Interfaz orientado al objeto para una biblioteca de programación paralela

Oscar Gonzalez-Ossorio, Javier Fresno, Arturo Gonzalez-Escribano [1]

Resumen—**La distribución de datos entre diferentes procesos es una de las tareas fundamentales a 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++ para esta biblioteca. Nuestro objetivo es eliminar las limitaciones del interfaz actual aplicando soluciones de los lenguajes orientados a objeto. Esta primera fase se centra en la descripción de los problemas encontrados y las soluciones propuestas para implementar el módulo de manejo de datos de Hitmap.**

Palabras clave—**Programación paralela, Distribución de datos, Orientación al objeto**

I. Introducción

LAS herramientas clásicas para programar en entornos de memoria distribuida, por ejemplo las bibliotecas de paso de mensajes como MPI [1], no ofrecen más que un mecanismo para transmitir los datos. 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 [2] es una de las propuestas que pretende simplificar la tarea de la programación paralela. Es una biblioteca con funcionalidades de *tiling* jerárquico. 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 completas de datos.

Hitmap está implementada en el lenguaje de programación C [3]. 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 [4], el API no es suficientemente intuitivo para el programador final. Nuestro objetivo es desarrollar una interfaz en el lenguaje C++ [5] para poder aprovechar todas las ventajas de las programación orientada al objeto [6]. 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.

El resto del artículo está organizado de la siguiente forma. En la sección II se habla de im-

[1]Dpto. de Informática, Univ. de Valladolid, e-mail: oscar.gonzalez@alumnos.uva.es, jfresno@infor.uva.es, arturo@infor.uva.es.

plementaciones con propósitos similares a Hitmap. La sección III describe el funcionamiento y la estructura de la biblioteca Hitmap implementada en C. En el apartado IV se detalla la propuesta de Hitmap C++. Las secciones V y VI contienen respectivamente los datos de la experimentación y los resultados obtenidos. Las dos últimas secciones corresponden a las conclusiones de este artículo y al trabajo futuro.

II. Trabajo previo

Existen muchas propuestas de lenguajes de programación que facilitan a tarea del programador. 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 [2] 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 potenciar el paralelismo de procesos jerárquicos en entornos híbridos. Hitmap alcanza una eficiencia equiparable a UPC [7] reduciendo la complejidad de programación. Chapel [8] 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 [9].

Parray [10] 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.

III. Hitmap C: Estructura y características

Hitmap es una biblioteca de funciones diseñada para el reparto y mapping jerárquico de estructuras de datos densas [2]. También ha sido extendida para soportar estructuras de datos dispersas como matrices dispersas o grafos, usando la misma metodología e interfaz [2]. Hitmap está basado en el modelo de programación distribuido SPMD (Single Program Multiple Data). Presenta abstracciones para la gestión de estructuras de datos, el particionado y la comunicación en diferentes niveles jerárquicos forma automatizada, manteniendo un buen rendimiento.

Hitmap define varios conceptos para escribir programas paralelos. Un *shape* representa el dominio de datos usado en el programa mientras que un *tile* es la entidad que posee los datos. Los tiles se crean usando shapes. En Hitmap, una *topología* describe la estructura de los procesos disponibles. Un *layout* es la entidad que distribuye los shapes en una topología, dividiendo el dominio para cada elemento de la topología. El concepto de *comunicaciones* representa la transmisión de datos entre dos o más procesos. Finalmente, un *patrón* agrupa varias comunicaciones.

A. Arquitectura de Hitmap

En la fig. 1 se muestra la estructura de la biblioteca Hitmap. La biblioteca se puede dividir en 3 secciones:

- **Métodos de tiling:** Definición y manipulación de arrays y tiles, con una metodología tile a tile. Estos métodos se pueden usar de manera independiente a otros, para mejorar la localidad en el código secuencial, así como para generar distribuciones de datos manualmente para la ejecución en paralelo.
- **Métodos de mapping:** Distribución de datos y métodos del layout para particionar dominios automáticamente, dependiendo de la topología virtual seleccionada. Estos métodos están orientados a la

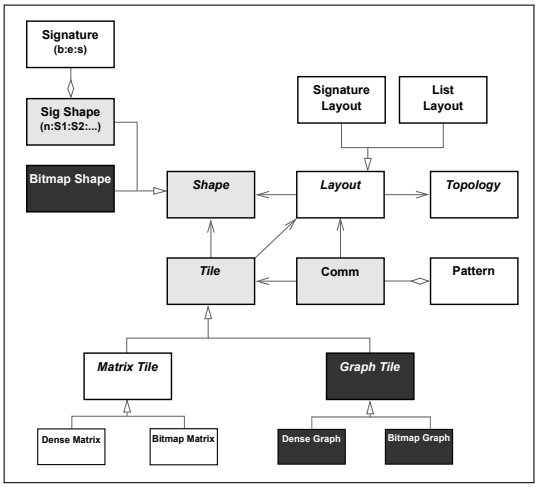

Fig. 1. Estructura de Hitmap

distribución de datos y tareas en entornos paralelos. Estas funciones devuelven (1) los rangos de los tiles que necesitan ser creados, (2) el mapping entre tiles y procesadores virtuales y (3) la información de los vecinos, encapsulada en una única estructura.

- **Métodos 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 mapping para crear patrones de comunicación dependientes del mapping.

En las tablas II y III se muestran, respectivamente, la API antigua y las nuevas modificaciones de Hitmap.

B. Descripción de las clases

En este apartado se describen los elementos comunes entre las dos implementaciones existentes de Hitmap: Dominios y Tiles.

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:

B.1 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 las signaturas:

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

B.2 Shapes

Los Shapes son los elementos de dominio que sirven de base para los Tiles. A continuación se describen los dos tipos de Shapes en Hitmap:

B.2.a 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 multidimensional. Los Shapes tienen cardinalidad, entendida como el número de combinaciones de los índices en el dominio.

B.2.b Shapes de mapa de bits. 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 usa internamente una matriz de booleanos (Bitmap) como estructura auxiliar.

B.3 Clases de Tiles

Existen dos tipos de estructuras para mapping, Tiles y BTiles. Un Tile es un array de n dimensiones (de 1 a 4) cuyo dominio se define por un SigShape. Un BTile es una matriz con un dominio acotado por un BShape. Ambos tipos de Tiles están formados por variables de tipo genérico.

Los Tiles y BTiles mapean 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 o BTile original. La razón de la inclusión de una jerarquía de Tiles es la consecución del reparto, localización y acceso eficiente a los datos entre diferentes procesadores.

Además, se gestiona el estado de la memoria asignada, para gestionar 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.

IV. Descripción de la propuesta: Hitmap C++

En este caso de estudio hemos construido un conjunto de clases en C++ a partir de la interfaz de biblioteca en C ya existente. Esta biblioteca permite tratar y particionar los datos usando contenedores (Tiles) con dominios configurables (Shapes). La parte construida en C++ tiene como objetivos 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 del compilador de C++ en diferentes entornos y se mostrará un estudio sobre la eficiencia y el rendimiento de las soluciones propuestas.

A. 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 el begin, end y stride de la signatura, mientras que en las clases BShape y SigShape se crean tantas signaturas como número de dimensiones tendrá el dominio.

B. 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. Los métodos de tipo "getters" y "setters" permiten declarar los campos como privados.

B.1 Métodos de creación

Hemos implementado constructores 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 15 de la fig. 3:

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

B.2 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. Los tiles están organizados en una estructura jerárquica, con una posición más alta o más baja dependiendo del grado de partición y selección interna realizado sobre los tiles ancestros.

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:

```
tile1.alloc()
```

Para destruir un elemento en la jerarquía se utiliza un destructor recursivo. 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: Base con coordenadas de array (absoluta) y base con coordenadas de Tile (relativa al subdominio del Tile padre).

El término "base absoluta" implica que las posiciones indicadas en el dominio se corresponden directamente con cada posición del array de datos usando el stride entre cada elemento. La base relativa toma como referencia las posiciones del Tile, en orden relativo respecto al array.

B.3 Métodos de acceso a elementos

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

- Acceso sin tener en cuenta el 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:

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. Hemos comprobado que existe un problema al usar parámetros por defecto que provoca una pérdida de rendimiento cuando se realiza la llamada al método con un número de argumentos menor que el máximo (ver sección de experimentación).
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.
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.

C. 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. [11] 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).

C.1 Métodos de creación

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

C.2 Métodos de manejo de memoria

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

C.3 Métodos de acceso a elementos

Además de los métodos usados en los Tiles, también se incluyen otros de acceso a elementos en grafos. Para acceder a la posición del dato, es necesario realizar el cálculo correspondiente al producto del número de fila por el número de columnas más el número de columna, ya que los elementos se almacenan en un array contiguo usado como mapa de bits.

V. Experimentación

Para verificar experimentalmente la eficiencia del nuevo API en C++ se ha utilizado un test de multiplicación de matrices (Cannon) implementado en C++, bajo diferentes entornos y usando 2 compiladores distintos (gcc e icc). 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 que existe una diferencia notable entre el uso de

funciones inline y macros para algunas combinaciones de sistema operativo y compilador. La versión en C++ tiene un menor rendimiento debido al sobrecoste de operaciones de asignación al transformar las estructuras de C a sus análogas en C++. Aunque esto suponga un tiempo de ejecución mayor que en la versión en C, no afecta directamente ya que solo se mide el tiempo del producto de matrices.

A. Problema con parámetros por defecto

Existe un problema de rendimiento al utilizar parámetros por defecto en los métodos de acceso. Este hecho implica que solo funcionan bien cuando se invoca al método con el número máximo de argumentos.

Las diferencias en el tiempo de ejecución al usar parámetros por defecto son grandes. Al utilizar el compilador de icc para 1, 2 y 3 dimensiones, el tiempo de ejecución con parámetros opcionales es equiparable, mientras que para 4 dimensiones disminuye en gran medida. Las diferencias de tiempo entre el uso de funciones inline y parámetros por defecto con el compilador gcc son:

- 1 dimensión: Un 13 % más de tiempo en parámetros por defecto respecto al inlining.
- 2 dimensiones: Un 8 % más de tiempo en parámetros por defecto.
- 3 dimensiones: Un 7 % más de tiempo en parámetros por defecto.
- 4 dimensiones: Los resultados para 4 dimensiones son aproxidamente idénticos.

B. 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.

B.1 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.

Las optimizaciones aplicadas han sido las incluidas por el compilador por defecto con -O3.

C. 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. 2 se presenta una descripción del funcionamiento en pseudocódigo.

El benchmark seleccionado (algoritmo de Cannon [12]) 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.

C.1 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.

C.2 Tamaños de entrada

Se han escogido los siguientes tamaños de entrada para las matrices para tener 4 referencias diferentes, desde un tiempo reducido hasta uno más largo:

- 700×700
- 1000×1000
- 1300×1300
- 1500×1500

VI. Resultados de la experimentación

En la tabla I 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++.

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 alguna 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 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".

VII. Conclusiones

Este trabajo ha consistido en implementar una parte de la interfaz de biblioteca de Hitmap 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++.
- Habilitación de diferentes opciones de mejora del rendimiento de cara al futuro (ver siguiente sección).
- Presentación de resultados para un caso de estudio en el que se muestra el rendimiento, validez y adecuación de las funciones de Hitmap++.

VIII. 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.

Agradecimientos

Este trabajo está parcialmente soportado por la Junta de Castilla y León (proyecto ATLAS, VA172A12-2); MICINN (España) y el programa ERDF de la Unión Europea (proyecto MOGECOPP TIN2011-25639, proyecto HomProg-HetSys TIN2014-58876-P, red CAPAP-H5 TIN2014-53522-REDT)

Fig. 2. Estructura del benchmark

```
function cannonsMM(int Afils, Acols, Bcols)
    BTile tile1 <- new BTile
    BTile tile2 <- new BTile
    ...       ...       ...

    constructor_layout()
    constructor_topology()
    ...       ...       ...

    HitTile hitTile1 <- constructor_HitTile()
    HitTile hitTile2 <- constructor_HitTile()
    ...       ...       ...

    select(hitTile1)
    select(hitTile2)
    alloc(hitTile1)
    alloc(hitTile2)
    ...       ...       ...

    initMatrices(Acols, Bcols, hitTile1,
        hitTile2, hitTile3, layoutA, layoutB)

    comm_processes()
    ...       ...       ...

    tile1 <- HitTile2BTile(hitTile1)
    tile2 <- HitTile2BTile(hitTile2)
    ...       ...       ...

    measure_time()
    matrixProduct(tile1, tile2, tile3)
    measure_time()

    print_elapsed_time()

    free(hitTile1)
    free(hitTile2)
    free(layoutA)
    ...     ...
    free(topology)

end cannonsMM
```

Fig. 3. Ejemplo de acceso y manejo del tile: C y 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 );


// Nueva interfaz C++
// No need to declare a new tile type
Tile<double>A(n,m);
for(...){
  A.elem(i,j) = ; // No need to define the
       number of dimensions
}
// Destructor is called automatically
```

Fig. 4. Ejemplo de acceso y manejo del tile: C y 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(...){
  A.elem(i,j) = ;
}

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

TABLA I

TABLA DE RESULTADOS

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

TABLA II

CLASES Y MÉTODOS DE LA API 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.
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(shape, datatype)`	Constructor del Tile. Crea un objeto tile usando el dominio definido por un objeto shape. El parámetro datatype determina el tipo de datos de los elementos.
	`allocate()`	Reserva memoria para el tile
	`elemAt(x,y,...)`	Método para acceder al elemento en el tile.
	`select(dims,[begin,end]*)`	Método para crear un subtile.

TABLA III

CLASES Y MÉTODOS NUEVOS EN LA API DE HITMAP.

Objeto	**Método**	**Descripción**
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.
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
	`get{Param}()`	Método para obtener el valor de un atributo privado. Param es el nombre del atributo.
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.

Referencias

[1] Message Passing Interface Forum, "MPI: A Message-Passing Interface Standard Version 3.0," Tech. Rep., 2012.

[2] 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*, vol. 25, no. 5, pp. 1145–1154, May 2014.

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

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

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

[6] 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.

[7] 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*, New York, NY, USA, 2009, PGAS '09, pp. 9:1–9:7, ACM.

[8] B.L. Chamberlain, D. Callahan, and H.P. Zima, "Parallel programmability and the chapel language," *Int. J. High Perform. Comput. Appl.*, vol. 21, no. 3, pp. 291–312, Aug. 2007.

[9] Basilio B. Fraguela, Ganesh Bikshandi, Jia Guo, MaríA J. GarzaráN, David Padua, and Christoph Von Praun, "Optimization techniques for efficient hta programs," *Parallel Comput.*, vol. 38, no. 9, pp. 465–484, Sept. 2012.

[10] Yifeng Chen, Xiang Cui, and Hong Mei, "Parray: A unifying array representation for heterogeneous parallelism," *SIGPLAN Not.*, vol. 47, no. 8, pp. 171–180, Feb. 2012.

[11] 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*, pp. 242–250, 2010.

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