Modelling the execution time of hybrid-parallelism scientific codes

Domingo Giménez
Departamento de Informática y Sistemas, University of Murcia, Spain

ComplexHPC network meeting, Timisoara, January 25, 2012

Outline

Modelling

- What: the execution time of parallel routines
- Why: to accurately predict the execution time and decide how to apply the routine, depending on the system and the problem
- How: parameterized routines and models, with theoretical or empirical estimation of the parameters of the system and selection of the routine parameters at execution time

Modelling

- What: the execution time of parallel routines
- Why: to accurately predict the execution time and decide how to apply the routine, depending on the system and the problem
- How: parameterized routines and models, with theoretical or empirical estimation of the parameters of the system and selection of the routine parameters at execution time

Modelling

- What: the execution time of parallel routines
- Why: to accurately predict the execution time and decide how to apply the routine, depending on the system and the problem
- How: parameterized routines and models, with theoretical or empirical estimation of the parameters of the system and selection of the routine parameters at execution time

Hybrid parallelism

Routines combining different sources of parallelism:

- 2-level parallelism with OpenMP
- OpenMP+BLAS parallelism
- CPU+GPU parallelism
- There are possible extensions:
  MPI+2lOpenMP+BLAS+MultiGPU...

Applications

In the Scientific Computing and Parallel Programming (SCPP) group at the University of Murcia we work on parallel computing applications and modelling and auto-tuning of parallel routines (http://dis.um.es/~domingo/investigacion.html)

In this presentation we summarize our on-going work in applications with hybrid-parallelism routines:

- Linear algebra
- Metaheuristics
- CPU+GPU

Applications: Linear algebra

- Basic routines: matrix multiplication, factorizations
- with OpenMP+BLAS parallelism
- in large NUMA systems
- to be used in large computational problems (electromagnetism, statistic models...)
- Collaboration with other members of the SCPP group:
  Jesús Cámara
  Javier Cuenca
  Luis-Pedro García (Polytechnic University of Cartagena)

Applications: Metaheuristics

- Parameterized scheme of metaheuristics (multiple metaheuristics)
- with independent parallelization of the functions in the scheme
- and parallelism parameters.
- With 2-level OpenMP parallelism.
- Applied to:
  - Simultaneous Equations Models, p-hub problem, tasks-to-processors (Javier Cuenca; Jose J. López-Espín, University Miguel Hernández; Francisco Almeida, University of La Laguna; Melquiades Pérez-Pérez, University of Gran Canaria)
  - Electrical consumption in exploitation of wells (José-Matías Cutillas-Lozano; Luis-Gabino Cutillas-Lozano, Municipalized water of Alicante)

Applications: CPU+GPU

- Combination of OpenMP and GPU or MultiGPU parallelism
- preliminary analysis.
- How to model?
- Scientific problems:
  - Green functions in Electromagnetism (Carlos Pérez-Alcaraz; Alejandro Álvarez-Melcón, Fernando D. Quesada, Polytechnic University of Cartagena)
  - Simultaneous Equations Models (Jose J. López-Espín; Carla Ramírez, Antonio M. Vidal, Polytechnic University of Valencia)

General ideas

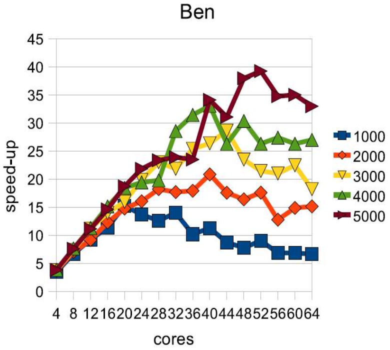
- Scientific and engineering problems solved with large parallel systems: NUMA with cores sharing a hierarchical memory
- Kernel of the computation: BLAS multithread
  Degradation in the performance when the system size increases
- Our goal:
  Nested parallelism: OpenMP+BLAS
  Model of the execution time and auto-tuning methodology
- Experiments with matrix multiplication

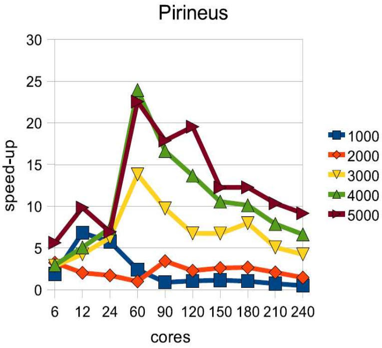
Computational systems

- `Ben`:
  Part of `Ben-Arabí` of the Supercomputing Center of Murcia.
  NUMA system with 128 cores (16 nodes, each with four CPUs dual core Itanium-2).
  Hierarchical composition with crossbar interconnection.
  The maximum memory bandwidth in a node is 17.1 GB/s and with the crossbar commuters 34.5 GB/s.
  Four different costs in the access to memory.
- `Pirineus`:
  In the Centre de Supercomputacio de Catalunya.
  SGI Altix UV 1000, with 1344 cores (224 Intel Xeon six-core serie 7500)
  An interconnection NUMAlink 5 in a paired node 2D torus.
- `Saturno`:
  In the laboratory of the SCPP group.
  24 cores: four nodes hexacore.

BLAS multithread

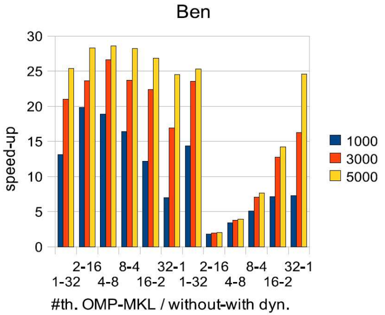
- A multithread version of the MKL `dgemm` routine
- The optimum number of threads changes from one platform to another
- A number of threads equal to that of the available cores is not a good option

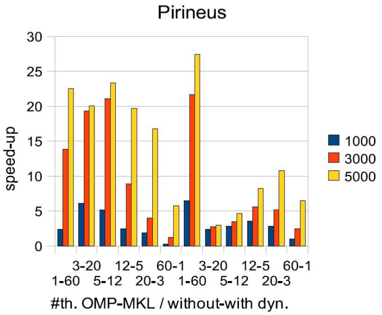




OpenMP+BLAS parallelism

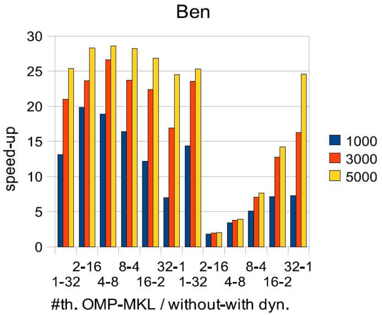
- Dynamic selection of threads: number of MKL threads used is just one
- No Dynamic Selection of threads: the highest speed-up by combining OpenMP and MKL parallelism

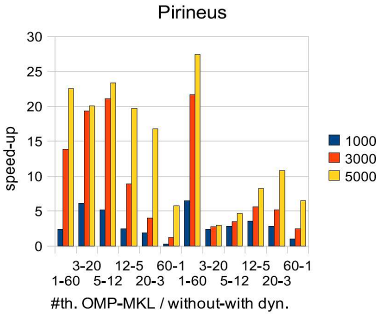




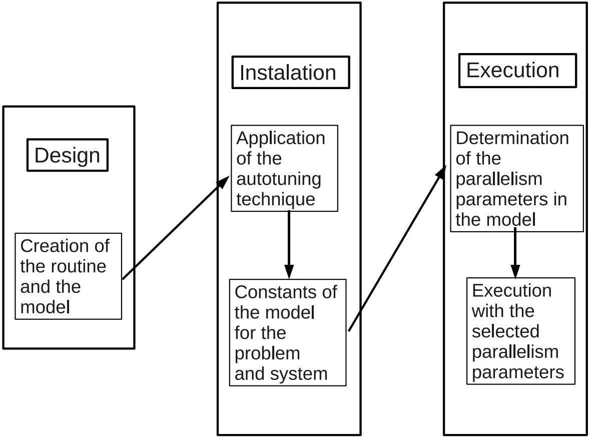
OpenMP+BLAS parallelism

- Dynamic selection of threads: number of MKL threads used is just one
- No Dynamic Selection of threads: the highest speed-up by combining OpenMP and MKL parallelism





Auto-tuning methodology



Design phase: MKL 1-level

Model: $t_{dgemm} = \frac{2n^3}{p} k_{dgemm}$

$k_{dgemm} = \alpha k_{dgemm\_NUMA}(p) + (1-\alpha) k_{dgemm\_M_1}$

$k_{dgemm\_M_1}$: when data are in the memory closest to the core

k_{dgemm_NUMA}: when data are in any level in the memory

α: directly proportional to the use by each thread of data assigned to the other threads; inversely proportional to data reuse degree:

$$\alpha = min\{1, \frac{p(p-1)}{n}\}$$

Design phase: MKL 1-level, general

- Platform: L memory levels, c_l cores have a similar access speed to level l, $1 \leq l \leq L$
- k_{dgemm_NUMA}:
  if $0 < p \leq c_1$ then $k_{dgemm\_NUMA}(p) = k_{dgemm\_M_1}$
  if $c_1 < p \leq c_2$ then
  $k_{dgemm\_NUMA}(p) = \frac{c_1 k_{dgemm\_M_1} + (p - c_1) k_{dgemm\_M_2}}{p}$
  ...
  if $c_{L-1} < p \leq c_L$ then
  $k_{dgemm\_NUMA}(p) = \frac{\sum_{l=0}^{L-2} (c_l - c_{l-1}) k_{dgemm\_M_l} + (p - c_{L-1}) k_{dgemm\_M_L}}{p}$

Design phase: MKL 1-level, general

- Platform: L memory levels, c_l cores have a similar access speed to level l, $1 \leq l \leq L$
- k_{dgemm_NUMA}:
  if $0 < p \leq c_1$ then $k_{dgemm\_NUMA}(p) = k_{dgemm\_M_1}$
  if $c_1 < p \leq c_2$ then
  $k_{dgemm\_NUMA}(p) = \frac{c_1 k_{dgemm\_M_1} + (p - c_1) k_{dgemm\_M_2}}{p}$
  ...
  if $c_{L-1} < p \leq c_L$ then
  $k_{dgemm\_NUMA}(p) = \frac{\sum_{l=0}^{L-2} (c_l - c_{l-1}) k_{dgemm\_M_l} + (p - c_{L-1}) k_{dgemm\_M_L}}{p}$

Design phase: OpenMP+MKL

Model: $t_{2L\_dgemm} = \frac{2n^3}{R} k_{2L\_dgemm}$

$R = q * p$, q threads OpenMP, p threads MKL

$k_{2L\_dgemm} = \alpha k_{2L\_dgemm\_NUMA}(q,p) + (1-\alpha) k_{2L\_dgemm\_M_1}$

$k_{2L\_dgemm\_M_1}$: when data are in the closest memory to the core

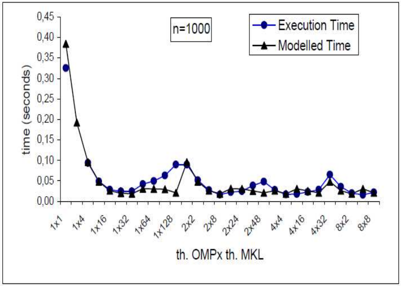
$k_{2L\_dgemm\_NUMA}$: when data are at any level in the memory
$k_{2L\_dgemm\_NUMA}(q,p) = \frac{k_{dgemm\_NUMA}(R)+k_{dgemm\_NUMA}(p)}{2}$

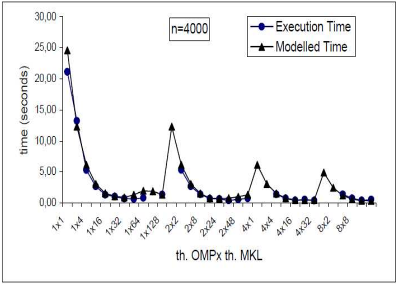
$\alpha = min\{1, \frac{R(R-1)}{n}\}$

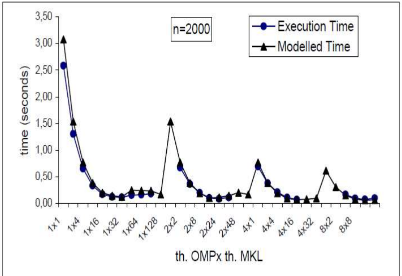
Installation

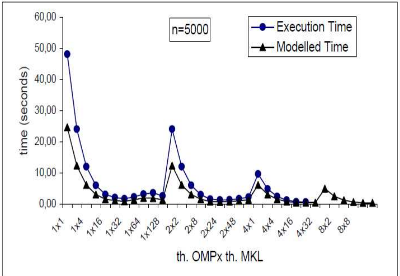
- Estimation of the parameters in the theoretical model:$k_{dgemm\_M_1}$, ..., $k_{dgemm\_M_L}$
- For each memory level l, $0 \leq l \leq L$, execute `dgemm` for a number of threads p_l, with $c_{l-1} < p_l \leq c_l$
  This execution time + routine model $\rightarrow$ k_{dgemm_NUMA} for p_l threads
  k_{dgemm_NUMA} value for p_l + k_{dgemm_NUMA} model + values of $k_{dgemm\_M_1}, \ldots k_{dgemm\_M_{l-1}} \rightarrow k_{dgemm\_M_l}$

Comparison model-experimental: Ben

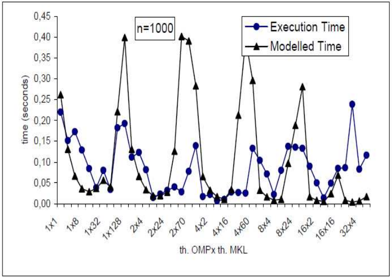


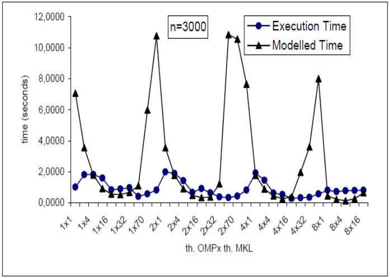


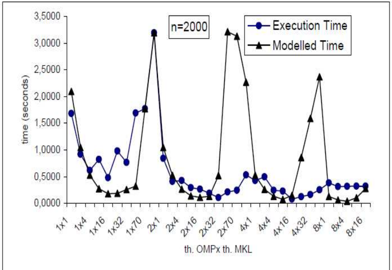


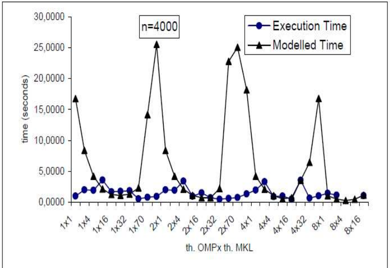


Comparison model-experimental: `Pirineus`









Execution

size	SEQ	MIN-MKL	MC-MKL	AUTO
Ben				
1000	0.320	0.024	0.091	0.012 (2×8)
2000	2.60	0.12	0.39	0.07 (4×16)
3000	8.60	0.32	0.82	0.23 (4×16)
4000	20.22	0.59	1.40	0.74 (4×32)
5000	40.23	1.12	2.11	1.44 (4×32)
Pirineus				
1000	0.224	0.034	0.441	0.021 (16×4)
2000	1.74	0.48	1.19	0.25 (8×8)
3000	5.46	0.39	1.31	0.39 (8×8)
4000	13.14	0.54	1.89	0.95 (8×8)
5000	25.12	1.13	2.65	1.02 (8×16)

Empirical installation

- Not to design and use the model of the execution time
- but to run some selected executions at installation time:
  - a large installation time can be necessary
  - For some problem sizes search the best parameters combination:
    exhaustive search
    guided search: search in the most promising direction
    Experiments with:
      Installation set={500,1000,3000,5000}
      Validation set={700,2000,4000}
      Cores in the experiment:
      `Ben 96; Saturno 24; Pirineus 240`

Empirical installation

- Not to design and use the model of the execution time
- but to run some selected executions at installation time:
  - a large installation time can be necessary
  - For some problem sizes search the best parameters combination:
    exhaustive search
    guided search: search in the most promising direction
    Experiments with:
      Installation set={500,1000,3000,5000}
      Validation set={700,2000,4000}
      Cores in the experiment:
      `Ben` 96; `Saturno` 24; `Pirineus` 240

Empirical installation

- Not to design and use the model of the execution time
- but to run some selected executions at installation time:
  - a large installation time can be necessary
  - For some problem sizes search the best parameters combination:
    exhaustive search
    guided search: search in the most promising direction
    Experiments with:
      Installation set={500,1000,3000,5000}
      Validation set={700,2000,4000}
      Cores in the experiment:
      `Ben 96; Saturno 24; Pirineus 240`

Empirical installation

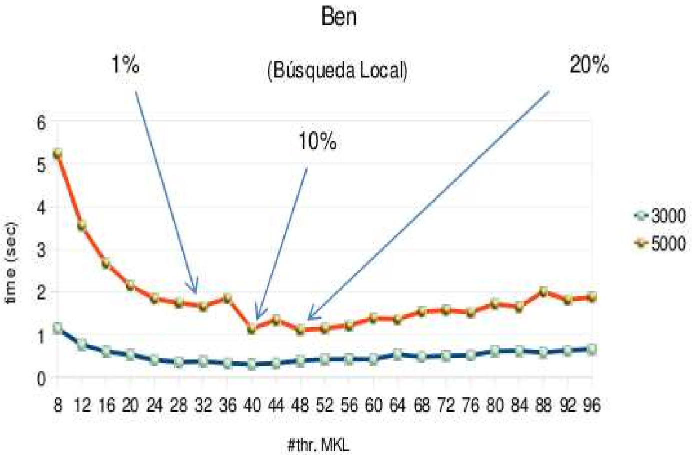
- Not to design and use the model of the execution time
- but to run some selected executions at installation time:
  - a large installation time can be necessary
  - For some problem sizes search the best parameters combination:
    exhaustive search
    guided search: search in the most promising direction
    Experiments with:
      Installation set={500,1000,3000,5000}
      Validation set={700,2000,4000}
      Cores in the experiment:
      `Ben` 96; `Saturno` 24; `Pirineus` 240

Exhaustive search

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

Guided search

There are local optima ⇒ use of a percentage of improvement to stop the search:



Guided search

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)

Installation time

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

Linear algebra: perspectives

- Higher level routines:
  - Use of basic routines in higher level routines (matrix factorizations, in collaboration with Parallel Computing group of the Polytechnic University of Valencia) and scientific applications (microstrip circuits, Computational Electromagnetism group, Polytechnic University of Cartagena).
  - Application of the techniques to higher level routines.
- Improvement of the technique:
  - Better models and search techniques.
  - Combination of modelling and search.

Linear algebra: perspectives

- Higher level routines:
  - Use of basic routines in higher level routines (matrix factorizations, in collaboration with Parallel Computing group of the Polytechnic University of Valencia) and scientific applications (microstrip circuits, Computational Electromagnetism group, Polytechnic University of Cartagena).
  - Application of the techniques to higher level routines.
- Improvement of the technique:
  - Better models and search techniques.
  - Combination of modelling and search.

Parallel-parametrized scheme

Initialize(S,ParamIni,ThreadsIni)
while (**not** EndCondition(S,ParamEnd,ThreadsEnd))
    SS = Select(S,ParamSel,ThreadsSel)
    $SS1$ = Combine(SS,ParamCom,ThreadsCom)
    $SS2$ = Improve($SS1$,ParamImp,ThreadsImp)
    S = Include($SS2$,ParamInc,ThreadsInc)

Independent parallelization of the functions, with parallelism parameters (number of threads) for each function. The optimum value of the parallelism parameters depends on the values of the metaheuristic parameters (the metaheuristic or combination of metaheuristics).

Identify functions with the same parallel scheme:

One-level parallel scheme (scheme 1)

omp_set_num_threads($threads - one - level$(MetaheurParam))
#pragma omp parallel for
loop in elements
    treat element

i.e.: Initialize, Combine...

Two-level parallel scheme (scheme 2)

two–level(MetaheurParam) :
  omp_set_num_threads($threads - first - level$(MetaheurParam))
  #pragma omp parallel for
  loop in elements
    second–level(MetaheurParam,$threads - first - level$)

second–level(MetaheurParam,$threads - first - level$):
  omp_set_num_threads($threads - second - level$(MetaheurParam,$threads - first - level$))
  #pragma omp parallel for
  loop in neighbors
    treat neighbor

i.e.: Initialize, Improve...

Allows fine and coarse grained parallelism by changing the number of threads in each level

Design

- A model is obtained for each basic routine. Two basic models can be used, one for one-level routines and another for nested parallelism.
- The generation of the initial population in function Initialize with an initial number of elements in the reference set *INEIni*, can be modelled:

$$t_{1-level} = \frac{k_g \cdot INEIni}{p} + k_p \cdot p \quad (1)$$

- And the improvement of a percentage of the initial elements *PEIIni* with an intensification (extension of the considered neighborhood) *IIEIni* is modeled:

$$t_{2-levels} = \frac{k_i \cdot \frac{INEIni \cdot PEIIni \cdot IIEIni}{100}}{p_1} + k_{p,1} \cdot p_1 + k_{p,2} \cdot p_2 \quad (2)$$

Design

- A model is obtained for each basic routine. Two basic models can be used, one for one-level routines and another for nested parallelism.
- The generation of the initial population in function Initialize with an initial number of elements in the reference set *INEIni*, can be modelled:

$$t_{1-level} = \frac{k_g \cdot INEIni}{p} + k_p \cdot p \tag{1}$$

- And the improvement of a percentage of the initial elements *PEIIni* with an intensification (extension of the considered neighborhood) *IIEIni* is modeled:

$$t_{2-levels} = \frac{k_i \cdot \frac{INEIni \cdot PEIIni \cdot IIEIni}{100}}{p_1} + k_{p,1} \cdot p_1 + k_{p,2} \cdot p_2 \tag{2}$$

Design

- A model is obtained for each basic routine. Two basic models can be used, one for one-level routines and another for nested parallelism.
- The generation of the initial population in function Initialize with an initial number of elements in the reference set *INEIni*, can be modelled:

$$t_{1-level} = \frac{k_g \cdot INEIni}{p} + k_p \cdot p \tag{1}$$

- And the improvement of a percentage of the initial elements *PEIIni* with an intensification (extension of the considered neighborhood) *IIEIni* is modeled:

$$t_{2-levels} = \frac{k_i \cdot \frac{INEIni \cdot PEIIni \cdot IIEIni}{100}}{p_1} + k_{p,1} \cdot p_1 + k_{p,2} \cdot p_2 \tag{2}$$

Application problem

- Electricity consumption in exploitation of water resources:
  - Water pumping in exploitation of water resources.
  - There are a number of technical constraints to be complied with (restrictions).
  - Our goal is to apply an algorithm that allows us to optimize the cost of electricity subject to the restrictions.
  - The space of possible solutions is very large and exhaustive methods are not applicable here $\Rightarrow$
- Metaheuristic:
  - Pure metaheuristics: GRASP, Genetic algorithms (GA), Scatter search (SS)
  - Combinations: GRASP+GA, GRASP+SS, GA+SS, GRASP+GA+SS

Application problem

- Electricity consumption in exploitation of water resources:
  - Water pumping in exploitation of water resources.
  - There are a number of technical constraints to be complied with (restrictions).
  - Our goal is to apply an algorithm that allows us to optimize the cost of electricity subject to the restrictions.
  - The space of possible solutions is very large and exhaustive methods are not applicable here $\Rightarrow$
- Metaheuristic:
  - Pure metaheuristics: GRASP, Genetic algorithms (GA), Scatter search (SS)
  - Combinations: GRASP+GA, GRASP+SS, GA+SS, GRASP+GA+SS

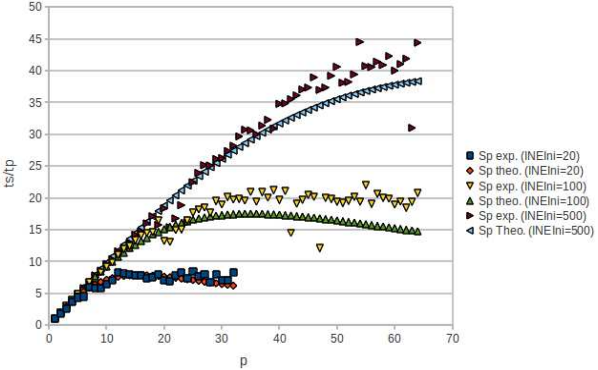
Installation

Experiments with some metaheuristic parameters, and system parameters in the formula obtained by least-square:

- For the one-level routine studied, in the experiments with $INEIni = 20$: $k_g = 2.38 \cdot 10^{-3}$ and $k_p = 1.94 \cdot 10^{-4}$, all in seconds.
- For the two-level routine studied, with metaheuristic parameters $INEIni = 20$, $PEIIni = 50$, $IIEIni = 20$ and $p_2 = 1$: $k_i = 9.10 \cdot 10^{-4}$, $k_{p,1} = 6.50 \cdot 10^{-4}$ and $k_{p,2} = 6.31 \cdot 10^{-3}$ seconds

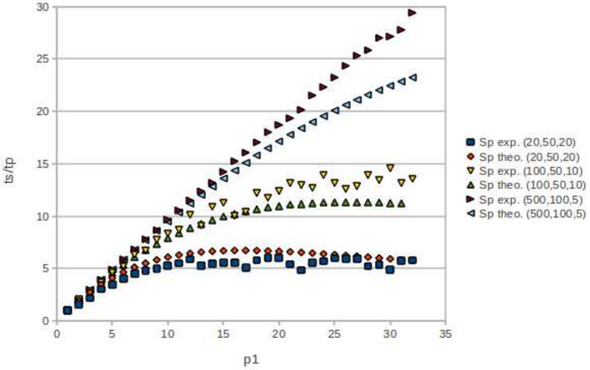
Theoretical-experimental comparison. One-level routine

Theoretical and experimental speed-up for three parameters when varying the number of threads in the initial generation of the reference set



Theoretical-experimental comparison. Two-level routine

Theoretical and experimental speed-up for three combinations of the parameters *INEIni*, *PEIIni* and *IIEIni* when varying the number of threads in the improvement routine



Execution. One-level routine

Initial generation of the reference set:

$$p_{opt.} = \sqrt{\frac{k_g}{k_p} \cdot INEIni} = 3.50 \cdot \sqrt{INEIni} \tag{3}$$

Speed-up and number of threads for $INEIni = 100$ and 500 in the one-level parallel routine. Optimum experimental values (optimum) and values obtained with autotuning (model)

	threads		speed-up	
INEIni	optimum	model	optimum	model
100	55	35	22	18
500	64	78	44	39

Execution. Two-levels routine

Improvement of the generated elements:

$$p_{1,opt.} = 1.18 \cdot 10^{-1} \cdot \sqrt{INEIni \cdot PEIIni \cdot IIEIni} \qquad (4)$$

Speed-up and number of threads for other parameter combinations in the two-level parallel routine. Optimum experimental values (optimum) and values obtained with autotuning (model)

			threads		speed-up	
INEIni	*PEIIni*	*IIEIni*	optimum	model	optimum	model
100	50	10	30	26	15	11
500	100	5	32	59	29	27

Metaheuristics: perspectives

- Inclusion of more "pure" metaheuristics (Tabu, Ant...).
- Design of hyperheuristics to automatically select the values of the metaheuristic parameters for a particular problem.
- Inclusion of autotuning in the parallel scheme, with some engine to autonomously select the number of threads.
- Develop unified parallel schemes for other computational systems (message-passing, hybrid, GPU...).

Context

- Accelerate the resolution of scientific problems by combining CPU+GPU
  - Combination of OpenMP+CUDA parallelism
  - Heterogeneous system
  - MultiGPU
- Systems:
  - UM: 4 cores + GPU
  - UPV: 12 cores + 2 GPU
- Problems:
  - Green functions for waveguides
  - Simultaneous Equations Models

Context

- Accelerate the resolution of scientific problems by combining CPU+GPU
  - Combination of OpenMP+CUDA parallelism
  - Heterogeneous system
  - MultiGPU
- Systems:
  - UM: 4 cores + GPU
  - UPV: 12 cores + 2 GPU
- Problems:
  - Green functions for waveguides
  - Simultaneous Equations Models

Context

- Accelerate the resolution of scientific problems by combining CPU+GPU
  - Combination of OpenMP+CUDA parallelism
  - Heterogeneous system
  - MultiGPU
- Systems:
  - UM: 4 cores + GPU
  - UPV: 12 cores + 2 GPU
- Problems:
  - Green functions for waveguides
  - Simultaneous Equations Models

Modelling CPU+GPU computation ?

- Design: extend the ideas of modelling in multicore:

$$\frac{t_s}{c + s_{g/c} g} + t_{sc} c + t_{sk} g$$

  - t_s sequential time
  - c, g number of cores and of GPUs
  - $s_{g/c}$ speed-up of one GPU with respect to one core for the problem in question
  - t_{sc}, t_{sk} cost of generation of a core and a kernel

- Installation: use some installation methodology to estimate the values of the parameters in a particular system.

- Execution: for a particular entry (problem size) and in a particular system (the computational system+the implemented algorithms) select the algorithm and the part of the computational system to use in the solution of the problem.

Modelling CPU+GPU computation ?

- Design: extend the ideas of modelling in multicore:

$$\frac{t_s}{c + s_{g/c} g} + t_{sc} c + t_{sk} g$$

  - t_s sequential time
  - c, g number of cores and of GPUs
  - $s_{g/c}$ speed-up of one GPU with respect to one core for the problem in question
  - t_{sc}, t_{sk} cost of generation of a core and a kernel

- Installation: use some installation methodology to estimate the values of the parameters in a particular system.
- Execution: for a particular entry (problem size) and in a particular system (the computational system+the implemented algorithms) select the algorithm and the part of the computational system to use in the solution of the problem.

Modelling CPU+GPU computation ?

- Design: extend the ideas of modelling in multicore:

$$\frac{t_s}{c + s_{g/c} g} + t_{sc} c + t_{sk} g$$

  - t_s sequential time
  - c, g number of cores and of GPUs
  - $s_{g/c}$ speed-up of one GPU with respect to one core for the problem in question
  - t_{sc}, t_{sk} cost of generation of a core and a kernel

- Installation: use some installation methodology to estimate the values of the parameters in a particular system.

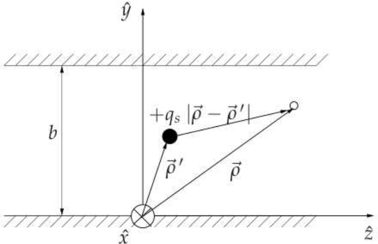
- Execution: for a particular entry (problem size) and in a particular system (the computational system+the implemented algorithms) select the algorithm and the part of the computational system to use in the solution of the problem.

Green functions

- Used to solve non homogeneous differential equations with boundary conditions.
- Applied to waveguides, which are used in the design and analysis of integrated circuits MMIC (Monolithic Microwave Integrated Circuits).
- They can be expressed in the form of infinite series, in the spatial or spectral domain.
- It is necessary to calculate hundreds or thousands of Green functions.

Application to waveguides

- There is a parallel plate guide along z axis.
- Inside this guide is a set of source and observer points which move in axes $\hat{y}$ and $\hat{z}$.
- The Green function associated to each pair of points is calculated.
- The two series in the Ewald method are computed.
- The number of terms can be fixed for all the pairs or be dynamically calculated as a function of the distance between the two points.



One-dimensional problem

```
{For each source point}
for i = 1 to m do
   {For each observer point}
   for j = 1 to n do
      {For the number of modes (terms)}
      {Calculation of summation in the spectral domain}
      for k = 1 to nmod do
         trigonometric operations
      end for
      {Summation of the trigonometric functions}
      for k = 1 to nmod do
         trigonometric operations
      end for
      Apply the method of acceleration of Kummer
   end for
end for
```

Cost $O\left(m \cdot n \cdot nmod\right)$

Two-dimensional problem

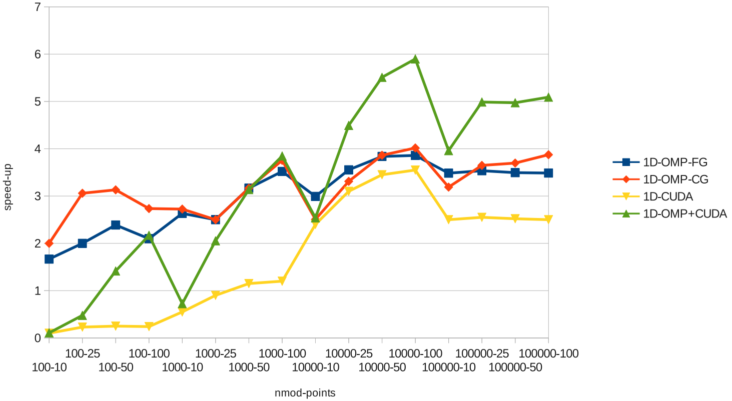
```
Initialization: obtain and sort modes
for i = 1 to m do
    for j = 1 to n do
        {Spectral part}
        for k = 1 to nmod do
            GF[i, j]+ = spectral(k)
        end for
        {Spatial part}
        {For images in axes x and y}
        for r = −mimag to mimag do
            for s = −imag to nimag do
                GF[i, j]+ = spatial(r, s)
            end for
        end for
    end for
end for
```

$$\text{Cost } O\left(m \cdot n \cdot mimag \cdot nimag\right)$$

One-dimensional implementations

- **1D-OMP-FG**: A fine grain version with OpenMP the calculation. The two innermost loops are parellelized.
- **1D-OMP-CG**: Coarse grain parallelism with OpenMP, parallelizing the work in the outer loop.
- **1D-CUDA**: The computation of each Green's function (fine grain parallelism) is performed by the GPU.
- **1D-OMP+CUDA**: Hybrid implementation. In an shared-memory program (with OpenMP) the number of threads generated is one more than the number of cores. One of the threads is in charge of calling the CUDA kernel. The other threads follow the coarse grain shared-memory scheme.

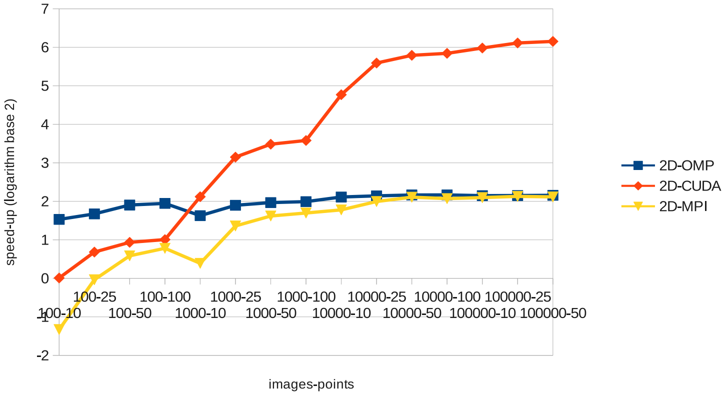
Speed-up


7
6
5
4
3
2
1
0
speed-up
100-10
100-25
100-50
100-100
1000-10
1000-25
1000-50
1000-100
10000-10
10000-25
10000-50
10000-100
100000-10
100000-25
100000-50
100000-100
nmod-points
1D-OMP-FG
1D-OMP-CG
1D-CUDA
1D-OMP+CUDA

Two-dimensional implementations

- **2D-OMP**: Parallelizing the first loop of the spatial part. The access to some variables to store partial results is done with reduction.
- **2D-CUDA**: Each thread is in charge of the computation of one image. An auxiliar matrix is used to store the partial sum obtained by each thread, and the values in the matrix are added sequentially.
- **2D-MPI**: The spatial part is parallelized, and the spectral part is done sequentially. Similar to that of OpenMP, and the final sum is obtained with `MPI_Reduce`.

Speed-up


7
6
5
4
3
2
1
0
-1
-2
speed-up (logarithm base 2)
100-10
100-25
100-50
100-100
1000-10
1000-25
1000-50
1000-100
10000-10
10000-25
10000-50
10000-100
100000-10
100000-25
100000-50
images-points
2D-OMP
2D-CUDA
2D-MPI

Simultaneous Equations Models

- N interdependent variables (endogenous variables) which depend on K independent variables (exogenous variables).
- Each endogenous variable can be expressed as a linear combination of the other endogenous variables, the exogenous variables, and white noise:

$$\mathbf{Y} = \mathbf{Y}\mathbf{B}^{\mathbf{T}} + \mathbf{X}\mathbf{\Gamma}^{\mathbf{T}} + \mathbf{u}$$

where $\mathbf{Y} \in \mathbb{R}^{d\times N}$, $\mathbf{X} \in \mathbb{R}^{d\times K}$ and $\mathbf{u} \in \mathbb{R}^{d\times N}$ are matrices with N endogenous variables, K exogenous variables and N white noise variables respectively, being d the sample size, and elements $\mathbf{B_{ii}} = 0$.
- Solving a SEM is equivalent to obtaining $\mathbf{B}$ and $\mathbf{\Gamma}$, from a representative sample of the model (a set of values of the data variables $\mathbf{X}$ and $\mathbf{Y}$) in order to explicitly know a matrix equation which represents the relationship between both sets of variables.

Simultaneous Equations Models

- N interdependent variables (endogenous variables) which depend on K independent variables (exogenous variables).
- Each endogenous variable can be expressed as a linear combination of the other endogenous variables, the exogenous variables, and white noise:

$$\mathbf{Y} = \mathbf{Y}\mathbf{B}^{\mathbf{T}} + \mathbf{X}\boldsymbol{\Gamma}^{\mathbf{T}} + \mathbf{u}$$

  where $\mathbf{Y} \in \mathbb{R}^{d \times N}$, $\mathbf{X} \in \mathbb{R}^{d \times K}$ and $\mathbf{u} \in \mathbb{R}^{d \times N}$ are matrices with N endogenous variables, K exogenous variables and N white noise variables respectively, being d the sample size, and elements $\mathbf{B_{ii}} = 0$.
- Solving a SEM is equivalent to obtaining $\mathbf{B}$ and $\boldsymbol{\Gamma}$, from a representative sample of the model (a set of values of the data variables $\mathbf{X}$ and $\mathbf{Y}$) in order to explicitly know a matrix equation which represents the relationship between both sets of variables.

Simultaneous Equations Models

- N interdependent variables (endogenous variables) which depend on K independent variables (exogenous variables).
- Each endogenous variable can be expressed as a linear combination of the other endogenous variables, the exogenous variables, and white noise:

$$\mathbf{Y} = \mathbf{Y}\mathbf{B}^{\mathbf{T}} + \mathbf{X}\mathbf{\Gamma}^{\mathbf{T}} + \mathbf{u}$$

  where $\mathbf{Y} \in \mathbb{R}^{d\times N}$, $\mathbf{X} \in \mathbb{R}^{d\times K}$ and $\mathbf{u} \in \mathbb{R}^{d\times N}$ are matrices with N endogenous variables, K exogenous variables and N white noise variables respectively, being d the sample size, and elements $\mathbf{B_{ii}} = 0$.
- Solving a SEM is equivalent to obtaining $\mathbf{B}$ and $\mathbf{\Gamma}$, from a representative sample of the model (a set of values of the data variables $\mathbf{X}$ and $\mathbf{Y}$) in order to explicitly know a matrix equation which represents the relationship between both sets of variables.

Two-Stage Least Squares

Require: $\mathbf{X} \in \mathbb{R}^{d\times K}$, $\mathbf{Y} \in \mathbb{R}^{d\times N}$ and zero pattern of $\mathbf{B}$ and $\mathbf{\Gamma}$

Ensure: $\mathbf{B} \in \mathbb{R}^{N\times N}$ and $\mathbf{\Gamma} \in \mathbb{R}^{N\times K}$

  Obtain $\mathbf{Q}$, $\mathbf{R}$ and $\tilde{\mathbf{Y}}$ such that $\mathbf{X} = \mathbf{QR}$ (QRD of $\mathbf{X}$) and $\tilde{\mathbf{Y}} = \mathbf{Q}^\mathbf{T}\mathbf{Y}$

  for i=1…N **do**

    if i-th equation is identified (i.e. it can be solved) **then**

      $[\mathbf{R_{i,1}}|\tilde{\mathbf{Y}}_{\mathbf{i},1}] \leftarrow$ Select columns from $[\mathbf{R_1}|\tilde{\mathbf{Y}}_1]$

      Obtain $\tilde{\mathbf{Q}}_\mathbf{i}$, $\tilde{\mathbf{R}}_{\mathbf{i},1}$ and $\tilde{\tilde{\mathbf{y}}}_{\mathbf{i},1}$ such that $[\mathbf{R_{i,1}}|\tilde{\mathbf{Y}}_{\mathbf{i},1}] = \tilde{\mathbf{Q}}_\mathbf{i}\tilde{\mathbf{R}}_{\mathbf{i},1}$ and $\tilde{\tilde{\mathbf{y}}}_{\mathbf{i},1} = \tilde{\mathbf{Q}}_\mathbf{i}^\mathbf{T}\tilde{\mathbf{y}}_{\mathbf{i},1}$

      Solve $\tilde{\mathbf{R}}_{\mathbf{i},1}\hat{\eta}_\mathbf{i} = \tilde{\tilde{\mathbf{y}}}_{\mathbf{i},1}$

    end if

  end for

Implementations

- Parallelization by distribution of the equations among the various computational elements:
  - **OMP**: Only the cores in the CPU are used.
  - **OMPGPU**: Distributes the solution of the equations among the cores of the CPU and of the GPU.
- Parallelize the computation of the QRD, by using Givens rotations, and taking advantage of the structure of the matrix $[R_{i,1}|\tilde{Y}_{i,1}]$:
  - **C1T**: On one GPU.
  - **C2T**: Distributes dynamically, with OpenMP, the equations between the two GPUs. Each GPU applies the parallel QRD on its set of equations.
- Hybrid parallelization: The set of equations to be solved are divided dynamically among the various computational elements. The GPU applies the parallel QRD on its set of equations.
  - **MULTI1T**: Uses the cores in the CPU + 1 GPU.
  - **MULTI2T**: Uses the cores in the CPU + 2 GPU.

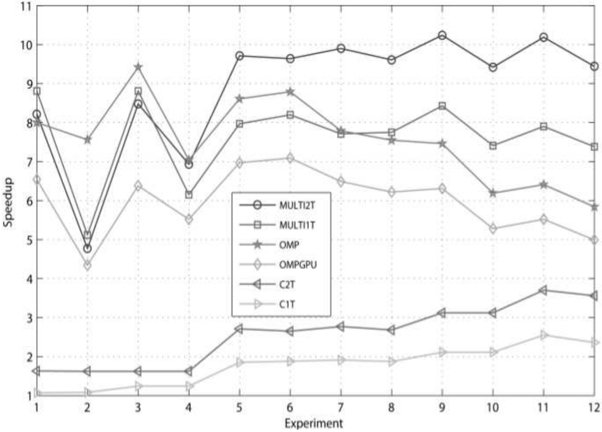
Implementations

- Parallelization by distribution of the equations among the various computational elements:
  - **OMP**: Only the cores in the CPU are used.
  - **OMPGPU**: Distributes the solution of the equations among the cores of the CPU and of the GPU.
- Parallelize the computation of the QRD, by using Givens rotations, and taking advantage of the structure of the matrix $[R_{i,1}|\tilde{Y}_{i,1}]$:
  - **C1T**: On one GPU.
  - **C2T**: Distributes dynamically, with OpenMP, the equations between the two GPUs. Each GPU applies the parallel QRD on its set of equations.
- Hybrid parallelization: The set of equations to be solved are divided dynamically among the various computational elements. The GPU applies the parallel QRD on its set of equations.
  - **MULTI1T**: Uses the cores in the CPU + 1 GPU.
  - **MULTI2T**: Uses the cores in the CPU + 2 GPU.

Implementations

- Parallelization by distribution of the equations among the various computational elements:
  - **OMP**: Only the cores in the CPU are used.
  - **OMPGPU**: Distributes the solution of the equations among the cores of the CPU and of the GPU.
- Parallelize the computation of the QRD, by using Givens rotations, and taking advantage of the structure of the matrix $[R_{i,1}|\tilde{Y}_{i,1}]$:
  - **C1T**: On one GPU.
  - **C2T**: Distributes dynamically, with OpenMP, the equations between the two GPUs. Each GPU applies the parallel QRD on its set of equations.
- Hybrid parallelization: The set of equations to be solved are divided dynamically among the various computational elements. The GPU applies the parallel QRD on its set of equations.
  - **MULTI1T**: Uses the cores in the CPU + 1 GPU.
  - **MULTI2T**: Uses the cores in the CPU + 2 GPU.

Speed-up



Perspectives

- Modelling can help in the auto-tuning of basic parallel routines and scientific codes, so contributing to the efficient use of parallel programs.
- Hybrid parallelism (multiple level, different types of parallelism, different paradigms...) introduces additional difficulties.
- Sometimes the theoretical models are combined with empirical analysis.
- Some successful applications are shown, but better modelling techniques are needed, especially for complex scientific problems, more complex computational systems and more hybrid-heterogeneous-hierarchical programming.