INTERNATIONAL CONFERENCE ON
ENERGY-AWARE HIGH PERFORMANCE COMPUTING

DVFS-Control Techniques for Dense Linear Algebra Operations on Multi-Core Processors

Pedro Alonso[1], Manuel F. Dolz[2], Francisco D. Igual[2],
Rafael Mayo[2], Enrique S. Quintana-Ortí[2]

1 UNIVERSITAT POLITÈCNICA DE VALÈNCIA

2 UNIVERSITAT JAUME·I

September 07–09, 2011, Hamburg (Germany)

Motivation

- High performance computing:
  - Optimization of algorithms applied to solve complex problems
- Technological advance ⇒ improve performance:
  - Processors works at higher frequencies
  - Higher number of cores per socket (processor)
- Large number of processors and cores ⇒ High energy consumption
- Methods, algorithms and techniques to reduce energy consumption applied to high performance computing.
  - Reduce the frequency of processors with DVFS techniques

Outline

1. Introduction
2. Dense linear algebra operations
3. Slack Reduction Algorithm
   - Introduction
   - Application
   - Previous steps
   - Slack reduction
4. Race-to-Idle Algorithm
5. Experimental results
   - Simulator
   - Benchmark algorithms
   - Environment setup
   - Results
6. Conclusions

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

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

Dense linear algebra operations

LU factorization:

- Factor

$$A = LU,$$

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

- Two algorithms for LU factorization:
  - LU with partial (row) pivoting (traditional version)
  - 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-if-the-Art in Scientific Computing - PARA 2004
      Copenhaguen (Denmark), June 2004

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

- We consider a partitioning of matrix A into blocks of size $b \times b$

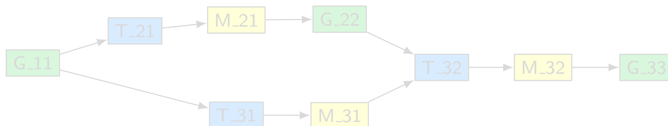
Dense linear algebra operations

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



Dense linear algebra operations

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**

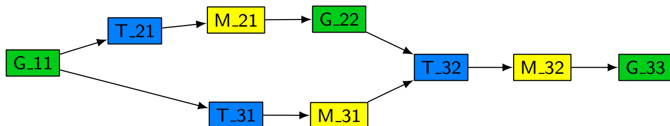
$A_{kj} \leftarrow L_{kk}^{-1} \cdot A_{kj}$ — TRIANGULAR SOLVE — b^3 flops

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

end for

end for

DAG with a matrix consisting of 3×3 blocks



Dense linear algebra operations

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

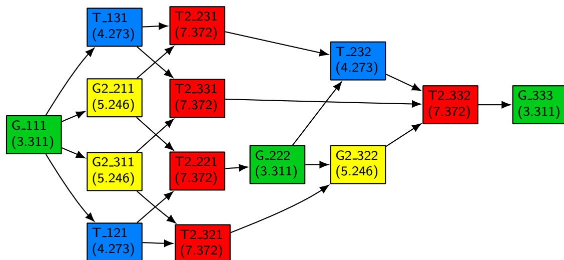
end for

end for

Dense linear algebra operations

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!

Slack Reduction Algorithm: Introduction

Idea

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

The Critical Path Method:

- **DAG of dependencies**
  - Nodes ⇒ Tasks
  - Edges ⇒ 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.

Slack Reduction Algorithm: Introduction

Idea

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

The Critical Path Method:

- **DAG of dependencies**
  - Nodes ⇒ Tasks
  - Edges ⇒ 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 to dense linear algebra algorithms

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 $\rightarrow$ *Slack Reduction Algorithm*

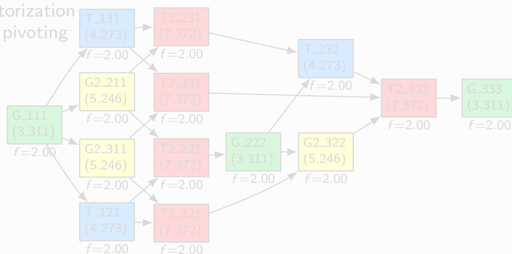
Slack Reduction Algorithm

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

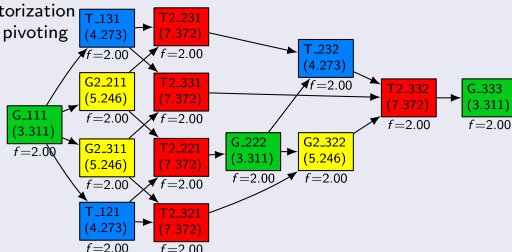
Slack Reduction Algorithm

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

Critical subpath extraction

2 Critical subpath extraction

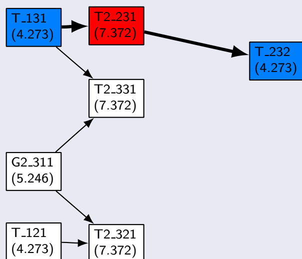
Iteration 0

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

Critical subpath extraction

2 Critical subpath extraction

Iteration 1

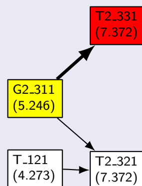


CP_i	Tasks	Execution time
CP_0	{G_111, G2_211, T2_221, G_222, G2_322, T2_332, G_333}	35.171 ms
CP_1	{T_131, T2_231, T_232}	15.918 ms

Critical subpath extraction

2 Critical subpath extraction

Iteration 2

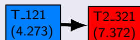


CP_i	Tasks	Execution time
CP_0	{G_111, G2_211, T2_221, G_222, G2_322, T2_332, G_333}	35.171 ms
CP_1	{T_131, T2_231, T_232}	15.918 ms
CP_2	{G2_311, T2_331}	12.619 ms

Critical subpath extraction

2 Critical subpath extraction

Iteration 3



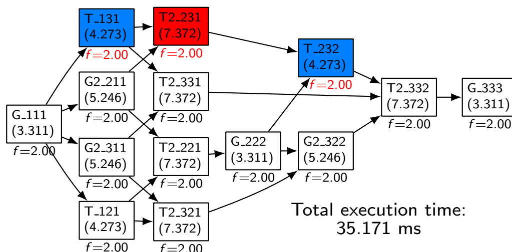
CP_i	Tasks	Execution time
CP_0	{G_111, G2_211, T2_221, G_222, G2_322, T2_332, G_333}	35.171 ms
CP_1	{T_131, T2_231, T_232}	15.918 ms
CP_2	{G2_311, T2_331}	12.619 ms
CP_3	{T_121, T2_321}	11.646 ms

Slack Reduction

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;

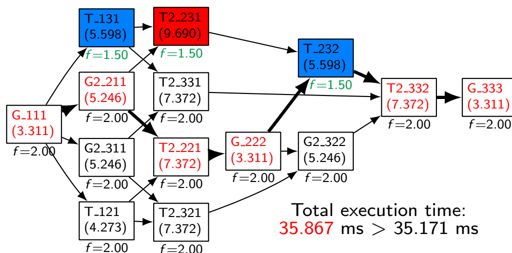


Slack Reduction

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;

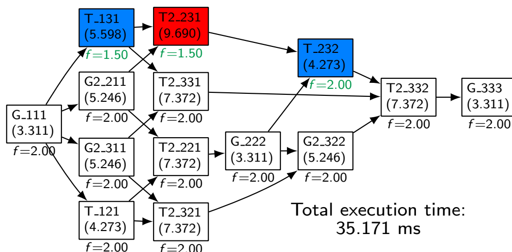


Slack Reduction

Iteration 1

Process critical subpath $CP_1 = \{\text{T_131}, \text{T2_231}, \text{T_232}\}$:

1. Increase ratio for CP_1: $\frac{d(\text{G_111} \rightsquigarrow \text{T_232}) - d(\text{G_111} \rightsquigarrow \text{T_131})}{l(CP_1)} = \frac{21{,}176}{15{,}919} = 1{,}33\,\%$
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;

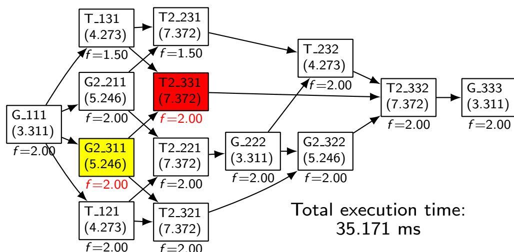


Slack Reduction

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;

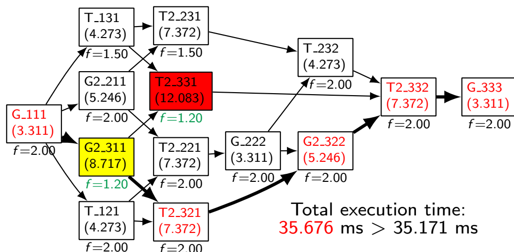


Slack Reduction

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;

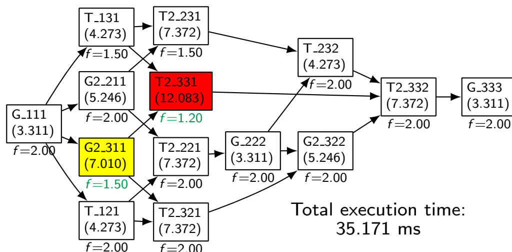


Slack Reduction

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;

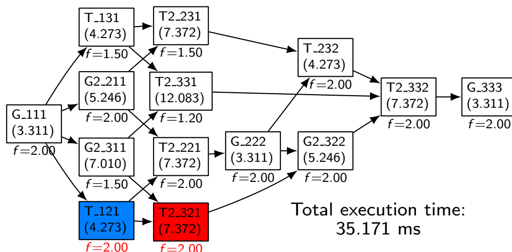


Slack Reduction

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;



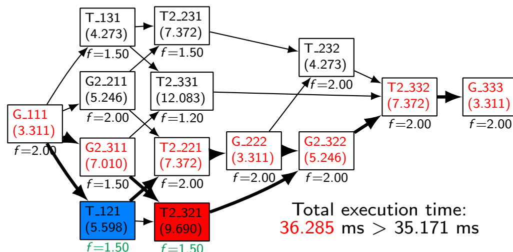
Total execution time:
35.171 ms

Slack Reduction

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;

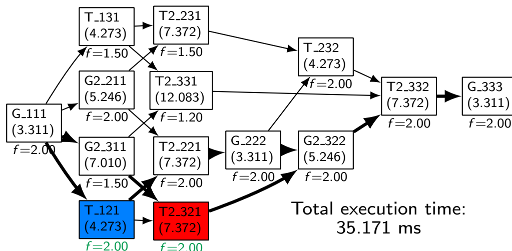


Slack Reduction

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;



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

- 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

Simulator

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

Input parameters:

- DAG capturing tasks and dependencies of a blocked algorithm and recommended frequencies 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

Benchmark algorithms

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

- 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 the DGEMM kernel at different frequencies:

Core	Frequency-Running/Idle								
	1	2	3	4	5	6	7	8	Power (W)
	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 (in microseconds):

Source freq.	Destination freq.				
	2.00	**1.50**	**1.20**	**1.00**	**0.80**
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	–

Metrics

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

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

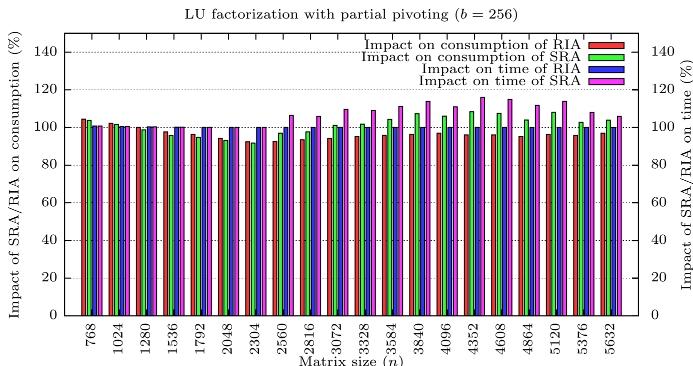
Consumption

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

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

LU factorization with partial pivoting

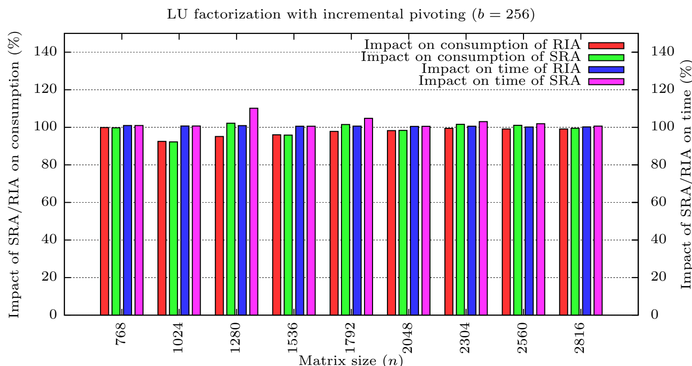
Impact of the SRA/RIA on energy and time for the LU factorization with partial pivoting:



- 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

LU factorization with incremental pivoting

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



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

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 %)

Conclusions and future work

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:

  Working at highest frequency $>$ Working at lowest frequency $\ggg$ Idle at lowest frequency

Energy savings

- Reduce environmental impact
- Reduce electrical costs

Thanks for your attention!

Questions?