Parallelizing Dense Matrix Factorizations on Clusters of Multicore Processors using SMPSs

R. M. Badia\textsuperscript{1} J. Labarta\textsuperscript{1} V. Marjanovic\textsuperscript{1} A. F. Martín\textsuperscript{2} R. Mayo\textsuperscript{3} E. S. Quintana-Ortí\textsuperscript{3} R. Reyes\textsuperscript{3}

\textsuperscript{1} Barcelona Supercomputing Center (BSC–CNS), Spain
\{rosa.m.badia, jesus.labarta, vladimir.marjanovic\}@bsc.es

\textsuperscript{2} CIMNE, Univ. Politècnica de Catalunya, Spain
amartin@cimne.upc.edu

\textsuperscript{3} DICC, Universitat Jaume I, Spain
\{quintana, rreyes\}@icc.uji.es

ParCo 2011, Ghent
Solution of linear systems

- Systems of linear equations,

\[ Ax = b, \]

are ubiquitous in scientific and engineering apps.

- "Large-scale" linear systems arise, e.g., in
  - Molecular dynamics simulations,
  - Fast acoustic scattering problems,
  - Dielectric polarization of nanostructures,
  - Magneto-hydrodynamics, etc.

In these apps., \( A \) is dense and \( x \) can easily have \( O(100,000) \) entries!

- The LINPACK benchmark (Top500) is a disguised linear system
The most efficient method to solve

\[ Ax = b, \]

when \( A \) is dense, is to “transform” (factorize) the matrix into a simpler form, and then solve the resulting linear systems.

Depending on the structure of the matrix:

- Cholesky factorization for s.p.d. \( A \rightarrow U^T U \)
- QR factorization for nonsquare \( A \rightarrow QR \)
- LU factorization for general \( A \rightarrow P^T LU \)

Computing the factorization requires a cubic number of flops, but solving the transformed linear systems is easy and cheap; e.g.,

1. \( A \rightarrow U^T U \) (Cholesky factorization)
2. \( U^T y = b \)
3. \( Ux = y \)
Motivation

Libraries for clusters

- Message-passing
  - ScaLAPACK: http://www.netlib.org/scalapack/
  - PLAPACK: http://www.cs.utexas.edu/~plapack/

- Clusters of multi-core processors?
  - One MPI process per node + multi-threaded BLAS
  - One MPI process per core

Both are suboptimal!
Motivation

Goal:

- Exploit task-level parallelism of dense matrix factorizations with little intrusion in existing legacy codes
- ScaLAPACK routine for the Cholesky factorization
- Other matrix factorizations/apps. parallelized as part of project text:

  http://www.project-text.eu/
  - Barcelona Supercomputer Center
  - HLRS Stuttgart
  - Jülich Supercomputer Center
  - EPCC
  - FORTH
  - The University of Manchester
  - Universitat Jaume I
  - IBM Research Zurich
  - Université de Pau
Introduction and motivation

1. Introduction and motivation
2. Brief overview of Cholesky factorization
3. Message-passing Cholesky factorization in ScaLAPACK
4. MPI/SMPSs task-ification of ScaLAPACK/Cholesky
5. Performance evaluation
6. Conclusions
Brief overview of Cholesky factorization

Right-Looking Blocked Cholesky Factorization

Definition

Factors $A$ into the product $A = U^T U$, where $A$ is s.p.d. and $U$ upper triangular.

Single R-L step

1. Factorize diagonal block
   $A_{11} \rightarrow U_{11}^T U_{11}$
**Definition**

Factors $A$ into the product $A = U^T U$, where $A$ is s.p.d. and $U$ upper triangular.

**Single R-L step**

1. Factorize diagonal block $A_{11} \rightarrow U_{11}^T U_{11}$
2. Compute panel $U_{12} \leftarrow (U_{11}^T)^{-1} A_{12}$
Brief overview of Cholesky factorization

Right-Looking Blocked Cholesky Factorization

Definition

Factors $A$ into the product $A = U^T U$, where $A$ is s.p.d. and $U$ upper triangular

Single R-L step

1. Factorize diagonal block $A_{11} \rightarrow U_{11}^T U_{11}$
2. Compute panel $U_{12} \leftarrow (U_{11}^T)^{-1} A_{12}$
3. Update trailing submatrix $\tilde{A}_{22} \leftarrow A_{22} - U_{12}^T U_{12}$
Outline

1. Introduction and motivation
2. Brief overview of Cholesky factorization
3. Message-passing Cholesky factorization in ScaLAPACK
4. MPI/SMPSs task-ification of ScaLAPACK/Cholesky
5. Performance evaluation
6. Conclusions
Message-passing Cholesky factorization in ScaLAPACK

Routine \texttt{p_potrf} in ScaLAPACK

2D Block-Cyclic Data Distribution

Global View

Local View

Data Distribution Blocking Factor
Processor 0 performs the Cholesky factorization of the first diagonal block.
Message-passing Cholesky factorization in ScaLAPACK

Routine `p_potrf` in ScaLAPACK

Global View

Cholesky factor broadcast rowwise by processor 0

Local View

Parallelizing Matrix Factorizations on Clusters using SMPSs
Message-passing Cholesky factorization in ScaLAPACK

Routine \texttt{p\_potrf} in ScaLAPACK

First row panel computed by the first row of processes

Global View

Local View

Parallelizing Matrix Factorizations on Clusters using SMPSs
Message-passing Cholesky factorization in ScaLAPACK

Routine $\text{p}\_\text{potrf}$ in ScaLAPACK

First row panel broadcast columnwise by the first row of processes

Global View

Local View

Parallelizing Matrix Factorizations on Clusters using SMPSs

R. M. Badia et al.
Message-passing Cholesky factorization in ScaLAPACK

Routine \texttt{p_potrf} in ScaLAPACK

First row panel transposed by the second row (source) and column (target) of processes

Parallelizing Matrix Factorizations on Clusters using SMPSs

R. M. Badia et al.
Message-passing Cholesky factorization in ScaLAPACK

Routine \texttt{p_potrf} in ScaLAPACK

First row panel transposed by the second row (source) and column (target) of processes

Global View

Local View

Read
Write
Read/Write
Message-passing Cholesky factorization in ScaLAPACK

Routine \texttt{p_potrf} in ScaLAPACK

First row panel transposed by the second row (source) and column (target) of processes.
Message-passing Cholesky factorization in ScaLAPACK

Routine \texttt{p_potrf} in ScaLAPACK

First row panel transposed by the second row (source) and column (target) of processes
Message-passing Cholesky factorization in ScaLAPACK

Routine \texttt{p_potrf} in ScaLAPACK

Transpose of the first row panel broadcast rowwise by the second column of processes
Message-passing Cholesky factorization in ScaLAPACK

Routine `p_potrf` in ScaLAPACK

Processes update their local portion of the leading diagonal block

Data Distribution Blocking Factor

Global View

Local View

Parallelizing Matrix Factorizations on Clusters using SMPSs
Message-passing Cholesky factorization in ScaLAPACK

Routine \texttt{p_potrf} in ScaLAPACK

Processes update their local portion of the off-diagonal block

Parallelizing Matrix Factorizations on Clusters using SMPSs
Message-passing Cholesky factorization in ScaLAPACK

Routine \( p_{\text{potrf}} \) in ScaLAPACK

Processes update their local portion of the trailing diagonal block.
Outline

1. Introduction and motivation
2. Brief overview of Cholesky factorization
3. Message-passing Cholesky factorization in ScaLAPACK
4. MPI/SMPSs task-ification of ScaLAPACK/Cholesky
5. Performance evaluation
6. Conclusions
The parallel/distributed symmetric rank-k update uses a larger algorithmic blocking factor to increase level-3 BLAS granularity.

Algorithmic and data distribution partitionings don’t need to be aligned → Blocking factors chosen independently (library vs. user).

If level-3 BLAS kernels are naively encapsulated into SMPSs tasks ... → Accessed data can partially overlap the accesses of others tasks → Base-address dependency test cannot handle this scenario as is.

Solution: align algorithmic/data distribution partitionings.
Taskification of computational kernels

- Coarse-grain level-3 BLAS update off-diagonal blocks, load unbalance → Need for finer-grain parallelism via blocking techniques
- Desirable that granularity of tasks could be determined independently of the distribution blocking factor, but . . .
- . . . transposition operations of a row panel (i.e., pack, unpack) are blocked conformally/aligned with data distribution partitioning
- Solution: granularity determined by the distribution blocking factor
Taskification of communication kernels

-Recv. calls must be blocked conformally with computational kernels
- Avoid decomposing a Recv. call into a set of receives → Preserve latency/bandwidth requirements of the original application
- We instead decompose a Recv. call into:
  - Recv. Task: actually receives the message
  - Set of artificial/void/ghost? tasks: do nothing, used to create the data dependency path among communication and computation kernels
Taskification of communication kernels

- Send calls must be blocked conformally with computational kernels
- Avoid decomposing a Send call into a set of sends → Preserve latency/bandwidth requirements of the original application
- We instead decompose a Send call into:
  - Send Task: actually sends the message
  - Set of artificial/void/ghost? tasks: do nothing, used to create the data dependency path among communication and computation kernels
Outline

1. Introduction and motivation
2. Brief overview of Cholesky factorization
3. Message-passing Cholesky factorization in ScaLAPACK
4. MPI/SMPSs task-ification of ScaLAPACK/Cholesky
5. Performance evaluation
6. Conclusions
Target platform:
- peco.act.uji.es, small cluster at Universitat Jaume I
- 8 nodes, 2 Intel QuadCore E5520 per node (64 cores total)
- Infiniband interconnect

Compilers and libraries:
- Intel C and Fortran77 compilers v11.1
- MPI/SMPSs rev13/svn/TRAC-TEXT
- ScaLAPACK v1.8.0 + BLACS v1.1 (OpenMPI v1.4)
- MKL 10.3 (single and multi-threaded BLAS)
We used **Extrae** and **Paraver** to extract execution traces.

Analysis of critical path and idle times allowed us to detect performance bottlenecks.

BSC used trace information to add scheduling options to SMPSs runtime.
Performance evaluation

Default scheduling options

Excessive idle time
Performance evaluation

Local thread queues disabled
Performance evaluation

Impact of memory scheduling options (2x2 mesh)

Default behavior
disable.local_queues=1

Matrix size (n)

GFLOPS
Performance evaluation

Impact of memory scheduling options (2x4 mesh)

- Default behavior
- disable.local_queues=1

Matrix size (n) vs. GFLOPS
Performance evaluation on 32 cores

Performance of parallel solutions on 4 nodes (32 cores)

Matrix size (n) vs. GFLOPS (% of peak performance)

- Pure MPI, 4x8
- MPI + MT-BLAS, 2x2, 4 nodes, 1 proc/node, 7 th/proc.
- MPI+SMPSs, 2x2
- Comp. thr=7, disable local queues=0, COMM_THREAD=0
- MPI+SMPSs, 2x2
- Comp. thr=7, disable local queues=1, COMM_THREAD=0
- MPI+SMPSs, 2x2
- Comp. thr=8, disable local queues=1, COMM_THREAD=1
Performance evaluation

Performance evaluation on 64 cores

Performance of parallel solutions on 8 nodes (64 cores)

- Red circles: Pure MPI, 4x16
- Green crosses: MPI+MT-BLAS, 2x4, Comp. thr=8
- Blue stars: MPI+SMPSs, 2x4, Comp. thr=7, disable local queues=0, COMM_THREAD=0
- Purple pentagrams: MPI+SMPSs, 2x4, Comp. thr=7, disable local queues=1, COMM_THREAD=0
- Yellow squares: MPI+SMPSs, 2x4, Comp. thr=6, disable local queues=1, COMM_THREAD=1
Experimental framework (large scale)

Target platform:
- JuRoPa at Juelich Supercomputing Center
- 2,028 nodes, 2 Intel Xeon X5570 per node
- Infiniband interconnect

Compilers and libraries:
- Intel C and Fortran77 compilers v11.1
- MPI/SMPSs Minor changes from svn/TRAC-TEXT version
- ScaLAPACK v1.8.0 + BLACS v1.1
- MKL 10.3 (single and multi-threaded BLAS)
Performance evaluation

Preliminary large scale report

Parallelizing Matrix Factorizations on Clusters using SMPSs

Performance on JuRoPa

% Of peak performance

Number of cores

Reference
BLAS-MT
MPI/SMPSs
Conclusions

- Current status:
  - ROutine for the Cholesky factorization in ScaLAPACK adapted to leverage current MPI/SMPSs for clusters
    → Match algorithmic and distribution blocking factors
  - Superior performance compared with conventional parallel solutions for clusters of multi-core processors

- Ongoing tasks:
  - Fine tuning, optimization guided by detailed performance analysis
  - Clusters of hardware accelerators
  - Other matrix kernels
Parallelizing Dense Matrix Factorizations on Clusters of Multicore Processors using SMPSs

R. M. Badia¹  J. Labarta¹  V. Marjanovic¹  A. F. Martín²  
R. Mayo³  E. S. Quintana-Ortí³  R. Reyes³

¹ Barcelona Supercomputing Center (BSC–CNS), Spain  
{rosa.m.badia, jesus.labarta, vladimir.marjanovic}@bsc.es

² CIMNE, Univ. Politècnica de Catalunya, Spain  
amartin@cimne.upc.edu

³ DICC, Universitat Jaume I, Spain  
{quintana, rreyes}@icc.uji.es

ParCo 2011, Ghent