Saving Energy in Sparse and Dense Linear Algebra Computations

P. Alonso*, M. F. Dolz†, F. Igual‡, R. Mayo†,
E. S. Quintana-Ortí†, V. Roca†

UNIVERSITAT JAUME•I

*Univ. Politécnica de Valencia, Spain

†Univ. Jaume I de Castellón, Spain

‡The Univ. of Texas at Austin, TX

†quintana@icc.uji.es

Motivation

- Reduce energy consumption!
  - Costs over lifetime of an HPC facility often exceed acquisition costs
  - Carbon dioxide is a risk for health and environment
  - Heat reduces hardware reliability

Personal view

- Hardware features mechanisms and modes to save energy
- Scientific apps. are in general energy-oblivious

Motivation

- Reduce energy consumption!
  - Costs over lifetime of an HPC facility often exceed acquisition costs
  - Carbon dioxide is a risk for health and environment
  - Heat reduces hardware reliability

Personal view

- Hardware features mechanisms and modes to save energy
- Scientific apps. are in general energy-oblivious

Motivation (Cont'd)

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

Scheduling tasks + DVFS

$\Downarrow$

Energy-aware execution on multi-core processors

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

Motivation (Cont'd)

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

Scheduling tasks + DVFS

$$\Downarrow$$

Energy-aware execution on multi-core processors

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

Outline

1. LU Factorization with Partial Pivoting (LUPP)

LU factorization

- Factor

$$A = LU,$$

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

- For numerical stability, permutations are introduced to prevent operation with small pivot elements

$$PA = LU,$$

with $P \in \mathbb{R}^{n \times n}$ a permutation matrix that interchanges rows of A

Blocked algorithm for LUPP (no PP, for simplicity)

---

for $k = 1 : s$ **do**
  $A_{k:s,k} = L_{k:s,k} \cdot U_{kk}$ — LU FACTORIZATION
  for $j = k+1 : s$ **do**
    $A_{kj} \leftarrow L_{kk}^{-1} \cdot A_{kj}$ — TRIANGULAR SOLVE
    $A_{k+1:s,j} \leftarrow A_{k+1:s,j} - A_{k+1:s,k} \cdot A_{kj}$ — MATRIX-MATRIX PRODUCT
  end for
end for

DAG with a matrix consisting of 5×5 blocks (s=5)

Blocked algorithm for LUPP (no PP, for simplicity)

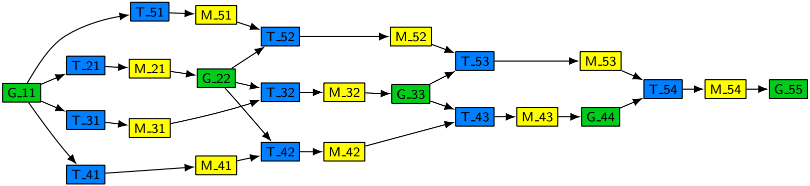
for $k = 1 : s$ **do**
  $A_{k:s,k} = L_{k:s,k} \cdot U_{kk}$
  for $j = k+1 : s$ **do**
    $A_{kj} \leftarrow L_{kk}^{-1} \cdot A_{kj}$
    $A_{k+1:s,j} \leftarrow A_{k+1:s,j} - A_{k+1:s,k} \cdot A_{kj}$
  end for
end for

LU factorization

Triangular solve

Matrix-matrix product

DAG with a matrix consisting of 5×5 blocks (s=5)



1.1 Slack Reduction Algorithm

Strategy

Search for "slacks" (idle periods) in the DAG associated with the algorithm, and try to minimize them applying e.g. DVFS

1. Search slacks via the Critical Path Method (CPM):
   - **DAG of dependencies**
     - Nodes ⇒ Tasks
     - Edges ⇒ Dependencies
   - **ES**$_i$/**LF**$_i$: Early and latest times task T_i, with cost C_i, can start/finalize without increasing the total execution time of the algorithm
   - **S**$_i$: Slack (time) task T_i can be delayed without increasing the total execution time
   - **Critical path**: Collection of tasks directly connected, from initial to final node of the graph, with total slack = 0
2. Minimize the slack of tasks with $S_i > 0$, reducing their execution frequency via SRA

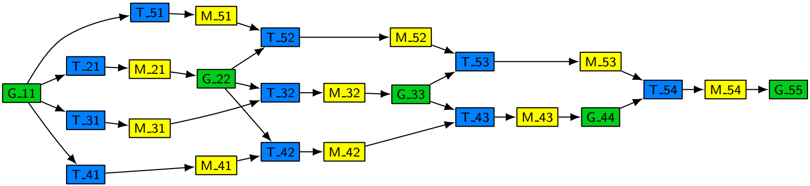
1.1 Slack Reduction Algorithm

Strategy

Search for “slacks” (idle periods) in the DAG associated with the algorithm, and try to minimize them applying e.g. DVFS

1. Search slacks via the Critical Path Method (CPM):
   - **DAG of dependencies**
     - Nodes $\Rightarrow$ Tasks
     - Edges $\Rightarrow$ Dependencies
   - $\mathbf{ES}_i/\mathbf{LF}_i$: Early and latest times task T_i, with cost C_i, can start/finalize without increasing the total execution time of the algorithm
   - $\mathbf{S}_i$: Slack (time) task T_i can be delayed without increasing the total execution time
   - **Critical path**: Collection of tasks directly connected, from initial to final node of the graph, with total slack $= 0$
2. Minimize the slack of tasks with $S_i > 0$, reducing their execution frequency via SRA

Application of CPM to the DAG of the LUPP of a 5×5 blocked matrix:



Task	C	ES	LF	S
G_11	109.011	0.000	109.011	0
T_21	30.419	109.011	139.430	0
T_41	30.419	109.011	287.374	147.944
T_51	30.419	109.011	326.306	186.877
T_31	30.419	109.011	225.082	85.652
⋮	⋮	⋮	⋮	⋮

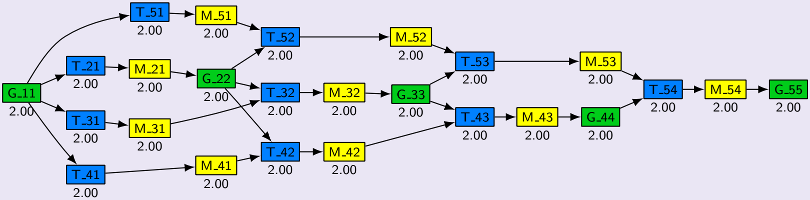
Duration of tasks (cost, C) of type G and M depends on the iteration!

Evaluate the time of 1 flop for each type of task and, from its theoretical cost, approximate the execution time

Slack Reduction Algorithm

1. **Frequency assignment: Set initial frequencies**
2. Critical subpath extraction
3. Slack reduction

1 Frequency assignment

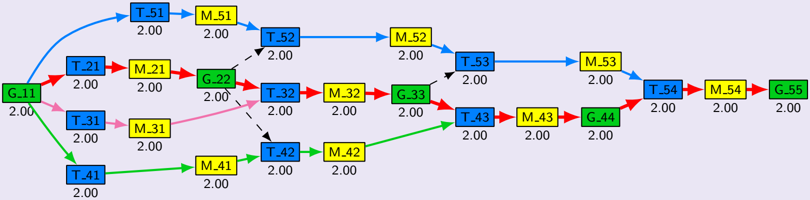


- Discrete range of frequencies: {2.00, 1.50, 1.20, 1.00, 0.80} GHz
- Duration of tasks of type `G` and `M` depends on the iteration.
- Initially, all tasks to run at the highest frequency: 2.00 GHz

Slack Reduction Algorithm

1. Frequency assignment
2. **Critical subpath extraction: Identify critical subpaths**
3. Slack reduction

2 Critical subpath extraction

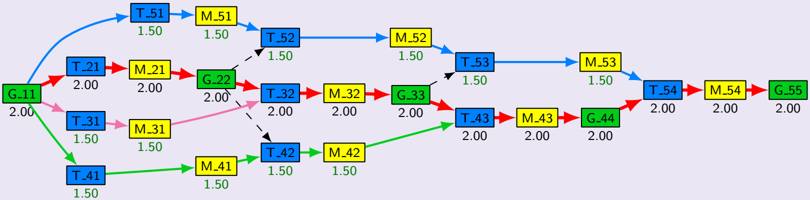


- Critical subpath extraction:
  1. Identify and extract the critical (sub)path(s)
  2. Eliminate the graph nodes/edges that belong to it
  3. Repeat process until the graph is empty
- Results:
  - Critical path: CP_0
  - 3 critical subpaths (from largest to shortest): $CP_1 > CP_2 > CP_3$

Slack Reduction Algorithm

1. Frequency assignment
2. Critical subpath extraction
3. **Slack reduction: Determine execution frequency for each task**

3 Slack reduction



- Slack reduction algorithm:
  1. Reduce frequency of tasks of the largest unprocessed subpath
  2. Check the lowest frequency reduction ratio for each task of that subpath
  3. Repeat process until all subpaths are processed
- Results:
  - Tasks of CSP_1, CSP_2 and CSP_3 are assigned to run at 1.5 GHz!

1.2 Race-to-Idle Algorithm

Race-to-Idle ⇒ complete execution as soon as possible by executing tasks of the algorithm at the highest frequency to “enjoy” longer inactive periods

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

1.3 Simulation

Use of 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 associated voltages
  - Frequency changes at core/socket level
  - Collection of real power for each combination of frequency idle/busy state per core
  - 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 are prioritized

AMD Opteron 6128 (8 cores):

- Evaluate the time of 1 flop for each type of task and, from its theoretical cost, approximate the execution time
- Frequency changes at core level, with $f \in \{2.00, 1.50, 1.20, 1.00, 0.80\}$ GHz

Blocked algorithm for LUPP:

- Simulation independent of actual implementation (LAPACK, `libflame`, etc.)
- Matrix size n from 768 to 5,632; block size: $b = 256$

Metrics:

- Compare simulated time and energy consumption of original algorithm with those of modified SRA/RIA algorithms

Simulated time

- $T_{SRA/RIA}$
- $T_{original}$
- Impact of SRA/RIA on time

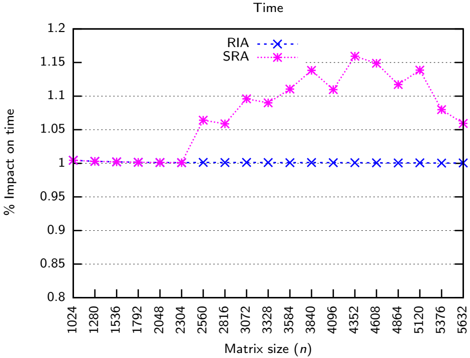
$$\boxed{IT = \frac{T_{SRA/RIA}}{T_{original}} \cdot 100}$$

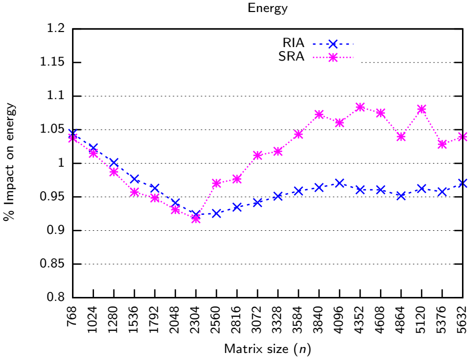
Simulated energy

- $E_{SRA/RIA} = \sum_{i=1}^{n} W_n \cdot T_n$
- $E_{original} = v^2 \ T(f_{max})$
- Impact of SRA/RIA on energy

$$\boxed{IE = \frac{E_{SRA/RIA}}{E_{original}} \cdot 100}$$

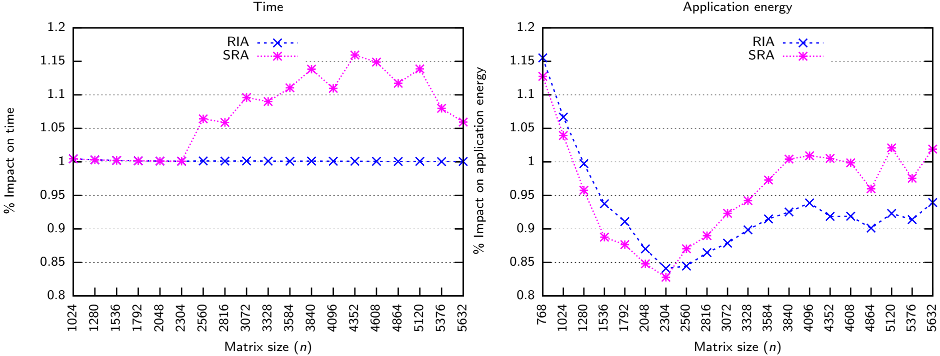
Impact of SRA/RIA on simulated time/energy for LUPP:





- SRA: Time is compromised, increasing the consumption for largest problem sizes
  - Increase in execution time due to SRA being oblivious to the actual resources
- RIA: Time is not compromised and consumption is reduced for large problem sizes

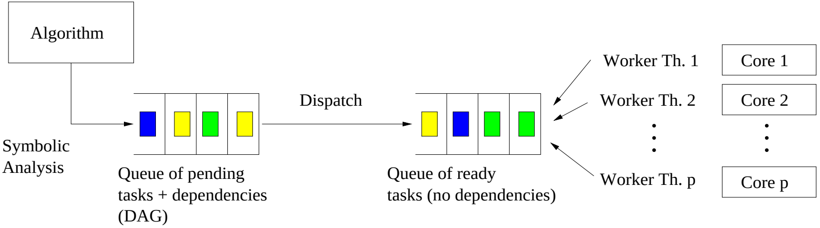
Impact of SRA/RIA on simulated time/energy for LUPP:
only power/energy due to workload (application)!



- SRA: Time is compromised, increasing the consumption for largest problem sizes
  - Increase in execution time due to the SRA being oblivious to the actual resources
- RIA: Time is not compromised and consumption is reduced for large problem sizes

1.4 Experimental Results

Integrate RIA into a runtime for task-parallel execution of dense linear algebra algorithms: `libflame+SuperMatrix`



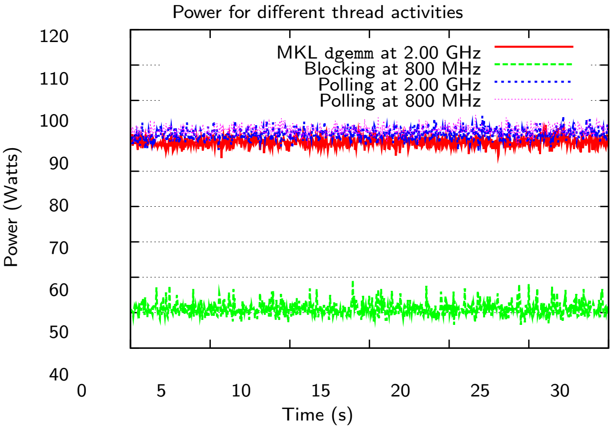
Two energy-aware techniques:

- RIA1: Reduce operation frequency when there are no ready tasks (Linux governor)
- RIA2: Remove polling when there are no ready tasks (while ensuring a quick recovery)

Applicable to any runtime: SuperMatrix, SMPSs, Quark, etc!

"**Doing nothing well**" – *David E. Culler*

AMD Opteron 6128, 1 core:



AMD Opteron 6128 (8 cores):

- Frequency changes at core level, with $f_{\text{max}} = 2.00$ GHz and $f_{\text{min}} = 800$ MHz

Blocked algorithm for LUPP:

- Routine `FLA_LU_piv` from `libflame v5.0`
- Matrix size n from 2,048 to 12,288; block size: $b = 256$

Metrics:

- Compare actual time and energy consumption of original runtime with those of modified RIA1/RIA2 runtime

Actual time

- T_{RIA}
- $T_{original}$
- Impact of RIA on time

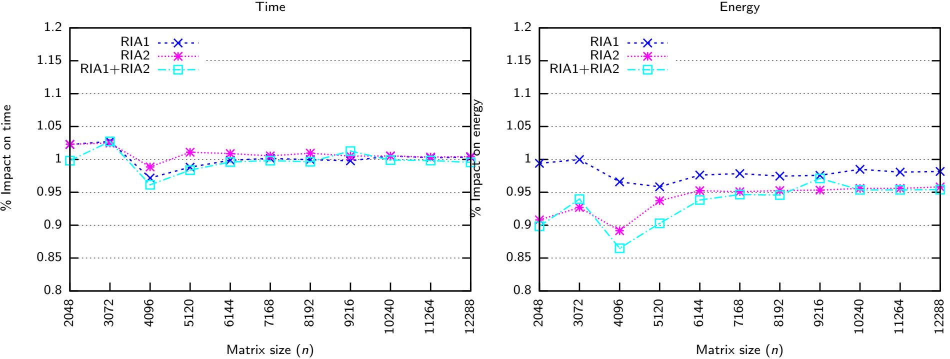
$$\boxed{IT = \frac{T_{RIA}}{T_{original}} \cdot 100}$$

Actual energy

- E_{RIA} and $E_{original}$ measured using internal powermeter with sampling f=25 Hz
- Impact of RIA on energy

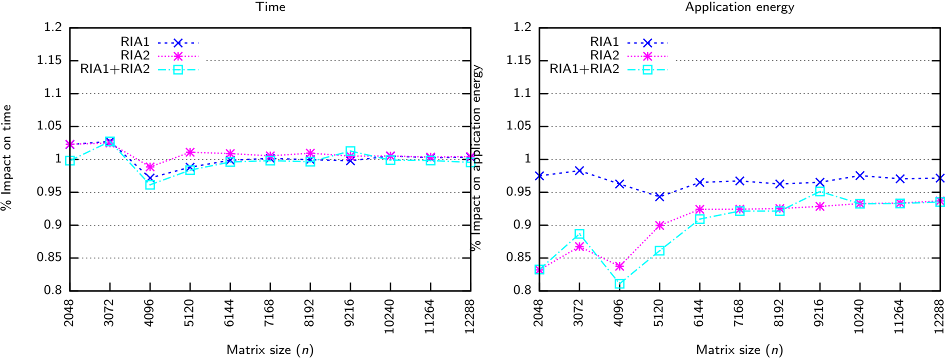
$$\boxed{IE = \frac{E_{RIA}}{E_{original}} \cdot 100}$$

Impact of RIA1/RIA2 on actual time/energy for LUPP:



- Small impact on execution time
- Consistent savings around 5 % for RIA1+RIA2:
- Not much hope for larger savings because there is no opportunity for that: the cores are busy most of the time

Impact of RIA1/RIA2 on actual time/energy for LUPP:
only power/energy due to workload (application)!



- Small impact on execution time
- Consistent savings around 7–8 % for RIA1+RIA2:
- Not much hope for larger savings because there is no opportunity for that: the cores are busy most of the time

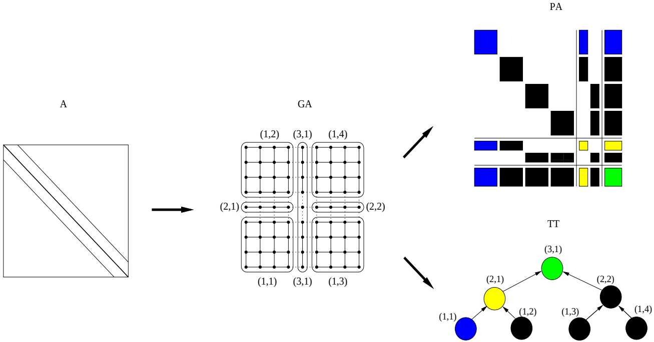
2. ILUPACK

Sequential version – M. Bollhöfer

- Iterative solver for large-scale sparse linear systems
- Multilevel ILU preconditioners for general and complex symmetric/Hermitian positive definite systems
- Based on inverse-based ILUs: incomplete LU decompositions that control the growth of the inverse triangular factors

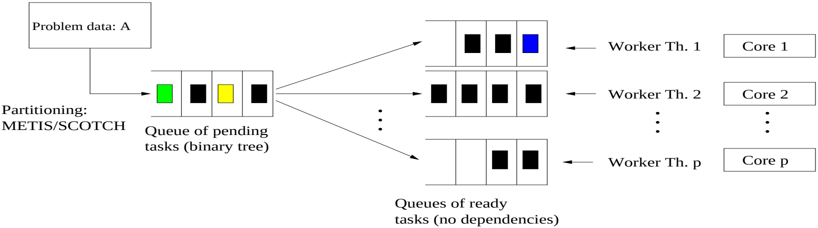
Multi-threaded version – J.I. Aliaga, M. Bollhöfer, A. F. Martín, E. S. Quintana-Ortí

- Real symmetric positive definite systems
- Construction of preconditioner and PCG solver
- Algebraic parallelization based on a task tree
- Leverage task-parallelism in the tree
- Dynamic scheduling via tailored run-time (OpenMP)



A
GA
(1,2)
(3,1)
(1,4)
(2,1)
(2,2)
(1,1)
(3,1)
(1,3)
PA
TT
(3,1)
(2,1)
(2,2)
(1,1)
(1,2)
(1,3)
(1,4)

Integrate RIA into multi-threaded version of ILUPACK



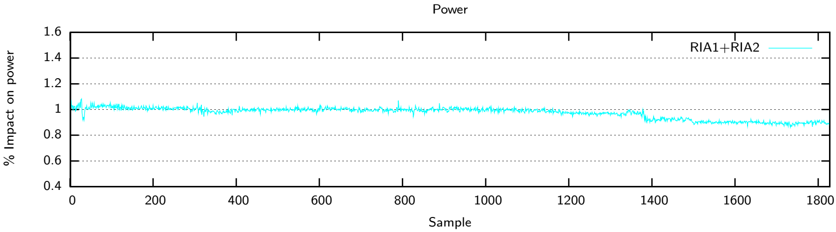
Same energy-aware techniques:

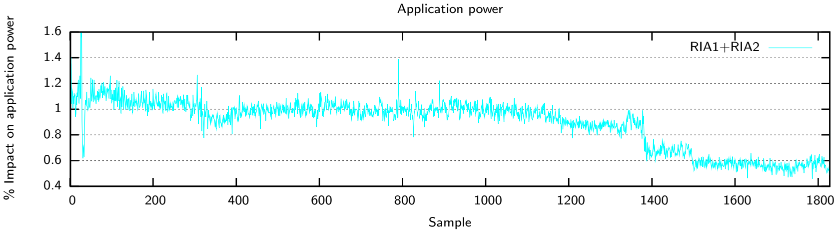
- RIA1+RIA2: Reduce operation frequency when there are no ready tasks (Linux governor) and remove polling when there are no ready tasks (while ensuring a quick recovery)

AMD Opteron 6128 (8 cores)

Linear system associated with Laplacian equation ($n \approx$16M)

Impact of RIA1+RIA2 on actual power/application power for ILUPACK (only preconditioner):





3. Conclusions

Goal: Combine scheduling+DVFS to save energy during the execution of linear algebra algorithms on multi-threaded architectures

For (dense) LUPP:

Slack Reduction Algorithm

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

Race-to-Idle Algorithm

- Algorithm is applied on-the-fly, no pre-processing needed
- Maintains in all of cases execution time
- Reduce application energy consumption (around 7–8 %)

As the ratio between the problem size and the number of resources grows, the opportunities to save energy decrease!

For (sparse) ILUPACK:

Race-to-Idle Algorithm

- DAG requires no processing
- Algorithm is applied on-the-fly
- Maintains in all of cases execution time
- Reduce power dissipated by application up to 40 %

Significant opportunities to save energy!

General:
We need architectures that know how to do nothing better!

More information

- "Improving power-efficiency of dense linear algebra algorithms on multi-core processors via slack control"
  P. Alonso, M. F. Dolz, R. Mayo, E. S. Quintana-Ortí
  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-Ortí
  2nd Int. Conf. on Energy-Aware High Performance Computing – EnaHPC 2011, Hamburg (Germany), Sept. 2011
- "Saving energy in the LU factorization with partial pivoting on multi-core processors"
  P. Alonso, M. F. Dolz, F. Igual, R. Mayo, E. S. Quintana-Ortí
  20th Euromicro Conf. on Parallel, Distributed and Network based Processing – PDP 2012, Garching (Germany). Feb. 2012

Saving Energy in Sparse and Dense Linear Algebra Computations

P. Alonso[*], M. F. Dolz[†], F. Igual[‡], R. Mayo[†],
E. S. Quintana-Ortí[†], V. Roca[†]

[*]Univ. Politécnica de Valencia, Spain

[†]Univ. Jaume I de Castellón, Spain

[‡]The Univ. of Texas at Austin, TX

[†]quintana@icc.uji.es