IC804/IC805 Cost Action Meeting

Tools and Models for Power and Energy Analysis of Parallel Scientific Applications

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

1 UNIVERSITAT POLITÈCNICA DE VALÈNCIA
2 UNIVERSITAT JAUME•I

May 31st – June 1st, 2012, Poznań (Poland)

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 three assistant researchers and one Ph.D. student.

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 ⇒ improve performance:
  - Higher number of cores per socket (processor)
- Large number of processors and cores ⇒ High energy consumption
- Tools to analyze performance and power in order to detect code inefficiencies and reduce energy consumption

Outline

Introduction

- **Parallel scientific applications**
  - Examples for dense linear algebra: Cholesky, QR and LU factorizations
- **Tools for power and energy analysis**
  - Power profiling in combination with Extrae+Paraver tools

Parallel applications + Power profiling

⇓

Environment to identify sources of power inefficiency

- **Power modeling**:
  - Predict power consumed by applications without power measurement devices even without executing them
  - Performance inefficiency normally results in hot spots in hardware and power sinks in source code

⇓

Energy savings

Introduction

- **Parallel scientific applications**
  - Examples for dense linear algebra: Cholesky, QR and LU factorizations
- **Tools for power and energy analysis**
  - Power profiling in combination with Extrae+Paraver tools

Parallel applications + Power profiling

⇓

Environment to identify sources of power inefficiency

- **Power modeling**:
  - Predict power consumed by applications without power measurement devices even without executing them
  - Performance inefficiency normally results in hot spots in hardware and power sinks in source code

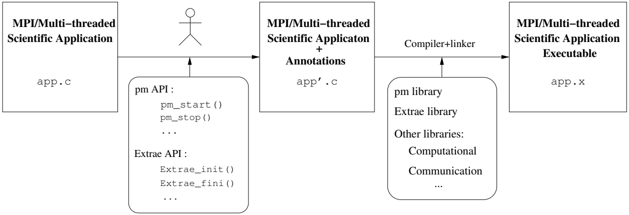
⇓

Energy savings

Tools for performance and power tracing

Why traces?

- Details and variability are important (along time, processors, etc.)
- Extremely useful to analyze performance of applications, also at power level!



- Scientific application `app.c`
- Application with annotated code `app'.c`
- Executable code `app.x`

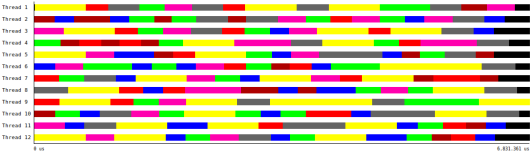
Tracing framework

`Extrae`: instrumentation and measurement package of BSC (Barcelona Supercomputing Center):

- Intercept calls to MPI, OpenMP, PThreads
- Records relevant information: time stamped events, hardware counter values, etc.
- Dumps all information into a single trace file.

`Paraver`: graphical interface tool from BSC to analyze/visualize trace files:

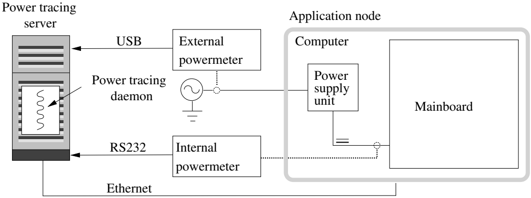
- Inspection of parallelism and scalability
- High number of metrics to characterize the program and performance application



Power measurement framework

pmlib library

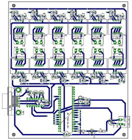
- Power measurement package of Jaume I University (Spain)
- Interface to interact and utilize our own power meters
- Also compatible with commercial power meters



- **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

Scientific application

Cholesky factorization:

$$A = U^T U$$

$A \in \mathbb{R}^{n \times n}$ symmetric definite positive (s.p.d.) matrix

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

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

Example of performance and power tracing with the Cholesky factorization:

- LAPACK routine `dpotrf`
- Shared-memory parallelism is extracted by calling to the multi-thread implementations of:
  - `dpotf2`, `dtrsm`, `dsyrk` kernels from Intel MKL, AMD ACML or IBM ESSL.

Code annotation

Cholesky factorization using LAPACK code:

```
#define A_ref(i,j) A[((j)-1)*Alda+((i)-1)]

void dpotrf( int n, int nb, double *A, int Alda, int *info ){



  for (k=1; k<=n; k+=nb) {
    // Factor current diagonal block

    dpotf2( nb, &A_ref(k,k), Alda, info );


    if( k+nb <= n ) {
      // Triangular solve

      dtrsm( "L", "U", "T", "N", nb, n-k-nb+1,
             &done, &A_ref( k, k ),     Alda,
                    &A_ref( k, k+nb ), Alda );


      // Update trailing submatrix

      dsyrk( "U", "T", n-k-nb+1, nb,
             &dmone, &A_ref( k, k+nb ),     Alda,
             &done,  &A_ref( k+nb, k+nb ), Alda );

    }
  }


}
```

Code annotation

Cholesky factorization using LAPACK code (Extrae routines):

```
#define A_ref(i,j) A[((j)-1)*Alda+((i)-1)]

void dpotrf( int n, int nb, double *A, int Alda, int *info ){


  Extrae_init();
  for (k=1; k<=n; k+=nb) {
    // Factor current diagonal block

    dpotf2( nb, &A_ref(k,k), Alda, info );


    if( k+nb <= n ) {
      // Triangular solve

      dtrsm( "L", "U", "T", "N", nb, n-k-nb+1,
             &done, &A_ref( k, k ),     Alda,
                    &A_ref( k, k+nb ), Alda );


      // Update trailing submatrix

      dsyrk( "U", "T", n-k-nb+1, nb,
             &dmone, &A_ref( k, k+nb ),     Alda,
             &done,  &A_ref( k+nb, k+nb ), Alda );

    }
  }
  Extrae_fini();

}
```

Code annotation

Cholesky factorization using LAPACK code (Extrae routines):

```
#define A_ref(i,j) A[((j)-1)*Alda+((i)-1)]

void dpotrf( int n, int nb, double *A, int Alda, int *info ){


  Extrae_init();
  for (k=1; k<=n; k+=nb) {
    // Factor current diagonal block
    Extrae_event(500000001,1);
    dpotf2( nb, &A_ref(k,k), Alda, info );
    Extrae_event(500000001,0);

    if( k+nb <= n ) {
      // Triangular solve
      Extrae_event(500000001,2);
      dtrsm( "L", "U", "T", "N", nb, n-k-nb+1,
             &done, &A_ref( k, k ),     Alda,
                    &A_ref( k, k+nb ), Alda );
      Extrae_event(500000001,0);

      // Update trailing submatrix
      Extrae_event(500000001,3);
      dsyrk( "U", "T", n-k-nb+1, nb,
             &dmone, &A_ref( k, k+nb ),     Alda,
             &done,  &A_ref( k+nb, k+nb ), Alda );
      Extrae_event(500000001,0);
    }
  }
  Extrae_fini();

}
```

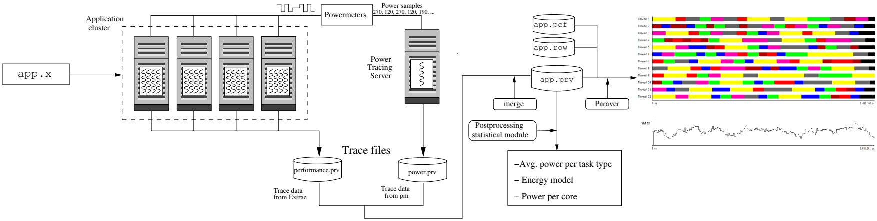
Code annotation

Cholesky factorization using LAPACK code (pmlib routines):

```
#define A_ref(i,j) A[((j)-1)*Alda+((i)-1)]

void dpotrf( int n, int nb, double *A, int Alda, int *info ){

  pm_start_counter(&pm_ctr);
  Extrae_init();
  for (k=1; k<=n; k+=nb) {
    // Factor current diagonal block
    Extrae_event(500000001,1);
    dpotf2( nb, &A_ref(k,k), Alda, info );
    Extrae_event(500000001,0);

    if( k+nb <= n ) {
      // Triangular solve
      Extrae_event(500000001,2);
      dtrsm( "L", "U", "T", "N", nb, n-k-nb+1,
             &done, &A_ref( k, k ),     Alda,
                    &A_ref( k, k+nb ), Alda );
      Extrae_event(500000001,0);

      // Update trailing submatrix
      Extrae_event(500000001,3);
      dsyrk( "U", "T", n-k-nb+1, nb,
             &dmone, &A_ref( k, k+nb ),    Alda,
             &done,  &A_ref( k+nb, k+nb ), Alda );
      Extrae_event(500000001,0);
    }
  }
  Extrae_fini();
  pm_stop_counter(&pm_ctr);
}
```

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.

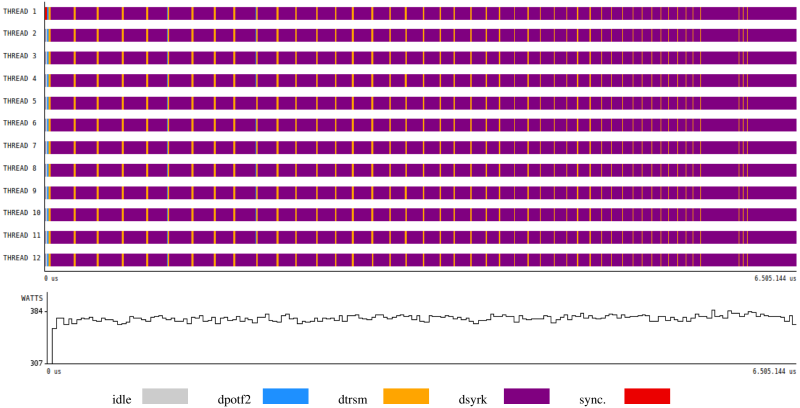
Experimental results

Experiments:

- Cholesky and LU factorization with partial pivoting from LAPACK and Intel MKL (`dgetrf` routine)
- Block size $b = 256$
- Matrix size $16,384$
- 12 cores
- Environment setup:
  - 4x AMD 6172 processors (total of 48 cores) (2.00 GHz) with 256 Gbytes of RAM
  - Powermeter: Internal ASIC @ 25 Hz

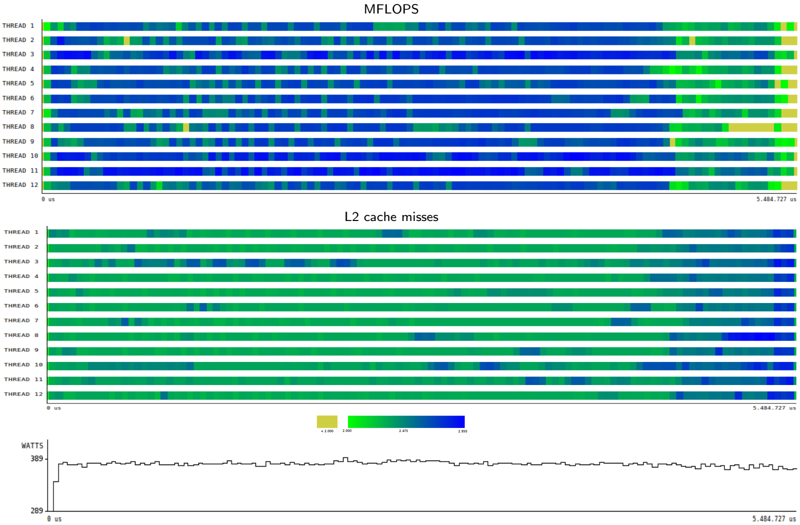
Experimental results

Cholesky factorization from LAPACK (`dpotrf`)



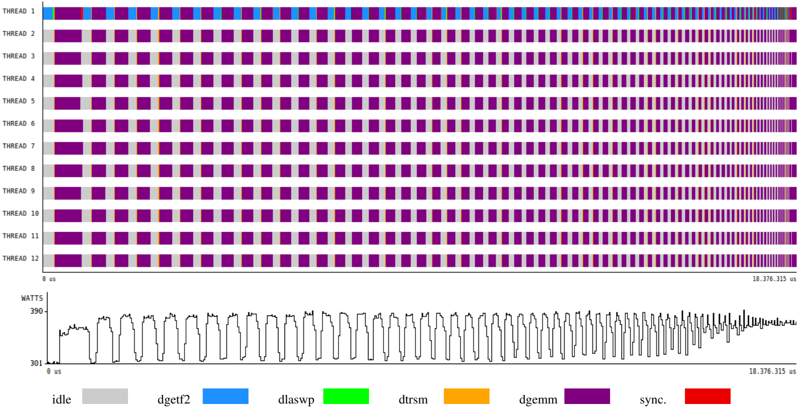
Experimental results

Cholesky factorization from MKL (dpotrf)



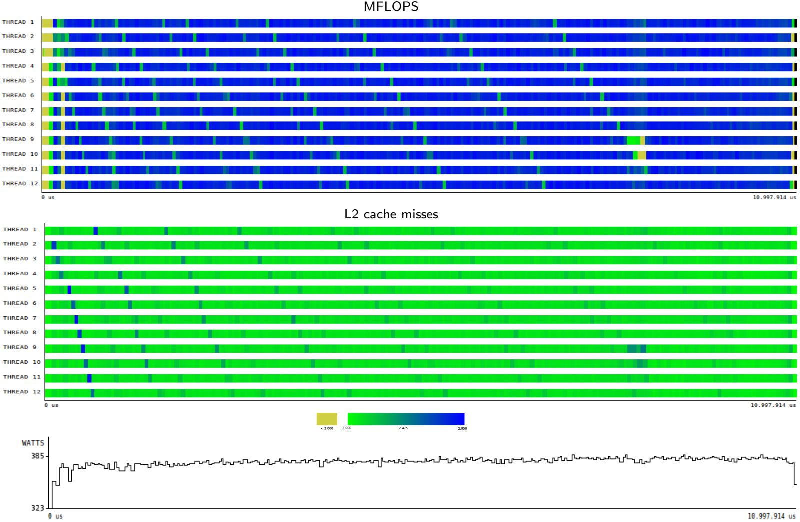
Experimental results

LU factorization with partial pivoting from LAPACK (`dgetrf`)



Experimental results

LU factorization with partial pivoting from MKL (`dgetrf`)



Power model

Power model:

$$P = P^{C(PU)} + P^{U(ncore)} = P^{S(tatic)} + P^{D(ynamic)} + P^{U(ncore)}$$

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

$P^{U(uncore)}$ Power of remaining components (e.g. RAM)

Considerations:

- Study case: **Cholesky factorization**. It exercises CPU+RAM and discards other power sinks (network interface, PSU, etc.)
- We assume P^U 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

Power model

Power model:

$$P = P^{C(PU)} + P^{U(ncore)} = P^{S(tatic)} + P^{D(ynamic)} + P^{U(ncore)}$$

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

$P^{U(uncore)}$ Power of remaining components (e.g. RAM)

Considerations:

- Study case: **Cholesky factorization**. It exercises CPU+RAM and discards other power sinks (network interface, PSU, etc.)
- We assume P^U 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

Power model

Power model:

$$P = P^{C(PU)} + P^{U(ncore)} = P^{S(tatic)} + P^{D(ynamic)} + P^{U(ncore)}$$

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

$P^{U(uncore)}$ Power of remaining components (e.g. RAM)

Considerations:

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

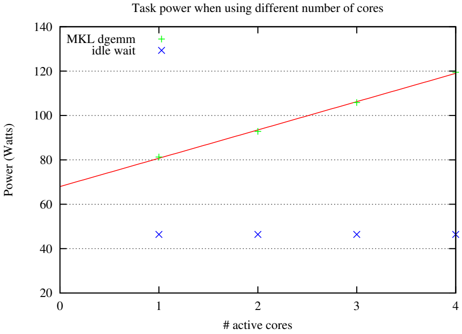
Environment setup:

- 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

Uncore and static power

Obtaining $P^{U(uncore)}$ and $P^{S(tatic)}$ components:

- P^U directly obtained measuring idle platform: $P^U = 46.37\,Watts$
- P^S obtained by executing dgemm kernel using 1 to 4 cores and adjusting via linear regression:



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

$$P^S \approx \alpha - P^U = 67.97 - 46.37 = 21.6 \text{ Watts}$$

Dynamic power

Dynamic power of kernels of the Cholesky factorization:

- To obtain P_K^D we continuously invoke the kernel K until power stabilizes and then sample this value. Example for dgemm:

$$P_G^D = P_{\mathtt{dgemm}} - P^S - P^U = P_{\mathtt{dgemm}} - 67.97\ Watts$$

	1 kernel mapped to 1 core				2 kernels mapped to 2 cores of different sockets			
	Block size, b				Block size, b			
Task	128	192	256	512	128	192	256	512
P_P^D (dpotrf)	10.26	10.35	10.45	11.28	9.05	9.09	9.28	10.44
P_T^D (dtrsm)	10.12	10.31	10.32	10.80	9.45	9.57	9.60	11.08
P_S^D (dsyrk)	11.22	11.47	11.67	12.60	10.42	10.63	10.82	11.80
P_G^D (dgemm)	11.98	12.54	12.72	13.30	10.90	12.16	11.28	11.96
P_B^D (busy)	7.62	7.62	7.62	7.62	7.62	7.62	7.62	7.62

- Power increases linearly with the number of threads, from 1 to 4 mapped to a single core
- When two sockets are used, linear function changes, so we take into account this issue:

$$P_G^D = \frac{P_{\mathtt{dgemm}-67.97}}{2}$$

Power/energy model testing

Power model:

$$P_{Chol}(t) = P^U + P^S + P^D_{Chol}(t) = P^U + P^S + \sum_{i=1}^{r}\sum_{j=1}^{c} P^D_i N_{i,j}(t)$$

- r stands for the number of different types of tasks, (r=5 for Cholesky)
- c stands for the number of threads/cores
- P^D_i 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:

$$E_{Chol} = (P^U + P^S)T + \int_{t=0}^{T} P^D_{Chol}(t)$$

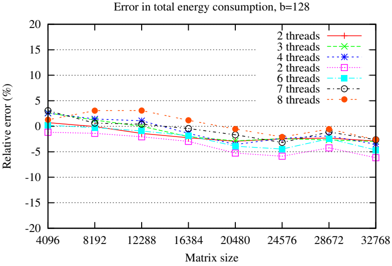
$$= (P^U + P^S)T + \sum_{i=1}^{r}\sum_{j=1}^{c} P^D_i \left(\int_{t=0}^{T} N_{i,j}(t)\right) = (P^U + P^S)T + \sum_{i=1}^{r}\sum_{j=1}^{c} P^D_i T_{i,j}$$

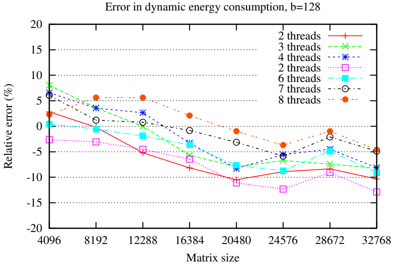
- $T_{i,j}$ total execution time for task of type i onto the core j

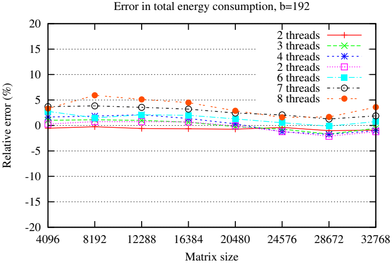
Experiments:

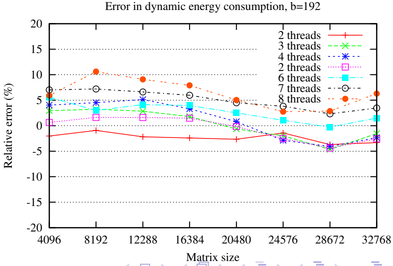
- Matrix sizes: $n = 4096, 8192, \ldots, 32768$
- Block sizes $b = 128, 192, 256, 512$
- Cores/threads $c = 2, 3, \ldots, 8$

Experimental results

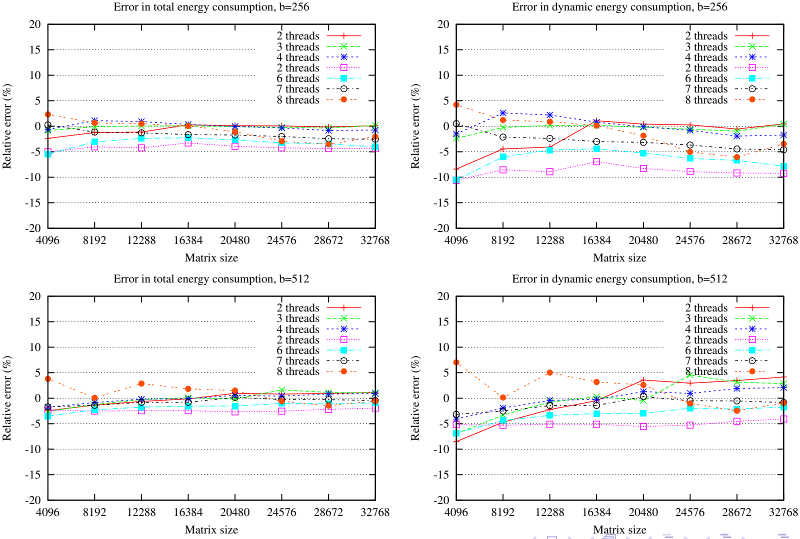








Experimental result



Related publications

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

Pedro Alonso, Rosa M. Badia, Jesus Labarta, Maria Barreda, Manuel F. Dolz, Rafael Mayo, Enrique S. Quintana-Ortí, Ruymán Reyes
Tools for Power and Energy Analysis of Parallel Scientific Applications
The 41st International Conference on Parallel Processing, 2012.

Maria Barreda, Sandra Catalán, Manuel F. Dolz, Rafael Mayo, Enrique 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.

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

Conclusions and future work

Performance and power tracing:

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

Power model:

- Evaluation of hybrid analytical-experimental model, based on a reduced group of experimental data
- High accuracy in the estimated total energy ($\pm 5\%$) and estimated dynamic energy ($\pm 15\%$)

Future work:

- Developing models for numerical libraries

Thanks for your attention!

Questions?