Asociación Argentina
de Mecánica Computacional

Mecánica Computacional Vol XXXIX, págs. 1411-1417 (artículo completo)
F.E. Dotti, M. Febbo, S.P. Machado, M. Saravia, M.A. Storti (Eds.)
Bahía Blanca, 1-4 Noviembre 2022

USO DE MODELOS PGAS EN LA SOLUCIÓN DIRECTA DE SISTEMAS LINEALES DENSOS

USE OF PGAS MODELS IN THE DIRECT SOLUTION OF DENSE LINEAR SYSTEMS

Sofía S. Sarraf[a], Ezequiel J. López[a], Gustavo A. Ríos Rodríguez[b], Laura Battaglia[b] y Jorge D'Elía[b]

[a]*Instituto de Investigación en Tecnologías y Ciencias de la Ingeniería (IITCI), Universidad Nacional del Comahue (UNComa), CONICET, Buenos Aires 1400, 8300 Neuquén, Argentina, https://iitci.conicet.gov.ar/, (sofia.sarraf, ezequiel.lopez)@fain.uncoma.edu.ar*

[b]*Centro de Investigación en Métodos Computacionales (CIMEC), Universidad Nacional del Litoral (UNL), CONICET, Predio CONICET "Dr. Alberto Cassano", colec. RN 168 s/n – Par. El Pozo, 3000 Santa Fe, Argentina, https://cimec.org.ar/, (gusadrr,lbattaglia,jdelia)@santafe-conicet.gov.ar*

Palabras clave: coarreglod fortran, cómputo de alto rendimiento, comunicación unilateral, computación paralela, espacio de direcciones global particionado (PGAS).

Resumen. El espacio de direcciones global particionado (PGAS, por *Partitioned Global Address Space*) es un modelo de programación paralela desarrollado para computadoras con memoria compartida o distribuida, y es la base para los lenguajes de programación con comunicaciones unilaterales de alto nivel como UPC (*Unified Parallel C*) y *Coarray Fortran* (CAF), entre otros, así como también de bibliotecas especializadas. Todos estos recursos son extensiones para proporcionar una comunicación unilateral de alto nivel. En este trabajo se muestra el empleo de CAF utilizando el compilador gfortran a través de *Open Coarrays*, y se lo aplica a la clásica factorización LU (*Lower-Upper*), sin pivoteo, de un sistema lineal denso regular, un tópico de interés en el método de elementos de borde (o BEM, por *Boundary Element Method*), utilizando memoria primaria compartida en computadores multi-núcleo, comparando rendimientos versus recursos clásicos como HPL (*High-Performance Linpack Benchmark*).

Keywords: coarray fortran (CAF), High Performance Computing (HPC), one-sided communication, parallel computation, Partitioned Global Address Space (PGAS).

Abstract. The Partitioned Global Address Space (PGAS) is a parallel programming model that has been developed for computers with shared or distributed memory, and is the basis for programming languages with unilateral high-level communications such as Unified Parallel C (UPC) and Coarray Fortran (CAF), among others, as well as other specialized libraries. All of these resources are extensions to provide one-sided high-level communication. In this work its use is shown through CAF utilising the gfortran compiler with Open Coarrays, and it is applied to the classic LU (Lower-Upper) factorization, without pivoting, of a regular dense linear system, which is a topic of interest in the Boundary Element Method (BEM). The tests are performed using shared primary memory in multi-core computers and compared with the performance of classic resources such as the High-Performance Linpack Benchmark (HPL).

1. INTRODUCCIÓN

El modelo PGAS (*Partitioned Global Address Space*) utiliza un espacio de direcciones de memoria que es global y particionado siguiendo el esquema clásico de paralelización de un único programa con múltiples datos (o SPMD, por *Single-Program Multiple-Data*). En el modelo PGAS se combina el enfoque SPMD utilizado en las computadoras de memoria distribuida con la semántica de las computadoras de memoria compartida, donde cada proceso (o imagen) tiene su propio espacio de direcciones de memoria, pero además puede acceder a una parte del espacio de direcciones en los otros procesos (o *imágenes* en la terminología fortran). Por otra parte, el fortran estándar, desde fortran 2008 (ISO/IEC 1539-1:2004/Cor 3:2008) en adelante y con última actualización en fortran 2018 (ISO/IEC 1539-1:2018), admite el paralelismo como una propiedad de primeria importancia en el lenguaje, buscando a la vez que los cambios a introducir sean los menores posibles, que se logre un rendimiento aceptable, y con cierta flexibilidad para expresar diversos patrones de comunicación paralela. Para hacerlo se utiliza el modelo PGAS a través del CoArray Fortran (CAF), como un conjunto de extensiones en el lenguaje de programación fortran, diseñado como el cambio más pequeño posible para convertirlo en un lenguaje de programación paralelo, apto tanto en computadoras multi-núcleo como en *clusters* multi-nodo. El objetivo principal del modelo de coarreglos es permitir a usuarios codificar programas paralelos sin la sobrecarga usual de invocar explícitamente funciones detalladas de comunicación como aquellas de la interfase de paso de mensajes (o MPI (*Message Passing Interface*)) cuyo último estándar es MPI 4.0. La extensión de coarreglos es parte del estándar desde fortran 2008 (ISO/IEC 1539-1:2004/Cor 3:2008) en adelante, con última actualización en fortran 2018 (ISO/IEC 1539-1:2018). La sintaxis de coarreglos agrega algunas palabras reservadas y funciones intrínsecas, dejando al usuario libre uso de la sintaxis de arreglos estándar de fortran.

En cuanto a la bibliografía, por ejemplo, un raconto del paralelismo SPMD y CAF es dado en Reid et al. (2020), mientras que una comparación entre CAF y MPI para solucionadores de *Partial Differential Equation* (PDE) en mallas estructuradas es dada en Garain et al. (2015). Asimismo, una evaluación de PGAS para un solucionador de Euler invíscido es dada en Prugger et al. (2016). Por otra parte, un estudio del equilibrio de carga usando CAF en arquitecturas heterogéneas o en multi-núcleos es dado en Cardellini et al. (2017), mientras que una comparación entre MPI y CAF más allá de 100k núcleos en el caso de autómatas celulares es dada en Shterenlikht y Cebamanos (2019). En cuanto a aplicaciones computacionales de CAF resilientes a fallos parciales, e.g. ver Fanfarillo et al. (2019), también Fanfarillo y Del Vento (2018). La presente línea de trabajo se basa en GFortran, con un mayor uso de la capa de transporte MPI, y se lo empleará en la solución de un sistema lineal denso regular mediante la clásica factorización LU (*Lower-Upper*), sin pivoteo, un tópico de interés en el método de elementos de borde (o BEM, por *Boundary Element Method*), utilizando memoria primaria compartida en computadores multi-núcleo, comparando rendimientos versus recursos clásicos tales como *Scalapack* y HPL (*High-Performance Linpack Benchmark*).

2. PGAS Y COARREGLOS

2.1. Algunos compiladores con coarreglos

A la fecha se disponen de compiladores que admitan el uso de coarreglos, tal como los comerciales de Cray o de NAG, así como también el de libre distribución GFortran, además el de Intel de uso libre reciente con previa registración. El compilador de Cray solo se ejecuta en sus computadoras propietarias, mientras que el oneAPI HPC Toolkit de Intel está disponible

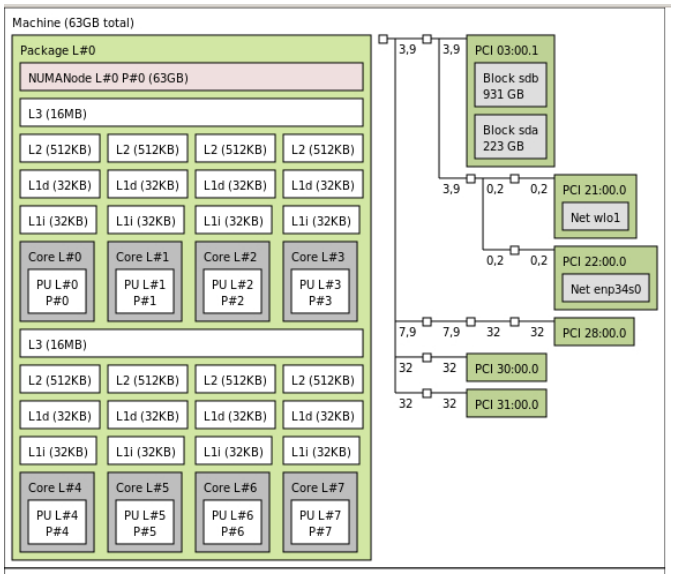

Figura 1: Disposición de los *cores* (o núcleos) y de la memoria primaria en el *host* AMD Ryzen 7 3700X (2.2GHz, 8-core, 32MB) utilizado en los tests, según el utilitario `lstopo` del *Portable Hardware Locality (hwloc)*.

para computadoras de diversas arquitecturas basadas en chips de Intel, y ya sea que utilicen Linux OS, Windows OS, o macOS.

2.2. Coarreglos en GFortran

El compilador GFortran es desarrollado por GNU como parte de la GNU Compiler Collection y admite el modelo de coarreglos desde la versión GCC 5 (e.g. ver News), con soporte completo de coarrays y de operaciones colectivas de transmisión/reducción, además de operaciones atómicas definidas en TS18508, pero con un soporte incompleto en casos especiales.

La biblioteca de comunicación incluida por omisión en GFortran asume una sola imagen, por lo que para poder acceder a más de una imagen (e.g. multi-núcleo), hay que instalar alguna librería de comunicación suplementaria. Eso se puede lograr utilizando OpenCoarrays, el cual ofrece dos alternativas: una basada en MPI, y la restante en GASNet (Global-Address Space Networking). En cuanto a la capa de MPI se ha experimentado con los *stable releases* de tres alternativas de libre distribución:

- Open MPI: versión 4.1.3 (Mar 31, 2022);
- MPICH: versión 4.2.0 (Apr 7, 2022);
- MVAPICH2: versión 2.3.7 (Mar 2, 2022).

Para activar la compilación con OpenCoarrays, se invoca a GFortran adicionando la bandera `-fcorray=[single|lib]` donde, en la primera alternativa, se utiliza una versión secuencial de la librería para una imagen, mientras que en la segunda alternativa se accede a cualquiera de las dos capas de transporte ofrecidas en OpenCoarrays: una biblioteca basada en MPI y la otra en GASNet (*Global-Address space networking*).

En la Fig. 1 se muestra la disposición de los núcleos y la distribución de la memoria primaria en el *host* utilizado en los experimentos, un AMD Ryzen 7 3700X (2.2 GHz, 8-core, 32 MB), según el utilitario `lstopo` perteneciente al *Portable Hardware Locality (hwloc)*. Dicho utilitario proporciona una abstracción portable de la topología jerárquica de la arquitectura utilizada, donde la nomenclatura utilizada por *hwloc* es la siguiente:

- *Package*: una agrupación de uno o más procesadores;
- *NUMANode*: un nodo NUMA (*Non-Uniform Memory Access*) es un conjunto de procesadores alrededor de la memoria primaria a la que los procesadores acceden directamente;
- *Cache*: es la memoria cache. Si existen varios tipos de caches en el sistema, cada una se identifica por su nivel (e.g. L1, L2, L3) y, opcionalmente, por su tipo (e.g. L1i, L1d);
- *Core*: es una única unidad de procesamiento físico que aún puede contener múltiples procesadores lógicos, como subprocesos de hardware.

En la Fig. 1, en particular, se observa que la memoria L3 en el Package L#0 está dividida en dos, que cada una tiene 16 MiB (16384 KiB), y es compartida por 4 *cores* (o núcleos).

3. SOLUCIÓN POR LU DE UN SISTEMA LINEAL DISTRIBUIDO

En la etapa de factorización en el *solver* de un sistema lineal de ecuaciones por factorización LU con CAF, cada núcleo tiene que operar con las tres submatrices ("baldosas" o *tilings*) $\mathbf{T}_{ij}$, $\mathbf{T}_{ik}$, y $\mathbf{A}_{kj}$ listadas en la Ec. (1c), las cuales son submatrices cuadradas auxiliares que tienen que ser alojadas simultáneamente en la memoria:

$$\mathbf{A}_{kk} := \mathbf{A}_{kk}^{-1} \tag{1a}$$
$$\mathbf{T}_{ik} := \mathbf{A}_{ik}\mathbf{A}_{kk} \tag{1b}$$
$$\mathbf{T}_{ij} := \mathbf{T}_{ik}\mathbf{A}_{kj} \tag{1c}$$
$$\mathbf{A}_{ij} := \mathbf{A}_{ij} - \mathbf{T}_{ij} \tag{1d}$$

para $k = 1, 2...., z_n$, e $i, j = (k+1), (k+2)...., z_n$, donde z_n es la extensión de las baldosas (i.e. el número de baldosas a lo largo de una dimensión), dado por $z_n = \lceil n/N_B \rceil$ donde n es el número de incógnitas del sistema lineal, N_B es la extensión (común) de cada baldosa, mientras que $(...)^{-1}$ en la Ec. (1a) denota la matriz inversa. Se elige un tamaño de la baldosa $N_B \times N_B$ tal que cada *core* pueda alojar a las tres submatrices $\mathbf{T}_{ik}$, $\mathbf{A}_{ik}$, y $\mathbf{A}_{kk}$ de la Ec. (1b) simultáneamente, por lo que teniendo en cuenta los datos de la Fig. 1, se exploran 3 posibilidades:

- Cada `L1d` tiene 32 KiB en cada *core*, entonces:

$$L_1 = 32 \text{ KiB } \left(2^{10}\text{B/GiB}\right) = 32768 \text{ B} \tag{2a}$$
$$M_1 = \lfloor L_1/(3 \text{ submatrices}) \rfloor = 10992 \text{ B} \tag{2b}$$
$$N_{B_1} = \left\lfloor \sqrt{M_1/(8 \text{ B double precision})} \right\rfloor = 36 \tag{2c}$$

  por lo que las Ecs. (2a-2c) dan un tamaño de baldosa de $N_{B_1} = 36$;
- Cada `L2` tiene 512 KiB en cada *core*, entonces:

$$L_2 = 512 \text{ KiB } \left(2^{10}\text{B/GiB}\right) = 524288 \text{ B} \tag{3a}$$
$$M_2 = \lfloor L_2/(3 \text{ submatrices}) \rfloor = 174762 \text{ B} \tag{3b}$$
$$N_{B_2} = \left\lfloor \sqrt{M_2/(8 \text{ B en double precision})} \right\rfloor = 147 \tag{3c}$$

  por lo que las Ecs. (3a-3c) dan un tamaño de baldosa de $N_{B_2} = 147$;
- Cada `L3` tiene 16384 KiB y además es compartida por 4 *cores*, entonces:

$$L_3 = 16384 \text{ GiB } \left(2^{10}\text{B/GiB}\right) = 16777216 \text{ B} \tag{4a}$$
$$M_3 = \lfloor L3/(4 \text{ núcleos})/(3 \text{ submatrices}) \rfloor = 1398101 \text{ B} \tag{4b}$$
$$N_{B3} = \left\lfloor \sqrt{M_3/(8 \text{ B en double precision})} \right\rfloor = 418 \tag{4c}$$

  por lo que las Ecs. (4a-4b) dan un tamaño de baldosa de $N_{B_3} = 418$.

3.1. Distribución de la matriz del sistema

En el sistema de ecuaciones lineales $\mathbf{AX} = \mathbf{B}$, donde $\mathbf{A} \in \mathbb{R}^{n\times n}$ es la matriz del sistema, $\mathbf{X} \in \mathbb{R}^{n\times m}$ es la solución, y $\mathbf{B} \in \mathbb{R}^{n\times m}$ es el término independiente, hay que definir la distribución de la matriz del sistema $\mathbf{A}$ dentro de una matriz de procesadores $P_r \times P_c$, donde P_r y P_c son los números de procesadores (*cores* o imágenes) asignados para las filas y las columnas de la matriz del sistema $\mathbf{A}$, respectivamente, de acuerdo al número total de procesadores P disponibles, donde $P = P_r P_c$. Después de algunos experimentos se corroboraron que las recomendaciones dadas en ScaLapack (Sec. *Obtaining High Performance with ScaLAPACK Codes*) para lograr un escalamiento monótono creciente (o casi) cuando se incrementa el número de *cores* (procesadores o imágenes) P, también son aplicables a CAF y a HPL, y que son reproducidas en las Ecs. (5a-5b).

$$P_r = 1 \quad \wedge \quad P_c = P \qquad \text{si } P < 9 \tag{5a}$$

$$P_r = P_c = \left\lfloor \sqrt{P} \right\rfloor \qquad \text{si } P \geq 9 \tag{5b}$$

4. EJEMPLOS NUMÉRICOS

La matriz (densa) del sistema $\mathbf{A}$ es generada en forma pseudo-aleatoria pero dominante en la diagonal principal, para asegurar que sea regular, y se consideran los tamaños n iguales a 10k, 20k, 30k, 40k, y 50k. En la Fig. 2 se muestra las gráficas del $\langle \text{Speedup} \rangle_{n_z}$ (izq.), y de los $\langle \text{GFLOPS} \rangle_{n_z}$ en doble precisión (der.), trazadas en función del número p de imágenes (*cores*), donde cada punto es el promedio aritmético de $n_z = 10$ experimentos, mientras que el *speedup* se lo calcula con $\langle \text{Speedup} \rangle_{n_z} = \langle T_1 \rangle_{n_z} / \langle T_p \rangle_{n_z}$, donde $\langle T_1 \rangle_{n_z}$ es el *wall time* promedio con un proceso, mientras que $\langle T_p \rangle_{n_z}$ es el *wall time* promedio con p procesos. Además, CAF denota el *solver* por factorización LU mediante CAF, mientras que con HPL al *solver* de referencia *High-Performance Linpack Benchmark*. El valor de $N_b = 256$ para HPL indicados en la Fig. 2 fue obtenido como el mejor luego de una serie de pruebas preliminares, además Kinghom en 2018 obtiene un valor cercano para el *benchmark* de HPL en otra AMD Ryzen Threadripper (2990WX 32-core), mientras que los valores N_B de 36, 147, y 418 son las estimaciones según las Ecs. (2a-4c). De un análisis de la Fig. 2: (i) se observa que las curvas de *Speedup* con CAF quedan por debajo de HPL en todas las subfiguras; (ii) pero a la vez las curvas de los GFLOPS con CAF quedan por encima de HPL en (casi) todas las subfiguras; (iii) a los efectos prácticos es suficiente considerar sólo el tamaño de la memoria cache `L3`.

5. CONCLUSIONES

La codificación con CAF del *solver* de un sistema lineal denso por factorización LU involucró un salto cualitativo aunque con pocos cambios con respecto a la codificación secuencial tradicional; resultando, no obstante, similar al equivalente con MPI con una distribución `(*,cyclic)` de la matriz del sistema $\mathbf{A}$, e.g. ver la presentación asociada en el texto clásico de Aoyama y Nakano (1999). De los experimentos numéricos, se observa que en los sistemas más chicos hay bastante diferencia en el *speed-up* cuando el número de imágenes aumenta, mientras que desde otro punto de vista de los GFLOPS se puede concluir que, a los efectos prácticos, es suficiente considerar el tamaño de la memoria cache `L3` para decidir el tamaño de las submatrices (baldosas). Además, aunque no se muestra, en los tests no se observaron diferencias apreciables en las curvas de rendimiento entre las distribuciones Open MPI, MPICH, o MVAPICH2. Este *solver* LU alternativo se está incorporando como una opción adicional en el código GBEM, basado en el *Boundary Element Method* (BEM) y está orientado principalmente

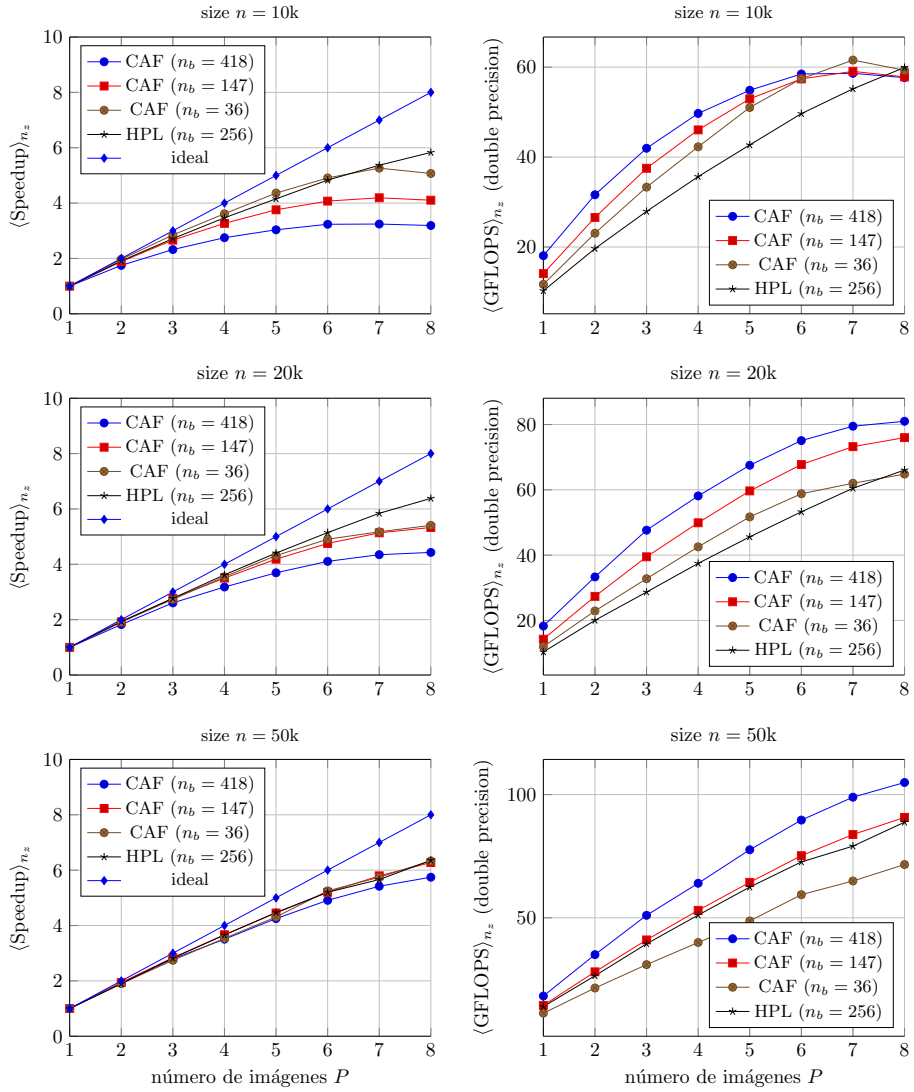

Figura 2: Gráficas del $\langle \text{Speedup} \rangle_{n_z}$ (izq.); y $\langle \text{GFLOPS} \rangle_{n_z}$ (der.), en doble precisión, para una matriz del sistema densa de tamaño n creciente, indicado en cada fila, en función del número p de imágenes, donde cada punto es un promedio de $n_z = 10$ corridas en cada uno.

a flujos de Stokes usando, o bien colocación por puntos, o bien ponderación de Galerkin, y que fue presentado en trabajos previos (Sarraf et al., 2018b, 2017, 2018a). Finalmente un demo muy simple, pero a la vez de muy bajo rendimiento, apto bajo Linux con GFortran y OpenCoarrays está disponible en el sitio caf-gausscycli-toy.f90.

AGRADECIMIENTOS

Este trabajo ha sido financiado por el Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET, proyecto PIP 112-201501-00588CO), la Agencia Nacional de Promoción Científica y Tecnológica (ANPCyT, proyectos PICT-2018-1607, PICT-2018-2920), la Universidad Nacional del Litoral (UNL, proyectos CAI+D 2020-506-201901-00110-LI, CAI+D 2020-506-201901-00140-LI)), la Universidad Tecnológica Nacional (UTN, proyecto PID-UTN-8132), 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 LaTeX, PGF, TikZ, y PGFPlots.

REFERENCIAS

Aoyama Y. y Nakano J. *RS/6000 SP: Practical MPI Programming*. International Technical Support Organization, IBM, 1999.

Cardellini V., Fanfarillo A., y Filippone S. Coarray-based load balancing on heterogeneous and many-core architectures. *Parallel Comput.*, 68:45–58, 2017.

Fanfarillo A. y Del Vento D. Notified access in coarray-based hydrodynamics applications on many-core architectures: Design and performance. *Parallel Computing*, 75:118–129, 2018. doi:https://doi.org/10.1016/j.parco.2018.04.002.

Fanfarillo A., Garain S.K., Balsara D., y Nagle D. Resilient computational applications using coarray fortran. *Parallel Comput.*, 81:58–67, 2019.

Garain S., Balsara D.S., y Reid J. Comparing coarray fortran (CAF) with MPI for several structured mesh PDE applications. *J. Comp. Physics*, 297:237–253, 2015.

Prugger M., Einkemmer L., y Ostermann A. Evaluation of the partitioned global address space (PGAS) model for an inviscid Euler solver. *Parallel Comput.*, 60:22–40, 2016.

Reid J., Long B., y Steidel J. History of coarrays and SPMD parallelism in Fortran. *Proceedings of the ACM on Programming Languages (PACMPL)*, 4(HOPL):72:1–72:30, 2020. doi:https://doi.org/10.1145/3386322.

Sarraf S., López E., Battaglia L., Ríos Rodríguez G., y D'Elía J. Validación numérica de un método de elementos de borde para flujo de Stokes oscilatorio alrededor de cuerpos rígidos. En *Mecánica Computacional, vol. XXXV*. La Plata, 2017.

Sarraf S., López E., Battaglia L., Ríos Rodríguez G., y D'Elía J. An improved assembling algorithm in boundary elements with Galerkin weighting applied to three-dimensional Stokes flows. *J Fluids Eng- Trans ASME*, 140(1):011401, 2018a. doi:10.1115/1.4037690.

Sarraf S., López E., Battaglia L., Ríos Rodríguez G., y D'Elía J. Simulación numérica del flujo a bajo número de Reynolds en un microresonador del tipo placa. Parte 1: OpenFOAM y GBEM. En *Mecánica Computacional, vol. XXXVI*. Tucumán, 2018b.

Shterenlikht A. y Cebamanos L. MPI vs fortran coarrays beyond 100k cores: 3D cellular automata. *Parallel Comput.*, 84:37–49, 2019.