Binding Performance and Power of Dense Linear Algebra Operations

Maria Barreda, Manuel F. Dolz, Rafael Mayo, Enrique S. Quintana-Ortí, Ruymán Reyes

July 11th, 2012, Leganés – Madrid (Spain)
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

1 Introduction

2 Tools for performance and power tracing
   • Performance tracing framework
   • Power tracing framework
   • Example

3 Experimental results
   • Environment setup
   • LU factorization
   • Cholesky factorization
   • Reduction to tridiagonal form
   • Results

4 Conclusions
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

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

\[ \Downarrow \]

Environment to identify sources of power inefficiency

\[ \Downarrow \]

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 and commercial power meters

![Diagram of power measurement framework]

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

**Power meter:**
- ASIC-based powermeter (own design!)
- LEM HXS 20-NP transductors with PIC microcontroller
- Sampling rate 25 Hz
Scientific application

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

- 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

Example of performance and power tracing with the LU factorization:

- LAPACK routine dgetrf
- Shared-memory parallelism is extracted by calling to the multi-thread implementations of:
  - dgetf2, dlaswp, dtrsm and dgemm kernels from Intel MKL, AMD ACML or IBM ESSL.
Code annotation

LU factorization using LAPACK code:

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

void dgetrf( int m, int n, int b, double *A, int Alda, int *ipiv, int *info ){
    // Declaration of variables (omitted)

    for (j=1; j<=min(m, n); j+=b) {
        // Factor current panel
        dgetf2( m-j+1, b, &Aref(j,j), Alda, &ipiv[j-1], info );

        // Apply permutations to left and right of panel
        dlaswp( j-1, A, Alda, j, j+b-1, ipiv, 1 );
        dlaswp( n-j-b+1, &Aref(1, j+b), Alda, j, j+b-1, ipiv, 1 );

        // Triangular solve
        dtrsm( "L", "L", "N", "U", b, n-j-b+1, done, &Aref(j,j), Alda, &Aref( j, j+b ), Alda );

        // Update trailing submatrix
        dgemm( "N", "N", m-j-b+1, n-j-b+1, b, done, &Aref(j+b, j), Alda, &Aref( j, j+b ), Alda, done, &Aref( j+b, j+b ), Alda );
    }
}
```
LU factorization using LAPACK code (Extrae routines):

```c
#define Aref(i, j) A[((j)−1)*Alda+(i)−1]

void dgetrf( int m, int n, int b, double *A, int Alda, int *ipiv, int *info ){
    // Declaration of variables (omitted)
    Extrae_init();
    for (j=1; j<=min( m, n ); j+=b) {
        // Factor current panel
        dgetf2( m−j+1, b, &Aref(j, j), Alda, &ipiv[j−1], info );
        // Apply permutations to left and right of panel
        dlaswp( j−1, A, Alda, j, j+b−1, ipiv, 1 );
        dlaswp( n−j−b+1, &Aref( 1, j+b ), Alda, j, j+b−1, ipiv, 1 );
        // Triangular solve
        dtrsm( "L", "L", "N", "U", b, n−j−b+1, done, &Aref( j, j ), Alda, &Aref( j, j+b ), Alda );
        // Update trailing submatrix
        dgemm("N", "N", m−j−b+1, n−j−b+1, b, done, &Aref( j+b, j ), Alda,
              &Aref( j, j+b ), Alda, done, &Aref( j+b, j+b ), Alda );
    }
    Extrae_fin();
}
```

LU factorization using LAPACK code (Extrae routines):

```c
#define Aref(i,j) A[((j)-1)*Allda+((i)-1)]
void dgetrf( int m, int n, int b, double *A, int Allda, int *ipiv, int *info ){
    // Declaration of variables (omitted)
    Extrae_init();
    for (j=1; j<=min(m,n); j+=b) {
        Extrae_event(500000001,1);
        // Factor current panel
        dgetf2( m-j+1, b, &Aref(j,j), Allda, &ipiv[j-1], info );
        Extrae_event(500000001,0);
        Extrae_event(500000001,2);
        // Apply permutations to left and right of panel
        dlaswp( j-1, A, Allda, j, j+b-1, ipiv, 1 );
        dlaswp( n-j-b+1, &Aref(1,j+b), Allda, j, j+b-1, ipiv, 1 );
        Extrae_event(500000001,0);
        Extrae_event(500000001,3);
        // Triangular solve
        dtrsm( "L", "L", "N", "U", b, n-j-b+1, done, &Aref(j,j), Allda, &Aref(j,j+b), Allda );
        Extrae_event(500000001,0);
        Extrae_event(500000001,4);
        // Update trailing submatrix
        dgemm( "N", "N", m-j-b+1, n-j-b+1, b, done, &Aref(j+b,j), Allda,
               &Aref(j,j+b) , Allda, done, &Aref(j+b,j+b), Allda );
        Extrae_event(500000001,0);
    }
    Extrae_fini();
}
```

LU factorization using LAPACK code (pmlib routines):

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

void dgetrf( int m, int n, int b, double *A, int Alda, int *ipiv, int *info ){
    // Declaration of variables (omitted)
    pm_start_counter(&pm_ctr);
    Extrae_init();
    for (j=1; j<=min(m, n); j+=b) {
        Extrae_event(500000001,1);
        // Factor current panel
        dget2( m-j+1, b, &Aref(j, j), Alda, &ipiv[j-1], info );
        Extrae_event(500000001,0);
        Extrae_event(500000001,2);
        // Apply permutations to left and right of panel
        dlaswp( j-1, A, Alda, j, j+b-1, ipiv, 1 );
        dlaswp( n-j-b+1, &Aref(1, j+b), Alda, j, j+b-1, ipiv, 1 );
        Extrae_event(500000001,0);
        Extrae_event(500000001,3);
        // Triangular solve
        dtrsm( "L", "L", "N", "U", b, n-j-b+1, done, &Aref(j, j), Alda, &Aref(j, j+b), Alda );
        Extrae_event(500000001,0);
        Extrae_event(500000001,4);
        // Update trailing submatrix
        dgemm( "N", "N", m-j-b+1, n-j-b+1, b, done, &Aref(j+b, j), Alda,
               &Aref(j, j+b), Alda, done, &Aref(j+b, j+b), Alda );
    }
    Extrae_fini();
    pm_stop_counter(&pm_ctr);
}
```

Manuel F. Dolz et al
Binding Performance and Power of Dense Linear Algebra Operations
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
Experimental results

Environment setup:

- 4 AMD Opteron 6172 processors, 4x12 cores at 2.1 GHz, 256 GB of RAM
- Intel MKL (v10.3.9) using IEEE double-precision arithmetic
- Performance traces obtained with Extrae (v2.2.0) and Paraver (v4.1.0)
- Power traces obtained with our power library pmlib (v2.0) and a microcontroller-based internal powermeter measuring 12 V motherboard lines at 25 samples/sec.
- Problem size: $n=10,240$
Implementations

**LAPACK**
- Netlib routines for:
  - LU factorization with partial pivoting (`dgetrf`)
  - Cholesky factorization (`dpotrf`)
  - Reduction to tridiagonal form (`dsytrd`)
- Parallelism exploited within the invocations to Intel (multi-threaded)
- 12 cores and block size $b=128$
- Routine `dpotrf` was modified to compute the Cholesky factorization via a right-looking algorithmic variant

**MKL**
- Intel MKL routines for:
  - LU factorization with partial pivoting (`dgetrf`)
  - Cholesky factorization (`dpotrf`)
  - Reduction to tridiagonal form (`dsytrd`)
- 12 cores and block size $b=128$

**SMPSs**
- C codes for:
  - LU factorization with incremental pivoting
  - Cholesky factorization
- Linked to the sequential MKL BLAS, with task-level parallelism extracted by the SMPSs runtime system
- 6 cores, block size $b=256$ and internal block size $ib=64$
Experimental results: LU factorization

LU factorization with partial pivoting from LAPACK (dgetrf)

- Sequential execution of dgetf2 and dlaswp (low power) and parallel execution for dtrsm and dgemm (high power)
- Synchronization points after dgemm execution, due to unbalanced distribution of work among cores
Experimental results: LU factorization

LU factorization with partial pivoting from MKL (dgetrf)

- \texttt{dgemm} and \texttt{dtrsm} are BLAS-3, thus deliver a high MFLOPS rate
- \texttt{dgetrf} is performed by only one core but overlapped with matrix updates (MKL code uses look-ahead techniques)
- Synchronization point at the end of execution ⇒ Algorithmic reasons
**Experimental results: LU factorization**

LU factorization with incremental pivoting parallelized with SMPS

- **Kernels**
  - thread 1
  - thread 2
  - thread 3
  - thread 4
  - thread 5
  - thread 6
  - thread 7
  - thread 8
  - thread 9
  - thread 10
  - thread 11
  - thread 12

- **Watts**
  - idle
  - dgetrf
  - dgetrf2x1
  - dtrsm
  - dgemm2x1
  - sync.

- **Observations**
  - `dgemm2x1` dominates the execution time of the algorithm.
  - Plain power profile corresponding to `dgemm2x1` BLAS-3 kernel and the lack of idle periods.

---

**Manuel F. Dolz et al.**

*Binding Performance and Power of Dense Linear Algebra Operations*
Experimental results: Cholesky factorization

Cholesky factorization from LAPACK (dpotrf)

- Synchronization points due to unbalanced distribution of work among cores during dsyrk kernel ⇒ Idle periods
- Idle periods are so short and do not exert a visible change in the power profile
Experimental results: Cholesky factorization

Cholesky factorization from MKL (dpotrf)

- High variability in MFLOPS rate taking into account that most of the operations are BLAS-3.
- About 3/4 of the execution time a drastic decrease of MFLOPS is done ⇒ Change in MKL algorithm strategy.
- Plain power profile even decreasing MFLOPS rate.
Experimental results: Cholesky factorization

Cholesky factorization parallelized with SMPS

- Better performance and low energy consumption of the SMPSs parallelization compared with the LAPACK and MKL implementations

Experimental results: Reduction to tridiagonal form

Interleaved execution of serial (dsymv) and parallel phases (dsyr2k)

dsymv becomes a bottleneck because of the lack of concurrency of MKL implementation and low MFLOPS rate
Experimental results: Reduction to tridiagonal form

Reduction to tridiagonal form from MKL (dsytrd)

- Alternates periods of low and high activity for MFLOPS rate at high frequency!
- MKL employs a narrow block size to reduce latency of the panel factorization
Experimental results

Comparative table for evaluated algorithms and implementations:

<table>
<thead>
<tr>
<th></th>
<th>LU factorization</th>
<th></th>
<th>Cholesky factorization</th>
<th></th>
<th>Reduction to tridiagonal form</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LAPACK</td>
<td>MKL</td>
<td>SMPSs</td>
<td>LAPACK</td>
<td>MKL</td>
<td>SMPSs</td>
</tr>
<tr>
<td>( T (s) )</td>
<td>18.37</td>
<td>10.99</td>
<td>13.25</td>
<td>6.50</td>
<td>5.48</td>
<td>5.09</td>
</tr>
<tr>
<td>GFLOPS</td>
<td>38.96</td>
<td>65.13</td>
<td>54.02</td>
<td>55.06</td>
<td>65.31</td>
<td>70.31</td>
</tr>
<tr>
<td>( P_{\text{max}} ) (W)</td>
<td>390.70</td>
<td>385.78</td>
<td>392.81</td>
<td>384.61</td>
<td>389.06</td>
<td>393.52</td>
</tr>
<tr>
<td>( P_{\text{min}} ) (W)</td>
<td>301.64</td>
<td>294.37</td>
<td>328.12</td>
<td>307.27</td>
<td>289.92</td>
<td>292.04</td>
</tr>
<tr>
<td>( P_{\text{avg}} ) (W)</td>
<td>359.72</td>
<td>377.94</td>
<td>385.56</td>
<td>373.13</td>
<td>377.80</td>
<td>373.73</td>
</tr>
<tr>
<td>( P_{\text{wrk}} ) (W)</td>
<td>112.22</td>
<td>130.44</td>
<td>138.06</td>
<td>125.63</td>
<td>130.30</td>
<td>125.23</td>
</tr>
<tr>
<td>( E_{\text{tot}} ) (J)</td>
<td>6,608.60</td>
<td>4,155.61</td>
<td>5,109.44</td>
<td>2,427.28</td>
<td>2,072.07</td>
<td>1,905.70</td>
</tr>
<tr>
<td>( E_{\text{wrk}} ) (J)</td>
<td>2,061.48</td>
<td>1,433.54</td>
<td>1,829.30</td>
<td>816.60</td>
<td>714.04</td>
<td>643.65</td>
</tr>
</tbody>
</table>

- **LU factorization**
  - Due to lack of synchronization points MKL leads better performance in terms of execution time over LAPACK
  - SMPSs: longer execution time due to high number of flops to perform LU factorization with incremental pivoting!

- **Cholesky factorization**
  - Superiority for the SMPSs parallelization from performance and energy!
  - SMPSs: Gains in execution time around 7% and improvement of energy savings about 9%

- **Reduction to tridiagonal form**
  - MKL outperforms the execution time of LAPACK due to a narrow block size and parallel version of \( \text{dsymv} \) kernel
Conclusions and future work

Implementations:
- MKL/SMPSs routines produce higher average power than LAPACK but provide a reduced execution time!
- MKL/SMPSs apply “race-to-idle” technique keeping the cores busy the most of the time!
  
  MKL/SMPSs take advantage in energy efficiency!

Performance and power tracing:
- 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

Future work:
- Developing power models for numerical libraries in order to predict energy consumption even without execution the code.
Thanks for your attention!

Questions?