Técnicas de Optimización de Rutinas Paralelas de Álgebral Lineal en Sistemas Heterogéneos

Jesús Cámara

Tesis Doctoral para la obtención del Grado de Doctor en Informática

Dirigida por:

Javier Cuenca y Domingo Giménez

Escuela Internacional de Doctorado
Universidad de Murcia

23 de Julio de 2020

Motivación

Tesis

Presenta una metodología jerárquica para la auto-optimización de rutinas de álgebra lineal en sistemas computacionales heterogéneos.

Enfoque

Basado en el uso de técnicas de auto-optimización con capacidad de adaptación a los distintos niveles de complejidad del software (rutinas y librerías de álgebra lineal) y del hardware (cluster de nodos heterogéneos).

Finalidad

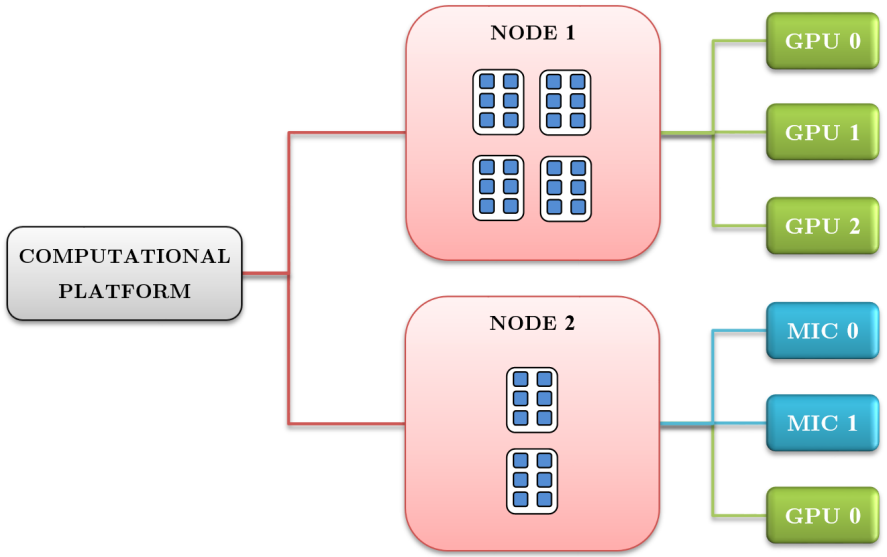
Proporcionar a usuarios una librería software que pueda ser utilizada en sus aplicaciones para obtener una ejecución optimizada con un uso eficiente de los recursos computacionales del sistema.

Índice de Contenidos

Índice de Contenidos

Programación Paralela

- Aumento de capacidad computacional de unidades de procesamiento.
- Abordar problemas complejos con alta demanda computacional.



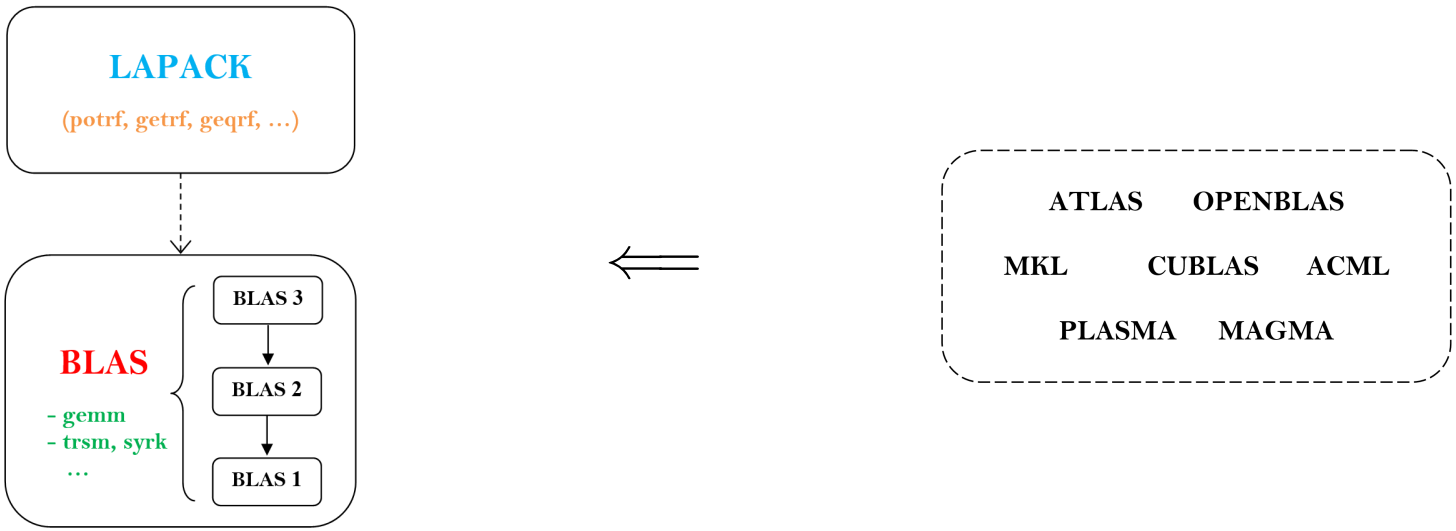
Unidades computacionales con distinta arquitectura hardware, capacidad computacional y velocidad de acceso a niveles de jerarquía de memoria.

Entornos de Programación Paralela

- Librería Pthreads de C y OpenMP para sistemas memoria compartida.
- Librería PVM y MPI para sistemas con memoria distribuida.
- Entorno de desarrollo CUDA para la programación de GPUs de NVIDIA.
- Entornos emergentes: computación en GPUs virtualizadas con rCUDA.

Librerías de Álgebra Lineal

Básica | Implementaciones Optimizadas



Objetivo

General

Desarrollo de una metodología de auto-optimización jerárquica para la obtención de rutinas eficientes de álgebra lineal en plataformas heterogéneas.

Metodología Jerárquica:

- Determinar valores del conjunto de parámetros ajustables (de rutinas y del entorno computacional) que permitan obtener tiempos de ejecución cercanos al óptimo experimental.
- Guiar optimización de rutinas en diferentes niveles de la jerarquía.
- Válida para múltiples rutinas, librerías y sistemas computacionales.

Metodología

Comenzar en niveles más bajos de la jerarquía (hardware y software) y pasar a siguientes niveles hasta llegar al mayor nivel en cada caso.

Varias Rutinas y Sistemas

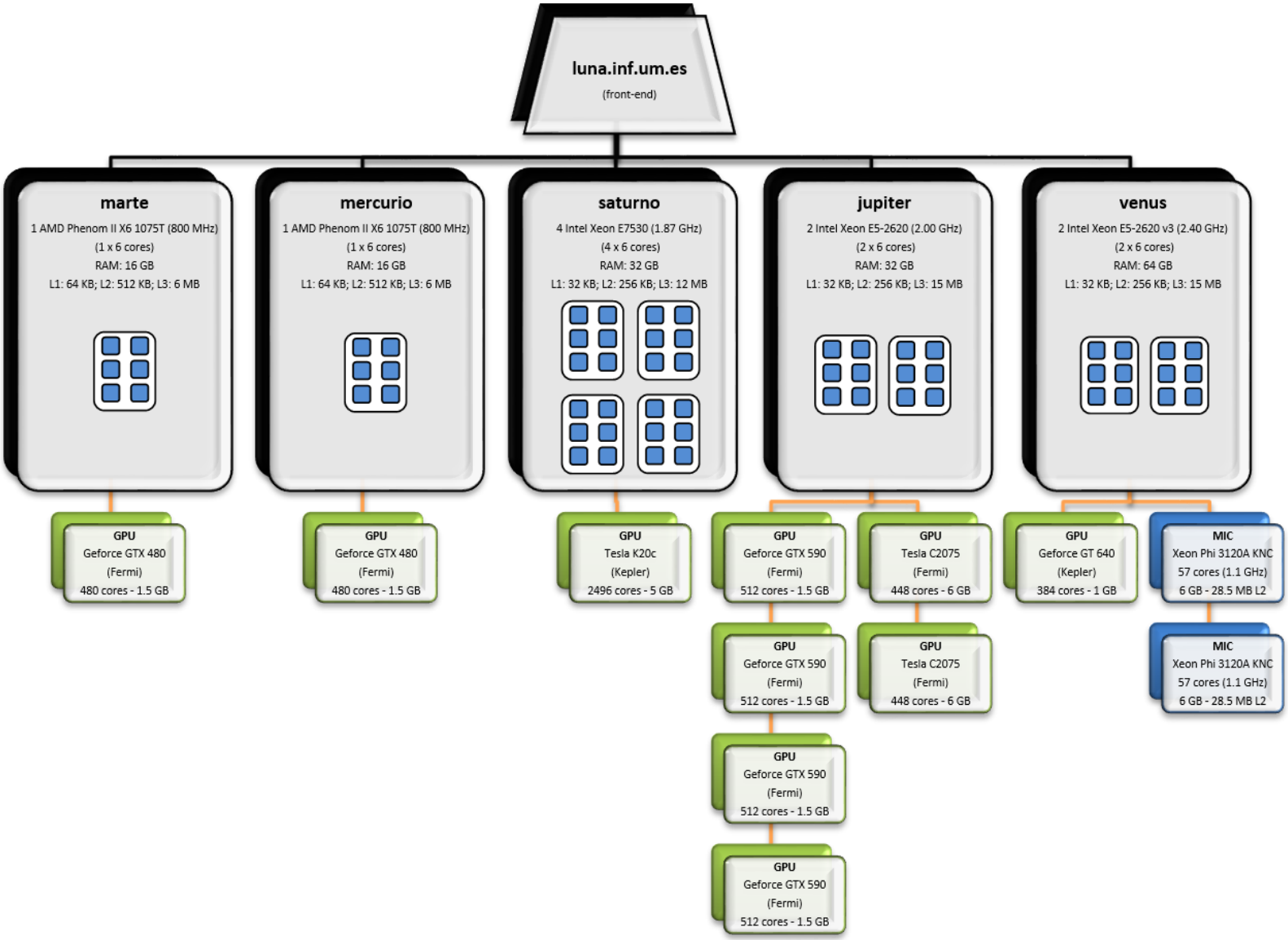
- Multiplicación de matrices en elemento computacional, nodos y cluster.
- Rutinas con parámetros de distinto tipo (nivel de recursión en Strassen o tamaño de bloque en LU) y que invoquen a multiplicación básica.
- Diferentes configuraciones en mismo nivel hardware (multicores con distinto número de cores o agrupaciones de elementos de cómputo a nivel de nodo o de cluster).

Estudio Experimental

Con diferentes rutinas en cada nivel de jerarquía software y diferentes combinaciones de elementos de cómputo en cada nivel de la jerarquía hardware, analizando qué información de los niveles inferiores se puede usar para acelerar la toma de decisiones de los valores de los parámetros ajustables.

Índice de Contenidos

Sistema Computacional: **Heterosolar**



Heterosolar

Apropiado para análisis experimental de metodología jerárquica propuesta:

- Componentes computacionales organizados en niveles.
- CPU multicore con diferente número de cores (con/sin *hyperthreading*).
- Nodos con GPUs de distinto tipo y capacidad computacional.
- Nodo que combina CPU multicore con GPU e Intel Xeon Phi.

Agrupación de Elementos Computacionales

A nivel de nodo o cluster $\rightarrow$ experimentar con sistemas computacionales con distintos grados de complejidad y heterogeneidad.

Herramientas Software

Esquema de Programación Híbrida

- OpenMP para explotar paralelismo y arquitectura de memoria a nivel de nodo (combinado con CUDA para GPU y con directivas para MIC).
- MPI para comunicación entre nodos a nivel de cluster.

Librerías de Álgebra Lineal

Implementaciones eficientes de las rutinas que son invocadas en elementos computacionales básicos (Intel MKL en CPU + MIC y cuBLAS en GPU).

Rutinas de Álgebra Lineal

Para ilustrar funcionamiento de la metodología jerárquica propuesta.

- Básicas (multiplicación de matrices): sobre las que se implementan otras rutinas de mayor nivel.
- Mayor Nivel (Strassen y LU): delegan parte de la optimización a rutinas básicas a las que invocan (multiplicación de matrices).

Índice de Contenidos

Técnicas de Optimización y Auto-Optimización

Objetivo

Dotar al software de capacidad para adaptarse de forma autónoma a las características del sistema computacional donde se ejecuta.

Técnicas

Obtener buenas prestaciones mediante la aplicación de un proceso para la selección de los mejores valores de los parámetros ajustables (algorítmicos o de la plataforma).

Escenario

- ¿Qué sistema es el más apropiado para resolver un problema?
- ¿Cuál es la mejor forma de resolverlo en un sistema concreto?

Técnicas de Optimización y Auto-Optimización

En función de la metodología que utilizan.

Modelado del Tiempo de Ejecución

Contiene modelo del tiempo de ejecución de la rutina (SP hardware + librerías básicas) y AP seleccionados.

Experimental

Búsqueda experimental (guiada o exhaustiva) de los valores de los AP de las rutinas (en rango o conjunto discreto predeterminado) que ofrecen las mejores prestaciones.

Híbrida: Modelo + Experimentación

Experimentos para obtener visión del comportamiento de la rutina y uso del modelo para determinar mejores valores de los AP.

Índice de Contenidos

Contexto

Plataformas Actuales

- Unidades de procesamiento paralelo distintas (CPU, GPU, MIC).
- Agrupación de elementos de cómputo de diferente forma $\rightarrow$ Adoptar enfoque jerárquico por niveles en el diseño de técnicas de optimización.

Agrupación Jerárquica

- Diferentes parámetros en función de nivel de jerarquía y UC utilizadas.
- Diferentes formas de paralelizar las rutinas para optimizar su ejecución.

Técnicas de Optimización Existentes

- En distintos niveles de jerarquía.
- Uso eficiente de recursos computacionales.
- Selección automática de valores de parámetros ajustables (de rutinas, de unidades de cómputo o propios del nivel) en cada nivel de jerarquía.

Optimización Jerárquica

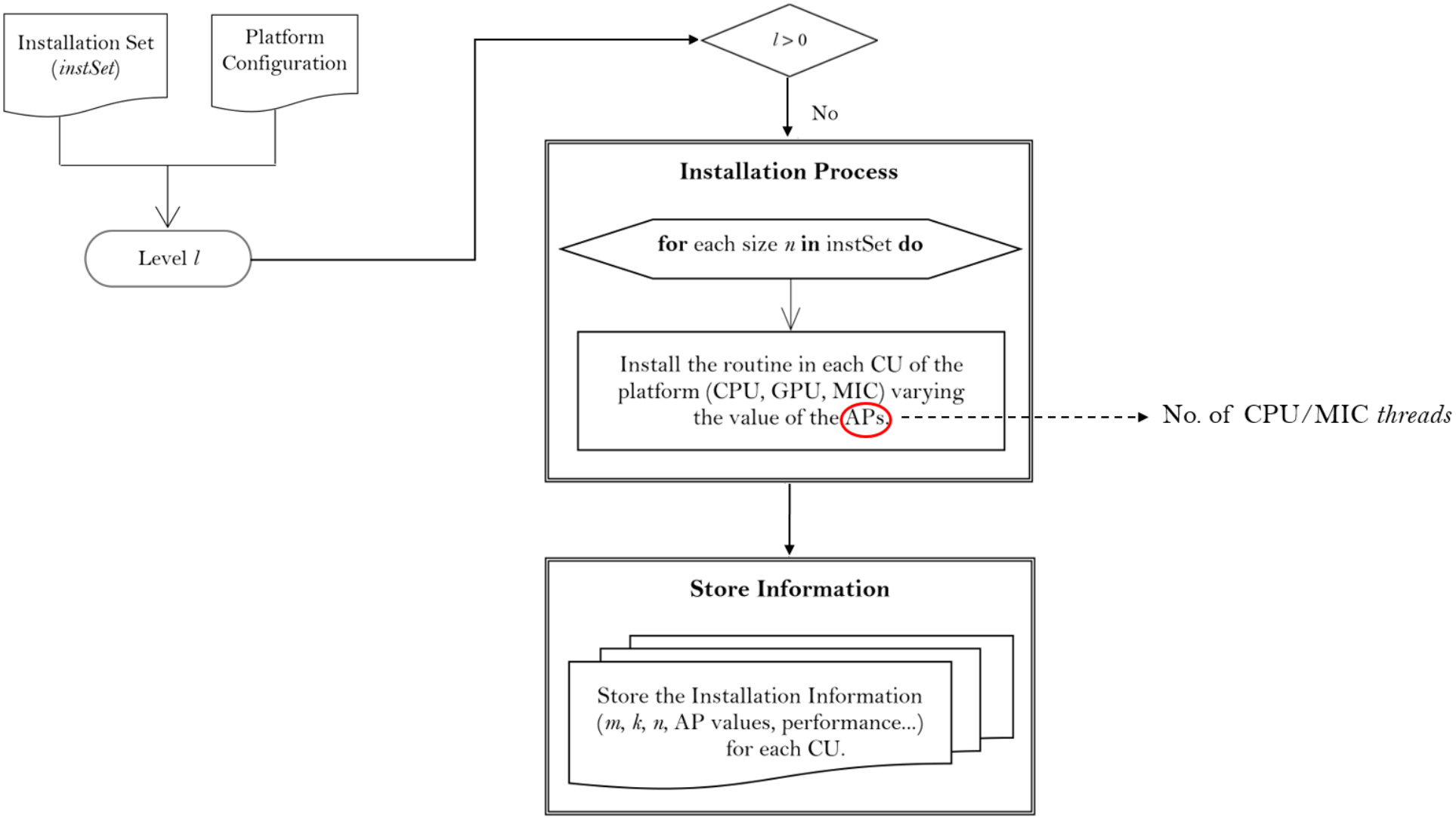
Objetivo

Minimizar tiempo de ejecución de rutinas en cada nivel reutilizando la información de optimización proporcionada por niveles inferiores de la jerarquía.

Metodología Propuesta

- Diseño con estructura jerárquica multinivel.
- Organiza unidades de cómputo y rutinas de forma jerárquica.
- Segmenta el proceso de instalación de rutinas $\rightarrow$ mayor escalabilidad.
  - Obtención de APs se reparte entre los diferentes niveles de la jerarquía.
  - Toma de decisiones usando información de instalación del nivel inferior.
- Ejecuta rutinas decidiendo de forma automática la mejor configuración hardware y software en cada nivel.
- Enfoque *bottom-up*: comienza en nivel más bajo de la jerarquía y asciende progresivamente a niveles superiores.

Instalación Jerárquica (I)



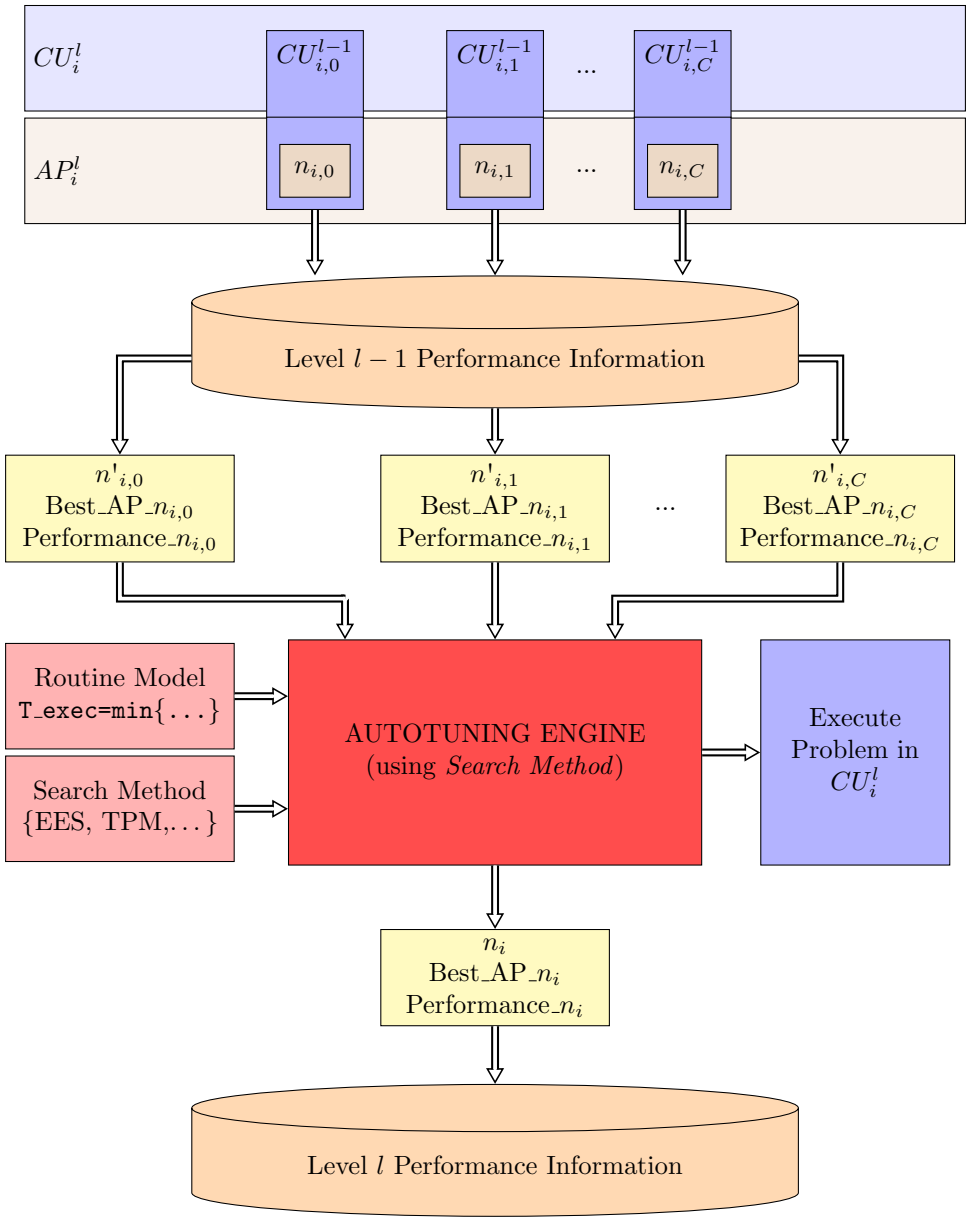
Cada UC se trata de forma aislada e independiente → El CI puede ser diferente.
No se considera tiempo de transferencia de datos (pero se almacena para su uso en niveles superiores de la jerarquía)

Instalación Jerárquica (II)

Instalación en Nivel Superior

- La rutina se instala sobre UC compuestas por UC de nivel inferior (nodos o unidades básicas de procesamiento).
- Considera coste de comunicaciones intra-nodo.
- AP: cantidad de trabajo a asignar a cada UC para tamaño dado.
- Optimización: usando información de instalación almacenada en nivel inferior para cada UC sobre la que se ha instalado la rutina.
- Almacena configuración de reparto entre UC con la que se obtiene el menor tiempo de ejecución (y el rendimiento).

Uso de Información de Instalación



Jerarquía Bidimensional

Diferentes Escenarios Hardware y Software

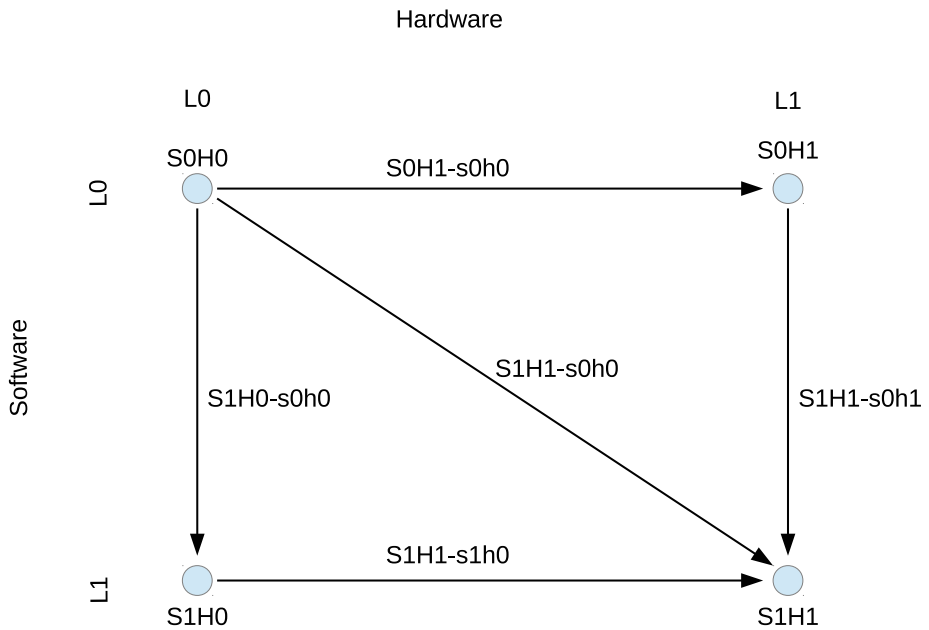
- Jerarquía de Elementos Computacionales:
  - Plataforma vista como UC global compuesta recursivamente por un conjunto de UC más sencillas de las que sólo se obtiene información de instalación para optimizar ejecución de rutinas.
  - Subniveles del nivel actual (agrupación de UC).
- Jerarquía de Rutinas: rutinas de mayor nivel que hacen uso de rutinas básicas optimizadas en niveles inferiores de la jerarquía.

Parámetros Ajustables

- Tipo depende del nivel hardware y software de la jerarquía.
- Obtención de valores en función del método aplicado (teórico, experimental o guiado).

Jerarquía Bidimensional

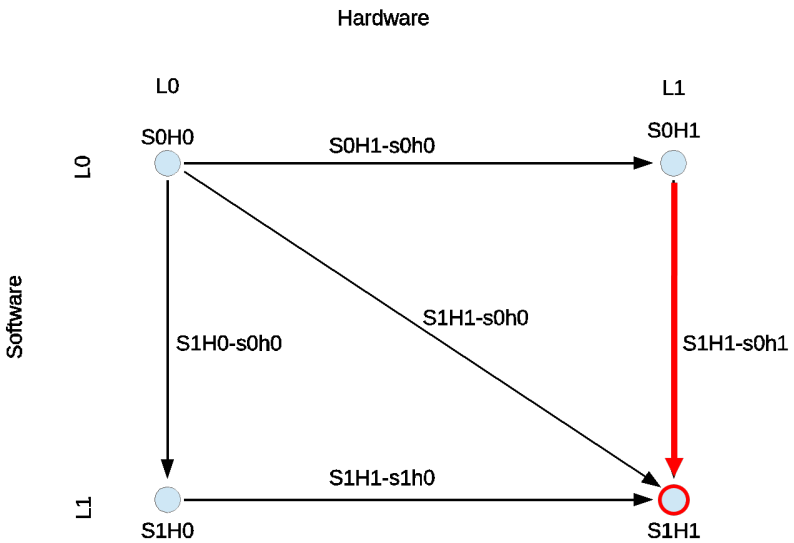
Diferentes formas de realizar la instalación jerárquica (círculos) y de aplicar la metodología de optimización multinivel (flechas) en función de cómo se use la información de instalación almacenada para obtener el valor de APs.

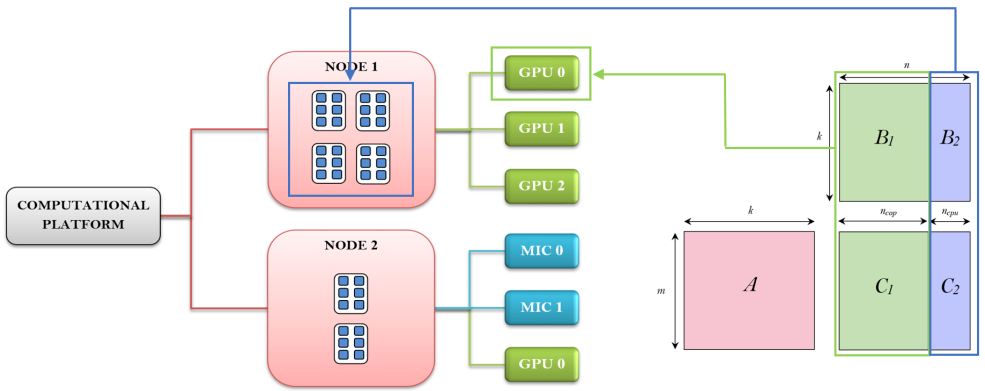


Opciones de instalación en niveles 0 y 1 de la jerarquía.

S1H1-s0h1

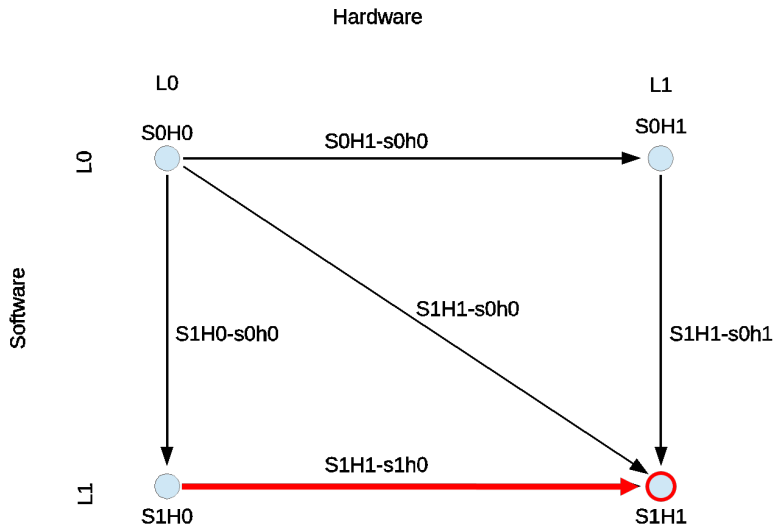
Usa en el caso base la versión de la rutina básica de multiplicación de matrices instalada en nivel S0H1 (APs: reparto de la carga entre UC básicas + propios de cada UC).





S1H1-s1h0

Ejecuta multiplicaciones básicas de Strassen asignándolas a UC básicas.

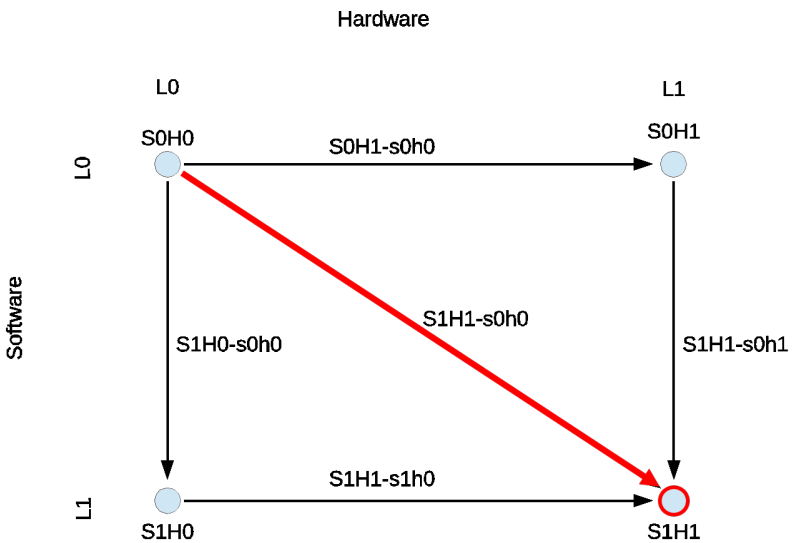


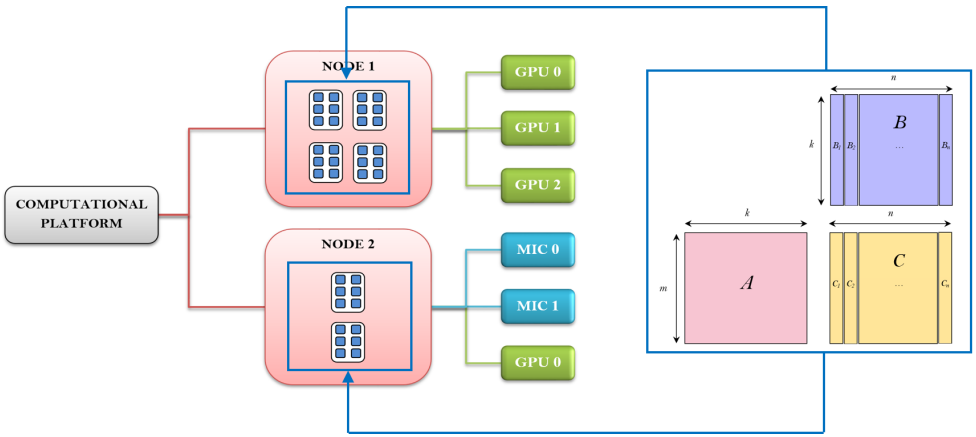
En nivel S1H0

Hace uso de versión optimizada de la multiplicación básica de matrices instalada para cada UC básica (S1H0-s0h0).

S1H1-s0h0

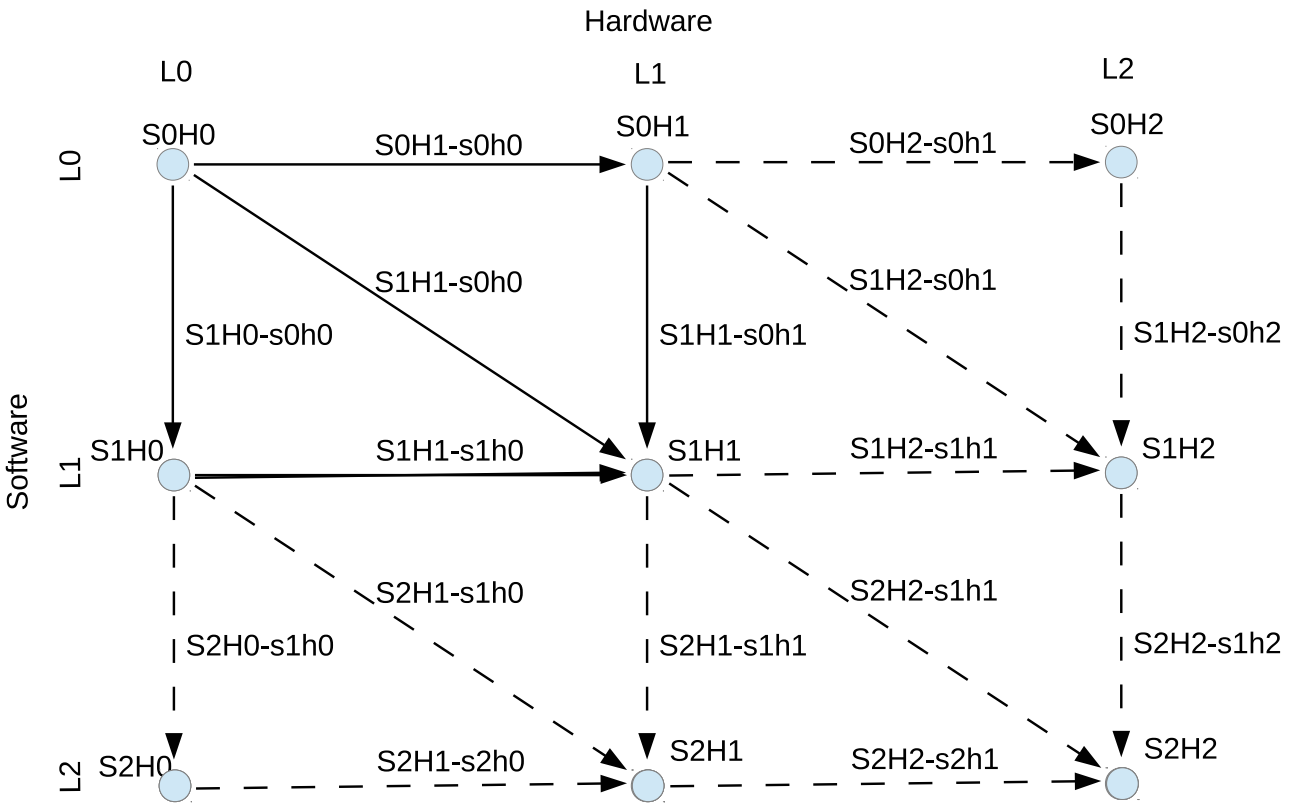
Usa en el caso base la versión de la rutina básica de multiplicación de matrices instalada en nivel S0H0 (APs: propios de cada UC).
Ejecuta de forma sucesiva multiplicaciones básicas en UC básicas.





Jerarquía Bidimensional

Extensible a siguiente nivel de la jerarquía (S2H2) $\rightarrow$ Aumento considerable del número de opciones de instalación.



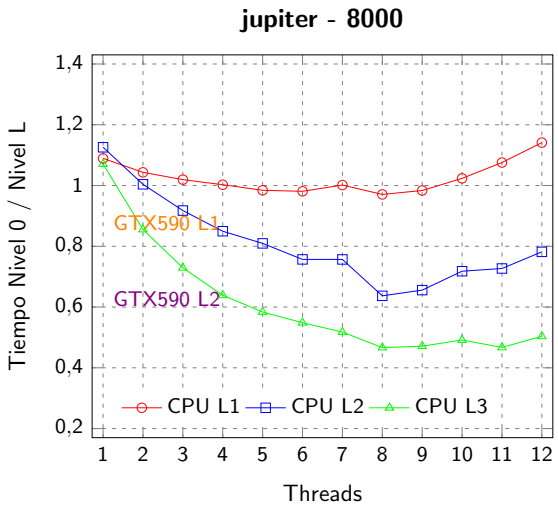
Índice de Contenidos

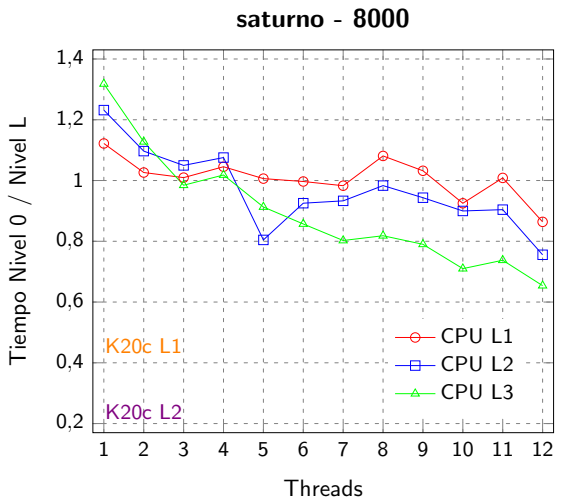
Escenario

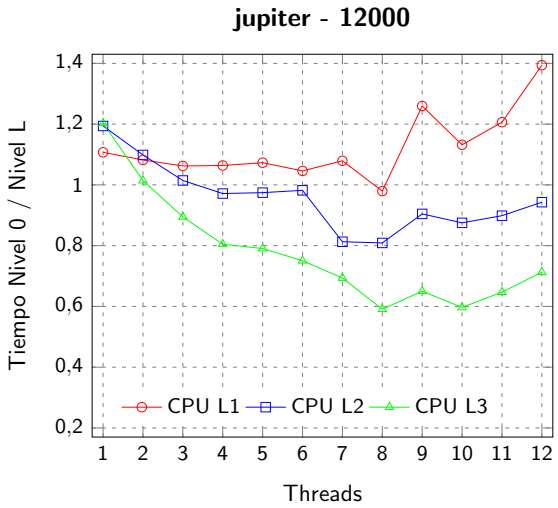
Ejemplos puntuales de utilización de la metodología en distintos niveles de la jerarquía (software y hardware)

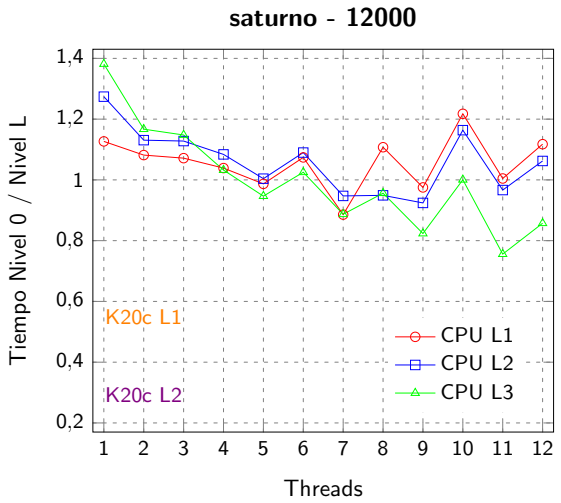
	UC Básicas (H0)	Nodo (H1)	Plataforma (H2)
MM (S0)	CPU GPU MIC	CPU+multiGPU CPU+multiMIC CPU+GPU+multiMIC	cluster
Strassen (S1)	CPU GPU	CPU+multiGPU CPU+multiMIC	nodos virtuales
LU (S1)	-	CPU+multiGPU	-

Multiplicación de Strassen en UC Básicas (S1H0)





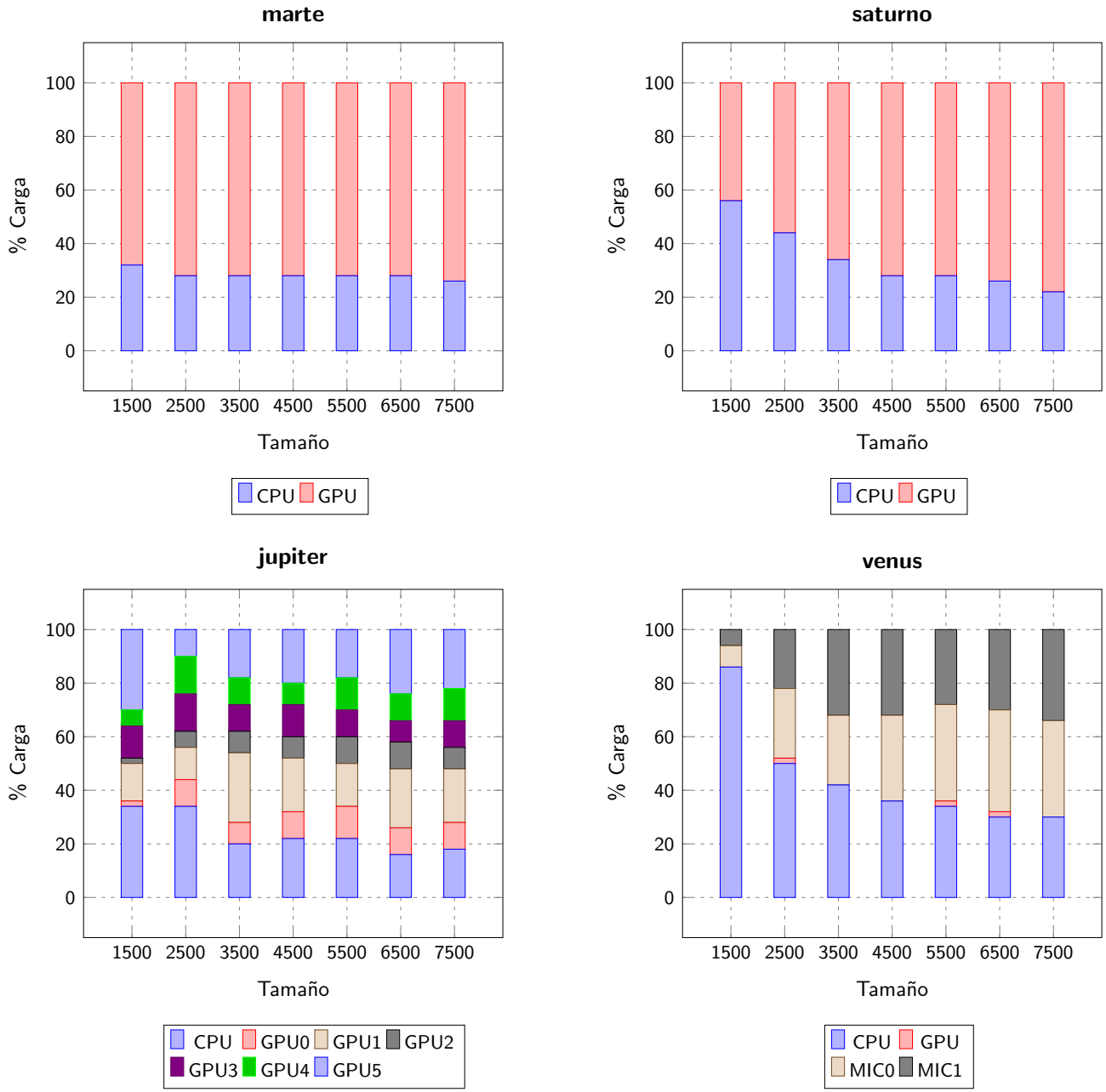




Con diferentes configuraciones (threads CPU y tipo de GPUs), decisiones distintas.

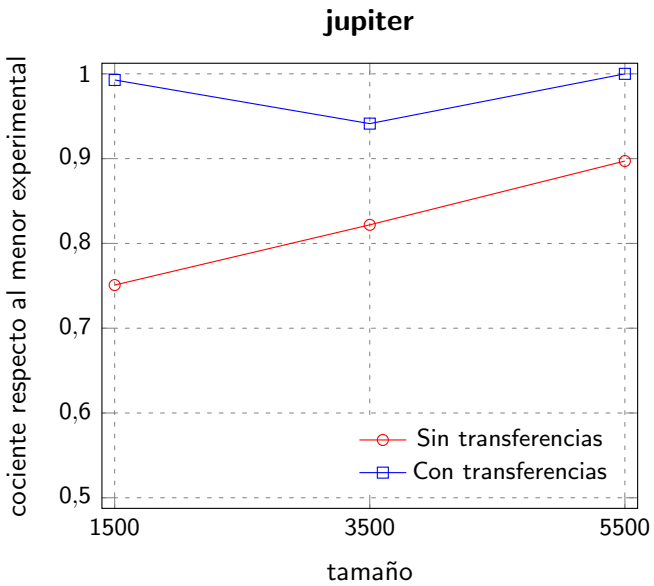
Al usar GPU, el nivel de recursión es 0 (multiplicación directa con cuBLAS)

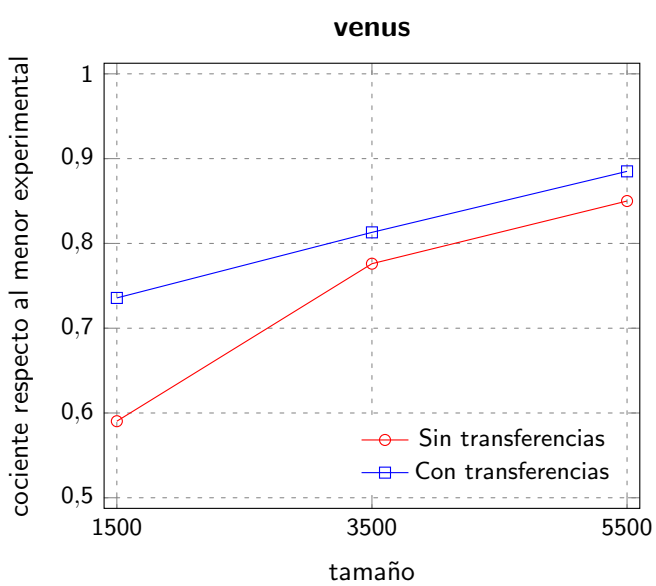
Multiplicación de Matrices en Nodo (S0H1)



Considerar Transferencias

- Mejores decisiones → Predicciones más precisas.
- Incluir coste en modelo (término K_{comm}) y obtener tiempo experimental con decisiones tomadas.





$$\text{CI-L0} = \{1000, 2000, \ldots, 6000\}$$

Mejores decisiones cuando se consideran transferencias (cociente cercano a 1) y mejoran al aumentar el tamaño del problema.

Multiplicación de Strassen en Nodo (S1H1)

Método con el que se obtiene el menor tiempo de ejecución en **jupiter**, UC y valores de los AP seleccionados por la metodología de auto-optimización dentro de la multiplicación de matrices a nivel de nodo (S1H1-s0h1)

	Decisiones Nivel 1		Decisiones Nivel 0			
				Distribución del Trabajo		
n	Método	Combinación de UC	CPU threads	CPU	1ª C2075	2ª C2075
2000	Direct	CPU+2 C2075	7	240	880	880
3000	Direct	CPU+2 C2075	11	600	1200	1200
4000	Direct	CPU+2 C2075	12	880	1520	1600
5000	Direct	CPU+2 C2075	10	1000	2000	2000
6000	StrassenL1	CPU+2 C2075	11	600	1200	1200
7000	StrassenL1	CPU+2 C2075	11	630	1470	1400
8000	StrassenL1	CPU+2 C2075	12	880	1520	1600
9000	StrassenL1	CPU+2 C2075	11	900	1800	1800
10000	StrassenL1	CPU+2 C2075	10	1000	2000	2000
11000	StrassenL1	CPU+2 C2075	12	1320	2090	2090
12000	StrassenL2	CPU+2 C2075	11	600	1200	1200

Optimización de Tipo S1H1-s0h1

Delegando a multiplicación híbrida la selección de los AP (distribución del trabajo en nivel H1 y número de threads MKL en CPU en nivel H0). No selección de APs en nivel S1 (tamaño de bloque fijo).

En **jupiter** con $n = 10000$ y $tb = 1000$

$n \times 1000 \times n$ / n	**Distribución de la carga** CPU, GTX590, C2075, GTX590	GFlops
9000	1980,1800,3420,1800	236
8000	1920,1600,3040,1440	377
7000	1540,1400,2660,1400	359
6000	1560,960,2280,1200	340
5000	1100,1000,1900,1000	320
4000	800,800,1600,800	292
3000	780,720,1500,0	273
2000	440,520,1040,0	211
1000	480,0,520,0	134

Se usan menos UC conforme disminuye el tamaño de las matrices.

Multiplicación de Strassen usando Nodos Virtuales

Uso de Nodos Virtuales

- Nodo visto como UC de nivel 2 mediante combinación de subnodos (UC de nivel 1 formadas por agrupación de UC de `H0`) dentro de nodo.
- Permite definir múltiples niveles hardware dentro de un nodo.
- Explotar paralelismo de 2 niveles (OpenMP+MKL).

	1 thread OpenMP (no paralelismo anidado)	
n	*12 threads MKL*	*12 threads MKL + C2075*
3500	1.86	0.57
7000	9.27	3.25

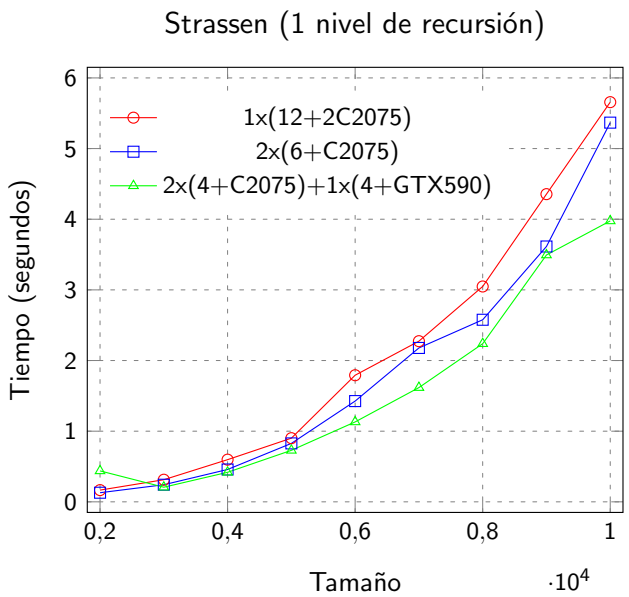
	2 threads OpenMP		
n	*12 threads MKL*	*6 threads MKL*	*6 threads MKL + C2075*
3500	0.83	0.95	0.51
7000	4.78	5.29	1.97

Menor tiempo de ejecución con paralelismo anidado
(explota de forma más eficiente el *hyperthreading* de la CPU)

Ejecución (`S1H2-s0h1`)

- Asigna dinámicamente multiplicaciones básicas a subnodos.
- En cada subnodo, las decisiones se delegan a multiplicación básica (rutina de MKL), que se ejecuta haciendo uso de versión auto-optimizada.

Diferentes configuraciones de subnodos en **jupiter**:



El uso de subnodos reduce el tiempo de ejecución.

Herramientas para obtención de APs

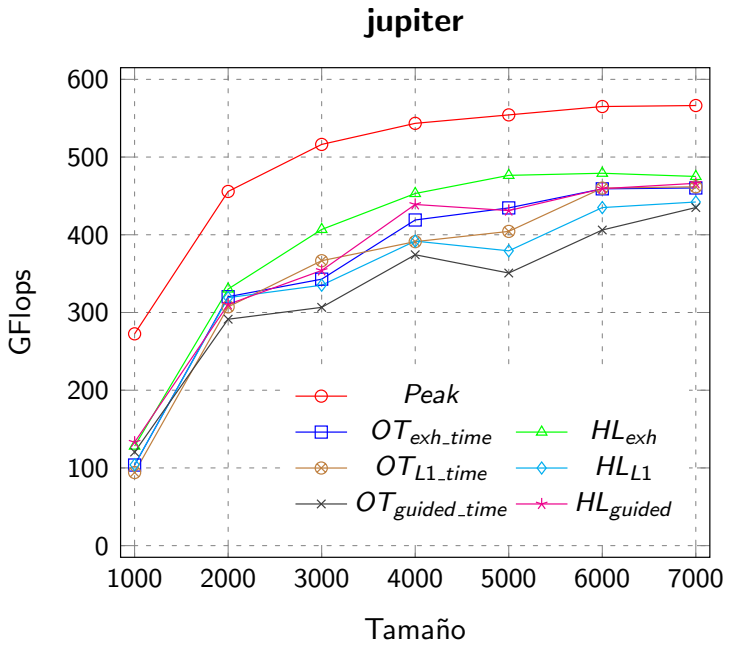
OpenTuner

- Herramienta software para obtención de valores de AP.
- Realiza búsqueda de AP mediante el uso simultáneo de diferentes técnicas heurísticas sobre un dominio de valores concreto.
- Integrada en metodología de optimización jerárquica como una técnica más de búsqueda dentro del proceso de auto-optimización.

Experimentos

- Rutina de multiplicación de matrices.
- Plataforma con CPU+GTX590+C2075 (subnodo de **jupiter**).
- Parámetros: threads de CPU y distribución de la carga entre UC.
- Limitando el tiempo de búsqueda, estableciendo como cota superior el empleado por la metodología de auto-optimización al instalar la rutina de forma Exhaustiva y Guiada (a partir de información obtenida de `H0`) y Totalmente Exhaustiva (`H1`).

OpenTuner



Resultados

- Rendimiento obtenido por OpenTuner se aproxima a óptimo exhaustivo conforme aumenta el límite establecido en el tiempo de búsqueda.
- La metodología jerárquica combinada con búsqueda guiada (HL_{guided}), mejora resultados obtenidos y reduce tiempo de búsqueda.

Índice de Contenidos

Librerías con Asignación Dinámica de Tareas

Objetivo

Aplicar metodología jerárquica a librería cuyas rutinas se ejecutan haciendo uso de un planificador de tareas con asignación dinámica de kernels computacionales básicos a unidades de procesamiento paralelo.

Aplicación de Metodología

- En nivel 1 (hardware y software) de la jerarquía (`S1H1`).
- La ejecución de rutinas básicas en elementos computacionales de nivel 0 se delega al planificador de tareas.

Selección de APs

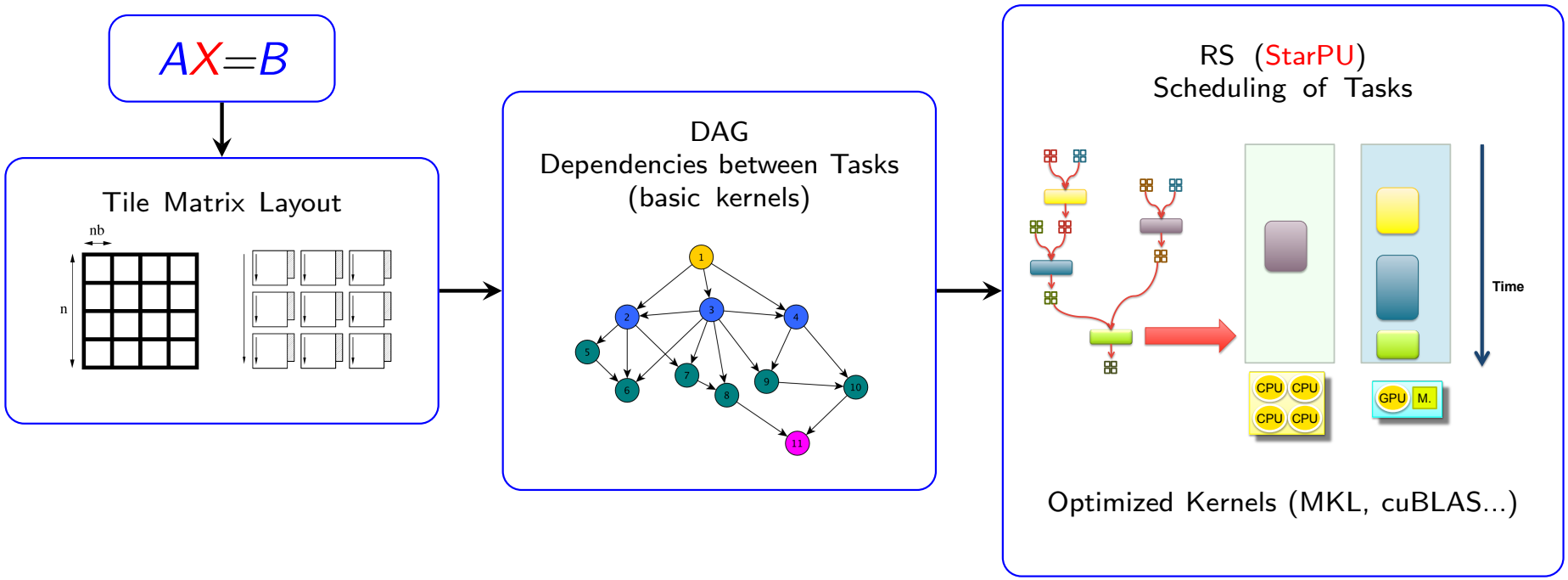
Mediante estrategias de búsqueda adoptando enfoque empírico y simulado.

Parámetros

Tamaño de bloque (y bloque interno), número de UC, política planificación.

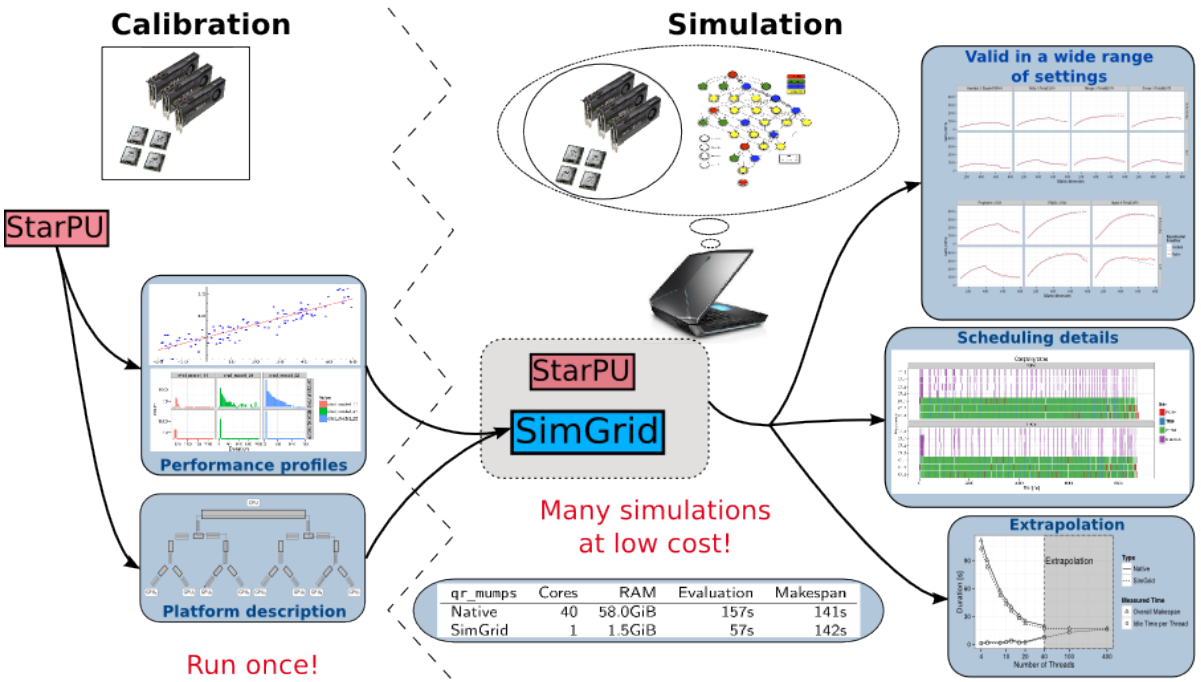
Estructura General de Chameleon

- Librería de álgebra lineal densa con asignación dinámica de tareas.
- Derivada de PLASMA $\rightarrow$ Esquema algorítmico basado en *tiles*.



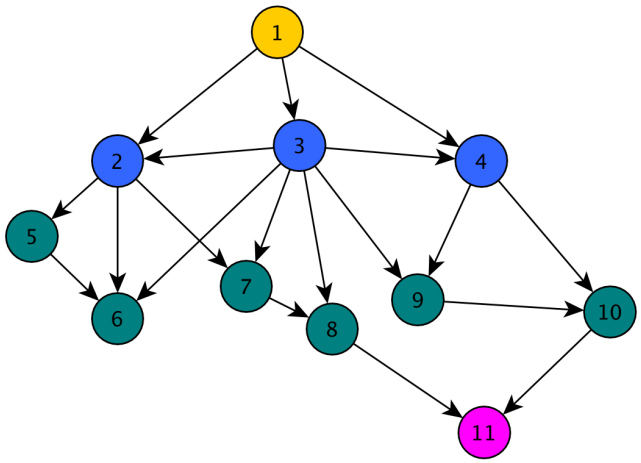
Uso de Simulación

- Estimación del tiempo de ejecución (y del rendimiento) de las rutinas.
- Uso de modelos de rendimiento (*codelets*) generados durante la instalación de la rutina para cada kernel computacional.
- Estimaciones precisas → calibración de los modelos durante instalación.
- Nuevas estimaciones sin requerir uso de la plataforma.



Descomposición de Cholesky

```
for (j = 0; j < n/nb; j++) {
    POTRF(RW,A[j][j]);
    for (i = j+1; i < n/nb; i++)
        TRSM(RW,A[i][j], R,A[j][j]);
    for (i = j+1; i < n/nb; i++) {
        SYRK(RW,A[i][i], R,A[i][j]);
        for (k = j+1; k < i; k++) {
            GEMM (RW,A[i][k],
                  R,A[i][j],
                  R,A[k][j]);
        }
    }
}
```



Rutinas básicas actúan sobre diferentes bloques de datos cuyo tamaño viene determinado por valor de $nb \rightarrow$ AP a determinar.

Factorización LU

```
for (j = 0; j < n/nb; j++) {
     DGETRF(R,A[j][j], W,L[j][j], W,U[j][j]);
     for (i = j+1; i < n/nb; i++)
          DGESSM(R,A[j][i], R,L[j][j], W,U[j][i]);
     for (i = j+1; i < n/nb; i++) {
          DTSTRF(R,A[i][j], R,L[i][j], RW,U[j][j]);
          for (k = j+1; k < n/nb; k++)
               DSSSSM(R,L[i][j], RW,U[j][k], RW,A[i][k]);
     }
}
```

Rutinas básicas actúan sobre diferentes bloques de datos cuyo tamaño viene determinado por valor de *nb* e *ib* $\rightarrow$ APs a determinar.

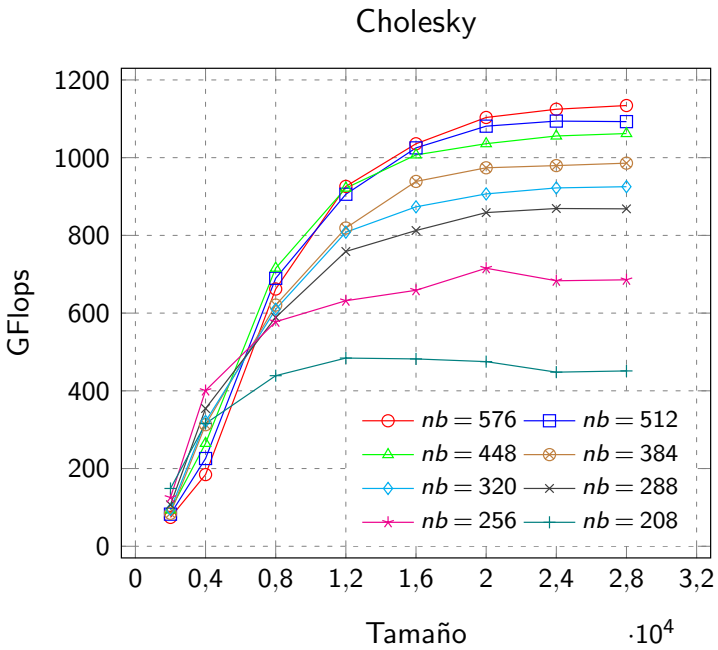
Metodología de Optimización

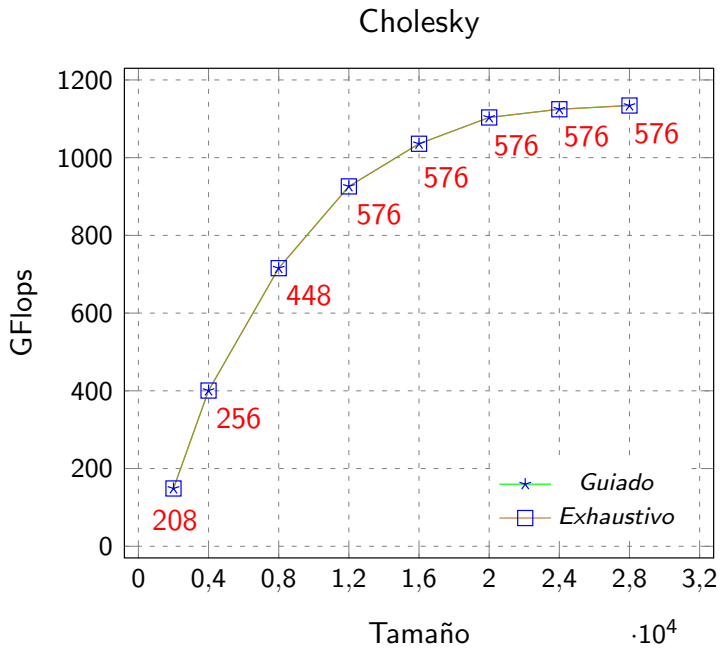
Se aplica con diferentes estrategia de búsqueda (Exhaustiva, Guiada_1D, Guiada_2D) usando los dos enfoques propuestos: empírico (en **jupiter**) y simulado (con **SimGrid**).

Experimentos

- CI = {2000, 4000, 8000, 12000, 16000, 20000, 24000, 28000}
- CI_Bloque = {208, 256, 288, 320, 384, 448, 512, 576}

Enfoque Empírico

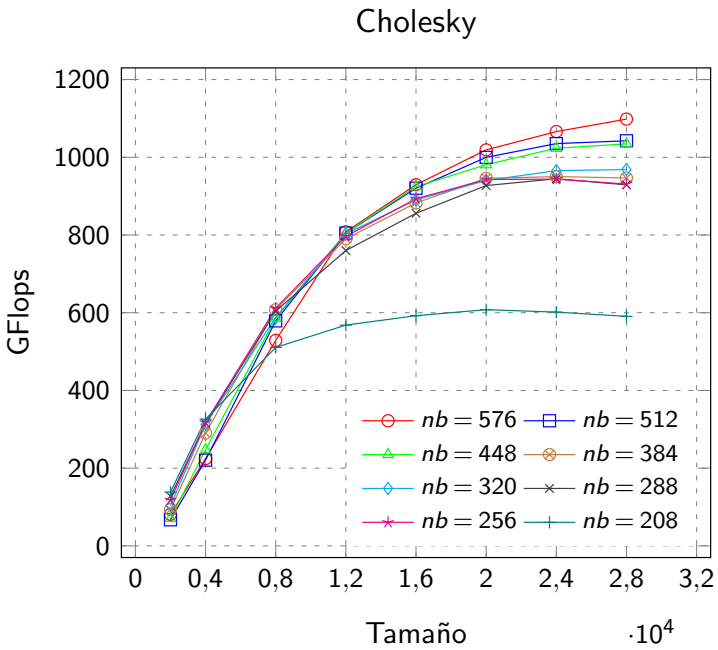


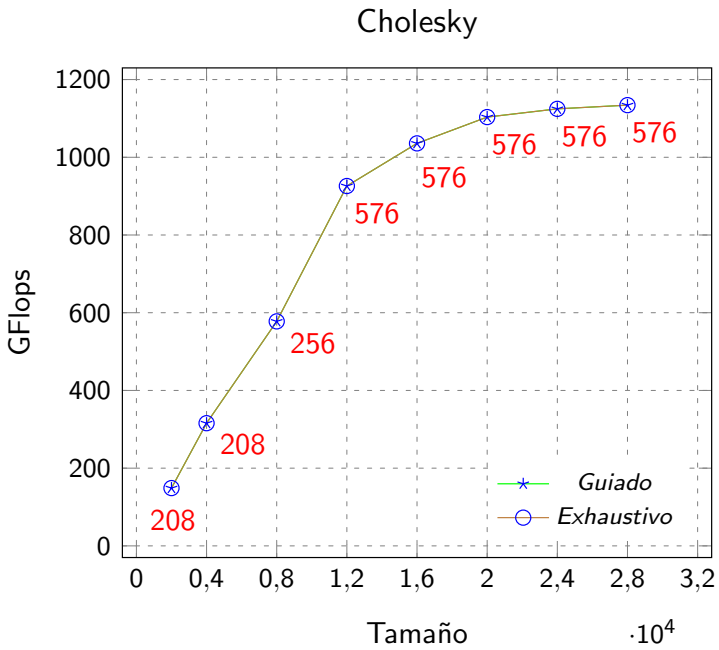


Valores obtenidos para *nb* coinciden.
Tiempo Instalación: Exhaustivo = 30 min ; Guiado = 154 seg.
Para tamaños de problema $>$ 8000, el mejor valor para *nb* corresponde al mayor del conjunto utilizado.

Enfoque Simulado

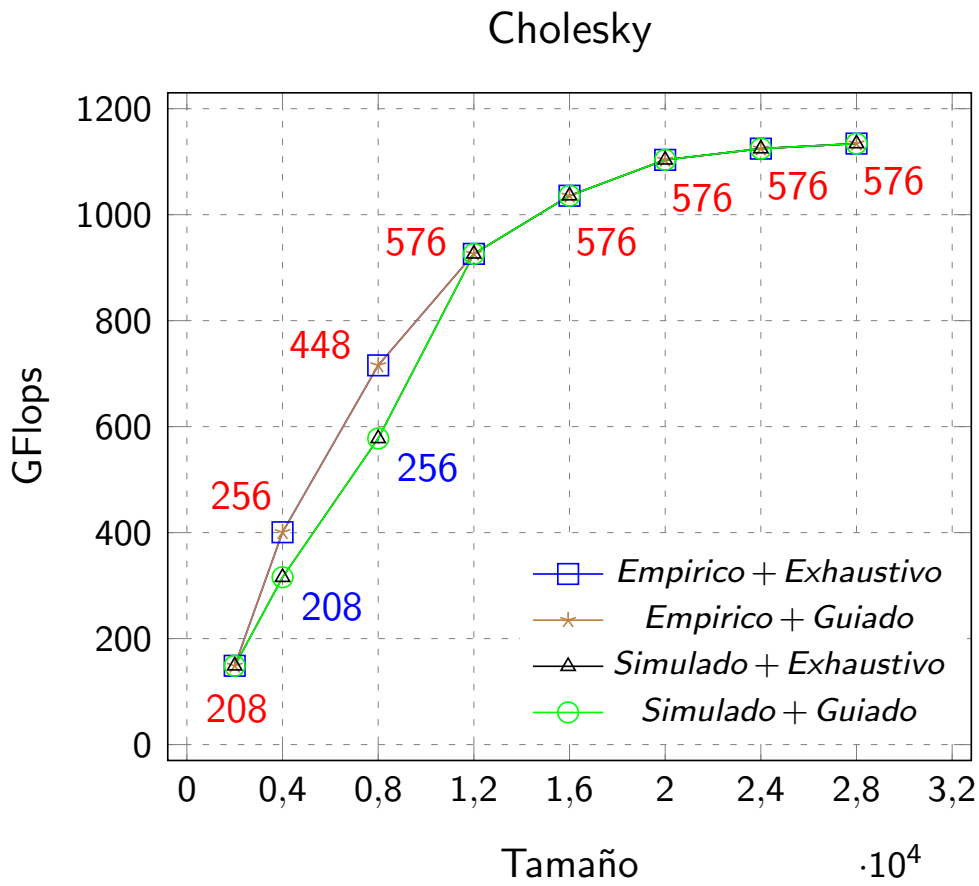
Uso del simulador SimGrid con mismas estrategias de búsqueda haciendo uso de información almacenada tras la instalación de la rutina.





Tiempo Instalación: Exhaustivo = 30 min ; Guiado = 84 seg.

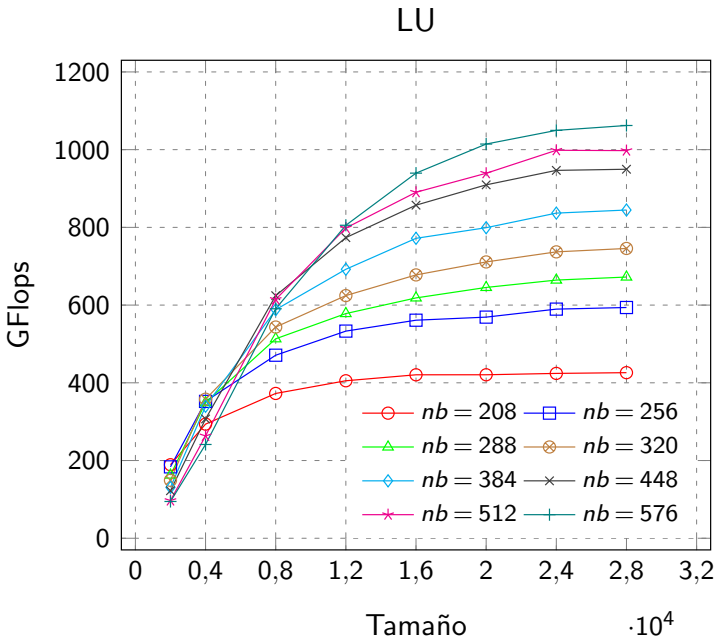
Empírico vs. Simulado

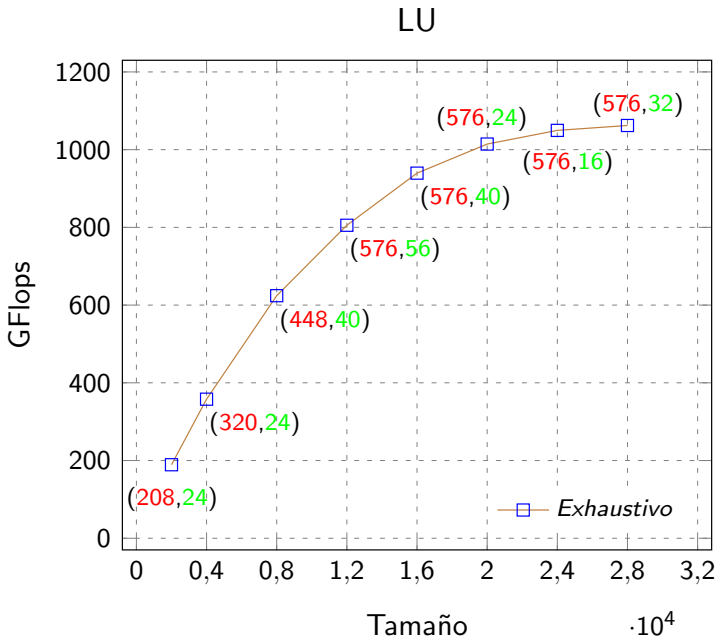


Desviación para tamaños de problema pequeños → Recalibrar *codelets*.

Enfoque Empírico

Evolución del rendimiento para cada par (n, nb) usando mejor valor de ib.

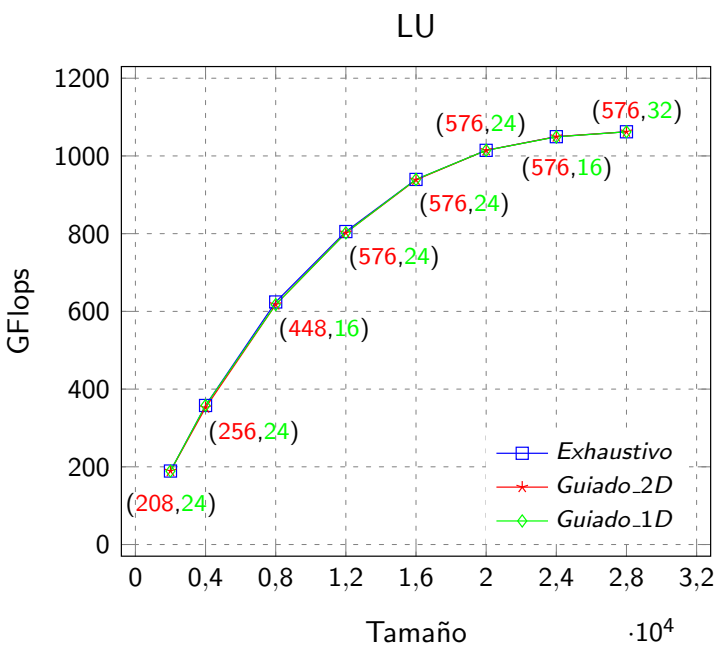


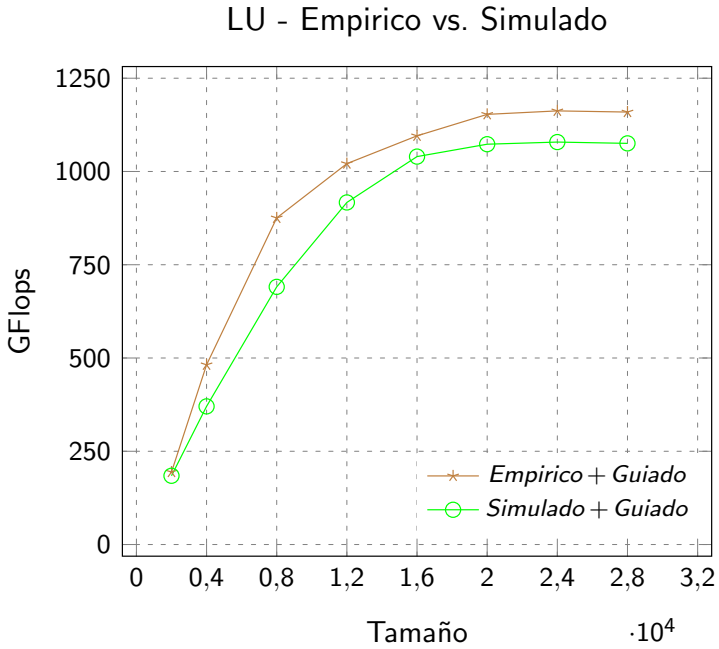


Tiempo Instalación: 6 h.

Los valores de ib varían entre 16 y 56.
Para tamaños de problema > 8000, el mejor valor para nb corresponde al mayor del conjunto utilizado.

Optimización





Tiempo Instalación: Guiado_1D = 32 min ; Guiado_2D = 6 min.

Pequeñas variaciones en selección de APs (principalmente en *ib*) pero no afecta en gran medida al rendimiento obtenido.

Parámetros del Sistema

Número de cores de CPU y número de GPUs.

Algoritmo

- Añade de forma sucesiva UC en función de su capacidad computacional (siempre que mejore el rendimiento obtenido).
- Aplica metodología de auto-optimización con búsqueda guiada bidireccional para obtener el mejor valor de *nb*.
- En siguiente iteración usa como punto de partida el mejor valor de *nb*.

n	*nb*	*CPU_Cores*	*GPUs*	*Tuned*	*Default*	%
1000	112	12	$\{-\}$	76	46	40
2000	192	9	$\{1,5,0\}$	164	117	29
3000	192	8	$\{1,5,0,2\}$	285	196	31
4000	240	7	$\{1,5,0,2,3\}$	412	352	15
5000	256	6	$\{1,5,0,2,3,4\}$	545	465	15
6000	256	6	$\{1,5,0,2,3,4\}$	626	554	12
7000	320	6	$\{1,5,0,2,3,4\}$	687	608	12
8000	320	6	$\{1,5,0,2,3,4\}$	753	682	10

Parámetros de la Librería

Política de Planificación

Establece el momento en que los kernels pasan a ejecutarse en UC básicas.

Planificador

Planifica tareas sin tener en cuenta capacidad computacional de UC básicas.

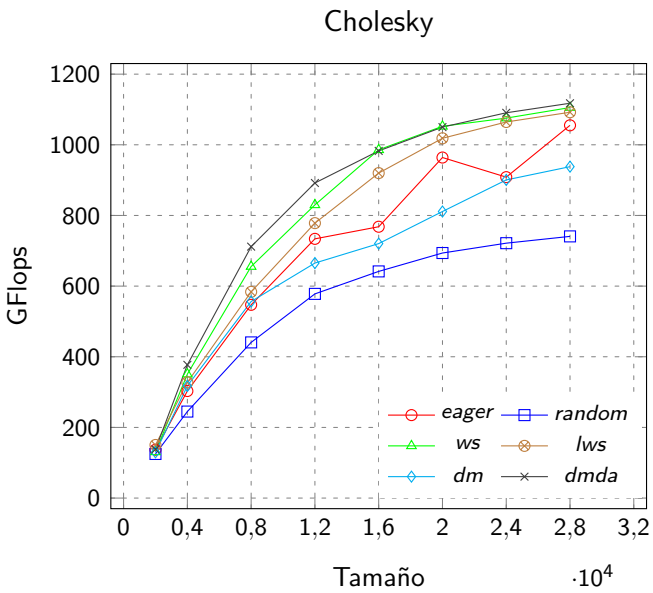
Decisión de Planificación

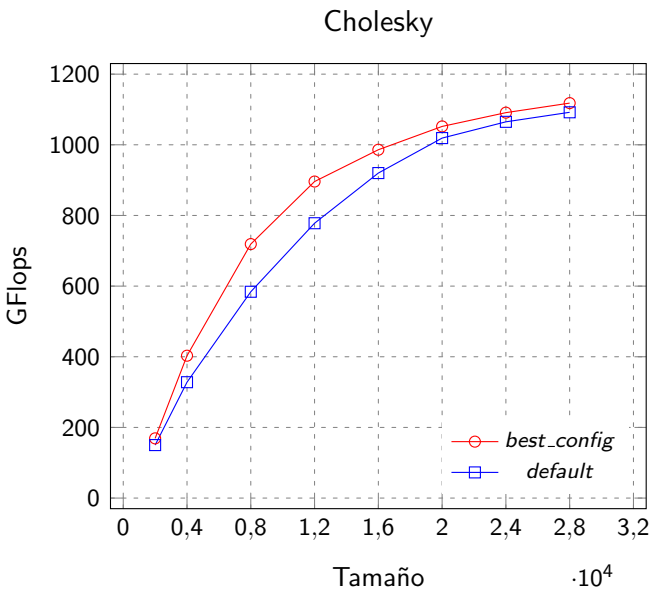
- Basándose en prioridades o en la disponibilidad de las UC.
- Usando información de modelos de rendimiento de las rutinas *codelets*.

Aplicación de Metodología

- Rutina previamente instalada con cada una de las políticas de planificación (*eager*, *random*, *ws*, *lws* (default), *dm*, *dmda*)
- Selecciona mejor política de planificación, obteniendo el mejor valor de *nb* para cada tamaño de problema.

Parámetros de la Librería





Todos los Parámetros

n	*Politica*	*nb*	*CPU_Cores*	*GPUs*	*GFlops*
2000	dmda	192	10	{1,5}	169
4000	dmda	256	7	{1,5,0,2,3}	403
8000	dmda	320	6	{1,5,0,2,3,4}	719
12000	dmda	576	6	{1,5,0,2,3,4}	896
16000	dmda	576	6	{1,5,0,2,3,4}	1010
20000	ws	672	6	{1,5,0,2,3,4}	1051
24000	ws	672	6	{1,5,0,2,3,4}	1120
28000	ws	672	6	{1,5,0,2,3,4}	1150

Índice de Contenidos

Conclusiones

Tesis

Desarrollo de una metodología jerárquica para la auto-optimización de rutinas de álgebra lineal en plataformas heterogéneas.

Metodología Jerárquica

- Genérica: diferentes niveles de jerarquía hardware y software.
- Flexible: se adapta a sistemas computacionales existentes $\rightarrow$ uso eficiente de recursos computacionales del sistema.
- Instalación de rutinas por niveles, considerando en cada nivel los parámetros que afectan al rendimiento.
- Uso de diferentes técnicas para determinar el valor de los parámetros.
- Resultados satisfactorios con diferentes escenarios (sistemas y rutinas) y parámetros en cada nivel.

Conclusiones

Extensión de Funcionalidad

- Uso de herramientas software (OpenTuner) para obtención del valor de los AP en diferentes niveles de la jerarquía.
- Aplicación a rutinas de otras librerías (Chameleon). Obtención de prestaciones satisfactorias con buena selección del valor de los parámetros (algorítmicos, del sistema y propios de la librería).

Sistema Software

Implementa la funcionalidad de la metodología propuesta para la instalación de rutinas auto-optimizadas de forma jerárquica.

Disponible en:
`http://luna.inf.um.es/grupo_investigacion/software`

Manual de Usuario y de Desarrollador en Anexo.

Difusión de Resultados

Asistencia a Congresos

- 18th International Conference on Computational and Mathematical Methods in Science and Engineering (CMMSE 2018), Cádiz, 2018.
- XXIX Jornadas de Paralelismo (JP2018) de la Sociedad de Arquitectura y Tecnología de Computadores (SARTECO), Teruel, 2018.
- 19th International Conference on Computational and Mathematical Methods in Science and Engineering (CMMSE 2019), Cádiz, 2019.
- International Conference on Parallel Computing (ParCo 2019), Praga, República Checa, 10-13 Septiembre, 2019.
- SIAM Conference on Parallel Processing for Scientific Computing (PP20), Seattle, Washington, U.S., 12-15 Febrero, 2020.

Difusión de Resultados

Publicaciones

- Jesús Cámara, Javier Cuenca, Luis-Pedro García, Domingo Giménez. Empirical Modelling of Linear Algebra Shared-Memory Routines. *In Proceedings of the International Conference on Computational Science*, pages 110-119, 2013.
- Jesús Cámara, Javier Cuenca, Luis-Pedro García, Domingo Giménez. Auto-tuned nested parallelism: A way to reduce the execution time of scientific software in NUMA systems. *Parallel Computing*, 40(7):309-327, 2014.
- Jesús Cámara, Javier Cuenca, Luis-Pedro García, Domingo Giménez. Empirical Installation of Linear Algebra Shared-Memory Subroutines for Auto-Tuning. *International Journal of Parallel Programming*, 42(3):408-434, 2014.
- Jesús Cámara, Javier Cuenca, Domingo Giménez. **Integrating software and hardware hierarchies in an autotuning method for parallel routines in heterogeneous clusters**. *JoS*, 2020.

Trabajo Futuro

Extensión de Funcionalidad

- Nuevos parámetros, rutinas y librerías de álgebra lineal.
- Técnicas de modelado teórico del tiempo de ejecución de las rutinas.
- Soporte a optimización bi-objetivo, tanto a nivel de rendimiento como de consumo de energía.

Integración en Software Científico

- Como componente software en librería Chameleon para permitir obtención de modelos de rendimiento optimizados que puedan ser usados por el simulador SimGrid.
- Simuladores.
  - Actualmente trabajando en integración de metodología jerárquica en software para simulación de sistemas multicuerpo (cinemática computacional). Resultados preliminares presentados en SIAM PPSC20.

¡¡ GRACIAS !!

Directores:
Javier Cuenca y Domingo Giménez

Investigador:
Dr. Emmanuel Agullo

Tribunal:
Presidente: Dr. Leonel A. Pires Seabra de Sousa
Secretario: Dr. Pedro E. López de Teruel Alcolea
Vocal: Dr. Víctor M. García Mollá