# Incorporación de la Maleabilidad en un Método Iterativo

Iker Martín-Álvarez[1], María Isabel Castillo[1], José I. Aliaga[1] y Diego Andrade[2]

*Resumen*— **Este artículo compara las prestaciones de dos implementaciones paralelas del Gradiente Conjugado con maleabilidad. La maleabilidad permite que una aplicación reconfigure en tiempo de ejecución sus recursos, pudiendo solicitar más recursos para mejorar sus prestaciones o reducirlos para permitir que otras aplicaciones en el sistema puedan iniciar su ejecución. En el primer caso, se pretende mejorar el tiempo de ejecución, mientras que el segundo permite reducir el tiempo de espera de otros trabajos, mejorando la productividad del sistema.**

**Las dos versiones se diferencian en el tipo de comunicaciones que utilizan entre los procesos durante el redimensionado. Una utiliza únicamente comunicaciones síncronas mientras que la otra combina síncronas y asíncronas. Los resultados obtenidos muestran que ambas mejoran el rendimiento de la aplicación paralela original al aumentar el número de procesos asignados obteniendo un rendimiento muy similar entre ellas.**

*Palabras clave*— **Maleabilidad, Programación MPI, SLURM, Métodos iterativos, Sistemas lineales dispersos.**

## I. Introducción

En los centros de cálculo científico, la utilización de los recursos de computación se apoya en un sistema de gestión de colas de trabajos. Cada aplicación/trabajo solicita un conjunto de recursos cuando se envía al sistema de gestión y éste determina el momento de su ejecución en función de la disponibilidad de esos recursos. El modo de funcionamiento de la mayoría de los sistemas actuales es muy rígido y se apoya en dos aspectos clave:

1. **Espera hasta la disponibilidad de todos los recursos solicitados**. Esto provoca situaciones en las que hay disponibles recursos para llevar a cabo la ejecución de un nuevo trabajo, pero no en la cantidad solicitada, y por tanto, el trabajo tiene que continuar esperando, aumentando el tiempo de espera del trabajo y en consecuencia el tiempo de finalización del mismo así como la pérdida de explotación del sistema.
2. **Dedicación de recursos de forma exclusiva**. Habitualmente, los recursos asignados a un trabajo están dedicados en exclusiva durante su ejecución y no hay posibilidad de variación. Sin embargo, la mayoría de las aplicaciones se descomponen en fases y, raramente utilizan todos estos recursos en cada una de ellas.

En general, este modo de funcionamiento, por un lado, aumenta el tiempo de finalización de los trabajos y, por otro, reduce la productividad del sistema. Estos problemas pueden ser solucionados si las aplicaciones a ejecutar son maleables, es decir, son capaces de redimensionar el número de recursos asignados en tiempo de ejecución.

Este procedimiento supone parar la ejecución de la aplicación en un determinado punto, crear nuevos procesos con el nuevo número de recursos, redistribuir los datos desde los procesos originales a los nuevos y finalmente reanudar la ejecución a partir de ese punto. El proceso de maleabilidad puede afectar negativamente al rendimiento de una aplicación en particular, pues las tareas a realizar para llevar a cabo el redimensionamiento de recursos generan un sobrecoste que puede incrementar su tiempo de ejecución. Pero, si se consigue que la reducción del tiempo de espera sea mayor que la penalización en el tiempo de ejecución, se puede reducir el tiempo total de finalización.

En este tipo de aplicaciones también es posible que en aquellos momentos en los que no todos los recursos del sistema se encuentren reservados, los trabajos que estén en ejecución puedan solicitar un aumento de recursos para mejorar su rendimiento. En este caso se espera que el tiempo de ejecución después de asignar un número mayor de recursos sea menor que el que se obtuviese si no se realizará este cambio. Incluso teniendo en cuenta el sobrecoste que supone el redimensionado.

En este trabajo se evalúa el resultado de añadir maleabilidad en la resolución paralela del Gradiente Conjugado. Así pues, el objetivo de este trabajo es analizar como afecta al rendimiento de la aplicación paralela original, la implementación de esta característica. Para realizar este estudio se han considerado dos variantes de maleabilidad, que se muestran en la Figura 1, donde se realiza un redimensionado al doble de recursos. Ambas se diferencian en la forma en la que se redistribuyen los datos según si se utilizan solo comunicaciones síncronas o comunicaciones síncronas y asíncronas. La ventaja de estas últimas es que permiten solapar parte de la redistribución de los datos con el cómputo.

## II. Antecedentes

Los primeros pasos en el desarrollo de aplicaciones maleables aparecen descritos en [1], donde en sistemas de memoria compartida se utilizan políticas para expulsar los trabajos activos y redistribuir la carga de los procesadores entre los trabajos activos y en espera.

Los primeros mecanismos que se utilizaron para implementar maleabilidad estaban basados en los mecanismos *Checkpoint/Restart*. Un ejemplo aparece

[1]Dpto. de Informática, Universidad Jaume I, e-mail: {martini,castillo,aliaga}@uji.es.
[2]Dpto. de Ingeniería de Computadores, Universidade da Coruña, e-mail: diego.andrade@udc.es.

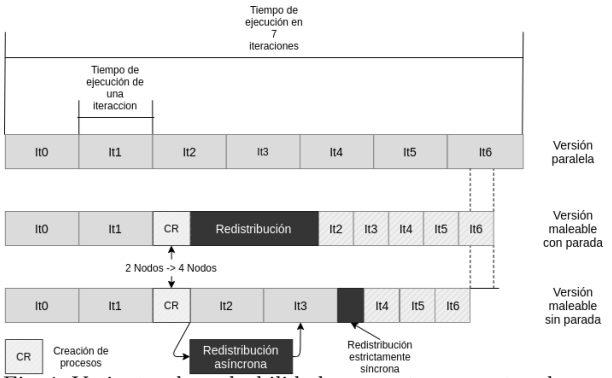

Fig. 1: Variantes de maleabilidad propuestas respecto a la versión paralela.

en [2], donde se investiga la maleabilidad añadiendo un punto de recuperación en la aplicación donde se escriben ficheros de recuperación en almacenamiento secundario, y al reiniciar la aplicación se utilizan estos ficheros con la nueva distribución de procesos.

En [3] se plantean nuevas alternativas para implementar la maleabilidad, una de ellas utiliza la biblioteca SCR (*Scalable Checkpoint / Restart*) para MPI [4], que permite guardar el estado de un proceso en almacenamiento secundario, y relanzar el código MPI con un número diferente de procesos, donde cada proceso sólo busca aquellos archivos de recuperación que coinciden con el identificador del proceso de MPI que guardó dichos ficheros. Además, los procesos pueden solicitar la información de los ficheros cuando la necesiten. Sin embargo, es necesario que el usuario maneje la maleabilidad con la API de la biblioteca SCR.

En [5], se utiliza el entorno EasyGrid AMS para la ejecución de trabajos MPI en grids de clústeres de computadores y plantean estrategias para redimensionar de forma automática esos trabajos a través de una serie de funciones que sirven para determinar los puntos de reconfiguración, calcular el nuevo grado de paralelismo y ejecutar la reconfiguración. Esta estrategia es similar a la que se presenta en [6], basada en la biblioteca Flex-MPI [7], una biblioteca que integra tres nuevas funcionalidades: monitorización, balanceo de carga y redistribución de datos. En este caso, la maleabilidad se centra únicamente en la visión del rendimiento del trabajo.

Existen algunos entornos que plantean la interacción entre aplicaciones maleables y gestores de recursos. Uno de los más estudiados es ReSHAPE [8], que integra las técnicas de reconfiguración de trabajos en la planificación de recursos, pero el entorno requiere que todas las aplicaciones sean maleables [9].

También aparecen infraestructuras que permiten la ejecución de aplicaciones maleables como Elastic MPI[10], que es además una extensión de las APIs de MPICH[1] y Slurm[2]. La integración de ambos sistemas en uno permite a Slurm crear o destruir procesos mientras administra la asignación de recursos. A la implementación de MPICH se le modifican y añaden funciones para definir la aplicación como maleable y para permitir a los procesos comprobar periódicamente si Slurm ha iniciado una reconfiguración.

Trabajos más recientes [11][12][13] han desarrollado herramientas que permiten determinar el número de recursos con los que una aplicación puede empezar a ejecutarse y, más tarde, modificarlos en tiempo de ejecución. Esta variación tiene en cuenta tanto el rendimiento de la aplicación como el del sistema completo. La redistribución de los datos se realiza en un punto de control definido por la aplicación, donde se detiene su ejecución y se realiza la reconfiguración.

## III. Métodos

La implementación de una aplicación maleable requiere la integración de las siguientes consideraciones:

- Punto de redimensionado.
- Mecanismo de comunicación con el gestor de recursos.
- Mecanismo de reconfiguración:
  - Creación y distribución física de los nuevos procesos.
  - Redistribución de los datos hacía los nuevos procesos.
- Otras consideraciones.

A continuación se describe la aplicación paralela sobre la que se implementará la maleabilidad. En los apartados posteriores se detallan que implica cada una de estas consideraciones en la aplicación maleable a desarrollar.

### A. Gradiente Conjugado

El gradiente conjugado (CG) es un método para resolver numéricamente sistemas de ecuaciones lineales ($Ax = b$) de matrices simétricas y definidas positivas [14]. Se trata de un método iterativo que realiza proyecciones ortogonales sobre un subespacio de Krylov, y que se suele utilizar cuando la matriz es dispersa.

En la Tabla I se muestran los diferentes tipos de datos, escalares, vectores y matrices que requiere este método en su resolución. La mayoría de los vectores aparecen distribuidos entre todos los procesos, salvo el vector *d_full* que aparece replicado en todos. Respecto de los escalares, indicar que $n$ define la dimensión del problema mientras que $P$ es el número de procesos de la aplicación.

En la Tabla II se describe un algoritmo paralelo de este método. La primera columna muestra la secuencia de operaciones numéricas que tienen que realizarse para resolver el problema. La segunda incluye el tipo de operación del que se trata. Y en la tercera columna se muestran las comunicaciones necesarias para llevar a cabo la operación. Este método requiere de una operación matriz-vector (SPMV), varios productos escalares de dos vectores con acumulación sobre el primer vector (XPAY) y el segundo vector (AXPY), cálculo de la norma vectorial (NORM) y el producto escalar de dos vectores (DOT).

[1]https://www.mpich.org/
[2]https://www.schedmd.com/

Como se se observa en la Tabla II, el último paso del algoritmo requiere el cálculo de la norma vectorial. Esta se utiliza para comprobar si el método ha convergido. Se considerará una solución factible cuando la norma ($T$) sea menor que un umbral dado ($T\_min$) o cuando se hayan ejecutado un número de iteraciones determinado.

Tabla I: Tipos de los datos en el gradiente conjugado.

| Escalares | Vectores | Matriz dispersa |
|---|---|---|
| n, P, T_min, iter, T, rho, beta, alpha | b, x, z, res, d, d_full | A |

Tabla II: Operaciones del Gradiente Conjugado paralelizado.

| Algoritmo | Operaciones | Comunicaciones |
|---|---|---|
| while (T >T_min) | | |
| z = A * d_full | SPMV | Allgatherv (to: d_full, from: d) |
| rho = d * z | DOT | AllReduce (rho, SUM) |
| rho = beta / rho | Escalar | |
| x+= rho * d | AXPY | |
| res-= rho * z | AXPY | |
| alpha = beta | | |
| beta = res * res | DOT | AllReduce (beta, SUM) |
| alpha = beta / alpha | Escalar | |
| d = alpha * d | XPAY | |
| d+= res | | |
| T = sqrt(beta) | Norma | |

### B. Punto de redimensionado

Determina el lugar en la aplicación donde se incluye el fragmento de código que se añadirá para considerar si se lleva a cabo el mecanismo de comunicación con el gestor de recursos y el redimensionado de la aplicación. La localización de este punto debe asegurar que tras una redimensión sea posible continuar la ejecución como si no se hubiera aplicado la maleabilidad.

En el gradiente conjugado, este punto se puede situar al comienzo o a la finalización de una iteración del bucle de cómputo. De esta forma, en el caso de que se realizase una reconfiguración, el conjunto de procesos padres (grupo de procesos que crean nuevos procesos) habrá terminado una iteración completa del bucle. Este planteamiento simplifica el inicio de la ejecución del nuevo conjunto de procesos hijos (grupo de procesos creados por otro grupo) que comenzarán desde el principio de la siguiente.

Cabe destacar que por motivos de rendimiento, el redimensionado no se lleva a cabo en todas las iteraciones, sino cada vez que se hayan realizado *numIt_PR* iteraciones. En este punto también se incluye esta comprobación.

### C. Mecanismo de comunicación con el gestor de recursos

En esta parte se solicita al gestor de recursos si éste puede aumentar/disminuir los recursos que tiene asignados el trabajo en función de sus necesidades y/o las del gestor.

En este trabajo, este apartado no se considera puesto que el objetivo es analizar el rendimiento de una aplicación maleable desde el punto de vista de la redistribución de los datos.

Así pues, suponemos que siempre existen recursos suficientes disponibles para redimensionar la aplicación y solo se contactará con el gestor para conocer el número de recursos que se están utilizando en un momento dado.

Para más información sobre este procedimiento se puede consultar en [15], en la que se implementa este mecanismo con el fin de mejorar la productividad y reducir el consumo de energía del sistema, disminuyendo el tiempo de espera de los trabajos.

### D. Mecanismo de reconfiguración

Este proceso consta de dos partes. La primera crea y asigna físicamente los nuevos procesos. La segunda parte realiza la redistribución de los datos desde los padres a los hijos.

#### D.1 Creación y distribución física de los nuevos procesos

En primer lugar se define la distribución física de los nuevos procesos sobre los recursos disponibles. Para llevar a cabo esta tarea se han implementado dos políticas distintas:

- *Best-fit*: orientada a equilibrar el número de procesos entre todos los nodos asignados al trabajo.
- *Worst-fit*: orientada a completar la capacidad máxima de un nodo antes de ocupar otro.

A continuación, se crearán y asignarán los procesos según la política elegida entre las anteriores. En cualquiera de ellas se utiliza la función `MPI_Comm_spawn_multiple`, que permite crear un nuevo grupo de procesos con la posibilidad de dividirlos en subgrupos y crearlos con características diferentes. Así, se pueden crear tantos subgrupos como nodos se vayan a usar o bien crear todos los procesos y especificar una información diferente para cada uno de ellos. De esta forma, a cada subgrupo se le puede asignar, por ejemplo, un nodo concreto indicando el nombre del mismo. El conjunto de los subgrupos creado está unido por un único comunicador *MPI_COMM_WORLD* que es distinto al comunicador del grupo de procesos que llama a esta función.

Ambos grupos de procesos, tanto los que llaman a la función como los creados por la misma, se incluyen en un intercomunicador para poder realizar comunicaciones entre ellos. Este será utilizado para la redistribución de datos.

D.2 Redistribución de los datos

Esta fase se puede llevar a cabo utilizando dos mecanismos de comunicación, lo que ha dado lugar a dos implementaciones maleables diferentes. En la primera, el envío de los datos entre los procesos se realiza de forma totalmente síncrona. En la segunda, el conjunto de información a enviar se divide en dos. Una que se enviará de forma asíncrona y la otra síncrona. La principal ventaja de esta última implementación es que mientras se distribuye la información de forma asíncrona, los procesos padres pueden continuar con el cómputo.

En la Tabla III se muestra una clasificación de los datos del CG según su tipo, estructura interna y distribución entre los procesos. Esta tabla no incluye a los escalares *rho* y *alpha* puesto que sus valores son calculados en cada iteración a partir del resto de variables.

El tipo de los datos es el parámetro que va a determinar como se pueden enviar. Esto es, si se modifican en cada iteración (tipo variable), se tienen que enviar de forma síncrona, es decir, con la aplicación detenida, ya que en cada iteración pueden cambiar su valor. En cambio, si no se modifican durante la ejecución (tipo constante), se podrán enviar asíncronamente mientras se está realizando cómputo.

El tipo de distribución indica que función habría que utilizar para enviarlos. Los datos replicados se encuentran en todos los procesos, por lo que se pueden comunicar con la función `MPI_Bcast`. Mientras que para los de tipo distribuidos se utiliza la función `MPI_Alltoallv`. Esta última función es idónea para la redistribución de datos entre ambos grupos de procesos, ya que permite que cada proceso envíe(padres)/reciba(hijos) un número de datos distinto de los otros procesos.

Tabla III: Clasificación de los datos en el CG.

| **Tipo** | **Estructura interna** | **Distribución** | |
|---|---|---|---|
| | | *Replicado* | *Distribuido* |
| *Constante* | *Escalares* | P, n, T_min | - |
| | *Matriz* | - | A |
| *Variable* | *Escalares* | T, beta, iter | - |
| | *Vectores* | d_full | d, res, z, x |

Para reducir las comunicaciones se han utilizado los tipos derivados de MPI ya que permiten enviar conjuntos de datos. En este caso es posible agrupar los datos que están contenidos en una misma celda de la Tabla III y enviarlos con una única comunicación. Esto se cumple para todos excepto para la matriz dispersa $A$, que está compuesta por tres vectores y será necesario realizar tres comunicaciones distintas. La primera distribuye el vector *vlen* para que cada hijo conozca cuantas filas va a recibir. Las dos restantes se utilizan para redistribuir los vectores de índices (*vpos*) y de valores (*vval*).

El vector $d$ no participa en la redistribución puesto que cada proceso recibe el vector *d_full* y, a partir de él, cada proceso puede obtener el correspondiente vector *d*.

Otro aspecto a tener en cuenta es que el orden en el que se hace el envío de los datos es muy importante. Los procesos hijos necesitan conocer a priori los valores $n$ y $P$ para reservar la memoria de otros datos más complejos y poder recibirlos. Así pues, estos dos valores serán los primeros en ser enviados.

A continuación se muestra, para la primera versión maleable que utiliza únicamente la redistribución síncrona de la información, la serie de comunicaciones a realizar y, para cada una de ellas, que datos se comunican en cada paso:

- Enviar con `MPI_Bcast` los valores *P, n, T_min, T, beta, iter*.
- Enviar con `MPI_Bcast` el vector *d_full*.
- Redistribuir con `MPI_Alltoallv` los vectores *res, z, x*.
- Redistribuir con tres operaciones `MPI_Alltoallv` la matriz dispersa A.

A continuación se detalla para la otra versión maleable, el conjunto de comunicaciones asíncronas y los datos que se comunican en cada paso:

- Enviar con `MPI_Ibcast` los valores *P, n, T_min*.
- Redistribuir con tres operaciones `MPI_Ialltoallv` la matriz dispersa A.

Una vez que los padres han iniciado este proceso pueden continuar realizando iteraciones del bucle (cómputo). Tras cada iteración, comprobarán si ha finalizado el envío asíncrono, si no es el caso continúan con el cómputo y, si por el contrario han finalizado, iniciarán el envío síncrono. Esta tarea conlleva:

- Enviar con `MPI_Bcast` los valores *T, beta, iter* y el vector *d_full*. Como los hijos ya conocen su distribución es posible crear un tipo derivado para enviar los cuatro datos juntos.
- Redistribuir con `MPI_Alltoallv` los vectores *res, z, x*.

Cabe destacar que en esta segunda variante, los procesos hijos aprovechan la espera de la recepción asíncrona para reservar la memoria de los datos que reciben en la comunicación síncrona, solapando así ambas tareas.

*E. Otras consideraciones*

Además de la comprobación del punto de redimensionado en el bucle de computo, hay que añadir otra comprobación al inicio del programa que diferencia entre el grupo de procesos que inicializa la aplicación y los grupos de procesos generados en tiempo de ejecución. El primer caso pertenece al código original y es el utilizado en la aplicación no maleable por los procesos que inicializan la aplicación. El segundo es para los nuevos procesos creados en tiempo de ejecución que contactan con los padres para realizar la redistribución de los datos.

Una vez ha finalizado la redistribución de la información entre ambos grupos de procesos, la ejecución debe continuar en los hijos en la siguiente iteración del bucle de cómputo donde se realizó el redimensionado. Por su parte, los procesos padres terminarán su ejecución y liberarán sus recursos.

Finalmente, cuando un grupo de procesos finaliza el bucle de cómputo, hay que comprobar si lo ha hecho porque se ha creado otro grupo que continua con la ejecución o porque ha convergido y se ha obtenido la solución, en cuyo caso se tiene que mostrar el resultado.

## IV. Resultados experimentales

En este apartado se muestra el análisis de rendimiento que se ha realizado de las dos versiones maleables del gradiente conjugado implementadas. En primer lugar, se describirán las características del sistema de computación que se ha usado para realizar dicho análisis.

### A. Sistema de computación y obtención de resultados

El sistema de computación usado para la experimentación ha sido el Finisterrae II, un supercomputador integrado en la Instalación Científico Técnica Singular (ICTS), de la Red Española de Supercomputación (RES) y mantenido por el centro de supercomputación de Galicia (CESGA). El sistema utiliza la versión *17.02.11-Bull.1.0* del gestor de recursos Slurm y para la compilación y ejecución del código se ha utilizado la versión 1.10.7 de OpenMPI.

Los nodos que se han usado en los experimentos son los de tipo *Thin*, que se componen de 2 procesadores Haswell 2680v3 de 24 núcleos con 128 GB de memoria principal, un disco duro de 1 TB, dos conexiones 1 GbE y una conexión Infiniband FDR de 56 Gbps. En total se disponen de 306 nodos de este tipo y están divididos en varias particiones, que pueden ser exclusivas o compartidas. Para los experimentos se ha utilizado la partición *cola-corta*, que proporciona nodos del tipo *Thin* en exclusiva.

Las pruebas se han hecho utilizando una matriz dispersa llamada Audikw [3], que tiene una dimensión de 943,695$x$943,695 y un total de 77,651,847 elementos no nulos.

El número máximo de iteraciones de cómputo, si no se llega a la convergencia del resultado, se ha fijado en 1000 iteraciones y el punto de redimensionado en las versiones maleables se lleva a cabo cada 600 iteraciones. En la matriz Audikw, 1000 iteraciones no son suficientes para llegar a la convergencia, por lo que en los experimentos siempre se ejecutarán ese número de iteraciones.

Las pruebas se realizan con las siguientes configuraciones del sistema:

- Una redistribución física de los procesos en los nodos de acuerdo a los modos *Best-fit* o *Worst-fit*.
- Se utilizan 2, 4, 8, 16 y 24 procesos iniciales.
- En las versiones maleables se redimensiona de $P$ procesos padres a $H$ hijos, con valores 2, 4, 8, 16 y 24 diferentes a *numP*.
- Se utilizan un total de 2 nodos.
- Solo se realiza una reconfiguración por ejecución.
- Los procesos iniciales son asignados al nodo 0.

Cada prueba se ejecuta un total de 10 veces y la medida temporal es la media de todas ellas. Para asegurar que dos tiempos son diferentes se utilizan T-tests [4] con una confianza del 95 %.

### B. Análisis del rendimiento del tiempo de ejecución

En este apartado se realiza un estudio exhaustivo del rendimiento de la versión paralela original frente a las dos versiones paralelas maleables y se presenta un estudio detallado del tiempo empleado en la reconfiguración respecto al tiempo de ejecución del bucle de cómputo. Todas las medidas temporales se expresan en segundos.

En la Figura 2 se representa la escalabilidad fuerte del CG de una versión secuencial frente a la paralela (sin aplicar maleabilidad). Esta gráfica muestra que la eficiencia obtenida es baja y, decrementa aún más a medida que se añaden más procesos en la ejecución. Esto nos lleva a la conclusión que, al aplicar maleabilidad y redimensionar a 16 o más procesos, las versiones maleables no puedan mejorar a la versión paralela original porque la propia aplicación no es muy escalable.

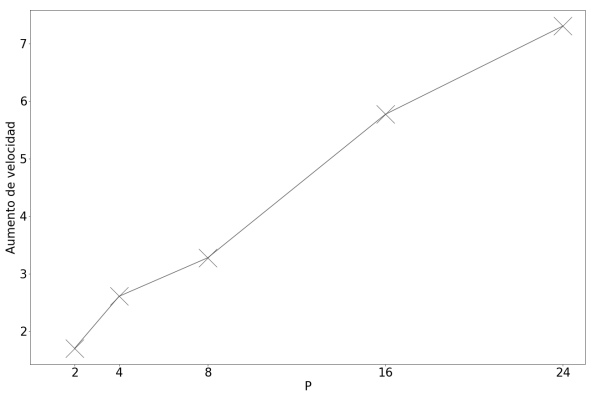


Fig. 2: Escalabilidad fuerte del Gradiente Conjugado.

En la Figura 3 se muestra el incremento de velocidad de las versiones maleables respecto de la versión no maleable. En la gráfica, *Pa* se corresponde con la implementación paralela original y se muestra como una linea horizontal con un aumento de velocidad 1. Las dos columnas, etiquetadas como $S$ y $A$, se corresponden con las versiones maleables síncrona y asíncrona, respectivamente. Además, la gráfica se divide por cuatro líneas horizontales según el número de procesos con los que se inicia la ejecución en las versiones maleables. Estas líneas también determinan el número de procesos con los que se ha realizado la ejecución de la versión paralela original, lo que

[3] https://sparse.tamu.edu/GHS_psdef/audikw_1

[4] https://en.wikipedia.org/wiki/Student%27s_t-test

permite comparar el incremento de velocidad en las diferentes versiones.

Este análisis muestra que en todos los casos hay una mejora de rendimiento cuando se incrementa el número de procesos, excepto al reconfigurar de 16 a 24 procesos. Esto es debido a la escalabilidad de la propia aplicación. Respecto a la comparación entre las dos versiones maleables, se puede observar que los resultados son muy similares, aunque se puede afirmar que, cuando se aumenta el número de procesos en la reconfiguración, es preferible usar la versión síncrona, mientras que cuando se reduce es mejor la asíncrona.

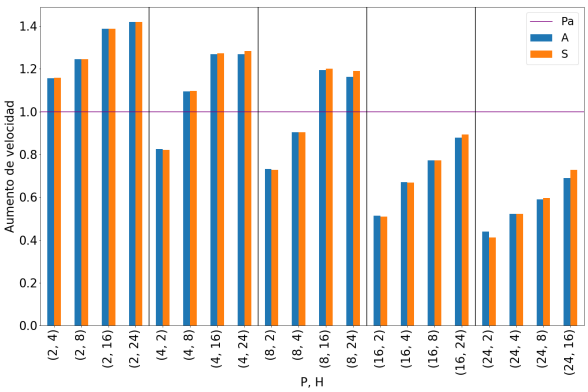


Fig. 3: Aumento de velocidad de las versiones paralelas maleables.

La Tabla IV expone el análisis de cómo influye la distribución física de los procesos sobre los nodos en el rendimiento de la aplicación. En esta tabla se muestran los tiempos de ejecución según el número de procesos padre (P), el número de procesos hijo (H) y el tipo de distribución (*Best-fit* y *Worst-fit*). Así, *Bf* es el tiempo para la distribución *Best-fit* mientras que *Wf* corresponde con *Worst-fit*.

Según estos resultados, la distribución *Best-fit* obtiene mejores prestaciones para todos los casos y sobretodo cuando hay más de 24 procesos involucrados (máximo número de núcleos en un nodo). Esto último se debe a que, si la redistribución es *Worst-fit*, todos los procesos (padres e hijos) se asignan en un único nodo en el momento del redimensionado y habrán más procesos que núcleos. Esto causa que durante un periodo de tiempo los recursos estén compartidos entre padres e hijos y el entorno de MPI brinde peores prestaciones para elegir que procesos pueden ejecutar. Esto no ocurre al utilizar la distribución *Best-fit*, ya que como máximo pueden llegar a tener 24 procesos en cada nodo y cada proceso puede utilizar un núcleo en exclusiva. Este comportamiento es independiente de la versión maleable empleada en la ejecución.

Tras este análisis, los tiempos que se van a mostrar en las siguientes tablas serán para el caso de distribución *Worst-fit*. Para el otro tipo, se presenta el cambio porcentual respecto a la *Worst-fit*.

En las Tablas V y VI se muestra el rendimiento de las dos versiones maleables en su variante síncrona y asíncrona, respectivamente. En la primera se detallan el tiempo de ejecución total(TE) y el tiempo que se

Tabla IV: Tiempos de ejecución según la distribución física de procesos.

| | | **Asíncrona** | | **Síncrona** | |
|---|---|---|---|---|---|
| **P** | **H** | **TE (s) Bf** | **TE (s) Wf** | **TE (s) Bf** | **TE (s) Wf** |
| **2** | **4** | 134,58 | 134,95 | 134,41 | 135,01 |
| | **8** | 123,87 | 126,51 | 123,75 | 126,75 |
| | **16** | 111,65 | 113,24 | 111,72 | 112,99 |
| | **24** | 107,73 | 112,13 | 107,85 | 112,03 |
| **4** | **2** | 122,94 | 124,02 | 123,23 | 124,79 |
| | **8** | 91,95 | 94,22 | 91,4 | 94,52 |
| | **16** | 79,59 | 81,13 | 79,42 | 80,75 |
| | **24** | 75,6 | 84,96 | 75,24 | 83,43 |
| **8** | **2** | 109,55 | 111,74 | 110,51 | 111,9 |
| | **4** | 89,77 | 89,71 | 89,55 | 89,73 |
| | **16** | 67,38 | 68,41 | 67,06 | 68,01 |
| | **24** | 63,56 | 75,92 | 63,04 | 73,2 |
| **16** | **2** | 88,93 | 90,37 | 89,53 | 90,98 |
| | **4** | 68,46 | 68,75 | 68,7 | 68,99 |
| | **8** | 58,27 | 61,01 | 58,16 | 61,05 |
| | **24** | 48,67 | 56,08 | 47,43 | 55,73 |
| **24** | **2** | 80,83 | 84,56 | 87,47 | 89,53 |
| | **4** | 69,8 | 69,61 | 69,42 | 69,61 |
| | **8** | 60,18 | 63,23 | 58,93 | 62,87 |
| | **16** | 51,31 | 54,16 | 47,6 | 52,41 |

invierte en realizar la distribución de los datos de forma síncrona (TS) según el número de procesos padres (P) e hijos (H).

En la tabla de la versión asíncrona se añade más información dado que la redistribución de los datos se lleva a cabo en dos pasos, una parte de forma asíncrona y otra síncrona. Así, en este caso *TS* indica el tiempo que se utiliza para enviar la parte de información que se comunica de forma síncrona, *TA* es el tiempo de comunicación asíncrona, y *Nit* son las iteraciones realizadas por los procesos padres mientras llevan a cabo la comunicación asíncrona. Esta tabla también incluye una última columna que indica el tiempo que se tarda en realizar una iteración del bucle de cómputo principal antes de iniciarse el redimensionado de la aplicación (Tit).

Como se puede observar la versión asíncrona tiene generalmente tiempos más altos para la redistribución de los datos (TS + TA) que la síncrona. Además, cuando hay 24 procesos padres en la redistribución, este tiempo aumenta considerablemente mostrando una baja escalabilidad. El beneficio de la versión asíncrona frente la síncrona está relacionado con el valor de la variable *Nit*, ya que indica el solapamiento que se puede llevar a cabo entre las comunicaciones y el cómputo.

En la distribución *Best-fit*, el tiempo *TE* tiene un decremento porcentual aproximado de 4 % y 3,8 % y el tiempo *TS* de 22 % y 4,5 % en las versiones síncrona y asíncrona, respectivamente. Además, para la versión asíncrona el tiempo *TA* se reduce en un 23 % y la variable *Nit* incrementa el número de iteraciones en un 17 %.

Por otro lado, si se estudian con detalle los tiempos *TA* y *Tit* y se compara con el valor de la variable *Nit*,

se puede observar que el número de iteraciones que se realizan mientras se está llevando a cabo la comunicación asíncrona son mucho menores de las previstas. Un ejemplo se puede observar en la Tabla VI en el que se redimensiona de 24 a 16 procesos. En este caso, el tiempo por iteración (Tit) para 24 procesos es 0,036s y *TA* es 4,246s. El número de iteraciones que se deberían realizar sería un valor próximo a 117, en cambio los resultados muestran que se ejecuta una iteración.

Estas observaciones llevaron a realizar un estudio más detallado del coste temporal de una iteración del bucle de computo de la aplicación paralela y la versión maleable asíncrona mientras se realizaba la comunicación de la matriz (asíncronamente).

Tabla V: Tiempos de ejecución y comunicación de la versión maleable síncrona con una distribución *Worst-fit*.

| P | H | TE (s) | TS (s) |
|---|---|---|---|
| **2** | **4** | 135,012 | 0,281 |
| | **8** | 126,75 | 0,266 |
| | **16** | 112,992 | 0,28 |
| | **24** | 112,029 | 2,011 |
| **4** | **2** | 124,794 | 0,29 |
| | **8** | 94,516 | 0,251 |
| | **16** | 80,754 | 0,295 |
| | **24** | 83,429 | 2,338 |
| **8** | **2** | 111,903 | 0,278 |
| | **4** | 89,728 | 0,26 |
| | **16** | 68,014 | 0,363 |
| | **24** | 73,197 | 5,018 |
| **16** | **2** | 90,984 | 0,292 |
| | **4** | 68,991 | 0,306 |
| | **8** | 61,053 | 0,468 |
| | **24** | 55,735 | 6,806 |
| **24** | **2** | 89,533 | 4,601 |
| | **4** | 69,613 | 4,54 |
| | **8** | 62,867 | 3,987 |
| | **16** | 52,413 | 2,3 |

### C. *Análisis del coste temporal de las iteraciones con maleabilidad asíncrona*

Este estudio analiza el coste que supone el solapamiento de las comunicaciones asíncronas y el cómputo. Por un lado, se medirá en la versión paralela original, el tiempo que supone la ejecución de una iteración completa del bucle de computo y de la operación *Allgatherv* que se realiza en cada iteración de dicho bucle y, por otro lado, se hará lo mismo en la maleable asíncrona mientras se está enviando asíncronamente la matriz.

La Tabla VII muestra este detalle, donde las dos primeras columnas indican el número de procesos padre (P) y procesos hijo (H) que intervienen en el redimensionado. La segunda y tercera columnas muestran respectivamente, los tiempos para una iteración de la aplicación paralela cuando no se redimensiona (PI) a cuando se redimensiona asíncronamente (AI). Las dos siguientes denominadas PG y AG indican el tiempo que conlleva llevar a cabo la comunicación

Tabla VI: Tiempos de ejecución y comunicación de la versión maleable asíncrona con una distribución *Worst-fit*.

| P | H | TE (s) | TS (s) | TA (s) | Nit | Tit (s) |
|---|---|---|---|---|---|---|
| **2** | **4** | 134,949 | 0,024 | 0,4 | 3 | 0,156 |
| | **8** | 126,514 | 0,023 | 0,415 | 3 | |
| | **16** | 113,244 | 0,025 | 0,45 | 2 | |
| | **24** | 112,131 | 0,048 | 1,911 | 2 | |
| **4** | **2** | 124,021 | 0,034 | 0,335 | 3 | 0,102 |
| | **8** | 94,219 | 0,04 | 0,353 | 3 | |
| | **16** | 81,132 | 0,028 | 0,39 | 3 | |
| | **24** | 84,962 | 0,088 | 3,1 | 2 | |
| **8** | **2** | 111,736 | 0,04 | 0,306 | 4 | 0,081 |
| | **4** | 89,708 | 0,061 | 0,307 | 4 | |
| | **16** | 68,409 | 0,055 | 0,375 | 4 | |
| | **24** | 75,919 | 0,16 | 5,765 | 2 | |
| **16** | **2** | 90,368 | 0,063 | 0,291 | 6 | 0,046 |
| | **4** | 68,753 | 0,105 | 0,281 | 5 | |
| | **8** | 61,009 | 0,208 | 0,313 | 5 | |
| | **24** | 56,076 | 0,882 | 7,119 | 1 | |
| **24** | **2** | 84,559 | 0,86 | 2,825 | 38 | 0,036 |
| | **4** | 69,609 | 3,031 | 1,176 | 9 | |
| | **8** | 63,232 | 3,303 | 1,235 | 3 | |
| | **16** | 54,161 | 1,904 | 4,246 | 1 | |

mediante la función *Allgatherv* del CG por un lado cuando no se redimensiona, y por otro lado cuando se solapa con una redimensión asíncrona, respectivamente.

La conclusión que se extrae de este análisis es que, a medida que están involucrados más procesos en la redistribución, más lentamente se ejecutan las iteraciones en la versión asíncrona, siendo el número de padres el factor más determinante.

Una de las principales causas de este problema es el incremento de comunicaciones que se están produciendo en el sistema ya que se que se están solapando tanto las comunicaciones de la redistribución de la matriz de forma asíncrona y las que conlleva la operación *Allgatherv* del bucle de computo, provocando una ralentización de todo el proceso. Se observa que esta operación llega a tener un tiempo hasta 20 veces superior.

En la distribución *Best-fit* el tiempo $AI$ tiene un incremento porcentual aproximado de $3,82\,\%$, mientras que el tiempo $AG$ tiene un decremento porcentual aproximado de $0,1\,\%$.

Este análisis nos lleva a concluir, que una de las principales causas de que la versión asíncrona muestre un rendimiento muy similar y no mejore respecto a la versión síncrona es que el solapamiento de comunicaciones del bucle de computo y de redistribución resulte en peores prestaciones para ambas tareas.

### D. *Análisis de los tiempos de creación de procesos*

El último análisis que se realizó fue el sobrecoste que supone la creación de los nuevos procesos para llevar a cabo el redimensionado. En la Tabla VIII se muestra la media (Media), mediana (Mediana) y desviación estándar (Std) de los tiempos para la creación de procesos en ambas distribuciones físicas, (*Worst-*

Tabla VII: Tiempos de las iteraciones y la operación *Allgatherv* con una distribución *Worst-fit*.

| P | H | PI (s) | AI (s) | PG (s) | AG (s) |
|---|---|---|---|---|---|
| **2** | **4** | 0,156 | 0,1733 | 0,00144 | 0,00144 |
| | **8** | | 0,1732 | | 0,00143 |
| | **16** | | 0,1737 | | 0,00144 |
| | **24** | | 0,3222 | | 0,00144 |
| **4** | **2** | 0,1019 | 0,1151 | 0,00204 | 0,00209 |
| | **8** | | 0,1132 | | 0,00205 |
| | **16** | | 0,1129 | | 0,00205 |
| | **24** | | 0,1346 | | 0,002 |
| **8** | **2** | 0,081 | 0,0855 | 0,0025 | 0,00254 |
| | **4** | | 0,085 | | 0,00259 |
| | **16** | | 0,0832 | | 0,00252 |
| | **24** | | 0,1676 | | 0,0026 |
| **16** | **2** | 0,046 | 0,0498 | 0,005 | 0,00501 |
| | **4** | | 0,0491 | | 0,005 |
| | **8** | | 0,0473 | | 0,00499 |
| | **24** | | 0,2264 | | 0,00513 |
| **24** | **2** | 0,0364 | 0,06 | 0,00777 | 0,00814 |
| | **4** | | 0,2072 | | 0,0082 |
| | **8** | | 0,3904 | | 0,0081 |
| | **16** | | 0,2803 | | 0,008 |

*fit* y *Best-fit*), según el número de procesos padres (P) y el número de hijos (H).

Estos resultados muestran que la distribución física *Worst-fit* es ligeramente más rápida que la *Best-fit* en casi todos los casos, excepto cuando se redimensiona a 24 procesos hijos o desde 24 procesos padres. Esto último se debe a que la distribución *Worst-fit* pasa a tener más procesos en un nodo que núcleos contiene, pasando a un estado de *Oversubscription*. Como se ha explicado anteriormente, está situación conlleva a peores prestaciones mientras se realiza el redimensionado y los tiempos son aun peores cuantos más procesos ocupan ese único nodo.

El problema del *Oversubscription* incide negativamente sobre el valor de la desviación típica, ya que la creación de un mayor número de procesos en el sistema introduce retrasos en su creación. En general, los valores de la media y de la mediana suelen ser muy similares cuando este problema no aparece, siendo habitual que la mediana tome valores menores. Sin embargo, se puede observar que son significativamente diferentes cuando se utiliza la distribución *Worst-fit* desde 24 a 16 procesos, en la que también se obtiene un valor de desviación típica más alta.

También destacar que uno de los factores que más influye en el rendimiento es el número de procesos padre e hijos que intervienen. A un mayor número de procesos padres, menor es el coste de la creación, mientras que un mayor número de procesos hijos resulta en un mayor coste. El número de hijos tiene una mayor repercusión que el de padres. Además, esta tendencia es indiferente del tipo de redistribución física elegida.

Finalmente, cabe indicar que el tiempo de creación de procesos supone un sobrecoste en el proceso de redimensionado de la aplicación paralela que es mayor, en casi todos los casos, que el tiempo necesario para la redistribución de datos. En concreto, cuando el número de procesos padres o hijos que intervienen es elevado.

Tabla VIII: Tiempos para la creación de procesos.

| Distribución | P | H | Media | Mediana | Std |
|---|---|---|---|---|---|
| ***Best-fit*** | **2** | **4** | 0,51 | 0,47 | 0,12 |
| | | **8** | 0,51 | 0,51 | 0,02 |
| | | **16** | 0,56 | 0,55 | 0,02 |
| | | **24** | 0,64 | 0,63 | 0,04 |
| | **4** | **2** | 0,42 | 0,42 | 0,04 |
| | | **8** | 0,49 | 0,43 | 0,15 |
| | | **16** | 0,49 | 0,47 | 0,05 |
| | | **24** | 0,57 | 0,57 | 0,02 |
| | **8** | **2** | 0,4 | 0,39 | 0,02 |
| | | **4** | 0,43 | 0,42 | 0,02 |
| | | **16** | 0,51 | 0,47 | 0,13 |
| | | **24** | 0,56 | 0,55 | 0,02 |
| | **16** | **2** | 0,34 | 0,34 | 0,02 |
| | | **4** | 0,35 | 0,34 | 0,01 |
| | | **8** | 0,37 | 0,37 | 0,01 |
| | | **24** | 1,37 | 1,34 | 0,49 |
| | **24** | **2** | 1,2 | 1,18 | 0,21 |
| | | **4** | 1,66 | 1,65 | 0,28 |
| | | **8** | 2,2 | 2,23 | 0,15 |
| | | **16** | 3,6 | 3,26 | 0,77 |
| ***Worst-fit*** | **2** | **4** | 0,42 | 0,42 | 0,01 |
| | | **8** | 0,45 | 0,45 | 0,02 |
| | | **16** | 0,5 | 0,5 | 0,02 |
| | | **24** | 0,62 | 0,62 | 0,04 |
| | **4** | **2** | 0,34 | 0,32 | 0,02 |
| | | **8** | 0,36 | 0,36 | 0,02 |
| | | **16** | 0,41 | 0,41 | 0,02 |
| | | **24** | 0,68 | 0,66 | 0,09 |
| | **8** | **2** | 0,33 | 0,32 | 0,02 |
| | | **4** | 0,34 | 0,33 | 0,02 |
| | | **16** | 0,44 | 0,44 | 0,03 |
| | | **24** | 1 | 1,02 | 0,18 |
| | **16** | **2** | 0,31 | 0,31 | 0,02 |
| | | **4** | 0,3 | 0,3 | 0,01 |
| | | **8** | 0,38 | 0,39 | 0,03 |
| | | **24** | 1,98 | 1,9 | 0,4 |
| | **24** | **2** | 1,21 | 1,2 | 0,22 |
| | | **4** | 1,56 | 1,54 | 0,1 |
| | | **8** | 3,73 | 3,7 | 1,09 |
| | | **16** | 5,9 | 6,16 | 1,22 |

## V. Conclusiones

Este trabajo analiza como afecta la incorporación de la maleabilidad a la versión paralela del gradiente conjugado. La incorporación de esta característica en una aplicación paralela consta de tres partes: la comunicación con el gestor de recursos, la creación y redistribución de nuevos procesos, y la redistribución de los datos desde los procesos padres a los hijos. En el análisis de las implementaciones, únicamente se han considerado los dos últimos aspectos. Se han desarrollado dos versiones maleables distintas según la forma en la que se ha llevado a cabo la redistribución de los datos. En una de ellas, la comunicación será

totalmente síncrona mientras que en la otra, parte de los datos se envían de forma asíncrona solapando cómputo y comunicaciones y, otra parte, se enviará síncronamente. Este trabajo también compara el rendimiento de ambas implementaciones.

Los resultados de este análisis muestran que el gradiente conjugado se beneficia del uso de maleabilidad en tiempo de ejecución independientemente de la variante utilizada, mejorando su rendimiento cuando se realiza un redimensionado hacia un mayor número de procesos. Según nuestra experimentación, en general, es preferible usar la versión síncrona para redimensionar a más procesos y la versión asíncrona para el caso contrario.

Realmente se esperaba obtener un mayor beneficio de la versión que redistribuye parte de los datos de forma asíncrona frente a la síncrona para todos los casos. Pero las prestaciones no han sido las esperadas debido a que el solapamiento entre las comunicaciones asíncronas y las propias del bucle de cómputo suponen un sobrecoste de comunicaciones que imposibilita mejorar las prestaciones.

También se puede concluir que el tiempo empleado en el redimensionado se debe mayoritariamente a la creación de procesos más que a la redistribución de los datos. Sin embargo, se puede deducir que el porcentaje de tiempo empleado para realizar una redimensión respecto al tiempo de ejecución total es en general bajo excepto si intervienen un gran número de procesos.

Este trabajo ha supuesto un primer paso en el análisis del impacto de la incorporación de la maleabilidad en una aplicación paralela. A partir de estas conclusiones se pretende abordar este estudio de una forma más detallada evaluando el impacto de su incorporación en otras aplicaciones con diferente carga de comunicación, para comprobar su repercusión en la versión asíncrona cuando se solapa cómputo con la redistribución. De esta forma se podrá determinar cuando resulta más conveniente utilizar la versión síncrona o asíncrona.

## Agradecimientos

El presente trabajo ha sido financiado mediante los siguientes proyectos:

- Técnicas algorítmicas para computación de alto rendimiento consciente del consumo energético y resistente a errores. TIN2017-82972-R
- Elasticidad de aplicaciones científicas MPI. UJI-B2019-36

## Referencias

[1] Jitendra Padhye and Lawrence Dowdy, "Dynamic versus adaptive processor allocation policies for message passing parallel computers: An empirical comparison," in *Job Scheduling Strategies for Parallel Processing*, Dror G. Feitelson and Larry Rudolph, Eds., Berlin, Heidelberg, 1996, pp. 224–243, Springer Berlin Heidelberg.

[2] K El Maghraoui, Travis J Desell, Boleslaw K Szymanski, and Carlos A Varela, "Malleable iterative mpi applications," *Concurrency and computation practice & experience*, vol. 21, no. 3, pp. 393–413, 2009.

[3] Pierre Lemarinier, Khalid Hasanov, Srikumar Venugopal, and Kostas Katrinis, "Architecting malleable mpi applications for priority-driven adaptive scheduling," in *Proceedings of the 23rd European MPI Users' Group Meeting*, New York, NY, USA, 2016, EuroMPI 2016, p. 74–81, Association for Computing Machinery.

[4] A. Moody, G. Bronevetsky, K. Mohror, and B. R. d. Supinski, "Design, modeling, and evaluation of a scalable multi-level checkpointing system," in *SC '10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, 2010, pp. 1–11.

[5] A. C. Sena, F. S. Ribeiro, V. E. F. Rebello, A. P. Nascimento, and C. Boeres, "Autonomic malleability in iterative mpi applications," in *2013 25th International Symposium on Computer Architecture and High Performance Computing*, 2013, pp. 192–199.

[6] Gonzalo Martín, David E. Singh, Maria-Cristina Marinescu, and Jesús Carretero, "Enhancing the performance of malleable mpi applications by using performance-aware dynamic reconfiguration," *Parallel Computing*, vol. 46, pp. 60 – 77, 2015.

[7] Gonzalo Martín, Maria-Cristina Marinescu, David E. Singh, and Jesús Carretero, "Flex-mpi: An mpi extension for supporting dynamic load balancing on heterogeneous non-dedicated systems," in *Euro-Par 2013 Parallel Processing*, Felix Wolf, Bernd Mohr, and Dieter an Mey, Eds., Berlin, Heidelberg, 2013, pp. 138–149, Springer Berlin Heidelberg.

[8] R. Sudarsan and C. J. Ribbens, "Reshape: A framework for dynamic resizing and scheduling of homogeneous applications in a parallel environment," in *2007 International Conference on Parallel Processing (ICPP 2007)*, 2007, pp. 44–44.

[9] R. Sudarsan and C. J. Ribbens, "Scheduling resizable parallel applications," in *2009 IEEE International Symposium on Parallel Distributed Processing*, 2009, pp. 1–10.

[10] Isaías Comprés, Ao Mo-Hellenbrand, Michael Gerndt, and Hans-Joachim Bungartz, "Infrastructure and api extensions for elastic execution of mpi applications," in *Proceedings of the 23rd European MPI Users' Group Meeting*, New York, NY, USA, 2016, EuroMPI 2016, p. 82–97, Association for Computing Machinery.

[11] S. Iserte, R. Mayo, E. S. Quintana-Ortí, V. Beltran, and A. J. Peña, "Efficient scalable computing through flexible applications and adaptive workloads," in *2017 46th International Conference on Parallel Processing Workshops (ICPPW)*, 2017, pp. 180–189.

[12] Sergio Iserte, Rafael Mayo, E. S. Quintana-Ortí, V. Beltran, and A. J. Peña, "Dynamic management of resource allocation for ompss jobs," in *1st PhD Symposium on Sustainable Ultrascale Computing Systems – NESUS PhD 2016*, 2016.

[13] Sergio Iserte, Héctor Martínez, Sergio Barrachina, Maribel Castillo, Rafael Mayo, and Antonio J Peña, "Dynamic reconfiguration of noniterative scientific applications: A case study with hpg aligner," *The International Journal of High Performance Computing Applications*, vol. 33, no. 5, pp. 804–816, 2019.

[14] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*, SIAM, Philadelphia, PA, 1994.

[15] S. Iserte, *High-throughput Computation through Efficient Resource Management*, Ph.D. thesis, Universitat Jaume I, 2018.