Power-Aware Execution of Sparse and Dense Linear Algebra Libraries

Enrique S. Quintana-Ortí

quintana@icc.uji.es

Motivation

- Reduce energy consumption!
  - Costs over the lifetime of an HPC facility in the range of acquisition costs
  - Produces carbon dioxide, a risk for the health and the environment
  - Produces heat which reduces hardware reliability
  - It gave us a reason to meet here in nice Lausanne ;-)

Personal view

- Hardware features mechanisms and modes to save energy
- Software (scientific apps) are in general power oblivious

Motivation

- Reduce energy consumption!
  - Costs over the lifetime of an HPC facility in the range of acquisition costs
  - Produces carbon dioxide, a risk for the health and the environment
  - Produces heat which reduces hardware reliability
  - It gave us a reason to meet here in nice Lausanne ;-)

Personal view

- Hardware features mechanisms and modes to save energy
- Software (scientific apps) are in general power oblivious

Outline

- **Part 1.** Scheduling dense linear algebra kernels in multi-core processors
- **Part 2.** Dense linear algebra message-passing libraries for clusters
- **Part 3.** Sparse linear algebra kernels in multi-core and many-core processors

Part 1. Scheduling dense linear algebra kernels in multi-core processors

Goal:

To improve power-performance ratio via scheduling and DVFS

Jointly with:

- P. Alonso
  Universitat Politécnica de Valencia
- M. Dolz, F. Igual, R. Mayo
  Universitat Jaume I

Outline of Part 1

1. Introduction
2. Dense linear algebra operations
3. Slack Reduction Algorithm
4. Race-to-Idle Algorithm
5. Experimental results
6. Conclusions

1.1 Introduction

- **Scheduling tasks** of dense linear algebra algorithms
  - Examples: Cholesky, QR and LU factorizations
- **Energy saving tools** available for multi-core processors
  - Example: Dynamic Voltage and Frequency Scaling (DVFS)

Scheduling tasks + DVFS

Power-aware scheduling on multi-core processors

- **Our strategies**:
  - Reduce the frequency of cores that will execute non-critical tasks to decrease idle times without sacrifying total performance of the algorithm
  - Execute all tasks at highest frequency to "enjoy" longer inactive periods

Energy savings

1.1 Introduction

- **Scheduling tasks** of dense linear algebra algorithms
  - Examples: Cholesky, QR and LU factorizations
- **Energy saving tools** available for multi-core processors
  - Example: Dynamic Voltage and Frequency Scaling (DVFS)

Scheduling tasks + DVFS

$\Downarrow$

Power-aware scheduling on multi-core processors

- **Our strategies**:
  - Reduce the frequency of cores that will execute non-critical tasks to decrease idle times without sacrifying total performance of the algorithm
  - Execute all tasks at highest frequency to "enjoy" longer inactive periods

$\Downarrow$

Energy savings

1.2 Dense linear algebra operations

LU factorization

- Factor

$$A = LU,$$

  where $L/U \in \mathbb{R}^{n \times n}$ unit lower/upper triangular matrices

- For numerical stability, permutations are introduced to prevent operation with small pivot elements
- Two algorithms of LU factorization
  - LU with partial (row) pivoting (traditional version) and
  - LU with incremental pivoting

    "Rapid development of high-performance out-of-core solvers for electromagnetics"
    T. Joffrain, E. S. Quintana, R. van de Geijn
    State-of-the-Art in Scientific Computing – PARA 2004, Copenhaguen (Denmark), June 2004

    Later called "Tile LU factorization" or "Communication-Avoiding LU factorization with flat tree"

LU factorization with partial (row) pivoting

for $k = 1 : s$ **do**

$A_{k:s,k} = L_{k:s,k} \cdot U_{kk}$ — LU FACTORIZATION — $(s - k + \frac{2}{3})b^3$ flops

for $j = k + 1 : s$ **do**

$A_{kj} \leftarrow L_{kk}^{-1} \cdot A_{kj}$ — TRIANGULAR SOLVE — b^3 flops

$A_{k+1:s,j} \leftarrow A_{k+1:s,j} - A_{k+1:s,k} \cdot A_{kj}$ — MATRIX-MATRIX PRODUCT — $2(s - k)b^3$ flops

end for

end for

DAG with a matrix consisting of 3×3 blocks

LU factorization with partial (row) pivoting

for $k = 1 : s$ **do**

$A_{k:s,k} = L_{k:s,k} \cdot U_{kk}$ — LU FACTORIZATION — $(s - k + \frac{2}{3})b^3$ flops

for $j = k + 1 : s$ **do**

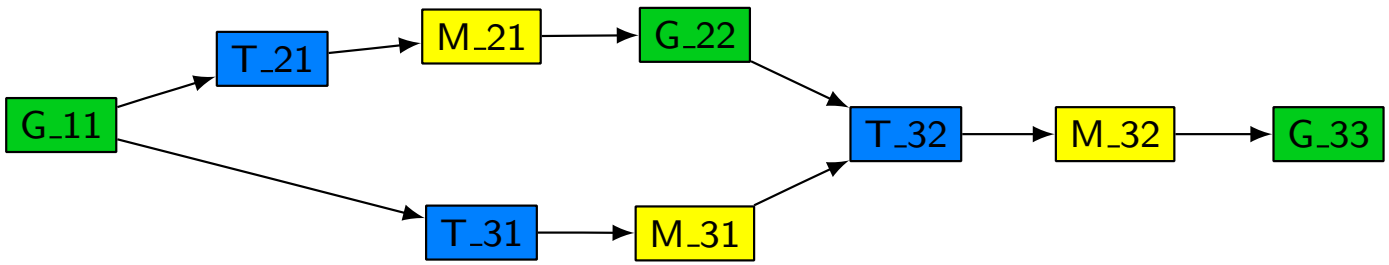
$A_{kj} \leftarrow L_{kk}^{-1} \cdot A_{kj}$ — TRIANGULAR SOLVE — b^3 flops

$A_{k+1:s,j} \leftarrow A_{k+1:s,j} - A_{k+1:s,k} \cdot A_{kj}$ — MATRIX-MATRIX PRODUCT — $2(s - k)b^3$ flops

end for

end for

DAG with a matrix consisting of 3×3 blocks



LU factorization with incremental pivoting

for $k = 1 : s$ **do**

$A_{kk} = L_{kk} \cdot U_{kk}$ — LU FACTORIZATION — $\frac{2b^3}{3}$ flops

for $j = k + 1 : s$ **do**

$A_{kj} \leftarrow L_{kk}^{-1} \cdot A_{kj}$ — TRIANGULAR SOLVE — b^3 flops

end for

for $i = k + 1 : s$ **do**

$$\begin{pmatrix} A_{kk} \\ A_{ik} \end{pmatrix} = \begin{pmatrix} L_{kk} \\ L_{ik} \end{pmatrix} \cdot U_{ik}$$ — 2×1 LU FACTORIZATION — b^3 flops

for $j = k + 1 : s$ **do**

$$\begin{pmatrix} A_{kj} \\ A_{ij} \end{pmatrix} \leftarrow \begin{pmatrix} L_{kk} & 0 \\ L_{ik} & I \end{pmatrix}^{-1} \cdot \begin{pmatrix} A_{kj} \\ A_{ij} \end{pmatrix}$$ — 2×1 TRIANGULAR SOLVE — $\frac{b^3}{2}$ flops

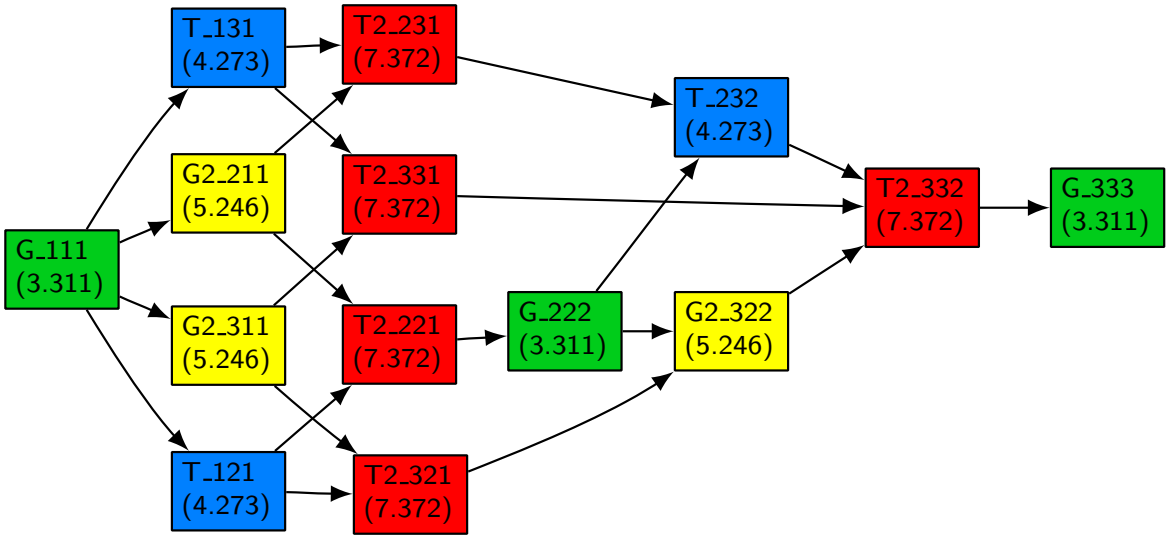
end for

end for

end for

LU factorization with incremental pivoting

DAG with a matrix consisting of 3×3 blocks



- Nodes contain execution time of tasks (in milliseconds, ms), for a block size $b = 256$ on a single-core of and AMD Opteron 6128 running at 2.00 GHz.
- We will use this info to illustrate our power-saving approach of the SRA!

1.3 Slack Reduction Algorithm

Stragegy

Obtain the dependency graph corresponding to the computation of a dense linear algebra algorithm; apply the Critical Path Method to analize slacks; and reduce them with our Slack Reduction Algorithm

The Critical Path Method:

- **DAG of dependencies**
  - Nodes $\Rightarrow$ Tasks
  - Edges $\Rightarrow$ Dependencies
- **Times**: Early and latest times to start and finalize execution of task T_i with cost C_i
- **Total slack**: Amount of time that a task can be delayed without increasing the total execution time of the algorithm
- **Critical path**: Formed by a succession of tasks, from initial to final node of the graph, with total slack $= 0$

1.3 Slack Reduction Algorithm

Stragegy

Obtain the dependency graph corresponding to the computation of a dense linear algebra algorithm; apply the Critical Path Method to analize slacks; and reduce them with our Slack Reduction Algorithm

The Critical Path Method:

- **DAG of dependencies**
  - Nodes $\Rightarrow$ Tasks
  - Edges $\Rightarrow$ Dependencies
- **Times**: Early and latest times to start and finalize execution of task T_i with cost C_i
- **Total slack**: Amount of time that a task can be delayed without increasing the total execution time of the algorithm
- **Critical path**: Formed by a succession of tasks, from initial to final node of the graph, with total slack $= 0$

Application of CPM to the DAG of the LU factorization with incremental pivoting of a matrix consisting of 3×3 blocks:

Task	C	ES	LF	S
G_111	3.311	0.000	3.311	0
T_121	4.273	3.311	8.558	0.973
G2_211	5.246	3.311	8.558	0
G2_311	5.246	3.311	11.869	3.311
T_131	4.273	3.311	12.842	5.257
T2_321	7.372	8.558	19.241	3.311
G2_322	5.246	19.241	24.488	0
T2_332	7.373	24.488	31.861	0
G_333	3.311	31.861	35.171	0
T2_331	7.372	8.558	24.488	8.558
T2_221	7.372	8.558	15.930	0
G_222	3.311	15.930	19.241	0
T_232	4.273	19.241	24.488	0.973
T2_231	7.372	8.558	20.214	4.284

Objective: tune the slack of those tasks with $S > 0$, reducing its execution frequency and yielding low power usage $\longrightarrow$ *Slack Reduction Algorithm*

Slack Reduction Algorithm

1. Frequency assignment
2. Critical subpath extraction
3. Slack reduction

1 Frequency assignment

Example: LU factorization with incremental pivoting of 3×3 blocks:

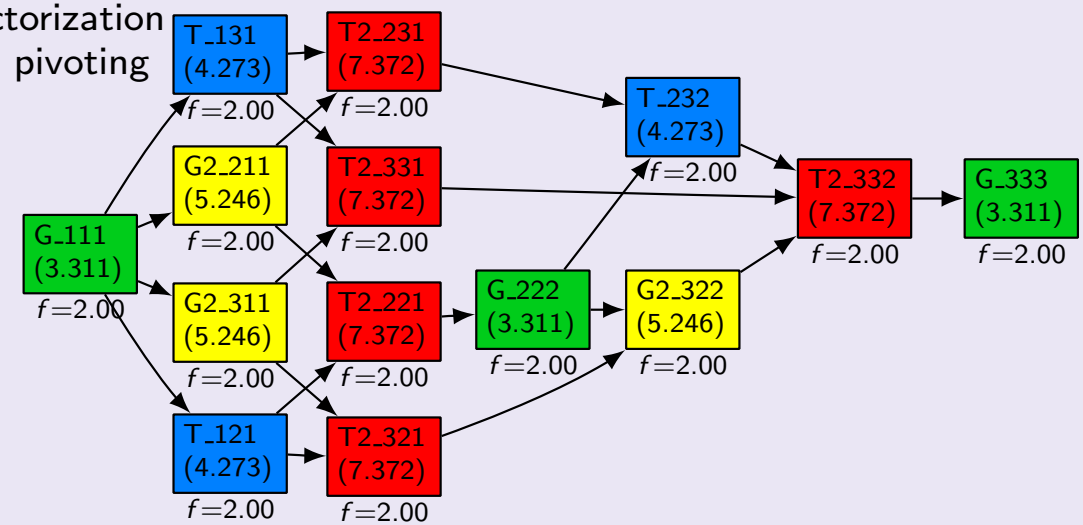
- Discrete collection of frequencies: {2.00, 1.50, 1.20, 1.00, 0.80} GHz
- We have obtained execution time of tasks running at each available frequency

Slack Reduction Algorithm

1. Frequency assignment
2. Critical subpath extraction
3. Slack reduction

1 Frequency assignment

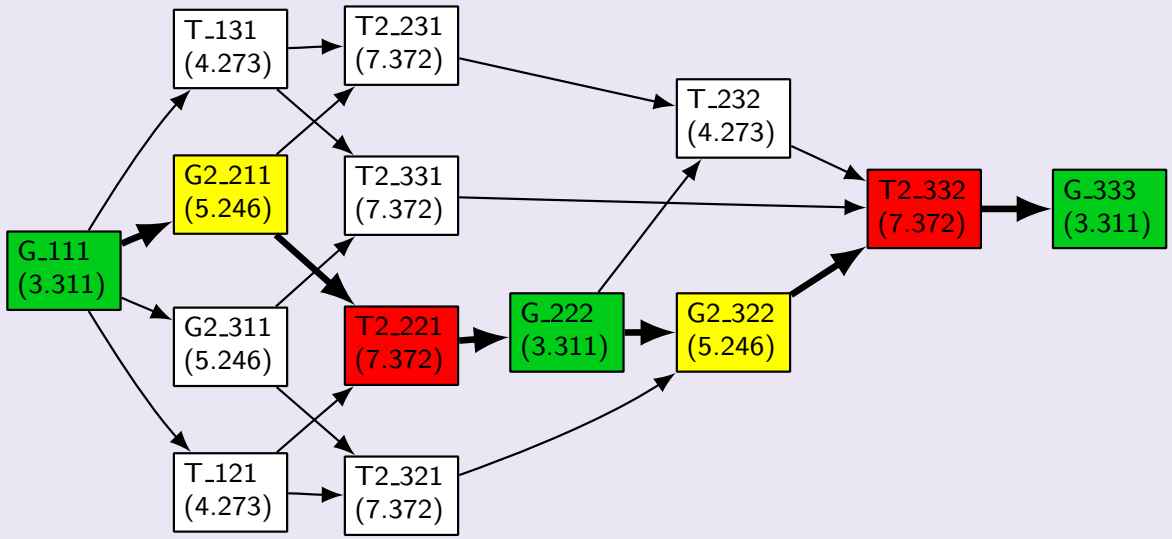
Example: LU factorization with incremental pivoting of 3×3 blocks:



- Discrete collection of frequencies: {2.00, 1.50, 1.20, 1.00, 0.80} GHz
- We have obtained execution time of tasks running at each available frequency

2 Critical subpath extraction

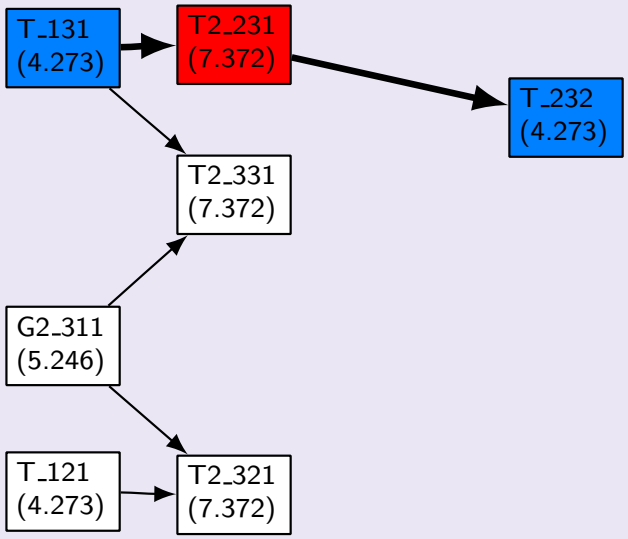
Iteration 0



CP_i	Tasks	Execution time
CP_0	{G_111, G2_211, T2_221, G_222, G2_322, T2_332, G_333}	35.171 ms

2 Critical subpath extraction

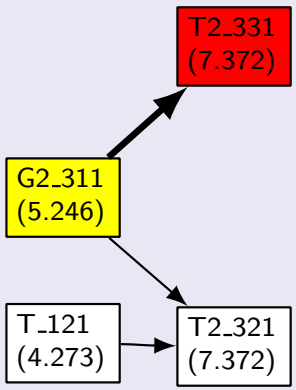
Iteration 1



CP_i	Tasks	Execution time
CP_0	{G_111, G2_211, T2_221, G_222, G2_322, T2_332, G_333}	35.171 ms
CP_1	{T_131, T2_231, T_232}	15.918 ms

2 Critical subpath extraction

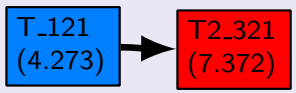
Iteration 2



CP_i	Tasks	Execution time
CP_0	{G_111, G2_211, T2_221, G_222, G2_322, T2_332, G_333}	35.171 ms
CP_1	{T_131, T2_231, T_232}	15.918 ms
CP_2	{G2_311, T2_331}	12.619 ms

2 Critical subpath extraction

Iteration 3

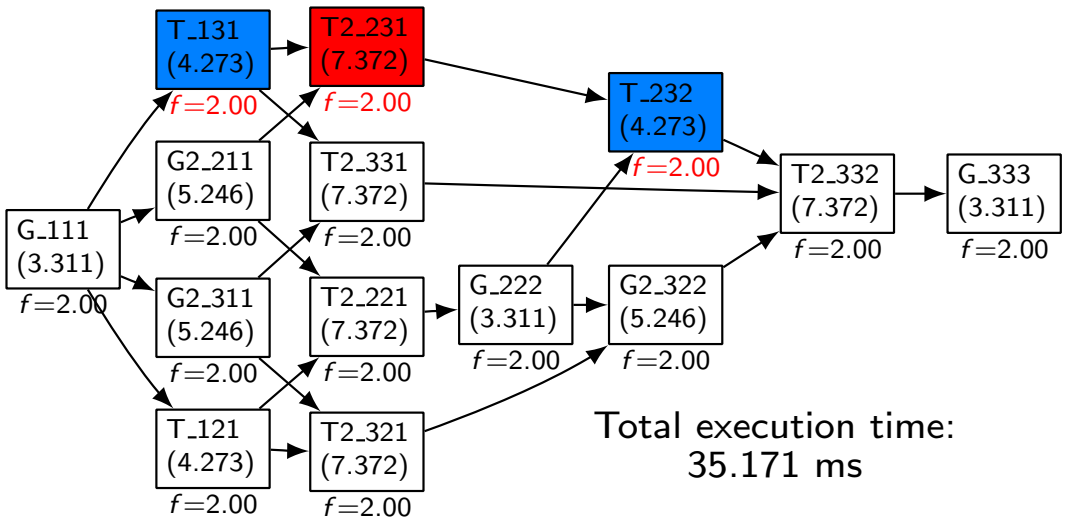


CP_i	Tasks	Execution time
CP_0	{G_111, G2_211, T2_221, G_222, G2_322, T2_332, G_333}	35.171 ms
CP_1	{T_131, T2_231, T_232}	15.918 ms
CP_2	{G2_311, T2_331}	12.619 ms
CP_3	{T_121, T2_321}	11.646 ms

Iteration 1

Process critical subpath $CP_1 = \{\texttt{T\_131}, \texttt{T2\_231}, \texttt{T\_232}\}$:

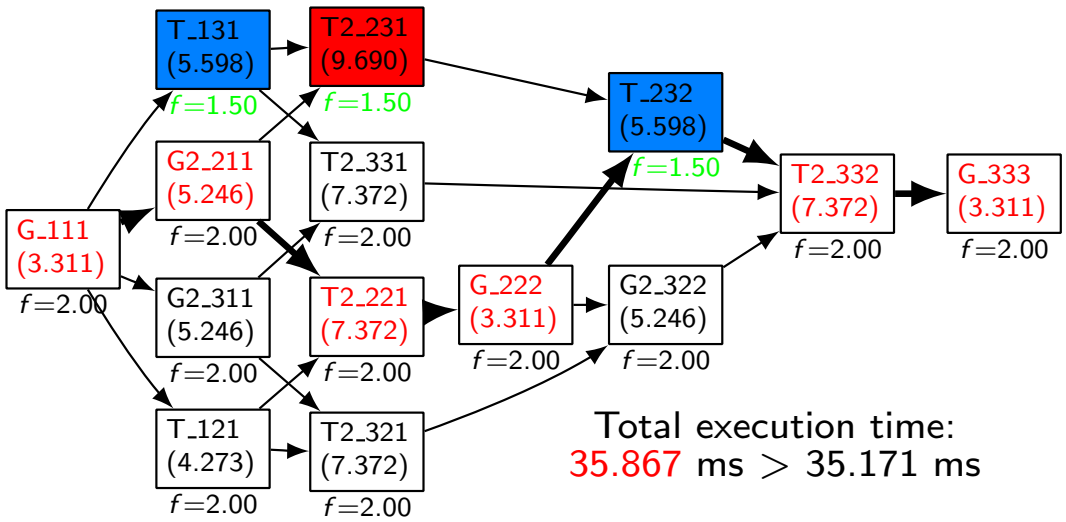
1. Increase ratio for CP_1: $\frac{d(\texttt{G\_111} \rightsquigarrow \texttt{T\_232}) - d(\texttt{G\_111} \rightsquigarrow \texttt{T\_131})}{l(CP_1)} = \frac{21{,}176}{15{,}919} = 1{,}33\,\%$
2. Slack is reduced by reducing execution frequency of task:
   - T_131: 2.00 GHz ⇒ 1.50 GHz; 4.273 ms ⇒ 5.598 ms;
   - T2_231: 2.00 GHz ⇒ 1.50 GHz; 7.372 ms ⇒ 9.690 ms;
   - T_232: 2.00 GHz ⇒ 1.50 GHz; 4.273 ms ⇒ 5.598 ms;



Iteration 1

Process critical subpath $CP_1 = \{\texttt{T\_131}, \texttt{T2\_231}, \texttt{T\_232}\}$:

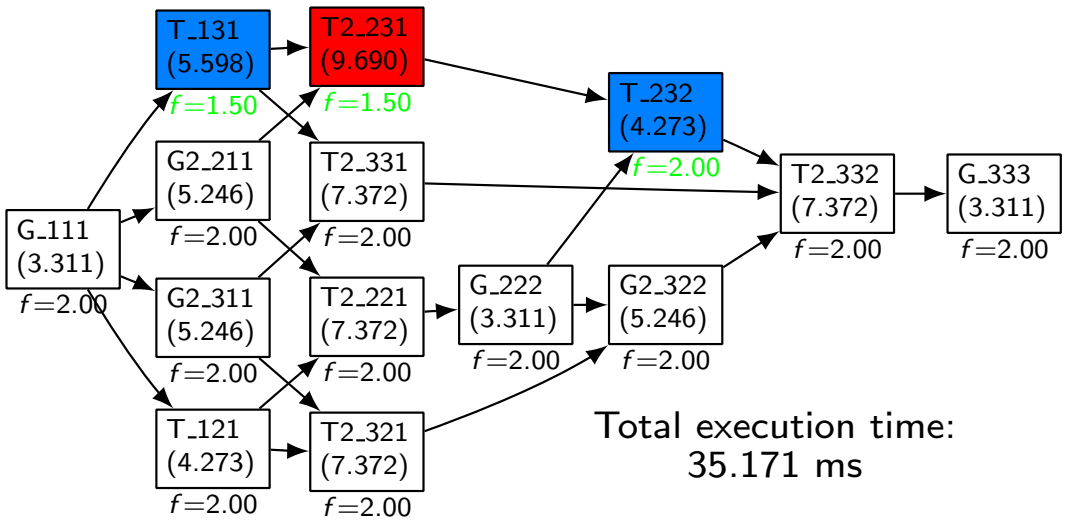
1. Increase ratio for CP_1: $\frac{d(\texttt{G\_111} \rightsquigarrow \texttt{T\_232}) - d(\texttt{G\_111} \rightsquigarrow \texttt{T\_131})}{l(CP_1)} = \frac{21{,}176}{15{,}919} = 1{,}33\,\%$
2. Slack is reduced by reducing execution frequency of task:
   - T_131: 2.00 GHz ⇒ 1.50 GHz; 4.273 ms ⇒ 5.598 ms;
   - T2_231: 2.00 GHz ⇒ 1.50 GHz; 7.372 ms ⇒ 9.690 ms;
   - T_232: 2.00 GHz ⇒ 1.50 GHz; 4.273 ms ⇒ 5.598 ms;



Iteration 1

Process critical subpath $CP_1 = \{\texttt{T\_131}, \texttt{T2\_231}, \texttt{T\_232}\}$:

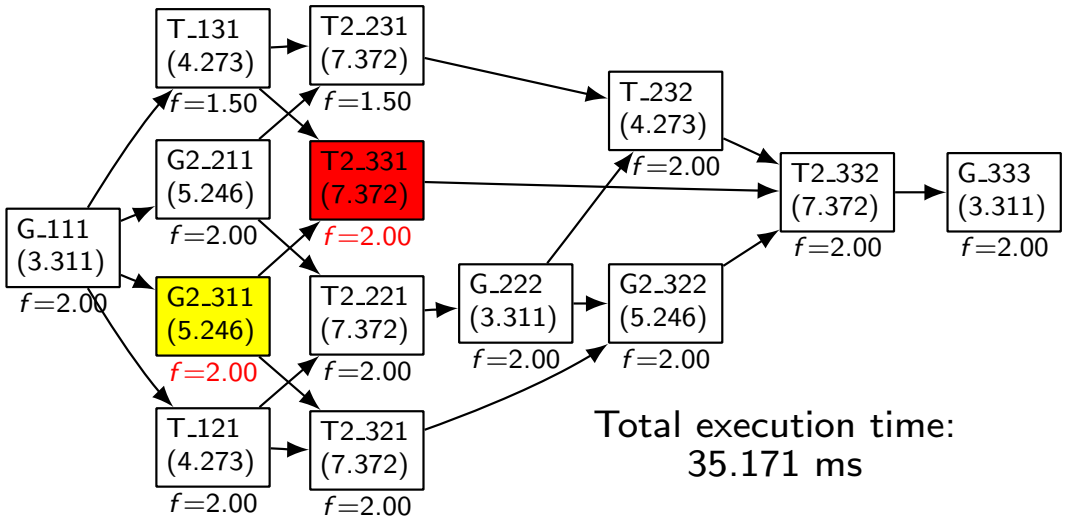
1. Increase ratio for CP_1: $\frac{d(\texttt{G\_111} \rightsquigarrow \texttt{T\_232}) - d(\texttt{G\_111} \rightsquigarrow \texttt{T\_131})}{l(CP_1)} = \frac{21{,}176}{15{,}919} = 1{,}33\,\%$
2. Slack is reduced by reducing execution frequency of task:
   - T_131: 2.00 GHz ⇒ 1.50 GHz; 4.273 ms ⇒ 5.598 ms;
   - T2_231: 2.00 GHz ⇒ 1.50 GHz; 7.372 ms ⇒ 9.690 ms;
   - T_232: 2.00 GHz ⇒ 1.50 GHz 2.00 GHz; 4.273 ms ⇒ 5.598 ms 4.273 ms;



Iteration 2

Process critical subpath $CP_2 = \{\texttt{G2\_311}, \texttt{T2\_331}\}$:

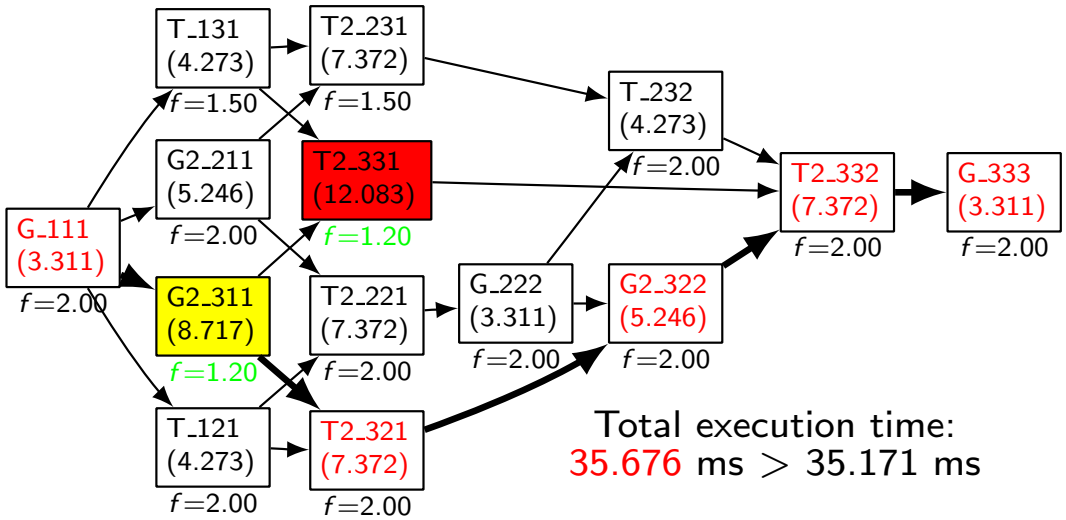
1. Increase ratio for CP_2: $\frac{d(\texttt{G\_111} \rightsquigarrow \texttt{T2\_331}) - d(\texttt{G\_111} \rightsquigarrow \texttt{G2\_311})}{l(CP_2)} = \frac{21{,}176}{12{,}619} = 1{,}67\,\%$
2. Slack is reduced by reducing execution frequency of task:
   - G2_311: 2.00 GHz ⇒ 1.20 GHz; 5.246 ms ⇒ 8.717 ms;
   - T2_331: 2.00 GHz ⇒ 1.20 GHz; 7.372 ms ⇒ 12.083 ms;



Iteration 2

Process critical subpath $CP_2 = \{\texttt{G2\_311}, \texttt{T2\_331}\}$:

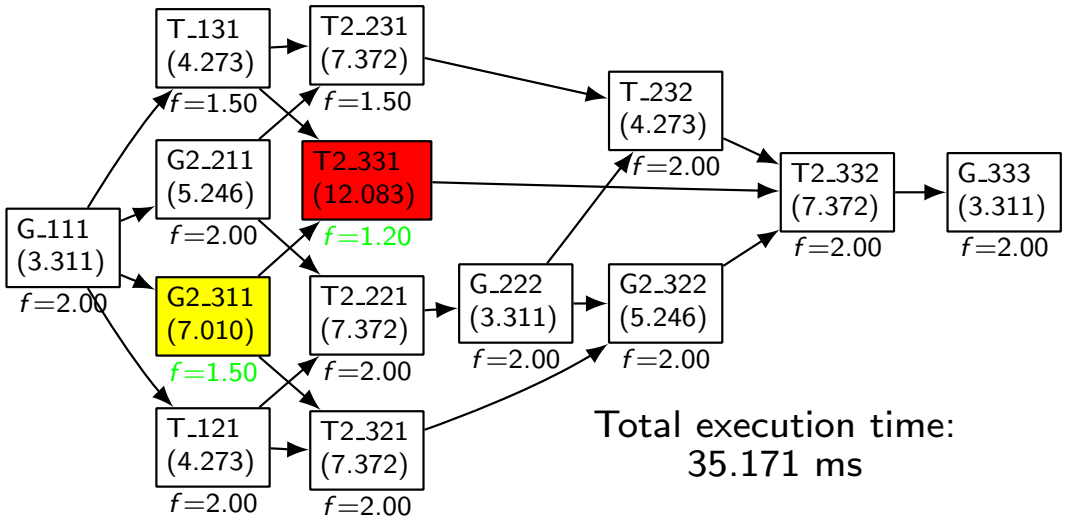
1. Increase ratio for CP_2: $\frac{d(\texttt{G\_111} \rightsquigarrow \texttt{T2\_331}) - d(\texttt{G\_111} \rightsquigarrow \texttt{G2\_311})}{l(CP_2)} = \frac{21{,}176}{12{,}619} = 1{,}67\,\%$
2. Slack is reduced by reducing execution frequency of task:
   - G2_311: 2.00 GHz ⇒ 1.20 GHz; 5.246 ms ⇒ 8.717 ms;
   - T2_331: 2.00 GHz ⇒ 1.20 GHz; 7.372 ms ⇒ 12.083 ms;



Iteration 2

Process critical subpath $CP_2 = \{\texttt{G2\_311}, \texttt{T2\_331}\}$:

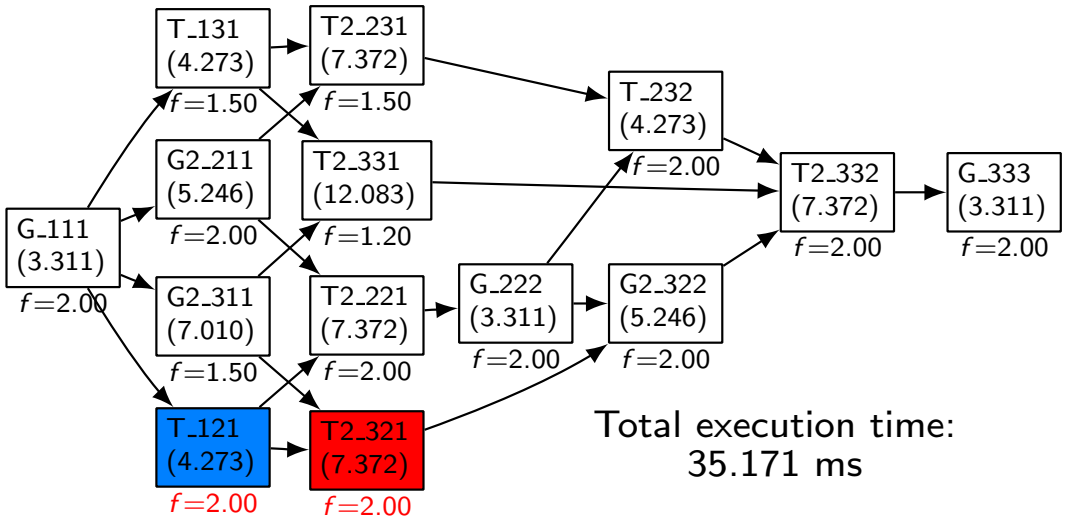
1. Increase ratio for CP_2: $\frac{d(\texttt{G\_111} \rightsquigarrow \texttt{T2\_331}) - d(\texttt{G\_111} \rightsquigarrow \texttt{G2\_311})}{l(CP_2)} = \frac{21{,}176}{12{,}619} = 1{,}67\,\%$
2. Slack is reduced by reducing execution frequency of task:
   - G2_311: 2.00 GHz ⇒ 1.20 GHz 1.50 GHz; 5.246 ms ⇒ 8.717 ms 7.010 ms;
   - T2_331: 2.00 GHz ⇒ 1.20 GHz; 7.372 ms ⇒ 12.083 ms;



Iteration 2

Process critical subpath $CP_3 = \{\texttt{T\_121}, \texttt{T2\_321}\}$:

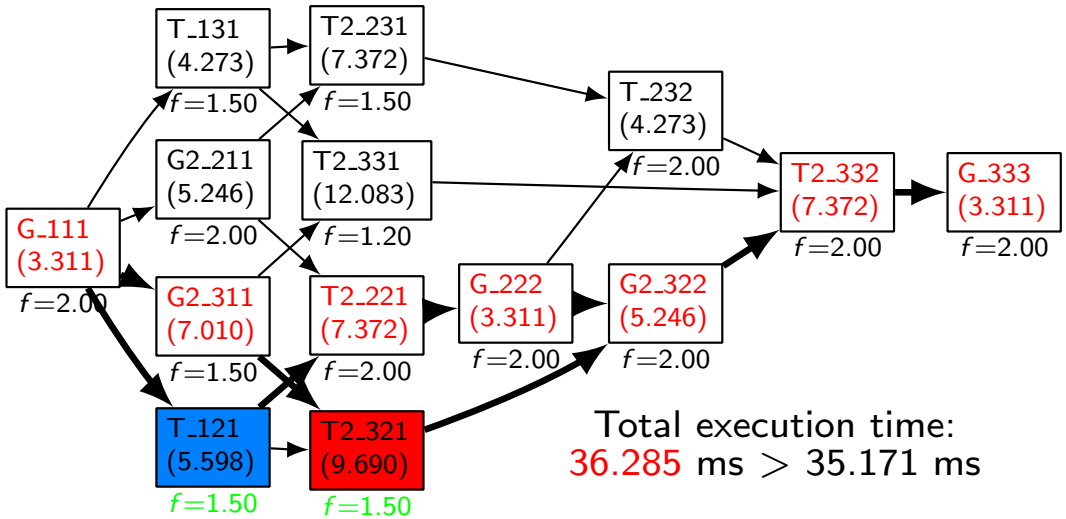
1. Increase ratio for CP_3: $\frac{d(\texttt{G\_111} \rightsquigarrow \texttt{T2\_321}) - d(\texttt{G\_111} \rightsquigarrow \texttt{T\_121})}{l(CP_3)} = \frac{15{,}930}{11{,}646} = 1{,}36\,\%$
2. Slack is reduced by reducing execution frequency of task:
   - T_121: 2.00 GHz ⇒ 1.50 GHz; 4.273 ms ⇒ 5.598 ms;
   - T2_321: 2.00 GHz ⇒ 1.50 GHz; 7.372 ms ⇒ 9.690 ms;



Iteration 2

Process critical subpath $CP_3 = \{\texttt{T\_121}, \texttt{T2\_321}\}$:

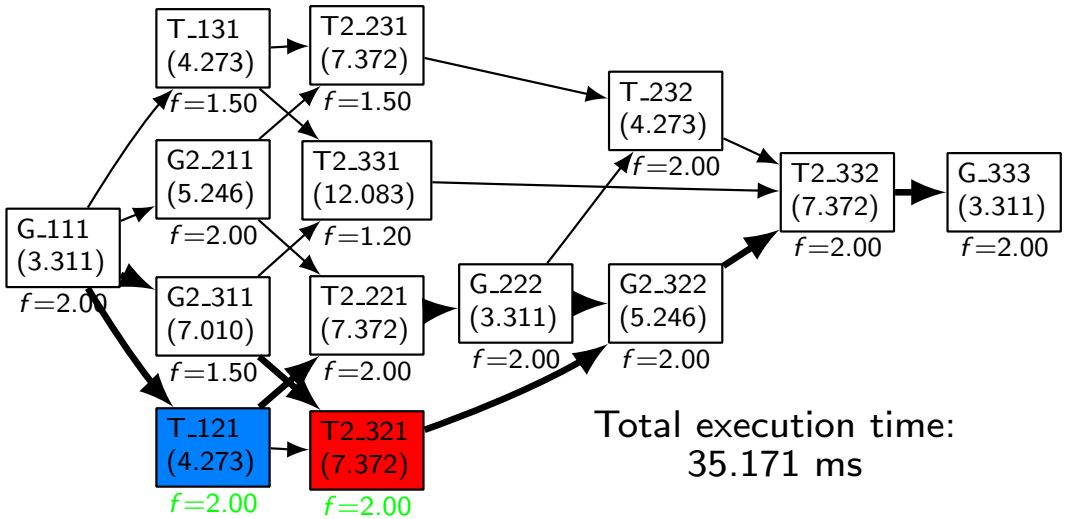
1. Increase ratio for CP_3: $\frac{d(\texttt{G\_111} \rightsquigarrow \texttt{T2\_321}) - d(\texttt{G\_111} \rightsquigarrow \texttt{T\_121})}{l(CP_3)} = \frac{15{,}930}{11{,}646} = 1{,}36\,\%$
2. Slack is reduced by reducing execution frequency of task:
   - T_121: 2.00 GHz ⇒ 1.50 GHz; 4.273 ms ⇒ 5.598 ms;
   - T2_321: 2.00 GHz ⇒ 1.50 GHz; 7.372 ms ⇒ 9.690 ms;



Iteration 2

Process critical subpath $CP_3 = \{\texttt{T\_121}, \texttt{T2\_321}\}$:

1. Increase ratio for CP_3: $\frac{d(\texttt{G\_111} \rightsquigarrow \texttt{T2\_321}) - d(\texttt{G\_111} \rightsquigarrow \texttt{T\_121})}{l(CP_3)} = \frac{15{,}930}{11{,}646} = 1{,}36\,\%$
2. Slack is reduced by reducing execution frequency of task:
   - T_121: 2.00 GHz ⇒ 1.50 GHz 2.00 GHz; 4.273 ms ⇒ 5.598 ms 4.273 ms;
   - T2_321: 2.00 GHz ⇒ 1.50 GHz 2.00 GHz; 7.372 ms ⇒ 9.690 ms 7.372 ms;



1.4 Race-to-Idle Algorithm

Race-to-Idle $\Rightarrow$ complete execution as soon as possible by executing tasks of the algorithm at the highest frequency to "enjoy" longer inactive periods

- Alternative strategy to reduce power consumption
- DAG requires no processing, unlike SRA
- Tasks are executed at highest frequency, during idle periods CPU frequency is reduced at lowest possible
- Why?
  - Current processors are quite efficient at saving power when idle
  - Power of idle core is much smaller than power in working periods

1.5 Experimental results

We use a simulator to evaluate the performance of the two strategies

Input parameters:

- DAG capturing tasks and dependencies of a blocked algorithm and frequencies recommended by the Slack Reduction Algorithm and Race-to-Idle Algorithm
- A simple description of the target architecture:
  - Number of sockets (physical processors)
  - Number of cores per socket
- Discrete range of frequencies and its associated voltages
- Collection of real power for each combination of frequency idle/busy state per core
- The cost (overhead) required to perform frequency changes

Static priority list scheduler:

- Duration of tasks at each available frequency is known in advance
- Tasks that lie on critical path must be prioritized

Blocked algorithms:

- LU with partial/incremental pivoting
- Block size: $b = 256$
- Matrix size varies from 768 to 5,632
- Execution time of tasks on AMD Opteron 6128 (8 cores)
  - LU with incremental pivoting: tasks `G`, `T`, `G2` and `T2`
  - LU with partial (row) pivoting: Duration of tasks `G` and `M` depends on the iteration!

    We evaluate the time of 1 flop for each type of task; then, from the theoretical cost of the task we obtain an approximation of its execution time

- Environment setup
  - AMD Opteron 6128 (1 socket of 8 cores)
  - Discrete range of frequencies: {2.00, 1.50, 1.20, 1.00, 0.80} GHz
  - Power required by the tasks: we measure the power running p copies of DGEMM at different frequencies:

Core	Frequency-Running/Idle								Power (W)
	1	2	3	4	5	6	7	8	
	2.00-R	2.00-R	2.00-R	2.00-R	2.00-R	2.00-R	2.00-R	2.00-R	157.60
	2.00-R	2.00-R	2.00-R	2.00-R	2.00-R	2.00-R	2.00-R	1.50-R	156.86
					...				...
	1.20-R	1.20-R	1.00-R	1.00-R	1.00-R	0.80-R	0.80-I	0.80-I	113.45
	1.20-R	1.20-R	1.00-R	1.00-R	1.00-R	0.80-I	0.80-I	0.80-I	110.37
					...				...
	0.80-R	0.80-R	0.80-I	0.80-I	0.80-I	0.80-I	0.80-I	0.80-I	91.81
	0.80-R	0.80-I	0.80-I	0.80-I	0.80-I	0.80-I	0.80-I	0.80-I	88.58

We measure with an internal power meter (ASIC with 25 samples/sec)

  - Frequency change latency:

		Destination freq.				
		2.00	**1.50**	**1.20**	**1.00**	**0.80**
Source freq.	**2.00**	–	40.36	43.18	43.77	49.85
	1.50	302.5	–	50.98	54.00	58.19
	1.20	301.7	302.7	–	61.60	66.05
	1.00	297.4	302.3	306.0	–	74.70
	0.80	291.6	292.7	294.0	295.80	–

Evaluation $\Rightarrow$ In order to evaluate experimental results obtained with the simulator, we compare execution time and consumption with no policy and with SRA/RIA

Metrics:

Execution time

- $T_{SRA/RIA\ \ Policy}$
- $T_{No\ \ policy}$
- Impact of SRA/RIA on time

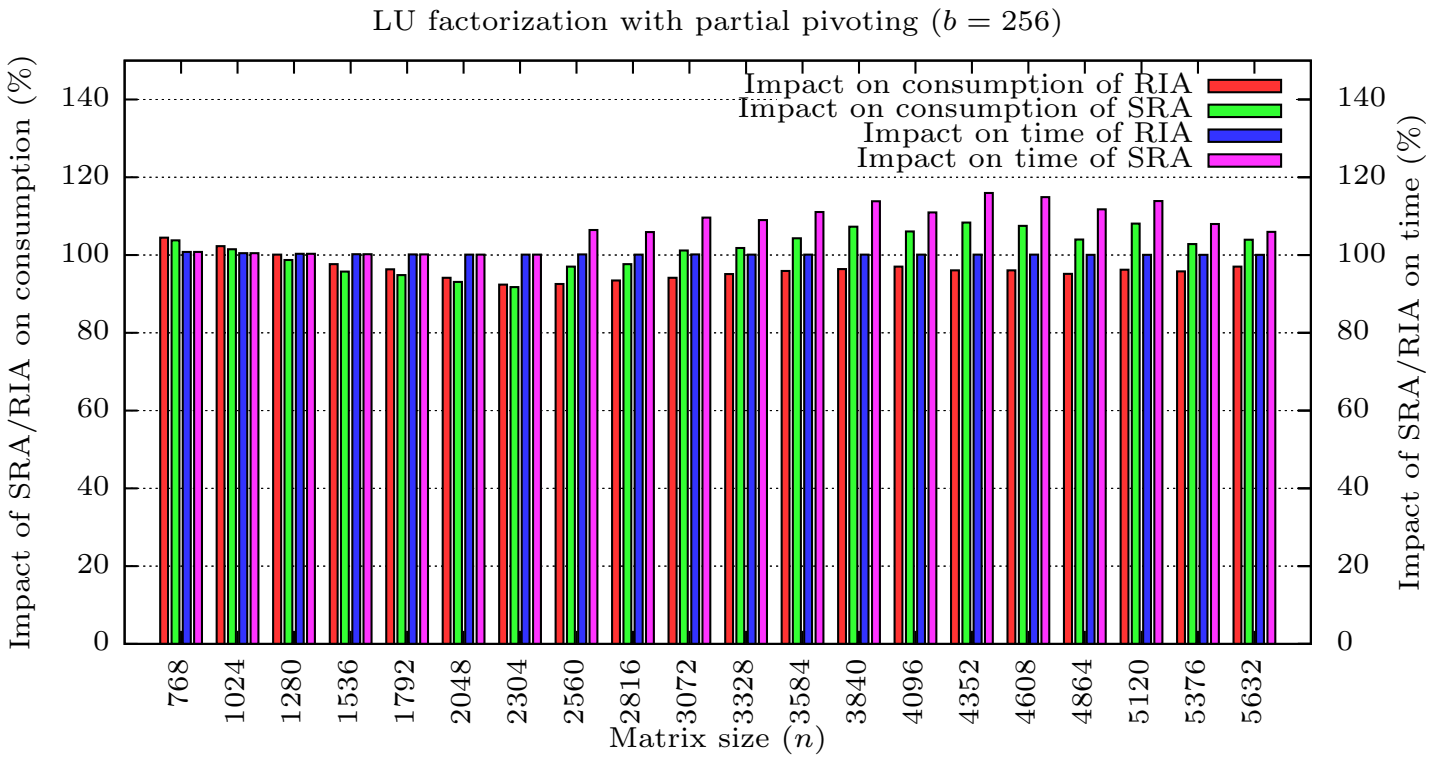
$$\%T_{SRA/RIA} = \frac{T_{SRA/RIA\ \ policy}}{T_{No\ \ policy}} \cdot 100$$

Consumption

- $C_{SRA/RIA\ \ policy} = \sum_{i=1}^{n} W_n \cdot T_n$
- $C_{No\ \ policy} = v^2\ \ T(f_{max})$
- Impact of SRA/RIA on consumption

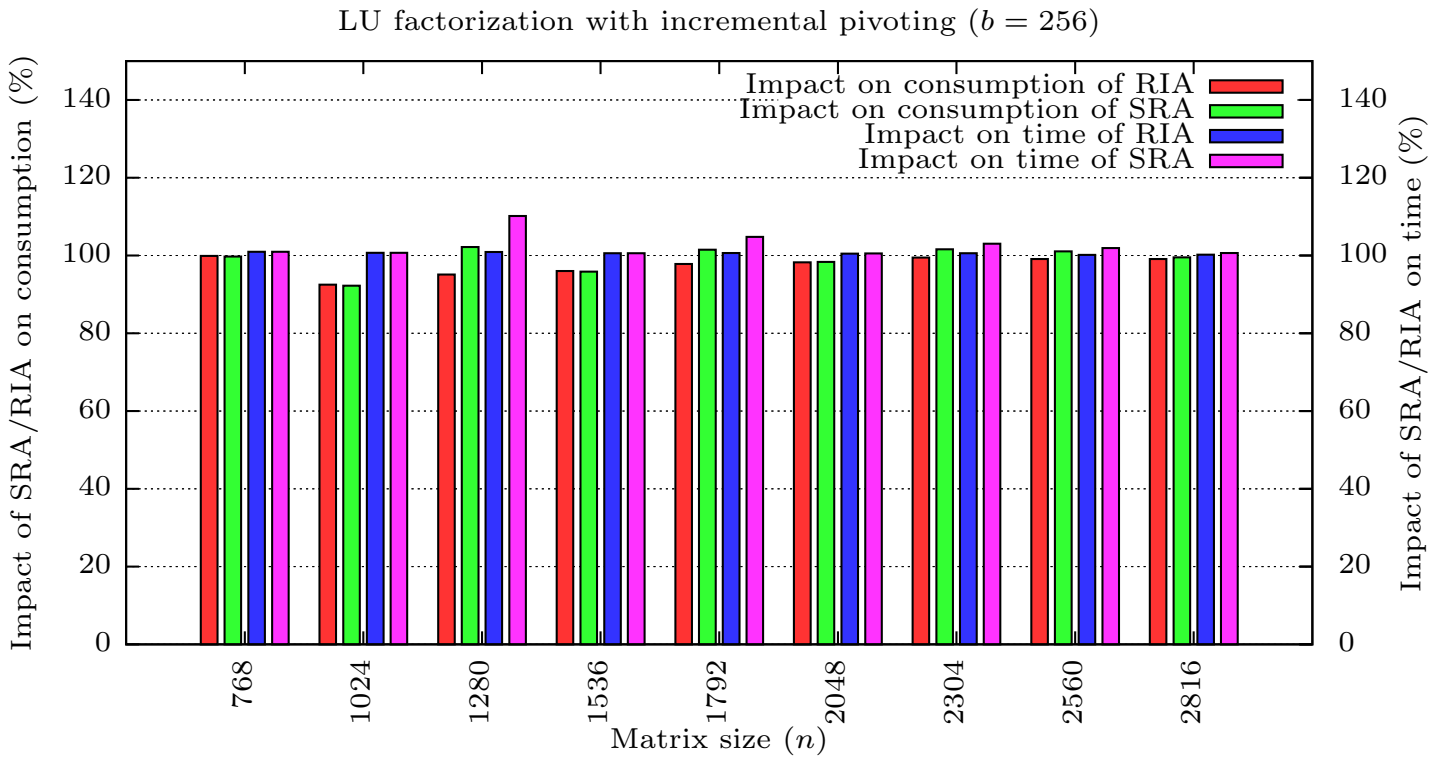
$$\%C_{SRA/RIA} = \frac{C_{SRA/RIA\ \ Policy}}{C_{No\ \ policy}} \cdot 100$$

Impact of the SRA/RIA on energy and time for the LU factorization with partial pivoting:



- SRA: Time is compromised and increases the consumption for largest problem sizes
  - The increase in execution time is due to the SRA being oblivious to the real resources
- RIA: Time is not compromised and consumption is maintained for largest problem sizes

Impact of the SRA/RIA on energy and time for the LU factorization with incremental pivoting:



- SRA: Yields higher execution time that produces an increase in power consumption
- RIA: Maintains execution time but reduces energy needs

1.6 Conclusions

Idea: Use of DVFS to save energy during the execution of dense linear algebra algorithms on multi-core architectures

Objective: To evaluate two alternative strategies to save energy consumption

Slack Reduction Algorithm

- DAG requires a processing
- Currently does not take into account number of resources
- Increases execution time when matrix size increases
- Increases, also, energy consumption

Race-to-Idle Algorithm

- DAG requires no processing
- Algorithm is applied on-the-fly
- Maintains in all of cases execution time
- Reduce energy consumption (around 5 %)

Results of dense linear algorithms: LU with partial/incremental pivoting

- Simulation under realistic conditions show that RIA produces more energy savings than SRA
- Current processors are quite good saving power when idle, so It's generally better to run as fast as possible to produce longer idle periods
- In our target platform (AMD Opteron 6128) RIA strategy is capable to produce more energy savings than SRA
- Power consumption:
  Working at highest frequency $>$ Working at lowest frequency $\ggg$ Idle at lowest frequency

More information

- "Improving power-efficiency of dense linear algebra algorithms on multi-core processors via slack control"
  P. Alonso, M. Dolz, R. Mayo, E. S. Quintana
  Workshop on Optimization Issues in Energy Efficient Distributed Systems – OPTIM 2011, Istanbul (Turkey), July 2011
- "DVFS-control techniques for dense linear algebra operations on multi-core processors"
  P. Alonso, M. F. Dolz, F. Igual, R. Mayo, E. S. Quintana
  2nd Int. Conf. on Energy-Aware High Performance Computing – EnaHPC 2011, Hamburg (Germany), Sept. 2011

Part 2. Dense linear algebra message-passing libraries for clusters

Goal:

To analyze the impact of power-saving strategies on the performance and power-consumption of mesage-passing dense linear algebra operations

Jointly with:

- M. Castillo, J. C. Fernández, R. Mayo, V. Roca
  Universitat Jaume I

Outline of Part 2

1. Target application
2. Experimental setup
3. Analysis of power consumption
   1. Multi-thread vs. multi-process
   2. DVFS (Linux governors)
   3. MPI communication mode
   4. Clusters of hw. accelerators
4. Conclusions

2.1 Target application

Generalized symmetric definite eigenproblems

- Solve

$$AX = BX\Lambda, \tag{1}$$

  with $A, B \in \mathbb{R}^{n \times n}$ given, $\Lambda \in \mathbb{R}^{n \times n}$ a diagonal matrix with the eigenvalues, and $X \in \mathbb{R}^{n \times n}$ the eigenvectors
- Collaboration with Structural Bioinformatics Research Group – CSIC, to analyze and model molecular structures: Solve (1) with $n \approx 300,000$
- First (and more expensive) step: Factorize the s.p.d. matrix

$$A = U^T U$$

  with $U \in \mathbb{R}^{n \times n}$ upper triangular

2.2 Experimental setup

Hardware platform

- IEEE double-precision arithmetic
- 9-node Linux cluster
  - Intel Xeon Quad-core E5520 (2.27 GHz)
  - 24 GB of DDR3 memory
  - NVIDIA Tesla C2050 (disconnected during CPU-only experiments)
  - PCI-Express (16×)
  - Standard HD, PSU, network card...
- Infiniband Mellanox switch/cards

Measurement setup

- 2× Watts-Up? .NET, connected to switch and node 0
- Sampling frequency of 1 Hz
- Power consumption of switch only varied slightly: 102.7–103 Watts: Discarded from results!

Message-passing kernels from ScaLAPACK

- Matrix-vector product (`PDGEMV`): memory-bounded operation/high concurrency
- Matrix-matrix product (`PDGEMM`): CPU-bounded operation/high concurrency
- Cholesky factorization (`PDPOTRF`): CPU-bounded operation/complex data dependencies

Basic libraries

- BLAS and LAPACK from GotoBLAS 1.11/CUBLAS 3.2.16
- MPI from MVAPICH2 1.5.1

2.3 Analysis of power consumption

Experiment #1.1

How to exploit core parallelism in clusters of multi-core processors?

1. 1 MPI rank per node combined with a multi-threaded BLAS
2. 1 MPI rank per core combined with a serial BLAS

3. For complex operations (e.g., Cholesky factorization), exploit task parallelism at node level via a run-time. See

   "Parallelizing dense matrix factorizations on clusters of multi-core processors using SMPSs"
   R. M. Badia, J. Labarta, V. Marjanovic, A. F. Martin, R. Mayo, E. S. Quintana-Ortí, R. Reyes
   Int. Conf. on Parallel Computing – ParCo 2011, Ghent (Belgium)

Experiment #1.2

What is the impact of DVFS (via Linux governors)?

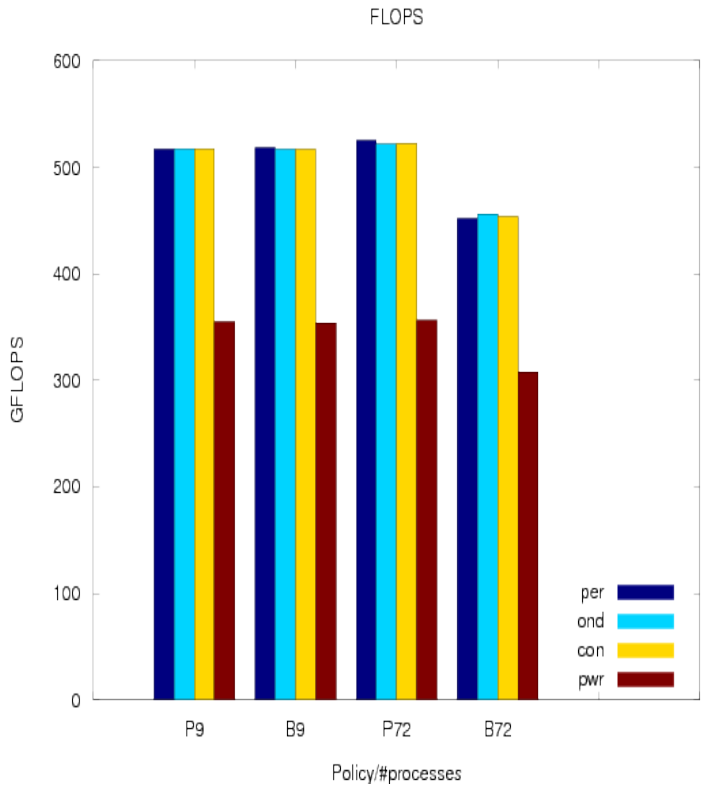
1. `Performance` (per). Static frequency at f_{max}
2. `Ondemand` (ond). Dynamic frequency with rapid increase, slow decrease
3. `Conservative` (con). Dynamic frequency with slow increase, slow decrease
4. `Powersave` (pwr). Static frequency at f_{min}

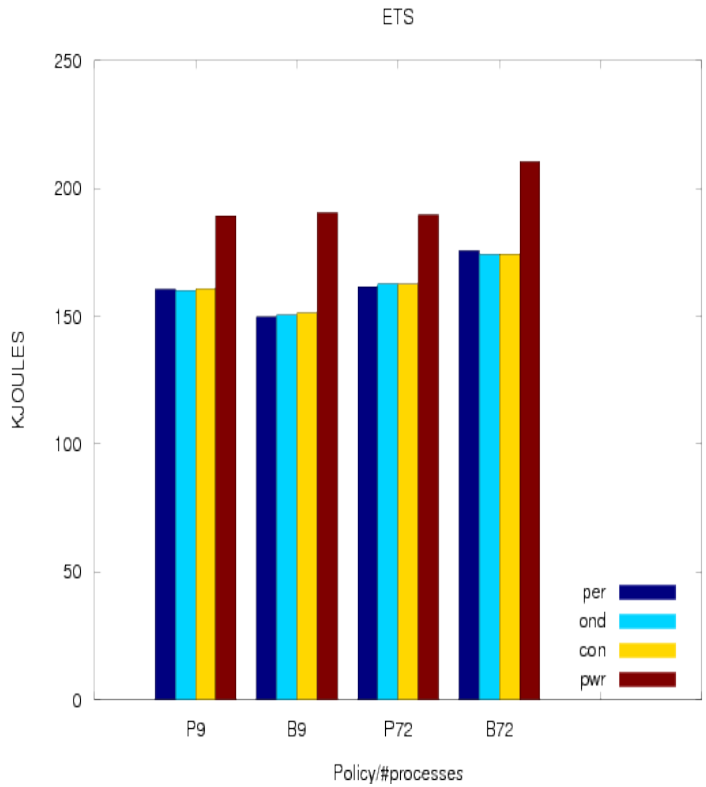
Experiment #1.3

What is the impact of the MPI operation mode?

1. Blocking primitives
2. Polling primitives

PDGEMM, 9 nodes, $m = n = k$=45,000

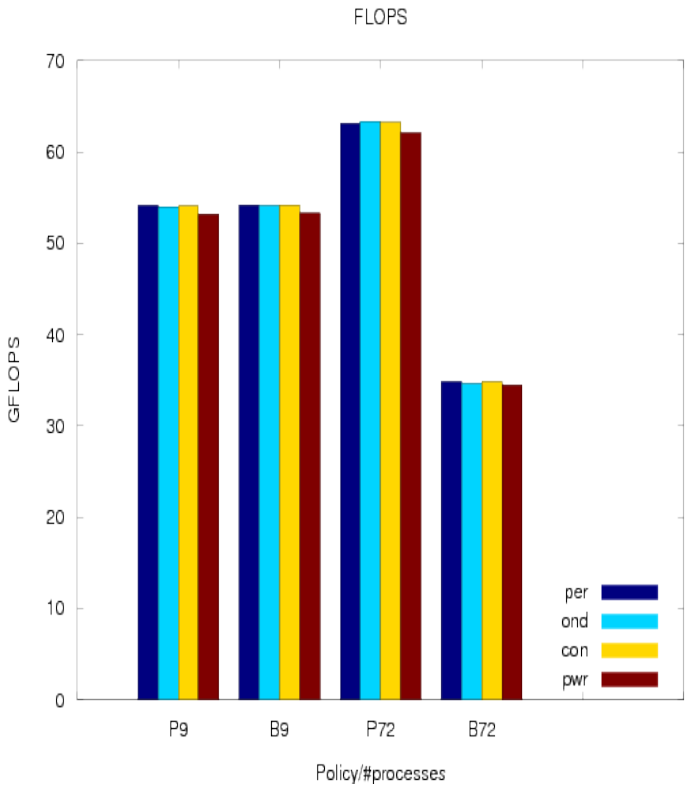


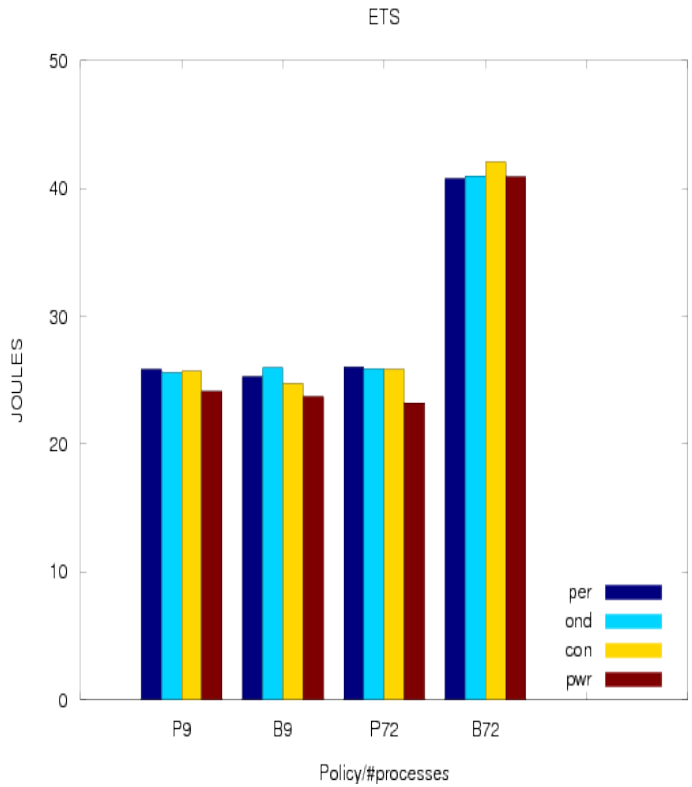


	Performance of PDGEMM (GFLOPS)			
	P9	B9	P72	B72
per	517.1	518.3	524.9	451.6
ond	517.2	517.2	521.8	456.3
con	517.3	517.2	522.6	453.9
pwr	354.7	354.1	356.2	308.1

	Power consumption of PDGEMM (KJoules)			
	P9	B9	P72	B72
per	160.5	149.7	161.7	175.7
ond	160.1	150.4	162.6	174.1
con	160.8	151.2	162.5	174.4
pwr	189.4	190.6	189.6	210.6

PDGEMV, 9 nodes, $m = n$=45,000

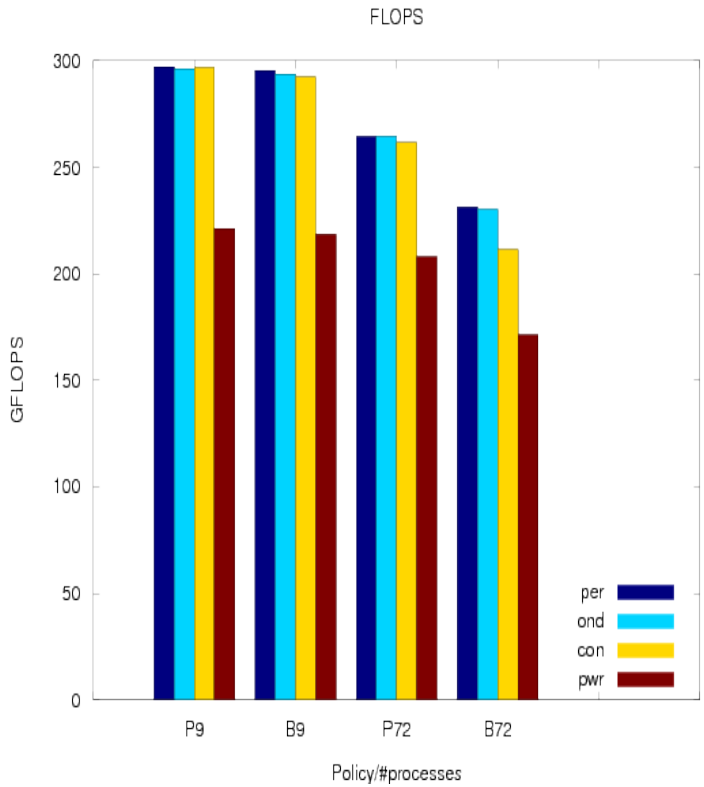


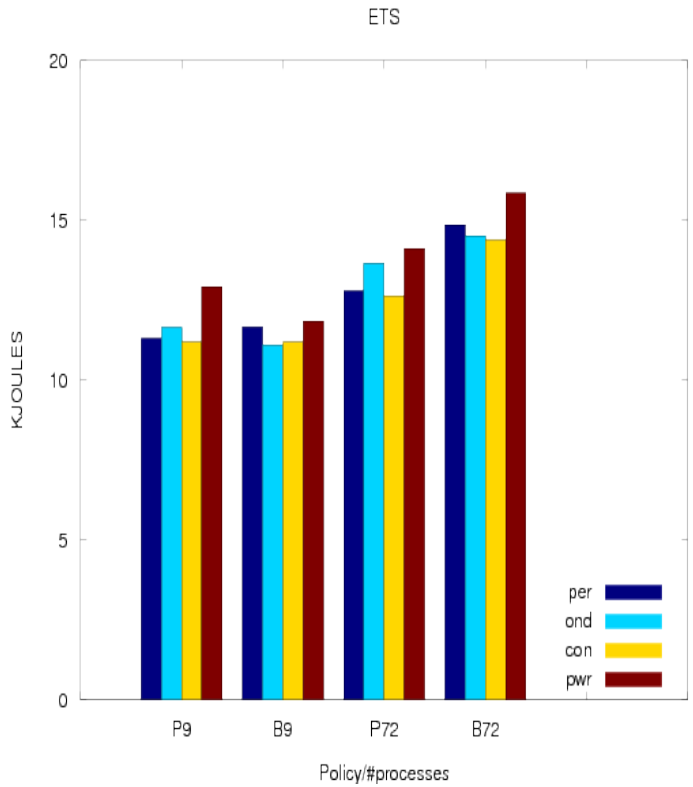


	Performance of PDGEMV (GFLOPS)			
	P9	B9	P72	B72
per	54.13	54.21	63.15	34.82
ond	54.05	54.14	63.41	34.64
con	54.09	54.17	63.27	34.86
pwr	53.21	53.29	62.13	34.48

	Power consumption of PDGEMV (Joules)			
	P9	B9	P72	B72
per	25.86	25.27	26.05	40.77
ond	25.61	25.99	25.92	41.03
con	25.70	24.75	25.86	42.09
pwr	24.12	23.72	23.24	40.92

PDPOTRF, 8 nodes, n=30,000





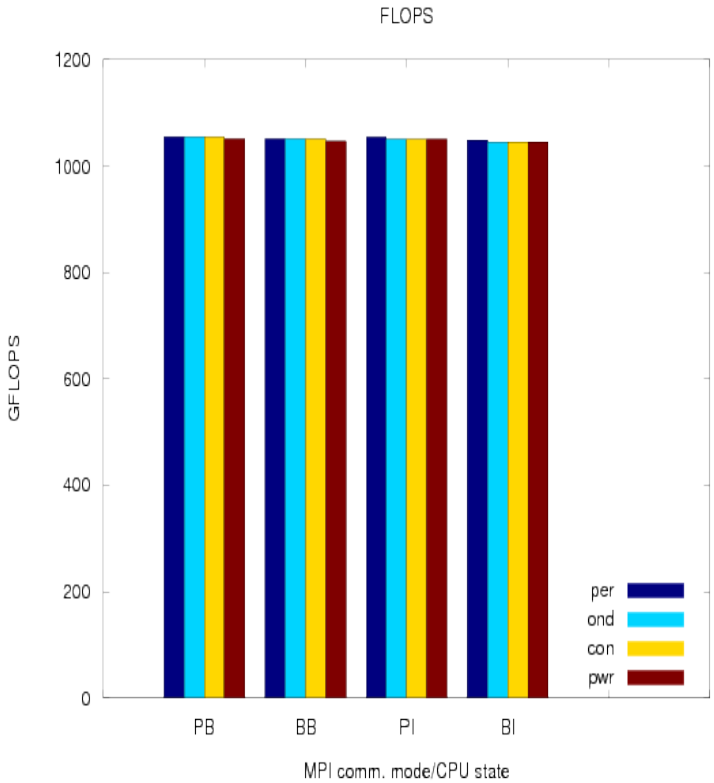
	Performance of PDPOTRF (GFLOPS)			
	P8	B8	P64	B64
per	297.3	295.5	264.5	231.3
ond	296.4	293.9	264.6	230.2
con	296.9	292.4	261.5	211.3
pwr	221.4	218.5	208.1	171.2

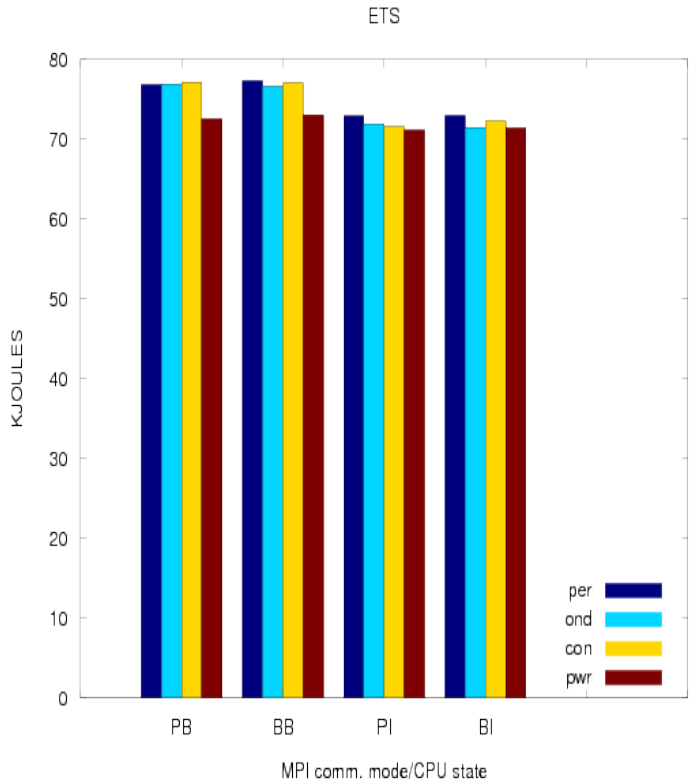
	Power consumption of PDPOTRF (KJoules)			
	P8	B8	P64	B64
per	11.30	11.67	12.79	14.86
ond	11.65	11.09	13.66	14.52
con	11.19	11.19	12.64	14.38
pwr	12.94	11.85	14.13	15.85

Experiment #2

What is the impact of the use of hardware accelerators (1 per node)?

1. Only Cholesky factorization (`PDPOTRF`)
2. Off-load matrix-matrix products to GPU
3. Polling/blocking (busy/idle) detection of GPU kernel termination





	Performance of GPDGEMM (GFLOPS)			
	PB	BB	PI	BI
per	1,055	1,050	1,054	1,049
ond	1,055	1,050	1,049	1,045
con	1,055	1,051	1,050	1,045
pwr	1,050	1,046	1,050	1,045

	Power consumption GPDGEMM (KJoules)			
	PB	BB	PI	BI
per	76.81	77.33	72.94	72.98
ond	76.95	76.62	71.93	71.56
con	77.14	77.07	71.63	72.27
pwr	72.52	73.06	71.15	71.45

2.4 Conclusions

Linux governors

- CPU-bounded `PDGEMM`: lower frequency implies higher execution time/energy consumption
- Memory-bounded `PDGEMV`: selecting the appropriate governor can render savings with no impact on performance
- CPU-bounded with complex dependencies `PDPOTRF`: delicate balance between power and performance

Communication mode

- Polling mode is beneficial with one process per core
- Savings of blocking mode lower than expected

Hybrid computing

- Idle-wait of multi-core processors saves energy

More information

- "Evaluation of the energy performance of dense linear algebra kernels on multi-core and many-core processors"
  M. Castillo, J. C. Fernández, R. Mayo, E. S. Quintana-Ortí, V. Roca
  7th Workshop on High-Performance, Power-Aware Computing – HPPAC 2011, Anchorage (Alaska, USA), May 2011
- "Strategies to save energy for message-passing dense linear algebra kernels"
  M. Castillo, J. C. Fernández, R. Mayo, E. S. Quintana-Ortí, V. Roca
  Tech. Report DICC, Universitat Jaume I, July 2011
  20th Euromicro Int. Conf. on Parallel, Distributed and Network-based Processing – PDP 2012, Garching (Germany). Submitted

Part 3. Sparse linear algebra kernels in multi-core and many-core processors

Goal:

To analyze in detail energy consumption in the execution of sparse linear system solvers (via iterative methods) on current platforms

Jointly with:

- M. Castillo, J. C. Fernández, R. Mayo
  Universitat Jaume I

- H. Anzt, V. Heuveline
  Karlsruhe Institute of Technology

Outline of Part 3

1. Target application
2. Experimental setup
3. Analysis of power consumption
4. DVFS
5. Idle-wait
6. Conclusions

3.1 Target application

Sparse linear systems

- Solve

$$Ax = b,$$

with A sparse and large, arise in many apps. that involve PDEs modeling physical, chemical or economical processes

- Low-cost iterative Krylov-based solvers for large-scale systems: A s.p.d. $\rightarrow$ Conjugate Gradient (CG), Preconditioned CG (PCG)

CG (Matlab)

```
1 function [x] = cg(A,b,x,tol)   % BLAS    SBLAS    Arch.
2                                %-------------------------
3   r=b-A*x;                     %         spmv     CPU/GPU
4   p=r;
5   rsold=r'*r;                  % dot               CPU
6
7   for i=1:size(A,1)
8     Ap=A*p;                    %         spmv     CPU/GPU
9     alpha=rsold/(p'*Ap);       % dot               CPU
10    x=x+alpha*p;               % axpy              CPU
11    r=r-alpha*Ap;              % axpy              CPU
12    rsnew=r'*r;                % dot               CPU
13    if sqrt(rsnew)<tol
14      break;
15    end
16    p=r+rsnew/rsold*p;         % axpy              CPU
17    rsold=rsnew;
18  end
19 end
```

3.2 Experimental setup

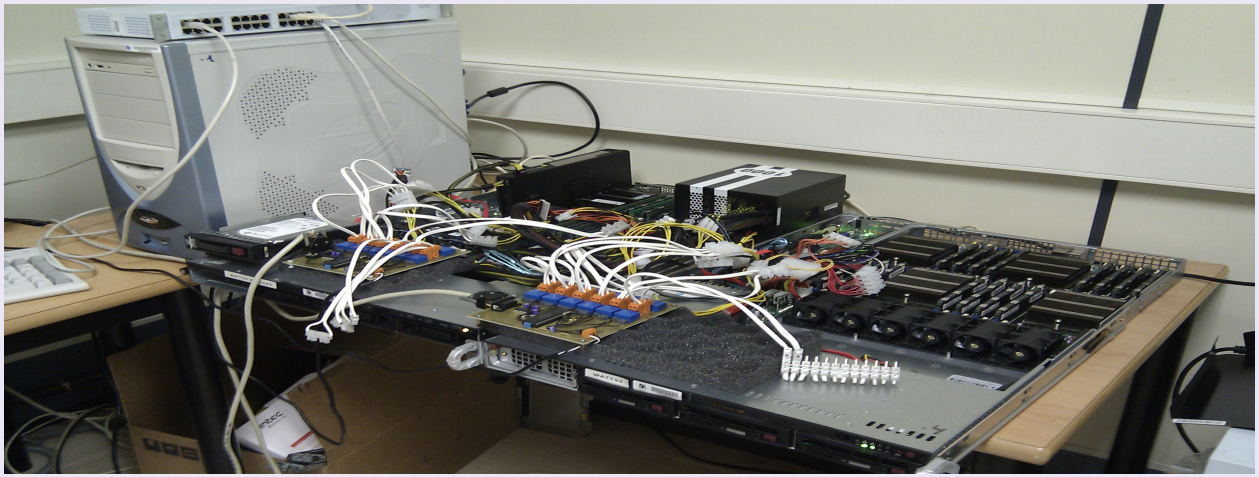
Hardware platform

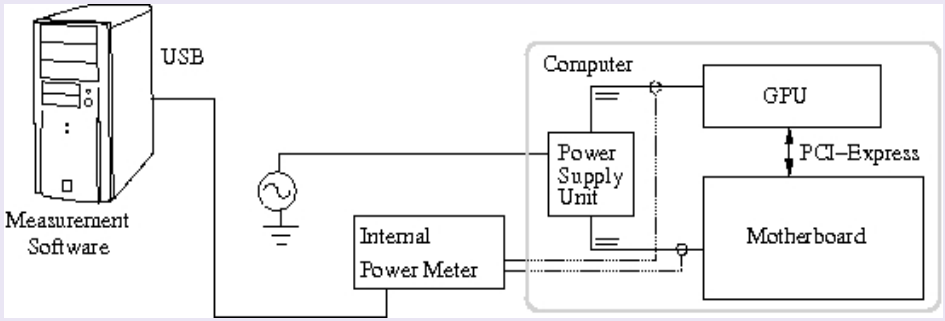
- AMD Opteron 6128 (8 cores)@2.0 GHz with 24 GBytes of RAM
- NVIDIA Tesla C1060 (240 cores). Disconnected during CPU-only experiments!
- PCI-Express (16×)

Software implementation of CG, PCG

- AMD: Intel MKL (11.1) for BLAS-1 and own implementation of `spmv`
- NVIDIA: CUBLAS (3.0) and implementation of `spmv` based on Garland and Bell's approach
- `gcc -O3` (4.4.3) and `nvcc` (3.2)

Measurement setup





- ASIC with sampling frequency of 25 Hz

Linear systems

Matrix name	Size (n)	Nonzeros (nnz)
A318	32,157,432	224,495,280
APACHE2	715,176	4,817,870
AUDIKW_1	943,695	77,651,847
BONES10	914,898	40,878,708
ECOLOGY2	999,999	4,995,991
G3_CIRCUIT	1,585,478	7,660,826
LDOOR	952,203	42,493,817
ND24K	72,000	28,715,634

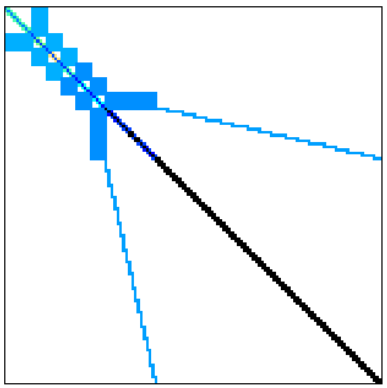
Solvers $Ax = b$

- Iterative: $x_0 \rightarrow x_1 \rightarrow x_2 \rightarrow \cdots \rightarrow x_n \approx x$
- Stopping criterion: $\varepsilon = 10^{-10} \|r_0\|_2$
- Initial solution: $x_0 \equiv 0$

3.3 Analysis of power consumption

Experiment #1

- Power consumption of CG and PCG on CPU (1T, 2T, 4T, 8T on 1, 2, 4, 8 cores) and hybrid CPU (4T)+GPU
- G3_CIRCUIT (moderate dimension, complex sparsity pattern)



CG method

Hardware	# iter	Time [s]	Energy consumption [Wh]		
			Chipset	GPU	Total
CPU 4T	21,424	1,076.97	42.18	-	42.18
GPU 4T	21,467	198.43	8.04	3.44	11.48

- Hybrid CPU-GPU code clearly outperforms CPU one in both performance ($5\times$) and energy ($4\times$)
- Energy gap mostly from reduction in execution time:

CPU 4 T	GPU 4 T
$\frac{42.18}{1,076.97} \cdot 3,600 = 140.0$ W	$\frac{11.48}{198.43} \cdot 3,600 = 208.2$ W

PCG method (Jacobi preconditioner)

Hardware	# iter	Time [s]	Energy consumption [Wh]		
			Chipset	GPU	Total
CPU 4T	4,613	348.79	13.31	-	13.31
GPU 4T	4,613	46.28	1.89	0.83	2.72

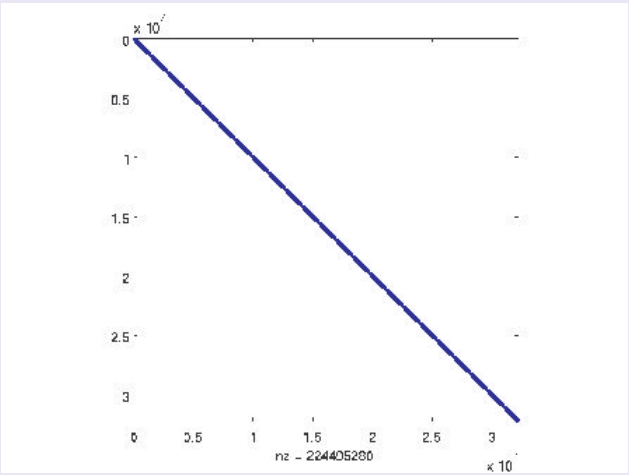
- Important reduction in #iterations: 21,424 $\rightarrow$ 4,613
- Time/iteration and energy/iteration not significantly increased (preconditioning this matrix only requires diagonal scaling):

CG GPU 4 T	PCG GPU 4 T
$\frac{198.43}{21,467} = 0.0092$ s/iter	$\frac{46.28}{4,613} = 0.0100$ s/iter
$\frac{11.48}{21,467} = 5.34 \cdot 10^{-4}$ Wh/iter	$\frac{2.72}{4,613} = 5.89 \cdot 10^{-4}$ Wh/iter

3.4 DVFS

Experiment #2

- In general, for memory-bounded operations a decrease of the processor operation frequency can yield energy savings
  - Memory-bounded or I/O-bounded?
  - Decreasing processor frequency impacts memory latency?
- The sparse matrix-vector product is indeed memory-bounded: $2nnz$ flops vs. nnz memops
- AMD Opteron 6128: 800 MHz – 2.0 GHz
- $\mathrm{A318}$ (large size to match powermeter sampling rate)



CG method

Hardware	Freq. [MHz]	Time [s]	Power/Energy consumption Chipset [Avg. W]	GPU [Avg. W]	Total [Wh]
CPU 4T	2,000	1441.78	123.99	-	49.66
CPU 4T	800	1674.62	108.11	-	50.29
GPU 4T	2,000	253.22	149.04	61.89	14.84
GPU 4T	800	254.25	138.50	61.45	14.12

- For the CPU solver, lowering the processor frequency increases the execution time, which blurs savings in power consumption
- For the hybrid CPU-GPU solver, as the computationally intensive parts are executed on the GPU, lowering the frequency yields some energy savings... Why not larger?

- GPU and CPU operate in asynchronous mode but... when the GPU is executing a kernel, and the CPU encounters a call to a second kernel, it enters into a *polling* loop
- In the polling state, the power usage of the CPU is almost as high as that of a fully-loaded processor!
- Alternatives:

  (i) Plain solver
  (ii) Solver + DVFS during GPU execution

Power-friendly CPU modes

CG method: Energy consumption of chipset+GPU

matrix	Energy consumption [Wh]		improvement [%]
	(i)	(ii)	(i)→(ii)
A318	14.84	14.12	5.1
APACHE2	1.98	1.99	-0.5
AUDIKW_1	no convergence		-
BONES10	no convergence		-
ECOLOGY2	2.30	2.27	-1.3
G3_CIRCUIT	11.48	11.11	3.3
LDOOR	no convergence		-
N24K	26.43	25.42	3.97

- A moderate gain, in some cases a loss...

PCG method: Energy consumption of chipset+GPU

matrix	Energy consumption [Wh]		improvement [%]
	(i)	(ii)	(i)→(ii)
A318	14.84	14.12	5.1
APACHE2	1.75	1.76	-0.6
AUDIKW_1	47.98	38.15	5.2
BONES10	157.32	150.16	4.8
ECOLOGY2	2.51	2.45	2.4
G3_CIRCUIT	2.71	2.38	3.0
LDOOR	43.22	41.18	5.0
N24K	34.62	32.97	5.0

- Moderate but more consistent gain

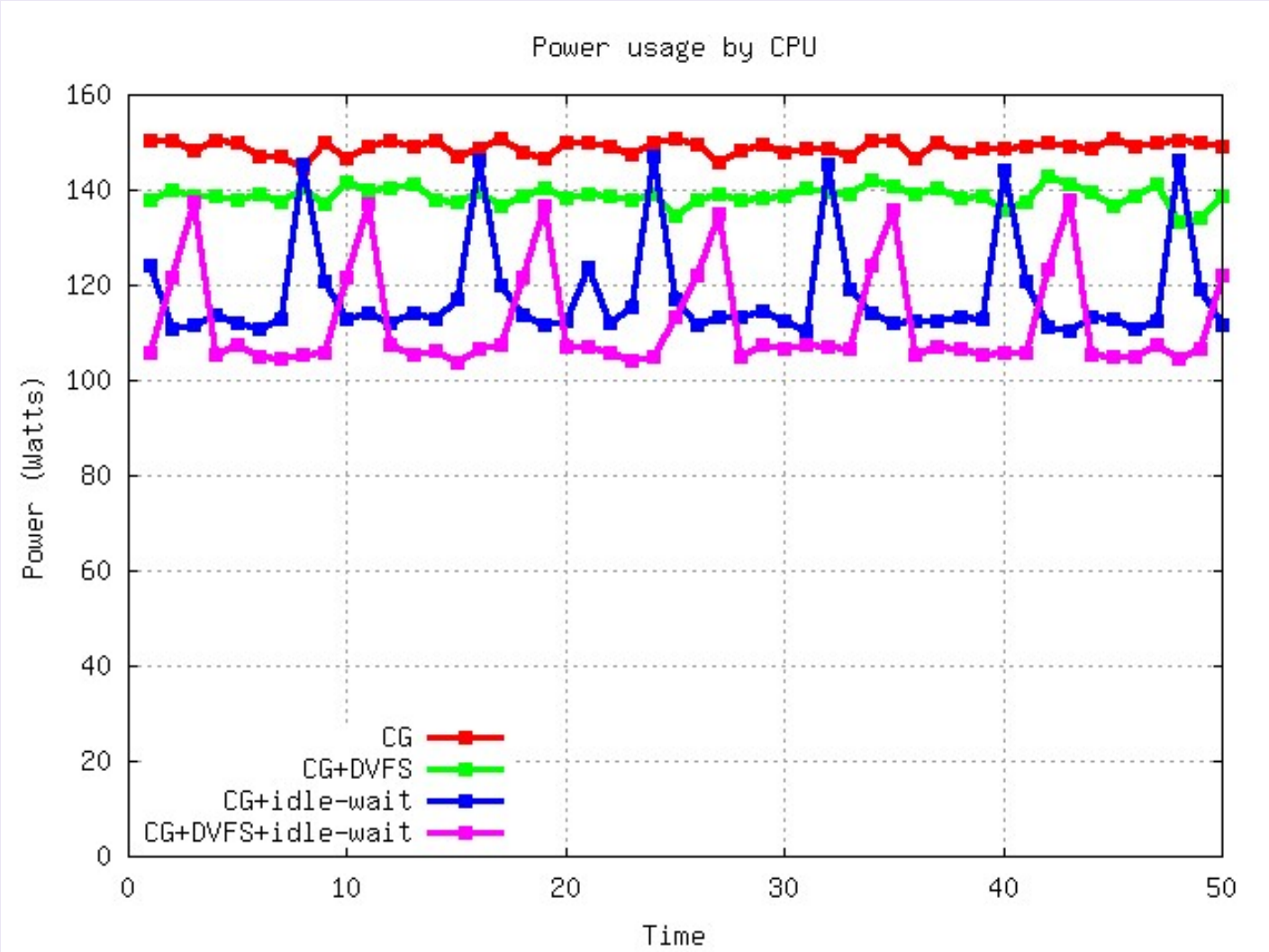
3.5 Idle-wait

Experiment #3

- Solution: set the CPU to "sleep" during the execution of the GPU kernels: Execution time of GPU `spmv` can be measured and accurately adjusted
- Use of `nanosleep()` function from `sys/time.h`
- Alternatives:

  (i) Plain solver
  (ii) Solver + DVFS during GPU execution
  (iii) Solver + DVFS + idle-wait during GPU execution

Power-friendly CPU modes



CG method: Energy consumption of chipset+GPU

matrix	energy consumption [Wh]			improvement [%]	
	(i)	(ii)	(iii)	(i)→(ii)	(i)→(iii)
A318	14.84	14.12	12.18	5.1	21.8
APACHE2	1.98	1.99	1.82	-0.5	8.8
AUDIKW_1	no convergence			-	-
BONES10	no convergence			-	-
ECOLOGY2	2.30	2.27	2.09	-1.3	10.0
G3_CIRCUIT	11.48	11.11	10.10	3.3	13.7
LDOOR	no convergence			-	-
N24K	26.43	25.42	21.17	3.97	24.8

PCG method: Energy consumption of chipset+GPU

matrix	energy consumption [Wh]			improvement [%]	
	(i)	(ii)	(iii)	(i)→(ii)	(i)→(iii)
A318	14.84	14.12	12.18	5.1	21.8
APACHE2	1.75	1.76	1.64	-0.6	6.7
AUDIKW_1	47.98	45.61	38.15	5.2	25.8
BONES10	157.32	150.16	125.78	4.8	25.1
ECOLOGY2	2.51	2.45	2.29	2.4	9.6
G3_CIRCUIT	2.71	2.63	2.38	3.0	13.9
LDOOR	43.22	41.18	34.79	5.0	24.2
N24K	34.62	32.97	27.64	5.0	25.3

3.6 Conclusions

- The concurrency of `spmv` enables the efficient usage of GPUs, that render important savings in execution time and energy consumption
- For memory-bounded operations, DVFS can potentially render energy savings...
  but the busy-wait of the host system during the kernel calls still consumes about 80 % of full-demand power
- The use of GPU-accelerated HPC-systems combined with power-saving techniques leads to more reduced energy consumption of all test problems without impacting the performance

More information

- "Power consumption of mixed precision in the iterative solution of sparse linear systems"
  H. Anzt, M. Castillo, J. C. Fernández, V. Heuveline, R. Mayo, E. S. Quintana, B. Rocker
  7th Workshop on High-Performance, Power-Aware Computing – HPPAC 2011, Anchorage (Alaska, USA), May 2011
- "Analysis and optimization of power consumption in the iterative solution of sparse linear systems on multi-core and many-core platforms"
  J. I. Aliaga, H. Anzt, M. Castillo, J. C. Fernández, V. Heuveline, R. Mayo, E. S. Quintana
  Int. Workshop on Power Measurement and Profiling – PMP 2011, Orlando (Miami, USA), July 2011
- "Optimization of power consumption in the iterative solution of sparse linear systems on graphics processors"
  H. Anzt, V. Heuveline, M. Castillo, J. C. Fernández, R. Mayo, E. S. Quintana, F. D. Igual
  2nd Int. Conf. on Energy-Aware High Performance Computing – EnaHPC 2011, Hamburg (Germany), Sept. 2011

Power-Aware Execution of Sparse and Dense Linear Algebra Libraries

Enrique S. Quintana-Ortí

quintana@icc.uji.es