Mecanismo de equilibrado de carga en sistemas heterogéneos

Fernando Alonso [1] , Arturo Gonzalez-Escribano[2], Yuri Torres[2] y Diego R. Llanos[2]

Resumen— **El reparto de la carga de trabajo en sistemas heterogéneos es una tarea complicada ya que no todos los nodos de un sistema contienen los mismos recursos computacionales. El tipo de distribuciones de datos más comúnmente utilizado es el reparto equitativo entre todos los procesos. Sin embargo, para los sistemas heterogéneos es necesario una política de reparto más sofisticada. Este trabajo propone la integración de un sistema de reparto de la carga computacional por pesos en una herramienta programación paralela distribuida. Para utilizarlo se indica el porcentaje de carga total que se desea otorgar a cada proceso. De esta forma, se puede explotar al máximo la capacidad de cómputo de cada uno de los dispositivos que componen el sistema. Nuestro estudio experimental muestra que nuestra propuesta se adapta perfectamente al contexto del modelo de programación escogido, consiguiendo una mejora de rendimiento significativa comparado con una distribución equitativa de la carga de trabajo.**

Palabras clave— **Balanceo de Carga, Computación Heterogénea, Distribución de la Carga, HPC, Particionado de Datos.**

I. Introducción

Actualmente, los sistemas heterogéneos proporcionan una gran capacidad computacional y bajo coste energético ya que son capaces de explotar, de forma conjunta, dispositivos de naturaleza diferente, tales como CPUs, GPUs, FPGAs [1, 2]. La programación de este tipo de sistemas requiere un conocimiento detallado de los recursos hardware que componen el sistema, tanto para gestionar y explotar cada dispositivo como para asignar la cantidad de carga computacional a cada uno de ellos.

Realizar una distribución correcta de la carga en un sistema heterogéneo no es una tarea sencilla. En general, las técnicas de particionado y mapeado de datos más comunes, se apoyan habitualmente, en una distribución equitativa de los datos por nodo. En un sistema heterogéneo, un particionado de datos equitativo no aprovecha, de forma eficiente, toda la capacidad de cómputo debido a la diferencia de recursos hardware de cada uno de las unidades de cómputo.

Hitmap [3] es una librería desarrollada por el Grupo de Investigación Trasgo [4], pensada para el particionado de datos (tiling) jerárquico y el mapeado de arrays y estructuras dispersas. Hitmap propone un array multidimensional a través de un nuevo tipo de dato, el Tile. Hitmap realiza una distribución de Tiles o SubTiles automática entre los diferentes nodos de un sistema ya que enmascara todas las operaciones explícitas de particionado, reparto y comunicación de datos a través de una interfaz, sencilla de utilizar, en lenguaje C. Aunque se puede elegir entre múltiples tipos de particionado, a través de los Layouts (bloques, cíclica, por dimensión, particionado para estructuras dispersas, etc. . .). Sin embargo, no es posible realizar una distribución de datos en los que se puede elija la carga o el volumen de datos que se le otorga a cada nodo.

Este trabajo propone el desarrollo de un plug-in para la herramienta Hitmap, que permita realizar un particionado de datos por pesos para distribuir la carga computacional proporcional a la capacidad computacional de cada uno de los nodos de un sistema heterogéneo. El plug-in permite un reparto en datos de una y varias dimensiones así como la elección el peso de los datos por nodo y por proceso.

El trabajo está organizado en las siguientes secciones: En la sección II se muestran los trabajos relacionados. La sección III describe la librería Hitmap. La sección IV ropuesta del trabajo. La sección V describe los detalles de la implementación. La sección VI contiene los resultados experimentales. Finalmente, la sección VII presenta las conclusiones obtenidas y el trabajo futuro.

II. Trabajo relacionado

Aumentar la localidad de los datos respectos a los elementos de cómputo que los utilizan permite reducir los costes de comunicaciones o de acceso a los datos. Las técnicas orientadas a ello se han convertido en un elemento clave para la paralelización efectiva de aplicaciones. Especialmente en el contexto de los actuales sistemas paralelos, cada vez más heterogeneos, más complejos en sus jerarquías de memoria, con espacios de datos separados y/o distribuidos [5].

En modelos de programación clásicos, como HPF (High-Performance-Fortran) [6, 7] ya se considera la idea de expresar afinidades entre elementos de estructuras de datos y de proceso, o elementos de estructuras de datos que deben ser mapeadas en el mismo elemento de proceso. La mayor parte de las técnicas propuestas en este sentido se han orientado a un análisis en tiempo de compilación.

Un objetivo perseguido por diversas propuestas de lenguajes o modelos de programación paralela ha sido introducir abstracciones que separen las especificaciones de los algoritmos de las técnicas de partición y distribución de las estructuras de datos. De esta forma, el programador puede probar diversas estrategias de partición sin necesidad de reorganizar el código por completo. Esto es muy evidente, por ejemplo, en los modelos de programación de tipo PGAS, como Chapel [8], STAPL [9] o DASH [10].

[1]Univ. Valladolid, e-mail: fernando.alonso@alumnos.uva.es
[2]Dpto. de Informática, Univ. Valladolid, e-mail: {yuri.torres|arturo|diego}@infor.uva.es

En cuanto a las técnicas de partición y distribución especificamente orientadas a sistemas heterogéneos, ya en [11], se muestran algunas técnicas que permiten realizar una asignación de carga a cada nodo de una forma eficiente, aunque no se diseñe una política o mecanismo de particionado. En el año 2005, Dongarra et.al [12] se focalizan en la computación heterogénea y los clústers, describiendo el trabajo futuro por hacer. En [13] se realiza un estudio del estado del arte sobre la computación heterogénea, centrándose en el uso de algunos co-procesadores como GPUs, FPGAs junto con los procesadores tradicionales para la resolución de problemas en sistemas heterogéneos. Más modernamente, el trabajo de Unat et.al [5] clasifica las técnicas actuales sobre localidad de datos en general, mostrando las tendencias actuales a utilizar sistemas dinámicos de equilibrado de carga, pero basados en tareas de grano fino o medio, con el consiguiente sobrecoste en el sistema de ejecución.

Nuestra propuesta se presenta como una técnica sencilla de reparto de datos en grano grueso, apropiado para sistemas distribuidos o con alto coste de transferencia de datos. Se integra de forma modular dentro de un sistema de programación paralela, sin necesidad de reestructurar el código para utilizarla. Se aplica en tiempo de ejecución, permitiendo adaptar la computación a la plataforma sin necesidad de recompilar el código.

III. Librería Hitmap

En esta sección se introducirán algunos conceptos sobre la librería Hitmap [3], tales como la arquitectura y Layouts de la misma.

A. Conceptos clave

Hitmap es una librería altamente eficiente pensada para particionado de datos (tiling) jerárquico y mapeado de arrays y estructuras dispersas. [14] Hitmap introduce los siguiente conceptos:

Signature: Una *signature* S se define como una terna que representa un subespacio de índices sobre un array unidimensional. La cardinalidad de una signature es el número de índices diferentes del dominio.

$$S \in \text{Signature} = (\text{begin} : \text{end} : \text{stride})$$
$$\text{Card}(S) = \lfloor (s.end - s.begin + 1)/s.stride \rfloor$$

Shape: Un *shape* h es una n-pla de signaturas. Representa una selección de un subespacio de los índices de un array con dominio multidimensional. La cardinalidad de una shape es el número de diferentes combinaciones de índices del dominio.

$$h \in \text{Shape} = (S_0, S_1, S_2, \ldots, S_{n-1})$$
$$\text{Card}(h \in \text{Shape}) = \prod_{i=0}^{n-1} \text{Card}(S_i)$$

Tile: Un *tile* es un array n-dimensional. Su dominio es definido por un shape, y tiene un tiene un número de elementos de un tipo (*type*) dado.

$$\text{Tile}_{h \in \text{Shape}} : (S_0 \times S_1 \times S_2 \times \ldots S_{n-1}) \rightarrow \texttt{<type>}$$

B. Arquitectura

La figura 1 muestra la arquitectura de la librería Hitmap.

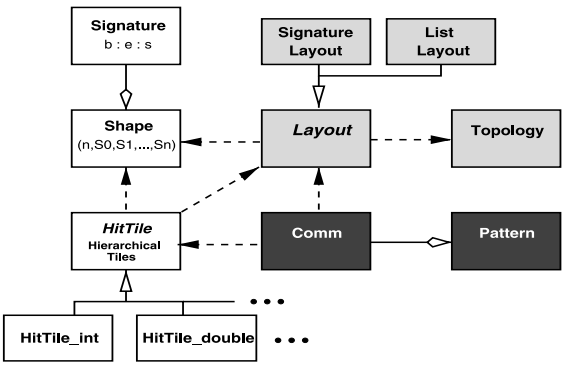

Fig. 1
Arquitectura de Hitmap

Existen tres conjuntos de funcionalidades distinguibles en Hitmap:

- ***Tiling:*** Parte encargada de la definición y manipulación de arrays y tiles. Los HitTiles, que podemos clasificar de forma general como arrays multidimensionales, son la estructura de datos principal en la que se fundamenta Hitmap. Los HitTiles pueden ser de cualquier tipo básico e incluso de nuevos tipos definidos por el usuario y están definidos por un Shape. Dichos Shapes, incluyen una Signature por cada dimensión que tiene el Tile. De esta forma, un HitTile, bidimensional que contuviese, por ejemplo, 10 posiciones en cada dimensión, contaría con un Shape, de dos elementos, que llevaría la asignatura $0 : 9 : 1$ asociada tanto en S_0 con en S_1.

- ***Mapeo:*** Estas funciones se encargan de la distribución y disposición de datos. A través de la topología (*Topology*), se elige una una distribución de los procesadores determinada. Algunos ejemplos de Topologies son: plana (1 dimensión), rectangular o cuadrada (2 dimensiones). De esta forma, los procesos, pueden intercambiar datos en cualquiera de las direcciones que permita la Topology. Los Layouts, en cambio, nos permiten elegir una distribución de la carga determinada, según el propio Layout, a través de los procesos y de forma automática. Para poder utilizar un Layout sobre un HitTile, se necesita establecer una Topology. Mediante los Layouts, los procesos pueden conocer que parte del Tile es la suya. Además, existen varios Layouts diferentes, dependiendo de como se quiera repartir la carga. Algunos ejemplos son: plug_lay_Blocks, que

reparte el Tile por bloques de forma equitativa entre todos los procesadores o plug_lay_Cyclic, que reparte el Tile de forma cíclica y equitativa entre todos los procesos.

- ***Comunicación:*** Por último, la librería también incluye toda una parte de comunicaciones. Mediante la información que proporciona el Layout y la Topology (incluida en el Layout), las funciones de comunicación de Hitmap permiten el intercambio de Tiles o elementos de Tiles entre procesos, de una forma sencilla. Además, en HitMap, existen los Communication Patterns, que permiten establecer un patrón o secuencia de comunicaciones básicas de forma automática. El Comm. Pattern, almacena la secuencia de comunicaciones que se desea realizar, y, simplemente, permite ejecutar dicha secuencia de comunicaciones en un punto del programa, de una forma sencilla.

IV. Propuesta

Estas sección presenta la propuesta de equilibrado de carga para sistemas heterogéneos.

A. Modelo propuesto

Hitmap dispone de varios tipos de distribuciones de carga, a través de los Layouts. Sin embargo, no incluye ninguna forma de reparto en la que se pueda especificar distintos tamaños de carga para cada proceso.

El propósito de este trabajo es dotar a Hitmap de un reparto de carga por bloques balanceado según el peso que se le desee otorgar a cada proceso. De esta forma, se puede equilibrar la carga computacional que tiene cada proceso, indicándole el peso que queremos que tenga respecto a la estructura de datos original. Dicho reparto de carga puede realizarse respecto a una dimensión del Tile. Para estructuras unidimensionales, simplemente, se le asigna a cada proceso una parte del Tile que corresponda con el peso otorgado. En la figura 2 se muestra un ejemplo de la distribución de la carga por pesos en un Tile de 10 elementos $(0:9:1)$ de una sola dimensión y tres procesos distintos. A Proc 1 se le otorga un peso de 0.5 del Tile, a Proc 2 se le proporcionó un peso de 0.2 y a Proc 3 0.3.

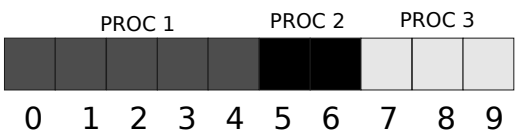

Fig. 2
Tile distribuido por pesos entre tres procesadores

Para los HitTiles multidimensionales, se permite repartir por pesos en una sola dimensión en cada particionado, es decir, repartimos sola la signatura de una dimensión por pesos manteniendo las otras signaturas de las dimensiones restantes igual que en el HitTile original. Por ejemplo, dado un Tile T, con un Shape $S/S \in \text{Shape} = (0:9:1, 0:9:1, 0:9:1)$, si se quiere repartir la primera dimensión (S_0) en $[0.5, 0.2, 0.3]$, entre los procesadores 1,2, y 3, como se ha hecho en el caso de un Tile unidemensional, los subtiles que obtendrá cada proceso, tendrían los siguientes shapes:

$$\textit{Proceso 1: } S_{P1} = (0:4:1, 0:9:1, 0:9:1)$$
$$\textit{Proceso 2: } S_{P2} = (5:6:1, 0:9:1, 0:9:1)$$
$$\textit{Proceso 3: } S_{P3} = (7:9:1, 0:9:1, 0:9:1)$$

De esta manera, se permite repartir por filas o columnas en una estructura bidimensional. Para repartir un HitTile multidimensional por varias dimensiones con diferentes pesos en cada dimensión, se puede aplicar la partición n veces, una por cada dimensión, y, conseguir una sistema de reparto por pesos jerárquico. En la figura 3, se muestra una matriz o Tile de dos dimensiones, distribuido entre tres nodos por filas y dentro de cada nodo se ha vuelve a aplicar la partición en cada respectivo subtile por columnas. De esta manera, se pueden realizar repartos por pesos por nodos y dentro de cada nodo de los nodos un reparto por pesos entre los procesos de dicho nodo.

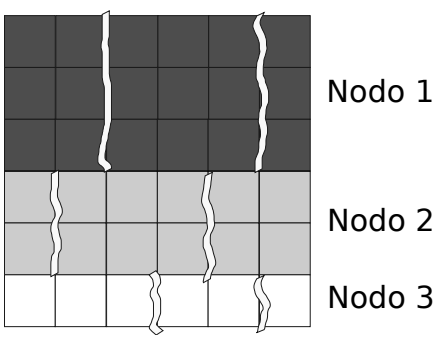

Fig. 3
Tile multidimensional distribuido por pesos

La forma correcta de agregar nuevos tipos de particionado de datos en Hitmap, es mediante la implementación de nuevos Layouts. Con la creación de un nuevo Layout, conseguimos integrar en la librería Hitmap un nuevo tipo de reparto de carga. En este caso, se realizarán dos nuevos Layouts. El primero y principal, permitirá el reparto por bloques, dados unos determinados pesos y una determinada dimensión, y operará como acabamos de explicar en los párrafos anteriores. También, implementaremos otro Layout que nos permitirá un reparto por pesos solamente en una dimensión, y, las dimensiones restantes repartirá con Layout de reparto de bloques simple, la carga. Profundizaremos en los detalles de ambos, en la sección siguiente.

Ambos Layouts, poseerán un parámetro que será, un array de números reales (float), en el que se indicarán los ratios de carga deseados para cada pro-

ceso. La posición en el array, indicará el número de proceso al que va destinado el peso almacenado. Para distribuir la carga entre nodos, debemos hacerlo de forma manual, es decir, comprobando que procesadores pertenecen a cada nodo y asignándoles un peso. De esta forma, se puede otorgar a los procesos de los nodos más lentos de un sistema heterogéneo, una carga computacional menor, y, por el contrario, a los nodos más rápidos un mayor ratio del HitTile, permitiendo una mejor explotación del sistema y sus recursos.

B. Integración con Hitmap

Se desarrollar un sistema para el balanceo de carga en sistemas heterogéneos mediante la adición de un plug-in a la librería Hitmap debido a las razones siguientes:

En primer lugar, la librería Hitmap ya posee toda la funcionalidad necesaria para el particionado de datos jerárquico y el mapeado de arrays, es decir, tenemos una librería que nos provee la funcionalidad de particionado, reparto de carga, comunicaciones, etc... de forma transparente al usuario. Los Layouts de Hitmap están diseñados para poder ampliarse sin necesidad de modificar otras partes de la librería reduciendo así, la tarea de programación.

En segundo lugar, se pueden ejecutar todas las aplicaciones que sean desarrolladas con el nuevo sistema de balanceo de carga con la librería Hitmap únicamente seleccionando el Layout específico y deseado para dicho código. Como se muestra en la figura 4, declarar un Layout en Hitmap es tan sencillo como llamar a *hit_layout*, con los argumentos que necesite el Layout. Es necesario cambiar la declaración de la primera línea por la declaración de la segunda (figura 4).

```
1 HitLayout oneLayout = hit_layout(
      ↪ plug_layBlocks, topo, shape );
2 HitLayout otherLayout = hit_layout(
      ↪ plug_layBlocksWeighted, topo, shape,
      ↪ dim, weights);
```

Fig. 4
Ejemplo de declaración de Layouts

Por último, el HitTile, de la librería Hitmap se utiliza como estructura de datos base en la librería Controller [15], también desarrollada por el Grupo de Investigación Trasgo. Por ello, la distribución balanceada de carga se puede utilizar también en Controller sin tener que desarrollar específicamente el software necesario para implantarla en la librería.

V. Detalles de Implementación

En esta sección se muestra la implementación final del plug-in para Hitmap. Dicho plug-in consiste en el desarrollo de dos Layouts que reparten la carga por pesos entre procesos.

A. Layout plug_layDimBlocksWeighted

Mediante el uso de este Layout se realiza un reparto de la carga por bloques siguiendo los pesos indicados para cada proceso en una sola dimensión. Se deja el resto de las dimensiones completas para cada proceso. Este tipo de partición queda descrito en sección IV-A como modelo principal de reparto por pesos.

```
1 /* plug_layDimBlocksWeighted */
2 typedef struct
3 {
4    int num_procs;
5    float *ratios;
6 } Load_ratios;
7
8 HitLayout hit_layout( plug_layDimBlocksWeighted
      ↪ , HitTopology topo, HitShape shape, int
      ↪  restrictDim, Load_ratios weights);
```

Fig. 5
Interfaz del Layout plug_layDimBlocksWeighted

En la figura 5 se muestra la interfaz del Layout. A través de Load_ratios, se le indica el número de procesos totales, junto con el peso de cada proceso. Cada posición del array, indica el número de proceso, y, el contenido de la posición, el peso que se le da a dicho proceso. El parámetro restrictDim, indica la dimensión a la que se va a aplicar el Layout por pesos. Dichos pesos, no han de sumar 1 necesariamente. El ratio de carga de cada procesador será siempre $\frac{weights[proc]}{\sum_{i=0}^{procs-1} weights[i]}$, por lo que, se puede usar la escala que se desee. En cuanto al número de procesos, solo se activarán aquellos que se pase como parámetro a Load_ratios, es decir, si se pasamos 4 procesos y el programa se lanza con 8 procesos, estarán activados $[0, 3]$ y desactivados $[4, 7]$. Si, por el contrario, el número de procesos que se le pasa al Layout es mayor que el número de procesos con los que ha lanzado el programa, se ignorarán los pesos que no pertenezcan a ningún proceso real.

B. Layout plug_layBlocksWeightedToSelectedDim

Por otra lado, este segundo Layout, realiza la repartición de la dimensión seleccionada por pesos, y las demás dimensiones las reparte por bloques de forma equitativa. Está pensado para realizar repartos sencillos y rápidos en Topologies 2D. De esta forma, una de las dimensiones se parte por pesos y la otra u otras equitativamente.

Por ejemplo, si tenemos una Topology 2D de 4×2 procesadores y, un HitTile de 10×10 elementos y, seleccionamos la dimensión 0 (filas) para el reparto balanceado, con los pesos $[0.3, 0.3, 0.2, 0.2]$ para cada proceso, obtendremos los siguientes elementos:

$$\begin{aligned}
&Procesos\ [P_{00}, P_{01}] \rightarrow 0.3 * 1/2 * 100 = 15\\
&Procesos\ [P_{10}, P_{11}] \rightarrow 0.3 * 1/2 * 100 = 15\\
&Procesos\ [P_{20}, P_{21}] \rightarrow 0.2 * 1/2 * 100 = 10\\
&Procesos\ [P_{30}, P_{31}] \rightarrow 0.2 * 1/2 * 100 = 10
\end{aligned}$$

En laTabla I podemos ver el Shape exacto que obtendría cada proceso, en el ejemplo:

TABLA I
SHAPE DE CADA PROCESO DE UNA TOPOLOGY 2D DE 4×2 PROCESADORES Y, UN HITTILE DE 10×10 ELEMENTOS

	0	1
0	(0:2:1,0:4:1)	(0:2:1,5:9:1)
1	(3:5:1,0:4:1)	(3:5:1,5:9:1)
2	(6:7:1,0:4:1)	(6:7:1,5:9:1)
3	(8:9:1,0:4:1)	(8:9:1,5:9:1)

En cuanto a la interfaz de este Layout, es similar a la anterior, y los campos necesarios también son iguales. A través de Load_ratios, se le indica el número de procesos totales, junto con el peso de cada proceso en la dimensión seleccionada. El parámetro selectedDimToWeight, indica la dimensión a la que se va a aplicar el Layout por pesos. Dicha interfaz se muestra en la figura 6

```
1 /* plug_layDimBlocksWeighted */
2 typedef struct
3 {
4 int num_procs;
5 float *ratios;
6 } Load_ratios;
7
8 HitLayout hit_layout(
      ↪ plug_layBlocksWeightedToSelectedDim,
      ↪ HitTopology topo, HitShape shape, int
      ↪ selectedDimToWeight, Load_ratios
      ↪ weights);
```

Fig. 6
INTERFAZ DEL LAYOUT PLUG_LAYBLOCKSWEIGHTEDTOSELECTEDDIM

VI. ESTUDIO EXPERIMENTAL

En esta sección se realizará un estudio experimental para comprobar la aceleración obtenida en un cluster heterogéneo, utilizando un reparto de carga balanceado según la capacidad computacional del nodo.

Para realizar la experimentación, vamos a emplear tres máquinas diferentes del cluster del Grupo Trago, que es heterogéneo. Utilizaremos 12 procesos en cada una, ya que, es el máximo de una de estas máquinas, y, para esta experimentación, usaremos el mismo número de procesadores en todos los nodos (en este caso 3). Estas máquinas son Manticore, Heracles e Hydra. Sus especificaciones son las siguientes:

Maticore:
- CPU: 2 x Intel Xenon Platinum 8160 @ 2.10GHz (96 cores), memoria: 256 GB DDR4 y coprocesaores: NVIDIA TESLA V100.

Heracles:
- CPU: AMD Opteron 6376 @ 2.3 GHz (64 cores) y memoria: 250 GB.

Hydra:
- CPU: 2 x Xenon E5-2690v3 @ 1.9 GHz (12 cores) memoria: 64GB y coprocesaores: 4 x NVIDIA GTX TITAN BLACK 2880.

En orden decreciente, Manticore, Heracles e Hydra son las máquinas con más recursos computacionales. Otorgaremos más carga computacional a los nodos con mejor capacidad de procesamiento. Por ello, la distribución de carga será la siguiente:

15 % de la carga se dispondrá en Hydra.
38 % de la carga se ubicará en Heracles.
47 % de la carga se situará en Manticore.

En cada máquina, los 12 procesos tendrán el mismo ratio de carga.

Para verificar la velocidad de ejecución en diferentes entradas se utilizara la aplicación Stencil de Jacobi en 2D [16], que ya ha sido analizado, resuelto y probado en Hitmap. Mostraremos la mejora en tiempo de ejecución al utilizar una distribución en bloques por pesos frente a usar un reparto equitativo por bloques entre los tres nodos elegidos del cluster. Utilizaremos, por una parte, el Layout mostrado en la sección V-A, para medir el tiempo de ejecución cuando se reparte la matriz por filas de forma balanceada, según la carga indicada, entre todos los procesos. Por otro lado, utilizaremos *lay_DimBlocks* como Layout de reparto por bloques y por dimensión de forma equitativa, para medir el tiempo de ejecución de un reparto por filas de la matriz entre todos los procesos.

A. Resultados

Medimos el tiempo de ejecución empleado por el ejemplo del Stencil de Jacobi en 2D, con ambos Layouts (equitativo y balanceado por pesos). Por una parte, a medida que aumentan las iteraciones, con un tamaño de matriz fijo figura 7. Por otro lado, con un número de iteraciones fijo (1000), a medida que aumenta el tamaño de la matriz.

En la figura 7 se muestra el resultado de una serie de ejecuciones con un tamaño de matriz fija, 3000×3000, lo suficientemente grande como para que los 36 procesos utilizados, tengan la suficiente capacidad de cómputo en cada iteración. Se puede observar como la mejora al realizar un reparto de la carga balanceado según las capacidades computacionales de cada nodo, es cada vez mayor. El tiempo por iteración es más corto,ya que, la carga está mejor distribuida. Por tanto, la mejoría es más notable cuantas más iteraciones. En el último punto, 10 0000 iteraciones, se alcanza la aceleración máxima de un 110 % con 520 segundos en el caso del reparto equitativo y 247 segundos en el caso de un reparto por pesos.

En la figura 8 se muestra el resultado de una serie de ejecuciones con un número de iteraciones fijo de 1000, y un tamaño n, variable, de la matriz $(n \times n)$. En la figura 8 podemos fácilmente apreciar, que, la mejora del un Layout por pesos, se aprecia en matrices generalmente grandes, que es, dónde un número grande de procesos, como 36, tienen una carga por

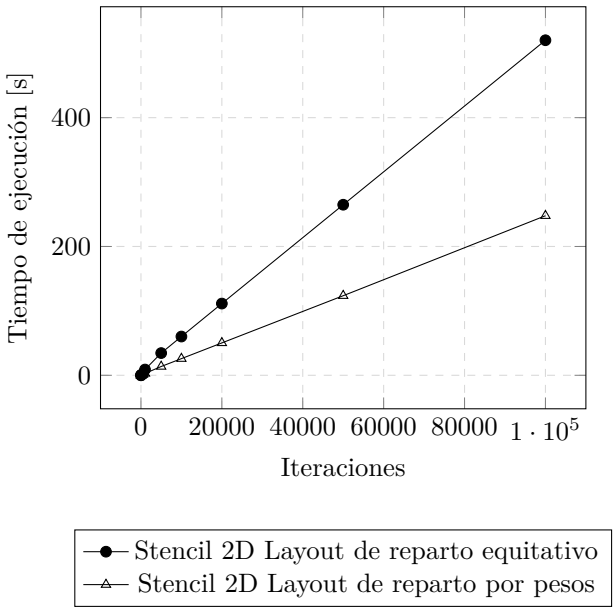

Fig. 7
Tiempo de ejecución por iteraciones, tamaño de la matriz fija: 3000×3000

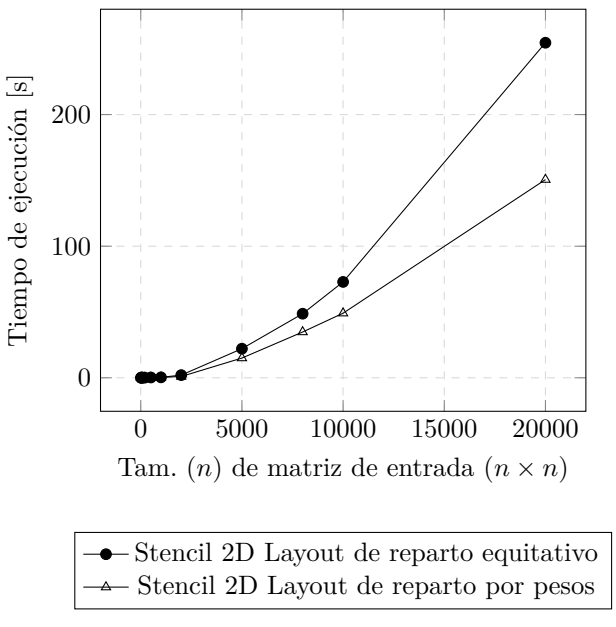

Fig. 8
Tiempo de ejecución por tamaño de la matriz, iteraciones fijas: 1000

iteración elevada. Sin embargo, en matrices pequeñas prácticamente no existe mejora. En estos casos el resultado podría empeorar. Esto es debido a que cuando la carga por iteración es pequeña, con un reparto no equitativo, podríamos dejar algunos nodos desactivados o con muy poca carga computacional comparada con otros, resultando en una espera de los nodos menos potentes a los nodos más potentes. La mejora máxima, se alcanza con la matriz de 20000×20000, una aceleración del 70 % con 154 segundos en el caso del reparto equitativo y 250 segundos en el caso de un reparto por pesos.

En general, podemos observar que, en problemas grandes el rendimiento puede llegar a ser incluso el doble, como se ha visto en la figura 7, al realizar un buen reparto de carga en un sistema heterogéneo.

VII. Conclusiones y Trabajo Futuro

Este trabajo presenta una propuesta del equilibrado de carga en sistemas heterogéneos. Se ha desarrollado un plug-in para HitMap que permite realizar una distribución de carga balanceada según los pesos que se indiquen para cada proceso. Esto permite efectuar un particionado no equitativo en sistemas en los que los nodos no poseen la misma capacidad de procesamiento. El uso del modelo que se ha propuesto, ha llegado a obtener una aceleración del 110 % con respecto a un reparto equitativo por bloques de la carga computacional.

El trabajo futuro incluye, en primer lugar, agregar un reparto entre nodos por identificación del nodo, permitiendo un grano más grueso de particionado. En segundo lugar, se pretende realizar de forma automatizada el ajuste de los pesos que cada proceso debe llevar, en tiempo de ejecución, partiendo de un cálculo previo. Por último, se pretende integrar todo este sistema de particionado automático con la librería Controller para su uso en aceleradores hardware.

Referencias

[1] Isaac Gelado, John E. Stone, Javier Cabezas, Sanjay Patel, Nacho Navarro, and Wen-mei W. Hwu, "An asymmetric distributed shared memory model for heterogeneous parallel systems," *SIGPLAN Not.*, vol. 45, no. 3, pp. 347–358, Mar. 2010.

[2] C. Luk, S. Hong, and H. Kim, "Qilin: Exploiting parallelism on heterogeneous multiprocessors with adaptive mapping," in *2009 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Dec 2009, pp. 45–55.

[3] 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.

[4] "Grupo Trasgo – Trasgo Research Group (Universidad de Valladolid) - Home Page," .

[5] Didem Unat, Anshu Dubey, Torsten Hoefler, John Shalf, Mark Abraham, Mauro Bianco, Bradford L. Chamberlain, Romain Cledat, H. Carter Edwards, Hal Finkel, Karl Fuerlinger, Frank Hannig, Emmanuel Jeannot, Amir Kamil, Jeff Keasler, Paul H J Kelly, Vitus Leung, Hatem Ltaief, Naoya Maruyama, Chris J. Newburn, and Miquel Pericas, "Trends in data locality abstractions for hpc systems," *IEEE TPDS*, vol. 28, no. 10, pp. 3007–3020, Oct 2017.

[6] D.B. Loveman, "High performance fortran," *Parallel & Distributed Technology: Systems & Applications*, vol. 1, no. 1, pp. 25–42, Feb 1993.

[7] I. Foster, "Task parallelism and high-performance languages," *IEEE Parallel & Distributed Technology*, pp. 27–36, Fall 1994.

[8] B.L. Chamberlain, S.J. Deitz, D. Iten, and S-E. Choi, "User-defined distributions and layouts in Chapel: Philosophy and framework," in *2nd USENIX Workshop on Hot Topics in Parallelism*, June 2010.

[9] Antal Buss, Harshvardhan, Ioannis Papadopoulos, Olga Pearce, Timmie Smith, Gabriel Tanase, Nathan Thomas, Xiabing Xu, Mauro Bianco, Nancy M. Amato, and Lawrence Rauchwerger, "STAPL: standard template adaptive parallel library," in *SYSTOR'10*. 2010, ACM.

[10] Karl Furlinger, Colin Glass, Jose Gracia, Andreas Knupfer, Jie Tao, Denis Hunich, Kamran Idrees, Matthias Mai-

terth, Yousri Mhedheb, and Huan Zhou, "DASH: data structures and algorithms with support for hierarchical locality," in *Euro-Par 2014*. 2014, pp. 542–552, Springer.

[11] Christopher A. Bohn and Gary B. Lamont, "Load balancing for heterogeneous clusters of PCs," *Future Generation Computer Systems*, vol. 18, no. 3, pp. 389–400, Jan. 2002.

[12] J. Dongarra, T. Sterling, H. Simon, and E. Strohmaier, "High-performance computing: clusters, constellations, MPPs, and future directions," *Computing in Science Engineering*, vol. 7, no. 2, pp. 51–59, Mar. 2005.

[13] Andre R. Brodtkorb, Christopher Dyken, Trond R. Hagen, Jon M. Hjelmervik, and Olaf O. Storaasli, "State-of-the-art in Heterogeneous Computing," 2010.

[14] "Hitmap – Grupo Trasgo," .

[15] Ana Moreton–Fernandez, Hector Ortega–Arranz, and Arturo Gonzalez–Escribano, "Controllers: An abstraction to ease the use of hardware accelerators," *The International Journal of High Performance Computing Applications*, p. 109434201770296, May 2017.

[16] Sriram Krishnamoorthy, Muthu Baskaran, Uday Bondhugula, J. Ramanujam, Atanas Rountev, and P Sadayappan, "Effective Automatic Parallelization of Stencil Computations," in *Proceedings of the 28th ACM SIGPLAN Conference on Programming Language Design and Implementation*, New York, NY, USA, 2007, PLDI '07, pp. 235–244, ACM, event-place: San Diego, California, USA.