Modelling and optimisation of scientific software for multicore platforms

Domingo Giménez

... and the list of collaborators within the presentation

Group page: http://www.um.es/pcgum

Presentations:
http://dis.um.es/%7Edomingo/investigacion.html

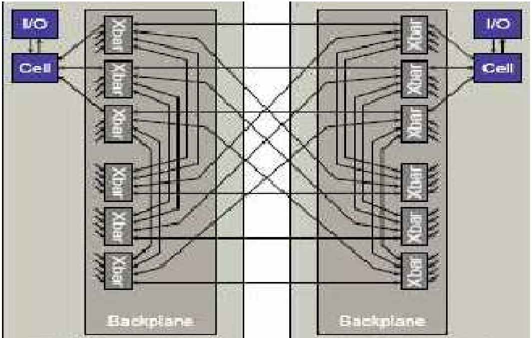
June 2010, Workshop Scheduling in Aussois

Contents

- Rosebud (Polytechnic Univ. of Valencia):
  cluster with 38 cores
  2 nodes single-processors, 2 nodes dual-processors, 2 nodes with 4 dual-core, 2 nodes with 2 dual-core, 2 nodes with 1 quad-core
- Hipatia (Polytechnic Univ. of Cartagena):
  cluster with 152 cores
  16 nodes with 2 quad-core, 2 nodes with 4 quad-core, 2 nodes with 2 dual-core
- Ben-Arabi (Supercomputing Centre of Murcia):
  Shared-memory + cluster: 944 cores
  Arabi: cluster of 102 nodes with 2 quad-core
  Ben: HP Superdome, cc-NUMA with 128 cores

Ben architecture

Hierarchical composition with crossbar interconnection.
Two basic components: the computers and two backplane crossbars.
Each computer has 4 dual-core Itanium-2 and a controller to connect the CPUs with the local memory and the crossbar commuters.
The maximum memory bandwidth in a computer is 17.1 GB/s and with the crossbar commuters 34.5 GB/s.
The access to the memory is non uniform and the user does not control where threads are assigned.



Scientific code optimisation

- Modelling scientific code
    - From basic routines...
    - ... to scientific codes
    - For multicore, clusters, supercomputers
- Installation tools and methodology
    - Using the previous models...
    - ... and empirical analysis for the particular routine and computational system
- Adaptation methodology:
    - With the model and the empirical study at installation time...
    - ... adapt the software to the entry and system conditions at running time

Scientific code optimisation

- Modelling scientific code
  - From basic routines...
  - ... to scientific codes
  - For multicore, clusters, supercomputers
- Installation tools and methodology
  - Using the previous models...
  - ... and empirical analysis for the particular routine and computational system
- Adaptation methodology:
  - With the model and the empirical study at installation time...
  - ... adapt the software to the entry and system conditions at running time

Scientific code optimisation

- Modelling scientific code
  - From basic routines...
  - ... to scientific codes
  - For multicore, clusters, supercomputers
- Installation tools and methodology
  - Using the previous models...
  - ... and empirical analysis for the particular routine and computational system
- Adaptation methodology:
  - With the model and the empirical study at installation time...
  - ... adapt the software to the entry and system conditions at running time

Regional meteorology simulations

Joint work with Sonia Jerez, Juan-Pedro Montávez, Regional Atmospheric Modelling Group, Univ. Murcia

Sonia Jerez, Juan-Pedro Montávez, Domingo Giménez, Optimizing the execution of a parallel meteorology simulation code, IEEE IPDPS, 10th Workshop on Parallel and Distributed Scientific and Engineering Computing, Rome, May 25-29, 2009

- MM5: parallel versions with OpenMP and MPI
- Optimise the use on multicore systems of the parallel codes

Regional meteorology simulations

Joint work with Sonia Jerez, Juan-Pedro Montávez, Regional Atmospheric Modelling Group, Univ. Murcia

Sonia Jerez, Juan-Pedro Montávez, Domingo Giménez, Optimizing the execution of a parallel meteorology simulation code, IEEE IPDPS, 10th Workshop on Parallel and Distributed Scientific and Engineering Computing, Rome, May 25-29, 2009

- MM5: parallel versions with OpenMP and MPI
- Optimise the use on multicore systems of the parallel codes

Regional meteorology simulations: modelling

After the simulation of a period of fixed length (*spin-up* period, T_s) the influence of the initial condition is discarded.
The value of T_s depends on each experiment.

- **Time parallelization**:
  Divide the period P in N_t subperiods and simulate each subperiod with the *spin-up* time T_s:

  $$T = \left(\frac{P}{N_t} + T_s \right) t$$

  where t is the cost of the simulation of a unity-length period
- **Spatial parallelization**: Using the PARALLEL CODE that divides the spatial domain, each portion is solved in a core.
  Use $N_p = N_x N_y$ cores for each simulation
  The total number of cores is $N = N_t N_p$
  The cost of a basic operation depends on the parameters: $t = f(N_t, N_x, N_y)$ and mesh configuration

Regional meteorology simulations: modelling

After the simulation of a period of fixed length (*spin-up* period, T_s) the influence of the initial condition is discarded.
The value of T_s depends on each experiment.

- **Time parallelization**:
  Divide the period P in N_t subperiods and simulate each subperiod with the *spin-up* time T_s:

$$T = \left(\frac{P}{N_t} + T_s \right) t$$

  where t is the cost of the simulation of a unity-length period
- **Spatial parallelization**: Using the PARALLEL CODE that divides the spatial domain, each portion is solved in a core.
  Use $N_p = N_x N_y$ cores for each simulation
  The total number of cores is $N = N_t N_p$
  The cost of a basic operation depends on the parameters: $t = f(N_t, N_x, N_y)$ and mesh configuration

Regional meteorology simulations: modelling

After the simulation of a period of fixed length (*spin-up* period, T_s) the influence of the initial condition is discarded.
The value of T_s depends on each experiment.

- **Time parallelization**:
  Divide the period P in N_t subperiods and simulate each subperiod with the *spin-up* time T_s:

$$T = \left(\frac{P}{N_t} + T_s \right) t$$

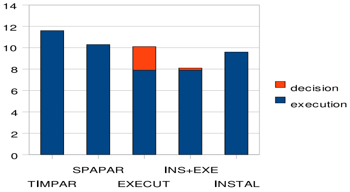
  where t is the cost of the simulation of a unity-length period
- **Spatial parallelization**: Using the PARALLEL CODE that divides the spatial domain, each portion is solved in a core.
  Use $N_p = N_x N_y$ cores for each simulation
  The total number of cores is $N = N_t N_p$
  The cost of a basic operation depends on the parameters:
  $t = f\left(N_t, N_x, N_y\right)$ and mesh configuration

- Installation:
  - A short period of time is simulated for all the possible combinations of N_t with N_p
  - with a limit: $N_t N_p \leq 2N$
  - for some trial domains
  - and different mesh shapes: combinations of N_x and N_y
- Execution:
  - Select the values of N_t, N_x and N_y
  - tacking into consideration the size and characteristics of the problem to be solved
  - with the values $t = f(N_t, N_x, N_y)$ estimated at installation time for domains close to the current domain
  - to update the information generated at installation time:

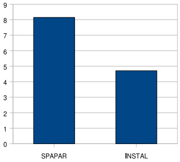
- Installation:
  - A short period of time is simulated for all the possible combinations of N_t with N_p
  - with a limit: $N_t N_p \leq 2N$
  - for some trial domains
  - and different mesh shapes: combinations of N_x and N_y
- Execution:
  - Select the values of N_t, N_x and N_y
  - tacking into consideration the size and characteristics of the problem to be solved
  - with the values $t = f\left(N_t, N_x, N_y\right)$ estimated at installation time for domains close to the current domain
  - to update the information generated at installation time:

Regional meteorology simulations: results

- DEFAUL: uses default parameters
- INSTAL: with installation information selects the values which gives lowest modelled time
- INS+EXE: repeats the experiments for the current problem for the parameter combinations which provide lowest modelled time
- EXECUT: repeats installation running for the current domain, and selects the parameters which give the lowest estimated time



rayo



hipatia

Reduction between 25 % and 40 % of the execution time

Hydrodynamic simulations

Joint work with Francisco López-Castejón, Oceanography Group, Polytechnic Univ. of Cartagena

Francisco López-Castejón, Domingo Giménez, Auto-optimisation on parallel hydrodynamic codes: an example of COHERENS with OpenMP for multicore, XVIII International Conference on Computational Methods in Water Resources, Barcelona, June 21-24, 2010

- Easy parallelisation and optimisation of COHERENS
- parallelize each loop separately
- with a different number of threads for each loop
- select the number of threads in each loop
  - with information obtained at installation time
  - and adaptation in the initial iterations

Simultaneous Equation Models

Joint work with José-Juan López-Espín, Univ. Miguel Hernández of Elche, Antonio M. Vidal, Polytechnic Univ. Valencia

José J. López-Espín, Domingo Giménez, Solution of Simultaneous Equations Models in high performance systems, International Congress on Computational and Applied Mathematics, Leuven, Belgium, July 5-9, 2010

- Use of matrix decompositions to obtain a number of algorithms with low execution time
- Basic operations: QR decomposition, matrix multiplications, Givens rotations
- Two types of parallelism: in the basic operations, and OpenMP parallelism in the computation of different equations
- Model of the execution time to decide the algorithm to use for an entry and system
- Estimation at installation time of the values of the parameters in the models
- Include two-level parallelism

Simultaneous Equation Models

Joint work with José-Juan López-Espín, Univ. Miguel Hernández of Elche, Antonio M. Vidal, Polytechnic Univ. Valencia

José J. López-Espín, Domingo Giménez, Solution of Simultaneous Equations Models in high performance systems, International Congress on Computational and Applied Mathematics, Leuven, Belgium, July 5-9, 2010

- Use of matrix decompositions to obtain a number of algorithms with low execution time
- Basic operations: QR decomposition, matrix multiplications, Givens rotations
- Two types of parallelism: in the basic operations, and OpenMP parallelism in the computation of different equations
- Model of the execution time to decide the algorithm to use for an entry and system
- Estimation at installation time of the values of the parameters in the models
- Include two-level parallelism

Parameterised shared-memory metaheuristics

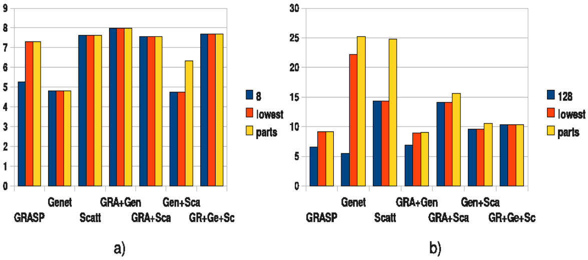
Joint work with José-Juan López-Espín, Univ. Miguel Hernández of Elche, Francisco Almeida, Univ. of La Laguna

Jose-Juan López-Espín, Francisco Almeida, Domingo Giménez, A parameterised shared-memory scheme for parameterised metaheuristics, 10h International Conference on Computational and Mathematical Methods in Science and Engineering, Minisymposium on High Performance Computing, Almería, June 26-30, 2010

- Parameterised metaheuristic scheme
  facilitates development and tuning of metaheuristics and hybridation/combination of metaheuristics
- Unified parallel shared-memory scheme for metaheuristics
  facilitates development of parallel metaheuristics or of their hybridation/combination
- Parameterised parallel shared-memory scheme for metaheuristics
  facilitates optimisation of parallel metaheuristics

Parameterised shared-memory metaheuristics: results

- Applied to obtaining satisfactory Simultaneous Equation Models given a set of values of variables
- Metaheuristics: GRASP, genetic, scatter search, GRASP+genet., GRASP+SS, Gent.+SS, GRASP+genet.+SS
- With different number of threads in each function and two-level parallelism better results



Arabi Ben

Other scientific problems

- Integral equations to study breaking of microstrip components
  Joint work with José-Ginés Picón, Supercomputing Centre Murcia, and Alejandro Álvarez and Fernando D. Quesada, Computational Electromagnetism Group Univ. Polytechnic of Cartagena
  Parallelise and optimise code, with nested parallelism and basic linear algebra routines (`zgemv` and `zgemm`)
- Bayesian simulations
  Joint work with Manuel Quesada, and Asunción Martínez-Mayoral and Javier Socuellamos, Univ. Miguel Hernández
  Web application to study bayesian distributions, to be installed on different platforms and with parallelism hidden to the user
- Possible collaboration with a company: design of bridges, with metaheuristics and parallelism, in supercomputer BenArabi

Modelling basic routines

Joint work with Javier Cuenca, Computer Architecture Department, Univ. of Murcia,

Luis-Pedro García, Polytechnic Univ. of Cartagena

- The goal:
  on multicore systems, with OpenMP,
  to model routines of high level
  by using information obtained from routines of low level
- Basic work:
  threads generation
  loop work distribution
  synchronisation
- Higher level routines:
  matrix-vector multiplication
  Jacobi iteration
  matrix-matrix multiplciation
  Strassen multiplication

Modelling basic routines

Joint work with Javier Cuenca, Computer Architecture Department, Univ. of Murcia,
Luis-Pedro García, Polytechnic Univ. of Cartagena

- The goal:
  on multicore systems, with OpenMP,
  to model routines of high level
  by using information obtained from routines of low level
- Basic work:
  threads generation
  loop work distribution
  synchronisation
- Higher level routines:
  matrix-vector multiplication
  Jacobi iteration
  matrix-matrix multiplciation
  Strassen multiplication

Modelling basic routines

Joint work with Javier Cuenca, Computer Architecture Department, Univ. of Murcia, Luis-Pedro García, Polytechnic Univ. of Cartagena

- The goal:
  on multicore systems, with OpenMP,
  to model routines of high level
  by using information obtained from routines of low level
- Basic work:
  threads generation
  loop work distribution
  synchronisation
- Higher level routines:
  matrix-vector multiplication
  Jacobi iteration
  matrix-matrix multiplciation
  Strassen multiplication

Modelling: test routines

- **R-generate**
  Creates a series of threads with a fixed quantity of work to do per thread
  To compare the time of creating and managing threads
- **R-pfor**
  A simple for loop where there is a significant work inside each iteration
  To compare the time of distributing dynamically a set of homogeneous tasks
- **R-barriers**
  A barrier primitive set after a parallel working area
  To compare the times to perform a global synchronisation of all the threads

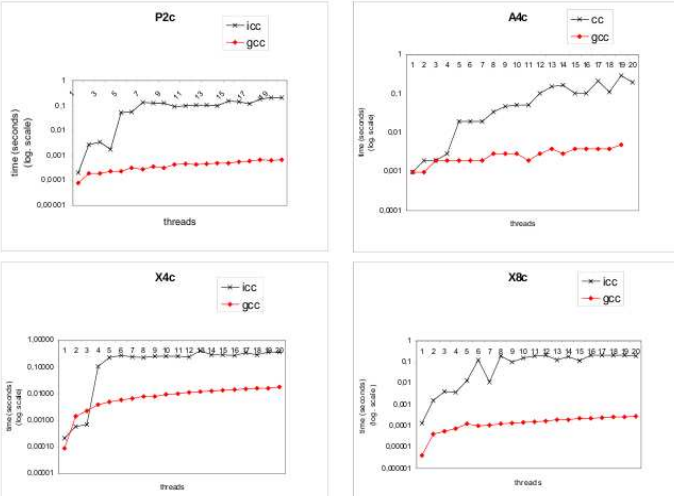
Modelling: systems

- **P2c** (a laptop)
  Intel Pentium, 2.8 GHz, with 2 cores.
  Compilers: icc 10.1 and gcc 4.3.2.
- **A4c**
  Alpha EV68CB, 1 GHz, with 4 cores.
  Compilers: cc 6.3 and gcc 4.3.
- **X4c**
  Intel Xeon, 3 GHz, with 4 cores.
  Compilers: icc 10.1 and gcc 4.2.3.
- **X8c** (a node of Hipatia)
  Intel Xeon, 2 GHz, with 8 cores.
  Compilers: icc 10.1 and gcc 3.4.6

Modelling: R-generate

threads ≤ # cores: $T_{R-generate} = PT_{gen} + NT_{work}$

threads > # cores: $T_{R-generate} = PT_{gen} + NT_{work} \frac{P}{C}\left(1 + \frac{T_{swap}}{T_{cpu}}\right)$



Modelling higher level routines: Jacobi

- Estimation of the parameters:

	P2c		X4c		A4c		X8c	
	icc	gcc	icc	gcc	cc	gcc	icc	gcc
$T_{gen}(\mu sec)$	75	25	75	25	75	25	75	25
$T_{work}(nsec)$	2	2	4	7	3	10	1.5	1.5
T_{swap}/T_{cpu}	2	1.5	15	0.8	15	1.8	1	0.4

- Substitution of estimated values of the parameters in the model of the routine:
  $\#$ threads $\leq \#$ cores:

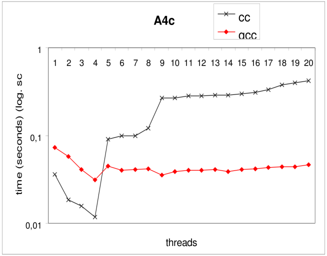
$$T_{Jacobi} = PT_{gen} + 11\frac{n^2}{P}T_{work}$$

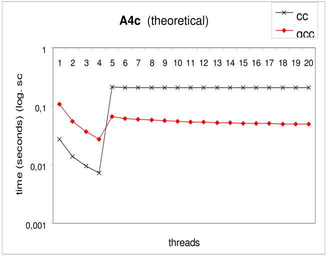
  $\#$ threads $> \#$ cores:

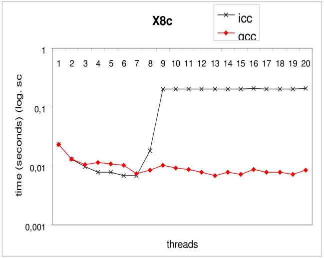
$$T_{Jacobi} = PT_{gen} + 11\frac{n^2}{C}T_{work}\left(1 + \frac{T_{swap}}{T_{cpu}}\right)$$

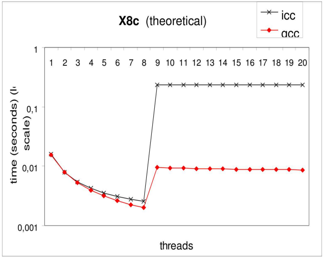
- Decision of the number of threads and compiler to use in the solution of the problem.

Modelling: Jacobi, results









Modelling: Strassen multiplication

threads ≤ # cores:

$$T_{Strassen} = PT_{gen} + \frac{7}{4}\frac{n^3}{P}T_{mult} + \frac{9}{2}n^2T_{add}$$

$$T_{Strassen} = PT_{gen} + \frac{49}{32}\frac{n^3}{P}T_{mult} + \frac{63}{8}\frac{n^2}{P_1}T_{add} + \frac{9}{2}n^2T_{add}$$

threads > # cores:

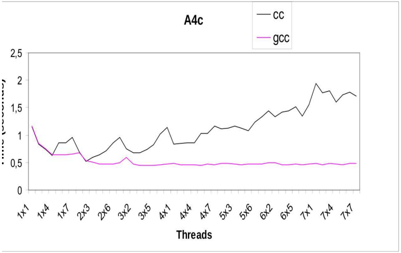
$$T_{Strassen} = PT_{gen} + \frac{7}{4}\frac{n^3}{P}T_{mult}\left(1 + \frac{T_{swap}}{T_{cpu}}\right) + \frac{9}{2}n^2T_{add}$$

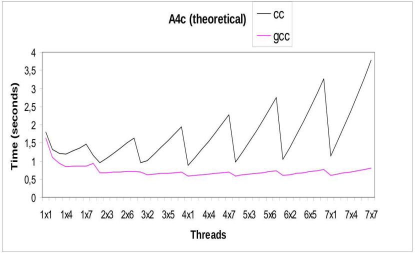
$$T_{Strassen} = PT_{gen} + \frac{49}{32}\frac{n^3}{C}T_{mult}\left(1 + \frac{T_{swap}}{T_{cpu}}\right) + \frac{63}{8}\frac{n^2}{\text{mín}\{P_1, C\}}T_{add}\left(1 + \frac{T_{swap}}{T_{cpu}}\right) + \frac{9}{2}n^2T_{add}$$

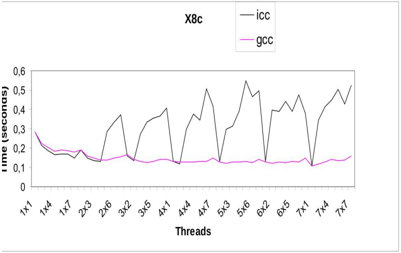
Modelling higher level routines: Strassen, SP values

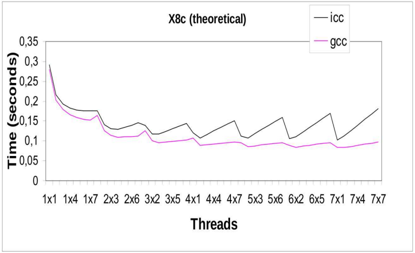
	P2c		X4c		A4c		X8c	
	icc	gcc	icc	gcc	cc	gcc	icc	gcc
$T_{gen}(\mu sec)$	75	25	75	25	75	25	75	25
T_{swap}/T_{cpu}	2+ 0.01P	7- 0.01P	0.9+ 0.3P	0.9+ 0.01P	0.8+ 0.2P	0.8+ 0.02P	6+ 0.05P	0.5+ 0.01P
$T_{add}(\mu sec)$	20+ 0.05P	20	23+ 0.3P	30- 0.3P	40+ P	40- 0.1P	10	10
$T_{mult}(\rho sec)$	400+ 100P	400+ 0.1P	140+ 10P	140- P	60 60	60- 0.5P	100	100

Modelling: Strassen, results









Modelling higher routines: Strassen, results

Problem size 1000

Combination giving the best results:

	P2c	X4c	A4c	X8c
compiler	gcc	gcc	gcc	gcc
# thr. level 1	7	4	4	7
# thr. level 2	7	1	1	2

Execution time for different values of the parameters:

	P2c	X4c	A4c	X8c
PCE	1.19	0.50	0.49	0.16
ORA	1.17	0.49	0.45	0.11
HW	1.37	0.55	0.65	0.12
SW	1.22	1.31	1.20	0.32

Matrix multiplication on platforms composed of multicore

The goal:

- To identify the shape matrix multiplication has in a multicore as a function of the problem size and the number of threads, to decide the number of threads to use to obtain the lowest execution time
- To use this information to develop two-level (OpenMP+BLAS) versions of the multiplication, and select the number of threads in each level
- To use this information to develop three-level (MPI+OpenMP+BLAS) versions, and select the number of processes and threads in each level
- To use this information to develop heterogeneous/distributed three-level (MPI+OpenMP+BLAS) versions, and select the number of processes and its distribution or the data partition, and in each processor the number of threads in each level

Matrix multiplication on platforms composed of multicore

The goal:

- To identify the shape matrix multiplication has in a multicore as a function of the problem size and the number of threads, to decide the number of threads to use to obtain the lowest execution time
- To use this information to develop two-level (OpenMP+BLAS) versions of the multiplication, and select the number of threads in each level
- To use this information to develop three-level (MPI+OpenMP+BLAS) versions, and select the number of processes and threads in each level
- To use this information to develop heterogeneous/distributed three-level (MPI+OpenMP+BLAS) versions, and select the number of processes and its distribution or the data partition, and in each processor the number of threads in each level

Matrix multiplication on platforms composed of multicore

The goal:

- To identify the shape matrix multiplication has in a multicore as a function of the problem size and the number of threads, to decide the number of threads to use to obtain the lowest execution time
- To use this information to develop two-level (OpenMP+BLAS) versions of the multiplication, and select the number of threads in each level
- To use this information to develop three-level (MPI+OpenMP+BLAS) versions, and select the number of processes and threads in each level
- To use this information to develop heterogeneous/distributed three-level (MPI+OpenMP+BLAS) versions, and select the number of processes and its distribution or the data partition, and in each processor the number of threads in each level

Matrix multiplication on platforms composed of multicore

The goal:

- To identify the shape matrix multiplication has in a multicore as a function of the problem size and the number of threads, to decide the number of threads to use to obtain the lowest execution time
- To use this information to develop two-level (OpenMP+BLAS) versions of the multiplication, and select the number of threads in each level
- To use this information to develop three-level (MPI+OpenMP+BLAS) versions, and select the number of processes and threads in each level
- To use this information to develop heterogeneous/distributed three-level (MPI+OpenMP+BLAS) versions, and select the number of processes and its distribution or the data partition, and in each processor the number of threads in each level

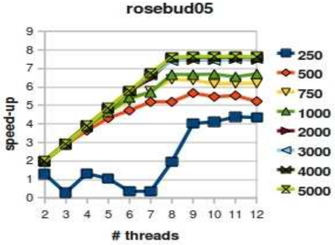
Systems, basic components

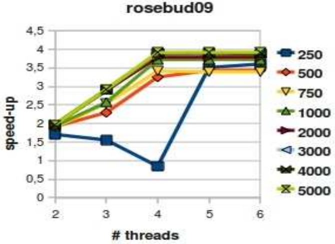
name	architecture	icc	MKL
rosebud05	4 Itanium dual-core 8 cores	11.1	10.2
rosebud09	1 AMD quad-core 4 cores	11.1	10.2
hipatia8	2 Xeon E5462 quad-core 8 cores	10.1	10.0
hipatia16	4 Xeon X7350 quad-core 16 cores	10.1	10.0
arabi	2 Xeon L5450 quad-core 8 cores	11.1	10.2
ben	HP Integrity Superdome 128 cores	11.1	10.2

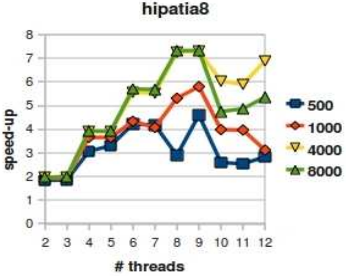
Using MKL

- The library is multithreaded.
- Number of threads estabished with the environment variable MKL_NUM_THREADS or in the program with the function mkl_set_num_threads.
- Dynamic parallelism is enabled with MKL_DYNAMIC=true or mkl_set_dynamic(1). The number of threads to use in dgemm is decided by the system, and is less or equal to that established.
- To enforce the utilisation of the number of threads, dynamic parallelism is turned off with MKL_DYNAMIC=false or mkl_set_dynamic(0).

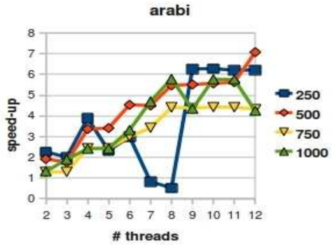
MKL, results

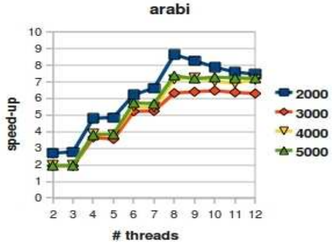

rosebud05
speed-up
threads
250
500
750
1000
2000
3000
4000
5000

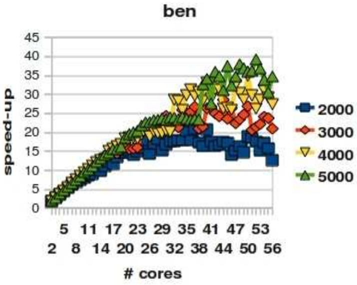

rosebud09
speed-up
threads
250
500
750
1000
2000
3000
4000
5000


hipatia8
speed-up
threads
500
1000
4000
8000

MKL, results


arabi
speed-up
threads
2 3 4 5 6 7 8 9 10 11 12
0 1 2 3 4 5 6 7 8
250
500
750
1000


arabi
speed-up
threads
2 3 4 5 6 7 8 9 10 11 12
0 1 2 3 4 5 6 7 8 9 10
2000
3000
4000
5000


ben
speed-up
cores
5 11 17 23 29 35 41 47 53
2 8 14 20 26 32 38 44 50 56
0 5 10 15 20 25 30 35 40 45
2000
3000
4000
5000

MKL, results

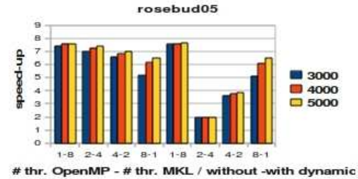
size	Seq.	Max.	Low.
rosebud05			
250	0.0081	0.0042	0.0019 (11)
rosebud09			
250	0.0042	0.0050	0.0012 (5)
hipatia8			
250	0.0035	0.0021	0.0011 (7)
500	0.026	0.0088	0.0056 (9)
750	0.087	0.021	0.017 (9)
arabi			
250	0.0080	0.0015	0.0013 (9)
500	0.034	0.063	0.0049 (12)
ben			
250	0.021	0.017	0.0014 (10)
500	0.042	0.033	0.0044 (19)
750	0.14	0.063	0.010 (22)
1000	0.32	0.094	0.019 (27)
2000	2.6	0.39	0.12 (37)
3000	8.6	0.82	0.30 (44)
4000	20	1.4	0.59 (50)
5000	40	2.1	1.0 (48)

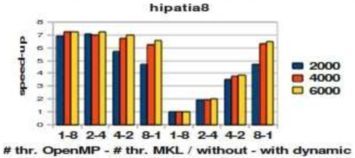
Two-level parallelism

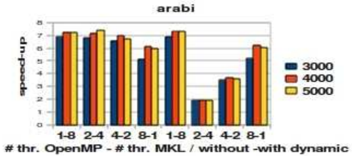
It is possible to use two-level parallelism: OpenMP + MKL.
The rows of a matrix are distributed to a set of OpenMP threads (*nthomp*).
A number of threads is established for MKL (*nthmkl*).
Nested parallelism must be allowed, with `OMP_NESTED=true` or `omp_set_nested(1)`.

```
omp_set_nested(1);
omp_set_num_threads(nthomp);
mkl_set_dynamic(0);
mkl_set_num_threads(nthmkl);
#pragma omp parallel
    obtain size and initial position of the submatrix of A to be
        multiplied
    call dgemm to multiply this submatrix by matrix B
```

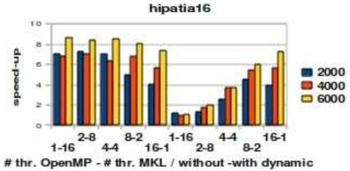
Two-level parallelism, results

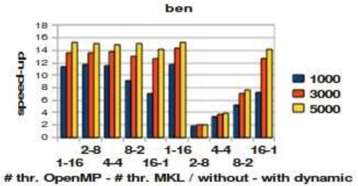




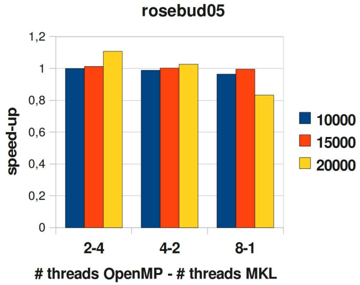


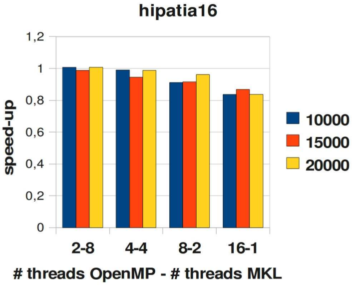
hipatia16

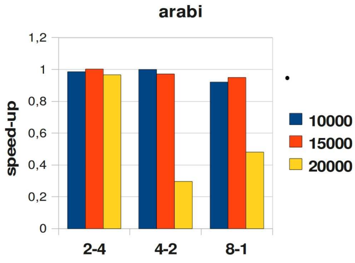


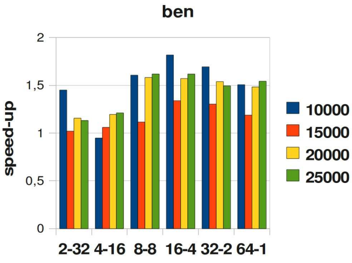


Two-level parallelism, results









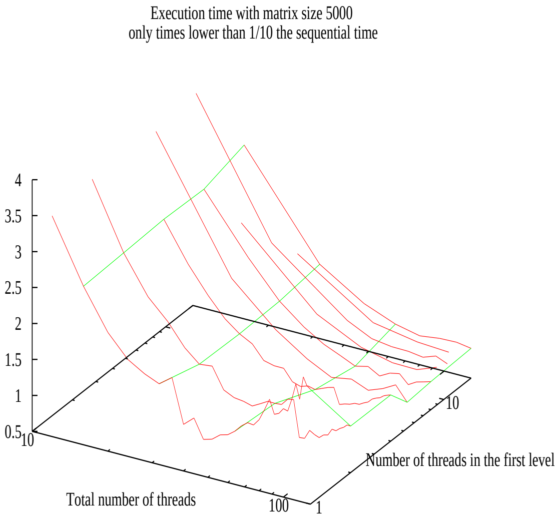
Two-level parallelism, conclusions

- In Hipatia (MKL version 10.0) the nested parallelism seems to disable the dynamic selection of threads.
- In the other systems, with dynamic assignation the number of MKL threads seems to be one when more than one OpenMP threads are running.
- When the number of MKL threads is established in the program bigger speed-ups are obtained.
- Normally the use of only one OpenMP thread is preferable.
- Only in Ben to use a higher number of OpenMP threads is a good option. Speed-ups between 1.2 and 1.8 are obtained with 16 OpenMP and 4 MKL threads.

Two-level parallelism, results

size	MKL	2-levels	Sp.
250	0.0014 (10)	0.0014 (1-10)	1.0
500	0.0044 (19)	0.0043 (4-11)	1.0
750	0.010 (22)	0.0095 (4-11)	1.1
1000	0.019 (27)	0.015 (4-10)	1.3
2000	0.12 (37)	0.072 (4-16)	1.6
3000	0.30 (44)	0.18 (4-24)	1.7
4000	0.59 (50)	0.41 (5-16)	1.4
5000	1.0 (48)	0.76 (6-20)	1.3
10000	10 (64)	5.0 (32-4)	2.0
15000	25 (64)	12 (32-4)	2.1
20000	65 (64)	22 (16-8)	3.0
25000	130 (64)	44 (16-8)	3.0

Two-level parallelism, surface shape



Matrix multiplication: research lines

- Development of a 2lBLAS prototype,
  and application to scientific problems
- Simple MPI+OpenMP+MKL version
  Experiments in large shared-memory (ben), large clusters (arabi), and heterogeneous (rosebud, ben+arabi)
- ScaLAPACK style MPI+OpenMP+MKL version
  Determine number of processors, and OpenMP and MKL threads
  From the model and empirical analysis or with adaptive algorithm
  In heterogeneous platform the number of processes per processor
- Heterogeneous ScaLAPACK style MPI+OpenMP+MKL version
  Determine volume of data for each processors, and OpenMP and MKL threads
  From the model and empirical analysis or with adaptive algorithm