IC804/IC805 Cost Action Meeting

Energy-aware Techniques and Models for Matrix Computations

Manuel F. Dolz
dolzm@icc.uji.es

Universitat Jaume I

October 18–19, 2012, Cork (Ireland)

Who we are

High Performance Computing & Architectures Group

- Composed of 12 researchers, all of them faculty members of the "Depto. de Ingeniería y Ciencia de Computadores" of the Jaume I University (Spain). There are also 5 Ph.D. students and 4 software engineers.

Main research lines:

- High performance libraries for dense/sparse linear algebra problems (BLAS, LAPACK, etc.)
  - Linear systems, eigenproblems, singular values, etc.: *libflame*, *ILUPACK*
  - Strong interest in GPUs
- Power-aware computing
  - Power-aware linear algebra libraries: Energy-aware SuperMatrix runtime in *libflame*
  - Virtualization of GPUs: Remote CUDA, *rCUDA*
  - Power-aware middleware: *EnergySaving Cluster*

UNIVERSITAT JAUME·I

HPC&A
High Performance Computing and Architectures

More info at http://www.hpca.uji.es

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
- Techniques to reduce energy consumption!
  - Costs over lifetime of an HPC facility often exceed acquisition costs
  - Carbon dioxide is a hazard for health and environment
  - Heat reduces hardware reliability
- Current status
  - Scientific apps are in general energy oblivious!
  - Learn how to exploit hardware features to obtain energy savings: P/C-states

Outline

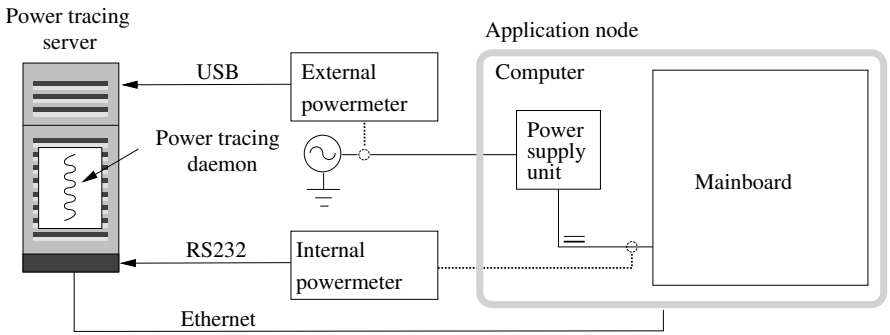
Performance and power measurement framework

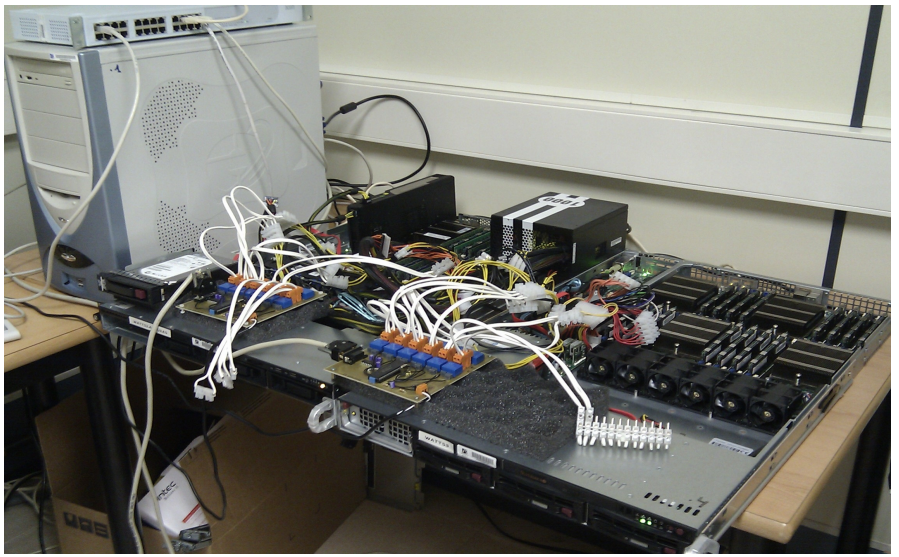
Performance tracing:

- `Extrae+Paraver`: instrumentation package and visualization tool from BSC

Power tracing:

- `pmlib` library: Power measurement package of Jaume I University (Spain)
  - Interface to interact and use our own design and commercial power meters

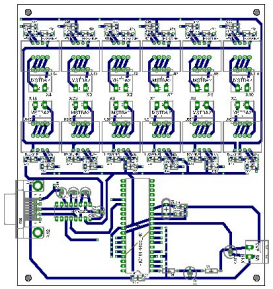




- **Server daemon**: collects data from power meters and send to clients
- **Client library**: enables communication with server and synchronizes with start-stop primitives

Power measurement devices

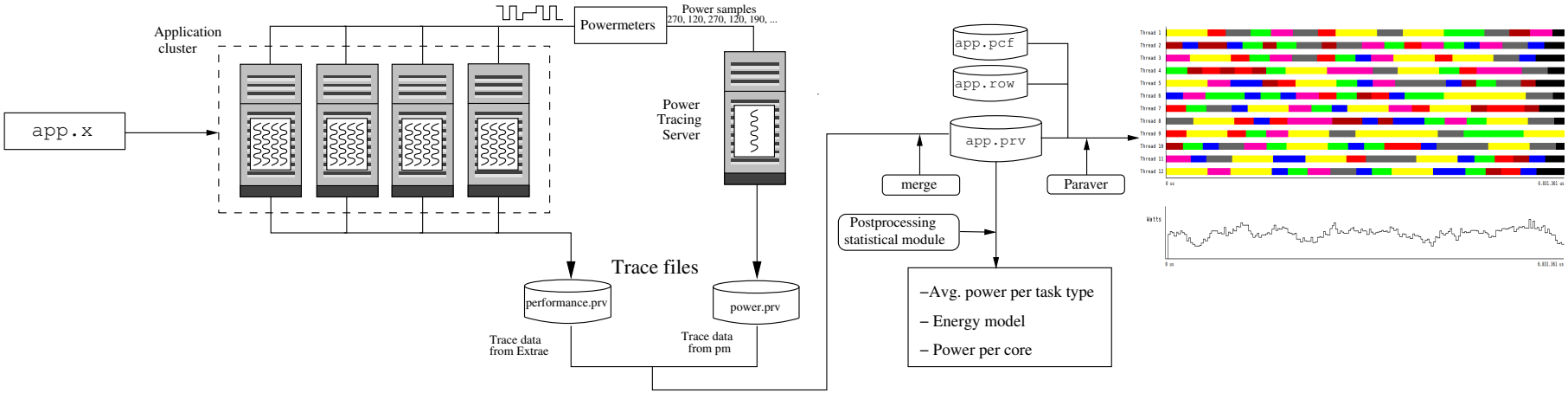
- **Internal devices**: measure power dissipated by the components in the mainboard
  - ASIC-based powermeter (own design!)
    - LEM HXS 20-NP transductors with PIC microcontroller
    - Sampling rate: from 25 Hz to 100 Hz
    - RS232 serial port
  - National Instruments data acquisition card
    - NI9205 / cDAQ-9178
    - Sampling rate: 7 KHz!
    - USB port



- **External devices**: measure overall machine power
  - WattsUp? Pro .NET
    - Sampling rate: 1 Hz
    - Only 1 outlet!
    - USB/Ethernet ports
  - Power Distribution Unit APC 8653
    - Sampling rate: 1 Hz
    - 24 outlets
    - SNMP/`ssh` via Ethernet

Code execution

Basic execution schema for tracing performance and power:



Trace files:

- `Extrae` outputs `performance.prv` file
- `pmlib` outputs `power.prv` file

Tools:

- `Paraver`: performance and power trace visualization
- `Post-processing statistic module`:
  - Energy model, power per core, etc.

Energy-aware hardware techniques

ACPI (Advanced Configuration and Power Interface):
Industry-standard interfaces enabling OS-directed configuration, power/thermal management of platforms

Performance states (P-states):

- P_0: Highest performance and power
- P_i, $i > 0$: As i grows, more savings but lower performance

P-state P_i	VCC_i	f_i
P_0	1.23	2.00
P_1	1.17	1.50
P_2	1.12	1.20
P_3	1.09	1.00
P_4	1.06	0.80

Server AMD:
Two AMD Opteron 6128 cores @ 2.0 GHz (16 cores)

- $P = g\,(V^2\,f)$
- $E = \int_0^T P\,dt = g(V^2)$

$\longrightarrow$ DVFS!

To DVFS or not? General concensus!

- Not for compute-intensive apps.: reducing frequency increases execution time linearly!
- Yes for memory-bounded apps. as cores are idle a significant fraction of the time.

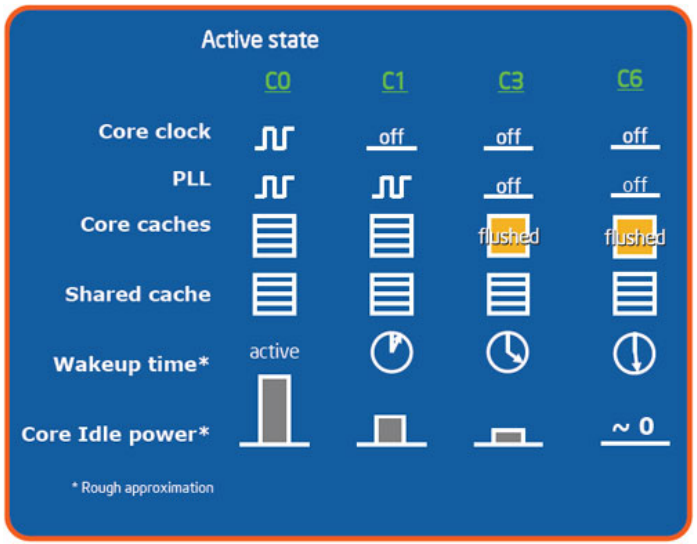
But take care! $\Rightarrow$ In some platforms (AMD) reducing frequency via DVFS also reduces memory bandwidth proportionally!

P-states can be managed at socket level in Intel and at core level in AMD!

Energy-saving states: P/C-states

Power states (C-states):

- C_0: normal execution (also a P-state)
- C_i, $i > 0$: no instructions being executed. As i grows, more savings but longer latency to reach C0

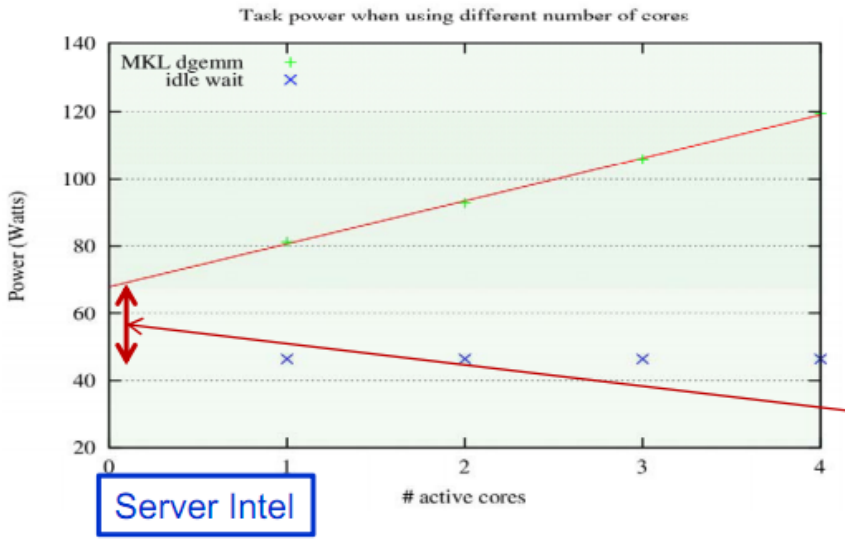


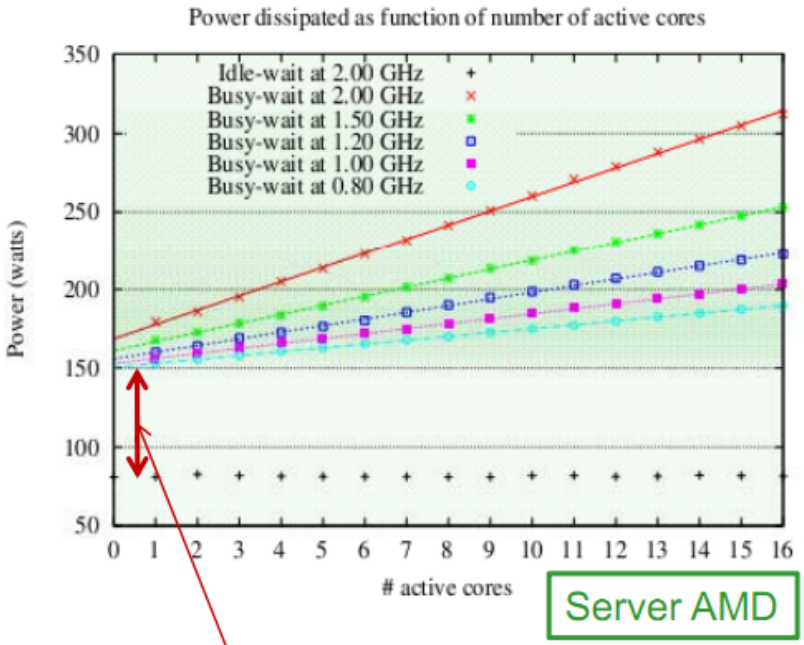
How to exploit C-states?

- Is impossible to change C-state at code level!
- Solution ⇒ Set necessary conditions so that hw promotes cores to energy-saving C-states

Examples: P-states/C-states

- "Do nothing, efficiently..." (V. Pallipadi, A. Belay)
- "Doing nothing well" (D. E. Culler)





Opportunities to save energy via C-states!

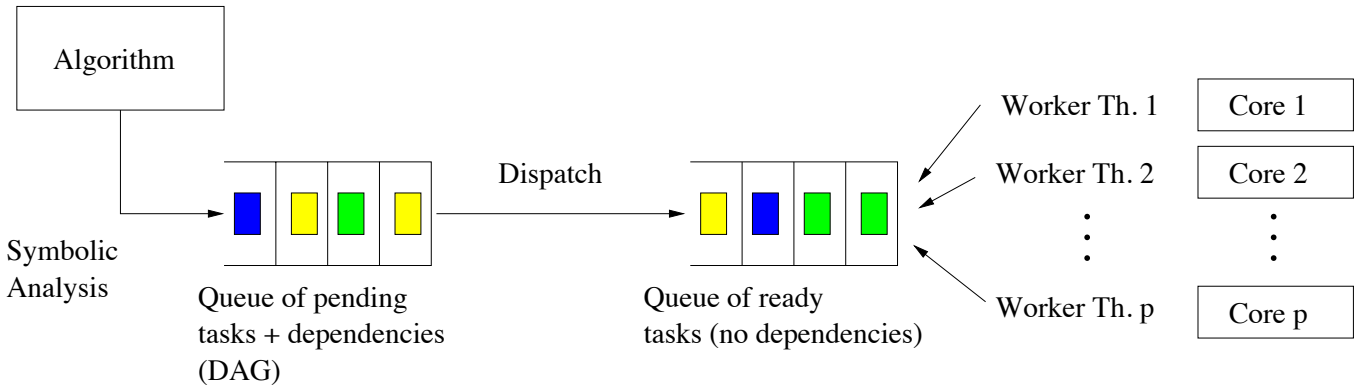
Problem! Not straight-forward. No direct user control over C-states!

Energy-aware software techniques

Energy-aware techniques focused only on the "processors"!

Dense linear algebra applications:

- Task-parallel execution of dense linear algebra algorithms: `libflame+SuperMatrix`



Problem:

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

Solution:

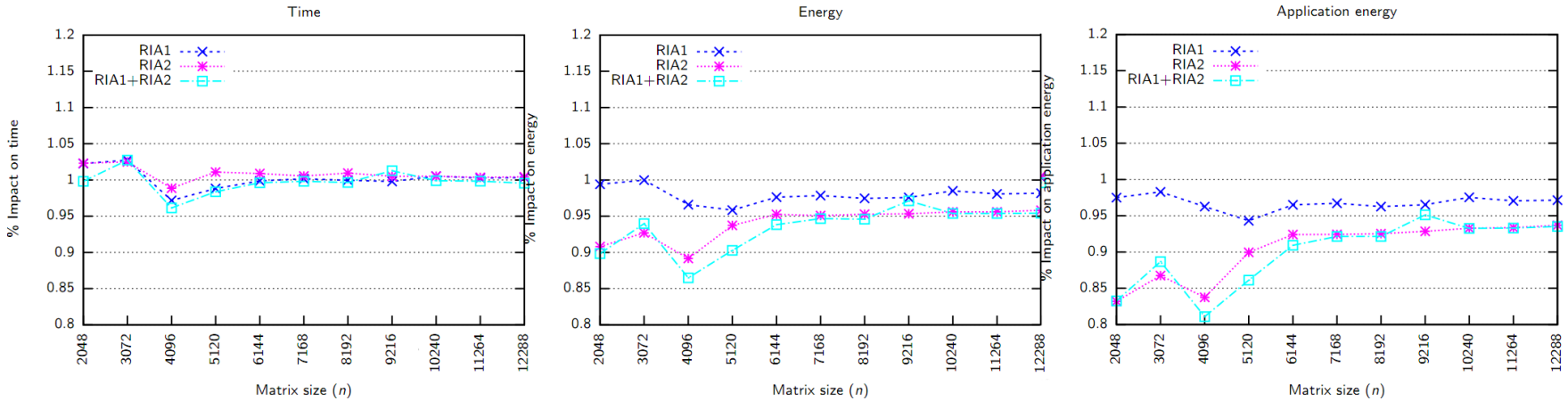
- *Race-to-idle*: Detect and replace "busy-waits" by "idle-waits": avoid idle processors doing polling!

Results: Dense linear algebra

Energy-aware techniques on **multicore** platforms:

- RIA1: Reduce operation frequency when there are no ready tasks: DVFS `ondemand` governor
- RIA2: Remove polling when there are no ready tasks (while ensuring a quick recovery): POSIX Semaphores

On multicore: `FLA_LU` (LUpp fact.) from libflame + SuperMatrix runtime



- Consistent savings around 5% for total energy and 7–8% for application energy
- Poor savings? Dense linear algebra operations exhibit little idle periods!

Results: Dense linear algebra

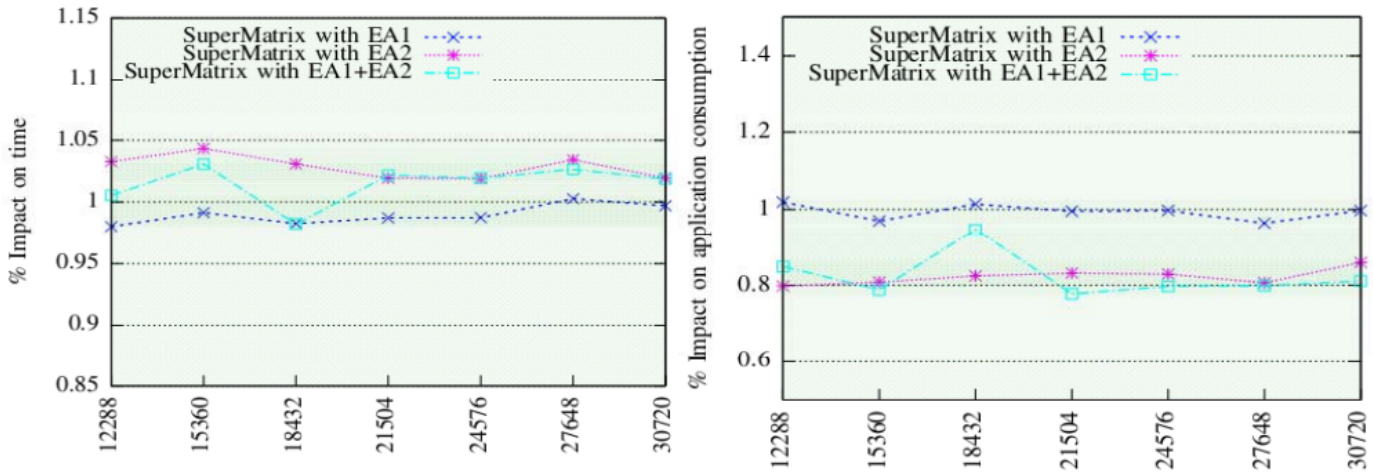
Why CPU+GPU (for some compute-intensive apps.)?

- High performance computational power / Affordable price / High FLOPS per watts ratio

Energy-aware techniques for **hybrid CPU+GPU** platforms:

- EA1: blocking for idle threads without task: POSIX Semaphores
- EA2: blocking for idle threads waiting for GPU task completion
  Set blocking operation mode (synchronous) for CUDA kernels

On hybrid CPU+GPU: `FLA_Chol` (Cholesky fact.) from libflame+SuperMatrix

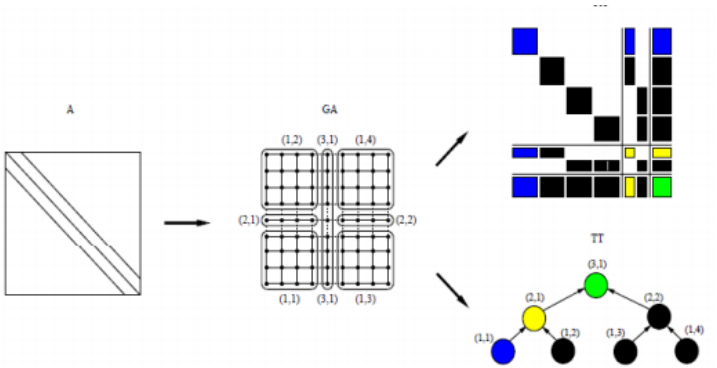


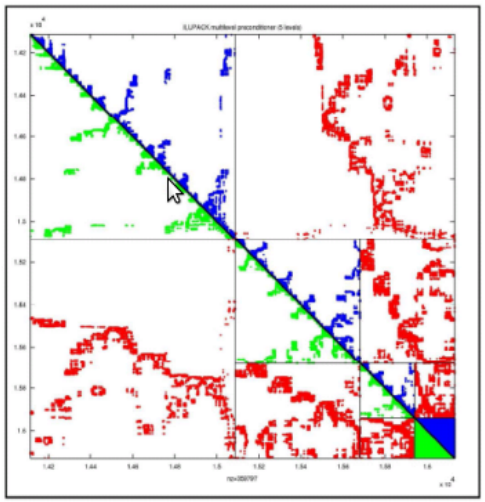
Execution of tasks in GPU makes CPU cores inactive during significant time!

Results: Sparse linear algebra

Sparse linear algebra applications:

- Task-parallel implementation of ILUPACK for multicore processors with *ad-hoc runtime*
- Sparse linear system from Laplacian eqn. in a 3D unit cube



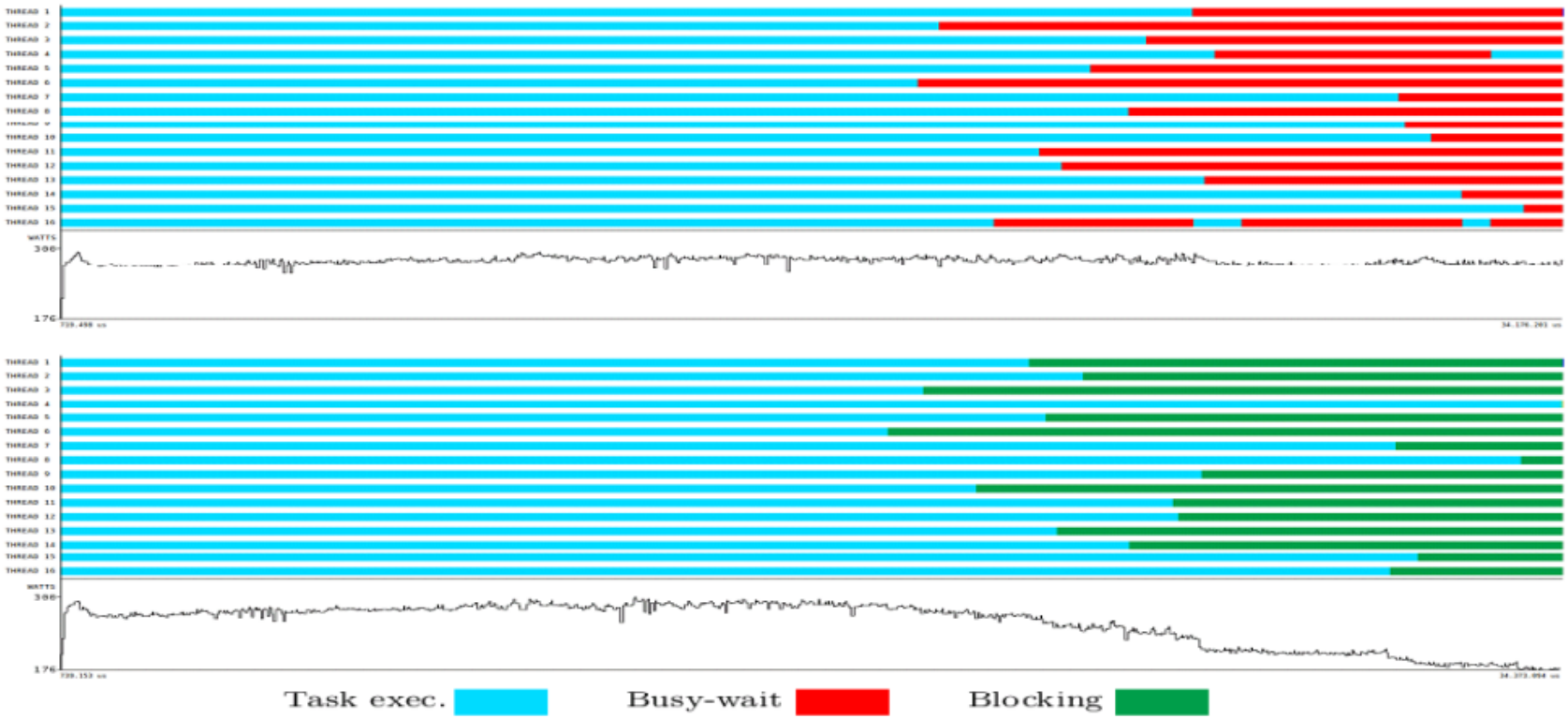


Energy-aware techniques:

- Application of RIA1+RIA2 techniques into *ad-hoc runtime*

Results: Sparse linear algebra

Polling vs. blocking for idle threads when obtaining ILU preconditioners:



Blocking vs polling for idle threads

- Saving around 7% of total energy
- Negligible impact on execution time

...but take into account that

- Idle time: 23.70%, Dynamic power: 39.32%
- Upper bound of savings: $39.32 \cdot 0.2370 = 9.32\%$

Power modeling

Simple power model:

$$P = P^{C(PU)} + P^{(S)Y(stem)} = P^{S(tatic)} + P^{D(ynamic)} + P^{(S)Y(stem)}$$

$P^{C(PU)}$ Power dissipated by the CPU: $P^{S(tatic)} + P^{D(ynamic)}$

$P^{(S)Y(stem)}$ Power of remaining components (e.g. RAM)

Some considerations:

- Study case: **Cholesky factorization**. It exercises CPU+RAM and discards other power sinks (network interface, PSU, etc.)
- We assume P^Y and P^S are constants!
- P^S grows with the temperature inertia till maximum! $\Rightarrow$ We consider a "hot" system!

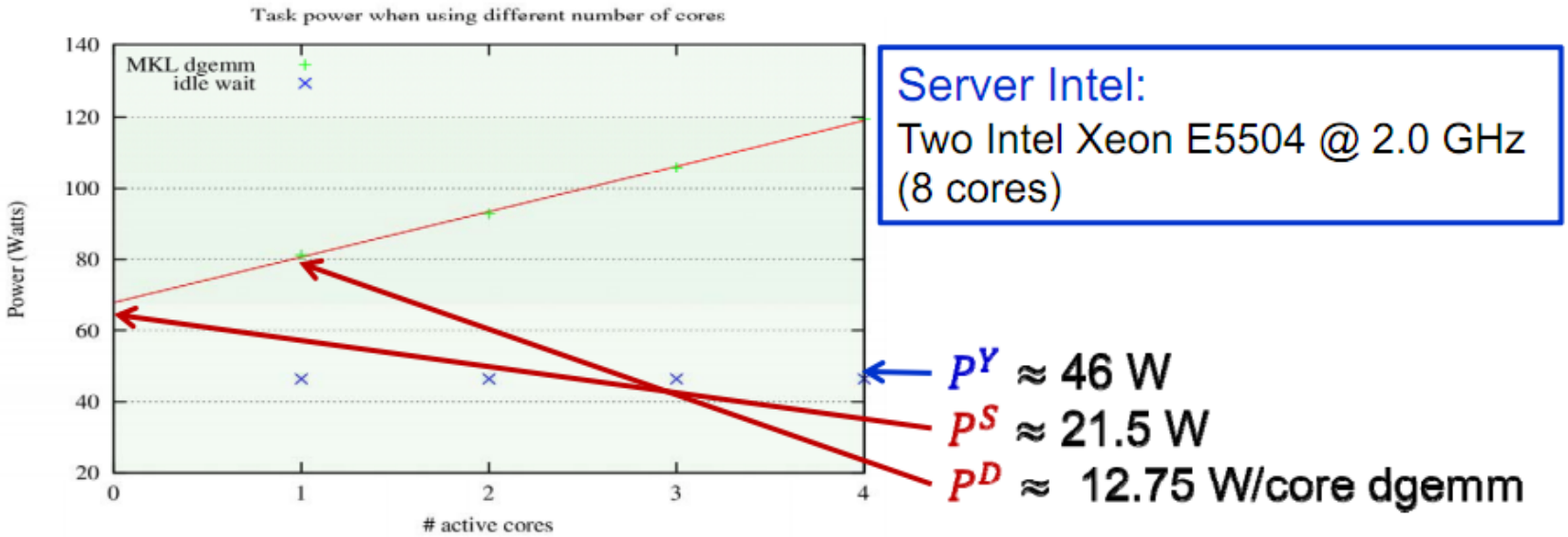
Environment setup:

- Intel Xeon E5504 (2 quad-cores, total of 8 cores) @ 2.00 GHz with 32 GB RAM
- Intel MKL 10.3.9 for sequential `dpotrf`, `dtrsm`, `dsyrk` and `dgemm` kernels
- SMPSs 2.5 for task-level parallelism
- Internal power meter sampling at 25 Hz

Component estimation

Obtaining power model components:

- P^Y directly obtained measuring idle platform: $P^Y = 46.37 Watts$
- P^S obtained by executing dgemm kernel using 1 to 4 cores and adjusting via linear regression
- P_K^D is obtained by continuously execute the kernel K until power stabilizes and then sample this value



Linear regression: $P_{\texttt{dgemm}}(c) = \alpha + \beta \cdot c = 67.97 + 12.75 \cdot c$

$$P^S \approx \alpha - P^Y = 67.97 - 46.47 = 21.5 \text{ Watts}$$

Power/energy model testing

Power model:

$$P(t) = P^Y + P^S + P^D(t) = P^Y + P^S + \sum_{i=1}^{r}\sum_{j=1}^{c} P_i^D N_{i,j}(t)$$

r stands for the number of different types of tasks, (r=5 for Cholesky)

c stands for the number of threads/cores

P_i^D average dynamic power for task of type i

$N_{i,j}(t)$ equals to 1 if thread j is executing a task of type i at time t; equals 0 otherwise

Energy model:

$$E = (P^Y + P^S)\,T + \int_{t=0}^{T} P^D(t)\ dt$$

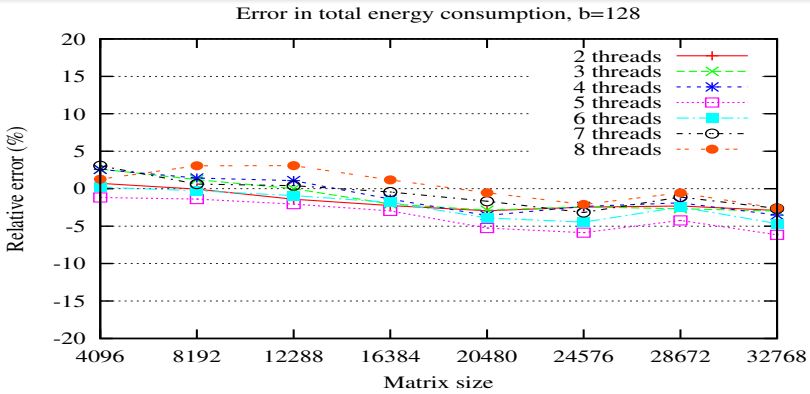
$$= (P^Y + P^S)\,T + \sum_{i=1}^{r}\sum_{j=1}^{c} P_i^D \left(\int_{t=0}^{T} N_{i,j}(t)\ dt\right) = (P^Y + P^S)\,T + \sum_{i=1}^{r}\sum_{j=1}^{c} P_i^D\, T_{i,j}$$

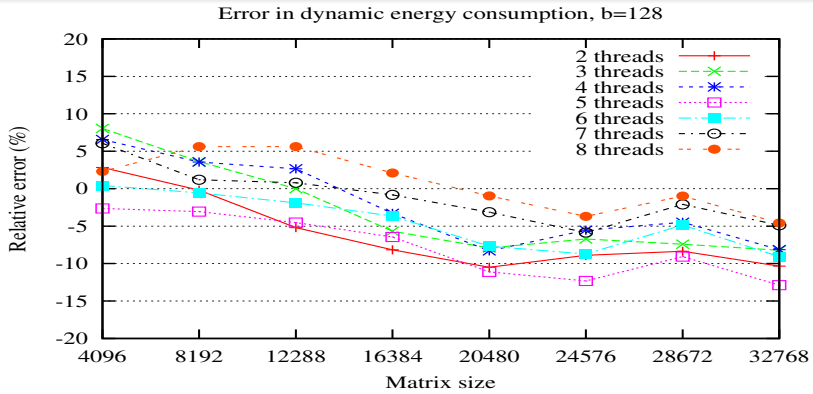
$T_{i,j}$ total execution time for task of type i onto the core j

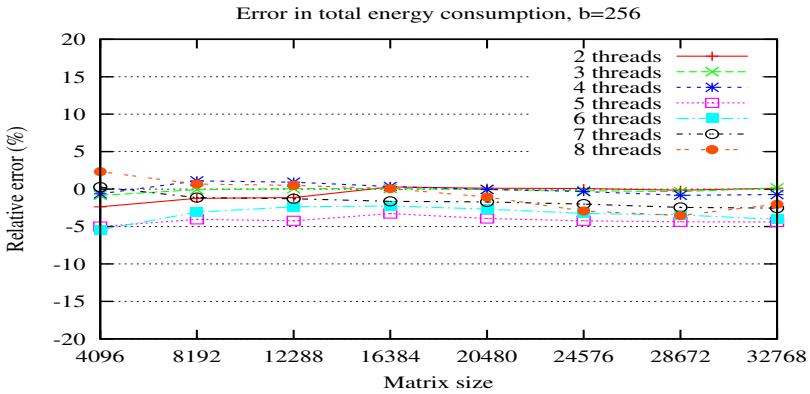
Experiments:

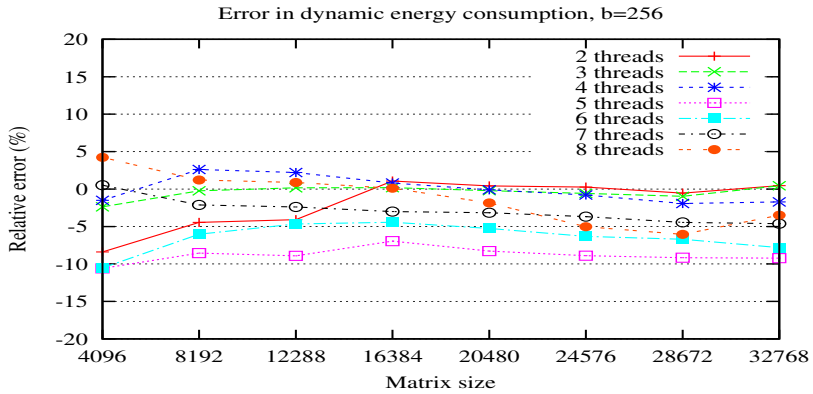
- Matrix sizes: $n = 4096, 8192, \ldots, 32768$
- Block sizes $b = 128, 256, 512$
- Cores/threads $c = 2, 3, \ldots, 8$

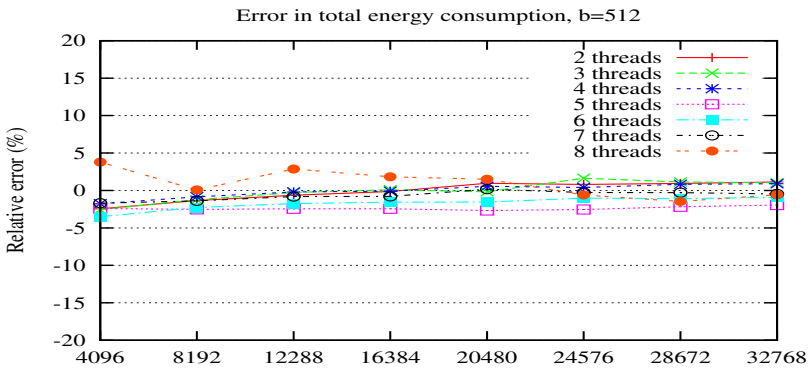
Experimental results

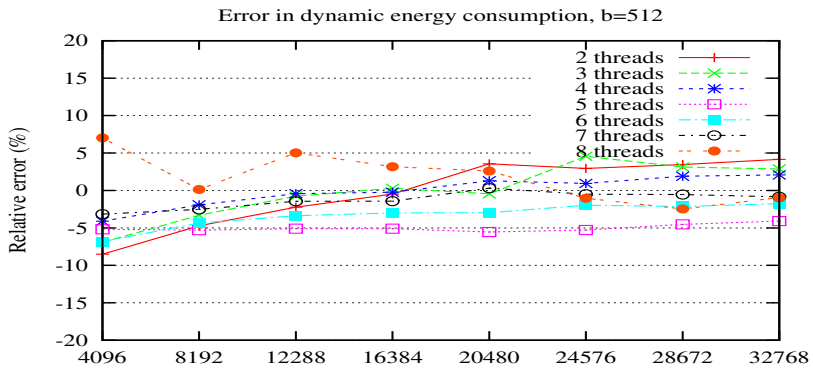












Conclusions and future work

Tools for power/energy analysis

- Detect code inefficiencies in order to reduce energy consumption
- Very useful to detect bottlenecks in the code:

  Performance inefficiency $\Rightarrow$ hot spots in hardware and power sinks in code

Energy-aware hardware/software

- *"Doing nothing well"*, D. E. Culler $\Rightarrow$ Avoid busy-waits when possible!
- Don't forget the cost of system+static power

Power modeling

- Evaluation of hybrid analytical-experimental model, based on a reduced group of experimental data
- Predict power consumed by applications without power measurement devices

Thanks to...

Universitat Jaume I:	E. S. Quintana-Ortí, R. Mayo, J. I. Aliaga, S. Barrachina, M. Barreda, S. Catalán
Univesitat Politècnica de València:	P. Alonso
Universidad Complutense Madrid:	F. D. Igual
Barcelona Supercomputing Center:	R. M. Badia, J. Planas
The University of Texas at Austin:	R. van de Geijn

Related publications

M. Barreda, M. F. Dolz, R. Mayo, E. S. Quintana-Ortí, R. Reyes
Binding Performance and Power of Dense Linear Algebra Operations
The 10th IEEE International Symposium on Parallel and Distributed Processing with Applications, 2012.

P. Alonso, R. M. Badia, J. Labarta, M. Barreda, M. F. Dolz, R. Mayo, E. S. Quintana-Ortí, R. Reyes
Tools for Power and Energy Analysis of Parallel Scientific Applications
The 41st International Conference on Parallel Processing, 2012.

M. Barreda, S. Catalán, M. F. Dolz, R. Mayo, E. S. Quintana-Ortí
Tracing the Power and Energy Consumption of the QR Factorization on Multicore Processors
12th International Conference on Computational and Mathematical Methods in Science and Engineering, 2012.

P. Alonso, M. F. Dolz, R. Mayo, E. S. Quintana-Ortí
Energy-efficient execution of dense linear algebra algorithms on multicore processors
Cluster Computing Journal, 2012

J. I. Aliaga, M. F. Dolz,, A. F. Martín, E. S. Quintana-Ortí
Leveraging task-parallelism in energy-efficient ILU preconditioners
2nd International Conference on ICT as Key Technology against Global Warming Held in conjunction with DEXA, 2012

P. Alonso, M. F. Dolz, F. D. Igual, R. Mayo, E. S. Quintana-Ortí
Reducing energy consumption of dense linear algebra operations on hybrid CPU-GPU platforms
The 10th IEEE International Symposium on Parallel and Distributed Processing with Applications, 2012.

Pedro Alonso, Manuel F. Dolz, Rafael Mayo, Enrique S. Quintana-Ortí
Modeling Power and Energy of the Task-Parallel Cholesky Factorization on Multicore Processors
3rd International Conference on Energy-Aware High Performance Computing. 2012.

Thanks for your attention!

Questions?