Asociación Argentina
de Mecánica Computacional

Mecánica Computacional Vol XXXII, págs. 961-969 (artículo completo)
Carlos G. García Garino, Aníbal E. Mirasso, Mario A. Storti, Miguel E. Tornello (Eds.)
Mendoza, Argentina, 19-22 Noviembre 2013

PRIMEROS PASOS EN EL USO DE UN MODELO PGAS EN EL METODO DE LOS ELEMENTOS DE BORDE

Jorge D'Elía [a], Lisandro Dalcin [a], Sofía Sarraf [a,b], Ezequiel López [a,b], Laura Battaglia [a,c], Gustavo Ríos Rodríguez [a] y Victorio Sonzogni [a]

[a] *Centro de Investigación de Métodos Computacionales (CIMEC)*
Universidad Nacional del Litoral (UNL) - CONICET
Predio Conicet-Santa Fe, Colectora Ruta Nac 168 / Paraje El Pozo, Santa Fe (S3000GLN), Argentina
e-mail: jdelia@intec.unl.edu.ar, dalcinl@intec.unl.edu.ar, gusadrr@yahoo.com.ar,
sonzogni@intec.unl.edu.ar, web page: http://www.cimec.org.ar

[b] *Departamento de Mecánica Aplicada, Facultad de Ingeniería, Universidad Nacional del Comahue,*
CONICET, Buenos Aires 1400, 8300-Neuquén, Argentina
e-mail: (sssarraf, ezequiel.jose.lopez)@gmail.com

[c] *Grupo de Investigación en Métodos Numéricos en Ingeniería (GIMNI)*
Universidad Tecnológica Nacional, Facultad Regional Santa Fe
Lavaysse 610, 3000-Santa Fe, Argentina
e-mail: lbattaglia@santafe-conicet.gob.ar

Palabras Clave: método de elementos de borde, técnica de colocación, ponderación de Galerkin, programación PGAS, computación distribuida, coarreglos Fortran.

Resumen. Los modelos de memoria basados en un espacio global de direcciones particionado (o PGAS, por *Partitioned Global Address Space*), representan una propuesta de compromiso para combinar las ventajas de las memorias compartida y distribuida, en particular, en *clusters* orientados a la computación numérica intensiva de alto rendimiento (o HPC, por *High Performance Computing*). A su vez, estos modelos de memoria dan lugar a los lenguajes de programación PGAS, tales como el UPC (*Unified Parallel C*), *Coarray Fortran* (CAF), y *Titanium* (Java). En este trabajo, se presentan las primeras etapas de una modificación basada en un lenguaje de programación PGAS de un código computacional basado en el BEM (por *Boundary Element Method*).

1. INTRODUCCION

Como es conocido, el método de paneles, o de elementos de borde BEM, se basa en la formulación de un problema de valores de borde de tipo elíptico, como una ecuación integral de borde (BIE, por *boundary integral equation*) (Sauter y Schwab, 2011; Hackbusch, 1995). En sus formulaciones más clásicas, los elementos interactúan todos contra todos, lo cual conduce a matrices de influencia llenas (densas), cuyo cómputo involucra un costo computacional $O(n^2)$, donde n puede ser ya sea el número de elementos E o bien el número de nodos N, de la malla de paneles.

En trabajos anteriores (Sarraf et al., 2013a,b; D'Elía et al., 2009, 2008), referidos al problema de flujo reptante alrededor de un cuerpo rígido tridimensional, se emplearon técnicas por colocación, Galerkin y jerárquico (o en árbol) en donde se infería, como regla aproximada, que los tamaños prácticos que se podían analizar en un equipo con 16 GB de RAM, cuando se emplea doble precisión para los números de punto flotante, están en el orden de:

- O (6 kpaneles): técnica de colocación, matriz del sistema densa, solver directo;
- O (12 kpaneles): ponderación de Galerkin, matriz del sistema densa, solver directo;
- O (36 kpaneles): algoritmo de Barnes-Hut, equivalente a una matriz rala, solver iterativo;

Para otros fines, puede interesar contar con una versión codificada usando un lenguaje de programación que soporte cómputo en paralelo, ya sea en entornos con memoria compartida como distribuida y, en particular, apto para su uso en *clusters* orientados a procesamiento intensivo de alto rendimiento.

Los modelos PGAS fueron considerados en un trabajo previo (D'Elía et al., 2013). En lo que sigue, se expone la etapa básica de contar con un compilador apto en un *cluster*. En los ejemplos numéricos, se muestra el rendimiento obtenido en la factorización LU distribuida con matriz llena de un sistema de ecuaciones, cuando la matriz del sistema es densa, cuadrada y regular, situación usual en las aplicaciones más usuales en BEM.

2. ESPACIO GLOBAL DE DIRECCIONES PARTICIONADO

2.1. Modelos de memoria PGAS

Los modelos de memoria basados en el espacio global de direcciones particionado (Partitioned Global Address Space, 2013), han sido propuestos como una solución de compromiso entre la memoria compartida y la memoria distribuida para combinar las ventajas de ambos modelos de memoria. A su vez, los modelos de memoria PGAS ha dado lugar a los lenguajes de programación PGAS, que son extensiones de algunos lenguajes de programación secuenciales para el soporte de la programación paralela en equipos con memoria tanto compartida como distribuida. Su ubicación en la era actual de los equipos multinúcleo, incluyendo las combinaciones CPU-GPGPU (CPU, por *Central Processing Unit*) y (GPGPU, por *General Purpose Computation on Graphics Processing Units*), es presentada en la revisión general, de hasta mediados de 2012, realizada por Diaz et al. (2012). En general, los modelos de memoria PGAS involucran dos conceptos:

- **Espacio global de direcciones particionado**: asume un espacio compartido de memoria, lo que permite que los procesadores puedan acceder a cualquier dato compartido global;
- **Memoria local** y **memoria remota**: en los datos compartidos introduce una distinción fuerte entre los datos locales y los datos remotos, por lo cual el espacio de direcciones es

particionado en una jerarquía de dos niveles dados por la memoria local y la memoria remota. Cada variable en el espacio global de direcciones es almacenada en la memoria del procesador asignado. Toda vez que se pueda, se aprovecha que las variables compartidas locales son mucho mas rápidas de acceder que las compartidas remotas, con lo cual se aminora una penalidad en el rendimiento.

2.2. Lenguajes de programación PGAS

Entre los lenguajes de programacion que soportan el modelo de memoria PGAS y con compiladores *open-source*, de libre disponibilidad y de fácil alcance en nuestras latitudes, se mencionan:

- Unified Parallel C: una extensión aún no estándar para el lenguaje C;
- Fortran 2008 (ISO/IEC 1539-1:2010): en lo referido a cómputo paralelo, introduce los coarreglos (por *coarrays* en inglés);
- Titanium: una extensión propuesta en Java, la que tampoco es estándar aún y que incluso se encuentra en un estadio algo más experimental (pero no debe confundirse con el disponible para móviles).

Los detalles particulares del modelo PGAS empleado en cada lenguaje de programación extendido difieren en algunas propiedades, e.g. el manejo de los arreglos particionados, punteros a objetos locales o remotos, por eso se prefiere aquí hablar de modelos de memoria PGAS, en plural. Una comparación muy básica entre dos lenguajes de programación PGAS, los basados en el Message Passing Interface (MPI), y los orientados a entornos con memoria compartida, e.g. (OpenMP, por *Open Multi-Processing*), es resumida en la Tabla 1.

	Modelo de memoria	Modelo de programación	Modelo de ejecución	Estructura de datos
MPI	distribuida	MPMD/SPMD	MPMD/SPMD	fragmentadas
OpenMP	compartida	paralela global	multihilos en memoria compartida	arreglos en memoria compartida
UPC	PGAS	SPMD	SPMD	coarreglos 1D punteros distribuidos
CAF	PGAS	SPMD	SPMD	coarreglos nD

Tabla 1: MPI, OpenMP y dos lenguajes de programación PGAS: modelos de memoria, de programación y de ejecución, y algunas estructuras de datos representativas.

donde están MPMD por (*Multiple Program Multiple Data*) y SPMD por (*Single Program Multiple Data*).

2.3. Algunos compiladores PGAS

La disponibilidad de compiladores PGAS prácticamente está restringida a sistemas GNU-Linux. Con respecto a los equipos de cómputo, pueden emplearse servidores, computadoras de escritorio y pórtatiles, con uno o multiples núcleos, y los *clusters* de cómputo científico. Si bien el énfasis de este trabajo es en compiladores PGAS de libre distribución, se menciona

que Intel ha anunciado recientemente una versión beta de su compilador Fortran con coarreglos para su Xeon Phi. El siguiente listado resume algunos de los compiladores PGAS de libre disponibilidad (en casi todos los casos):

- Berkeley UPC: es *open-source*, apto para *clusters*, y sin restricciones en el número de procesadores, para el lenguaje de programación C;
- OpenUH: idem caso anterior pero para lenguaje de programación Fortran;
- GNU Unified Parallel C (GNU UPC);
- GNU Gfortran compiler: a la fecha la parte de coarreglos está atrasada, solo soporta un proceso por razones de compatibilidad;
- G95 fortran compiler: con coarreglos y de libre distribución hasta 4 máquinas, cada una de éstas con cualquier número de núcleos. Empero, el soporte en tiempo de ejecución (o *run-time*) es un paquete cerrado. Además, su desarrollo a la fecha luce algo estancado.

2.4. Runtimes para compiladores PGAS

Para las diversas operaciones de comunicación y de sincronización, los compiladores Berkeley UPC y el OpenUH emplean los siguientes soportes en tiempo de ejecución (o *run-times*):

1. Global-Address space networking (GASNet): es una capa de librería de red de bajo nivel e independiente del lenguaje empleado, que proporciona primitivas para la comunicación unilateral (*one-side*), y se lo utiliza como herramienta para las bibliotecas en tiempo de ejecución. Internamente contiene dos capas, la inferior es una interfaz general implementada directamente en la cima de las diversas arquitecturas de red, mientras que la capa superior proporciona acceso a la memoria remota, así como las operaciones colectivas de alto nivel;
2. Aggregate Remote Memory Copy Interface (ARMCI): es otra librería de red de bajo nivel que también proporciona comunicación unilateral (*one side*) en la memoria remota, copia o asignación, así como operaciones mutuamente exclusivas, manejo de datos distribuidos regulares o irregulares. Exhibe compatibilidad con la Message Passing Interface (MPI) en equipos híbridos con memoria compartida y distribuida, y dispone de operaciones tanto bloqueantes como no-bloqueantes, donde éstas últimas son utilizadas en casos en donde se pueden solapar cómputos y comunicaciones.

2.5. Coarreglos en Fortran 2008

Fortran 2008 incorpora modificaciones con respecto al Fortran 2003. Entre otras, introduce una alternativa para la programación paralela mediante el uso de los coarreglos, en donde la sintaxis tradicional de arreglos en Fortran es extendida mediante una notación de índices al final y entre corchetes, denominados coíndices. Adopta el modelo de programación SPMD, donde múltiples copias del programa, llamadas imágenes para distinguirlas de los procesos e hilos, se ejecutan asíncronamente.

Cada imagen tiene su propio conjunto de datos tanto privados como los globalmente accesibles por las otras imágenes. Una propiedad distintiva es que cada imagen puede acceder a la memoria de otra imagen sin la participación explícita de esta última, como si los datos estuvieran compartidos. Como el acceso a la memoria remota es explícito mediante la notación entre corchetes, las operaciones remotas quedan en evidencia en el programa fuente.

Unicamente los coarreglos son accesibles desde otra imagen, y pueden ser tanto estáticos como dinámicos, pero deben existir en todas las imágenes. Si bien se admiten múltiples co-dimensiones, deben tener la misma forma en cada imágen. Además, dispone de barreras de sincronización ocasionales, globales o parciales.

Una diferencia entre los lenguajes PGAS y los de memoria compartida, es que estos últimos no distinguen los datos compartidos que son locales de aquellos que son remotos. Tampoco los coarreglos son reducibles a un arreglo global porque, en ese caso, se impondría una enorme carga al sistema operativo en garantizar la coherencia de la memoria. En su lugar, la programación debe prever sincronizaciones explícitas, por lo que el compilador es libre, entre las sincronizaciones, de utilizar todas sus técnicas de optimización usuales, como si se tratara de una única imagen. El modelo CAF fue desarrollado como una extensión por fuera del estándar en algunos compiladores, e.g. en la CRAY desde la versión 3.1. Cuando se incorporaron los coarreglos en el estándar Fortran 2008, el término "co-arreglo" (por *co-array*) no sólo perdió el guión sino que también parte de la sintaxis fue modificada. Un compilador con coarreglos puede ser implementado tanto en sistemas con memoria compartida (e.g. multinúcleos), o en sistemas distribuidos (e.g. en *clusters*). *Prima facie*, al incorporar coarreglos en Fortran 2008, un código que los emplee en forma conforme a dicho estándar, debería ejecutarse sin cambios en un ordenador de escritorio, portátil, en un *cluster* o incluso en una Xeon Phi.

3. COMPILADOR OPENUH EN EL CLUSTER COYOTE

3.1. Cluster Coyote

Los *clusters* disponibles en el CIMEC han sido ensamblados empleando *software* de libre distribución disponibles para los sistemas operativos GNU-Linux y, en particular, las distribuciones Fedora. Actualmente se está empleando la distribución Fedora 17. En particular, en los ejemplos se ha empleado el Cluster Coyote, del cual, a la fecha, se ha solicitado su adhesión en el Sistema Nacional de Computación de Alto Desempeño (SNCAD).

3.2. Compilador OpenUH

El compilador OpenUH está libremente disponible para los sistemas operativos GNU-Linux, tanto binarios en algunas distribuciones, como también los fuentes para su compilación, pero aún no cuenta con herramientas de gestión de paquetes tales como `yum`. En el caso de compilar los programas fuentes, el proceso de instalación es el usual, i.e. `make`, `make install`, etc., pero antes necesita como requisito otras capas de *software*, incluyendo algunas del sistema, una o ambas de las ya mencionadas *runtimes*, o capas, de comunicacion GASNet y ARMCI, así como también alguna distribución de la ubicua librería de pasos de mensajes MPI (2013). Para esta última, en particular, se ha optado por la distribución Open MPI. La capa de comunicación (GASNet o ARMCI) es usada para reservar una parte de la memoria total como memoria compartida, y que será empleada por los coarreglos, tanto los estáticos en su cima como por los dinámicos en la siguiente capa, y además otros datos globalmente accesibles (Eachempati et al., 2012).

3.3. Nota sobre el tamaño de la memoria compartida

El tamaño de la memoria compartida es definida por los parámetros SHMMAX y SHMALL, donde SHMMAX define el tamaño máximo del segmento de memoria compartida que un proceso puede asignar en el espacio de direcciones virtuales. Hay que destacar que el *kernel* de Linux tradicionalmente define un valor relativamente pequeño para SHMMAX, e.g. en *kernels*

2.2.x es de 32 MB en Intel, 16 MB en Sun Ultra y de 4 MB en Alpha (e.g. Global Arrays (GA), 2013, Platform-Specific Notes). Por otra parte, el parámetro SHMALL define el número máximo de páginas de memoria compartida que puede utilizar el sistema, por lo que siempre debería valer `ceil(shmmax/PAGE_SIZE)`. Dichos valores se pueden chequear en Fedora usando:

```
ipcs -l;
getconf PAGE_SIZE
cat /proc/sys/kernel/shmall
```

En cualquier caso, los valores por defecto suelen restringir severamente el tamaño máximo del problema que se podrá encarar usando un compilador PGAS, por lo que deben controlarse e incrementarse significativamente, y tiene que hacerse tanto en el servidor como en los nodos de cálculo. Una forma de modificar ambos parámetros es mediante los comandos:

```
pdsh -a sysctl -w kernel.shmmax=8589934592 # (8 GiB);
pdsh -a sysctl -w kernel.shmall=2097152
sysctl -w kernel.shmmax=8589934592
sysctl -w kernel.shmall=2097152
```

3.4. Nota sobre el tamaño de la pila

Otro problema que puede ocurrir se debe a que el tamaño de la pila (*stack size*) suele ser reducido, e.g. 8 GiB, lo cual a la larga origina errores de memoria cuando se utilizan arreglos locales automáticos de tamaño relativamente grande. Esto se puede chequear consultando `ulimit -a`, y se puede intentar sortear el problema reemplazando los arreglos locales por arreglos dinámicos. También se puede experimentar cambiando por `ulimit -s unlimited`, aunque en forma más prudente habría que definir un valor de compromiso.

4. EJEMPLOS NUMERICOS

Se considera la factorización LU distribuida con matriz llena del sistema de ecuaciones $\mathbf{Ax} = \mathbf{b}$, donde $\mathbf{A}$ es la matriz del sistema, $\mathbf{b}$ es el vector columna del término fuente, o término independiente, y $\mathbf{x}$ es el vector columna solución. Este tipo de sistemas considerado está orientado al BEM clásico en donde la matriz del sistema $\mathbf{A}$ en general es cuadrada, llena, regular, y no-simétrica. Como medidas de rendimiento se opta por el *speed-up* S y la eficiencia E (Parhami, 2002), calculados como $S = T(1)/T(p)$, donde $T(1)$ es el tiempo de ejecución con 1 imagen, y $T(p)$ el el tiempo de ejecución con p imágenes, mientras que la eficiencia se puede obtener con $E = S/p$. Los tests fueron realizados en el Cluster Coyote empleando desde 1 hasta 8 nodos, todos homogéneos, del tipo Xeon E5-1660, de 3.30 GHz, empleando en cada nodo uno solo de sus 6 núcleos, mientras que como librería de comunicaciones se emplea la *runtime* ARMCI.

El tamaño máximo del sistema que se podrá resolver quedará acotado por el tamaño de la memoria compartida y por la cantidad de octetos reservados para la representación de un número de punto flotante (igual a 8, para dobles). En el caso del Cluster Coyote, en el presente caso,

el sistema brinda `SHMMAX`=8 GiB, por lo que el número máximo de grados de libertad dado por `nmax=floor(sqrt(SHMMAX/8))` es aproximadamente igual a 30 000. Además, para validar las factorizaciones LU distribuidas, se considera también el caso de $n = 20\ 000$ grados de libertad, el cual se factoriza también en forma secuencial en la imagen 1 para chequear.

La matriz distribuida $\mathbf{A}$ para el caso con $n = 20\ 000$ grados de libertad se generó pseudo-aleatoriamente únicamente en la imagen 1, y luego se transfirió a cada imagen la porción que le correspondía. Mientras que para la matriz correspondiente al caso con $n = 30\ 000$ grados de libertad, se empleó el esquema arbitrario:

$$\mathbf{A} = \begin{bmatrix} 1 & n+1 & \dots & (n-1)n+1 \\ 2 & n+2 & \dots & (n-1)n+2 \\ \dots & \dots & \dots & \dots \\ n & 2n & \dots & n^2 \end{bmatrix} \quad ; \tag{1}$$

donde cada imagen realizó la porción de la matriz que tenía asignada.

La rutina que realiza la factorización LU distribuida emplea coarreglos para los datos compartidos, y la librería LAPACK (Anderson et al., 1999) para las operaciones dentro de cada nodo. La rutina básica fue provista por Mellor-Crummey (comunicación personal), y tuvo que ser modificada para poder ser compilada con OpenUH en el sistema operativo GNU-Linux porque, entre otros motivos, fue inicialmente desarrollada para su uso en la CRAY cuando aún no había sido incluido el modelo de coarreglos en el estándar Fortran 2008, por lo que hacía uso de sintaxis no incluida posteriormente en el estándar. Además, en las primeras pruebas surgieron problemas de insuficiente memoria estática los cuales fueron superados, en primera instancia, mediante el reemplazo de todos los arreglos locales estáticos por los correspondientes dinámicos. La matriz densa $\mathbf{A}$ se distribuye entre las imágenes por columnas, en bloques de tamaño $n \times n_b$, donde n_b es un parámetro elegido a priori. Usando p imágenes, los bloques se mantienen contiguamente en las imágenes (1,2, ..., p, 1,2,..., p, ...) en la forma bloque-columna, con distribución cíclica.

Unicamente la imagen 1 lee desde la línea de comandos de llamada el tamaño n del problema test y asigna el tamaño de bloque n_b, mientras que las demás imágenes esperan hasta que sean notificadas de la disponibilidad de estos valores. A continuación, todas las imágenes asignan memoria dinámica para la porción de la matriz del sistema y para el vector de permutaciones por pivoteo. En cada imagen se utiliza la rutina DGETRF de LAPACK (Anderson et al., 1999), la cual realiza una factorización derecha por bloques. Para n grande en comparación con el tamaño de bloque n_b, la mayor cantidad de trabajo se realiza en la rutina DGEMM, nivel 3 de BLAS, la cual está generalmente optimizada para cada tipo de máquina.

En la Fig. 1 se muestran el *speed-up* S y la eficiencia E (izq. y der., respectivamente), en función del número p de imágenes, de la factorización LU con pivoteo parcial por filas, con matrices de orden $n = 20$ y $n = 30$ k-grados de libertad, utilizando Aggregate Remote Memory Copy Interface (ARMCI) como *runtime*.

5. CONCLUSIONES

En este trabajo se ha considerado el uso de los modelos PGAS. Dichos modelos están orientados a entornos con memoria tanto compartida como distribuida. Una utilidad práctica de dichos modelos en los sistemas operativos GNU-Linux es la libre disponibilidad de los compiladores Berkeley UPC y OpenUH, ambos como extensiones propuestas en los lenguaje de programación C y Fortran, respectivamente, si bien sólo el último se encuentra normalizado a la fecha en su estándar Fortran 2008. Ambos compiladores son de tipo *open-source* y, en particular, aptos

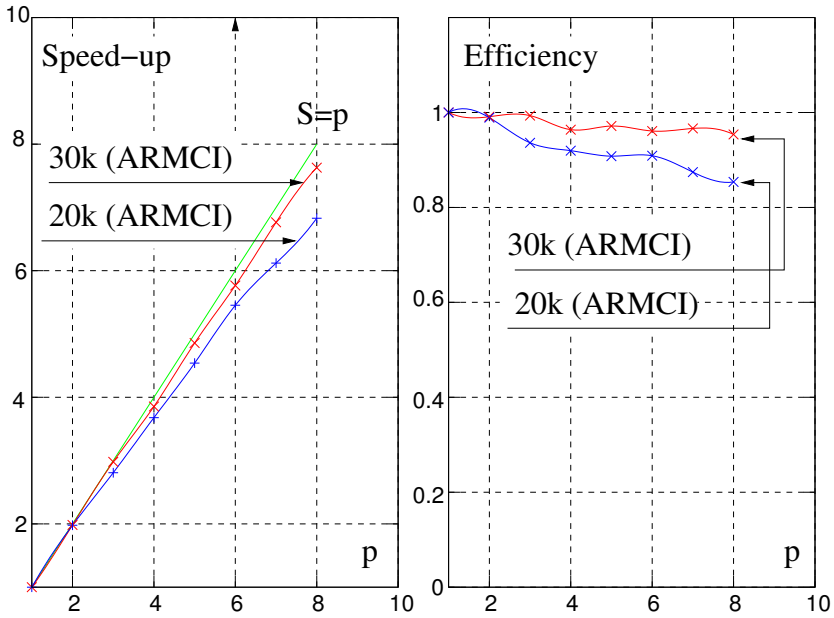

Figura 1: Factorización LU de un sistema lineal con $n = 20$ y $n = 30$ k-grados de libertad, con matriz densa y pivoteo parcial, usando coarreglos, OpenUH, y la *runtime* ARMCI. *speed-up* S (izq.) y eficiencia E (der.), en función del número p de imágenes.

para su uso en *clusters*, sin restricciones en el número de procesadores. El objetivo a largo plazo es incorporar el modelo PGAS como una opción complementaria en un código computacional basado en BEM y desarrollado en el CIMEC, el cual ya ha sido presentado en trabajos previos (e.g. D'Elía *et al.*, "A Galerkin boundary element method for Stokes flow around bodies with sharp corners and edges", Mecánica Computacional, vol. XXVIII, 2009). Dicho código está orientado a simulaciones numéricas del flujo reptante estacionario alrededor de cuerpos tridimensionales, tanto por ponderación de Galerkin como por colocación al centroide de los elementos. Finalmente, la tarea inmediata a futuro será el ensamblado del sistema de ecuaciones obtenido por BEM mediante el presente modelo de programación PGAS.

AGRADECIMIENTOS

Este trabajo ha sido financiado por el Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET, Argentina, proyecto PIP 112 20111 00978), Universidad Nacional del Litoral (UNL, Argentina, proyecto CAI+D 2009–III-4–2), Agencia Nacional de Promoción Científica y Tecnológica (ANCyT, Argentina, proyectos PICT 2492–10), EU-IRSES (PIRSES-GA-2009-246977), y ha sido parcialmente realizado con los recursos del *Free Software Foundation/GNU-Project*, tales como GNU–Linux-OS, GNU–GFortran, GNU–Octave, GNU–Git, GNU–Doxygen, y GNU–GIMP, así como otros recursos de código abierto, tales como Xfig y LaTeX. Se agradece la gentileza de J. Mellor-Crummey en proveer la rutina LU base, así como también la asistencia de Alejandro Dabin y del *team* OpenUH, en particular, D. Eachempati.

REFERENCIAS

Aggregate Remote Memory Copy Interface (ARMCI). http://www.emsl.pnl.gov/docs/parsoft/armci. 2013.

Anderson E., Bai Z., Bischof C., Blackford S., Demmel J., Dongarra J., Du Croz J., Greenbaum A., Hammarling S., McKenney A., y Sorensen D. *LAPACK Users' Guide*. Society for

Industrial and Applied Mathematics, 3 edición, 1999.
Berkeley UPC. http://upc.lbl.gov. 2013.
Cluster Coyote. http://www.cimec.org.ar/coyote. 2013.
D'Elía J., Battaglia L., Storti M., y Cardona A. Galerkin boundary integral equations applied to three dimensional Stokes flows. En A. Cardona, M. Storti, y C. Zuppa, editores, *Mecánica Computacional, vol. XXVII*, páginas 2397–2410. San Luis, 2008.
D'Elía J., Battaglia L., Storti M., y Cardona A. Galerkin boundary integral equations applied to three dimensional Stokes flows. En C. Bauza, P. Lotito, L. Parente, y M. Vénere, editores, *Mecánica Computacional, vol. XXVIII*, páginas 1453–1462. Tandil, 2009.
D'Elía J., Dalcin L., Sarraf S., López E., Battaglia L., Ríos Rodríguez G., y Sonzogni V. Use of the PGAS model for high performance computing in Beowulf Clusters. En C. Garcia Garino y M. Printista, editores, *HPCLaTAm 2013*, páginas 210–215. Mendoza, Argentina, 2013.
Diaz J., Muñoz Caro C., y Niño A. A survey of parallel programming models and tools in the multi and many-core era. *IEEE Trans. Parallel Distr. Syst.*, 23(8):1369–1388, 2012.
Eachempati D., Richardson A., Liao T., Calandra H., y Chapman B. A coarray fortran implementation to support data-intensive application development. *High Performance Computing, Networking Storage and Analysis, SC Companion*, 0:771–776, 2012.
Fedora. http://fedoraproject.org/wiki. 2013.
Fortran 2008. 2013. http://www.j3-fortran.org.
G95 fortran compiler. http://www.g95.org. 2013.
Global-Address space networking. http://gasnet.cs.berkeley.edu. 2013.
Global Arrays (GA). http://www.emsl.pnl.gov/docs/global. 2013.
GNU Gfortran compiler. http://gcc.gnu.org/wiki/GFortran. 2013.
GNU-Linux. http://www.gnu.org/gnu/linux-and-gnu.html. 2013.
GNU Unified Parallel C (GNU UPC). http://www.gccupc.org. 2013.
Hackbusch W. *Integral equations*. Birkhäuser, 1995.
ISO/IEC 1539-1:2010. 2013. http://www.iso.org.
Mellor-Crummey. 2013. http://www.cs.rice.edu/ johnmc/pubs-area.html.
Message Passing Interface (MPI). http://www.mpi-forum.org. 2013.
Open MPI. http://www.open-mpi.org. 2013.
OpenMP. http://www.openmp.org. 2013.
OpenUH. http://www2.cs.uh.edu/_openuh/index.shtml. 2013.
Parhami B. *Introduction to Parallel Processing*. Kluwer Academic Publishers, 2002.
Partitioned Global Address Space. http://www.pgas.org. 2013.
Sarraf S., D'Elía J., Battaglia L., y López E. Método de elementos de borde jerárquico basado en el árbol de Barnes-Hut aplicado a flujo reptante exterior. *Rev. Int. Met. Num. para Cálc. Dis. Ing.*, 2013a. In press.
Sarraf S., López E., Ríos Rodríguez G., y D'Elía J. Validation of a Galerkin technique on a boundary integral equation for creeping flow around a torus. *Comp. Appl. Math.*, 2013b. In press.
Sauter S.A. y Schwab C. *Boundary element methods*. Springer, 2011.
Sistema Nacional de Computación de Alto Desempeño (SNCAD). http://www.supercalculo.mincyt.gob.ar. 2013.
Titanium. http://titanium.cs.berkeley.edu. 2013.
Unified Parallel C. http://upc.gwu.edu. 2013.
Xeon Phi. http://software.intel.com/en-us/articles/intelr-composer-xe-2013-beta-registration-for-intelr-xeon-phitm-coprocessor. 2013.