20th Euromicro International Conference on Parallel, Distributed and Network-Based

Special Session on Energy-aware Systems

Saving Energy in the LU Factorization with Partial Pivoting on Multi-Core Processors

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

1 Universitat Politècnica de València

2 Universitat Jaume I

February 15–17, 2012, Garching bei München (Germany)

Motivation

- High performance computing:
  - Optimization of algorithms applied to solve complex problems
- Technological advance $\Rightarrow$ improve performance:
  - Higher number of cores per socket (processor)
- Large number of processors and cores $\Rightarrow$ High energy consumption
- Methods, algorithms and techniques to reduce energy consumption applied to high performance computing

Outline

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

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

$\Downarrow$

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

$\Downarrow$

Power-aware scheduling on multi-core processors

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

$\Downarrow$

Energy savings

Dense linear algebra operations

- LU factorization with partial pivoting

$$PA = LU$$

$A \in \mathbb{R}^{n \times n}$ nonsingular matrix
$P \in \mathbb{R}^{n \times n}$ permutation matrix
$L/U \in \mathbb{R}^{n \times n}$ unit lower/upper triangular matrices

- We consider a partitioning of matrix A into blocks of size $b \times b$
- For numerical stability, permutations are introduced to prevent operation with small pivot elements

LU factorization using FLAME notation

- $n(\cdot)$ stands for the number of columns of its argument
- $\text{TRILU}(\cdot)$ denotes the matrix consisting to the elements in lower triangular with diagonal replaced by ones
- pivoting omitted for simplicity

Some cost details

- Blocked algorithm performs $2n^3/3 + O(n^2)$ flops
- Most of them cast in terms of `gemm` operations

Algorithm: $A := \text{LUP\_BLK}(A)$

Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right)$
where A_{TL} is 0×0
while $n(A_{TL}) < n(A)$ **do**
Determine block size b
Repartition

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$$

where A_{11} is $b \times b$

$$\begin{array}{rcll} \left(\begin{array}{c} A_{11} \\ \hline A_{21} \end{array}\right) & := & \text{LUP\_UNB}\left(\begin{array}{c} A_{11} \\ \hline A_{21} \end{array}\right) & \\ A_{12} & := & \text{TRILU}(A_{11})^{-1} A_{12} & (\texttt{trsm}) \\ A_{22} & := & A_{22} - A_{21} A_{12} & (\texttt{gemm}) \end{array}$$

Continue with

$$\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$$

endwhile

Parallelization

Parallelization not trivial at code level!

We use SuperMatrix runtime to execute `libflame` routines:

- Operations are not executed in the order they appear in code ⇒ Control-flow Parallelism
- SuperMatrix schedules execution respecting data dependencies ⇒ Data-flow parallelism

SuperMatrix proceeds in two stages:

1. A symbolic execution produces a DAG containing dependencies
2. DAG dictates the feasible orderings in which task can be executed

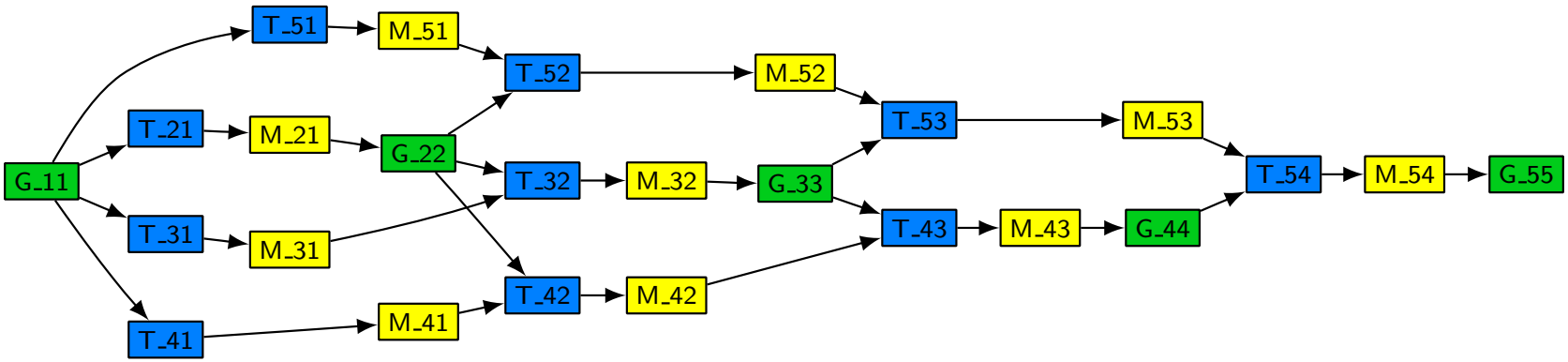
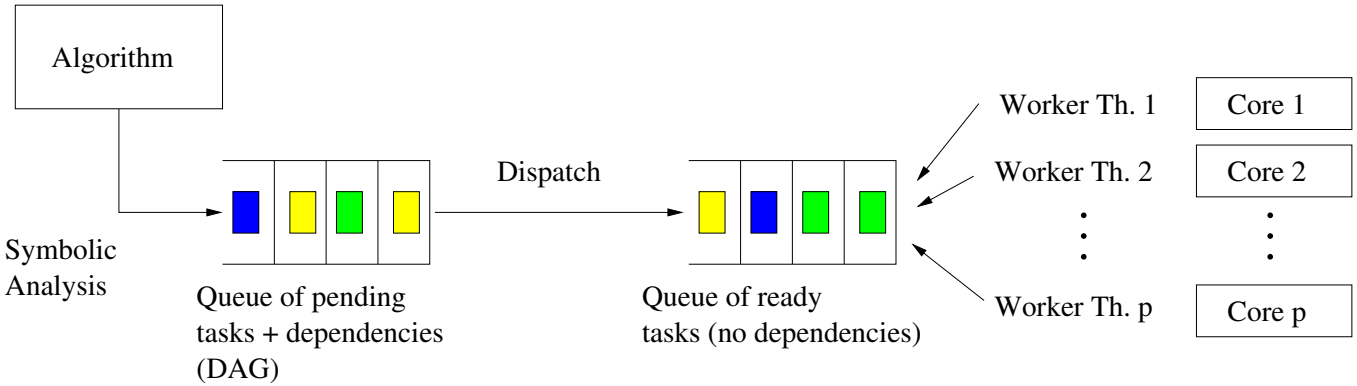


Figure: DAG with a matrix consisting of 5 × 5 blocks

SuperMatrix runtime:



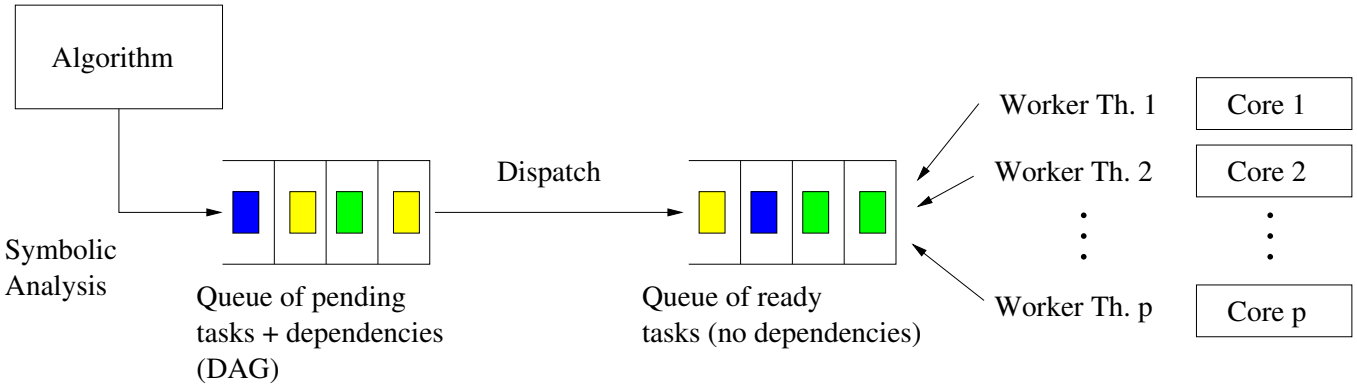
Basic scheduling:

1. Initially only one a task in *ready queue*
2. A thread acquires a task of the *ready queue* and runs the corresponding job
3. Upon completion checks tasks which were in the *pending queue* moving to *ready* if their dependencies are satisfied.

Problem!

- Idle threads (one per core) continuously check the ready list for work
  Busy-wait or polling $\Rightarrow$ Energy consumption!

SuperMatrix runtime:



Basic scheduling:

1. Initially only one a task in *ready queue*
2. A thread acquires a task of the *ready queue* and runs the corresponding job
3. Upon completion checks tasks which were in the *pending queue* moving to *ready* if their dependencies are satisfied.

Problem!

- Idle threads (one per core) continuously check the ready list for work
  Busy-wait or polling ⇒ Energy consumption!

Energy-Aware Techniques into SuperMatrix

Modern Linux distributions leverage DVFS: "`cpufreq-utils`"

- Increase/reduce operation frequency of cores
- Governors: control operation frequency

`performance`	frequency is always fixed to the highest
`ondemand`	frequency is controlled by cpufreq kernel module, sets the highest frequency when core is loaded and the lowest when core is idle
`userspace`	frequency is controlled by user

Basic idea:

- Dependencies between tasks $\Rightarrow$ idle periods
- Idle periods $\Rightarrow$ can be exploited to save energy

We consider Race-to-idle Algorithm. Why?

- Current processors are quite efficient at saving power when idle
- Power of idle core is much smaller than power in working periods

EA1: Reduce operation frequency when there are no ready tasks

Objective $\Rightarrow$ Do a more efficient Race-to-Idle technique

Is there a ready task for a polling thread in the *ready list*?

- No: runtime immediately sets the operation frequency of the associated core to the lowest possible
- Yes: frequency is raised back to the highest and task is executed by the thread

Settings:

- Linux governor is set to `userspace`: frequency is controlled by SuperMatrix
- Experiments on an 6128 AMD (8-core):
  - DVFS AMD PowerNow! technology allows to control frequency at core level!

EA2: Remove polling when there are no ready tasks

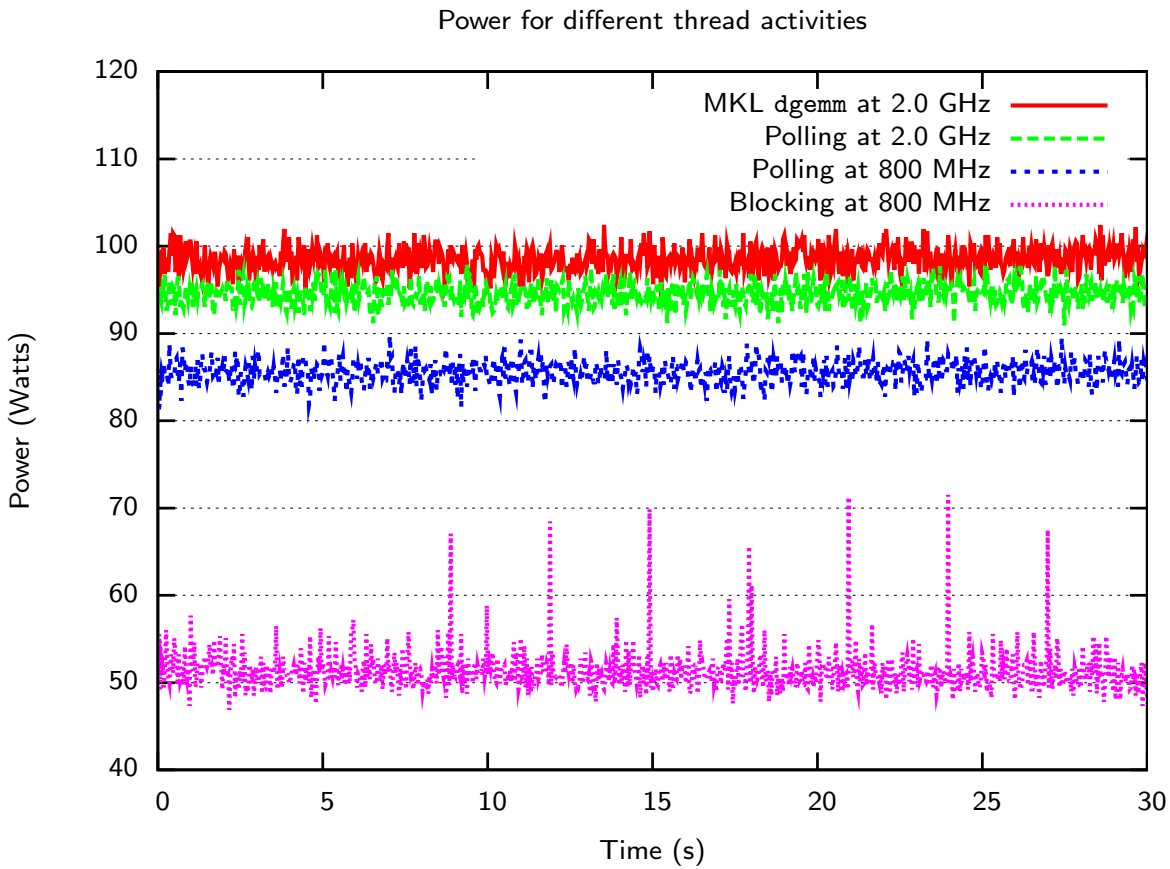
Objective $\Rightarrow$ Replace busy-waits by idle-waits

Is there a ready task for a polling thread in the *ready list*?

- No: the thread blocks/suspends itself in a POSIX semaphore by calling to `sem_wait()`
- Yes:
  1. Thread acquires and executes the task
  2. Updates dependencies: moves k tasks from *pending queue* to *ready queue*
  3. Thread activates $k-1$ sleeping threads to attend *ready* tasks by calling $k-1$ times to `sem_post()`

Settings:

- Linux governor is set to `ondemand`: highest frequency when working, and lowest when idle
- POSIX semaphores to allow suspension states for non working threads
  Idle-wait $\Rightarrow$ Low energy consumption



- MKL dgemm at 2.0 GHz: 97-100 W
- Polling at 2.0 GHz: 93-97 W
- Polling at 800 MHz: 83-88 W
- Blocking at 800 MHz: 49-55 W

Experimental results

Environment setup:

- 8-core AMD 6128 processor (2.0 GHz) with 24 Gbytes of RAM
  - Discrete collection of frequencies: {2.0, 1.5, 1.2, 1.0, 0.8} GHz
- Linux Ubuntu 10.04
- BLAS and LAPACK implementations from MKL 10.2.4
- SuperMatrix runtime in `libflame` v5.0-r5587
  - Two energy saving techniques: EA1+EA2

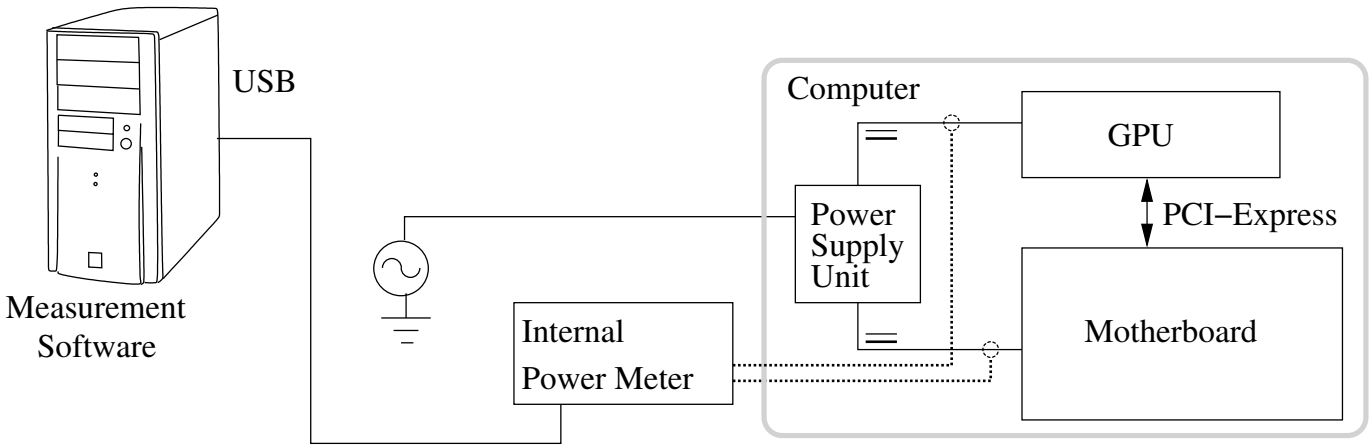
Benchmark algorithm:

- LU with partial pivoting
  - `FLASH_LU_piv` routine $\Rightarrow$ blocked right-looking variant of LU factorization
- Block size: $b = 512$
- Matrix size ranges from 2,048 to 12,288

Environment setup

Power measurements:

- Internal DC powermeter
- ASIC directly attached to the lines connecting the power supply unit and the motherboards (chipset+processors)
- Operation frequency 25 Hz $\Rightarrow$ 25 samples/sec.



Experiments

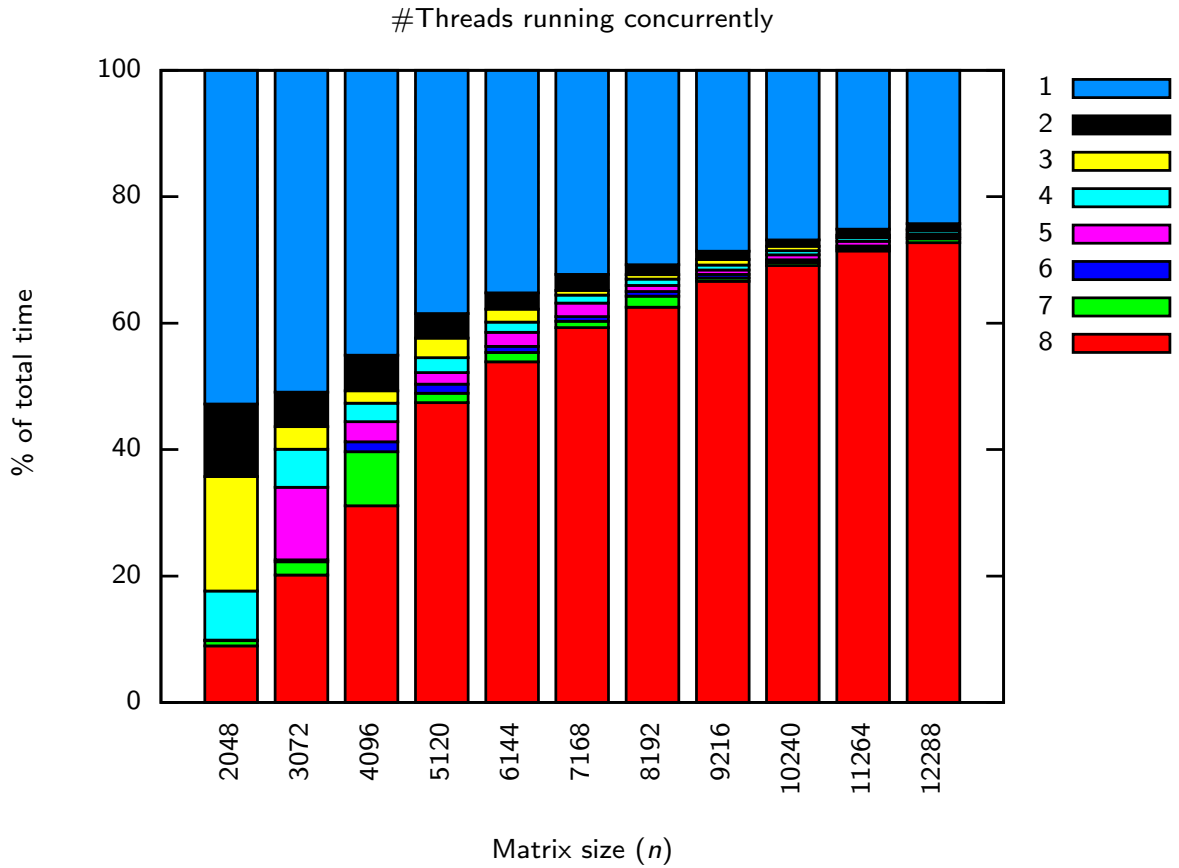
Evaluation:

1. We evaluate the existence and length of idle periods during the computation of the LU factorization
2. We measure the actual gains that can be attained by our energy-aware approaches with 3 versions of the SuperMatrix runtime:
   - Original runtime (`performance`)
   - EA1: Reduces operation frequency during polling periods (`userspace`)
   - EA2: Integrates semaphores to block idle threads and avoid polling (`ondemand`)
   - EA1+EA2 (`userspace`)

   $\Rightarrow$ Each experiment was repeated 30 times and average values of energy consumption were reported

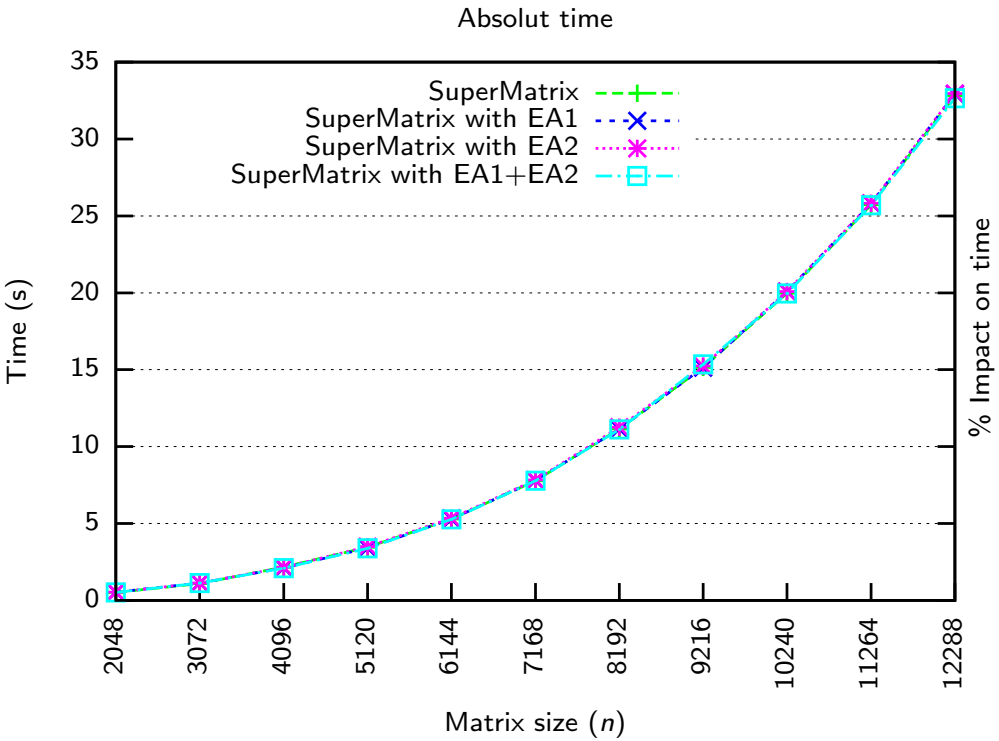
   $\Rightarrow$ Maximum coefficient of variation was lower than 0.8 %

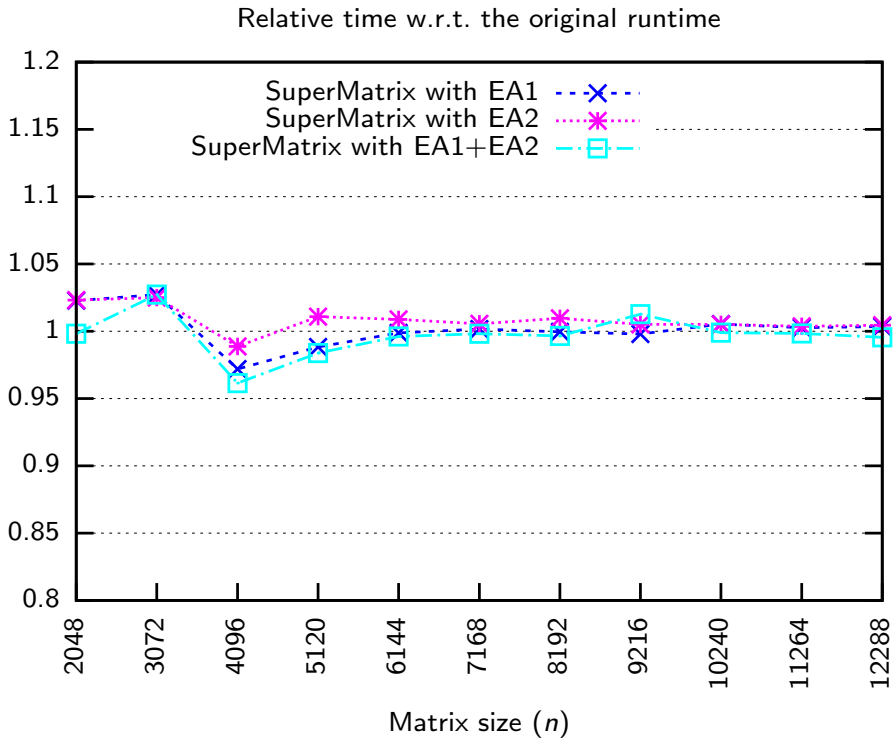
Thread activity during the execution of the LU factorization with partial pivoting



- For small problem sizes, 53 % of time there is a single active thread and only 9 % of time all threads are performing work
- For large problem sizes, 26 % of time there is a single active thread and only 74 % of time all threads are performing work

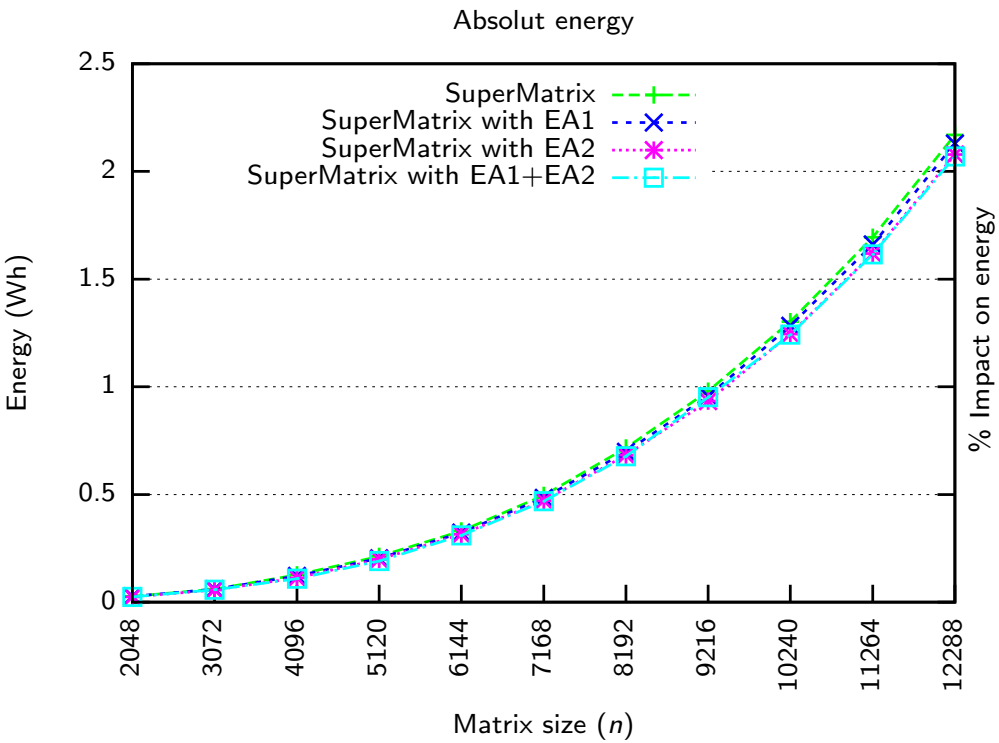
Impact on and relative execution time of LU factorization with partial pivoting using the energy-aware techniques

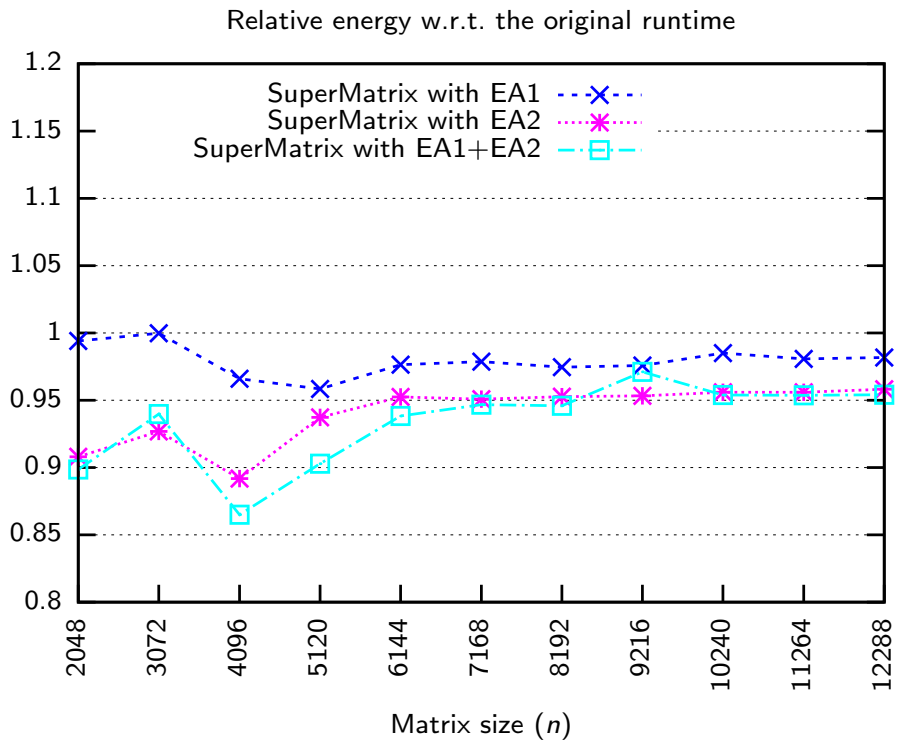




- These techniques introduce a minimal overhead in execution time
  - Longer execution time, due to the period required to "wake-up" blocked threads
- Increase of 2.5 % for the smallest problem sizes, for other sizes there is no difference

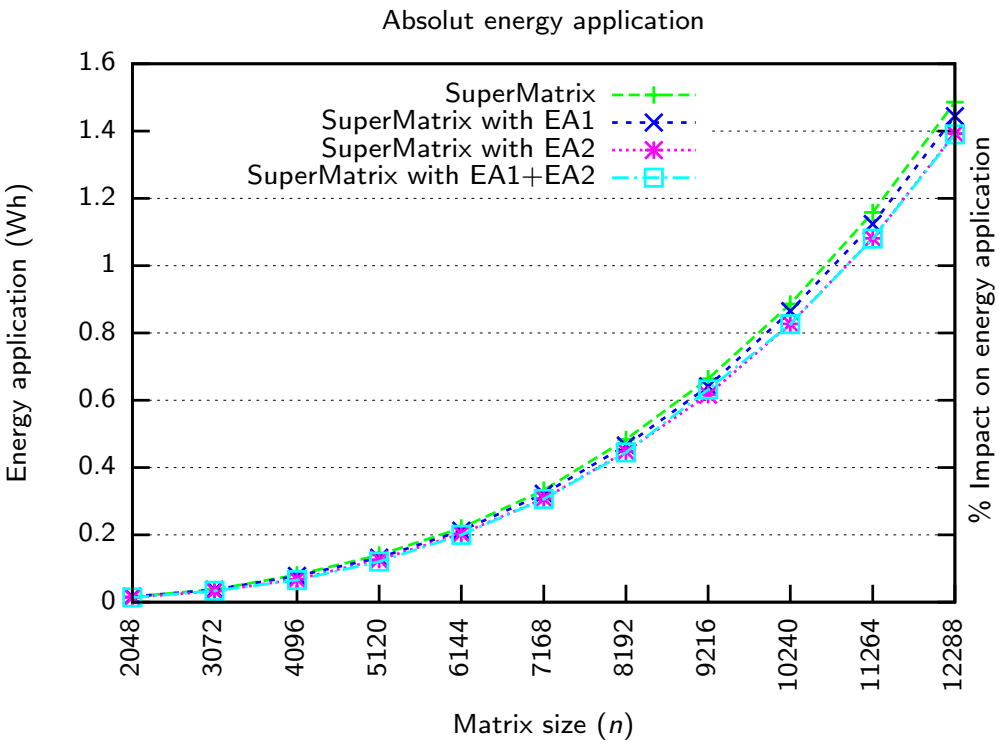
Impact on absolute and relative consumption of LU factorization with partial pivoting using the energy-aware techniques

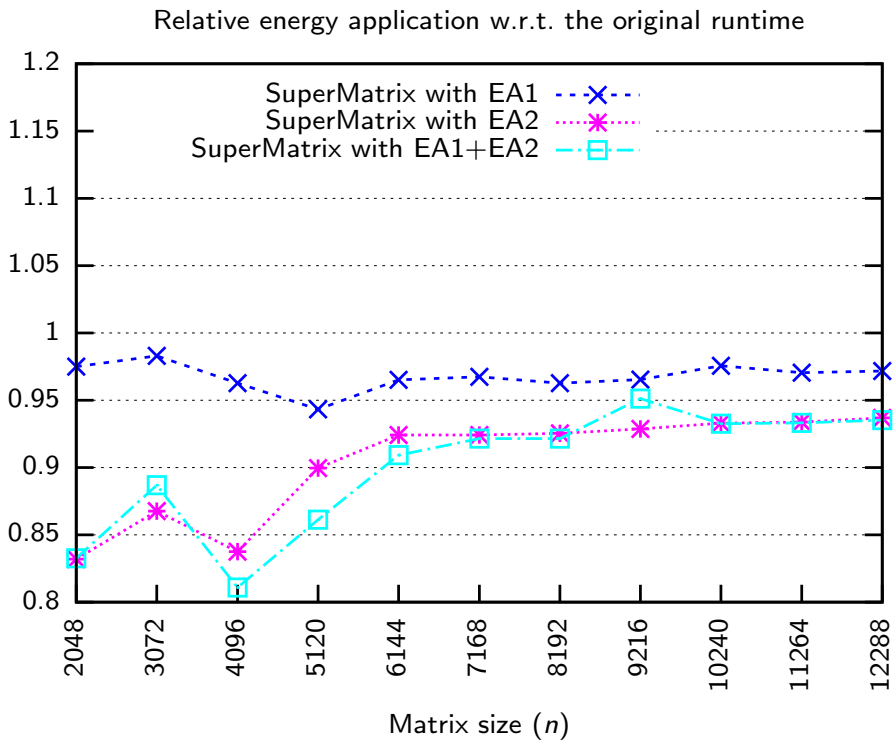




- For smallest problem sizes, the number of tasks is relatively low compared with the number of threads
  - Produces longer idle periods during the execution the algorithm ⇒ Energy savings
- EA1 potentially leads to savings of 2 % and 5 % when using EA2 for largest problem sizes
- The combination of both techniques produce similar energy-savings than as the use only of EA2

Impact on absolute and relative execution application consumption of LU factorization with partial pivoting using the energy-aware techniques





- New metric $\Rightarrow$ Application consumption
- To obtain energy consumption values subtract idle power (75W) for each result:

$$E = (\bar{P} - 75) \cdot t$$

- Energy gains considering only application using EA2 and EA1+EA2 range from 22 % for the smallest sizes to 7 % for the largest problem sizes.

Summary and conclusions

Some conclusions:

- Idle periods appear when executing algorithms with data dependencies
- **Race-to-Idle**: 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
- Optimize idle periods for energy saving

LU with partial pivoting $\Rightarrow$ Just an example of algorithm with idle periods!

- Two techniques to leverage inactive periods
- Results:
  - Reduction of energy consumption of 5 % and 22 % if we only consider application consumption
  - Negligible increase of execution time
- Basic idea to save energy: avoid active polling (busy-wait) in the runtime

Do nothing well!
David E. Culler

Thanks for your attention!

Questions?