Premio Turing 2021:

Jack J. Dongarra, el héroe de la ciencia e ingeniería computacional

Ganador del Premio Turing 2021

Foto: Tara Kneiser

Jack. J. Dongarra

CRISTÓBAL NAVARRO

Doctor en Ciencias de la Computación por la Universidad de Chile. Actualmente es Profesor Asociado del Instituto de Informática de la Universidad Austral de Chile. Sus líneas de investigación son la computación de alto rendimiento, en especial GPU Computing. Actualmente investiga en integrar el uso de núcleos regulares, de tensor, y de ray-tracing para acelerar aplicaciones que están fuera del marco de inteligencia artificial y computación gráfica. Lidera el grupo de investigación Temporal (http://temporal.uach.cl) y es el coordinador del supercomputador Patagón (https://patagon.uach.cl).

cristobal.navarro@uach.cl

NANCY HITSCHFELD KAHLER

Profesora Titular del Departamento de Ciencias de la Computación de la Universidad de Chile. Doctora en Technischen Wissenschaften por la ETH-Zurich, Suiza. Líneas de investigación: mallas de polígonos y poliedros, algoritmos paralelos (computación en GPU), algoritmos en ciencia e ingeniería computacional y educación en computación. Participa activamente en comisiones y actividades para atraer mujeres a STEM. Miembro del React Lab (Rethinking Education by Advancing Computational Thinking) y del CEMCEN (CEnter for Modern Computational ENgineering). En este último, lidera el +Lab, Meshing for Applied Science.

nancy@dcc.uchile.cl

RESUMEN. Para apreciar de manera más clara las contribuciones de Dongarra, en este artículo describimos, primero, brevemente las dos estrategias más usadas para abordar la solución de un problema de manera paralela y los desafíos existentes en la programación de computadores de alto rendimiento. A continuación, presentamos a Dongarra y detallamos sus principales contribuciones, tanto en la creación de bibliotecas de software de álgebra lineal numérica, de acceso abierto, para el desarrollo de aplicaciones en ciencia e ingeniería, como en las estrategias computacionales creadas para optimizar y hacer el uso más eficiente posible del hardware disponible, con impacto no solo en el ámbito que motivó su creación. Finalmente, concluimos resumiendo sus aportes y el impacto que han tenido.

Paralelismo: un pilar para la computación de alto rendimiento

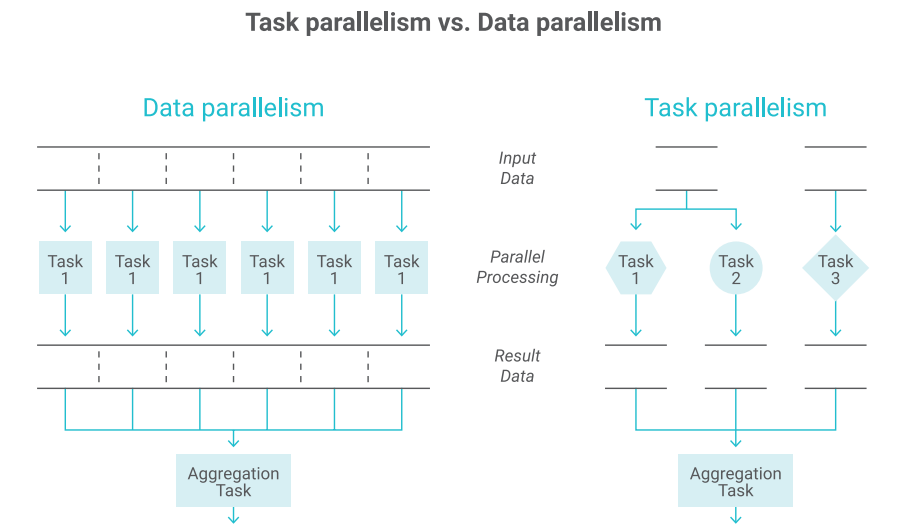
Desde las últimas décadas, el rendimiento computacional crece principalmente por el aumento en la cantidad de núcleos de procesamiento que se alojan en un chip [1], y no tanto por la velocidad (GHz) de cada núcleo (los límites térmicos del silicio impiden progresar significativamente en esta dirección). Este aumento de núcleos incrementa el paralelismo de un procesador. La programación de un procesador paralelo en general no es automática, es decir, el programador debe pensar en cómo dividir el trabajo, comunicar las partes, asignar recursos a cada subproblema, balancear trabajo, hacer uso de la jerarquía de memoria (registros, *caché*, RAM, almacenamiento) eficientemente, entre otras cosas, para poder lograr que el software pueda escalar su rendimiento ante la presencia de hardware paralelo. En términos generales, existen dos tipos de paralelismo; el paralelismo de datos (*data parallelism*) y el paralelismo de tareas (*task parallelism*) (ver Figura 1) [2, 3]. El primero, es el paralelismo que emerge cuando se dividen los datos para que puedan ser trabajados simultáneamente por alguna función replicada en los núcleos del procesador, así esta función actúa simultáneamente en las distintas regiones de datos. Ejemplos de problemas *data-parallel* son las operaciones de álgebra lineal, como operaciones con vectores (suma, resta, producto) o las operaciones con matrices (producto, suma, resta, valores propios, etc.). Por otro lado, el paralelismo de tareas consiste en dividir la funcionalidad en vez de los datos, es decir una función se subdivide en distintas funciones menores que puedan ser ejecutadas de forma simultánea por los núcleos del procesador, sobre un conjunto común de datos. Ejemplos de problemas *task-parallel*

son los sistemas de monitoreo redundantes en aviones o naves espaciales, un algoritmo para romper una clave de seguridad o una sucesión de funciones sobre datos que fluyen (más conocido como *pipeline*). En la práctica, un problema puede exhibir ambos tipos de paralelismo mencionados, sin embargo en la ciencia computacional podemos encontrar que predominan los problemas *data-parallel*, en gran parte debido a que la mayoría son planteados con álgebra lineal, es decir en base a operaciones de vectores y matrices.

La programación de las soluciones paralelas a estos problemas es lo que tradicionalmente se conoce como hacer computación de alto rendimiento o High Performance Computing (HPC), la cual es un área que busca desarrollar algoritmos que hagan un uso eficiente de los recursos de un computador. La evolución de los procesadores centrales (CPU) hacia la flexibilidad los han hecho los candidatos a solucionar todo tipo de problemas (cualquier combinación de *data* o *task parallel*), mientras que la evolución de los procesadores gráficos (GPU) al paralelismo masivo y recientemente a la inclusión de núcleos de *tensor* y *ray-tracing*, los ha convertido en ideales para acelerar aplicaciones *data-parallel*, como álgebra lineal, simulaciones de partículas, inteligencia artificial y computación gráfica, entre otras.

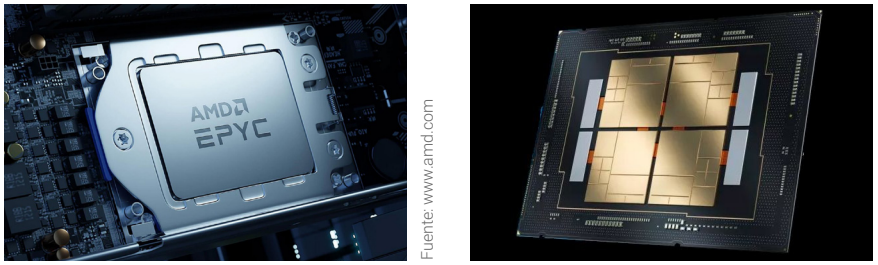
El desafío de programar CPUs y GPUs en paralelo (ver Figuras 2 y 3) se convierte en una tarea aún más compleja cuando se trata de supercomputadores, donde hay decenas de computadores conectados y esperando trabajar como un gran sistema. Es aquí donde las contribuciones de algoritmos y software de Dongarra han sido una herramienta clave para que desde distintas disciplinas de la ciencia, sea posible hacer ciencia computacional a gran escala empleando HPC y contestar preguntas que puedan expandir el horizonte del conocimiento científico.

Su mayor contribución ha sido en la creación de código abierto, bibliotecas de software y estándares para la resolución de sistemas de ecuaciones lineales.



Fuente: http://master.cmc.msu.ru/files/03_Parallel_Programming_Patterns.pdf.

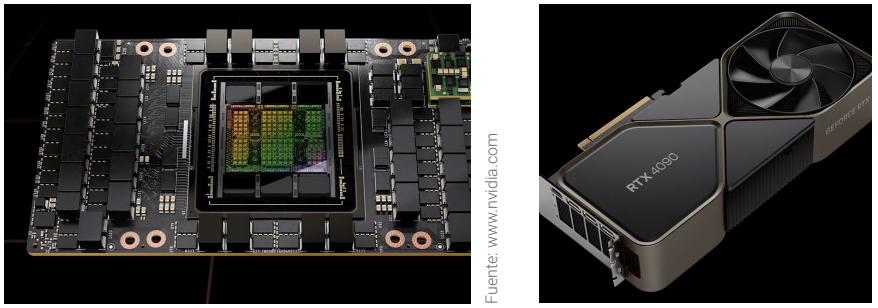
Figura 1. Paralelismo de datos versus paralelismo de tareas.



Fuente: www.amd.com

Fuente: www.tomshardware.com

Figura 2. CPUs servidor: AMD EPYC 7773 (izquierda) e Intel Sapphire Rapids (derecha).



Fuente: www.nvidia.com

Fuente: www.gsmarena.com

Figura 3. GPUs: NVIDIA H100 (izquierda) y GPU consumidor RTX 4090 (derecha).

Jack Dongarra y sus contribuciones en ciencias de la computación y en computación para la ciencia e ingeniería

Dongarra (de nacionalidad estadounidense) obtuvo una Licenciatura en Matemáticas de la Chicago State University (1972), un Máster en Informática del Instituto de Tecnología de Illinois (1973), y un Doctorado en Applied Mathematics de la University of New Mexico (1980). Su tesis de doctorado fue sobre cómo mejorar la precisión del cálculo computacional de los valores propios de una matriz. Desde esa fecha hasta el año 1989 trabajó en el Argonne National Laboratory, adquiriendo el cargo de científico senior. Tiene actualmente un nombramiento como profesor distinguido de ciencias de la computación tanto en los Departamentos de Ingeniería Eléctrica como Ciencias de la Computación de la University of Tennessee y mantiene el nombramiento de investigador distinguido en la División de Matemáticas y ciencias de la computación del Laboratorio Nacional de Oak Ridge desde 1989. También se desempeña como Turing Fellow en la Manchester University (Reino Unido) desde 2007 y es profesor adjunto del Departamento de Ciencias de la Computación de la Rice University. Es también director del Center for Information Technology Research de la University of Tennessee, el cual coordina y facilita los esfuerzos de investigación en TI dentro de la Universidad.

Dongarra se especializó en algoritmos numéricos para resolver problemas usando álgebra lineal, computación paralela, metodologías de programación y herramientas para programación paralela entre otras. Su mayor contribución ha sido en la creación de código abierto, bibliotecas de software (más conocidas como librerías por ser similar a *library*) y estándares para la resolución de sistemas de ecuaciones lineales. Es increíble observar que de las más de 100.000 citas que tiene en Google Scholar, más de 40.000 citas son de las guías de usuarios de las librerías de software desarrolladas, y entre los diez documentos más citados, nueve son guías al usuario [4].

Librerías para álgebra lineal numérica

Dongarra se ha dedicado al desarrollo de estas librerías por más de cuarenta años, asumiendo un rol de principal implementador o investigador principal de las bibliotecas conocidas como LINPACK, BLAS, LAPACK, ScaLAPACK, PLASMA, MAGMA, y SLATE (ver Figura 4). Todas estas bibliotecas son para resolver problemas usando álgebra lineal numérica, y las distintas versiones muestran su evolución en el tiempo al ser reimplementaciones para optimizar el uso del potencial de la última generación de computadores de alto rendimiento. Para lograr entender la evolución en el tiempo, creemos que es interesante describir el énfasis que tiene cada una de ellas.

LAPACK (Linear Algebra Package) [5] provee rutinas para resolver sistemas de ecuaciones lineales, mínimos cuadrados lineales, problemas relacionados a valores propios y descomposición en valores singulares (ver Figura 5). Las rutinas permiten trabajar con matrices de números reales e imaginarios, usando precisión simple (*floating point* 32-*bits*) y doble (*floating point* 64-*bits*). Además, provee varias rutinas para implementar factorización de matrices. Originalmente fue implementada en Fortran 77 pero el año 2008 fue reescrita a Fortran 90. LAPACK está implementado sobre la librería BLAS (Basic Linear Algebra Subprograms) [6] para proveer portabilidad de las rutinas mencionadas anteriormente. Es interesante mencionar que LAPACK no fue la primera librería estándar desarrollada bajo el liderazgo de Dongarra. LAPACK fue diseñada como sucesora de LINPACK, que proveía rutinas para el cálculo de mínimos cuadrados y la resolución de ecuaciones lineales,

Figura 4. Logos de las bibliotecas MAGMA y SLATE.

```
#define GLAS_COMPLEX
#include <glas/toolbox/bindings/dense_matrix_bindings.hpp>
#include <glas/toolbox/bindings/dense_vector_bindings.hpp>
#include <glas/container/dense_matrix.hpp>
#include <glas/container/dense_vector.hpp>
#include <boost/numeric/bindings/lapack/gees.hpp>

...

int main () {
   int n=100;

   / / Define a real n x n matrix
   glas::dens__matrix< double > matrix( n, n ) ;

   / / Define a complex n vector
   glas::dense_vector< std::complex<double> > eigval( n ) ;

   / / Fill the matrix
   ...

   / / Call LAPACK routine DGEES for computing the eigenvalue
   Schur form.
   / / We create workspace for best performance.
   bindings::lapack::gees( matrix, eigvale, bindings::lapack::optimal_
   workspace() ) ;

...
}
```

Fuente: www.semanticscholar.org

Figura 5. Ejemplo de código C++ para calcular los valores propios de una matriz, usando la librería LAPACK.

y de EISPACK que proveía rutinas relacionadas a valores propios. LINPACK y EISPACK fueron escritas en Fortran 77 entre los años 1970 y 1980, ambas diseñadas para correr en los computadores vectoriales modernos de aquella época. En comparación a sus antecesoras, LAPACK fue diseñada para explotar la memoria caché y el paralelismo a nivel de instrucciones de máquina, de los modernos procesadores superescalares de aquella época, y así lograr rutinas que corran varios órdenes de magnitud más rápido que las de LINPACK en esos computadores, usando una librería BLAS bien configurada. Cabe mencionar que LAPACK ha sido extendida para correr en ambientes distribuidos en bibliotecas conocidas como ScaLAPACK y PLAPACK.

Y como veremos a continuación, Dongarra no se pudo quedar atrás cuando aparecieron los procesadores multinúcleos y las GPUs (Graphics Processing Units) con su gran poder de cálculo. Es así cómo se desarrolla PLASMA [7] y MAGMA [8]. PLASMA es para abordar las deficiencias de rendimiento de las librerías LAPACK y ScaLAPACK sobre arquitecturas *multi-core*, pero no implementa todas las rutinas existentes en las librerías anteriores. PLASMA provee una nueva implementación de las rutinas en donde se logra mayor eficiencia al usar procesadores multinúcleos, pero no reemplaza completamente a LAPACK ni a ScaLAPACK. Además incorpora nuevos algoritmos y versiones mejoradas de otros. Entre sus funcionalidades están rutinas que permiten resolver sistemas de ecuaciones lineales densos, mínimos cuadrados lineales y distintos métodos de factorización de matrices, entre otras. Por otra parte, MAGMA se desarrolló con la idea de abordar los complejos desafíos de los ambientes híbridos emergentes, en donde las soluciones combinan las fortalezas de los algoritmos en distintas arquitecturas. Es así como los algoritmos para álgebra lineal fueron diseñados e implementados para correr sobre procesadores multinúcleos y GPUs explotando lo mejor de cada arquitectura.

Actualmente, en pleno proceso de desarrollo se encuentra SLATE (Software for Linear Algebra Targeting Exascale) [9], cuyo objetivo es reemplazar a todas las anteriores. La idea es que los algoritmos implementados usen todo el potencial de desempeño disponible y provean la escalabilidad máxima al correr sobre los actuales *clusters* de computadores, en donde cada nodo puede tener un gran número de *cores* y GPUs. En cuanto a eficiencia se desea lograr el peak teórico de desempeño y escalar al tamaño completo del computador, es decir, correr los algoritmos desde miles a decenas de miles de nodos. Esto debe ser provisto de una manera portable usando estándares como MPI (Message Passing Interface).

Innovaciones técnicas computacionales

La clave de la propagación casi universal del uso de las librerías de Dongarra es, por un lado, la funcionalidad que proveen, útiles por ejemplo, para resolver problemas en ciencia e ingeniería modelados con ecuaciones diferenciales parciales cuya solución numérica requiere resolver sistemas de ecuaciones lineales y, por otro lado, proveyendo versiones de sus bibliotecas que van a la par del desarrollo del hardware de alto rendimiento. Es así como estas librerías fueron evolucionando, han estado y están disponibles para procesadores individuales, computadoras paralelas, multinúcleo y varias GPU por nodo. Las bibliotecas de Dongarra no solo son importantes por su utilidad para los avances en nuevo conocimiento científico y tecnológico, sino también porque durante su desarrollo motivó innovaciones computacionales importantes en temáticas tales como, *autotuning*, aritmética de precisión mixta y cálculos *batch*, aportes que vamos a mencionar a continuación:

Autotuning: Dongarra fue pionero en desarrollar algoritmos para la generación automática y optimización de software numérico en computadores que van desde los computadores personales hasta computadores de alto desempeño [10]. Si bien BLAS le permitía generar librerías portables y eficientes para computadoras de alto rendimiento secuenciales, vectoriales y de memoria compartida, debían optimizarlas manualmente, siendo un proceso costoso y tedioso de lograr para cada arquitectura en particular. El programador debía comprender la arquitectura, la forma de usar la jerarquía de memoria para almacenar y acceder a los datos de manera óptima, manipular las unidades funcionales y los registros para generar instrucciones apropiadas en el momento correcto y la mejor manera de usar la optimización del compilador [10]. La tecnología presentada en

Estas librerías fueron evolucionando [...] están disponibles para procesadores individuales, computadoras paralelas, multinúcleo y varias GPU por nodo.

este primer artículo fue más tarde aplicada a la configuración automática de software en otras áreas. De cierta forma, si no fuese por las contribuciones de Dongarra, muchos supercomputadores habrían sido inutilizados o usados de forma muy ineficiente.

Aritmética de precisión mixta: Dongarra fue pionero en aprovechar múltiples precisiones de aritmética de punto flotante para ofrecer soluciones precisas más rápidamente [11]. La idea consiste en usar una precisión menor (menos bits) siempre donde sea posible (que no afecte el resultado final), debido a que las operaciones matemáticas que usan la mitad de bits que lo original (por ejemplo, 16 en vez de 32 bits) son al menos dos veces más rápidas. Este trabajo se ha vuelto fundamental y muy útil en los avances y desarrollo de aplicaciones de *machine learning*, como se mostró recientemente en el benchmark HPL-AI [12]. HPL-AI (High Performance LINPACK for Accelerator Introspection) benchmark permite el uso de aritmética de precisión mixta para resolver sistemas de ecuaciones lineales. A la vez, esta biblioteca resalta la reciente convergencia entre inteligencia artificial y HPC.

Subdivisión del trabajo: Dongarra fue pionero en el paradigma de dividir la forma de operar sobre matrices densas grandes, que se usan comúnmente en simulaciones, modelación y análisis de datos, en muchos cálculos de tareas pequeñas que se pueden calcular de forma independiente y simultánea. Para mostrar que esto era posible,

Dongarra dirigió y diseñó la reingeniería de la librería BLAS para que sus rutinas, en particular la multiplicación de matrices pueda ser hecha por muchos *threads* que corren en paralelo multiplicando pedazos pequeños de la matriz usando las GPUs [13].

Diseño de interfaces: Otras contribuciones incluyen MPI, el estándar de facto para el paso de mensajes portátiles en arquitecturas informáticas paralelas y distribuidas, y la API de rendimiento (PAPI), que proporciona una interfaz que permite recopilar y sintetizar el rendimiento de los componentes de un sistema heterogéneo. Los estándares que ayudó a crear, como MPI, LINPACK Benchmark y la lista Top500 (www.top500.org) de supercomputadoras, la cual es una referencia mundial, han permitido alcanzar logros computacionales desde la predicción del clima en tiempo real hasta el cambio climático, el desarrollo de curas para múltiples virus, análisis de datos de experimentos físicos y simulaciones a escalas interestelares.

Epílogo

Las contribuciones de Dongarra, base fundamental de las bibliotecas numéricas más utilizadas del mundo, han aportado profundamente en cada área de la computación científica, estableciendo un marco a partir del cual l@s científic@s e ingenier@s han podido hacer importantes descubrimientos e innovaciones en áreas que incluyen las energías renovables, predicción meteorológica y cambio climático, genómica, el descubrimiento de fármacos, la ingeniería aeroespacial, economía, diseño y fabricación de nuevos semiconductores, y análisis de grandes volúmenes de datos, por nombrar algunas. Con el desarrollo de sus librerías de álgebra lineal numérica ha permitido que la comunidad científica y profesional, sin la necesidad de conocimientos computacionales y de hardware profundos, pueda usarlos de manera óptima y avanzar en sus investigaciones y desarrollos para generar más tecnología y conocimiento.

Su trabajo pionero se remonta a 1979 y sigue siendo uno de los líderes más destacados y activamente comprometidos en la comunidad HPC. Su carrera ciertamente ejemplifica el reconocimiento del Premio Turing a las "grandes contribuciones de importancia duradera". Sin duda para la realización de los trabajos anteriores, Dongarra ha colaborado internacionalmente con muchas personas y ha dirigido y formado a muchas personas en su calidad de supervisor de tesis de magíster y doctorado. ■

BIBLIOGRAFÍA

[1] Hockney RW, Jesshope CR. Parallel Computers 2: architecture, programming and algorithms. CRC Press; 2019.

[2] Navarro C A, Hitschfeld-Kahler N y Mateu L. A survey on parallel computing and its applications in data-parallel problems using GPU architectures. Communications in Computational Physics. 2014; 15(2):285-329.

[3] Kirk DB y Wen-Mei WH. Programming massively parallel processors: a hands-on approach. Morgan Kaufmann; 2016.

[4] Jack J. Dongarra. Link Google Scholar.

[5] LAPACK. https://netlib.org/lapack/ y https://en.wikipedia.org/wiki/LAPACK.

[6] Blackford LS, Petitet A, Pozo R, Remington K, Whaley RC, Demmel J, et al. An updated set of basic linear algebra subprograms (BLAS). ACM Transactions on Mathematical Software. 2002;28(2):135–51.

[7] PLASMA. https://icl.utk.edu/projectsfiles/plasma/html/doxygen/.

[8] MAGMA. https://icl.utk.edu/magma/.

[9] SLATE. https://icl.utk.edu/slate/.

[10] R. Clint Whaley y Jack J. Dongarra. Automatically tuned linear algebra software. Proceedings of the 1998 ACM/IEEE conference on Supercomputing. 1988. pp: 1–27.

[11] Julie Langou, Piotr Luszczek, Jakub Kurzak, Alfredo Buttari y Jack Dongarra. Exploiting the performance of 32 bit floating point arithmetic in obtaining 64 bit accuracy (revisiting iterative refinement for linear systems). Proceedings of the 2006 ACM/IEEE conference on Supercomputing. 2006. pp 113–es.

[12] HPL-AI. https://www.r-ccs.riken.jp/labs/lpnctrt/projects/hpl-ai/index.html.

[13] Abdelfattah A, Haidar A, Tomov S y Dongarra J. Performance, Design, and Autotuning of Batched GEMM for GPUs. 2016 International Conference on High Performance Computing. LNCS.