Universidad de Valladolid

E. T. S. Ingeniería Informática

Trabajo Fin de Grado

Grado en Ingeniería Informática

Desarrollo de una interfaz con orientación al objeto para Hitmap

Autor:

Óscar González Ossorio

Tutores:

Dr. Arturo González Escribano

Javier Fresno Bausela

Agradecimientos

A mi familia por apoyarme cuando lo he necesitado. A Arturo, Diego y Javier por dedicar el tiempo y esfuerzo necesario para ayudarme a alcanzar los objetivos. A Alejandro y al grupo Trasgo por ayudarme en mis tareas.

Resumen

La distribución de datos entre diferentes procesos es una de las tareas fundamentales a la hora de desarrollar aplicaciones paralelas para sistemas de memoria distribuida. Hitmap es una biblioteca implementada en C que tiene métodos para distribuir estructuras de datos entre topologías de procesadores y un mecanismo para declarar comunicaciones que se adaptan automáticamente al resultado de la partición. En este trabajo presentamos una primera aproximación para la creación de una versión C++ de esta biblioteca. Nuestro objetivo es eliminar las limitaciones de la interfaz actual aplicando soluciones de los lenguajes orientados al objeto. Este trabajo se centra en la construcción de la interfaz de Hitmap para el manejo de sus estructuras de datos. Incluye tanto el desarrollo de dicha interfaz así como su validación experimental.

Abstract

The data distribution among different processes is one of the main tasks when it comes to develop parallel applications for distributed memory systems. Hitmap is a library implemented in C language that has methods for distributing data structures among processor topologies and it has a mechanism for declaring adaptive automatic communications to the result of a partition. In this work we present a first approximation to the creation of a new C++ version for this library. Our objective is to eliminate the limitations of the current interface by applying solutions of the object-oriented languages. This work focus on the Hitmap's data structures handling interface. It describes the interface development process as well as its experimental validation.

Índice general

Índice de figuras

Índice de cuadros

Parte I

Introducción y Planificación

Capítulo 1

Introducción

Las herramientas clásicas para programar en entornos de memoria distribuida, por ejemplo las bibliotecas de paso de mensajes como MPI [6], ofrecen mecanismos para transmitir los datos. Pero es el programador quien debe ocuparse de realizar la partición de datos, hacer un balanceo de carga adecuado y resolver los problemas de sincronización. Para poder programar este tipo de aplicaciones de forma eficiente, necesitamos sistemas de programación que adapten la estructura de comunicación y sincronización del programa, definida por el programador con abstracciones de alto nivel, en función de los resultados de la partición y de detalles concretos de la plataforma.

La biblioteca Hitmap [11] es una de las propuestas que pretende simplificar la tarea de la programación paralela. Es una biblioteca con funcionalidades de *tiling* jerárquico. El concepto de *tiling* representa la división o partición del dominio del problema en subelementos. Por otro lado, el *mapping* es la asignación de tareas a elementos de cómputo. Hitmap está diseñada para simplificar el uso de una vista global permitiendo la creación, manipulación, distribución y comunicación eficiente de tiles y sus jerarquías. En Hitmap, las técnicas de partición y balanceo de datos son elementos independientes que pertenecen a un sistema de módulos. Los módulos son invocados desde el código y aplicados en tiempo de ejecución según se necesitan para distribuir los datos usando la información interna de la topología del sistema. El programador no tiene que razonar en términos de procesadores físicos. Por el contrario, el programador puede usar patrones de comunicación abstractos para distribuir tiles que se adaptan a las características particulares de la plataforma de ejecución. Por tanto, es sencillo realizar las operaciones de programación y depuración con estructuras complejas de comunicación y sincronización.

Hitmap está implementada en el lenguaje de programación C [13]. El interfaz actual de la biblioteca tiene una serie de limitaciones porque utiliza una aproximación a la orientación al objeto mediante estructuras C junto con funciones y macros asociados. Aunque es posible utilizar técnicas de orientación al objeto en lenguajes estructurados [19], el API no es suficientemente intuitivo para el programador final. Nuestro objetivo es desarrollar una interfaz en el lenguaje C++ [22] para poder aprovechar todas las ventajas de las programación orientada al objeto [9]. Técnicas como la encapsulación, herencia y polimorfismo permiten ocultar los complejos detalles internos del manejo de las estructuras paralelas ofreciendo abstracciones de alto nivel al programador. De esta forma es posible centrarse en los detalles de diseño del programa paralelo.

1.1. Contexto

Existen muchas propuestas clásicas de lenguajes de programación que facilitan la tarea del programador, por ejemplo HPF [14]. La mayoría lleva a cabo transformaciones del código en tiempo de compilación, poseen pocas funciones de "mapping" y generan códigos de difícil adaptación a otros entornos o sistemas. Hitmap [11] soporta una jerarquía de contenedores de datos (Tiles) con dominios densos y dispersos. También permite la construcción de patrones de comunicación que hacen uso de la biblioteca de MPI.

Los modelos basados en PGAS (*Partitioned Global Address Space*) aportan una capa de abstracción para trabajar con sistemas de memoria distribuida y compartida. Estos modelos no proporcionan suficientes herramientas para permitir el paralelismo de procesos jerárquicos en entornos híbridos como Hitmap. Se ha demostrado además que Hitmap alcanza una eficiencia equiparable a UPC [15] reduciendo la complejidad de programación. Chapel [3] es otro ejemplo de PGAS. Proporciona una interfaz modular de mapping. Sin embargo, Hitmap permite además explotar mappings jerárquicos a través de una interfaz común. Así mismo, Hitmap mejora las capacidades de otras bibliotecas jerárquicas tales como HTA [7].

Parray [5] es otro modelo que comporta una interfaz de programación flexible basada en diferentes tipos de arrays en una jerarquía de paralelismo sobre sistemas heterogéneos. Algunos de los inconvenientes de esta propuesta son la separación de la gestión de datos densos y con stride y que las operaciones sobre el dominio de datos no son transparentes. Además, el programador necesita tomar decisiones que afectan a la granularidad y sincronización de los niveles jerárquicos. Hitmap presenta una interfaz más genérica, portable y transparente.

1.2. Motivación

En esta sección presentamos un resumen del uso del paralelismo en la computación.

1.2.1. Computación paralela

Desde el año 2003, en el cual se alcanzó la potencia máxima disipable por aire, el uso del paralelismo en tareas computacionales se ha incrementado en gran medida. Este hito permitió aumentar la potencia de los procesadores sin depender de un aumento en la frecuencia de reloj (que había sido hasta entonces el mecanismo principal de mejora), y la integración de múltiples núcleos en un zócalo de procesadores. Las perspectivas de futuro cercano son claras: el paralelismo es una vía factible para conseguir mejorar el rendimiento de los sistemas informáticos. Los supercomputadores actuales usados en proyectos a gran escala son básicamente clusters gigantescos de procesadores que trabajan en paralelo realizando realizando billones de operaciones de punto flotante por segundo y se comunican a través de redes internas especializadas mucho más rápidas que las redes de área local comunes. Los nodos de estos clusters están formados por multicores (CPUs) o manycores (GPUs). En la programación paralela existen lenguajes específicos para explotar este tipo de arquitecturas. Las alternativas son muchas:

- **Mecanismos de paso de mensajes:** Permiten transmitir información entre nodos de un cluster. Un ejemplo representativo es **MPI** [6], una especificación de una interfaz de biblioteca de paso de mensajes. El estándar MPI es gestionado por el MPI Forum.
- **Herramientas de memoria compartida:** Son APIs para programación multiproceso de memoria compartida y son multiplataforma. Como ejemplos representativos cabe destacar a **OpenMP** [4] y a **Pthreads** [16]. OpenMP es un conjunto de directivas de preprocesador que permiten paralelizar un programa. Los POSIX threads (Pthreads) son una interfaz de programación de hilos de ejecución estandarizada para el lenguaje C.
- **GPGPU:** Consiste en el uso de GPUs (*Graphics Processing Units*) para tareas de computación paralela de propósito general (*General Purpose*). **CUDA** [18] y **OpenCL** [10] son lenguajes de este ámbito de la computación paralela. CUDA es un lenguaje de programación para GPUs del fabricante NVIDIA. OpenCL permite desarrollar paralelismo de datos y tareas y funciona tanto para CPUs como para GPUs.

En el contexto del grupo Trasgo, los miembros del mismo llevamos a cabo tareas y estudios de investigación relacionados con el desarrollo de herramientas que potencien el paralelismo, como Hitmap.

1.3. Objetivos

Los objetivos de este proyecto son los siguientes:

- Obtener conocimientos sobre el manejo de un lenguaje orientado al objeto complejo de alto nivel (C++).
- Desarrollar una interfaz orientada al objeto para una biblioteca de programación paralela (Hitmap), adaptando el código ya existente.
- Realizar un ensayo experimental para probar el funcionamiento de Hitmap.
- Obtener una parte funcional de Hitmap y documentar el proceso de elaboración seguido.

Capítulo 2

Planificación del proyecto

El proyecto se ha completado siguiendo un modelo de planificación ágil [1], con reuniones semanales e hitos que cumplir durante el transcurso del mismo.

2.1. Propuesta inicial

En un principio estipulamos que debía completar 3 partes del proyecto, que se corresponden con los tres grupos de funcionalidades de Hitmap. Las 3 partes debían completarse con la ayuda del profesor y el cotutor del grupo de investigación. Esta proyección pretendía adaptar una gran parte de Hitmap, determinando los hitos de manera difusa y apriorística.

2.2. Desarrollo real

La planificación inicial resultó ser inexacta y en ocasiones difícil de cumplir. Por ello, y ante la imposibilidad de paliar el crecimiento desmesurado del camino crítico, optamos por reducir la carga de trabajo del proyecto, limitando el desarrollo al apartado de Tiling.

El proyecto comenzó en Septiembre de 2014. En Diciembre acabó la implementación de las clases base para el Tiling. La concreción del resto de clases de Tiling se dilató excesivamente en el tiempo debido a mis tareas académicas en la Universidad y a la dificultad de ciertos aspectos de la biblioteca Hitmap.

En Mayo terminó definitivamente la fase de implementación del Tiling, comenzando otra fase orientada a la presentación de un artículo científico en las Jornadas Sarteco de Paralelismo 2015. Este artículo se terminó finalmente y tras muchas iteraciones a mediados de Junio de 2015, siendo aceptado en las Jornadas. La presentación del artículo en el congreso será posterior a la fecha de defensa de este trabajo.

2.3. Artefactos

Los artefactos que se propusieron originalmente son los siguientes:

- Esquema jerárquico de la estructura de Hitmap original: Obtención de un esquema estructural de Hitmap.
- Análisis de funciones en desuso en Hitmap original: Revisión de las funciones a eliminar en Hitmap.
- Clases Shape, Sig y SigShape: Implementación de las clases que forman los Dominios de la biblioteca.
- Clase copyControl: Clase auxiliar.
- Clase Iterator: Clase auxiliar para recorrer un tipo de Shape.
- Clases Tile y BTile: Clases principales a desarrollar en el proyecto.
- Clases de Layout (*): Segundo apartado a desarrollar.
- Clases de Topology (*): Tercer apartado a desarrollar.

(*) Es conveniente destacar que los dos últimos artefactos no han sido completados por falta de tiempo, como se detalla en los subapartados anteriores.

2.4. Tareas

Como se puede observar en el cuadro 2.1, se han producido retrasos en ciertas etapas del proyecto. Estos retrasos han sido causados por el impacto de ciertos riesgos:

- **Retraso en la fase de implementación**
- **Imposibilidad de cumplir los hitos planteados inicialmente**

Los planes de contingencia de estos riesgos se describen en la sección 2.6: “Gestión de Riesgos”

2.4.1. Actividades periódicas

Cada semana hemos realizado una reunión entre Arturo, Javier y yo para resolver dudas, avanzar y marcar objetivos del proyecto. A partir de finales de Mayo de 2015, también se ha realizado una reunión semanal con todos los integrantes del grupo de investigación, para definir las estrategias de actuación a nivel de grupo.

2.4.2. Desglose de tareas del proyecto

A continuación se muestra la planificación detallada de las tareas a lo largo del proyecto en cada etapa del mismo. Las siguientes figuras se han elaborado con la herramienta Microsoft Project 2013. La Fig. 2.1 muestra una vista general de las fases y tareas periódicas del proyecto. En la Fig. 2.2 se listan las tareas de la fase de Análisis. La Fig. 2.3 indica la planificación de la fase de implementación. Las tareas llevadas a cabo para la elaboración del artículo científico se listan en la Fig. 2.4 La fase de pruebas y depuración se muestra en la Fig. 2.5. La Fig. 2.6 muestra el diagrama de Gantt del proyecto.

Tarea	Comentarios	Fecha planificada inicialmente	Fecha de término
Esquema jerárquico de Hitmap original	Obtenido con las herramientas Graphviz y Doxygen	17/09/2014	17/09/2014
Análisis de funciones en desuso de Hitmap original	Se redactaron documentos para dejar constancia del trabajo realizado	24/09/2014	24/09/2014
Análisis preliminar de la clase Shape	Se tomó como referencia la clase HitShape de Hitmap original	26/09/2014	26/09/2014
Implementación de las clases Shape, Sig y SigShape	Fue necesario compaginar el trabajo durante el cuatrimestre para terminar la tarea	03/11/2014	01/12/2014
Adición de la clase CopyControl.	Completado en el tiempo estipulado	03/11/2014	03/11/2014
Implementación de la clase BShape	Se añadieron nuevas funcionalidades a Hitmap	15/12/2014	15/01/2014
Implementación de las clases Tile y BTile	Esta es la parte que más tiempo ha conllevado del proyecto	29/03/2015	20/05/2015
Implementación de la clase Iterator	Utilizada en la clase BTile	15/04/2015	01/05/2015
Elaboración del documento académico para las Jornadas Sarteco 2015	Paper que será presentado en un congreso a nivel nacional	22/06/2015	22/06/2015
Realización de baterías de pruebas y corrección de errores	Última fase del proyecto	16/07/2015	16/07/2015

Cuadro 2.1: Fechas clave del desarrollo del proyecto

	Modo de	Nombre de tarea	Duración	Comienzo	Fin
1		Reuniones semanales con el tutor	221 días	mié 10/09/14	mié 15/07/15
47		Reuniones semanales con el grupo	51 días	jue 14/05/15	jue 23/07/15
59		Análisis y tareas preliminares	10 días	lun 15/09/14	vie 26/09/14
64		Implementación	169 días	vie 26/09/14	jue 21/05/15
71		Elaboración de un documento académico para las JP2015	37 días	vie 01/05/15	lun 22/06/15
75		Pruebas y depuración	41 días	jue 21/05/15	jue 16/07/15

Figura 2.1: Vista global de las tareas del proyecto

		Análisis y tareas preliminares	10 días	lun 15/09/14	vie 26/09/14	
60		Esquema jerárquico de Hitmap	3 días	lun 15/09/14	mié 17/09/14	
61		Análisis de funciones en desuso	3 días	lun 22/09/14	mié 24/09/14	
62		Análisis preliminar de la clase Shape	2 días	mié 24/09/14	vie 26/09/14	61
63		Fin de la fase preliminar	0 días	vie 26/09/14	vie 26/09/14	62;60;61

Figura 2.2: Tareas de análisis preliminar

64		**Implementación**	**169 días**	**vie 26/09/14**	**jue 21/05/15**	
65		Clases Shape, Sig y SigShape	46 días	vie 26/09/14	lun 01/12/14	61;63
66		Clase CopyControl	5 días	mar 28/10/14	lun 03/11/14	
67		Clase Bshape	45 días	sáb 15/11/14	jue 15/01/15	
68		Clases Tile y BTile	122 días	mar 02/12/14	mié 20/05/15	65;67FF
69		Clase Iterator	24 días	mar 31/03/15	vie 01/05/15	
70		Fin de la fase de implementación	0 días	jue 21/05/15	jue 21/05/15	65;66;67;68;69

Figura 2.3: Tareas de implementación

71		**Elaboración de un documento académico para las JP2015**	**37 días**	**vie 01/05/15**	**lun 22/06/15**	
72		Realización de experimentos	21 días	vie 01/05/15	vie 29/05/15	
73		Elaboración del informe	26 días	vie 15/05/15	vie 19/06/15	
74		Entrega final del paper	0 días	lun 22/06/15	lun 22/06/15	72;73

Figura 2.4: Tareas de elaboración del paper

75			**Pruebas y depuración**	**41 días**	**jue 21/05/15**	**jue 16/07/15**	
76			Batería de pruebas de las clases Shape, Sig, SigShape y BShape	16 días	mié 20/05/15	mié 10/06/15	
77			Baterías de pruebas de la clase Tile	14 días	lun 01/06/15	jue 18/06/15	
78			Baterías de pruebas de la clase Btile	19 días	vie 19/06/15	mié 15/07/15	
79			Depuración de errores en clases Tile y BTile	30 días	mié 20/05/15	mar 30/06/15	
80			Fin de la fase de depuración y pruebas	0 días	jue 16/07/15	jue 16/07/15	76;77;78;79

Figura 2.5: Tareas de pruebas y depuración

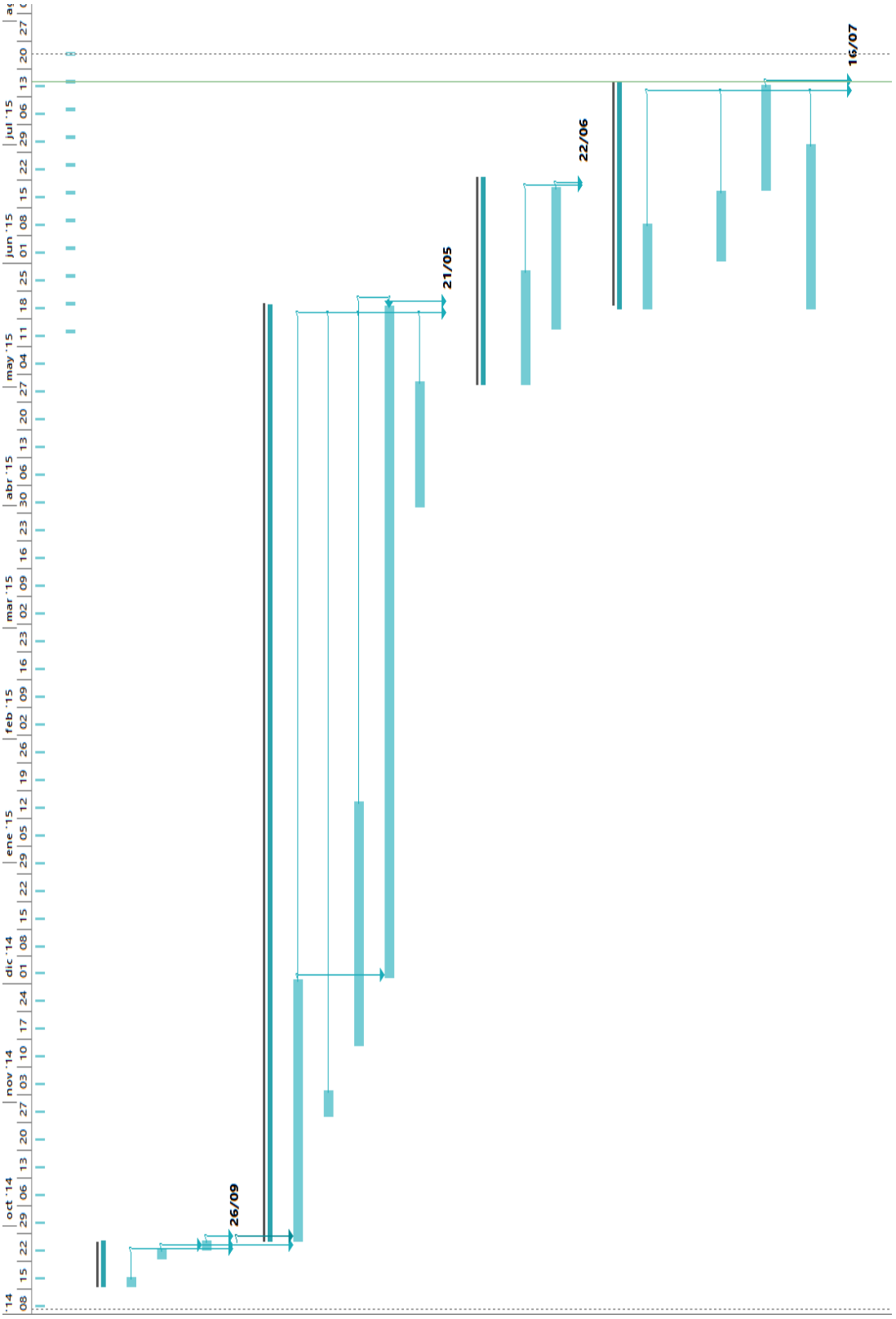


Figura 2.6: Diagrama de Gantt del proyecto

2.5. Gestión de recursos

En este apartado se describen las horas de trabajo dedicadas al proyecto, así como la amortización de los equipos de trabajo.

2.5.1. Análisis de costes

Los costes del proyecto se pueden resumir en horas de trabajo del estudiante y en la amortización de equipos informáticos usados para completar el trabajo.

Horas de trabajo

Las horas de trabajo en el proyecto se dividen en presenciales en el laboratorio y no presenciales (en casa del estudiante). Estimamos que el número de horas de trabajo se han repartido entre 350 horas presenciales y 100 horas no presenciales. En total, 450 horas de trabajo aproximadamente. El horario presencial en el laboratorio ha sido el mismo durante todo el segundo cuatrimestre, de martes a viernes de 09:00 a 14:00 y los lunes de 16:00 a 19:30. Además, durante el primer cuatrimestre he trabajado sobre el proyecto de manera esporádica, debido a la acumulación de asignaturas.

Amortización de equipos

El equipo que se ha amortizado es Chimera, en el que se ha realizado parte de la experimentación. La disponibilidad de esta máquina ha sido importante con vistas a la obtención de resultados válidos y con posibilidad de ser presentados y defendidos.

2.6. Gestión de riesgos

A continuación se detalla cada uno de los riesgos potenciales en el proyecto. Los riesgos aquí listados se subdividen en distintos ámbitos: De planificación, del personal, de proceso, externos y del material de trabajo.

2.6.1. Riesgos identificados

Riesgos de planificación

R-01. Limitación del número de recursos disponibles para una tarea debido a un fallo en la planificación inicial

Fase: Todas
Probabilidad: Variable, dependiente de la fase en la que aparezca el riesgo.

- Inicio: Baja
- Elaboración: Media

- Construcción: Muy alta
- Transición: Media

Magnitud:

- Inicio: Baja
- Elaboración: Media
- Construcción: Muy alta
- Transición: Alta

Descripción: En la planificación inicial se establecen una serie de hitos que deben completarse teniendo en cuenta los recursos disponibles: Tiempo, miembros del proyecto presentes y material de trabajo. Existe una carencia de uno o más de estos recursos en un lapso de tiempo y se produce un crecimiento del camino crítico del proyecto, alejándose la fecha de finalización del mismo.

Impacto: Dependiendo de la fase de afectación del riesgo puede ser mayor o menor el impacto producido, siendo posible reducir en mayor o menor medida el tiempo extra causado por el fallo en la planificación inicial.

Indicadores: No es posible acabar una tarea del proyecto a tiempo debido a la falta de recursos disponibles. Cuando la fecha de finalización de la tarea prevista se ha superado, el riesgo ya es un hecho.

Plan de mitigación: Revisión del estado del proyecto y de los recursos disponibles antes de comenzar cada fase.

Plan de contingencia: Aprovechar las holguras obtenidas en fases previas. De no ser posible, la solución obligada consistirá en ampliar la fecha de finalización del proyecto.

R-02. Decisiones técnicas incorrectas en etapas críticas del proyecto

Fase: Construcción y Transición.
Probabilidad:

- Construcción: Media
- Transición: Baja

Magnitud:

- Construcción: Muy alta
- Transición: Alta

Descripción: Durante la implementación de un módulo del proyecto, se toma una decisión de diseño que resulta ser muy compleja. La previsión inicial no tenía en cuenta un retraso en el camino crítico que, a pesar de todo, se produjo.

Impacto: Puede afectar en gran medida a la duración de una o más tareas, llegando en el peor caso a prolongar la fecha de finalización del proyecto.

Indicadores: Se detecta un problema de complejidad demasiado elevada en una tarea, debido a la imposibilidad de terminarla en el tiempo estipulado.

Plan de mitigación: Tomar decisiones razonadas y consensuadas con los tutores del proyecto para evitar realizar tareas excesivamente complejas que alarguen el camino crítico.

Plan de contingencia: La única solución posible es alargar la fecha de término de la tarea y/o eliminar componentes del producto final (solo en situación de tiempo crítica). Volver a planificar y rehacer la tarea conllevaría un retraso mayor aún

R-03. Imposibilidad de cumplir los hitos planteados inicialmente (eliminación de tareas)

Fase: Construcción

Probabilidad: Alta

Magnitud: Muy alta

Descripción: En la planificación inicial se determinaron todas las fases de implementación a acometer durante el transcurso del proyecto. Sin embargo, por problemas de tiempo no pudieron completar todas las etapas.

Impacto: El impacto sobre el proyecto es muy alto, debido a que afecta directamente al volumen de trabajo final, junto con los resultados que lleva aparejados.

Indicadores: La duración de una de las tareas se extiende demasiado, impidiendo la finalización de otras en el tiempo previsto.

Plan de mitigación: Establecer prioridades en las fases a realizar, para completarlas en orden de importancia.

Plan de contingencia: En el caso de que la fecha de finalización esté próxima, se debe intentar mejorar lo máximo posible los módulos ya completados, y evitar dejar partes incompletas y no funcionales.

Riesgos del personal

R-04. Dispersión del grupo (impide realizar reuniones semanales)

Fase: Todas

Probabilidad: Baja

Magnitud: Alta

Descripción: Pueden existir problemas que provoquen dispersión entre los miembros del grupo, dificultando las revisiones periódicas. Esto puede afectar a la finalización de una o más tareas, debido a las dudas no resueltas por el tutor.

Impacto: El impacto sobre el proyecto es medio, ya que este riesgo afecta directamente al tiempo necesario para completar las tareas, pero tiene una probabilidad baja de ocurrencia.

Indicadores: Un miembro del grupo no puede acudir a una reunión semanal.

Plan de mitigación: Establecer trabajo futuro de cara a las dos siguientes semanas, para evitar desorientación con el trabajo a seguir.

Plan de contingencia: Seguir trabajando en las tareas pendientes y establecer una reunión extraordinaria en la siguiente fecha libre.

R-05. Baja por enfermedad de un miembro del equipo

Fase: Todas

Probabilidad: Baja

Magnitud: Alta

Descripción: Un miembro del equipo causa baja de unos días por enfermedad.

Impacto: Alto, dependiente del número de días de baja.

Indicadores: El miembro del equipo comunica que se encuentra enfermo.

Plan de mitigación: Establecer trabajo futuro con vistas a las dos siguientes semanas. De esta forma se tiene trabajo planificado aún faltando el tutor.

Plan de contingencia: Solicitar ayuda necesaria del proyecto al co-tutor.

R-06. Falta de conocimientos y automatismos del estudiante

Fase: Todas

Probabilidad: Media

Magnitud: Alta

Descripción: El estudiante carece de automatismos para realizar una determinada tarea compleja. Esto provoca un retraso en la fecha de finalización del proyecto.

Impacto: El impacto es alto, debido a que el estudiante necesitará tutorías individualizadas, alargando de esta manera el tiempo de finalización de las tareas.

Indicadores: El estudiante muestra grandes dificultades al realizar una tarea determinada.

Plan de mitigación: Planificar las tareas a realizar en base a los conocimientos previos del estudiante.

Plan de contingencia: En el peor caso, no quedará más solución que alargar plazos. De cualquier forma, el tutor deberá tratar de solucionar las dudas y problemas del estudiante.

Riesgos de proceso

R-07. Retraso en la fase de implementación

Fase: Implementación

Probabilidad: Alta

Magnitud: Alta

Descripción: Durante la fase de implementación surgen problemas que dificultan la finalización de las tareas en el tiempo planificado.

Impacto: Alto, dado que la fase de implementación es la más importante y el proyecto se basa en completarla correctamente.

Indicadores: Aparecen retrasos en las tareas de implementación.

Plan de mitigación: Establecer hitos en la planificación con avances acumulativos pequeños, para evitar retrasos en el proyecto, y permitir un avance lento pero continuo.

Plan de contingencia: Reducir el contenido a implementar o alargar la fecha de finalización del proyecto.

R-08. Pruebas de software con resultados incompletos o erróneos

Fase: Implementación

Probabilidad: Alta

Magnitud: Alta

Descripción: Una vez implementados los módulos del proyecto, se realizan las pruebas de software correspondientes. Dichas pruebas muestran resultados incorrectos, lo que provoca una re-implementación de los módulos afectados.

Impacto: Muy alto, ya que requiere volver a revisar y depurar el código.

Indicadores: Un test de prueba devuelve un resultado incorrecto.

Plan de mitigación: Revisar el código implementado a un nivel bajo de granularidad para evitar la proliferación de errores.

Plan de contingencia: La única solución posible es aprovechar las holguras en fases previas para completar la corrección de errores en el código.

Riesgos externos

R-09. Ataque desde el exterior - DoS

Fase: Todas

Probabilidad: Muy baja

Magnitud: Muy Alta

Descripción: Un atacante provoca una denegación de servicio a las máquinas del laboratorio, lo que impediría su utilización temporalmente.

Impacto: Alto, ya que provocaría un retraso en el proyecto hasta que los técnicos se hayan ocupado del problema.

Indicadores: Interrupción en el funcionamiento de las máquinas de prueba.

Plan de mitigación: Establecer un firewall robusto frente a ataques externos.

Plan de contingencia: Notificar la incidencia a los técnicos para que se ocupen de resolverla lo antes posible.

R-10. Eventos inesperados (Terremotos, inundaciones, incendios)

Fase: Todas

Probabilidad: Muy baja

Magnitud: Muy alta

Descripción: Un incendio puede destruir el lugar de trabajo, así como las máquinas de pruebas.

Impacto: Bajo, ya que la probabilidad de que ocurra un incendio es muy reducida.

Indicadores: No existen indicadores frente a este tipo de eventos inesperados.

Plan de mitigación: Poseer un plan de evacuación frente a emergencias.

Plan de contingencia: Retrasar la fecha de finalización del proyecto de manera ineludible.

Riesgos del material de trabajo

R-11. Formateo del sistema operativo de las máquinas de experimentación

Fase: Todas

Probabilidad: Baja

Magnitud: Baja

Descripción: El formateo de las máquinas de experimentación se realiza para configurar el sistema de colas para la experimentación.

Impacto: Bajo. El tiempo que las máquinas no estarán operativas es muy reducido.

Indicadores: Necesidad de reconfigurar una máquina de experimentación.

Plan de mitigación: Utilización de otras máquinas de experimentación para las pruebas.

Plan de contingencia: Utilización de otras máquinas de experimentación hasta que se ha configurado la nueva.

R-12. Fallos del hardware

Fase: Todas

Probabilidad: Muy baja

Magnitud: Alta

Descripción: Las máquinas de experimentación están dispuestas en un cluster distribuido con un sistema de colas preparado para procesar las tareas que se envíen al mismo. El riesgo se concreta si una de las máquinas de experimentación sufre un fallo de hardware que requiere de una sustitución de componentes.

Impacto: Medio. Se pueden seguir usando las máquinas que funcionan correctamente.

Indicadores: Una de las máquinas de experimentación sufre fallos repetidos o cesa completamente su funcionamiento.

Plan de mitigación: Poseer redundancia de componentes y máquinas de experimentación.

Plan de contingencia: Utilizar otras máquinas hasta que se haya reparado o sustituido el componente averiado.

R-13. Cambio de versión del software en uso

Fase: Todas

Probabilidad: Baja

Magnitud: Alta

Descripción: Una de las herramientas principales que se utilizan en la experimentación es actualizada en una máquina del cluster o en la máquina del estudiante y la versión difiere de

forma notoria. Esto implica diferencias en el rendimiento que no pueden ser valoradas de la misma forma, dado que la versión del software no es la misma.

Impacto: Alto. Es necesario equiparar o equilibrar el software de las máquinas de experimentación lo máximo posible.

Indicadores: Se produce la actualización de uno o más componentes software en las máquinas de experimentación.

Plan de mitigación: Utilizar versiones estables de los elementos software y actualizar de manera planificada y a largo plazo.

Plan de contingencia: Valorar las posibles diferencias entre una versión y otra, y en el caso de que las diferencias sean notables, equilibrar las versiones de software en la medida de lo posible.

Parte II

Preliminares

Capítulo 3

Características de Hitmap (C)

3.1. Conceptos

En los siguientes apartados se describen conceptos básicos necesarios para comprender el trabajo desarrollado, además de conceptos propios de C++.

3.1.1. Conceptos básicos

Programación paralela

La programación paralela [12] consiste en el manejo de estructuras de datos y de unidades de procesamiento para distribuir la carga de trabajo correspondiente al programa en secuencial entre diferentes nodos. Es necesario tener en cuenta que la carga de trabajo total en un sistema paralelo se ve incrementada debido a la sobrecarga de paralelización, al acceso a datos entre diferentes procesadores y a la sincronización de procesos. El particionado de datos entre los distintos procesos hay que realizarlo distribuyendo porciones de una granularidad apropiada y que reduzcan al mínimo las comunicaciones con otros procesos.

Paralelismo de datos

Consiste en ejecutar la misma tarea sobre diferentes datos de una misma estructura. Incluye tareas de partición y distribución (datos fijados a elementos de proceso). Se realizan comunicaciones entre cada una de las particiones de la estructura. Las particiones se pueden realizar de distintas formas, variando el volumen de comunicaciones y el número y tamaño de las particiones.

Orientación al objeto

La orientación al objeto se basa en el uso de "objetos" (estructuras de datos que contienen datos y código) y clases (plantillas para crear objetos de un tipo determinado). En la Programación Orientada al Objeto (POO), existe una funcionalidad especial que permite a un objeto modificar sus datos a partir de sus métodos de objeto (this). Las variables de un objeto se pueden empaquetar protegiendo sus métodos (encapsulamiento). El encapsulamiento proporciona los siguientes beneficios a los desarrolladores de software:

- Modularidad: el código fuente de un objeto puede ser gestionado de manera independiente al código fuente de otros objetos. Un objeto puede ser transferido en el sistema sin modificar su estado y comportamiento.
- Ocultamiento de información: el objeto posee una interfaz pública que pueden utilizar otros objetos para comunicarse. Pero es posible mantener métodos y datos privados que son manejados únicamente por su objeto poseedor.

Este trabajo ha consistido principalmente en construir una interfaz orientada al objeto a partir de una biblioteca de programación paralela escrita en C. La versión original fue implementada basándose en la orientación al objeto, pero con las limitaciones propias de un lenguaje no orientado al objeto.

El principal objetivo de aplicar la orientación al objeto en este proyecto es conseguir una interfaz más entendible, manejable y mantenible.

3.1.2. Conceptos de C++

En esta sección se van a tratar conceptos directamente relacionados con C++ [22], en concreto con optimizaciones relativas a la declaración de métodos.

Templates

En C++ existen las plantillas o "templates" que sirven para crear uno o varios tipos de datos genéricos dentro de una clase determinada para poder construir diferentes objetos de la misma clase que utilizan distintos tipos de datos. En este proyecto los templates son útiles para poder definir las estructuras de datos distintos (tiles de enteros, doubles, float, char, etc.) que se repartirán entre los procesadores de una topología.

Parámetros por defecto

Cuando en C++ se declara un parámetro por defecto (u opcional) se determina que, en el caso de que se invoque al método sin pasarle el parámetro por defecto, este pasara a tener el valor que se le haya asignado por defecto en la declaración del método. Por tanto, el parámetro es "opcional" en el sentido de que no es obligatorio usarlo en la invocación al método. Solo pueden ser definidos como parámetros por defecto los últimos argumentos del método. Es decir, no es posible definir el primer parámetro como opcional y el resto como no opcionales, ya que en esta situación sería imposible para el compilador entender la invocación al método.

```
int f(int x, int y = 0, int z = 0){..}
```

Inlining

En C++ también es posible definir un método como "inline". Esto supone que se sustituye cada llamada al método por su propio código. Esto puede suponer una mejora en el tiempo de ejecución notable, ya que la llamada se resuelve de manera mucho más rápida. Sin embargo, la

principal contrapartida es el aumento del tamaño del código, siendo más grande cuanto mayor sea el cuerpo del método "inline". Las funciones inline recursivas hacen imposible garantizar que en cada llamada a una función inline realmente se realice el "inlining". Además, es necesario distinguir cuándo se debe realizar o no el "inlining", dado que si el tamaño del código del método es muy grande, lo más probable es que esta técnica no resulte efectiva, o que incluso provoque una pérdida de rendimiento.

```
inline int f(int x, int y, int z){..}
```

3.2. Versión antigua de Hitmap

3.2.1. ¿Qué es Hitmap?

Como ya se ha descrito en la Introducción, Hitmap es una biblioteca de programación paralela orientada al objeto, que incorpora funcionalidades de particionado, mapeo y distribución de datos en "Tiles" jerárquicos. Hitmap pretende simplificar la tarea de la programación paralela.

3.2.2. Estructura de Hitmap

La biblioteca se puede dividir en 3 secciones:

- **Funciones de particionado (tiling):** Definición y manipulación de arrays y tiles. Estas funciones se pueden usar de manera independiente al resto de funcionalidades de Hitmap, para mejorar la localidad en el código secuencial, así como para generar distribuciones de datos manualmente para la ejecución en paralelo.
- **Funciones de distribución (mapping):** Distribución de datos y métodos de layout para particionar dominios automáticamente, dependiendo de la topología virtual seleccionada. Estos métodos están orientados a la distribución de datos y tareas en entornos paralelos. Estas funciones devuelven (1) los rangos de los tiles que necesitan ser creados localmente, (2) la distribución entre tiles y procesadores virtuales y (3) la información de los vecinos, encapsulada en una única estructura.
- **Funciones de comunicaciones:** Creación de patrones de comunicación reutilizables para tiles distribuidos. Estas funciones son una abstracción del modelo de paso de mensajes para comunicar tiles entre procesadores virtuales, y puede ser usada con la información de distribución para crear patrones de comunicación dependientes del mapping.

En la Fig. 3.1 se muestra la estructura de la biblioteca Hitmap (En negro, clases de tiling. En blanco, clases de mapping. En gris, clases de comunicaciones).

En el cuadro 3.1 y se muestra la antigua API de Hitmap.

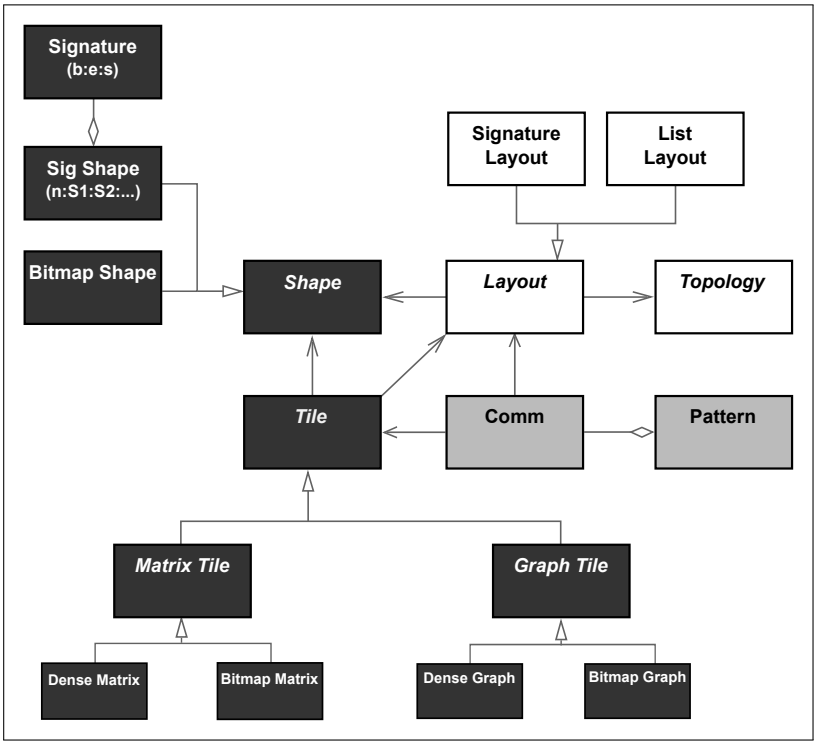
Descripción de los módulos

En este apartado se describen los elementos básicos para la gestión de estructuras de datos tipo array en Hitmap: Dominios y Tiles.

Cuadro 3.1: Clases y métodos de la API antigua de Hitmap.

Objeto	Método	Descripción
HitShape	`hit_shape(dims, [begin,end,stride]*)`	Constructor de una estructura Shape.
Topología	`hit_topology(name, plugin)`	Constructor de topologías. Este constructor crea un nuevo objeto topología usando el plugin seleccionado para estructurar los procesadores disponibles en una topología virtual. Podría ser uno de los plugins predefinidos por Hitmap o uno definido por el usuario.
Layout	`hit_layout(name, topo, shape)`	Un constructor de layouts determina la distribución de datos de un shape sobre una topología virtual. De la misma forma que con los objetos Topology, Hitmap ofrece muchos plugins predefinidos.
	`hit_layShape(layout)`	Este método devuelve el shape local asignado a un procesador dado, o el shape del procesador actual.
Comm	`hit_comType(hitTile, tile_type)`	Crea un nuevo objeto de comunicación MPI para un tile.
	`hit_comDo(comm)`	Realiza la comunicación encapsulada por la estructura comm.
Pattern	`hit_pattern(type)`	Crea un nuevo patrón de comunicación, que puede ser ejecutado en orden o fuera de orden.
	`hit_patternAdd(pattern, comm);`	Añade una nueva comunicación al patrón.
	`hit_patternDo(pattern)`	Realiza las comunicaciones del patrón.
Tile	`hit_tileDomainShape(tile, tile_dataType, shape))`	Constructor del Tile. Crea un objeto tile usando el dominio definido por un objeto shape. Es necesario indicar el tipo de datos a utilizar.
	`hit_tileAlloc(tile)`	Reserva memoria para el tile
	`hit_tileElemAt(tile, dims, ...)`	Método para acceder al elemento en el tile.
	`hit_tileSelect(newtile, oldTile, shape)`	Método para crear un subtile.

Figura 3.1: Estructura de Hitmap



Un **dominio** es una variable (objeto en C++) que proporciona una cardinalidad (número de dimensiones) y un rango de índices abarcado. Los tipos de dominios existentes en Hitmap son los siguientes:

Signaturas

Las signaturas son tuplas de 3 enteros que representan un subespacio de índices de arrays en un dominio unidimensional. Las 3 variables enteras indican el primer índice (comienzo) de la signatura, el último índice (final) y el salto entre celdas de la misma. La siguiente fórmula define los índices representados por una signatura:

$$S < b, e, s >= \{i : b \leq i \leq e, (i - b) \bmod s = 0\}$$

Shapes

Los Shapes son los elementos que definen un dominio multidimensional y sirven de base para los Tiles:

Shapes de signaturas

Un Shape de signaturas (SigShape) es una n-tupla de signaturas. Representa una selección de un subespacio de índices de arrays en un dominio n-dimensional. Los Shapes tienen cardinalidad, entendida como el número de combinaciones de los índices en el dominio.

Tiles

Un Tile es un array de n dimensiones (de 1 a 4) cuyo dominio se define por un SigShape.

Los Tiles relacionan los datos que contienen sobre el espacio definido por su Shape y poseen una ordenación jerárquica en base al número de subselecciones que se han realizado sobre el Tile original. La razón de la inclusión de un soporte para una jerarquía de Tiles es la utilidad de la misma en el reparto, localización y acceso eficiente a los datos entre diferentes procesadores.

Además, se gestiona el estado de la memoria asignada para controlar la destrucción de los elementos jerárquicos. La memoria se asigna de forma contigua. Existen métodos para modificar su estructura, para acotar porciones seleccionadas, acceso a elementos, etc.

Ejemplo de métodos de tiling en Hitmap antiguo

En Hitmap antiguo no se dispone de templates para definir el tipo del Tile, por lo que se usa el macro "hit_tileNewType()", cuyo uso es más incómodo y engorroso. En la figura 3.2 se puede ver un ejemplo de uso de métodos de acceso a elementos del Tile, y de un método de construcción para un Tile de 2 dimensiones.

En la figura 3.3 se muestra un ejemplo de uso de métodos del Tile antiguo.

Figura 3.2: Ejemplo de acceso y manejo del tile: C

```
 // Ejemplo interfaz C
hit_tileNewType(double);
...
HitTile_double A;
hit_tileDomain(&A, double, 2, n, m);
for(...){
  hit_tileElemAt(A,2,i,j) = ;
}
hit_tileFree( A );
```

Figura 3.3: Ejemplo de métodos básicos del tile: C

```
// Creación de los Shapes
HitShape shape1 = hit_bitmapShapeMatrix(n,m);
HitShape shape2 = hit_bitmapShapeMatrix(x,y);

// Declaracion del Tile
HitTile_double tile1;

// Uso del constructor del Tile
hit_tileDomainShape(&tile1, double, shape1);

// Declaración de un tile vacío para realizar la subselección
HitTile_double tile2;

// Subselección a partir de otro dominio
hit_tileSelect(&tile2, &tile1, shape2);

// Reserva de memoria para el tile2
hit_tileAlloc(&tile2);

// Zona de computación
for(...){
  hit_tileElemAt(tile2, 2, i, j) = ;
}

// Liberación de memoria reservada
hit_tileFree(tile2);
```

Topologías y Layouts

Las clases abstractas Topology y Layouts son interfaces para dos sistemas de plug-ins distintos. Estos plugins se seleccionan mediante su nombre al invocar al constructor. Los plugins

pueden ser reutilizados y creados con reglas distintas para otros programas. Los plugins de Topology implementan funcionalidades simples para organizar procesadores en topologías virtuales con relaciones de vecindad. Los plugins de Layout permiten distribuir un Shape entre los procesadores de una topología. Hitmap tiene diferentes técnicas de particionado y equilibrado de carga implementadas como plugins del Layout. El objeto Layout resultante contiene información sobre la parte local del dominio, relaciones de vecindad y métodos para localizar los otros subdominios. Los plugins de Topology y Layout pueden marcar algunos procesadores como inactivos, por ejemplo cuando existen más procesadores en la topología que índices a distribuir.

Comunicaciones y Patrones de comunicación

La clase Communication representa la información para sincronizar o comunicar tiles entre procesos. La clase proporciona varios constructores para crear diferentes esquemas de comunicación, en términos de dominios de tiles, información de los objetos del layout y reglas de los vecinos de la topología. Esta clase encapsula comunicaciones punto a punto, desplazamientos a lo largo de un eje de la topología, comunicaciones colectivas, etc. Los objetos Communication se pueden componer en patrones reutilizables (Pattern) para realizar varias comunicaciones relacionadas en una sola llamada.

Parte III

Propuesta de solución

Capítulo 4

Características de Hitmap (C++)

En este trabajo hemos construido un conjunto de clases en C++ a partir de la interfaz de biblioteca en C ya existente para el soporte de estructuras de datos tipo array (ver clases en Fig. 3.1). Esta biblioteca permite manejar datos usando contenedores (Tiles) con dominios configurables (Shapes). La parte construida en C++ tiene como objetivo simplificar el manejo de las funciones de la biblioteca (POO), así como facilitar el mantenimiento de la misma. En el apartado de experimentación se comentarán con más detalle aspectos relativos al comportamiento de los compiladores de C++ en diferentes entornos y se mostrará un estudio sobre la eficiencia y el rendimiento de las soluciones propuestas.

En la nueva versión de Hitmap hemos incluido dos componentes nuevos, un dominio (BShape) y un tipo de Tile (BTile). Estas clases encapsulan las estructuras y funciones que existen en la versión antigua de Hitmap. En la versión en C se incluyen algunas funcionalidades del BTile separadas de las del Tile, pero otras se entremezclan indistintamente. Este tipo de problemas se han solucionado en la nueva versión de Hitmap al encapsular el código en clases orientadas a objetos.

Un Shape de mapa de bits (**BShape**) se crea a partir de un Shape de Signaturas de cardinalidad 2 (coordenadas x e y) y representa de una forma compacta los índices de arrays dispersos que contienen en sus celdas valores cero, o diferente de cero. Para ello se usa internamente una matriz de booleanos (Bitmap) como estructura auxiliar. Un **BTile** es una matriz con un dominio acotado por un BShape y posee la misma interfaz que el Tile normal.

4.1. Estructura interna

4.1.1. Dominios

Hemos decidido organizar los dominios de la biblioteca de C++ en clases (Sig, SigShape y BShape) que heredan de otra abstracta (Shape). Cada clase posee métodos de construcción que inicializan los campos del objeto. En el caso de la clase Sig, es necesario inicializar los datos begin, end y stride de la signatura, mientras que en las clases BShape y SigShape se inicializan tantas signaturas como número de dimensiones se solicite para el dominio (entre 1 y 4).

4.1.2. Clase Tile

Esta clase utiliza por debajo un shape de signaturas para determinar el dominio del espacio de datos. Los campos definidos como structs en C se transforman en objetos de tipo Tile<T>, siendo T una variable de tipo genérico. La inclusión de métodos de tipo "getters" y "setters" para campos determinados permiten declarar los campos como privados y mantener la coherencia en el estado interno del objeto.

Métodos de creación

Hemos implementado un constructor para crear objetos tile al estilo de C++. Estos métodos de creación devuelven un Tile con sus campos inicializados a un valor dependiente de los parámetros del constructor. Se puede ver un ejemplo de uso del constructor del Tile en la línea 3 de la Fig. 4.1:

```
Tile<double>A(n,m)
```

Figura 4.1: Ejemplo de acceso y manejo del tile: C++

```
1 // New C++ interface
2 // No need to declare a new tile type
3 Tile<double>A(n,m);
4 for(...){
5   A.elem(i,j) = ; // No need to define the number of dimensions
6 }
7 // Destructor is called automatically
```

Métodos de manejo de memoria

Solo es necesario que los Tiles tengan memoria reservada antes de introducir los datos, siendo posible declarar el dominio de un Tile sin reservar aún la memoria.

La asignación de memoria se lleva a cabo mediante un método que reserva el tamaño necesario para almacenar el array de datos de manera dinámica. Ejemplo en la línea 15 de la Fig. 4.2:

```
tile2.alloc()
```

Se incluye un método destructor que libera los recursos del tile. Existe una función de clonado que permite obtener una copia de un Tile cuya memoria se reserva de forma independiente.

Existe la posibilidad de realizar subselecciones de Tiles. Una subselección es otro Tile con un dominio reducido que se construye sobre otro Tile. El acceso a través del dominio local de la subselección se dirige a los mismos datos (posiciones de memoria) del tile original. Esta funcionalidad es útil para particionar los datos y repartirlos entre diferentes procesos. Las dos opciones que hay se diferencian en la base de coordenadas usada para realizar la subselección:

Figura 4.2: Ejemplo de métodos básicos del tile: C++

```
// Creación de los Shapes
BShape shape1 = BShape(n,m);
BShape shape2 = BShape(x,y);

// Uso del constructor del BTile
BTile<double> tile1 ( shape1 );

// Declaración de un tile vacío para realizar la subselección
BTile<double> tile2;

// Subselección a partir de otro dominio
tile1.select( &tile2, shape2 );

// Reserva de memoria para el tile2
tile2.alloc();

// Zona de computación
for(...){
  tile2.elem(i,j) = ;
}

// Se llama al destructor automáticamente al acabar el programa
```

Base con coordenadas de array (absoluta) y base con coordenadas de Tile (relativa al subdominio construido sobre el Tile padre).

El término "base absoluta" implica que las posiciones indicadas en el dominio se corresponden directamente con cada posición del array de datos. La base relativa toma como referencia las posiciones del subtile, en orden relativo, que se traducen a coordenadas absolutas del array automáticamente.

Métodos de acceso a elementos

Existen varios métodos de acceso a los elementos de un Tile:

- Acceso sin tener en cuenta el stride: Versión optimizada del método de acceso para Tiles sin stride.
- Acceso en base a coordenadas de array: Se tiene en cuenta el stride del Tile y se accede a la posición absoluta (con respecto al array).
- Acceso en base a coordenadas de Tile: Se accede a los elementos del Tile padre teniendo en cuenta la posición relativa en el Tile hijo. Si el tile no tiene padre (memoria reservada para él), las coordenadas de array y tile son las mismas.

Para los métodos de acceso se han considerado varias opciones de implementación:

1. **Métodos con parámetros por defecto para varias dimensiones:** Los métodos de acceso a elementos del tile con parámetros por defecto pretenden unificar en un solo bloque de código todas las llamadas para cada número de dimensiones.

```
T & elemAt(int pos1, int pos2 = 0, int pos3
     = 0, int pos4 = 0){..}
```

2. **Métodos y funciones inline:** En C++ es posible definir métodos como inline. Esto es útil para casos como este en los que el código es pequeño y se puede optimizar. En nuestra propuesta hemos implementado un método inline para cada número de dimensiones. De esta forma, el volumen de código es 4 veces mayor que con parámetros por defecto.

```
inline T & elemAt(int pos1, int pos2,
      int pos3, int pos4){..}
```

3. **Macros al estilo C:** Para probar el rendimiento de los métodos de acceso de una tercera forma hemos optado por usar macros al estilo C. Para implementar estos macros hemos creado una macro-función para cada número de dimensiones, equivalente a cada método de la opción anterior. Esta solución es la que se usa en la versión original de Hitmap.

4.1.3. Clase BTile

Tiene una estructura de atributos equivalente a la de la clase Tile, incluyendo además soporte para grafos. BTile utiliza un BShape cuyos datos se almacenan en un vector de booleanos [17]. Este contenedor ahorra espacio en memoria al ocupar cada dato un bit. El problema del vector de booleanos es que el tiempo de ejecución es mayor que en otras implementaciones (std::vector<char>, boost::dynamic_bitset, bm::vector<>, Qt::QBitArray) [17].

Métodos de creación

Los métodos de creación son análogos a los del Tile, teniendo en cuenta los nuevos atributos y el manejo de grafos.

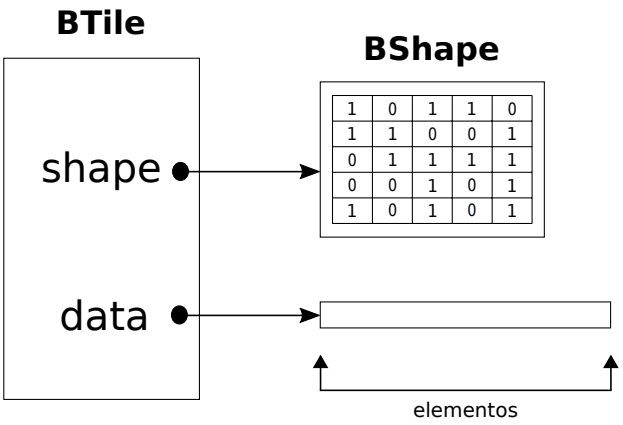
Métodos de manejo de memoria

Los BTiles utilizan un método de reserva de memoria destinado al uso de grafos, almacenando memoria también para el número de vértices en el caso de construirse como un grafo. Se mantienen los mismos métodos de reserva y destrucción del Tile.

Métodos de acceso a elementos

Además de los métodos usados en los Tiles, también se incluyen otros de acceso a elementos en grafos.

Figura 4.3: Diagrama de un BTile



4.2. Problemas resueltos

A lo largo del desarrollo del proyecto, han surgido distintos problemas de implementación que ha sido necesario solucionar. Debido a mi inexperiencia con C++, estos problemas han sido relevantes.

4.2.1. Liberación de memoria por duplicado *(double free or corruption)*

Este problema surge cuando se realiza una subselección de un tile y no se reserva memoria para el nuevo tile resultante. De este modo, si se libera la memoria de los dos tiles con un "delete", se produce un error en tiempo de ejecución del tipo *double free or corruption*. La solución para este problema es simple. En el caso de que se necesite usar la memoria del tile subseleccionado, se realiza la reserva de memoria con *alloc()* y se liberan los dos objetos tile. Si se realiza la subselección sin reservar memoria, basta con liberar la memoria del objeto "padre" (el tile primitivo sobre el que se ha realizado la subselección).

4.2.2. Asignación frente a *memcpy*

Al realizar una copia de un objeto con *memcpy*, se duplica exactamente su contenido. Sin embargo, en el caso de que exista un conteo de las copias que se crean del objeto a lo largo del programa, y no se realiza una asignación mediante el operador "=" (que contiene el código de gestión de copias del objeto), se producirá un fallo de segmento cuando llegue el momento de la destrucción del objeto. Este problema se resuelve fácilmente con un enfoque orientado al objeto. Sin embargo, como ha ocurrido en este caso, si no se aplican las características de C++ y se recurre a la forma clásica de C de copiar un objeto, se incurrirá en este error, que es fácil de resolver pero difícil de diagnosticar.

4.3. Métodos

En el cuadro 4.1 se muestran los métodos principales de Hitmap en su versión en C++.

4.3.1. Métodos de creación

- **Constructor vacío:** Se inicializan los atributos y el Shape del Tile a valores por defecto.
- **Constructor copia:** Este método de creación se invoca al realizar una declaración y asignación del Tile en la misma sentencia.
- **Constructor con número de dimensiones:** Inicialización del Shape del Tile en base al número de dimensiones pasado como argumento.
- **Constructor con lista de argumentos:** Realiza las mismas operaciones que el constructor con número de dimensiones, pero le da valores al campo "data". En el BTile, el BShape se inicializa con el mapa de bits a 0, lo que provoca que deba cambiarse el mapa de bits para que las actualizaciones entre niveles jerárquicos funcionen.
- **Operador de asignación:** Este constructor se utiliza cuando se realiza una asignación de un Tile a otro. La sentencia solo incluye la asignación, no la declaración.
- **Operador de asignación con una variable:** Se inicializan todos los valores del campo de datos al valor de la variable pasada como argumento.
- **Tile: Constructor con SigShape:** Se asignan los valores del Shape pasado como argumento al shape del Tile.
- **BTile: Constructor con BShape:** Se realizan las mismas asignaciones que en el constructor con SigShape del Tile, para inicializar el SigShape del BShape. Adicionalmente, se inicializan los atributos propios del BShape, asignando solo los valores correspondientes del campo "data' en base al stride del BShape pasado como argumento.

4.3.2. Métodos de manejo de memoria

- **Método para reservar memoria: alloc.** Reserva memoria para un tile en base al tamaño total determinado por el constructor correspondiente.
- **Método de clonado:** Crea un tile con su array de datos idéntico a los de otro que se utiliza como base.
- **Destructor:** Elimina los punteros del tile con memoria reservada y libera la memoria del propio tile. Existe un destructor recursivo, para liberar la memoria de tiles jerárquicos.
- **Método de subselección de tiles:** Es posible realizar una selección del contenido de un tile para obtener un fragmento de un tile mayor. Este fragmento o subselección puede ser utilizado reservando memoria para él o sin hacerlo, modificándose en este último caso el tile sobre el que se realiza la selección.

Cuadro 4.1: Clases y métodos de la API nueva de Hitmap.

Objeto	Método	Descripción
SigShape	`SigShape(dims, [begin,end,stride]*)`	Constructor del shape de Signaturas. Un SigShape se define por una selección de índices en cada una de sus dimensiones.
BShape	`BShape(numRows, numCols)`	Constructor del shape de bitmap. Un BShape se define por un Shape de Signaturas de cardinalidad 2 y representa los índices de arrays dispersos que contienen en sus celdas valores cero, o diferente de cero.
Topología	`Topology(plugin)`	Constructor de topologías. Este constructor crea un nuevo objeto topología usando el plugin seleccionado para estructurar los procesadores disponibles en una topología virtual. Podría ser uno de los plugins predefinidos por Hitmap o uno definido por el usuario.
Layout	`Layout(plugin, topology, shape)`	Un constructor de layouts determina la distribución de datos de un shape sobre una topología virtual. De la misma forma que con los objetos Topology, Hitmap ofrece muchos plugins predefinidos.
	`getShape([procId])`	Este método devuelve el shape local asignado a un procesador dado, o el shape del procesador actual.
Comm	`Comm(type, layout, TileIn, TileOut, [destination])`	Crea un nuevo objeto de comunicación que transmite datos desde los tiles usando los procesadores dentro del layout. El parámetro type define el modo de comunicación a realizar.
	`do()`	Realiza la comunicación encapsulada por el objeto Comm.
Pattern	`Pattern(type)`	Crea un nuevo patrón de comunicación, que puede ser ejecutado en orden o fuera de orden.
	`add(comm)`	Añade una nueva comunicación al patrón.
	`do()`	Realiza las comunicaciones del patrón.
Tile	`Tile<T>(shape)`	Constructor del Tile. Crea un objeto tile usando el dominio definido por un objeto shape. Es necesario indicar el tipo de datos a utilizar (tipo T genérico).
	`alloc()`	Reserva memoria para el tile
	`elemAt(x,y)`	Método para acceder al elemento en el tile.
	`get{Param}()`	Método para obtener el valor de un atributo privado. Param es el nombre del atributo.
	`select(tile, shape)`	Método para crear un subtile.
BTile	`BTile<T>(shape)`	Constructor del BTile. Crea un objeto btile usando el dominio definido por un objeto shape.
	`alloc()`	Reserva memoria para el tile
	`elemAt(x,y)`	Método para acceder al elemento en el tile.
	`select(tile, shape)`	Método para crear un subtile, análogo al de la clase Tile.
	`get{Param}()`	Método para obtener el valor de un atributo privado.

4.3.3. Métodos de acceso a elementos

Para acceder a los datos de un tile existen tres variantes que han sido explicadas previamente en el capítulo anterior.

4.4. Alternativas

Hemos probado diferentes opciones de implementación para construir los métodos de acceso a elementos en los Tiles. Las alternativas probadas son tres: macros, parámetros por defecto e "inlines". A continuación se describe cada una de estas alternativas.

4.4.1. Macros

Esta opción ha consistido en implementar macro-funciones de acceso a elementos en los Tiles. Estos macros se construyen de la misma forma que en C y son preprocesados por el compilador. [20] El uso de macros en C++ está desaconsejado aunque, como se ha comprobado en la experimentación, esta alternativa no se ve afectada por los problemas que afectan al inlining y a los parámetros opcionales. Las principales desventajas que conlleva el uso de macros son las siguientes:

1. El preprocesador reemplaza textualmente las llamadas a los macros por sus definiciones sin comprobar si es conveniente realizar la sustitución. Esto produce una carga adicional de responsabilidad sobre el programador.

2. No es posible definir punteros a macros de la misma forma que se definen punteros a funciones.

3. La evaluación de argumentos de macros con efectos laterales pueden causar comportamiento indeterminado porque cada argumento se evalúa tantas veces como se referencia en el cuerpo del macro.

4. Definir macro-funciones largas es antiestético. El cuerpo del macro debe caber en una línea lógica, por lo que cada línea física (excepto la última) se debe finalizar con una contrabarra para indicar la continuación de la línea lógica.

La ventaja principal del uso de macro-funciones es que su ejecución es más rápida, ya que los estados de las variables locales y el puntero de retorno no son guardados en la pila.

Un ejemplo de macro-función de acceso es el siguiente:

```
#define elemAt2(var, pos1, pos2)
    ((var).data[(pos1)*(var).qstride[0]*(var).origAcumSize[1]+
                (pos2)*(var).qstride[1]])
```

4.4.2. Parámetros opcionales

Los parámetros opcionales (parámetros por defecto) son útiles para reducir el tamaño del código, ya que permiten expresar con un solo método varias invocaciones al mismo con distinto número de parámetros. Sin embargo, se ven afectados por problemas de rendimiento que se explicarán con más detalle en el apartado de experimentación. Ejemplo de método de acceso con parámetros opcionales:

```
T & elemAt(int pos1, int pos2 = 0, int pos3 = 0, int pos4 = 0){
    return data[pos1*qstride[0]*origAcumSize[1]+
                pos2*qstride[1]*origAcumSize[2]+
                pos3*qstride[2]*origAcumSize[3]+
                pos4*qstride[3]];
}
```

4.4.3. Inlines

Los métodos inline fueron creados como alternativa a las macro-funciones y funcionan reemplazando cada llamada al método por el código del mismo. Debido a este funcionamiento, el código del método no debe ser muy extenso para evitar penalizaciones en el rendimiento. Para indicar que una función es inline, se utiliza la palabra clave "inline" al comienzo de su declaración como método.

La ventaja que proporciona el uso de métodos inline es un tiempo de ejecución menor que una llamada normal al método.

Las desventajas del uso de métodos inline son las siguientes:

1. El compilador generalmente decide qué funciones serán inline (es una recomendación al compilador).
   - No siempre resulta en una ganancia de velocidad.
   - La declaración como inline no siempre es necesaria
2. La ganancia en velocidad suele ser pequeña, por lo que a veces el inlining no es necesario (incluso puede ser contraproducente).
3. El inlining rompe el encapsulamiento porque el código se inserta en cada llamada al método. Además, el tamaño del fichero de cabecera es mayor.
4. Cuando se cambia un método inline, es necesario recompilar todo el código que usa ese método.

Ejemplo de método de acceso inline:

```
inline T & elemAt(int pos1, int pos2, int pos3, int pos4){
    return  data[pos1*qstride[0]*origAcumSize[1]+
                 pos2*qstride[1]*origAcumSize[2]+
                 pos3*qstride[2]*origAcumSize[3]+
                 pos4*qstride[3]];
}
```

En la versión actual de Hitmap, hemos optado por usar métodos inline para los accesos a tiles. Esta elección se debe a que son igual de eficientes que los macros, y más eficientes que los parámetros opcionales (ver el capítulo 5: Experimentación).

4.5. Tests realizados

Para realizar pruebas de funcionamiento no experimentales se han construido diferentes tests del apartado de "tiling". Estos tests pretenden comprobar que las clases construidas funcionan correctamente. Además de estos test, se han elaborado 2 más con propósito experimental. En total, se han construido 6 tests, que se pueden localizar en el directorio ***test/tiling/*** del soporte digital adjunto.

- **Test 1:** Con propósito experimental (se describe en el capítulo de "Experimentación" → Experimento 2).
- **Test 2:** Con propósito experimental (se describe en el capítulo de "Experimentación" → Experimento 2).
- **Test 3:** Prueba de funcionamiento de las subselecciones y actualizaciones hacia/desde el ancestro en el BTile. Se utilizan BShapes para definir el dominio de los BTiles.
- **Test 4:** Análogo al test 3 pero con Tiles y SigShapes en lugar de BTiles y BShapes.
- **Test 5:** Prueba de actualización desde/hacia el ancestro con BTiles, forzando un ancestro manualmente.
- **Test 6:** Segunda prueba de funcionamiento de susbselección y actualización de un BTile.

Parte IV

Estudio experimental

Capítulo 5

Experimentación

5.1. Experimento 1

5.1.1. Diseño

Para verificar experimentalmente la eficiencia del nuevo API en C++ se ha utilizado un test de multiplicación de matrices (Cannon [2]) implementado en C++, bajo diferentes entornos y usando 2 compiladores distintos: Intel C++ Compiler (ICC) [8] y GNU Compiler Collection (GCC) [21]. El test se ha ejecutado de manera secuencial y el producto de las matrices se realiza de dos formas, dependiendo del lenguaje de programación:

- En C++: utilizando la clase BTile para almacenar los datos (doubles).
- En C: utilizando Tiles normales con el mismo tipo de datos (doubles).

La finalidad de las pruebas es demostrar la perfecta integración del nuevo API de tratamiento de estructuras de datos en el contexto de programas paralelos implementados con Hitmap clásico, si las abstracciones propuestas introducen mayor overhead que la solución anterior en lenguaje C, y si existe una diferencia notable entre el uso de funciones inline y macros para algunas combinaciones de sistema operativo y compilador, lo que nos permitirá escoger la solución de diseño más apropiada. Por tanto, solo medimos el tiempo de las partes secuenciales del producto de matrices.

Descripción de las máquinas

Chimera: Posee un procesador Xeon E5-2620v2 con una frecuencia de reloj de 2.1GHz, 32 GB de memoria, 24 procesadores y sistema operativo CentOS. Como compilador utiliza gcc 4.8.3 y la versión 3.1.3 de mpich.

Ordenador portátil: Es un modelo ASUS K55VD con procesador i7-3610QM a 2.3GHz, 8 GB de memoria y sistema operativo Fedora 20. Posee el mismo compilador que Chimera, gcc 4.8.3 y la versión 3.0.4 de mpich.

Figura 5.1: Algoritmo de Cannon

```
1. Split all matrices in k*k pieces of (aprox.) the same size
   (For the simplified version, exactly the same size)

2.For s=0..k-1
    For i=0..k-1
      For j=0..k-1
         p = j-i-s
         q = i-j-s
         Update block: Ci,j = Ci,j + Ai,p * Bq,j
         (matrix multiplication of blocks)
      end-for
    end-for
  end-for

3. Return C
```

Configuración

Definimos dos configuraciones para describir las combinaciones de compilador y versión de mpich a probar en Chimera. Las configuraciones son las siguientes:

- Configuración 1 (conf1):
  - Biblioteca en C: gcc y mpich para gcc.
  - Biblioteca en C++: g++ y mpich para g++.
- Configuración 2 (conf2): Compilador de intel (icc) y mpich para icc para las dos bibliotecas.

En el ordenador portátil solo se ha probado la configuración 1 (gcc-g++).
Las optimizaciones aplicadas han sido las incluidas por el compilador por defecto con -O3.

Descripción del benchmark

El benchmark se basa en medir el tiempo de ejecución del producto entre dos matrices para almacenarlo en una tercera matriz, las tres con el mismo número de filas y columnas.

En la Fig. 5.2 se presenta una descripción de la estructura del programa Hitmap en pseudo-código.

El benchmark seleccionado (algoritmo de Cannon) se implementa en Hitmap con la siguiente estructura:

1. Construcción de BTiles
2. Inicialización de layouts y topologías (para comunicación entre procesos).
3. Selección de HitTiles y reserva de memoria.

4. Inicialización de las matrices
5. Comunicación entre procesos
6. Asignación de HitTile a BTile
7. Computación del producto de matrices
8. Presentación de los resultados
9. Liberación de la memoria asignada.

Este benchmark incluye las funcionalidades fundamentales implementadas en el nuevo API.

Estructura del benchmark

Las fases de construcción e inicialización son necesarias e imprescindibles en este benchmark por razones obvias. La fase de selección se realiza para reservar en la máquina local solo la parte de los tiles asignada por el proceso de partición y mapping en tiempo de ejecución. Es necesario asignar memoria a la parte local de los HitTiles debido a que se van a rellenar de datos. En la siguiente fase se rellenan los HitTiles de valores aleatorios. Posteriormente se realiza la conversión de la estructura de C al objeto de C++ (los campos son los mismos exceptuando la diferencia con los tipos genéricos, no disponibles en C).

Se realiza el producto de matrices midiendo los tiempos de ejecución, se presentan los resultados por pantalla y se libera la memoria reservada en el programa.

Selección de alternativas

Existen 3 alternativas para implementar los métodos de acceso a elementos del tile que ya se han comentado:

- **Inlines:** Útiles cuando el tamaño del código del método es reducido, como en este caso de estudio.
- **Parámetros por defecto:** En el experimento se determina si el funcionamiento es correcto con cada número de argumentos. Teóricamente, el compilador debe determinar el número de argumentos de cada invocación.
- **Macros:** La forma más clásica de elaborar un método en el que el tiempo de ejecución es importante. Esta altenativa es la que más se aleja de la orientación al objeto, y la que más se acerca al lenguaje C clásico.

5.1.2. Resultados de la experimentación

Eficiencia final

En los resultados de experimentación hemos comprobado que la eficiencia obtenida con la versión en C es mejor que la obtenida con la versión en C++. Sin embargo, las diferencias

Figura 5.2: Estructura del benchmark

```
1  function cannonsMM(int Afils, Acols, Bcols)
2      BTile tile1 <- new BTile
3      BTile tile2 <- new BTile
4      ...       ...      ...
5
6      constructor_layout()
7      constructor_topology()
8      ...       ...      ...
9
10     HitTile hitTile1 <- constructor_HitTile()
11     HitTile hitTile2 <- constructor_HitTile()
12     ...       ...      ...
13
14     select(hitTile1)
15     select(hitTile2)
16     alloc(hitTile1)
17     alloc(hitTile2)
18     ...       ...      ...
19
20     initMatrices(Acols, Bcols, hitTile1, hitTile2, hitTile3,
          layoutA, layoutB)
21
22     comm_processes()
23     ...       ...      ...
24
25     tile1 <- HitTile2BTile(hitTile1)
26     tile2 <- HitTile2BTile(hitTile2)
27     ...       ...      ...
28
29     measure_time()
30     matrixProduct(tile1, tile2, tile3)
31     measure_time()
32
33     print_elapsed_time()
34
35     free(hitTile1)
36     free(hitTile2)
37     free(layoutA)
38     ...     ...
39     free(topology)
40
41 end cannonsMM
```

son reducidas y la versión en C++ no tiene como objetivo prioritario mejorar el rendimiento de forma notable, sino que tratamos de obtener un rendimiento lo más parecido a C posible, pero mejorando la legibilidad y comprensibilidad del código al aplicar programación orientada al objeto.

Resultados y comentarios

En la Fig. 5.3 se muestra el comportamiento del test con cada configuración y tipo de prueba (código en C, funciones inline, macros y parámetros opcionales).

En el cuadro 5.1 se muestra la comparativa de tiempos entre la ejecución del test con la biblioteca de Hitmap en C y el test con la biblioteca de Hitmap en C++. Esta tabla contiene la misma información que la Fig. 5.3 en formato numérico.

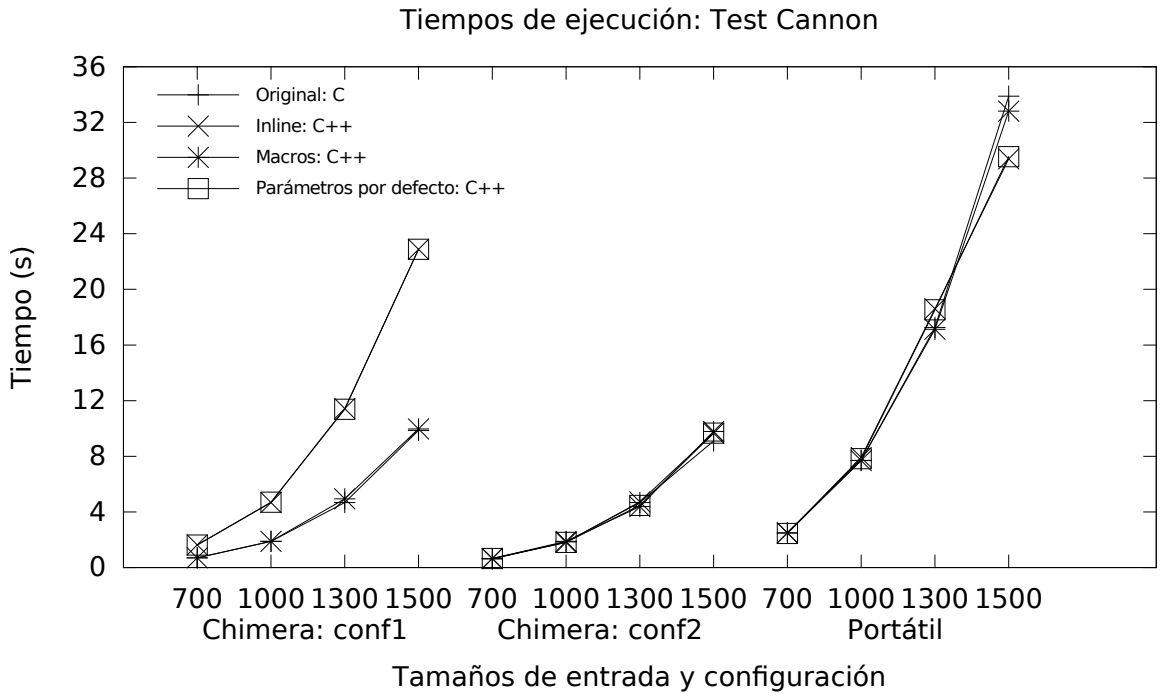
Los resultados de la experimentación indican que el compilador de Intel realiza correctamente el "inlining" de los métodos de acceso, mientras que el compilador de g++ en algunas distribuciones de Linux no realiza un inlining automático, lo que provoca una penalización de más de el doble de tiempo de ejecución. Este fallo del compilador es dependiente de las versiones del sistema operativo y del compilador.

Hemos comprobado que al usar macros en lugar de métodos, con el compilador de g++ se consigue alcanzar siempre un rendimiento similar al obtenido con el compilador de Intel (tanto con macros como con parámetros por defecto). Los resultados reflejan que solo los macros al estilo C garantizan la portabilidad del rendimiento en todas las combinaciones de compilador/sistema operativo. Los tiempos obtenidos al usar parámetros por defecto en los métodos de acceso a datos son equiparables a los tiempos de métodos "inline".

Cuadro 5.1: Tabla de resultados del experimento 1

Tipo de prueba	Tamaños	Chimera: conf1	Chimera: conf2	Portátil
Original: C	700 × 700	0.698 955	0.620 338	2.474 894
	1000 × 1000	1.884 447	1.846 236	7.705 859
	1300 × 1300	4.670 154	4.677 407	17.262 888
	1500 × 1500	9.859 584	9.093 761	33.889 508
Inline: C++	700 × 700	1.625 086	0.652 722	2.492 682
	1000 × 1000	4.676 312	1.812 905	7.909 313
	1300 × 1300	11.424 780	4.726 774	18.601 434
	1500 × 1500	22.874 937	9.668 605	29.376 619
Macros: C++	700 × 700	0.699 533	0.652 132	2.501 944
	1000 × 1000	1.885 313	1.879 940	7.699 871
	1300 × 1300	4.934 997	4.380 877	17.133 150
	1500 × 1500	9.965 866	9.787 394	32.813 568
Parámetros por defecto: C++	700 × 700	1.624 403	0.653 015	2.447 882
	1000 × 1000	4.700 732	1.811 135	7.823 098
	1300 × 1300	11.383 168	4.469 934	18.528 650
	1500 × 1500	22.886 274	9.669 877	29.533 245

Figura 5.3: Resultados del experimento 1



5.2. Experimento 2

5.2.1. Diseño

Hemos realizado un segundo experimento para comprobar la eficiencia de los accesos a un Tile con las 3 alternativas usando dos tests creados por el estudiante. Cada test comprueba un tipo de acceso:

- **Test 1:** Realiza los accesos al campo de datos del tile mediante el método "rootCoords" o el macro homónimo.
- **Test 2:** Realiza los accesos al campo de datos del tile usando el método "elemAt" o el macro homónimo.

La finalidad de las pruebas es aclarar qué alternativa es la más apropiada para implementar los accesos al tile (inlining, macros o parámetros por defecto). En los tests, se mide el tiempo transcurrido al sumar los campos de datos de dos tiles en varias iteraciones. Por tanto, solo medimos el tiempo de las partes secuenciales del producto de matrices.

Descripción de las máquinas

La máquina utilizada es Chimera, la misma que en el experimento 1.

Configuración

Se han utilizado dos configuraciones distintas para realizar los tests:

- Usando el compilador g++
- Usando el compilador icc

En ambos casos, las optimizaciones aplicadas son las usadas por el compilador por defecto con -O3.

Se han probado 10 combinaciones diferentes para implementar dos tests de acceso a elementos de la clase Tile, ejecutados 7 veces consecutivas para cada combinación. Los dos tests utilizados están implementados de la siguiente forma: Se reserva espacio para dos matrices de 16 filas con 16 MB de elementos en cada fila. Se han utilizado matrices de este tamaño de fila para adaptarlo al tamaño de caché. Se utilizan dos objetos de tipo Tile, cuyo campo "data" se inicializa con el número de elementos de una fila de su matriz asignada, a través del constructor de la clase. Se mide el tiempo de ejecución de cada iteración para una misma combinación usando una clase prediseñada por el co-tutor de este proyecto, Javier Fresno Bausela (Seq_Clock). La función principal de cada test consiste en recorrer los elementos del campo de datos de los Tiles y efectuar una suma de cada par de elementos en la misma posición.

Los **tipos a probar** con cada número de dimensiones son los siguientes:

1. **Macros expandidos:** Se utilizan macros como en el lenguaje C, expandiendo el método "array2Tile()". Esto es, con el contenido del método en el macro en lugar de su invocación.
2. **Macros sin expandir:** Análogo al tipo anterior pero sin expandir las llamadas a la función "array2Tile()".
3. **No inline métodos sin expandir:** Se usa un método distinto para cada número de dimensiones, y sin expandir las llamadas al método "array2Tile()".
4. **No inline métodos expandidos:** Análogo al anterior pero con las expansiones de las invocaciones a la función "array2Tile()".
5. **No inline parámetros por defecto sin expandir:** Utilización de parámetros por defecto, con los métodos "array2Tile()" sin expandir.
6. **No inline parámetros por defecto expandidos:** Análogo al anterior pero con las llamadas a "array2Tile()" expandidas.
7. **Inline métodos sin expandir:** Utilización de las funciones "inline" propias de C++, un método para cada número de dimensiones y sin las llamadas a "array2Tile()" expandidas.
8. **Inline métodos expandidos:** Análogo al anterior, incluyendo expansiones de código del método "array2Tile()".
9. **Inline parámetros por defecto sin expandir:** Utilización de funciones "inline", así como parámetros por defecto. Sin expansiones de código.

10. **Inline parámetros por defecto expandidos:** Análogo al anterior tipo, pero con expansiones del código de "array2Tile()".

La notación "expandidos" y "sin expandir" se refiere a la realización o no del despliegue del código del método "array2Tile" en el caso del acceso mediante rootCoords.

Descripción del benchmark

Cada test consiste en un "switch" con 4 etiquetas "case", una para cada número de dimensiones. Durante la ejecución de los tests se realiza la suma de cada par de elementos del campo de datos en la misma posición de cada Tile. La suma siempre será 0, dado que las matrices se inicializan con todos sus elementos a 0.

La estructura del benchmark es la siguiente:

1. Definición de los macros con el número de filas (16) y número de elementos por fila (16777216)
2. Declaración de matrices y tiles fuera del main
3. Comprobación del argumento introducido (número de dimensiones)
4. Asignación de memoria e inicialización de las matrices a 0.
5. Construcción y asignación de memoria de los tiles.
6. Inicio de la medición de tiempos
7. Realización de las 16 iteraciones en las que se suman los elementos de los dos tiles *(sum += tile1.data[i] + tile2.data[i])*.
8. Detención de la medición de tiempos
9. Impresión por pantalla del tiempo transcurrido
10. Liberación de la memoria asignada a los tiles, del objeto de la clase Seq_Clock y de las matrices.

Estructura del benchmark

En la Fig. 5.4 se puede ver el contenido del método "main" del tests para el método de acceso "elemAt". En la Fig. 5.5 se muestra el método de suma de los campos de datos del tile para el método de acceso "elemAt".

Selección de alternativas

Las 3 alternativas existentes son las mismas que en el experimento 1 (inlines, macros y parámetros opcionales) con la decisión adicional del compilador (g++ o icc).

Figura 5.4: Estructura del método “main” del benchmark del experimento 2

```
int main(int argc, char *argv[]){

  if(argc != 2){
    cout << "El numero de argumentos introducido es incorrecto" << endl;
    return(-1);
  }

  int i = 0;
  int j = 0;
  int dims = atoi(argv[1]);
  int size = 0;
  switch(dims){
    case 1: size = TAM;
      break;
    case 2: size = (int)sqrt(TAM);
      break;
    case 3: size = (int)pow(TAM, 1.0/3.0);
      break;
    case 4: size = (int)pow(TAM, 1.0/4.0);
      break;
  }

  SeqClock * clock = new SeqClock();

  if(dims < 1 || dims > 4){
    cout << "El numero de dimensiones introducido es incorrecto" << endl;
    return(-1);
  }

  mA = new int *[ITER];
  mB = new int *[ITER];

  for(i = 0; i < ITER; i++){
    mA[i] = new int[TAM];
    mB[i] = new int[TAM];

    for(j=0; j < TAM; j++){
      mA[i][j] = 0;
      mB[i][j] = 0;
    }
  }

  tile1 = Tile<int>(0, dims, size, size, size, size);
  tile2 = Tile<int>(0, dims, size, size, size, size);
  tile1.alloc();
  tile2.alloc();
  clock->start();
  for(i=0;i<ITER;i++){
    tile1.setData(mA[i]);
    tile2.setData(mB[i]);
    sumaMatricesElemAt(dims, size);
  }
  clock->stop();
  cout << "Tiempo: " << clock->seconds() << " segundos" << endl;

  delete clock;
  for (i = 0; i < ITER; i++){
    delete [] mA[i];
    delete [] mB[i];
    }
  delete [] mA;
  delete [] mB;
```

Figura 5.5: Estructura del método de suma del benchmark del experimento 2

```
1
2  void sumaMatricesElemAt(int numPos, int size){
3    int sum = 0;
4
5    switch(numPos){
6      case 1:
7        for(int i = 0;i < size; i++){
8    sum += tile1.elemAt(i) + tile2.elemAt(i);
9        }
10       cout << "La suma total es: " << sum << endl;
11       break;
12     case 2:
13       for(int i = 0;i < size; i++){
14   for(int j = 0;j < size;j++){
15     sum += tile1.elemAt(i,j) + tile2.elemAt(i,j);
16   }
17       }
18       cout << "La suma total es: " << sum << endl;
19       break;
20     case 3:
21       for(int i = 0;i < size; i++){
22   for(int j = 0;j < size;j++){
23     for(int k = 0;k < size; k++){
24       sum += tile1.elemAt(i,j,k) + tile2.elemAt(i,j,k);
25     }
26   }
27       }
28       cout << "La suma total es: " << sum << endl;
29       break;
30     default:
31       for(int i = 0;i < size; i++){
32   for(int j = 0;j < size;j++){
33     for(int k = 0;k < size; k++){
34       for(int l = 0;l < size; l++){
35         sum += tile1.elemAt(i,j,k,l) + tile2.elemAt(i,j,k,l);
36       }
37     }
38   }
39       }
40       cout << "La suma total es: " << sum << endl;
41       break;
42    }
43 }
```

5.2.2. Resultados de la experimentación

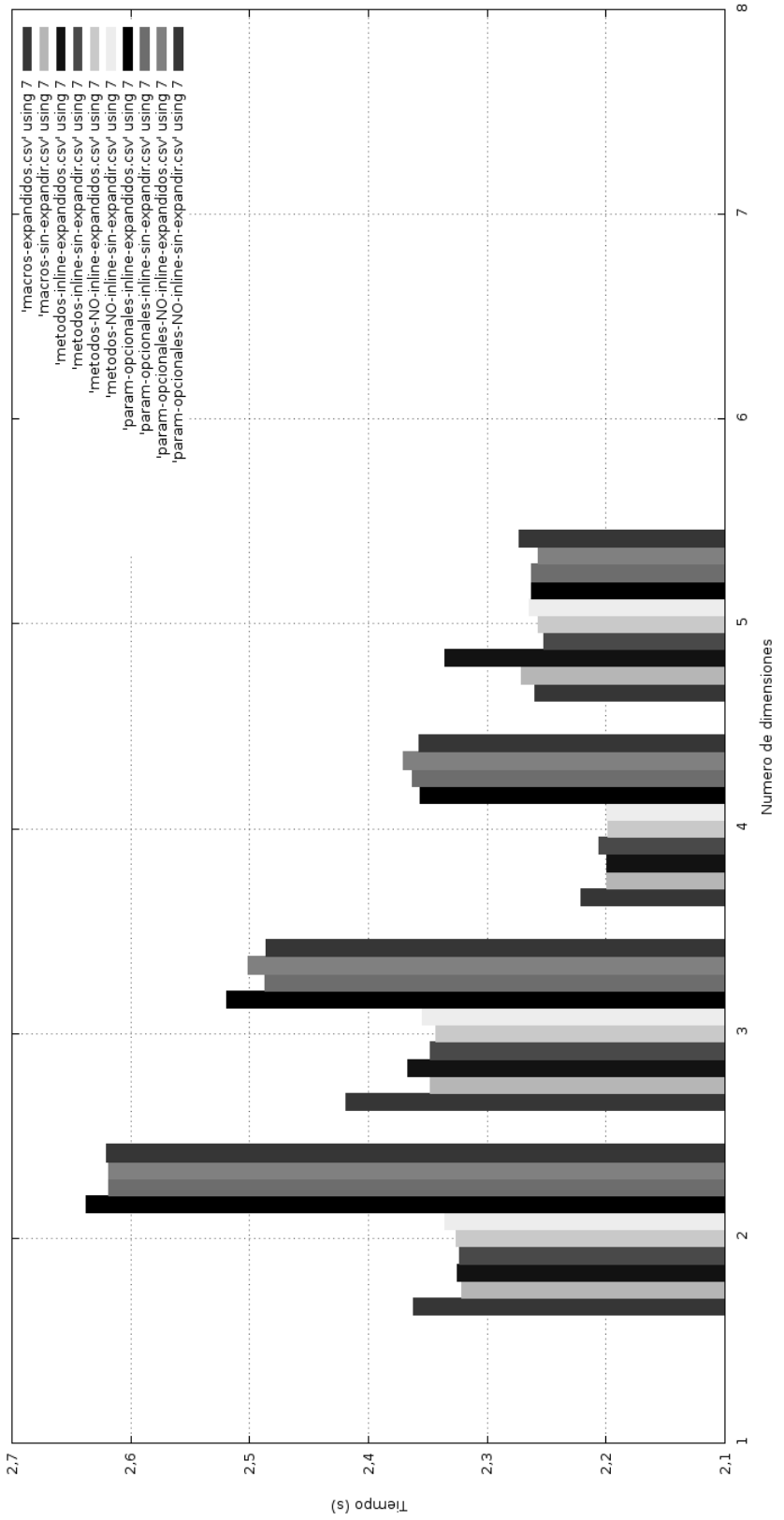
Eficiencia final

Se observa en general un mejor rendimiento con el compilador de Intel respecto al compilador de GNU.

Resultados y comentarios

Los resultados obtenidos muestran que el uso de funciones con parámetros opcionales es mucho más ineficiente que el uso de métodos. Las diferencias entre macros, funciones inline y no inline, expandidas y sin expandir son demasiado pequeñas como para demostrar una mejora significativa (la desviación típica de las muestras puede enmascarar las interpretaciones, convirtiéndolas en erróneas). En la Fig. 5.6 se puede ver el comportamiento del benchmark en el test1 (rootCoords) con el compilador de g++. A pesar de que el desarrollo de tiempos con cada número de dimensiones (1-4) puede parecer caótico, se observa que con parámetros opcionales se obtiene peor rendimiento que con el resto de alternativas, excepto cuando se realiza el test con 4 dimensiones. En este caso los tiempos están relativamente equilibrados en todas las alternativas.

Figura 5.6: Gráfica con resultados del experimento 2



Parte V

Conclusiones

Capítulo 6

Valoración del trabajo realizado

6.1. Objetivos cumplidos

Este trabajo ha consistido en implementar la parte de la interfaz de biblioteca de Hitmap relativa al manejo de estructuras de datos en C++. Como aspectos más importantes que se debían alcanzar, destacamos el rendimiento alcanzado y la adaptación a la Programación Orientada al Objeto, siendo este último aspecto consecuencia directa del lenguaje de programación escogido (C++) y el que se ha conseguido aplicar en todas las clases construidas.

Podemos decir que hemos cumplido los objetivos marcados para este trabajo, condensados en pocas palabras en elaborar, testar y medir los cambios realizados en la biblioteca de Hitmap para C++. Los resultados obtenidos después de la elaboración del conjunto de clases para Hitmap se recogen resumidos en los siguientes puntos:

- Obtención de una parte funcional de la biblioteca de Hitmap para C++, para el manejo de estructuras de datos.
- Habilitación de diferentes opciones de mejora del rendimiento de cara al futuro.
- Presentación de resultados para un caso de estudio en el que se muestra el rendimiento, validez y adecuación de las funciones de Hitmap en C++.

6.2. Competencias adquiridas

Durante la realización del proyecto he adquirido conocimientos acerca del manejo del lenguaje de programación C++, he publicado un artículo científico para un congreso a nivel nacional y he trabajado en el contexto de un proyecto de metodología ágil, obteniendo una realimentación de los contenidos cursados previamente en la Universidad. Así mismo, he concretado una tarea a largo plazo, valorando el aprendizaje alcanzado en un ámbito previo al entorno laboral o al área de la investigación. Estas competencias, complementando el recorrido conceptual desarrollado en la carrera, me han proporcionado la oportunidad de encarar el futuro como un Ingeniero Informático que conoce los aspectos más importantes de su profesión, así como las diferentes

opciones laborales y que tiene la seguridad de que durante un plazo medio-largo esta profesión tendrá un índice de ocupación elevado.

La asignatura de Programación impartida en el Grado ha sido el pilar básico sobre el que se asienta el trabajo realizado en este proyecto. La influencia de la asignatura Estructuras de Datos y Algoritmos ha sido importante de cara a la implementación de las clases de la biblioteca. La planificación y gestión del proyecto se basa en la asignatura homónima de la carrera.

6.3. Conclusiones

Como conclusiones de este proyecto, se puede extraer que:

1. La nueva biblioteca implementada en C++ se ha construido aplicando POO, y esto implica una mejora visible en cuanto a la legibilidad, estructuración y mantenibilidad del código.

2. Los riesgos en un proyecto a largo plazo pueden tener un impacto significativo, suponiendo o bien un sobrecoste importante o un retraso en la fecha de finalización del mismo. El impacto de un riesgo será mayor cuanto antes se haga efectivo.

3. El peso de la fase de implementación supera en importancia a cualquier otra fase de un proyecto de estas características. En este caso, solo ha trabajado en la implementación una persona, lo que ha provocado que al ocurrir un riesgo se ha prolongado irremisiblemente el proyecto.

4. Las tareas que se llevan a cabo en el contexto de un grupo de investigación supeditan la obtención de resultados válidos en la experimentación al trabajo constante, preciso y coordinado de todos los miembros del equipo.

5. La elaboración de un documento académico es útil para aprender a describir un proceso de investigación, sirviendo también de base para la realización posterior de este proyecto.

6. La importancia de haber participado en un proyecto a largo plazo es notoria de cara a la posible participación del estudiante en contextos similares una vez comenzada su etapa laboral. Además, las dificultades que se van superando a lo largo de la carrera inicialmente, y en la elaboración del proyecto finalmente otorgan al estudiante una valiosa experiencia académica y, ulteriormente, una experiencia vital.

6.4. Trabajo futuro

Es posible abordar la revisión de cambios futuros teniendo en cuenta diferentes aspectos, tales como el rendimiento, la liberación en la toma de decisiones al programador, la claridad y legibilidad del código, POO, etc. En el caso del rendimiento, existe un problema detectado y asumido con el vector<bool>. Tenemos conocimiento de que no es la implementación más eficiente en cuanto a tiempo de ejecución, pero su elección atendió a otra razón considerada de mayor trascendencia: la reducción del espacio ocupado por el BTile en memoria. Para las

siguientes revisiones a llevar a cabo sobre Hitmap++ se ha de tener en cuenta esta situación. En cuanto a las clases, se propone una segunda fase para adaptar el API de las funciones de mapping y comunicaciones que deberá ser concretada para obtener una biblioteca completamente funcional.

Parte VI

Apéndices y Bibliografía

Apéndice A

Contenido del CD-ROM y manuales

A.1. Contenido del CD-ROM

En el CD-ROM adjunto está incluida la última versión de Hitmap, preparada para ser compilada, además de la memoria en formato PDF y la versión antigua de Hitmap (necesaria para compilar el test Cannon).

A.2. Manual de usuario

A.2.1. Compilación

Para generar los ficheros Makefile, es necesario ejecutar el script "./autotools-all.sh" con el parámetro RELEASE. En el directorio raíz de Hitmap++, es necesario ejecutar "make" para compilar la biblioteca. En el caso de no añadir RELEASE al script generador de ficheros Makefile se compilará sin optimizaciones, en modo depuración. La versión antigua de la biblioteca ya se encuentra compilada. En el caso de que se necesite volver a compilarla, debe ejecutarse "make" en el directorio raíz de la versión antigua de la biblioteca.

A.2.2. Funciones más importantes

- Métodos de acceso a elementos del tile: *elemAt* y *rootCoords*. Ejemplo de uso:

```
T & elemAt(int pos1, int pos2, int pos3, int pos4){
    return  data[pos1*qstride[0]*origAcumSize[1]+
                 pos2*qstride[1]*origAcumSize[2]+
                 pos3*qstride[2]*origAcumSize[3]+
                 pos4*qstride[3]];
}
```

- Constructores del tile. Ejemplo de uso:

```
SigShape shape = SigShape(2,4,6,8);
Tile<int> * tile = new Tile<int>(0, shape);
```

- Métodos de actualización de tiles en la jerarquía: *updateToAncestor* y *updateFromAncestor*. Estos métodos se utilizan principalmente cuando se ha realizado una subselección de un tile, con el objetivo de actualizar los elementos de datos del tile ancestro o descendiente. Ejemplo completo de uso:

```
1  BShape shape1 = BShape(16,16);
2  BShape shape2 = BShape(8,8);
3
4  // Inicializacion de los BShapes
5  shape1.setBegin(0, 2);  // La dimensión 0 comienza en 2
6  shape1.setEnd(0, 17);   // La dimensión 0 acaba en 17
7  shape1.setBegin(1, 5);  // La dimensión 1 comienza en 5
8  shape1.setEnd(1, 20);   // La dimensión 1 acaba en 20
9  shape1.setStride(0, 1); // El stride de la dim. 0 vale 1
10 shape1.setStride(1, 1); // El stride de la dim. 1 vale 1
11
12 shape2.setBegin(0, 1);
13 shape2.setEnd(0, 8);
14 shape2.setBegin(1, 3);
15 shape2.setEnd(1, 10);
16 shape2.setStride(0, 1);
17 shape2.setStride(1, 1);
18
19 // Todos los elementos del BShape a 1
20 for(int i = 0; i < shape1.getSize(); i++){
21   shape1.setDataElem(i, 1);
22
23 }
24
25 // Todos los elementos del BShape a 1
26 for(int i = 0; i < shape2.getSize(); i++){
27   shape2.setDataElem(i, 1);
28
29 }
30
31 // Fin de inicialización
32
33 BTile<int> * tile1 = new BTile<int>(0, shape1);
34 BTile<int> * tile2 = new BTile<int>();
35
36 tile1->alloc();
37 tile2->alloc();
38
39 for(int i = 0; i < tile1->getCards(0); i++)
40  for(int j = 0; i < tile1->getCards(1); j++)
41   tile1->elemAt(i, j) = i * tile1->getCards(1) + j;
42 // Elemento 0 vale 0, elemento 1 vale 1, etc.
43
44 tile1->select(tile2, shape2);
45 tile2->alloc();
46
47 for(int i = 0; i < tile2->getCards(0); i++)
48  for(int j = 0; i < tile2->getCards(1); j++)
49   tile2->elemAt(i, j) = (i * tile2->getCards(1) + j) * 10;
50 // Elemento 0 vale 0, elemento 1 vale 10, etc.
51
52
```

```
53 tile2->updateFromAncestor();
54 /* Al ejecutar este método, el tile2 se actualizará
55 con los elementos del campo de datos del tile1, pero
56 únicamente aquellos con el valor del BShape a 1 */
57
58 delete tile1;
59 delete tile2;
```

A.3. Manual de desarrollador

A.3.1. Estructura jerárquica

En la carpeta "src" se encuentran los directorios con el código fuente, mientras que en el directorio "test" se encuentran los ficheros de test con los que se ha comprobado el funcionamiento de los componentes de "tiling" de la biblioteca.

Los ficheros "src/tiling/tile.h" y "src/tiling/btile.h" contienen los atributos y métodos de las clases de tiling principales. Los ficheros sig.h y sig.cpp contienen las funcionalidades de las Signaturas. En los ficheros shape.h, sigshape.h, sigshape.cpp, bshape.h y bshape.cpp se encuentra el código correspondiente a los dominios de Hitmap: Shapes de signaturas (SigShape) y Shapes de mapa de bits (BShape). Los ficheros tile_contiguous.h y btile_contiguous.h contienen las clases que heredan de Tile y BTile, conteniendo los métodos de acceso para stride 1 (elementos contiguos).

A.3.2. Árbol de directorios

- Hitmap1.1 (versión en C de Hitmap, necesaria para compilar el test Cannon)
- Hitmap++
  - docs. En este directorio se encuentra la documentación de la biblioteca.
  - examples. En este directorio se localizan los programas de ejemplo para la experimentación
  - external. En esta carpeta se encuentran recursos externos a la biblioteca.
  - m4. Contiene scripts para detectar la versión de MPI instalada.
  - other. Otros ficheros.
  - src. Ficheros fuente de la biblioteca.
    - comm. Clases de comunicaciones. No modificadas en este proyecto.
    - dataflow. No modificado en este proyecto.
    - mapping. Clases de mapeo. No modificado en este proyecto.
    - parutils. No modificado en este proyecto.

  - utils. Clases de complemento y herramientas.
  - tiling. Clases de particionado de tiles. Directorio modificado en este proyecto.
    - bshape.cpp y bshape.h
    - btile.h y btile_contiguous.h
    - tile.h y tile_contiguous.h
    - hit_namelist.cpp y hit_namelist.h
    - iterator.h
    - shape.h
    - sig.cpp y sig.h
    - sigshape.cpp y sigshape.h

- test. Programas de test para las pruebas y depuración de errores.

Bibliografía

[1] James Cadle and Donald Yeates. *Project Management for Information Systems.* Prentice Hall Press, Upper Saddle River, NJ, USA, 5th edition, 2007.

[2] Lynn Elliot Cannon. *A Cellular Computer to Implement the Kalman Filter Algorithm.* PhD thesis, Bozeman, MT, USA, 1969. AAI7010025.

[3] B.L. Chamberlain, D. Callahan, and H.P. Zima. Parallel programmability and the chapel language. *Int. J. High Perform. Comput. Appl.*, 21(3):291–312, aug 2007.

[4] B. Chapman, G. Jost, and R. van der Pas. *Glossary*, pages 321–330. MIT Press, 2007.

[5] Yifeng Chen, Xiang Cui, and Hong Mei. Parray: A unifying array representation for heterogeneous parallelism. *SIGPLAN Not.*, 47(8):171–180, feb 2012.

[6] Message Passing Interface Forum. Mpi: A message-passing interface standard version 3.0. Technical report, 2012.

[7] Basilio B. Fraguela, Ganesh Bikshandi, Jia Guo, MaríA J. GarzaráN, David Padua, and Christoph Von Praun. Optimization techniques for efficient hta programs. *Parallel Comput.*, 38(9):465–484, sep 2012.

[8] Christopher W. Fraser and David R. Hanson. *A Retargetable C Compiler: Design and Implementation.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[9] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[10] Benedict Gaster, Lee Howes, David R. Kaeli, Perhaad Mistry, and Dana Schaa. *Heterogeneous Computing with OpenCL: Revised OpenCL 1.2 Edition.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2 edition, 2013.

[11] Arturo Gonzalez-Escribano, Yuri Torres, Javier Fresno, and Diego R. Llanos. An Extensible System for Multilevel Automatic Data Partition and Mapping. *IEEE Transactions on Parallel and Distributed Systems*, 25(5):1145–1154, may 2014.

[12] Ananth Grama, Anshul Gupta, George Karypis, and Vipin Kumar. *Introduction to Parallel Computing.* Addison-Wesley Pearson Education Co., Inc., Harlow, Essex, England, 2003.

[13] Brian W. Kernighan. *The C Programming Language*. Prentice Hall Professional Technical Reference, 2 edition, 1988.

[14] D.B. Loveman. High performance fortran. *Parallel & Distributed Technology: Systems & Applications*, 1(1):25–42, Feb 1993.

[15] D. A. Mallón, A. Gómez, J. C. Mouriño, G. L. Taboada, C. Teijeiro, J. Touriño, B. B. Fraguela, R. Doallo, and B. Wibecan. Upc performance evaluation on a multicore system. In *Proceedings of the Third Conference on Partitioned Global Address Space Programing Models*, PGAS '09, pages 9:1–9:7, New York, NY, USA, 2009. ACM.

[16] Bradford Nichols, Dick Buttlar, and Jacqueline Proulx Farrell. *Pthreads Programming*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1996.

[17] Vreda Pieterse, Derrick G. Kourie, Loek Cleophas, and Bruce W. Watson. Performance of c++ bit-vector implementations. *Proceeding SAICSIT '10 Proceedings of the 2010 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists*, pages 242–250, 2010.

[18] Jason Sanders and Edward Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley Professional, 1st edition, 2010.

[19] Axel Schreiner. *Object Orientated Programming in ANSI-C*. Hanser publications, 1994.

[20] M. T. Skinner. *The Advanced C++ Book*. Silicon Press, Summit, NJ, USA, 1991.

[21] Richard M. Stallman and GCC DeveloperCommunity. *Using The Gnu Compiler Collection: A Gnu Manual For Gcc Version 4.3.3*. CreateSpace, Paramount, CA, 2009.

[22] Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley Professional, 4 edition, 2013.