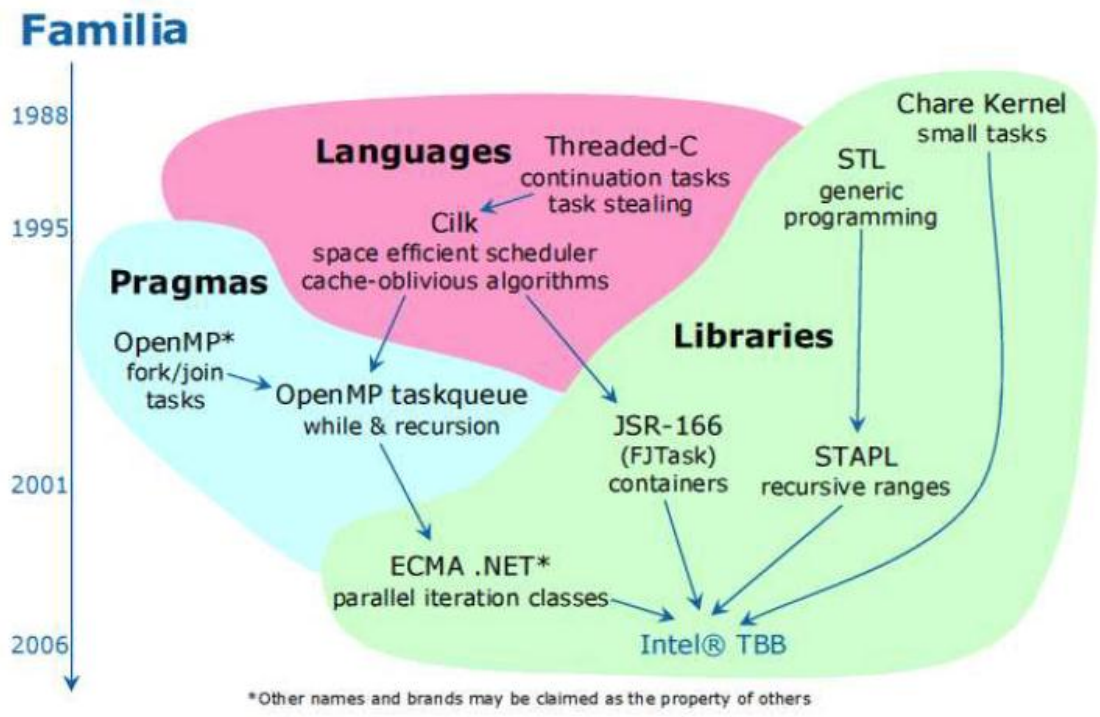


Ilustración 3: Evolución de las bibliotecas de paralelización

2.4.1 Implementación

La biblioteca TBB implementa el "Robo de tareas" para balancear la carga de trabajo de los núcleos de los procesadores para incrementar el uso de los núcleos y, por ende, la escalabilidad. El modelo de "Robo de tareas" de TBB es parecido al modelo aplicado en Cilk. Inicialmente, la carga de trabajo se reparte equitativamente entre todos los núcleos disponibles. Si uno de los núcleos completa su trabajo mientras que otros siguen teniendo una cantidad significativa de trabajo en sus colas, TBB reasigna parte del trabajo de los núcleos ocupados al núcleo libre. Esta capacidad dinámica separa al programador de la máquina, permitiendo que las aplicaciones que usan la biblioteca TBB puedan escalar para utilizar los núcleos de procesamiento disponibles sin cambios en el código fuente o en el archivo del programa ejecutable.

TBB, al igual que STL, usa plantillas ampliamente. Esto tiene una ventaja de baja sobrecarga de polimorfismo, desde que las plantillas son construidas en tiempo de compilación en los compiladores modernos de C++, pudiendo optimizar en gran medida.

Intel TBB esta disponible de forma comercial distribuido en forma de binario con soporte [29] y como código abierto tanto el código fuente como el binario [30].

2.4.2 Contenidos de la biblioteca

TBB es una colección de componentes para la programación paralela:

- Algoritmos básicos: parallel_for, parallel_reduce, parallel_scan

- Algoritmos avanzados: parallel_while, parallel_do, parallel_pipeline, parallel_sort
- Contenedores: concurrent_queue, concurrent_priority_queue, concurrent_vector, concurrent_hash_map
- Exclusión mutua: exclusión mutua , spin_mutex , queuing_mutex , spin_rw_mutex , queuing_rw_mutex , mutex recursivo
- Operaciones atómicas: fetch_and_add , fetch_and_increment , fetch_and_decrement , compare_and_swap , fetch_and_store
- Programador de tareas: acceso directo al control de creación y activación de tareas.

2.4.3 Historia

- La versión 1.0 fue publicada por Intel en 2006, un año después de la introducción del primer procesador de doble núcleo x86, el Pentuim D.
- La versión 1.1 fue liberada a principios de 2007. Esta versión introduce una nueva característica, el auto_partitioner el cual ofrece una alternativa automática para especificar el tamaño del grano para estimar la mejora granularidad para las tareas. Esta versión fue añadida al compilador de Intel de C++ 10.0.
- La versión 2.0 fue introducida a mediados de 2007. Esta nueva versión incluía la liberación del código fuente y la creación del proyecto de código abierto. TBB sigue estando disponible en su forma comercial (pero sin el código fuente) con soporte pero no tiene diferencias de funcionalidad con su versión de código abierto.
- La versión 2.1fue publicada a mediados de 2008. Esta versión incluía las características de afinidad de tareas a hilos, soporte a la cancelación, control de excepciones y envoltura portable de hilo.
- La versión 2.2 vio la luz a mediados de 2009 [29]. Esta versión añade soporte a la característica de funciones lambda en C++0x.
- La versión 3.0 fue publicada a principios de 2010. Incluye numerosas mejoras, entre ellas una nueva capacidad de sincronización de hilos: las variables condición y diseño de patrones manual [30].
- La versión 4.0 fue introducida a mediados de 2011. Incluía nuevas características, como colas de prioridad concurrente y conjunto concurrente desordenado entre otras [31].

2.4.4 Uso

Para poder usar la primitiva *parallel_for*, perteneciente a la biblioteca, es necesario usar funciones lambda. Estas funciones son, de manera simplificada, expresiones tal que su evaluación resulta en una función. Este tipo de funciones se puede guardar en una variable y puede ser llamada cuantas veces sea necesario. En el caso concreto de C++, no generan una función, sino un objeto de un tipo generado que sobrecarga el *operator ()* y se puede usar como una función. La sintaxis es la que sigue:

```
auto doble = [](int x) -> int { return 2 * x; };
```

Ilustración 4: Ejemplo función lambda

Los corchetes hacen de introductores a la expresión, seguidamente vienen los argumentos de la función entre los paréntesis; después una flecha y detrás el tipo de retorno; y para finalizar el

cuerpo de la función entre llaves. Al estar en mitad de una expresión, no hay que olvidar terminarla con *;*.

Al ser una función que sustituye a un bucle *for*, sólo se han modificado todos aquellos que conlleven un gran gasto de tiempo en ejecutarse, ya que la creación de hilos siempre conlleva a un gasto adicional que se traduce en tiempo, con lo que en los bucles de poco recorrido o de poca complejidad, no es necesario ponerlo.

2.5 OPENMP

OpenMP (Open MultiProcessing) es una API que soporta multi plataforma de memoria compartida con multiprocesador programando en C, C++ y Fortran, y esta soportado por la mayoría de las arquitecturas de los procesadores y sistemas operativos, incluyendo Solaris, Linux, Mac OS X y Windows. Consiste en una serie de directivas del compilador, rutinas de biblioteca y variables de entorno que influeyen en el comportamiento en tiempo de ejecución [32] [33] [34].

OpenMP usa un modelo portátil y escalable que otorga a los programadores una simple y flexible interfaz para el desarrollo en paralelo de aplicaciones para plataformas desde ordenadores de sobre mesa hasta supercomputadores [35].

2.5.1 Introducción

OpenMP es una implementación de múltiples hilos, un método de paralelización donde hay un hilo maestro (una serie de instrucciones que se ejecutan consecutivamente) y crea un número especifico de hilos esclavos y una tarea que se divide entre ellos. Entonces los hilos se ejecutan de manera concurrente, con un entorno de ejecución que coloca los hilos en los diferentes procesadores.

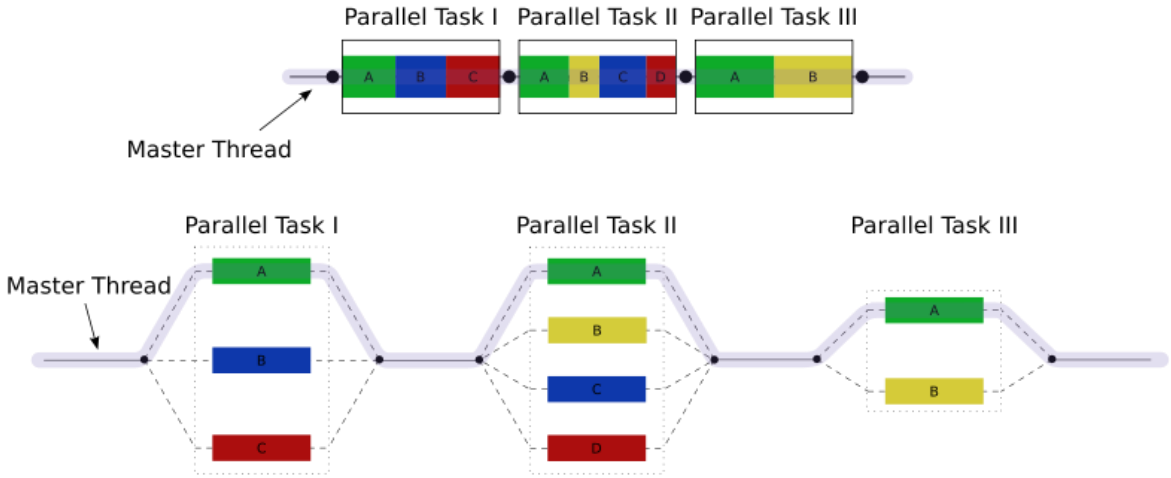


Ilustración 5: Funcionamiento de OpenMP

La sección de código que se desea ejecutar en paralelo es marcada con una directiva de preprocesado que causa que los hilos ejecuten dicha parte de código. Cada hilo tiene asociado

un *ID* que se puede obtener que cualquier momento usando una función (omp_get_thread_num ()). El id de l hilo es un número, y el hilo maestro tiene el id 0. Después de la ejecución de la parte paralelizada del código, los hilos se unen al hilo maestro, el cual continúa hasta el final del programa.

Por defecto, cada hilo ejecuta su sección de código a paralelizar de manera independiente. Se puede usar el constructor de trabajo compartido para dividir una tarea entre los hilos y de esta manera cada hilo ejecuta de manera local su parte de código. Tanto el paralelismo en procesos como el paralelismo en datos se pueden conseguir de esta manera en OpenMP.

El entrono de tiempo de ejecución coloca los hilos en los procesadores en función del uso, carga de la CPU y otros factores. El número de hilos puede ser asignado por el entrono de tiempo de ejecución basándose en las variables de entorno o en el código usando funciones. Las funciones de OpenMP están incluidas en el fichero de cabecera llamado *omp.h* en C y C++.

2.5.2 Historia

La OpenMP Architecture Review Board (ARB) publicó la primera especificación de la API de OpenMP para Fortran 1.0 en 1997. El año siguiente se publicó el estándar para C/C++. En el año 2000 se publicó la versión 2.0 con las especificaciones para Fortran y las especificaciones de la versión 2.0 de C/C++ en 2002. La versión 2.5 es una combinación de las especificaciones de C/C++/Fortran y fue publicada en 2005.

La versión 3.0, publicada en 2008, incluía en las nuevas características de la versión 3.0 el concepto de *task* y *task construct*. La última versión de OpenMP, la 3.1, fue publicada en 2011.

3. MANGOOSE

Mangoose es una estrategia propuesta para una reconstrucción multi-cama basada en FDK, con un kernel programado en lenguaje C y una interfaz gráfica de usuario implementada en IDL, que incluye módulos para los procesos de calibración y diferentes algoritmos de corrección. Estos módulos de reconstrucción están incluidos en la arquitectura software TAC. En el nivel más alto la aplicación principal del TAC, programada en IDL, incluye la interfaz de usuario y controla los módulos de adquisición, calibración y reconstrucción. La adquisición de datos es realizada por un procesador basado en Linux que controla el hardware del sistema TAC. La adquisición de datos es lanzada por la aplicación principal del TAC después de que el usuario seleccione los parámetros de adquisición, como la energía de los rayos X o el flujo, la posición del caso de estudio o el tamaño de punto de la proyección. El proceso de reconstrucción 3D se realiza a través de Mangoose.

Su función se divide en varias etapas, las más costosas en computación y en las que se ha puesto especial dedicación son la etapa de filtrado y retroproyección [36].

El reconstructor se compone de las siguientes fases:

- Etapa inicial: Período de tiempo destinado a la creación de contextos, reservas de memorias e inicialización de variables. Es un tiempo parcialmente fijo que puede ser reducido si existe un trabajo permanente que mantenga los dispositivos iniciados.

- Etapa de lectura: Los ficheros de origen con las proyecciones son leídos y almacenados en su correspondiente espacio de memoria reservada.

- Etapa de filtrado: Primera de las etapas de alto coste por su alto cómputo, durante el filtrado a las proyecciones leídas se les aplica unos pesos prefijados línea a línea. Una vez terminado el cálculo de todos los pesos se sobrescriben las proyecciones con los nuevos valores de la imagen.

- Etapa de retroproyección: Segunda etapa de alto coste, tras los filtros realizados en la anterior etapa, se toman las proyecciones filtradas y moviéndose sobre la rotación angular, se procesan dando lugar al que será el volumen, final o parcial dependiendo del caso, de la reconstrucción.

- Etapa de pegado: La generación del volumen reconstruido se hace mediante volúmenes parciales paralelizando el trabajo y reduciendo así el coste, es durante esta etapa que los volúmenes parciales son unidos para sumar un único volumen más grande y definitivo.

- Etapa de desviación: Afecta únicamente cuando el objeto de la reconstrucción es demasiado grande y debe ser dividido en trozos, de ahora en adelante serán denominados como “camas”. Debido a las imperfecciones mecánicas de la máquina reconstructora, las camas no tienen un casamiento perfecto y la desviación debe ser corregida para que el pegado de los volúmenes sea correcto. El tiempo invertido en esta etapa es el correspondiente a dicha corrección.

- Etapa de solapamiento: Cuando una reconstrucción abarca varias camas, una parte de la reconstrucción se realiza en los volúmenes parciales en la zona donde las camas son adyacentes, solapando información. Mediante unos filtros y unos pesos son elegidos los datos que deben ser tenidos en cuenta en la reconstrucción final. Durante esta etapa el tiempo es invertido en dicha selección.

- Etapa de escritura: Una vez se han terminado todos los procesamientos sobre las imágenes y el volumen ha sido reconstruido, se escribe en el fichero designado.

Las fases de filtrado y retroproyección son altamente paralelizables debido a la falta de dependencias y el alto número de operaciones paralelas con los datos [36].

4. ANÁLISIS

4.1 DETALLE DE LA APLICACIÓN

En esta seccioón presentamos una vista general de Mangoose y del disenño y la implementaci ón de la versión mejorada Mangoose++. El objetivo principal es mejorar la flexibilidad en la configuración de realización y escalabilidad de la aplicación, mientras que la aplicación alcance un alto grado de utilizacioón de los recursos hardware disponibles.

Una de las metodologías de paralelización más populares consiste en seguir cuatro etapas: descomposición, asignación, orquestación y mapeo.

Mientras que esta metodología puede ser aplicada directamente para un problema funcional dado, Mangoose requiere un enfoque más complejo motivado por cuatro razones. Primero, Mangoose consiste en cuatro módulos funcionales principales, que pueden requerir diferentes granularidades en la paralelización. Segundo, el desarrollo de dependencias entre m odubs muestra efectos no lineales. Tercero, el objetivo principal es mapear la aplicación sobre una arquitectura híbrida con una compleja jerarquía de memoria. Cuarto, hay un número de optimizaciones que pueden ser aplicadas en la interfaz entre distintas unidades funcionales.

La implementación eleva el grado de paralelismo tanto a nivel de aplicación como a nivel de hardware. La fase de lectura toma proyecciones en 2D de un fichero en memoria principal. En la implementación, la granularidad de la fase de lectura puede variar desde una proyección 2D hasta el total del número de proyecciones. La fase de lectura del fichero está implementada utilizando operaciones asíncronas (aio read), solapando la lectura con la fase de filtrado. La cantidad máxima de datos que pueden ser leídos de una vez viene limitada por el tamaño de ventana de la lectura adelantada, siendo éste un parámetro configurable. Dos de las cuestiones en el proceso de optimización son cúando es preferible utilizar operaciones síncronas o asíncronas y cómo elegir el valor óptimo para la granularidad y el tamaño de ventana en la fase de lectura. Esta decision no es trivial debido a los efectos no lineales que tienen las optimizaciones en las distintas fases de la aplicación.

En la fase de reparto (S) se transfieren los datos desde memoria principal a la memoria de los dispositivos. La transferencia puede ser solapada con la ejecución del kernel utilizando memoria no paginable. Es un enfoque recomendado por Nvidia para mejorar la transferencia entre el host y los dispositivos. Nuestra implementación proporciona como una opción la posibilidad de usar memoria no paginada en la fase S.

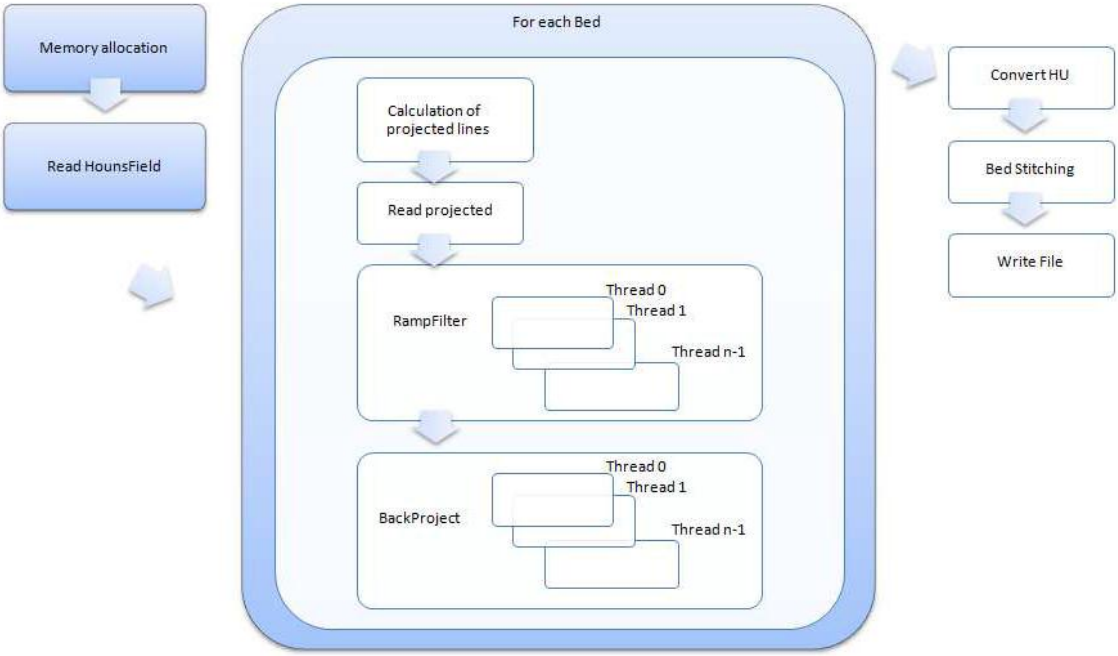


Ilustración 6: Orden de ejecución de Mangoose

4.2 REQUISITOS SOFTWARE

Para el correcto funcionamiento de Mangoose++ y que permita ser optimizado eficazmente, deben existir una serie de características indispensables que propicien el buen funcionamiento de las nuevas implementaciones, denominados requisitos.

La tabla explicativa se compone de diversos campos:

- ID Requisito: identificador unívoco para diferenciar unos de otros.
- Tipo: funcional o de usabilidad, según corresponda de acuerdo a su cometido.
- Evento: situación que debe ocurrir como disparador.
- Descripción: breve definición del cometido.
- Justificación: por qué es importante el requisito y qué motivación lo ha propiciado.
- Criterio de cumplimiento: comprobante de una implementación del requisito satisfactoria.
- Prioridad: Relevancia del requisito.
- Conflictos: requisitos que no pueden ser implementados si ése requisito es implementado.

Los requisitos que deben cumplirse son los siguientes:

ID Requisito	RF-01	Tipo:	Funcional
Evento	Compilar el código traducido a C++		
Descripción	El compilador no muestra ningún error ni aviso por pantalla		
Justificación	Se debe poder compilar totalmente el código para poder proceder a su ejecución y proceder con el cálculo de tiempos.		
Criterio de cumplimiento	Compilar el código y que no se muestre ningún aviso y error		
Prioridad	Alta	Conflictos	#

Tabla 2: Requisitos - 1

ID Requisito	RF-02	Tipo:	Funcional
Evento	Ejecución de Mangoose codificado en C++		
Descripción	Mangoose reconstruye un volumen		
Justificación	El reconstructor debe generar los volúmenes a partir de las proyecciones del objeto proporcionadas.		
Criterio de cumplimiento	Lanzar una ejecución y obtener el resultado esperado		
Prioridad	Alta	Conflictos	#

Tabla 3: Requisitos - 2

ID Requisito	RF-03	Tipo:	Funcional
Evento / Caso de Uso	Múltiples hilos en CPU usando Intel TBB		
Descripción	Paralelismo con threads		
Justificación	Permitir la ejecución paralela en su versión CPU con la generación masiva de hilos e hyperthreading		
Criterio de cumplimiento	Lanzar una ejecución con multitud de hilos y obtener el resultado esperado		
Prioridad	Alta	Conflictos	#

Tabla 4: Requisitos - 3

ID Requisito	RF-04	Tipo:	Funcional
Evento / Caso de Uso	Cálculo en flotantes/enteros.		
Descripción	Las reconstrucciones pueden hacerse utilizando números enteros o en coma flotante.		
Justificación	Niveles de precisión.		
Criterio de cumplimiento	Lanzar una ejecución y obtener el resultado esperado		
Prioridad	Alta	Conflictos	#

Tabla 5: Requisitos - 4

ID Requisito	RU-01	**Tipo:**	Usabilidad
Evento / Caso de Uso	Tiempo de reconstrucción.		
Descripción	Mostrará el tiempo de proceso invertido en cada etapa.		
Justificación	Dividir el tiempo total por etapas para observar el grado de mejora y aquellas fases donde el coste debe ser reducido.		
Criterio de cumplimiento	Tiempos mostrados correctamente al ejecutar Mangoose.		
Prioridad	Alta	**Conflictos**	#

Tabla 6: Requisitos - 5

ID Requisito	RU-02	**Tipo:**	Usabilidad
Evento / Caso de Uso	Etapa en curso		
Descripción	Mostrará por pantalla los avances en ciertas etapas durante el proceso de reconstrucción.		
Justificación	Que el usuario sepa que el proceso está en curso y progresa adecuadamente.		
Criterio de cumplimiento	Que indique los avances durante la ejecución de Mangoose.		
Prioridad	Alta	**Conflictos**	#

Tabla 7: Requisitos - 6

5. CATÁLOGO DE OPTIMIZACIONES

A continuación pasan a describirse las optimizaciones llevadas a cabo sobre el código para hacerlo más eficiente que en C.

5.1 Traducción del código

En esta parte es en la que menos se ha tardado, ya que la gran mayoría del código en C es aceptado por el compilador de C++, con lo que se ha cambiado la extensión de los ficheros de *.c* a *.cpp*, procediendo así a que el compilador de C++ compilase el código. Después de hacer esto, se solucionaron los siguientes problemas:

1. Varias variables que se declaraban no se usaban a lo largo del programa, con lo que se procedió a su eliminación, ahorrando así un pequeño espacio en memoria principal y ahorrando algo de tiempo a la compilación y ejecución del programa.
2. Los *casting* en C se hacen de diferente manera que en C++

Casting C	Casting C++
`int a = (int) b;`	`int a = static_cast<int>(b)`

Tabla 8: Diferencia *castings*

   Además, hay *castings* que en C son implícitos, pero en C++ hay que ponerlos de manera explícita.
3. Se han eliminado todos los *includes* referentes a bibliotecas del lenguaje C para usar las bibliotecas del propio C++ como, por ejemplo, la biblioteca para la escritura en ficheros.
4. Se han modificado todos los `printf("mensaje")` por la llamada estándar de C++ `cout<<"mensaje"<< endl`.

Estos son todos los cambios directos que se han realizado según se ha hecho el cambio para compilar. A diferencia de estos, os siguientes son para transformar el código C a código C++.

5.1.1 Modificación de punteros

Todas las variables puntero de C que fuesen de otro tipo que `char*` han sido modificadas a la clase *vector* de C++, ya que permite un control mayor sobre el tamaño del mismo y facilita la necesidad de reservar memoria para él. Además, como todas las instanciaciones de la clase *vector* son variables de una magnitud considerable, se han modificado todos los pasos de *vectors* como parámetros de función para que se pase siempre por referencia, ya que en C++, tanto los objetos como las variables de tipo básico se pasan por copia y no por referencia.

Desglose en detalle de la mejora en el paso de variables por referencia.

Paso por referencia	Tiempo de ejecución (proyección de 512x512x1)
No	30,76 seg.
Si	10,32 seg.
Mejora	33,55%

Tabla 9: Tiempo paso referencia

5.1.2 Uso de la clase *string*

Todos los punteros del tipo `char*` que contuviesen cadenas de caracteres legibles han sido modificadas por la clase de C++ string, ya que permite control y manejo más cómodo de este tipo de dato.

5.1.3 Transformación de estructuras *struct* a clases

Todas aquellas structs que se hayan creado englobar datos necesarios en las diferentes fases del programa en un solo lugar han sido rescritas como clases de C++, acercando más el código a la orientación a objetos. Además, se han tenido que modificar funciones para que pasasen estos datos y, como se ha mencionado antes, el paso de este objeto se hace como referencia, pasando la dirección de memoria donde empieza el objeto y aumentar la velocidad del programa y disminuir su peso en memoria principal.

5.2 LOCALIDAD EN MEMORIA

En su versión inicial los cálculos eran una traducción de los realizados con una aplicación específica para cálculo numérico como es Matlab. Sin embargo, en el tratamiento de matrices Matlab realiza los cálculos recorriéndolas por columnas en lugar de por filas.

Hacerlo de ésta manera provoca continuos fallos en la caché, aumentando en una gran cantidad el tiempo de computación necesario en las operaciones con matrices. Para solventarlo, el código fue editado rotando el volumen y modificando el eje por donde debían iniciarse las operaciones sobre las matrices.

Así se logra un alto grado de localidad en la memoria, puesto que las matrices pasan a recorrerse por filas en lugar de por columnas, minimizando el fallo en la caché y reduciendo los costosos accesos a memoria principal. En la etapa de escritura la rotación vuelve a invertirse para dejar el volumen reconstruido en su posición inicial.

El concepto de caja negra se mantiene ya que el usuario obtiene la misma salida a partir de la misma entrada que en su versión inicial. Sin embargo, el tiempo total se ve reducido, en mayor o menor medida dependiendo de las prestaciones hardware por el coste que genere un fallo en memoria caché.

5.3 Intel TBB

Durante el desarrollo de toda la aplicación, se realizan muchas iteraciones definidas por las instrucciones de control *for*, como es en procesos de lectura, filtrado o retroproyección.

Para alcanzar un mayor nivel de paralelismo, los bucles de mayor nivel se han dividido mediante la sentencia:

parallel_for

La forma de usar esta función es la que sigue:

```
parallel_for(int ini, int fin, int prog, [&](int n){
        …
});
```

Aspectos destacables:

- La variable ini representa el valor con el cual empieza el bucle.
  La variable fin representa el valor con el que se termina el bucle.
  La variable prog indica cuantos saltos del bucle.
  La variable n almacena el valor en el que se encuentra el bucle en cada momento.

- **[&]:** Esta opción representa que todas las variables, tanto las internas como las externas de la función se pasan como referencia y, lo más importante, se pueden usar tanto de escritura como de lectura.
- Como se puede apreciar, el uso de esta función se parece mucho al uso normal del bucle *for*, con lo que se facilita la lectura y comprensión del código a la vez que facilita su uso.

Intel TBB, si no se le dice lo contrario, se encarga automáticamente de comprobar cuantos núcleos posee el procesador y lanza tantos hilos como núcleos. Pero para poder desarrollar todas las pruebas con diferentes números de hilos, es necesario usar task_scheduler_init init(NUMERO_HILOS).

Esto ha sido necesario ya que la máquina en la que se ejecutan las pruebas es de 16 núcleos, pero las pruebas realizadas utilizan 4, 8, 16 y 32 hilos.

5.4 Comparación entre códigos

Después de realizar todos los cambios mencionados anteriormente, se van a comparar 2 fragmentos de código, uno escrito en C y otro escrito en C++ para observar las diferencias que existentes entre ambos códigos. Ambos fragmentos de código corresponden a la etapa de retroproyección.

5.4.1 Fragmento de código C

```
void backproject_cpu (float ***volume, float rad_angle,
                const struct backpar param, float *projection)
```

```
{
    int n;
    const int column = param.height_proj;
    const float cos_a = cos (rad_angle);
    const float sin_a = sin (rad_angle);

#pragma omp parallel for num_threads(threadsOMP)
    for (n = 0; n < param.n_index; ++n) {
        short int i, j, s_ind, z;
        float rot_ind_u, rot_ind_v, aux_dist, magnification, slice_weight,
            aux_s_ind, s_w, comp_s_w, *aux_volume;
        const float *aux_proy0, *aux_proy1;

        i = param.ROI_Xindex[n];
        j = param.ROI_Yindex[n];
        //i = param.ROI_Xindex[n];
        //j = param.ROI_Yindex[n];
        //rotated index
        rot_ind_u =
            (cos_a * (i - param.med_s_bin) * param.x_pix_size -
             sin_a * (j -
                      param.med_s_bin) * param.y_pix_size) /
            param.proj_pix_size;
        rot_ind_v =
            (sin_a * (i - param.med_s_bin) * param.x_pix_size +
             cos_a * (j - param.med_s_bin) * param.y_pix_size);

        //magnification
        aux_dist = param.so_distance - rot_ind_v;
        magnification = param.so_distance / aux_dist;
        slice_weight = magnification * magnification / aux_dist;

        //s index in projection
        aux_s_ind =
            (rot_ind_u) * magnification + param.centro_s_proy;
        s_ind = (short int) aux_s_ind;
        s_w = 1 - (aux_s_ind - (s_ind));
        comp_s_w = (1 - s_w);


        //position in volume
        aux_volume =
            volume[i - InputPar.min_coronal][j - InputPar.min_sagittal];
        aux_proy0 = projection + (s_ind * column);
        aux_proy1 = projection + ((s_ind + 1) * column);

        //<IVM: 2007-02-15: min_slice is now global --> we still need the
local_,min/max_slice>
        for (z = param.local_min_slice; z < param.local_max_slice + 1; ++z) {
            short int z_ind;
            float aux_z_ind, z_w, comp_z_w, a, b, c, d, acumul_vol;

            aux_z_ind =
                param.z_obj_proj * (z - param.med_z_bin) * magnification +
                param.centro_z_proy_roi;
            z_ind = (short int) aux_z_ind;
            z_w = 1 - (aux_z_ind - (z_ind));
            comp_z_w = (1 - z_w);
            a = s_w * z_w;
            b = comp_s_w * z_w;
            c = s_w * comp_z_w;
            d = comp_s_w * comp_z_w;
#ifdef PROJECTOR
            acumul_vol = ((aux_proy0[z_ind]) * a +
                          (aux_proy0[z_ind + 1]) * c +
                          (aux_proy1[z_ind]) * b +
                          (aux_proy1[z_ind + 1]) * d);
#else

            acumul_vol = ((aux_proy0[z_ind]) * a +
                          (aux_proy0[z_ind + 1]) * c +
                          (aux_proy1[z_ind]) * b +
                          (aux_proy1[z_ind + 1]) * d) * slice_weight;
#endif
            (*aux_volume++) += acumul_vol;
        }
```

```
    }
}
```

5.4.2 Fragmento código C++

```
void backproject_cpu(vector<vector<vector<float>>>& volume, float rad_angle,
const mangoose_backpar& param, vector<float>& projection,
             mangoose_cLineParam& InputPar) {

       const int column = param.height_proj;
       const float cos_a = cos(rad_angle);
       const float sin_a = sin(rad_angle);


       parallel_for(size_t(0), size_t(param.n_index), size_t(1), [&](size_t k){

              short int z;
              float rot_ind_u, rot_ind_v, aux_dist, magnification,
slice_weight, aux_s_ind, s_w, comp_s_w;
              int val_z = 0;
              short int z_ind, i, j, s_ind;
              double acumul_vol;
              int aux_proy0, aux_proy1, val_x, val_y;
              float aux_z_ind, z_w, comp_z_w, a, b, c, d;

              i = param.ROI_Xindex[k];
              j = param.ROI_Yindex[k];

              rot_ind_u = (cos_a * (i - param.med_s_bin) * param.x_pix_size -
sin_a * (j - param.med_s_bin) * param.y_pix_size) / param.proj_pix_size;
              rot_ind_v = (sin_a * (i - param.med_s_bin) * param.x_pix_size +
cos_a * (j - param.med_s_bin) * param.y_pix_size);

              //magnification
              aux_dist = param.so_distance - rot_ind_v;
              magnification = param.so_distance / aux_dist;
              slice_weight = magnification * magnification / aux_dist;


              aux_s_ind = (rot_ind_u) * magnification + param.centro_s_proy -
param.orla;
              s_ind = static_cast<short int>( aux_s_ind );
              if (s_ind < 0) s_ind = 0;
              s_w = 1 - (aux_s_ind - (s_ind));
              comp_s_w = (1 - s_w);

              aux_proy0 = s_ind * column;
              aux_proy1 = (s_ind + 1) * column;
              val_x = i - InputPar.min_coronal;
              val_y = j - InputPar.min_sagittal;
              vector<float> v = volume[val_x][val_y];
              val_z  = 0;
              for (z = param.local_min_slice; z < param.local_max_slice + 1;
z++) {

                     aux_z_ind = param.z_obj_proj * (z - param.med_z_bin) *
magnification + param.centro_z_proy_roi;
                     z_ind = static_cast<short int>(aux_z_ind);
                     z_w = 1 - (aux_z_ind - (z_ind));
                     comp_z_w = (1 - z_w);
                     a = s_w * z_w;
                     b = comp_s_w * z_w;
                     c = s_w * comp_z_w;
                     d = comp_s_w * comp_z_w;
```

```
                    acumul_vol = ((projection[aux_proy0 + z_ind]) * a +
(projection[aux_proy0 + z_ind + 1]) * c + (projection[aux_proy1 + z_ind]) * b
                                + (projection[aux_proy1 + z_ind + 1]) * d) *
slice_weight;
                    v[val_z] += static_cast<float>(acumul_vol / param.binning);
                    val_z++;
                }
        });
}
```

5.4.3 Diferencias en los fragmentos de código

Código C	Código C++
En la llamada a la función, todas las variables se pasan por copia	En la llamada a la función, todos los objetos se pasan por referencia y los tipos básicos se pasan por copia
La variable volume es un puntero tridimensional de floats	La variable volume es un vector tridimensional de floats
La variable projection es un puntero unidimensional de float	La variable projection es un vector unidimensional de float
La variable param es de tipo struct backpar	La variable param es un objeto de tipo mangoose_backparam
Se utiliza la sentencia #pragma omp parallel for num_threads(threadsOMP) Para paralelizar el trabajo de toda la función	Se modifica el bucle for por parallel_for(size_t(0), size_t(param.n_index), size_t(1), [&](size_t k){ para paralelizar el trabajo de toda la función

El cálculo de las posiciones que se utilizan para el cálculo del volumen acumulado:
 	
aux_proy0 = projection + (s_ind * column);
 	
aux_proy1 = projection + ((s_ind + 1) * column);
	El cálculo de la posición que se utilizan para el calculo de volumen acumulado:
	
	aux_proy0 = s_ind * column;
	
	aux_proy1 = (s_ind + 1) * column; vector<float> v = volume[val_x][val_y];

El calculo del volumen acumulado #ifdef PROJECTOR acumul_vol = ((aux_proy0[z_ind]) * a + (aux_proy0[z_ind + 1]) * c + (aux_proy1[z_ind]) * b + (aux_proy1[z_ind + 1]) * d);	El calculo del volumen acumulado acumul_vol = ((projection[aux_proy0 + z_ind]) * a + (projection[aux_proy0 + z_ind + 1]) * c + (projection[aux_proy1 + z_ind]) * b + (projection[aux_proy1 + z_ind + 1]) * d) * slice_weight;

#else
acumul_vol = ((aux_proy0[z_ind]) * a +
(aux_proy0[z_ind + 1]) * c +
(aux_proy1[z_ind]) * b +
(aux_proy1[z_ind + 1]) * d) * slice_weight;
#endif	

Modificación del valor de la variable aux_volume:
(*aux_volume++) += acumul_vol;
	Modificación del valor de la variable v:
	v[val_z] += static_cast<float>(acumul_vol / param.binning);
	val_z++;

Tabla 10: Diferencias de codificación entre C y C++

6. EVALUACIÓN DE RENDIMIENTO

En el siguiente capítulo se aborda la efectividad de las mejoras realizadas, se realizan diferentes ejecuciones para analizar características que puedan ayudar en líneas futuras de investigación y posibles optimizaciones adicionales.

6.1 PARALELISMO

Para las pruebas se ha dispuesto de un Intel(R) Xeon(TM) con un procesador de 8 núcleos con hyperthreading, pudiendo ejecutar hasta 16 hilos de ejecución; 512 KB de memoria caché y 32 GB de memoria RAM. Este equipo forma parte de la infraestructura de cómputo de altas prestaciones del grupo de investigación ARCOS.

Las siguientes secciones muestrarán las comparativas entre la utilización de 4 hilos de ejecución a 32 hilos de ejecución y su grado de mejora, así como el tiempo total empleado en cada ejecución.

6.1.1 Proyecciones 512x512 en C++

En este primer caso de estudio se han generado volúmenes de 512x512x512 píxeles utilizando el código en C++, obteniendo los siguientes tiempos:

Hilos	Filtro (seg.)	Retroproyección (seg.)	Pegado volúmenes (seg.)	Hounsfield (seg.)	Escritura volumen (seg.)	Total (seg.)
4	14,442	103,52	8,542	7,1e-8	2,386	130,612
8	11,344	69,722	8,478	1,02e-7	2,390	93,583
16	8,55	43,848	8,478	9,7e-8	2,392	65,01
32	10,918	43,557	8,586	7,6e-8	2,398	67,22

Tabla 11: Tiempos paralelismo en C++ 512x512

Con estos tiempos, en la Ilustración no figuran los de valores de lectura y *hounsfiled* dado que el tiempo empleado es ínfimo.

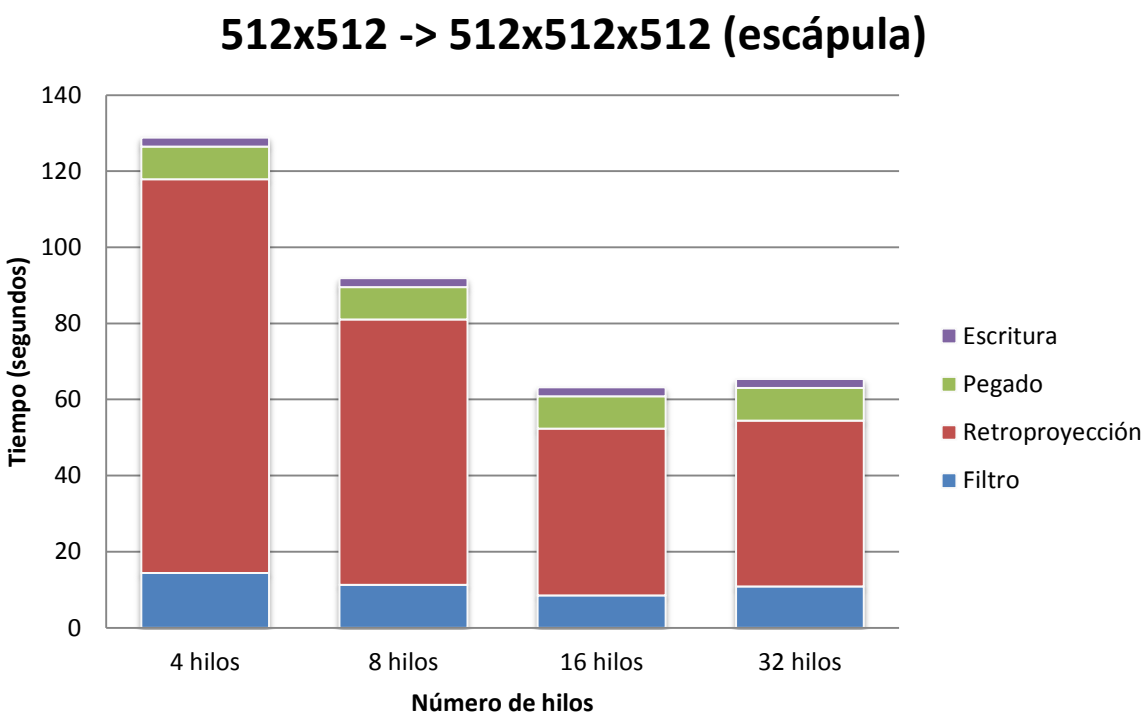


Ilustración 7: Paralelismo C++ 512x512

Como se puede apreciar en la imagen, el grado de mejora frente según se aumentan los hilos es bastante grande, llegando a ser la mitad en el caso de 16 hilos. En todos los casos, al lanzar un mayor número de hilos, se han experimentado mejoras en filtro y retroproyección, salvo en el último caso, en el cual el tiempo de filtro (10,918) es ligeramente mayor que en el caso de 16 hilos (8,55). Esto se debe a que la máquina en la que se han ejecutado las pruebas tiene un procesador con 16 núcleos, lo que hace que cuando se sobrepasa el número máximo de hilos que pueden ser concurrentes, se introduce un pequeño retraso por los cambios de contexto.

Sin embargo el tiempo en el resto de las fases se ha mantenido ya que el tiempo en todas ellas es mucho menor que el tiempo que se requiere para hacer el filtro y las retroproyecciones, con lo que no se ha intentado mejorar su tiempo.

Los tiempos de la fase de filtrado y retroproyección se han visto reducidos y se van reduciendo incrementalmente conforme aumenta el número de hilos (a excepción de lo mencionado antes).

Como se muestra en Ilustración , se centra el tiempo en las etapas más costosas y principal objetivo a optimizar, filtrado y retroproyección.

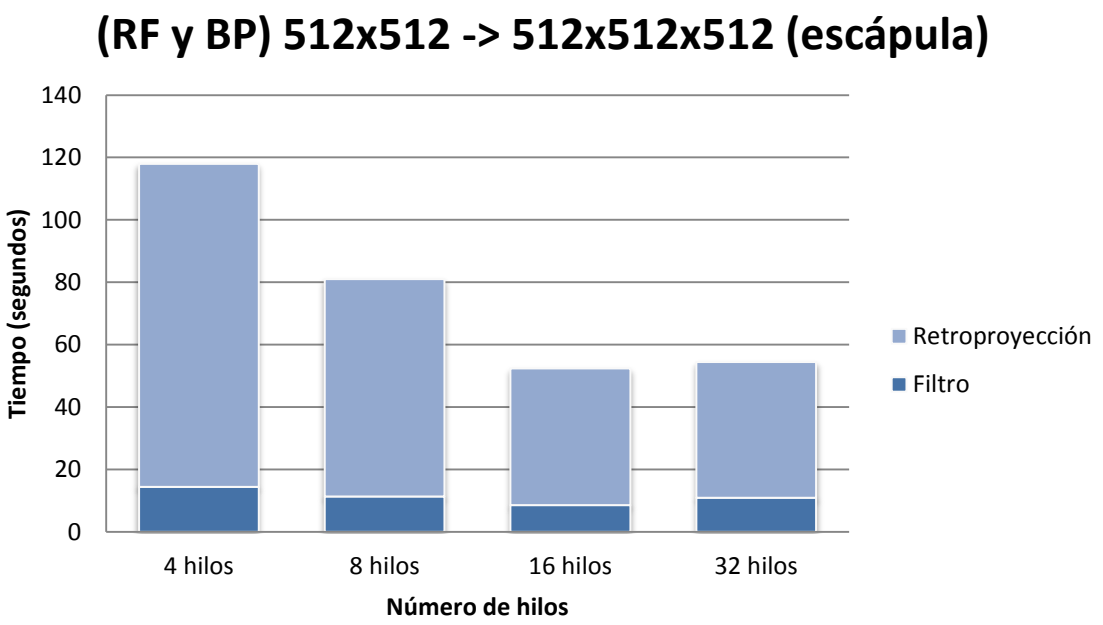


Ilustración 8: Paralelismo C++ (RF y BP)

A partir de esta gráfica se observa el amplio ratio de mejora existente al utilizar un mayor número de hilos en las etapas más importantes de Mangoose, reduciendo a partir de un tiempo inicial casi de 120 segundos hasta alcanzar un tiempo ligeramente superior a los 50 segundos. La aceleración alcanzada comparando estas dos etapas es de 2,25.

6.1.2 Proyecciones 512 x 512 en C

En este caso de estudio también se han generado volúmenes de 512x512x512 píxeles pero esta vez utilizando el código en C, obteniendo los siguientes tiempos:

Hilos	Filtro (seg.)	Retroproyección (seg.)	Pegado volúmenes (seg.)	Hounsfield (seg.)	Escritura volumen (seg.)	Total (seg.)
4	12,585	231,849	1,658	1,658	1,688	248,485
8	6,63	121,403	1,540	1,540	1,594	131,887
16	3,488	61,668	1,449	1,449	1,726	69,042
32	7,533	87,871	1,688	1,688	1,822	99,618

Tabla 12: Tiempos paralelismo C 512x512

Los tiempos de escritura han vuelto a ser ignorado por ser muy cercano a 0.

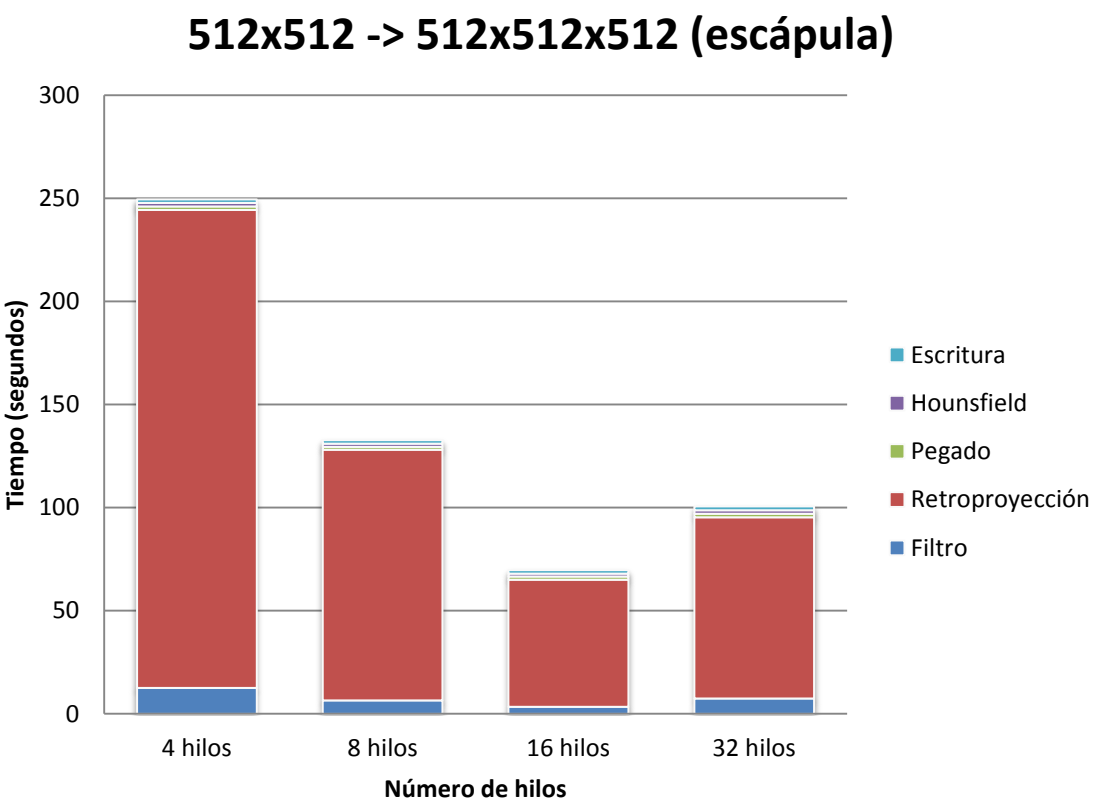


Ilustración 9: Paralelismo C 512x512

En este caso el grado de mejora en la reducción del tiempo se aprecia mucho más, alcanzando casi los 250 segundos con 4 hilos y llegando a estar por debajo de los 70 al utilizar 16 hilos, alcanzando un ratio de 3,6 de aceleración.

Análogo al caso de estudio anterior, centramos las optimizaciones en las etapas de filtrado y reconstrucción teniendo como resultado la Ilustración

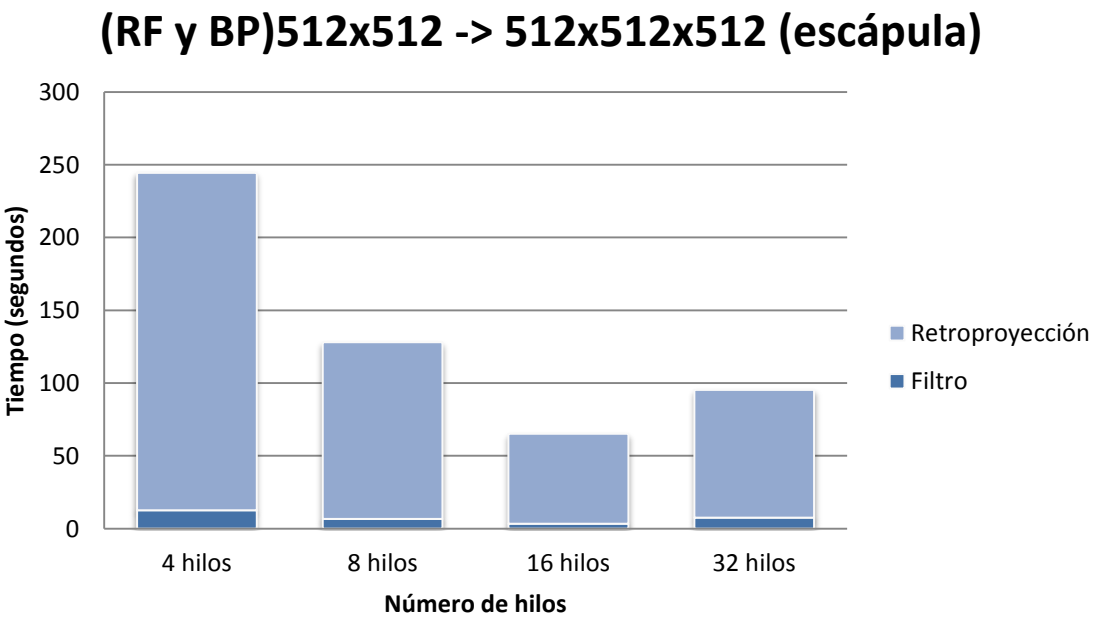


Ilustración 10: Paralelismo C 512x512 (RF y BP)

Con la obtención de esta gráfica y en base al tiempo fijo, se observa que el ratio de aceleración alcanzado con proyecciones grandes es superior al de proyecciones pequeñas, en este caso es de 6,76.

6.2 Comparativa de tiempos C VS C++

Haciendo un balance global de la utilización de los hilos para los casos de proyecciones con un tamaño de 512x512, se comparan los tiempos invertidos en las secciones de paralelismo con C y C++.

Para mostrar de una mejor manera los tiempos y mejoras alcanzadas, se va a comenzar por comparar los tiempos obtenidos en las etapas de filtrado y retroproyección y, por último, el tiempo total de ejecución en ambos lenguajes.

Antes de pasar a comprobar que código es mejor usando todos los recursos disponibles de la CPU, vamos a comprobar cual de los dos lenguajes se desenvuelve mejor utilizando un solo núcleo del procesador para ir comprendiendo mejor que lenguaje más optimo, independientemente de la cantidad de recursos hardware de los que se dispongan.

En este caso, sólo nos vamos a centrar en el tiempo total de ejecución del programa con ambos lenguajes y las dos etapas que más coste computacional tienen: filtrado y retroproyección. Mientras que en el caso de varios hilos, mostraremos los tiempos de todas las etapas de ejecución.

6.2.1 Comparativa de tiempos de filtrado con un hilo

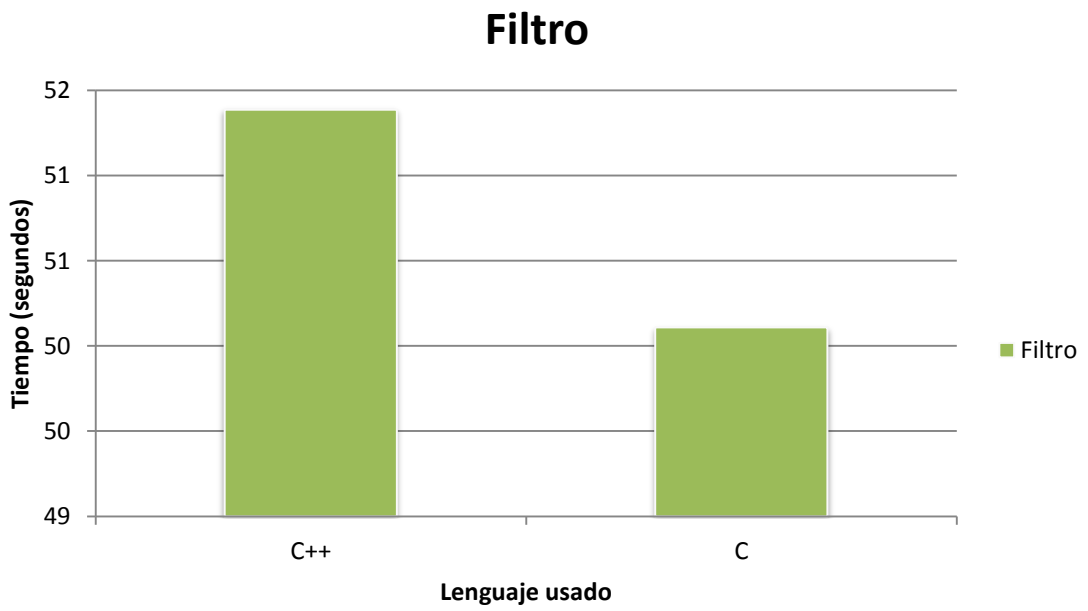


Ilustración 11: Comparación de tiempos de filtrado con un hilo

Vuelve a observarse lo mismo que en el anterior caso: en la etapa de filtrado, C es más rápido que C++. Pero en este caso, la diferencia es realmente pequeña, ya que C tarda un poco más de 50 segundos y C++ tarda casi 52 segundos, siendo, en el mejor de los casos, una diferencia total en el filtrado de 2 segundos de diferencia.

6.2.2 Comparativa de tiempos de retroproyección con un hilo

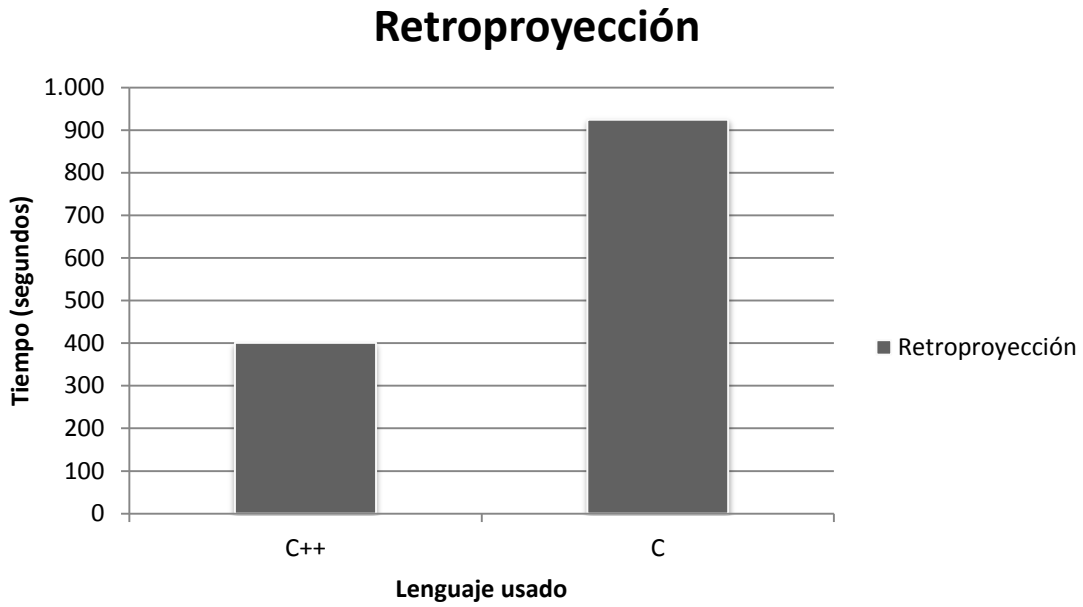


Ilustración 12: Comparación de tiempos de filtrado con un hilo

Podemos observar que el código en C++ es mucho más rápido en el cálculo de las retroproyecciones que C, llegando a tardar un 50% menos. Este caso es mucho más significativo que el anterior, ya que la diferencia de tiempo en esta etapa entre C y C++ es de unos 500 segundos, decantando la victoria de C++ sobre C sin necesidad de ver la última gráfica.

6.2.3 Comparativa de tiempos totales con un hilo

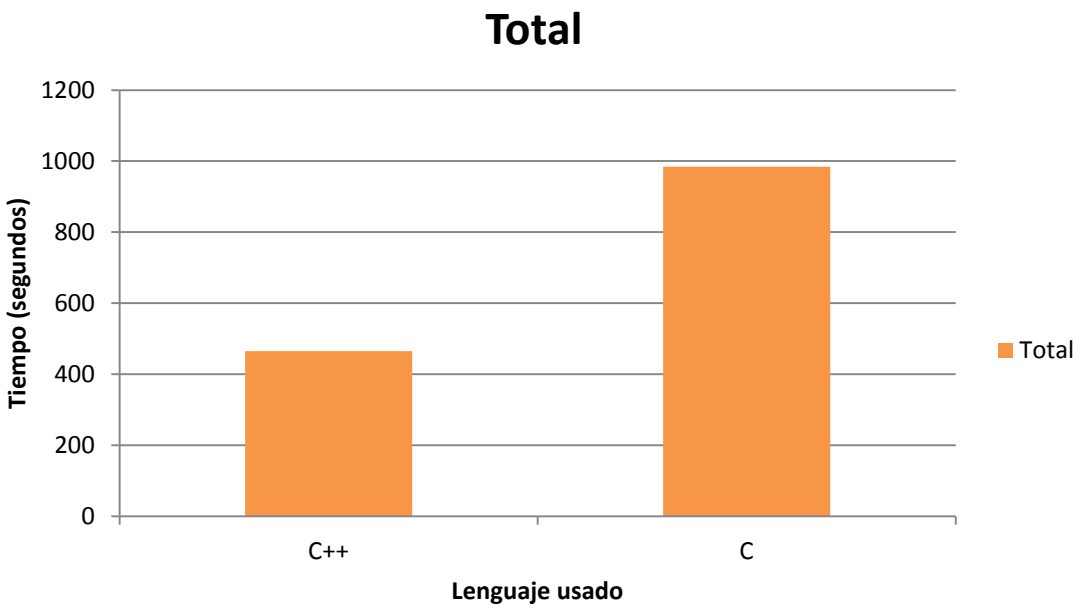


Ilustración 13: Comparación de tiempos de filtrado con un hilo

Como ya se había vaticinado antes, queda demostrado que el código en lenguaje C++ es más eficiente en cuanto a tiempo de ejecución se refiere con respecto a su homologo en lenguaje C. Esto ocurre porque C++ optimiza mejor el tiempo de ejecución en la etapa más costosa, la etapa de retroproyección.

6.2.4 Comparativa de tiempos de filtrado con varios hilos

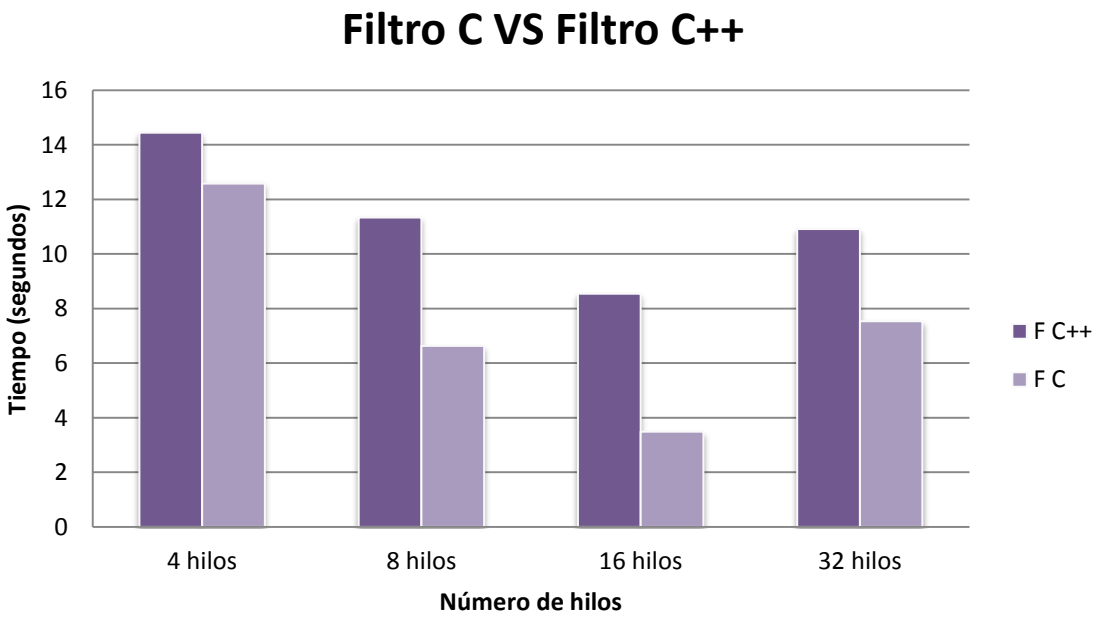


Ilustración 14: Filtro en C VS filtro en C++

Como se puede apreciar, los tiempos para el filtrado en código C son más rápidos que en C++, siendo C, en promedio, un 63,84% más rápido.

6.2.5 Comparativa de tiempos de retroproyección con varios hilos

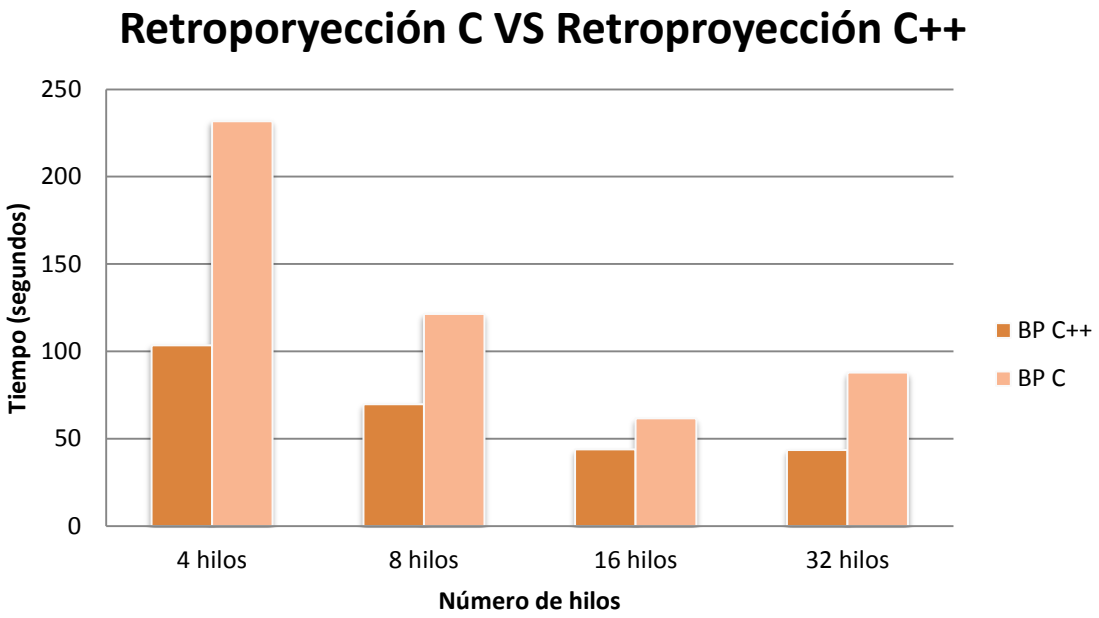
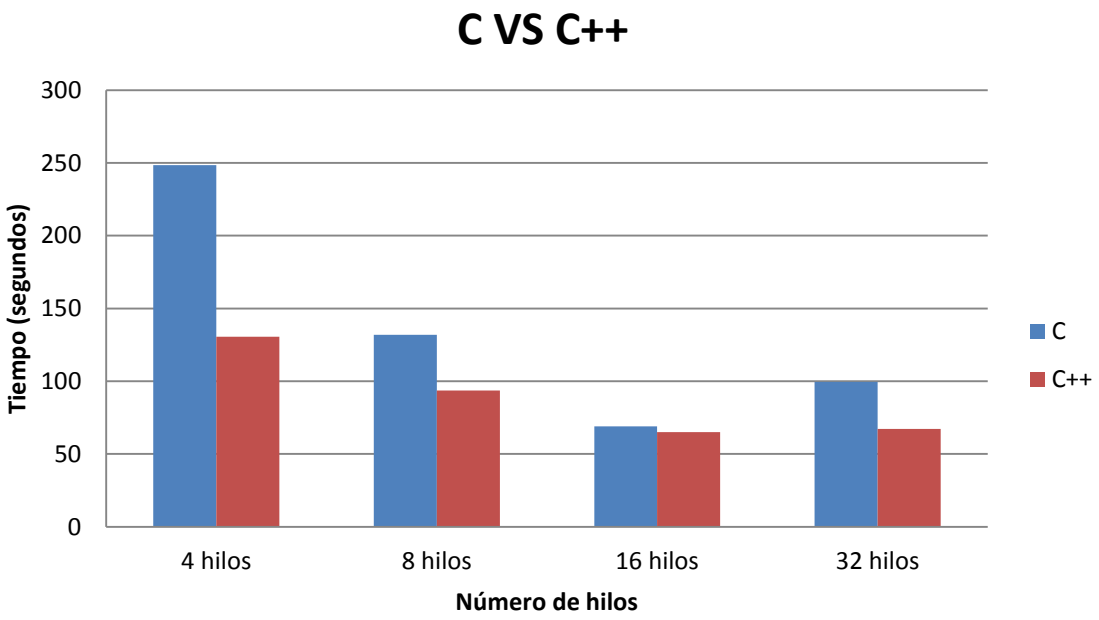


Ilustración 15: Retroproyección en C VS retroproyección en C++

Se observa que, en este caso, el tiempo necesario para llevar a cabo las retroproyecciones en C++ es mucho menor que en C, siendo C++, en promedio, un 55,69% más rápido.

Aunque en comparación con parece que C++ es más lento, ya que tiene un promedio menor que C, el tiempo que requiere hacer un filtrado es 10 veces menor, con lo que, sin necesidad de ver las siguientes gráficas, podemos asegurar que C++ es más eficiente a la hora de repartir y trabajar con varios hilos que C.

6.2.6 Comparativa de tiempos totales con varios hilos



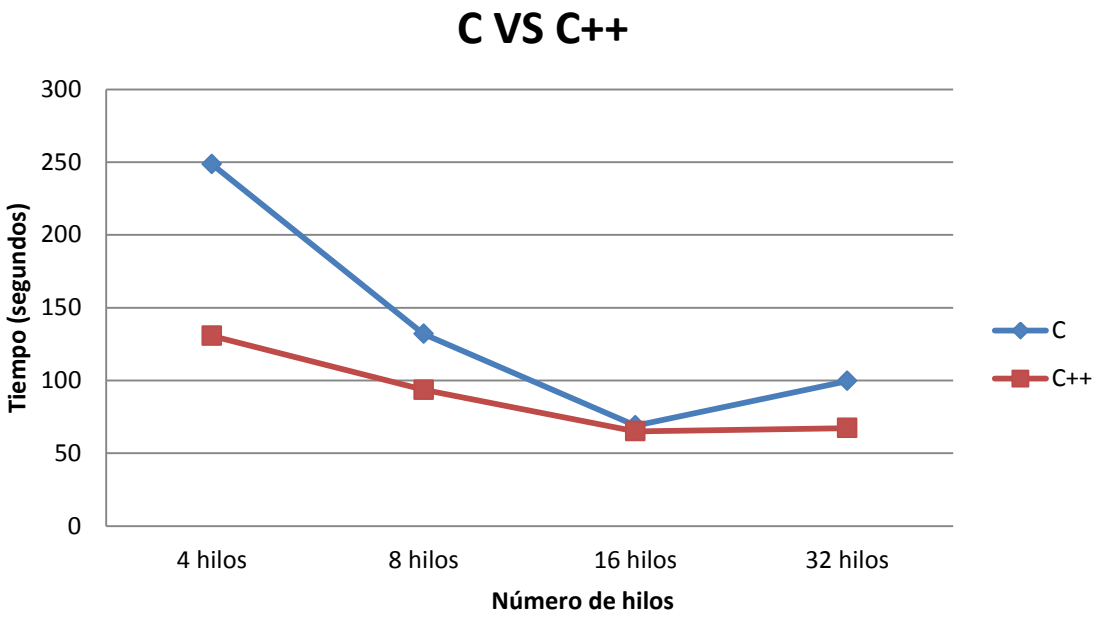


Ilustración 16: Comparación de tiempos paralelismo C VS C++

Como se puede observar en ambas gráficas, el tiempo que requiere el programa en C++ es mucho menor excepto en el caso de los 16 hilos ya que, como se ha explicado anteriormente, es el número máximo de hilos que puede ejecutar la máquina de pruebas de manera concurrente.

Se puede apreciar en la segunda gráfica que la progresión de tiempos en C es mucho más acentuada que en C++, siendo la línea de tiempo de este lenguaje mucho más parecido al de una recta, mientras que la del lenguaje C se asemeja más a una línea curva.

Cabe destacar que en el peor de los casos, el cual es el que usa sólo 4 hilos, la diferencia de tiempos de ejecución entre C y C++ es de 1,9 veces más rápido a favor de C++.

Y por último remarcar el último caso de prueba, el de 32 hilos, en el cual C++ se mantiene casi inalterable en el coste de tiempo. Sin embargo, C casi duplica su tiempo total de ejecución con respecto a su tiempo total de ejecución en el caso de 16 hilos.

7. ENTORNO DE PRUEBAS

Para la resolución del catálogo de optimizaciones y analizar el catálogo de optimizaciones implementado, se dispusieron de una serie de aplicaciones de soporte software que permitiera satisfacer las necesidades requeridas para la correcta resolución.

- **ImageJ:** Es un programa de procesamiento de imagen digital con licencia gratuita, realizado en Java y desarrollado por el *National Institutes of Health*, da un gran soporte como visor de imágenes por sus altas prestaciones, funcionalidades y la gran cantidad de formatos de imagen soportados, especialmente *RAW* (imágenes en crudo sin tratamiento digital).

En concreto se ha utilizado este soporte porque permite voltear el volumen, pudiendo analizarlo desde varias perspectivas como se muestra en las siguientes ilustraciones

Ilustración 17: ImageJ perspectiva 1

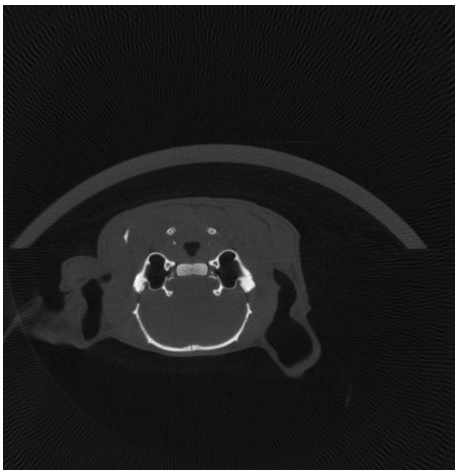


Ilustración 18: ImageJ perspectiva 2

Junto con la parte visual, donde inicialmente se aprecia la figura aparentemente correcta. En las imágenes anteriores puede apreciarse el caso de estudio de una rata donde, tomando la misma imagen, la **¡Error! No se encuentra el origen de la referencia.** muestra la reconstrucción vista desde el frente y la**¡Error! No se encuentra el origen de la referencia.** lo hace desde arriba.

Además otorga la posibilidad de mostrar el valor que tiene un punto, marcando sus coordenadas X e Y, como se aprecia en la Ilustración .

```
x=149, y=88, value=0.01027734
```

Ilustración 19: Peso ImageJ

Para comprobar la buena reconstrucción del volumen, una de las pruebas realizadas es la comparación del valor sobre distintos puntos elegidos aleatoriamente entre la versión inicial y la nueva compilada.

Otra funcionalidad por la que ImageJ resultaba una aplicación necesaria, era por su opción de restar dos imágenes. Esto consiste en restar los valores directos de cada punto, de esa manera si los puntos poseen el mismo valor o un valor muy próximo, el resultado es cercano a 0, por lo que el píxel toma el color negro. Así, si en la resta de imágenes se obtenía como resultado un lienzo negro, era indicador de una buena solución.

Con el amplio abanico de formatos soportados, permite comparar imágenes en 16 bit y 32 bit, pudiendo analizar resultados intermedios que restaran las imágenes generadas con números en coma flotante y tras la etapa del filtro. Dividiendo por etapas la reconstrucción gráfica para aislar posibles problemas.

- **Ztimer:** es una librería de Linux utilizada para cronometrar las etapas y observar los ratios de aceleración en lugar de utilizar cronómetros manuales. Pueden ser insertados en cualquier parte del código, con sentencias para iniciar y parar, mostrando por pantalla el tiempo invertido en cada una de las etapas, como muestra la siguiente ilustración:

```
Read 1.126682
Rampfilter 3.108522
Backproject 2.173479
Staging 2.205845
Deviation 0.502959
Overload 0.252145
Hounsfield 0.000000
Write file 4.030890
```

Ilustración 20: Ztimer

Esos mismos temporizadores eran traspasados y escritos en un fichero para ser importados posteriormente y realizar las comparativas entre los distintos casos de estudio.

7.1 Casos de estudio

Para las pruebas de los diferentes tamaños de proyección, se han utilizado tres casos de estudio reales:

- Proyecciones pequeñas, 512x512: Escápula de cocodrilo. Usados 2 ficheros de proyecciones con 180 proyecciones por fichero.

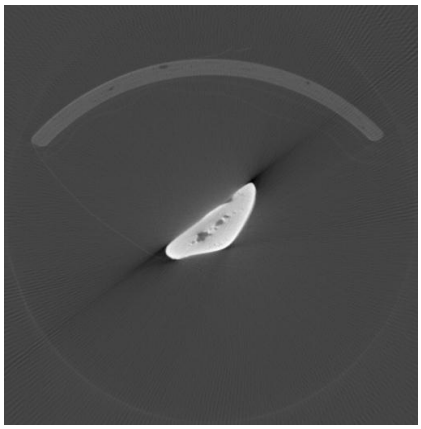


Ilustración 21 : Escápula Top

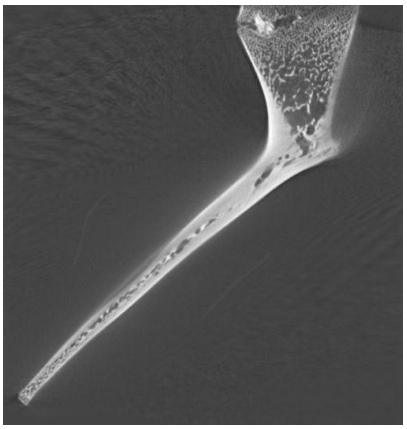


Ilustración 22 : Escápula Bottom

Estos casos de estudio fueron los dispuestos como objetivos de las pruebas y elementos a satisfacer para garantizar la calidad de las implementaciones de Mangoose. Con estos tamaños se abarcan los tamaños que serán utilizados con mayor frecuencia en el reconstructor con la escápula de cocodrilo y el maxilar, un valor irregular con la jeringuilla y comprobante del pegado y la desviación.

8. CONCLUSIONES Y LÍNEAS FUTURAS

Este Trabajo de Fin de Grado ha abordado las diferentes razones de usar C++ como alternativa más eficiente para optimizar los tiempos del código escrito en lenguaje C para el algoritmo desarrollado por Feldkamp, Davis y Kress y así favorecer la inclusión de nuevas tecnologías en las tareas de reconstrucción de imagen médica para ser aplicadas a la tomografía computarizada mediante el uso de haz cónico.

Para conseguir cumplir los objetivos planteados al inicio del proyecto, se han alcanzado diferentes hitos que han propiciado el satisfactorio desarrollo del proyecto:

- Traducir por completo el código en C a C++, ya que permite trabajar con objetos, es más eficiente que C y hace la programación más simple y llevadera.

- Proporcionar soporte para el paradigma de programación paralela Intel TBB mediante la generación de hilos y el aprovechamiento de la tecnología *hyperthreading*, dividiendo en un gran número de tareas simultáneas la ejecución y así reducir el lastre de tiempo generado por un comportamiento secuencial.

Cabe añadir que, en promedio, la aceleración que se ha conseguido usando C++ con respecto a usar C ha sido un 50% más rápido. Así se concluye que C++ es un lenguaje más eficiente que C para el problema planteado inicialmente.

Como líneas futuras que permitan expandir el proyecto:

- Además del uso de Intel TBB para poder paralelizar el trabajo, Intel también posee otra biblioteca llamada ArBB (ArrayBuilding Blocks), estando diseñada para el paralelismo vectorial.
- Una gran parte del tiempo invertido es a causa de los costes fijos de la creación de contextos, inicialización de variables, reservas de memoria o la lectura de las proyecciones. Una línea futura que permitiría reducir en una gran medida los tiempos, en especial las proyecciones de menor tamaño, sería la generación de un entorno que pudiera mantener estas necesidades de manera permanente, donde únicamente Mangoose se encargara del procesamiento desde la etapa de filtrado.
- Uso de aplicaciones visuales para hacer un análisis detallado del funcionamiento de la aplicación, con el objetivo de incrementar el rendimiento.

Aunando estas líneas futuras, podría alcanzarse una implementación automática y permanente de una aplicación que paraleliza y optimiza los recursos para la creación de volúmenes digitales que fuera independiente del hardware, del sistema operativo y que no necesitara supervisión más allá del mantenimiento, facilitado por la modularización en funciones de las distintas etapas de Mangoose++.

La importancia de poder exprimir al máximo los recursos de forma paralela tiene su razón y es que, con la tecnología actual no se pueden generar procesadores más rápidos por la cantidad de disipación de calor de la que tendrían que deshacerse los procesadores. Con lo cual, una forma de hacer los programas más rápidos es usar procesadores con varios núcleos y distribuir el programa por todos ellos de manera equitativa.

Con la evolución de las tecnologías en los procesadores multi núcleo, se han llegado a fabricar máquinas como Knights Corner, un co-procesador con arquitectura x86 desarrollado por la compañía Intel, teniendo este procesador en su interior la asombrosa cantidad de 50 núcleos, además de contar con otras características, como accesibilidad y programabilidad del coprocesador [37] [38].

Por avances como estos en la tecnología, es necesario conocer y usar las ventajas de generar código paralelo y, si es en C++, mucho más rápido y eficiente.

9. PRESUPUESTO

Para realizar el cálculo del presupuesto del proyecto se han tomado en consideración diferentes aspectos, en primer lugar los requisitos técnicos y de hardware:

Elemento	Coste unitario	Nº Unidades	Total
CPU Mangoose	2876,00 €	1	2876,00 €
Ordenador personal para desarrollo	840,00 €	1	840,00 €
Coste total			**9206,00 €**

Tabla 13 : Presupuesto Hardware

Tras los costes hardware, se pasa a tener en cuenta los recursos de software utilizados como se detallan a continuación:

Elemento	Coste unitario	Nº Unidades	Total
Licencia Windows 7	164,00 €	1	164,00 €
Licencia Microsoft Office 2010	199,00 €	1	199,00 €
Software libre	#	1	#
Coste total			**363,00 €**

Tabla 14 : Presupuesto Software

El tiempo de vida estimado de los recursos hardware, a donde los recursos software están adscritos, es de 30 meses de duración, por lo que se ajustan los costes amortizados, en base a la duración del proyecto, en este caso de 8 meses:

Elemento	Coste completo	Coste mensual	Coste amortizado
Amortizamiento HW	9206,00 €	306,87 €	2454,94 €
Amortizamiento SW	363,00 €	12, 10 €	96,80 €
Coste total			**2551,74 €**

Tabla 15 : Presupuesto amortizado

Se consideran los gastos de personal con el desglose de horas invertidas y el coste por hora:

Elemento	Coste hora	Coste mes	Horas totales	Coste proyecto

Becario	5 €	400,00 €	480	2400,00 €
Personal investigador	11,25 €	1800,00 €	300	3375,00 €
Personal investigador	11,25 €	1800,00 €	300	3375,00 €
Coste total				**9150,00 €**

Tabla 16 : Presupuesto personal

Por último, se consideran los gastos de transporte:

Medio de transporte	Tipo gasolina	Coste litro	Coste mensual	Coste total
Coche	Diesel	1,437 €/l	120,00 €	960,00 €

Tabla 17 : Presupuesto transporte

Unificando todos los costes desglosado anteriormente, obtenemos como resultado final que el coste de desarrollo del proyecto ha sido:

Elemento	Total
Hardware	2454,94 €
Software	96,80 €
Personal	9150,00 €
Medio de transporte	960,00 €
Coste total	**12393,74 €**

Tabla 18 : Presupuesto final

Dado que el proyecto ha sido desarrollado por la Universidad Carlos III de Madrid con fines de investigación y sin carácter comercial, no se aplican porcentajes extra por riesgos y beneficios, junto con los costes computados de facturación de personal. Quedando como resultado el coste real de las horas y los recursos.

APÉNDICE

Apéndice - Manual de usuario para Mangoose

Mangoosees una implementación multi-cama de un algoritmo de reconstrucción digital basado en el desarrollado por Feldkamp, Davis y Kresspara escáneres de tomografía computarizada que utilizan la geometría basada en el haz cónico.

La arquitectura del sistema se muestra en la siguiente figura. Para facilitar los cálculos geométricos, se hace uso de las coordenadas u,v,z comunes tanto al detector virtual situado en el centro del FOV como al volumen reconstruido. El origen O está localizado en el centro del FOV, que se corresponde también con el punto central del detector virtual y el centro de rotación.

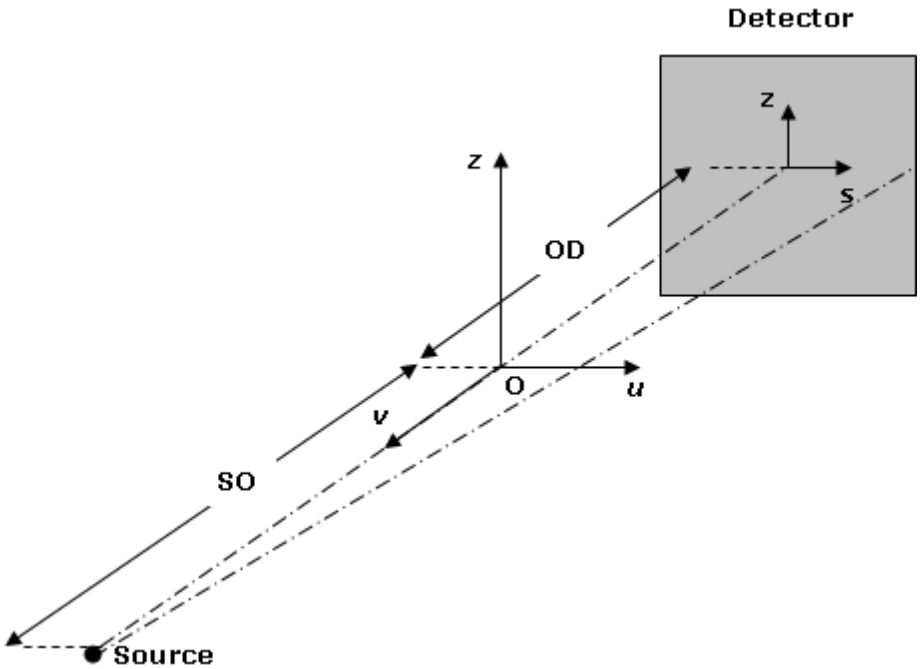


Ilustración 23 : Perspectiva de la arquitectura de haz cónico en CT. SO es la distancia entre el emisor y el origen; OD es la distancia entre el detector y el origen

Apéndice - Requisitos mínimos

Los requisitos mínimos son un procesador de 32 bits y al menos 512MB de memoria RAM. No obstante, la memoria mínima para ciertas reconstrucciones depende del tamaño y la resolución

de la imagen, llegando a necesitar 16GB de memoria RAM para volúmenes de 3 camas a la máxima resolución (tamaño de píxel de 50 microns). Para los estudios realizados normalmente, un equipo con 4GB de RAM es suficiente.

Apéndice - Ficheros de entrada

El programa necesita 2 ficheros para funcionar:

1. Fichero de calibración de unidades Hounsfield

El fichero de calibración de unidades Hounsfield contiene los datos para realizar la conversión de las unidades internas a los números utilizados por la tomografía computerizada en cada una de las calibraciones realizadas en los haces espectrales. Como resultado, los números obtenidos por la tomografía son una conversión lineal de los valores originales.

Cada línea en el fichero contiene 3 campos ASCII diferentes, la energía del haz (float), el valor del desplazamiento (float) y la pendiente del ajuste lineal (float).Los diferentes campos dentro de la línea van separados por comas.

Ejemplo: Para una energía en el haz de: 45 kV, desplazamiento: -956.151 y pendiente: 120128.09 los datos dentro del fichero serían:

45.0:-956.151:120128.09

2. Uno o más ficheros de proyecciones que contengan las imágenes obtenidas cubriendo los 360 ángulos, satisfaciendo los siguientes requisitos:

- Los nombres de estos ficheros seguirán este esquema:

  base_filemame_N_O.ctf

  Donde*base_filename*es el nombre del estudio, *N* es la posición de la cama actual dentro del rango {0..bed_positions-1}, y O es el fichero actual dentro del rango {0..(num_files/num_beds)-1}.

- Los valores de los píxeles son almacenados consecutivamente y ordenados por filas, los datos son codificados con unsigned int (16 bits).

- El número de proyecciones de cada fichero debe ser múltiplo de 5.

Apéndice - Archivos del programa

- mangoose.h
- mangoose_cpu.h
- mangoose_cLineParam.h
- mangoose_backpar.h
- mangoose_dev.h
- mangoose_stitch.h
- mangoose_rampf.h
- FDK_CPU.cpp
- InputHandle.cpp
- FDK.cpp

- Common.cpp
- Main.cpp

Apéndice - Funciones

- ***int parse_args (intargc, char *argv[],mangoose_cLineParam& parameters)***: Introduce los parámetros de lectura

- ***void write_progress (int progress, mangoose_cLineParam& InputPar)***: Actualización del progreso.

- ***void write_error (int error, mangoose_cLineParam& InputPar)***: Escribe una descripción del error en el fichero de salida.

- ***void deviation_func(mangoose_dev& deviation, mangoose_cLineParam& InputPar)***: Arregla los índices de la desviación con los parámetros introducidos.

- ***void stitch_func (mangoose_stitch& stichparam, mangoose_cLineParam& InputPar)***: Función de pegado para reconstrucciones de 2 camas.

- ***void stitch_func_alt (mangoose_stitch& stichparam, mangoose_cLineParam& InputPar)***: Función de pegado para reconstrucciones con más de 2 camas para solapes posteriores al realizado la primera vez.

- ***int ROI_index_gen (mangoose_cLineParam& InputPar)***: Creación de los índices VOI.

- ***int read_hounsfield_data( )***: Lee los datos necesarios para la conversion a Hounsfield.

- ***int mangoose(mangoose_cLineParam& InputPar)***: Núcleo de Mangoose para casos no simétricos

Apéndice - Parámetros de entrada

"-f" Ruta de los ficheros de proyecciones (sin el nombre del estudio, sólo la ruta terminada con "\")
"-m" Nombre del caso de estudio
"-w" Factor de la magnificación
"-q" Distancia desde el origen al centro de FOV (SO)
"-h" Fichero con las unidades Hounsfield
"-v" Voltaje inicial (keV)
"-n" número de ficheros ctf
"-p" número de proyecciones por fichero
"-b" número de posiciones de camas
"-g" projectionbinning espaciode interpolación
"-d" número de píxeles en cada proyección {proj_dim_sproj_dim_z}
"-a" Número total de proyecciones (posiciones angulares)
"-e" solape en mm entre 2 camas consecutivas
"-j" Espacio de reconstrucción
"-r" dimension del ROI en los tres ejes
"-o" desplazamiento del ROI en los tres ejes

"-z" second processor
"-i" ánguloinicial
"-t" sentido de la rotación {-1: sentido antihorario, 1: sentido horario}
"-k" generar resultado en flotantes (0: no, 1: si)
"-x" parámetros de corrección por el desalineamiento entre las camasalpha y betadesv_z

Apéndice - Línea de llamada para ejecución

mangoose64.exe **-f** path_projection_files**-m** study_name**–w**magnification_factor**-h** housnfield_file**-v** source_voltage**-n** number_ctf_files**–p**projections_per_file**-b** number_beds**-g** projection_binning**-d** proj_dim_sproj_dim_z**-a** num_angles**-e** bed_overlap**-j** image_uimage_binning_vimage_binning_z**-r** roi_size_uroi_size_vroi_size_z**-o** roi_offset_uroi_offset_vroi_offset_z**–I** init_angle**-t** rotation_dir**-x** alpha beta desv_z

BILIOGRAFÍA

[1] Dennis M. Ritchie (January 1993). "The Development of the C Language". Retrieved 1 January 2008. "The scheme of type composition adopted by C owes considerable debt to Algol 68, although it did not, perhaps, emerge in a form that Algol's adherents would approve of."

[2]Giannini, Mario; Code Fighter, Inc.; Columbia University (2004). "C/C++". In Hossein Bidgoli. The Internet encyclopedia. 1. John Wiley and Sons. p. 164. ISBN 0-471-22201-1.

[3]Patricia K. Lawlis, c.j. kemp systems, inc. (1997). "Guidelines for Choosing a Computer Language: Support for the Visionary Organization". Ada Information Clearinghouse. Retrieved 18 July 2006.

[4]"Programming Language Popularity". 2009. Consultado el 3 de septiembre de 2012.

[5]"TIOBE Programming Community Index". 2009. Consultado el 3 de septiembre de 2012.

[6]Stroustrup, Bjarne (1993). "A History of C++: 1979−1991". Consultado el 3 de septiembre de 2012.

[7]Kernighan, Brian W.; Dennis M. Ritchie (February 1978).The C Programming Language (1st ed.). Englewood Cliffs, NJ:Prentice Hall. ISBN 0-13-110163-3. The first widely available book on the C programming language.

[8]Kernighan; Dennis M. Ritchie (March 1988). The C Programming Language (2nd ed.). Englewood Cliffs, NJ:Prentice Hall. ISBN 0-13-110362-8. http://cm.bell-labs.com/cm/cs/cbook/.

[9]"JTC1/SC22/WG14 – C". Home page. ISO / IEC. Consultado el 3 de septiembre de 2012.

[10]Naugler, David (May 2007). "C# 2.0 for C++ and Java programmer: conference workshop". Journal of Computing Sciences in Colleges 22 (5). "Although C# has been strongly influenced by Java it has also been strongly influenced by C++ and is best viewed as a descendant of both C++ and Java."

[11]Schildt, Herbert (1 August 1998). C++ The Complete Reference(Third ed.). Osborne McGraw-Hill. ISBN 978-0-07-882476-0.

[12]Stroustrup, Bjarne (7 March 2010). "C++ Faq: When was C++ Invented". ATT.com. Retrieved 16 September 2010.

[13]C++ Applications

[14]"What's CvSDL?". http://www.cvsdl.com/: cvsdl. Consultado el 3 de septiembre de 2012. "CvSDL was introduced in 2003 as a C++ class framework with Verilog features that worked like a Verilog simulator. Since then it has been revamped to be a standard-compliant HDL simulator, currently supporting Verilog. It is capable of cosimulating with SystemC. It can be used just as an HDL simulator or to generate executable specifications written in Verilog and SystemC on the hardware side and in C, C++ and SystemC on the software side."

[15]"ISO/IEC 14882:2011". ISO. Consultado el 3 de septiembre de 2012.

[16]Stroustrup, Bjarne. "The C++ Programming Language". Consultado el 3 de septiembre de 2012.

[17]Stroustrup, Bjarne. "The C++ Programming Language". Consultado el 3 de septiembre de 2012.

[18]Voegele, Jason. "Programming Language Comparison". Consultado el 3 de septiembre de 2012.

[19]Bhatti, M. U.; Ducasse, S.; Rashid, A. (June 2008). "Aspect Mining in Procedural Object Oriented Code". Proceedings of the International Conference on Program Comprehension: 230–235. doi:10.1109/ICPC.2008.45. Consultado el 3 de septiembre de 2012.

[20]"Most Popular Programming Languages". Consultado el 3 de septiembre de 2012.

[21]"Bjarne Stroustrup's FAQ – Where did the name "C++" come from?". Consultado el 3 de septiembre de 2012.

[22]"ISO/IEC 14882:2011".

[23]"ISO/IEC TR 19768:2007".

[24]"ISO/IEC 14882:2003".

[25]"ISO/IEC 14882:1998".

[26]http://herbsutter.com/2011/08/12/we-have-an-international-standard-c0x-is-unanimously-approved/
"ISO/IEC 14882:2011". ISO. 2 September 2011. Consultado el 3 de septiembre de 2012.

[27]STLPort home page, quote from "The C++ Standard Library" by Nicolai M. Josuttis, p138., ISBN 0-201-37926-0, Addison-Wesley, 1999: "An exemplary version of STL is the STLport, which is available for free for any platform"

[28]Reinders, James (2007, July). Intel Threading Building Blocks: Outfitting C++ for Multi-core Processor Parallelism (Paperback) Sebastopol: O'Reilly Media, ISBN 978-0-596-51480-8.

[29]Voss, M. (2006, October). "Demystify Scalable Parallelism with Intel Threading Building Blocks' Generic Parallel Algorithms."

[30]Reinders, James (2010, May). TBB 3.0: New (today) Version of Intel Threading Building Blocks. Consultado el 3 de septiembre de 2012.

[31]http://threadingbuildingblocks.org/whatsnew.php Consultado el 3 de septiembre de 2012.

[32]http://openmp.org/wp/2008/10/openmp-tutorial-at-supercomputing-2008/OpenMP Tutorial at Supercomputing 2008

[33]http://openmp.org/wp/2009/04/download-book-examples-and-discuss/ Using OpenMP - Portable Shared Memory Parallel Programming - Download Book Examples and Discuss

[34]http://openmp.org/wp/openmp-compilers/ OpenMP Compilers

[35]http://openmp.org/wp/2008/11/openmp-30-status/ OpenMP 3.0 Status

[36]http://www.jornadassarteco.org/js2012/papers/paper_116.pdf

[37]http://www.xataka.com/componentes-de-pc/knights-corner-de-intel-sigue-creciendo-y-apunta-a-grandes-servidores

[38]http://www.madboxpc.com/intel-revela-destalles-de-la-arquitectura-mic-de-xeon-phi-knights-corner/

[39] ISO 2010: Programming Languages — C++, Draft International Standard, (2010)

http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2010/n3092.pdf