International Conference on
Energy-Aware High Performance Computing

Modeling Power and Energy of the Task-Parallel Cholesky Factorization on Multicore Processors

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

1 Universitat Politècnica de València

2 Universitat Jaume I

September 12, 2012, Hamburg (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
- 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 performance/tracing tools for HPC

Parallel applications + Power profiling

$\Downarrow$

Is it possible to predict power/energy consumption?

- **Objective**: Power modeling
  - Predict power consumed by applications without power measurement devices.
  - Estimations are needed to determine how to address the power-challenge for energy-efficient hardware and software

$\Downarrow$

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 performance/tracing tools for HPC

Parallel applications + Power profiling

$\Downarrow$

Is it possible to predict power/energy consumption?

- **Objective**: Power modeling
  - Predict power consumed by applications without power measurement devices.
  - Estimations are needed to determine how to address the power-challenge for energy-efficient hardware and software

$\Downarrow$

Energy savings

Algorithm specification

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

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

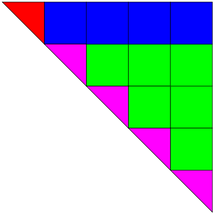
for $k = 1, 2, \ldots, s$ **do**
  $A_{kk} = U_{kk}^T U_{kk}$
  for $j = k+1, k+2, \ldots, s$ **do**
    $A_{kj} \leftarrow A_{kj} U_{kk}^{-T}$
  end for
  for $i = k+1, k+2, \ldots, s$ **do**
    $A_{ii} \leftarrow A_{ii} - A_{ki}^T A_{ki}$
    for $j = i+1, i+2, \ldots, s$ **do**
      $A_{ij} \leftarrow A_{ij} - A_{ki}^T A_{kj}$
    end for
  end for
end for

CHOL: CHOLESKY FACTORIZATION

TRSM: TRIANGULAR SYSTEM SOLVE

SYRK: SYMMETRIC RANK-b UPDATE

GEMM: MATRIX-MATRIX PRODUCT



Iteration 1

Algorithm specification

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

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

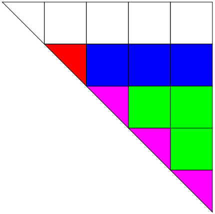
for $k = 1, 2, \ldots, s$ **do**
  $A_{kk} = U_{kk}^T U_{kk}$
  for $j = k+1, k+2, \ldots, s$ **do**
    $A_{kj} \leftarrow A_{kj} U_{kk}^{-T}$
  end for
  for $i = k+1, k+2, \ldots, s$ **do**
    $A_{ii} \leftarrow A_{ii} - A_{ki}^T A_{ki}$
    for $j = i+1, i+2, \ldots, s$ **do**
      $A_{ij} \leftarrow A_{ij} - A_{ki}^T A_{kj}$
    end for
  end for
end for

CHOL: CHOLESKY FACTORIZATION

TRSM: TRIANGULAR SYSTEM SOLVE

SYRK: SYMMETRIC RANK-b UPDATE

GEMM: MATRIX-MATRIX PRODUCT



Iteration 2

Algorithm specification

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

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

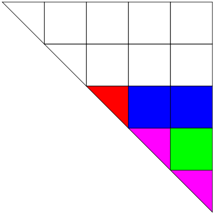
for $k = 1, 2, \ldots, s$ **do**
  $A_{kk} = U_{kk}^T U_{kk}$
  for $j = k+1, k+2, \ldots, s$ **do**
    $A_{kj} \leftarrow A_{kj} U_{kk}^{-T}$
  end for
  for $i = k+1, k+2, \ldots, s$ **do**
    $A_{ii} \leftarrow A_{ii} - A_{ki}^T A_{ki}$
    for $j = i+1, i+2, \ldots, s$ **do**
      $A_{ij} \leftarrow A_{ij} - A_{ki}^T A_{kj}$
    end for
  end for
end for

CHOL: CHOLESKY FACTORIZATION

TRSM: TRIANGULAR SYSTEM SOLVE

SYRK: SYMMETRIC RANK-b UPDATE

GEMM: MATRIX-MATRIX PRODUCT



Iteration 3

Algorithm specification

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

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

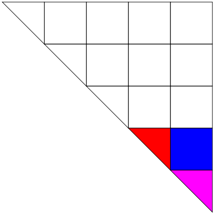
for $k = 1, 2, \ldots, s$ **do**
  $A_{kk} = U_{kk}^T U_{kk}$
  for $j = k+1, k+2, \ldots, s$ **do**
    $A_{kj} \leftarrow A_{kj} U_{kk}^{-T}$
  end for
  for $i = k+1, k+2, \ldots, s$ **do**
    $A_{ii} \leftarrow A_{ii} - A_{ki}^T A_{ki}$
    for $j = i+1, i+2, \ldots, s$ **do**
      $A_{ij} \leftarrow A_{ij} - A_{ki}^T A_{kj}$
    end for
  end for
end for

Chol: Cholesky factorization

Trsm: Triangular system solve

Syrk: Symmetric rank-b update

Gemm: Matrix-matrix product



Iteration 4

Algorithm specification

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

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

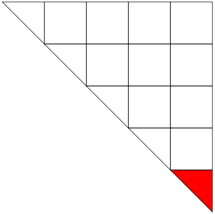
for $k = 1, 2, \ldots, s$ **do**
  $A_{kk} = U_{kk}^T U_{kk}$
  for $j = k+1, k+2, \ldots, s$ **do**
    $A_{kj} \leftarrow A_{kj} U_{kk}^{-T}$
  end for
  for $i = k+1, k+2, \ldots, s$ **do**
    $A_{ii} \leftarrow A_{ii} - A_{ki}^T A_{ki}$
    for $j = i+1, i+2, \ldots, s$ **do**
      $A_{ij} \leftarrow A_{ij} - A_{ki}^T A_{kj}$
    end for
  end for
end for

Chol: Cholesky factorization

Trsm: Triangular system solve

Syrk: Symmetric rank-b update

Gemm: Matrix-matrix product



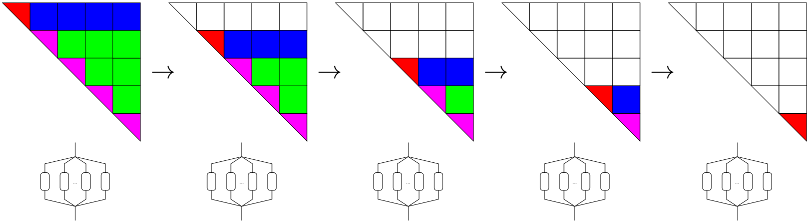
Iteration 5

Parallelization $\Rightarrow$ Not trivial at code level!

Parallelization

Option 1: Use **multi-threaded BLAS**

- Straightforward approach towards LAPACK-level parallelization
- Highly tuned multi-threaded kernels: Intel MKL, AMD ACML or IBM ESSL,...
- Fork/join approach: parallelism is not fully exploited



Parallelization

Option 2: Use a **runtime task scheduler**

- We use SMPSs runtime-compiler framework to exploit task-parallelism
- Functions in code are annotated as tasks using OpenMP-like pragmas `#pragma css task`
- Operations are not executed in the order they appear in the code but respecting data dependencies
- SMPSs easily obtains performance traces which can be analyzed using *Paraver* (Performance analysis tools from Barcelona Supercomputing Center)

SMPSs 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: Right-looking Cholesky DAG with a matrix consisting of 4×4 blocks

Parallelization

Option 2: Use a **runtime task scheduler**

- We use SMPSs runtime-compiler framework to exploit task-parallelism
- Functions in code are annotated as tasks using OpenMP-like pragmas `#pragma css task`
- Operations are not executed in the order they appear in the code but respecting data dependencies
- SMPSs easily obtains performance traces which can be analyzed using *Paraver* (Performance analysis tools from Barcelona Supercomputing Center)

SMPSs 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

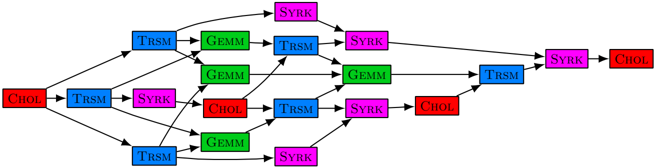


Figure: Right-looking Cholesky DAG with a matrix consisting of 4 $\times$ 4 blocks

Cholesky factorization with SMPSs pragma annotations

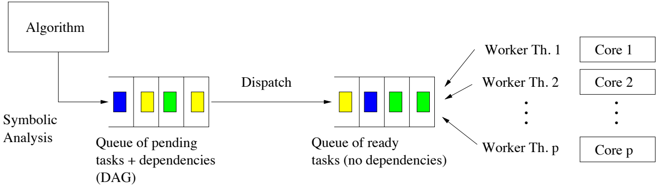
```
void dpotrf_smpss( int n, int b, double *A, int Alda, int *info ){
  for (k=1; k<=n; k+=b) {
   dpotrf_u( b, &A_ref(k,k), Alda, info );
    if( k+b <= n ) {
      for (j=k+b; k<=n; k+=b)
       dtrsm_lutn( b, &A_ref( k, k ), &A_ref( k, j ), Alda );
      for (i=k+b; i<=n; i+=b) {
       dsyrk_ut( b, &A_ref( k, i ), &A_ref( i, i ), Alda );
        for (j=i+b; j<=n; j+=b)
          dgemm_tn( b, &A_ref( k, i ), &A_ref( k, j ), &A_ref( i, j ), Alda );
      }
    }
  }
}

void dpotrf_u( int b, double A[], int ldm, int *info ){
  dpotrf( "Upper", &b, A, &ldm, info );
}

void dtrsm_lutn( int b, double A[], double B[], int ldm ){
  double done = 1.0;
  dtrsm( "Left", "Upper", "Transpose", "Non_unit", &b, &b, &done, A, &ldm, B, &ldm );
}

void dsyrk_ut( int b, double A[],  double C[], int ldm ){
  double dmone = -1.0, done = 1.0;
  dsyrk( "Upper", "Transpose", &b, &b, &dmone, A, &ldm, &done,  C, &ldm );
}

void dgemm_tn( int b, double A[], double B[], double C[], int ldm ){
  double dmone = -1.0, done = 1.0;
  dgemm( "Transpose", "No_transpose", &b, &b, &b, &dmone, A, &ldm, B, &ldm, &done,  C, &ldm );
}
```

Cholesky factorization with SMPSs pragma annotations

```
void dpotrf_smpss( int n, int b, double *A, int Alda, int *info ){
  for (k=1; k<=n; k+=b) {
   dpotrf_u( b, &A_ref(k,k), Alda, info );
    if( k+b <= n ) {
      for (j=k+b; k<=n; k+=b)
        dtrsm_lutn( b, &A_ref( k, k ), &A_ref( k, j ), Alda );
      for (i=k+b; i<=n; i+=b) {
        dsyrk_ut( b, &A_ref( k, i ), &A_ref( i, i ), Alda );
         for (j=i+b; j<=n; j+=b)
           dgemm_tn( b, &A_ref( k, i ), &A_ref( k, j ), &A_ref( i, j ), Alda );
      }
    }
  }
}
#pragma css task input( b, ldm ) inout( A[1], info[1] )
void dpotrf_u( int b, double A[], int ldm, int *info ){
  dpotrf( "Upper", &b, A, &ldm, info );
}
#pragma css task input( b, A[1], ldm ) inout( B[1] )
void dtrsm_lutn( int b, double A[], double B[], int ldm ){
  double done = 1.0;
  dtrsm( "Left", "Upper", "Transpose", "Non_unit", &b, &b, &done, A, &ldm, B, &ldm );
}
#pragma css task input( b, A[1], ldm ) inout( C[1] )
void dsyrk_ut( int b, double A[],  double C[], int ldm ){
  double dmone = -1.0, done = 1.0;
  dsyrk( "Upper", "Transpose", &b, &b, &dmone, A, &ldm, &done,  C, &ldm );
}
#pragma css task input( b, A[1], B[1], ldm ) inout( C[1] )
void dgemm_tn( int b, double A[], double B[], double C[], int ldm ){
  double dmone = -1.0, done = 1.0;
  dgemm( "Transpose", "No_transpose", &b, &b, &b, &dmone, A, &ldm, B, &ldm, &done,  C, &ldm );
}
```

SMPSs operation

SMPSs 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.

Power model formulation

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)

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!

Power model formulation

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)

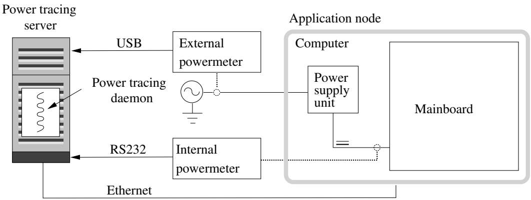
Considerations:

- 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

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
- Performance and tracing modes are enabled
- Power measurements: `pmlib` library

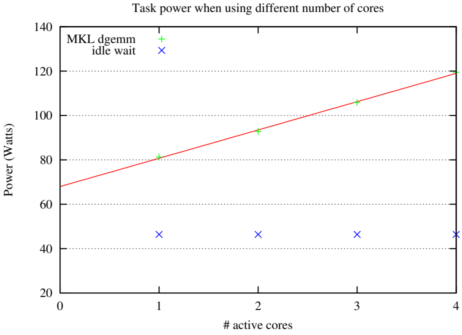


- Internal power meter:
  - ASIC-based powermeter (own design!)
  - LEM HXS 20-NP transductors with PIC microcontroller
  - Sampling rate: 25 Hz

System and static power

Obtaining $P^{S(Y)stem}$ and $P^{S(tatic)}$ 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:



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

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

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`:

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

	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:

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

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:

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

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

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

Experimental model evaluation:

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

Power/energy model testing

Power model:

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

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:

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

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

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

Experimental model evaluation:

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

Power/energy model testing

Power model:

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

- 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:

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

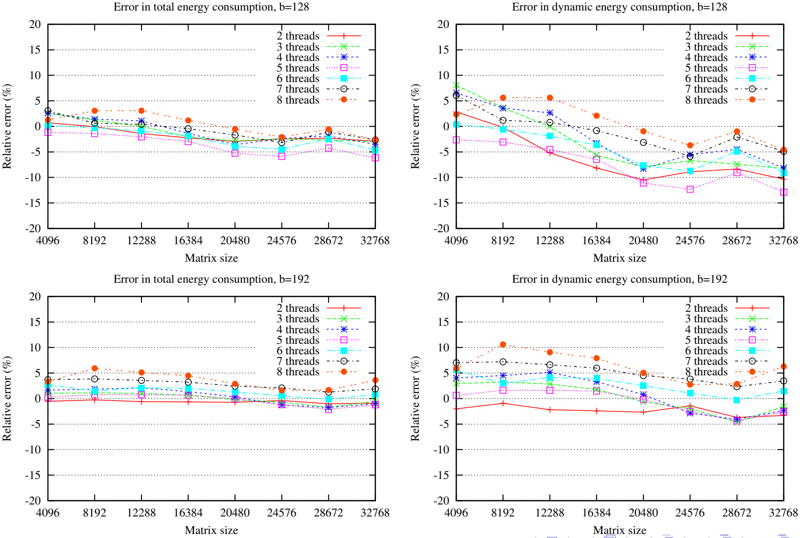
$$= (P^Y + P^S)T + \sum_{i=1}^{r} \sum_{j=1}^{c} P_i^D \left(\int_{t=0}^{T} N_{i,j}(t) \right) = (P^Y + P^S)T + \sum_{i=1}^{r} \sum_{j=1}^{c} P_i^D T_{i,j}$$

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

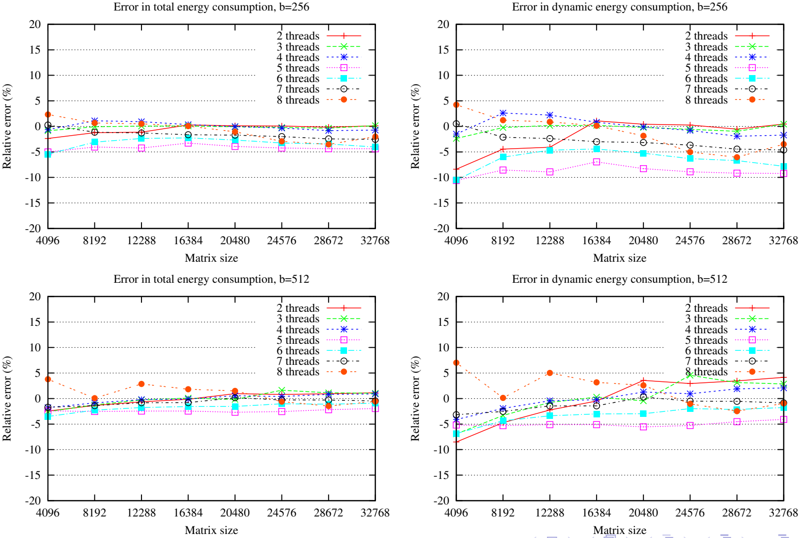
Experimental model evaluation:

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

Energy model evaluation



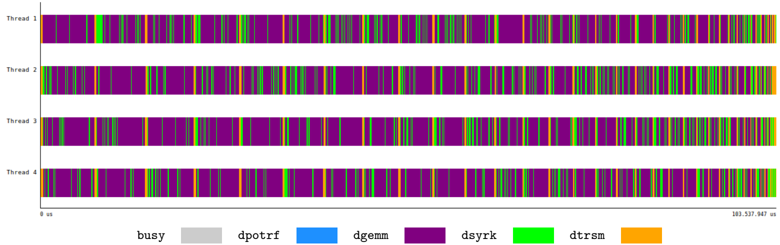
Energy model evaluation

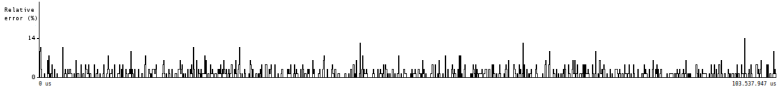


Power model evaluation

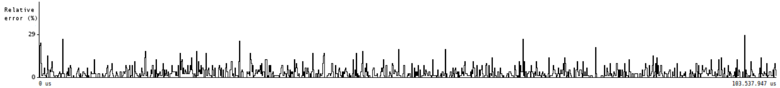
Reconstruction of power profile using the power model $\Rightarrow$ Performance trace is needed!

Trace of Cholesky factorization of order $n = 20,490$ and block size $b = 512$, using 4 cores





Relative error in the estimated total power (reconstruction); Average error of 2.92%



Relative error in the estimated dynamic power (reconstruction); Average error of 6.85%

Conclusions and future work

Conclusions:

- Elaboration and validation of an hybrid analytical-experimental model to estimate power/energy for the Cholesky factorization
- Experimental results reveal the accuracy of the model:
  - Energy consumption estimation: $\pm 5\%$ and $\pm 15\%$ of error for the total and dynamic energy, respectively
  - Power profile estimation: relative average error of 2.92% and 6.85% for total and dynamic power, respectively
- However, it is easier to obtain an energy estimation than a power profile estimation due to inaccuracy of power meter (around $\pm 5\%$)!

Future work:

- Predict power/energy even without executing the code!
- Initial step towards more ambitious goal $\Rightarrow$ Development of models for the functionality of LAPACK
- Model extension to task-parallel procedures for distributed-memory platforms

Thanks for your attention!

Questions?