Máster en Nuevas Tecnologías en Informática
Facultad de Informática
Universidad de Murcia

Optimización de Rutinas Multinivel de Álgebra Lineal en Sistemas Multicore

Autor:
Jesús Cámara Moreno

Directores:
A. Javier Cuenca Muñoz
Domingo Giménez Cánovas

Murcia, Septiembre de 2011

Índice de Contenidos

Introducción

- La mayoría de problemas científicos y de ingeniería llevan a cabo la computación mediante rutinas matriciales de álgebra lineal contenidas en librerías como BLAS, LAPACK, ScaLAPACK…

- La existencia de implementaciones *multithreading* eficientes de estas librerías (ATLAS, MKL) y con cierta capacidad de auto-adaptación (ATLAS) al entorno donde se estén utilizando, va a permitir obtener códigos paralelos eficientes.

- Tener acceso a estas librerías no significa que sean utilizadas eficientemente por científicos → necesidad de estudiar y evaluar su comportamiento en los sistemas donde son ejecutadas.

Introducción

- Motivación: vacío detectado en el desarrollo de librerías paralelas multinivel con capacidad de auto-optimización → diseñar prototipo de librería, que denominaremos 2L-BLAS.

- Este prototipo incluirá, inicialmente, la rutina de multiplicación de matrices utilizando 2 niveles de paralelismo: OpenMP+BLAS

- Objetivo: obtener una ejecución lo más eficiente posible → evaluar comportamiento de la rutina sobre distintos sistemas y aplicar un proceso de auto-optimización capaz de determinar el número de threads a establecer en cada nivel de paralelismo.

Introducción

- Entorno de Trabajo
  - **Ben**: alojado en el Centro de Supercomputación FPCMUR. Consta de un total de 128 cores.
  - **Pirineus**: alojado en el Centro de Supercomputación de Cataluña (CESCA). Consta de un total de 1344 cores (256 disponibles).
  - **Saturno**: multiprocesador alojado en el laboratorio de Computación Científica y Programación Paralela de la UM. Consta de un total de 24 cores.

Diseño de la Rutina dgemm2L

- Sintaxis:

```
dgemm2L(char transA, char transB, int m, int n, int k, double alpha, double *A,
         int lda, double *B, int ldb, double beta,double *C, int ldc,
         int thrOMP, int thrMKL)
```

- Implementada en el lenguaje de programación C.
- Definida siguiendo filosofía de rutinas de BLAS, en concreto, por la función `dgemm`:

```
dgemm(transA,transB,m,n,k,alpha,A,lda,B,ldb,beta,C,ldc)
```

- Añade 2 parámetros nuevos en la cabecera (`thrOMP`, `thrMKL`) para permitir especificar el número de threads a usar en cada nivel de paralelismo.

Diseño de la Rutina dgemm2L

- Codificada siguiendo el esquema de paralelismo anidado:

```
omp_set_nested(1);                    //Habilita paralelismo anidado
omp_set_num_threads(nthomp);          //Establece threads OpenMP
mkl_set_num_threads(nthmkl);          //Establece threads MKL

#pragma omp parallel
{
    obtener el tamaño y la posición inicial
    de la submatriz de A a ser multiplicada

    invocar a la rutina dgemm para multiplicar
    la submatriz de A por la matriz B
}
```

Evaluación de Prestaciones

- Estudio empírico del comportamiento de la rutina `dgemm2L` al variar el tamaño y forma de las matrices y el número de threads en cada nivel de paralelismo (OpenMP+MKL)

- Posibles Opciones:
  - Usar directamente paralelismo MKL con determinación dinámica de threads y sin paralelismo OpenMP.
  - Usar un número de threads igual al número de cores disponibles.

- Dependiendo del tamaño y forma de las matrices y del sistema computacional, se pueden obtener tiempos de ejecución menores usando los 2 niveles de paralelismo (OpenMP+MKL)

Evaluación de Prestaciones

- Experimentos realizados:
  - En el nivel de paralelismo MKL (1 thread OpenMP – Varios MKL)
  - En los 2 niveles de paralelismo (OpenMP+MKL)

- Tipo de experimentos:
  - Habilitando/Deshabilitando Selección Dinámica de Threads MKL.
  - Utilizando Matrices Cuadradas y Matrices Rectangulares.

Evaluación de Prestaciones
[MKL]

- Conclusiones:
  - Habilitar la selección dinámica de threads MKL no aporta mejoras significativas en el rendimiento de la rutina cuando aumenta el número de threads MKL y el tamaño de las matrices.
  - Usar matrices rectangulares ofrece un comportamiento similar al obtenido con matrices cuadradas.
  - El menor tiempo de ejecución no se alcanza al emplear un número de threads coincidente con el máximo número de cores.
  - Usar un único nivel de paralelismo no termina de ser una buena opción para obtener tiempos de ejecución cercanos al óptimo → empleo de 2 niveles de paralelismo (OpenMP+MKL)

Evaluación de Prestaciones
[OpenMP+MKL]

- Conclusiones:
  - Utilizar dos niveles de paralelismo deshabilita la selección dinámica de threads de la librería MKL, al menos en las versiones utilizadas en Ben, Saturno y Pirineus.
  - Utilizar matrices rectangulares no presenta diferencias significativas en el speed-up respecto al uso de matrices cuadradas → la mayoría de los experimentos se realizarán utilizando matrices cuadradas.
  - Se obtiene una mejora en el speed-up de la rutina respecto al empleo únicamente de paralelismo MKL.
  - Los mejores tiempos de ejecución se obtienen para combinaciones intermedias de threads OpenMP y MKL → establecer un proceso de auto-optimización que determine automáticamente el número de threads OpenMP+MKL a usar en cada nivel de paralelismo.

Evaluación de Prestaciones
[OpenMP+MKL]

n	Seq.	Max_Cores	Low. MKL	Low. OMP+MKL	Speed-Up (MKL/OMP+MKL)
Ben (96 cores)					
400	0.0223	0.0209	0.0033 (16)	0.0027 (4-4)	1.22
800	0.1573	0.0543	0.0154 (16)	0.0116 (10-3)	1.33
1000	0.3144	0.0759	0.0269 (22)	0.0181 (5-8)	1.49
2000	2.4626	0.3306	0.1294 (36)	0.0749 (10-6)	1.73
3000	8.2604	0.6640	0.3076 (40)	0.2131 (24-4)	1.44
4000	19.5511	1.3624	0.6387 (40)	0.4900 (32-2)	1.30
5000	38.1964	1.8791	1.1098 (48)	0.8964 (20-4)	1.24
Saturno (24 cores)					
400	0.0178	0.0025	0.0024 (21)	0.0021 (6-4)	1.19
600	0.0577	0.0067	0.0067 (24)	0.0064 (6-4)	1.05
800	0.1360	0.0399	0.0170 (21)	0.0146 (6-4)	1.17
1000	0.2498	0.0335	0.0318 (20)	0.0291 (8-3)	1.09
2000	1.9849	0.2257	0.2257 (24)	0.2151 (6-4)	1.05
3000	6.6704	0.7255	0.5205 (17)	0.5205 (1-17)	1.00
Pirineus (240 cores)					
750	0.0956	0.3139	0.0139 (24)	0.0134 (2-16)	1.04
1000	0.2199	0.4547	0.0322 (12)	0.0235 (2-16)	1.37
2000	1.6815	1.1569	0.4796 (16)	0.0797 (5-12)	6.01
3000	5.4696	1.2903	0.3955 (60)	0.2752 (4-15)	1.44
4000	12.9175	1.9495	0.5397 (60)	0.4670 (5-12)	1.16
5000	25.1401	2.7449	1.1149 (60)	0.8598 (10-9)	1.30

Técnicas de Auto-Optimización

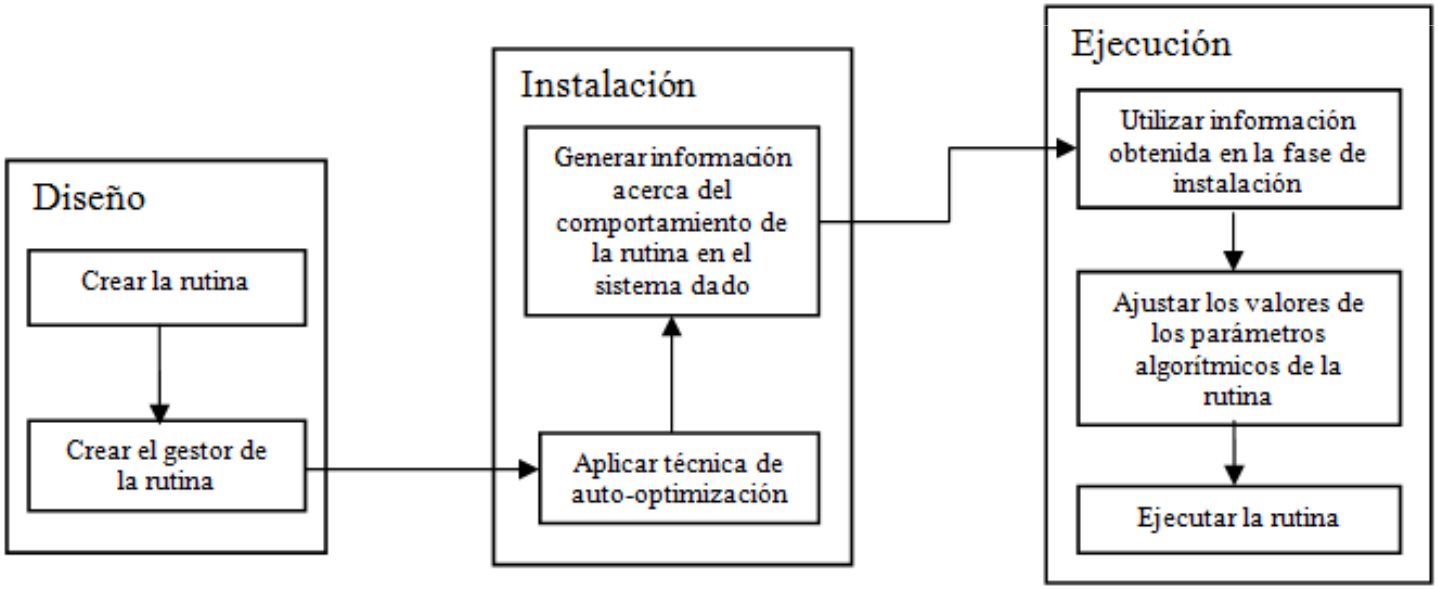
- Estudio de 3 técnicas de auto-optimización:
  - Auto-Optimización por Aproximación.
  - Auto-Optimización mediante Búsqueda Exhaustiva.
  - Auto-Optimización mediante Búsqueda Local con Incremento Variable.

- Objetivo: determinar la combinación más adecuada de threads OpenMP y MKL a establecer en cada nivel de paralelismo de la rutina `dgemm2L` con el fin de conseguir una ejecución lo más eficiente posible y con un tiempo de instalación mínimo.

Técnicas de Auto-Optimización

- Metodología de Diseño, Instalación y Ejecución de una rutina en un sistema con capacidad de ajuste automático:

- Tres Fases:



Técnicas de Auto-Optimización

- *Auto-Optimización por Aproximación:*
  - Sustituir parámetros thrOMP y thrMKL de la cabecera de la rutina `dgemm2L` por otro que indique el máximo número de cores a utilizar *(maxCores)*
  - Aplicar raíz cuadrada a *maxCores* para obtener los threads OMP y MKL a usar.

Ben:
maxCores = 90
thrOMP = 9, *thrMKL* = 10

Saturno:
maxCores = 20
thrOMP = 4, *thrMKL* = 5

n	Seq.	Max_Cores	Opt.	Auto-Opt.	Sp. (Opt/Auto-Opt)
(90) **Ben**					
400	0.0223	0.1203	0.0062 (18-5)	0.0099	0.6271
800	0.1573	0.1830	0.0168 (10-9)	0.0175	0.9600
1000	0.3144	0.1634	0.0255 (10-9)	0.0257	0.9895
2000	2.4626	0.2863	0.0847 (10-9)	0.0893	0.9482
3000	8.2604	0.6504	0.2645 (30-3)	0.2738	0.9660
4000	19.5511	1.3204	0.6092 (18-5)	0.6487	0.9392
5000	38.1964	1.9356	0.9939 (10-9)	1.1134	0.8926
(20) **Saturno**					
400	0.0178	0.0025	0.0024 (5-4)	0.0025	0.9501
800	0.1360	0.0171	0.0167 (5-4)	0.0174	0.9592
1000	0.2498	0.0489	0.0332 (4-5)	0.0332	1.0000
2000	1.9849	0.2521	0.2509 (5-4)	0.2519	0.9962
3000	6.6704	0.8369	0.8195 (5-4)	0.8345	0.9821

Técnicas de Auto-Optimización [MKL]

- *Auto-Optimización mediante Búsqueda Exhaustiva*:

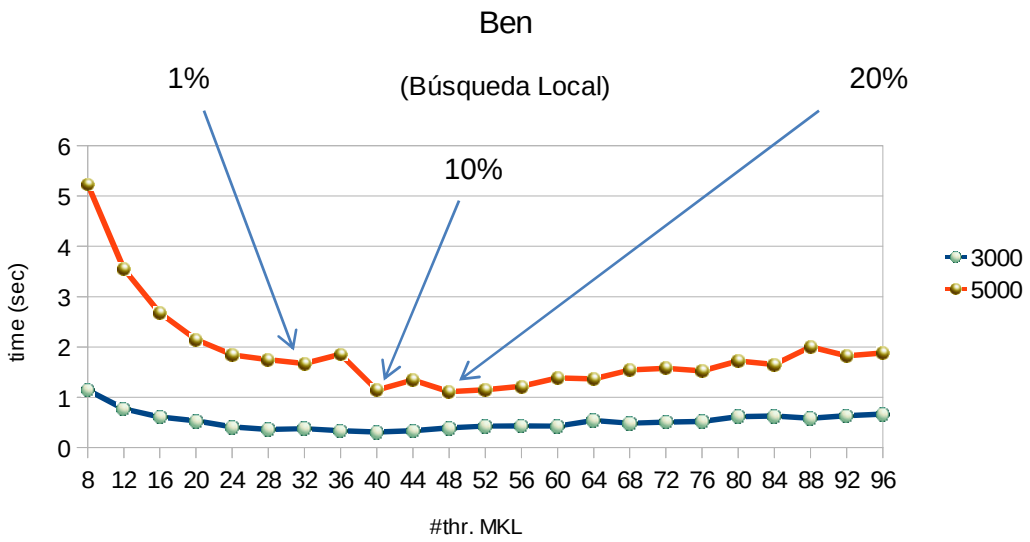
cjtoInst = {500,1000,3000,5000}
cjtoVal = {700, 2000, 4000}

Ben → 96 cores
Saturno → 24 cores
Pirineus → 240 cores

n	Opt.	Auto. Opt	Sp.	Tiempo Inst.
		Ben		
500	0.0056 (20)			1.40
700	0.0121 (24)	0.0142 (20)	0.85	
1000	0.0270 (20)			5.50
2000	0.1294 (36)	0.1790 (30)	0.72	
3000	0.3076 (40)			70.72
4000	0.6387 (40)	0.8121 (44)	0.79	
5000	1.1098 (48)			275.06
			TOTAL:	352.69
		Saturno		
500	0.0045 (24)			0.22
700	0.0099 (24)	0.0163 (22)	0.61	
1000	0.0318 (20)			1.43
2000	0.2257 (24)	0.2532 (22)	0.89	
3000	0.7255 (24)			26.44
4000	1.6461 (24)	1.9459 (20)	0.85	
5000	2.0901 (18)			77.67
			TOTAL:	105.77
		Pirineus		
500	0.0059 (24)			1.8
750	0.0139 (24)	0.0597 (16)	0.23	
1000	0.0322 (12)			2.74
2000	0.4796 (16)	0.7659 (32)	0.63	
3000	0.3955 (60)			24.61
4000	0.5397 (60)	0.5397 (60)	1.00	
5000	1.1149 (60)			81.41
			TOTAL:	110.56

Técnicas de Auto-Optimización
[MKL]

- *Búsqueda Local con Incremento Variable*
  - La precisión con la que se obtiene el número óptimo de threads MKL depende del porcentaje escogido. Con valores mayores se acerca más al tiempo óptimo, pues se realizan más ejecuciones, ignorando tiempos que correspondan a óptimos locales.



Técnicas de Auto-Optimización [MKL]

- *Búsqueda Local con Incremento Variable*:

cjtoInst = {500,1000,3000,5000}
cjtoVal = {700, 2000, 4000}

increm = 1%, 10%, 20%, 50%

Ben → 96 cores
Saturno → 24 cores
Pirineus → 240 cores

n	Opt.	1.00%	10.00%	20.00%	50.00%
Ben					
500	0.0056 (20)	*0.0056* (20)	*0.0056* (20)	*0.0056* (20)	*0.0056* (20)
700	0.0121 (24)	0.0142 (20)	0.0142 (20)	0.0142 (20)	0.0142 (20)
1000	0.0270 (20)	*0.0270* (20)	*0.0270* (20)	*0.0270* (20)	*0.0270* (20)
2000	0.1294 (36)	0.1398 (24)	0.1790 (30)	0.1790 (30)	0.1790 (30)
3000	0.3076 (40)	0.3621 (28)	*0.3076* (40)	*0.3076* (40)	*0.3076* (40)
4000	0.6387 (40)	1.0979 (30)	0.6387 (40)	0.8121 (44)	0.8121 (44)
5000	1.1098 (48)	1.6701 (32)	1.1424 (40)	*1.1098* (48)	*1.1098* (48)
Saturno					
500	0.0045 (24)	0.0135 (4)	0.0064 (12)	0.0056 (16)	*0.0045* (24)
700	0.0099 (24)	0.0196 (6)	0.0149 (16)	0.0157 (18)	0.0163 (22)
1000	0.0318 (20)	0.0411 (8)	*0.0318* (20)	*0.0318* (20)	*0.0318* (20)
2000	0.2257 (24)	0.2761 (8)	0.2532 (22)	0.2532 (22)	0.2532 (22)
3000	0.7255 (24)	0.9118 (8)	*0.7255* (24)	*0.7255* (24)	*0.7255* (24)
4000	1.6461 (24)	2.7364 (14)	1.6461 (24)	1.9210 (22)	1.9210 (22)
5000	2.0901 (18)	*2.0901* (18)	*2.0901* (18)	*2.0901* (18)	*2.0901* (18)
Pirineus					
500	0.0059 (24)	0.0162 (4)	0.0162 (4)	0.0162 (4)	0.0162 (4)
750	0.0139 (24)	0.0774 (8)	0.0774 (8)	0.0774 (8)	0.0774 (8)
1000	0.0322 (12)	*0.0322* (12)	*0.0322* (12)	*0.0322* (12)	*0.0322* (12)
2000	0.4796 (16)	0.4796 (16)	0.4796 (16)	0.7659 (32)	0.7659 (32)
3000	0.3955 (60)	0.8269 (16)	0.8269 (16)	*0.3955* (60)	*0.3955* (60)
4000	0.5397 (60)	1.6802 (16)	1.6802 (16)	0.5397 (60)	0.5397 (60)
5000	1.1149 (60)	2.0365 (16)	2.0365 (16)	*1.1149* (60)	*1.1149* (60)

Técnicas de Auto-Optimización
[OpenMP+MKL]

- *Auto-Optimización mediante Búsqueda Exhaustiva*:

cjtoInst = {500,1000,3000,5000}
cjtoVal = {700, 2000, 4000}

Ben → 96 cores
Saturno → 24 cores
Pirineus → 240 cores

n	Opt.	Auto. Opt	Sp.	Tiempo Inst.
		Ben		
500	0,0051 (23-1)			6,25
700	0,0102 (7-4)	0,0122 (16-2)	0,83	
1000	0,0177 (10-4)			19,38
2000	0,0795 (10-5)	0,0810 (17-3)	0,98	
3000	0,2191 (25-3)			228,36
4000	0,5088 (32-2)	0,6614 (22-3)	0,77	
5000	0,9207 (20-3)			902,47
			TOTAL:	1156,45
		Saturno		
500	0,0038 (6-4)			0,63
700	0,0098 (6-4)	0,0110 (7-3)	0,89	
1000	0,0291 (8-3)			4,13
2000	0,2151 (6-4)	0,2519 (4-5)	0,85	
3000	0,5205 (1-17)			81,61
4000	1,2580 (1-17)	1,9443 (4-5)	0,65	
5000	1,8915 (7-3)			267,21
			TOTAL:	353,58
		Pirineus		
500	0,0059 (1-24)			13,36
750	0,0134 (2-16)	0,0139 (1-24)	0,96	
1000	0,0235 (2-16)			19,56
2000	0,0797 (5-12)	0,0869 (3-20)	0,92	
3000	0,2752 (4-15)			191,15
4000	0,4670 (5-12)	0,5502 (6-10)	0,85	
5000	0,8598 (10-9)			452,42
			TOTAL:	676,49

Técnicas de Auto-Optimización
[OpenMP+MKL]

- *Búsqueda Local con Incremento Variable*:

cjtoInst = {500,1000,3000,5000}
cjtoVal = {700, 2000, 4000}

increm = 1%, 10%, 20%, 50%

Ben → 96 cores
Saturno → 24 cores
Pirineus → 240 cores

n	Opt.	1.00%	10.00%	20.00%	50.00%
Ben					
500	0.0050 (23-1)	0.0134 (1-4)	0.0051 (4-6)	0.0051 (4-6)	0.0055 (1-20)
700	0.0102 (7-4)	0.0148 (1-10)	0.0119 (5-6)	0.0119 (5-6)	0.0111 (1-19)
1000	0.0177 (10-4)	0.0297 (1-16)	0.0183 (6-7)	0.0183 (6-7)	0.0246 (1-19)
2000	0.0795 (10-5)	0.0984 (3-15)	0.0826 (6-8)	0.0826 (6-8)	0.0963 (3-16)
3000	0.2191 (25-3)	0.2303 (5-14)	0.2380 (6-10)	0.2380 (6-10)	0.2303 (5-14)
4000	0.5088 (32-2)	0.6291 (6-10)	0.6150 (7-8)	0.6150 (7-8)	1.0728 (6-10)
5000	0.9207 (20-3)	0.9612 (8-7)	0.9612 (8-7)	0.9612 (8-7)	0.9612 (8-7)
Saturno					
500	0.0038 (6-4)	0.0085 (2-2)	0.0085 (2-2)	0.0085 (2-2)	0.0042 (2-12)
700	0.0098 (6-4)	0.0227 (2-2)	0.0227 (2-2)	0.0227 (2-2)	0.0113 (2-12)
1000	0.0291 (8-3)	0.0325 (3-3)	0.0325 (3-3)	0.0325 (3-3)	0.0295 (2-12)
2000	0.2151 (6-4)	0.2604 (3-3)	0.2604 (3-3)	0.2604 (3-3)	0.2581 (2-10)
3000	0.5205 (1-17)	0.8338 (3-3)	0.8338 (3-3)	0.8338 (3-3)	0.7089 (3-8)
4000	1.2580 (1-17)	2.1135 (3-6)	2.1135 (3-6)	2.1135 (3-6)	1.9754 (3-7)
5000	1.8915 (7-3)	1.9567 (3-7)	1.9567 (3-7)	1.9567 (3-7)	1.9567 (3-7)
Pirineus					
500	0.0059 (1-24)	0.0075 (4-4)	0.0075 (4-4)	0.0075 (4-4)	0.0075 (4-4)
750	0.0134 (2-16)	0.0160 (4-4)	0.0251 (4-6)	0.0251 (4-6)	0.0251 (4-6)
1000	0.0235 (2-16)	0.0334 (4-3)	0.0264 (4-8)	0.0264 (4-8)	0.0264 (4-8)
2000	0.0797 (5-12)	0.2319 (4-8)	0.0813 (4-15)	0.0813 (4-15)	0.0813 (4-15)
3000	0.2752 (4-15)	0.2752 (4-15)	0.2752 (4-15)	0.2752 (4-15)	0.2752 (4-15)
4000	0.4670 (5-12)	0.4670 (5-12)	0.4670 (5-12)	0.4670 (5-12)	0.4670 (5-12)
5000	0.8598 (10-9)	0.8949 (5-18)	0.8949 (5-18)	0.8949 (5-18)	0.8949 (5-18)

MKL:

Ben					
	Exhaustivo	1.00%	10.00%	20.00%	50.00%
t_inst (seg):	352.69	9.5378	8.1567	13.0834	22.7303

Saturno					
	Exhaustivo	1.00%	10.00%	20.00%	50.00%
t_inst (seg):	105.77	30.6451	14.2651	25.0115	29.5082

Pirineus					
	Exhaustivo	1.00%	10.00%	20.00%	50.00%
t_inst (seg):	110.56	13.2863	14.2516	12.9708	17.1192

OpenMP + MKL:

Ben					
	Exhaustivo	1.00%	10.00%	20.00%	50.00%
t_inst (seg):	1156.45	38.81	33.94	46.74	125.48

Saturno					
	Exhaustivo	1.00%	10.00%	20.00%	50.00%
t_inst (seg):	353.58	36.83	36.83	39.74	44.13

Pirineus					
	Exhaustivo	1.00%	10.00%	20.00%	50.00%
t_inst (seg):	676.49	20.51	18.30	15.22	28.45

Conclusiones Generales

- Emplear varios niveles de paralelismo, unido a un proceso de auto-optimización, va a permitir reducir sustancialmente el tiempo de ejecución respecto a la ejecución secuencial o el empleo de un solo nivel de paralelismo → aumento de las prestaciones y del rendimiento de la aplicación.

- La metodología y técnicas empleadas son aplicables sobre otro tipo de rutinas y garantizan la obtención de resultados fiables independientemente del sistema utilizado.

Trabajo Futuro

- Estudiar la ganancia obtenida al utilizar la rutina dgemm2L en problemas reales donde se utilice la multiplicación de matrices.
- Ampliar el prototipo implementado con nuevas rutinas (LU, QR…)
- Refinar la implementación del algoritmo de búsqueda local utilizado en la técnica de auto-optimización con 2 niveles de paralelismo para evitar recorrer caminos que lleven a la obtención de tiempos de ejecución alejados del óptimo.
- Incluir un tercer nivel de paralelismo (MPI+OpenMP+BLAS), analizando su comportamiento en sistemas de gran dimensión, como clusters de computadores y sistemas heterogéneos.
- Desarrollar un prototipo que integre varios niveles de paralelismo y permita su uso en sistemas híbridos/heterogéneos compuestos por GPUs y multicores.
- Analizar la combinación de las técnicas empíricas de auto-optimización aquí estudiadas con técnicas basadas en el modelado de rutinas.

Muchas Gracias por su atención