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

1. Motivation
2. Linear algebra
3. Metaheuristics
4. CPU+GPU
5. Perspectives
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+2IOpenMP+BLAS+MultiGPU...
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

![Graphs showing speed-up comparison](image)
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

- Creation of the routine and the model

Installation

- Application of the autotuning technique
- Constants of the model for the problem and system

Execution

- Determination of the parallelism parameters in the model
- Execution with the selected parallelism parameters
Design phase: MKL 1-level

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

\( k_{dgemm} = \alpha k_{dgemm_{-NUMA}(p)} + (1 - \alpha) k_{dgemm_{-M1}} \)

\( k_{dgemm_{-M1}} \): when data are in the memory closest to the core

\( k_{dgemm_{-NUMA}} \): when data are in any level in the memory

\( \alpha \): 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 $k_{dgemm\_NUMA}(p) = k_{dgemm\_M1}$
  if $c_1 < p \leq c_2$ then
  $$k_{dgemm\_NUMA}(p) = \frac{c_1k_{dgemm\_M1} + (p - c_1)k_{dgemm\_M2}}{p}$$
  ...
  if $c_{L-1} < p \leq c_L$ then
  $$k_{dgemm\_NUMA}(p) = \frac{\sum_{l=0}^{L-2}(c_l - c_{l-1})k_{dgemm\_Ml} + (p - c_{L-1})k_{dgemm\_ML}}{p}$$
Design phase: MKL 1-level, general

- Platform: $L$ memory levels, $c_l$ cores have a similar access speed to level $l$, $1 \leq l \leq L$

- $k_{\text{dgemm\_NUMA}}$:
  
  if $0 < p \leq c_1$ then $k_{\text{dgemm\_NUMA}}(p) = k_{\text{dgemm\_M}_1}$
  
  if $c_1 < p \leq c_2$ then
  
  $$k_{\text{dgemm\_NUMA}}(p) = \frac{c_1 k_{\text{dgemm\_M}_1} + (p - c_1) k_{\text{dgemm\_M}_2}}{p}$$

  
  ... 

  if $c_{L-1} < p \leq c_L$ then

  $$k_{\text{dgemm\_NUMA}}(p) = \frac{\sum_{l=0}^{L-2} (c_l - c_{l-1}) k_{\text{dgemm\_M}_l} + (p - c_{L-1}) k_{\text{dgemm\_M}_L}}{p}$$
Design phase: OpenMP+MKL

Model: \( t_{2L_dgemm} = \frac{2n^3}{R} k_{2L_dgemm} \)

\( R = q \times p, \) \( q \) threads OpenMP, \( p \) threads MKL

\( k_{2L_dgemm} = \alpha k_{2L_dgemm_{NUMA}}(q, p) + (1 - \alpha) k_{2L_dgemm_{M1}} \)

\( k_{2L_dgemm_{M1}}: \) when data are in the closest memory to the core

\( k_{2L_dgemm_{NUMA}}: \) when data are at any level in the memory

\( k_{2L_dgemm_{NUMA}}(q, p) = \frac{k_{dgemm_{NUMA}(R)} + k_{dgemm_{NUMA}(p)}}{2} \)

\( \alpha = \min\{1, \frac{R(R-1)}{n}\} \)
Estimation of the parameters in the theoretical model: $k_{\text{dgemm} \_ M_1}, \ldots, k_{\text{dgemm} \_ M_L}$

For each memory level $l$, $0 \leq l \leq L$, execute \text{dgemm} for a number of threads $p_l$, with $c_{l-1} < p_l \leq c_l$

This execution time + routine model $\rightarrow k_{\text{dgemm} \_ \text{NUMA}}$ for $p_l$ threads

$k_{\text{dgemm} \_ \text{NUMA}}$ value for $p_l$ $+ k_{\text{dgemm} \_ \text{NUMA}}$ model $+ \text{values of } k_{\text{dgemm} \_ M_1}, \ldots, k_{\text{dgemm} \_ M_{l-1}} \rightarrow k_{\text{dgemm} \_ M_l}$
Comparison model-experimental: Ben

Motivation
Linear algebra
Metaheuristics
CPU+GPU
Perspectives

![Comparison model-experimental: Ben](image_url)
Comparison model-experimental: Pirineus
### Execution

<table>
<thead>
<tr>
<th>size</th>
<th>SEQ</th>
<th>MIN-MKL</th>
<th>MC-MKL</th>
<th>AUTO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Ben</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>0.320</td>
<td>0.024</td>
<td>0.091</td>
<td>0.012 (2×8)</td>
</tr>
<tr>
<td>2000</td>
<td>2.60</td>
<td>0.12</td>
<td>0.39</td>
<td>0.07 (4×16)</td>
</tr>
<tr>
<td>3000</td>
<td>8.60</td>
<td>0.32</td>
<td>0.82</td>
<td>0.23 (4×16)</td>
</tr>
<tr>
<td>4000</td>
<td>20.22</td>
<td>0.59</td>
<td>1.40</td>
<td>0.74 (4×32)</td>
</tr>
<tr>
<td>5000</td>
<td>40.23</td>
<td>1.12</td>
<td>2.11</td>
<td>1.44 (4×32)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>size</th>
<th>SEQ</th>
<th>MIN-MKL</th>
<th>MC-MKL</th>
<th>AUTO</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pirineus</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>0.224</td>
<td>0.034</td>
<td>0.441</td>
<td>0.021 (16×4)</td>
</tr>
<tr>
<td>2000</td>
<td>1.74</td>
<td>0.48</td>
<td>1.19</td>
<td>0.25 (8×8)</td>
</tr>
<tr>
<td>3000</td>
<td>5.46</td>
<td>0.39</td>
<td>1.31</td>
<td>0.39 (8×8)</td>
</tr>
<tr>
<td>4000</td>
<td>13.14</td>
<td>0.54</td>
<td>1.89</td>
<td>0.95 (8×8)</td>
</tr>
<tr>
<td>5000</td>
<td>25.12</td>
<td>1.13</td>
<td>2.65</td>
<td>1.02 (8×16)</td>
</tr>
</tbody>
</table>
**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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>0.0056 (20)</td>
<td></td>
<td></td>
<td>1.40</td>
</tr>
<tr>
<td>700</td>
<td>0.0121 (24)</td>
<td>0.0142 (20)</td>
<td>0.85</td>
<td>5.50</td>
</tr>
<tr>
<td>1000</td>
<td>0.0270 (20)</td>
<td></td>
<td></td>
<td>70.72</td>
</tr>
<tr>
<td>2000</td>
<td>0.1294 (36)</td>
<td>0.1790 (30)</td>
<td>0.72</td>
<td>275.06</td>
</tr>
<tr>
<td>3000</td>
<td>0.3076 (40)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4000</td>
<td>0.6387 (40)</td>
<td>0.8121 (44)</td>
<td>0.79</td>
<td></td>
</tr>
<tr>
<td>5000</td>
<td>1.1098 (48)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| TOTAL: | 352.69 |

## Saturno

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>0.0045 (24)</td>
<td></td>
<td></td>
<td>0.22</td>
</tr>
<tr>
<td>700</td>
<td>0.0099 (24)</td>
<td>0.0163 (22)</td>
<td>0.61</td>
<td>1.43</td>
</tr>
<tr>
<td>1000</td>
<td>0.0318 (20)</td>
<td></td>
<td></td>
<td>26.44</td>
</tr>
<tr>
<td>2000</td>
<td>0.2257 (24)</td>
<td>0.2532 (22)</td>
<td>0.89</td>
<td></td>
</tr>
<tr>
<td>3000</td>
<td>0.7255 (24)</td>
<td></td>
<td></td>
<td>77.67</td>
</tr>
<tr>
<td>4000</td>
<td>1.6461 (24)</td>
<td>1.9459 (20)</td>
<td>0.85</td>
<td></td>
</tr>
<tr>
<td>5000</td>
<td>2.0901 (18)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| TOTAL: | 105.77 |

## Pirineus

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>0.0059 (24)</td>
<td></td>
<td></td>
<td>1.8</td>
</tr>
<tr>
<td>750</td>
<td>0.0139 (24)</td>
<td>0.0597 (16)</td>
<td>0.23</td>
<td>2.74</td>
</tr>
<tr>
<td>1000</td>
<td>0.0322 (12)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td>0.4796 (16)</td>
<td>0.7659 (32)</td>
<td>0.63</td>
<td>24.61</td>
</tr>
<tr>
<td>3000</td>
<td>0.3955 (60)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4000</td>
<td>0.5397 (60)</td>
<td>0.5397 (60)</td>
<td>1.00</td>
<td></td>
</tr>
<tr>
<td>5000</td>
<td>1.1149 (60)</td>
<td></td>
<td></td>
<td>81.41</td>
</tr>
</tbody>
</table>

| TOTAL: | 110.56 |
Guided search

There are local optima $\Rightarrow$ use of a percentage of improvement to stop the search:
## Guided search

<table>
<thead>
<tr>
<th>n</th>
<th>Opt.</th>
<th>1.00%</th>
<th>10.00%</th>
<th>20.00%</th>
<th>50.00%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ben</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>0.0050 (23-1)</td>
<td>0.0134 (1-4)</td>
<td>0.0051 (4-6)</td>
<td>0.0051 (4-6)</td>
<td>0.0055 (1-20)</td>
</tr>
<tr>
<td>700</td>
<td>0.0102 (7-4)</td>
<td>0.0148 (1-10)</td>
<td>0.0119 (5-6)</td>
<td>0.0119 (5-6)</td>
<td>0.0111 (1-19)</td>
</tr>
<tr>
<td>1000</td>
<td>0.0177 (10-4)</td>
<td>0.0297 (1-16)</td>
<td>0.0183 (6-7)</td>
<td>0.0183 (6-7)</td>
<td>0.0246 (1-19)</td>
</tr>
<tr>
<td>2000</td>
<td>0.0795 (10-5)</td>
<td>0.0984 (3-15)</td>
<td>0.0826 (6-8)</td>
<td>0.0826 (6-8)</td>
<td>0.0963 (3-16)</td>
</tr>
<tr>
<td>3000</td>
<td>0.2191 (25-3)</td>
<td>0.2303 (5-14)</td>
<td>0.2380 (6-10)</td>
<td>0.2380 (6-10)</td>
<td>0.2303 (5-14)</td>
</tr>
<tr>
<td>4000</td>
<td>0.5088 (32-2)</td>
<td>0.6291 (6-10)</td>
<td>0.6150 (7-8)</td>
<td>0.6150 (7-8)</td>
<td>1.0728 (6-10)</td>
</tr>
<tr>
<td>5000</td>
<td>0.9207 (20-3)</td>
<td>0.9612 (8-7)</td>
<td>0.9612 (8-7)</td>
<td>0.9612 (8-7)</td>
<td>0.9612 (8-7)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Saturno</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>0.0038 (6-4)</td>
<td>0.0085 (2-2)</td>
<td>0.0085 (2-2)</td>
<td>0.0085 (2-2)</td>
<td>0.0042 (2-12)</td>
</tr>
<tr>
<td>700</td>
<td>0.0098 (6-4)</td>
<td>0.0227 (2-2)</td>
<td>0.0227 (2-2)</td>
<td>0.0227 (2-2)</td>
<td>0.0113 (2-12)</td>
</tr>
<tr>
<td>1000</td>
<td>0.0291 (8-3)</td>
<td>0.0325 (3-3)</td>
<td>0.0325 (3-3)</td>
<td>0.0325 (3-3)</td>
<td>0.0295 (2-12)</td>
</tr>
<tr>
<td>2000</td>
<td>0.2151 (6-4)</td>
<td>0.2604 (3-3)</td>
<td>0.2604 (3-3)</td>
<td>0.2604 (3-3)</td>
<td>0.2581 (2-10)</td>
</tr>
<tr>
<td>3000</td>
<td>0.5205 (1-17)</td>
<td>0.8338 (3-3)</td>
<td>0.8338 (3-3)</td>
<td>0.8338 (3-3)</td>
<td>0.7089 (3-8)</td>
</tr>
<tr>
<td>4000</td>
<td>1.2580 (1-17)</td>
<td>2.1135 (3-6)</td>
<td>2.1135 (3-6)</td>
<td>2.1135 (3-6)</td>
<td>1.9754 (3-7)</td>
</tr>
<tr>
<td>5000</td>
<td>1.8915 (7-3)</td>
<td>1.9567 (3-7)</td>
<td>1.9567 (3-7)</td>
<td>1.9567 (3-7)</td>
<td>1.9567 (3-7)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pirineus</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>0.0059 (1-24)</td>
<td>0.0075 (4-4)</td>
<td>0.0075 (4-4)</td>
<td>0.0075 (4-4)</td>
<td>0.0075 (4-4)</td>
</tr>
<tr>
<td>750</td>
<td>0.0134 (2-16)</td>
<td>0.0160 (4-4)</td>
<td>0.0251 (4-6)</td>
<td>0.0251 (4-6)</td>
<td>0.0251 (4-6)</td>
</tr>
<tr>
<td>1000</td>
<td>0.0235 (2-16)</td>
<td>0.0334 (4-3)</td>
<td>0.0264 (4-8)</td>
<td>0.0264 (4-8)</td>
<td>0.0264 (4-8)</td>
</tr>
<tr>
<td>2000</td>
<td>0.0797 (5-12)</td>
<td>0.2319 (4-8)</td>
<td>0.0813 (4-15)</td>
<td>0.0813 (4-15)</td>
<td>0.0813 (4-15)</td>
</tr>
<tr>
<td>3000</td>
<td>0.2752 (4-15)</td>
<td>0.2752 (4-15)</td>
<td>0.2752 (4-15)</td>
<td>0.2752 (4-15)</td>
<td>0.2752 (4-15)</td>
</tr>
<tr>
<td>4000</td>
<td>0.4670 (5-12)</td>
<td>0.4670 (5-12)</td>
<td>0.4670 (5-12)</td>
<td>0.4670 (5-12)</td>
<td>0.4670 (5-12)</td>
</tr>
<tr>
<td>5000</td>
<td>0.8598 (10-9)</td>
<td>0.8949 (5-18)</td>
<td>0.8949 (5-18)</td>
<td>0.8949 (5-18)</td>
<td>0.8949 (5-18)</td>
</tr>
</tbody>
</table>
## Installation time

### MKL:

<table>
<thead>
<tr>
<th></th>
<th>Ben</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Exhaustivo</td>
<td>1.00%</td>
<td>10.00%</td>
<td>20.00%</td>
<td>50.00%</td>
<td></td>
</tr>
<tr>
<td>t_inst (seg)</td>
<td>352.69</td>
<td>9.5378</td>
<td>8.1567</td>
<td>13.0834</td>
<td>22.7303</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Saturno</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Exhaustivo</td>
<td>1.00%</td>
<td>10.00%</td>
<td>20.00%</td>
<td>50.00%</td>
<td></td>
</tr>
<tr>
<td>t_inst (seg)</td>
<td>105.77</td>
<td>30.6451</td>
<td>14.2651</td>
<td>25.0115</td>
<td>29.5082</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Pirineus</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Exhaustivo</td>
<td>1.00%</td>
<td>10.00%</td>
<td>20.00%</td>
<td>50.00%</td>
<td></td>
</tr>
<tr>
<td>t_inst (seg)</td>
<td>110.56</td>
<td>13.2863</td>
<td>14.2516</td>
<td>12.9708</td>
<td>17.1192</td>
</tr>
</tbody>
</table>

### OpenMP + MKL:

<table>
<thead>
<tr>
<th></th>
<th>Ben</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Exhaustivo</td>
<td>1.00%</td>
<td>10.00%</td>
<td>20.00%</td>
<td>50.00%</td>
<td></td>
</tr>
<tr>
<td>t_inst (seg)</td>
<td>1156.45</td>
<td>38.81</td>
<td>33.94</td>
<td>46.74</td>
<td>125.48</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Saturno</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Exhaustivo</td>
<td>1.00%</td>
<td>10.00%</td>
<td>20.00%</td>
<td>50.00%</td>
<td></td>
</tr>
<tr>
<td>t_inst (seg)</td>
<td>353.58</td>
<td>36.83</td>
<td>36.83</td>
<td>39.74</td>
<td>44.13</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Pirineus</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Exhaustivo</td>
<td>1.00%</td>
<td>10.00%</td>
<td>20.00%</td>
<td>50.00%</td>
<td></td>
</tr>
<tr>
<td>t_inst (seg)</td>
<td>676.49</td>
<td>20.51</td>
<td>18.30</td>
<td>15.22</td>
<td>28.45</td>
</tr>
</tbody>
</table>
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($S, \text{ParamIni}, \text{ThreadsIni}$)
while (not EndCondition($S, \text{ParamEnd}, \text{ThreadsEnd}$))
    $SS = \text{Select}(S, \text{ParamSel}, \text{ThreadsSel})$
    $SS1 = \text{Combine}(SS, \text{ParamCom}, \text{ThreadsCom})$
    $SS2 = \text{Improve}(SS1, \text{ParamImp}, \text{ThreadsImp})$
    $S = \text{Include}(SS2, \text{ParamInc}, \text{ThreadsInc})$

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)

```c
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(\textit{threads} − \textit{first} − \textit{level}(\text{MetaheurParam}))
    
    \#pragma omp parallel for
    loop in elements

    second–level(MetaheurParam, \textit{threads} − \textit{first} − \textit{level})

second–level(MetaheurParam, \textit{threads} − \textit{first} − \textit{level}) :
    omp_set_num_threads(\textit{threads} − \textit{second} − \textit{level}(\text{MetaheurParam}, \textit{threads} − \textit{first} − \textit{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
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 \( \text{INEIni} \), can be modelled:

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

And the improvement of a percentage of the initial elements \( \text{PEIIni} \) with an intensification (extension of the considered neighborhood) \( \text{IIEIni} \) is modeled:

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

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

- And the improvement of a percentage of the initial elements $PEIni$ with an intensification (extension of the considered neighborhood) $IEIni$ is modeled:

$$t_{2\text{-levels}} = \frac{k_i \cdot \frac{INEIni \cdot PEIni \cdot IEIni}{100}}{p_1} + k_{p,1} \cdot p_1 + k_{p,2} \cdot p_2$$  \hspace{1cm} (2)$$
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 $\text{INEIni}$, can be modelled:

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

And the improvement of a percentage of the initial elements $\text{PEIni}$ with an intensification (extension of the considered neighborhood) $\text{IIIni}$ is modeled:

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

- 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 ⇒

- 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 $INElNi = 20$: $k_g = 2.38 \cdot 10^{-3}$ and $k_p = 1.94 \cdot 10^{-4}$, all in seconds.

- For the two-level routine studied, with metaheuristic parameters $INElNi = 20$, $PElIni = 50$, $IElIni = 20$ 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 $INE_{ini}$, $PEI_{ini}$ and $IEI_{ini}$ when varying the number of threads in the improvement routine.
Execution. One-level routine

Initial generation of the reference set:

\[ p_{opt.} = \sqrt{\frac{k_g}{k_p}} \cdot INElNi = 3.50 \cdot \sqrt{INElNi} \]  

(3)

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

<table>
<thead>
<tr>
<th>( INElNi )</th>
<th>threads</th>
<th>speed-up</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>optimum</td>
<td>model</td>
</tr>
<tr>
<td>100</td>
<td>55</td>
<td>35</td>
</tr>
<tr>
<td>500</td>
<td>64</td>
<td>78</td>
</tr>
</tbody>
</table>
Execution. Two-levels routine

Improvement of the generated elements:

\[ p_{1,\text{opt.}} = 1.18 \cdot 10^{-1} \cdot \sqrt{\text{INE} n_i \cdot \text{PE} n_i \cdot \text{II} n_i} \]  

(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)

<table>
<thead>
<tr>
<th>INE (n_i)</th>
<th>PE (n_i)</th>
<th>II (n_i)</th>
<th>threads optimum</th>
<th>model</th>
<th>speed-up optimum</th>
<th>model</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>50</td>
<td>10</td>
<td>30</td>
<td>26</td>
<td>15</td>
<td>11</td>
</tr>
<tr>
<td>500</td>
<td>100</td>
<td>5</td>
<td>32</td>
<td>59</td>
<td>29</td>
<td>27</td>
</tr>
</tbody>
</table>
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
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:

  \[ \frac{t_s}{c + \frac{s_g}{c} g} + t_{sc} c + t_{sk} g \]

  - \(t_s\) 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:

  \[
  \frac{t_s}{c + s_g/cg} + t_{sc}c + t_{sk}g
  \]

  - \(t_s\) 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:
  \[
  \frac{t_s}{c + s_g/cg} + t_{sc}c + t_{sk}g
  \]
  - $t_s$ 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 \( n \mod n \) do
      trigonometric operations
    end for
  end for
  Summation of the trigonometric functions
  for \( k = 1 \) to \( n \mod n \) do
    trigonometric operations
  end for
end for
Apply the method of acceleration of Kummer
end for

\text{Cost } O(m \cdot n \cdot n \mod n)
Two-dimensional problem

Initialization: obtain and sort modes

\[
\text{for } i = 1 \text{ to } m \text{ do}
\]
\[
\quad \text{for } j = 1 \text{ to } n \text{ do}
\]
\[
\quad \quad \{\text{Spectral part}\}
\]
\[
\quad \quad \text{for } k = 1 \text{ to } n\text{mod} \text{ do}
\]
\[
\quad \quad \quad GF[i, j] + = \text{spectral}(k)
\]
\[
\quad \text{end for}
\]
\[
\quad \{\text{Spatial part}\}
\]
\[
\quad \{\text{For images in axes } x \text{ and } y\}
\]
\[
\quad \text{for } r = -m\text{imag} \text{ to } m\text{imag} \text{ do}
\]
\[
\quad \quad \text{for } s = -n\text{imag} \text{ to } n\text{imag} \text{ do}
\]
\[
\quad \quad \quad GF[i, j] + = \text{spatial}(r, s)
\]
\[
\quad \quad \text{end for}
\]
\[
\quad \text{end for}
\]
\[
\text{end for}
\]

Cost $O(m \cdot n \cdot m\text{imag} \cdot n\text{imag})$
One-dimensional implementations

- **1D-OMP-FG**: A fine grain version with OpenMP the calculation. The two innermost loops are parallelized.

- **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

The chart illustrates the speed-up performance of different algorithms under varying nmod-points. The algorithms compared include:

- 1D-OMP-FG
- 1D-OMP-CG
- 1D-CUDA
- 1D-OMP+CUDA

The x-axis represents nmod-points, while the y-axis represents speed-up. The performance varies across different nmod-points, with some algorithms showing a consistent increase in speed-up as the nmod-points increase.
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 auxiliary 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

- 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:

$$Y = YB^T + X\Gamma^T + u$$

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

$$Y = YB^T + X\Gamma^T + u$$

where $Y \in \mathbb{R}^{d \times N}$, $X \in \mathbb{R}^{d \times K}$ and $u \in \mathbb{R}^{d \times N}$ are matrices with $N$ endogenous variables, $K$ exogenous variables and $N$ white noise variables respectively, being $d$ the sample size, and elements $B_{ii} = 0$.

- Solving a SEM is equivalent to obtaining $B$ and $\Gamma$, from a representative sample of the model (a set of values of the data variables $X$ and $Y$) 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:

$$Y = YB^T + X\Gamma^T + u$$

where $Y \in \mathbb{R}^{d \times N}$, $X \in \mathbb{R}^{d \times K}$ and $u \in \mathbb{R}^{d \times N}$ are matrices with $N$ endogenous variables, $K$ exogenous variables and $N$ white noise variables respectively, being $d$ the sample size, and elements $B_{ii} = 0$.

- Solving a SEM is equivalent to obtaining $B$ and $\Gamma$, from a representative sample of the model (a set of values of the data variables $X$ and $Y$) 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 \( \tilde{\mathbf{Y}} = \mathbf{Q}^T \mathbf{Y} \)

for \( i=1\ldots N \) do

\( i \)-th equation is identified (i.e. it can be solved)\( \textbf{then} \)

\( [\mathbf{R}_{i,1} | \tilde{\mathbf{Y}}_{i,1}] \leftarrow \text{Select columns from } [\mathbf{R}_1 | \tilde{\mathbf{Y}}_1] \)

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

Solve \( \tilde{\mathbf{R}}_{i,1} \hat{\mathbf{\eta}}_i = \tilde{\mathbf{y}}_{i,1} \)

\( \textbf{end if} \)

\( \textbf{end for} \)
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 GPUs.
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.