Design and Performance Issues of Cholesky and LU Solvers using UPCBLAS

Jorge González-Domínguez*, Osni A. Marques**,
María J. Martín*, Guillermo L. Taboada*, Juan Touriño*

*Computer Architecture Group, University of A Coruña, Spain
{jgonzalezd,mariam,taboada,juan}@udc.es
**Computational Research Division, Lawrence Berkeley National Laboratory, CA, USA
OAMarques@lbl.gov

10th IEEE International Symposium on Parallel and Distributed Processing with Applications
ISPA 2012

1 Introduction

2 Cholesky Solver

3 LU Solver

4 Experimental Evaluation

5 Conclusions

1 Introduction

2 Cholesky Solver

3 LU Solver

4 Experimental Evaluation

5 Conclusions

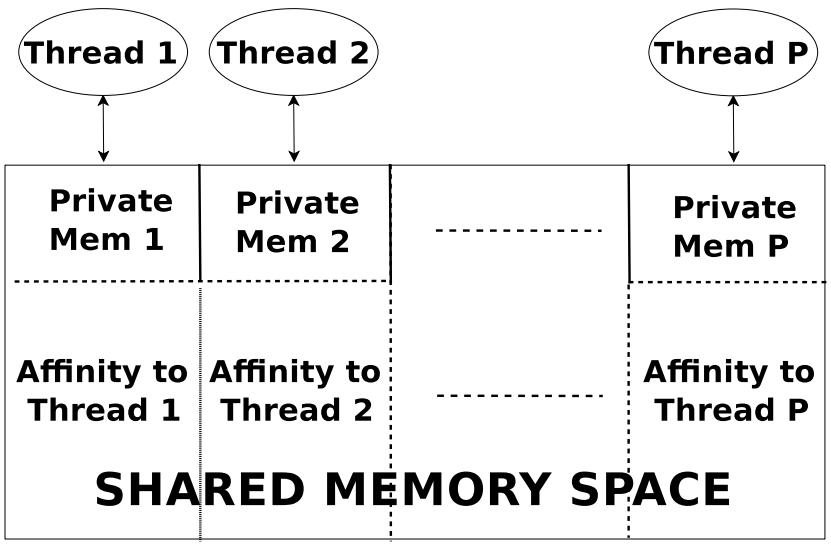
UPC: a Suitable Alternative for HPC in Multi-core Era

Programming Models:

- Traditionally: Shared/Distributed memory programming models
- Challenge: Hybrid memory architectures
  - PGAS (Partitioned Global Address Space)

PGAS Languages:

- UPC (C)
- Titanium (Java)
- Co-Array Fortran (Fortran)



Main advantages of the PGAS model:

- simplifies programming
- allows an efficient use of one-sided communications

UPCBLAS

Characteristics of the Library

- Includes parallel BLAS routines built on top of UPC
- Focused on increasing the programmability
- Distributed matrices and vectors are represented by shared arrays
  - Advantage: Shared arrays are implicitly distributed
  - Drawback: Only 1D distributions allowed
- Good trade-off between programmability and performance
- UPCBLAS parallel functions call internally BLAS routines to perform the sequential computations in each thread

UPCBLAS Matrix Vector Product

```
int upc_blas_sgemv(UPCBLAS_DIMMDIST dimmDist, int
block_size, int sec_block_size, UPCBLAS_TRANSPOSE
transpose, int m, int n, float alpha, shared void
*A, int lda, shared void *x, float beta, shared
void *y);
```

- Syntax similar to sequential BLAS
- Pointers point to shared memory
- Additional parameters to specify the distribution
  - dimmDist: enumerate value to specify the type of distribution (by rows or by columns)
  - The meaning of *block_size* and *sec_block_size* depends on the dimmDist value

UPCBLAS Matrix Vector Product

```
int upc_blas_sgemv(UPCBLAS_DIMMDIST dimmDist, int
block_size, int sec_block_size, UPCBLAS_TRANSPOSE
transpose, int m, int n, float alpha, shared void
*A, int lda, shared void *x, float beta, shared
void *y);
```

- Syntax similar to sequential BLAS
- Pointers point to shared memory
- Additional parameters to specify the distribution
  - dimmDist: enumerate value to specify the type of distribution (by rows or by columns)
  - The meaning of *block_size* and *sec_block_size* depends on the dimmDist value

UPCBLAS Matrix Vector Product

```
int upc_blas_sgemv(UPCBLAS_DIMMDIST dimmDist, int
block_size, int sec_block_size, UPCBLAS_TRANSPOSE
transpose, int m, int n, float alpha, shared void
*A, int lda, shared void *x, float beta, shared
void *y);
```

- Syntax similar to sequential BLAS
- Pointers point to shared memory
- Additional parameters to specify the distribution
  - dimmDist: enumerate value to specify the type of distribution (by rows or by columns)
  - The meaning of *block_size* and *sec_block_size* depends on the dimmDist value

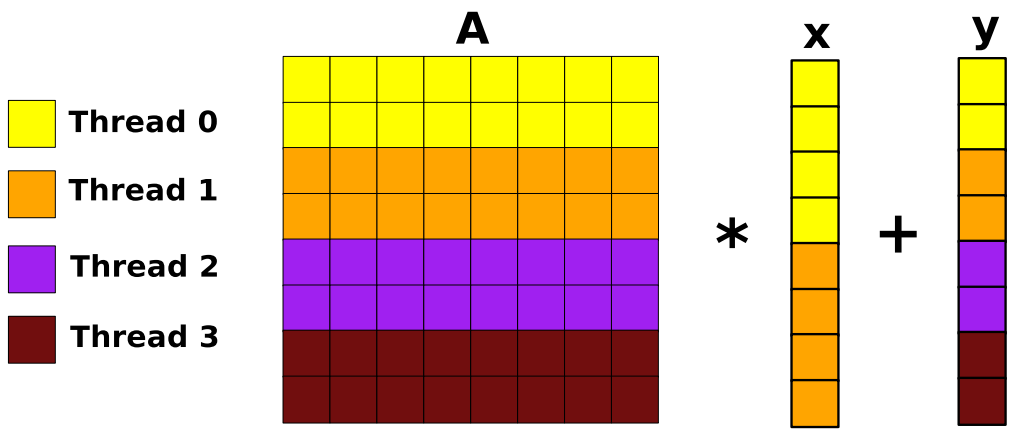
UPCBLAS Matrix Vector Product

```
int upc_blas_sgemv(UPCBLAS_DIMMDIST dimmDist, int
block_size, int sec_block_size, UPCBLAS_TRANSPOSE
transpose, int m, int n, float alpha, shared void
*A, int lda, shared void *x, float beta, shared
void *y);
```

- Syntax similar to sequential BLAS
- Pointers point to shared memory
- Additional parameters to specify the distribution
  - dimmDist: enumerate value to specify the type of distribution (by rows or by columns)
  - The meaning of *block_size* and *sec_block_size* depends on the dimmDist value

UPCBLAS Matrix Vector Product

shared [16] float *A*[64]; shared [4] float *x*[8]; shared [2] float *y*[8]




```
upc_blas_sgemv(upcblas_rowDist, 2, 4,
upcblas_noTrans, 8, 8, alpha, (shared void *)A, 8,
(shared void *)x, beta, (shared void *)y);
```

UPCBLAS More Information

Described in:

J. González-Domíguez, M. J. Martín, G. L. Taboada, J. Touriño, R. Doallo, D. A. Mallón and B. Wibecan, “UPCBLAS: A Library for Parallel Matrix Computations in Unified Parallel C”, Concurrency and Computation: Practice and Experience, 2012 (In Press), available at
`http://dx.doi.org/10.1002/cpe.1914`

Description of the Problem

Solution of Systems of Equations using UPCBLAS

- $A * X = B$ being:
  - A a mxm matrix
  - X and B mxn matrices (X overwrites B)
- First step: Factorization
  - Cholesky: $A = L * L^T$
  - LU: $A = L * U$
- Second step: Two triangular solvers
  - Cholesky : $L * Y = B$ and $L^T * X = Y$
  - LU: $L * Y = B$ and $U * X = Y$

1 Introduction

2 Cholesky Solver

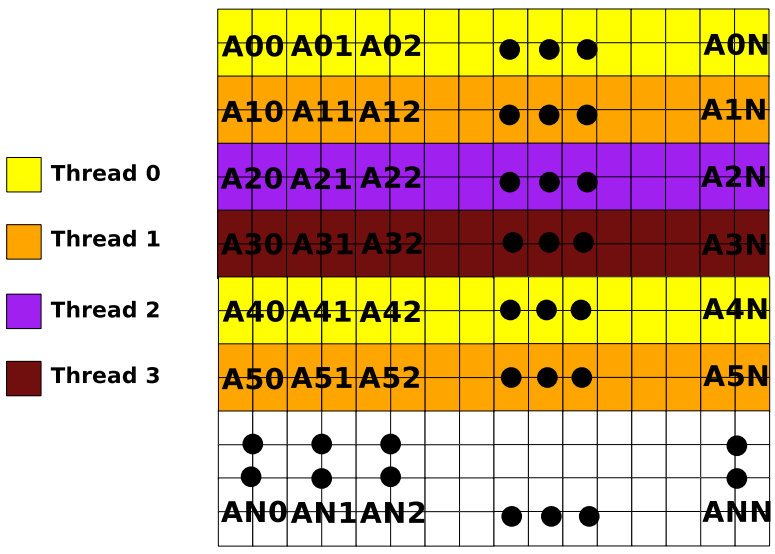
3 LU Solver

4 Experimental Evaluation

5 Conclusions

Cholesky Factorization

- Two different block algorithms were implemented
- Both are based on BLAS3 routines
  - Algorithm based on gemm (LAPACK)
  - Algorithm based on syrk (ScaLAPACK)
- Only 1D distributions available: block-cyclic distribution by rows or by columns



Block-cyclic distribution by rows
A_{ij} are submatrices

Cholesky Solver based on gemm

for *i=0;i<NB;i=i+1* **do**
  if *MYTHREAD has affinity to block i* **then**
    $A_{i,i} = A_{i,i} - A_{i,0..i-1} * A^T_{i,0..i-1} \rightarrow syrk$
    Sequential Cholesky Factorization of $A_{i,i}$
  end
  $A_{i+1..N,i} = A_{i+1..N,i} - A_{i+1..N,0..i-1} * A^T_{i,0..i-1} \rightarrow gemm$
  Solve $Z * A^T_{i,i} = A_{i+1..N,i} \rightarrow trsm$
  $A_{i+1..N,i} = Z$
end
Solve $Y * A^T = B \rightarrow trsm$
Solve $X * A = Y \rightarrow trsm$

Cholesky Solver based on gemm

for *i=0;i<NB;i=i+1* **do**

  if *MYTHREAD has affinity to block i* **then**

    $A_{i,i} = A_{i,i} - A_{i,0..i-1} * A^T_{i,0..i-1} \rightarrow syrk$

    Sequential Cholesky Factorization of $A_{i,i}$

  end

  $A_{i+1..N,i} = A_{i+1..N,i} - A_{i+1..N,0..i-1} * A^T_{i,0..i-1} \rightarrow gemm$

  $Solve\ Z * A^T_{i,i} = A_{i+1..N,i} \rightarrow trsm$

  $A_{i+1..N,i} = Z$

end

$Solve\ Y * A^T = B \rightarrow trsm$

$Solve\ X * A = Y \rightarrow trsm$

Cholesky Solver based on gemm

for *i=0;i<NB;i=i+1* **do**
  if *MYTHREAD has affinity to block i* **then**
    $A_{i,i} = A_{i,i} - A_{i,0..i-1} * A^T_{i,0..i-1} \rightarrow syrk$
    Sequential Cholesky Factorization of $A_{i,i}$
  end
  $A_{i+1..N,i} = A_{i+1..N,i} - A_{i+1..N,0..i-1} * A^T_{i,0..i-1} \rightarrow gemm$
  $Solve\ Z * A^T_{i,i} = A_{i+1..N,i} \rightarrow trsm$
  $A_{i+1..N,i} = Z$
end
$Solve\ Y * A^T = B \rightarrow trsm$
$Solve\ X * A = Y \rightarrow trsm$

Cholesky Solver based on gemm

for *i=0;i<NB;i=i+1* **do**

  if *MYTHREAD has affinity to block i* **then**

    $A_{i,i} = A_{i,i} - A_{i,0..i-1} * A^T_{i,0..i-1} \rightarrow syrk$

    Sequential Cholesky Factorization of $A_{i,i}$

  end

  $A_{i+1..N,i} = A_{i+1..N,i} - A_{i+1..N,0..i-1} * A^T_{i,0..i-1} \rightarrow gemm$

  $Solve\ Z * A^T_{i,i} = A_{i+1..N,i} \rightarrow trsm$

  $A_{i+1..N,i} = Z$

end

$Solve\ Y * A^T = B \rightarrow trsm$

$Solve\ X * A = Y \rightarrow trsm$

Cholesky Solver based on gemm

for *i=0;i<NB;i=i+1* **do**
  if *MYTHREAD has affinity to block i* **then**
    $A_{i,i} = A_{i,i} - A_{i,0..i-1} * A^T_{i,0..i-1} \rightarrow syrk$
    Sequential Cholesky Factorization of $A_{i,i}$
  end
  $A_{i+1..N,i} = A_{i+1..N,i} - A_{i+1..N,0..i-1} * A^T_{i,0..i-1} \rightarrow gemm$
  Solve $Z * A^T_{i,i} = A_{i+1..N,i} \rightarrow trsm$
  $A_{i+1..N,i} = Z$
end
Solve $Y * A^T = B \rightarrow trsm$
Solve $X * A = Y \rightarrow trsm$

Cholesky Solver based on syrk

for *i=0;i<NB;i=i+1* **do**
  if *MYTHREAD has affinity to block i* **then**
    Sequential Cholesky Factorization of $A_{i,i}$
  end
  Solve $Z * A_{i,i}^T = A_{i+1..N,i} \rightarrow trsm$
  $A_{i+1..N,i} = Z$
  $A_{i+1..N,i+1..N} = A_{i+1..N,i+1..N} - A_{i+1..N,i} * A_{i+1..N,i}^T \rightarrow syrk$
end
Solve $Y * A^T = B \rightarrow trsm$
Solve $X * A = Y \rightarrow trsm$

Cholesky Solver based on syrk

for $i=0;i<NB;i=i+1$ **do**
  if *MYTHREAD has affinity to block i* **then**
    Sequential Cholesky Factorization of $A_{i,i}$
  end
  $Solve\ Z * A_{i,i}^T = A_{i+1..N,i} \rightarrow trsm$
  $A_{i+1..N,i} = Z$
  $A_{i+1..N,i+1..N} = A_{i+1..N,i+1..N} - A_{i+1..N,i} * A_{i+1..N,i}^T \rightarrow syrk$
end
$Solve\ Y * A^T = B \rightarrow trsm$
$Solve\ X * A = Y \rightarrow trsm$

Cholesky Solver based on syrk

for *i=0;i<NB;i=i+1* **do**
  if *MYTHREAD has affinity to block i* **then**
    Sequential Cholesky Factorization of $A_{i,i}$
  end
  Solve $Z * A_{i,i}^T = A_{i+1..N,i} \rightarrow trsm$
  $A_{i+1..N,i} = Z$
  $A_{i+1..N,i+1..N} = A_{i+1..N,i+1..N} - A_{i+1..N,i} * A_{i+1..N,i}^T \rightarrow syrk$
end
Solve $Y * A^T = B \rightarrow trsm$
Solve $X * A = Y \rightarrow trsm$

Algorithm based on syrk vs based on gemm

- Advantage: Less computations performed only by one thread
- Drawback: Stronger data dependencies -> Increase of the overhead due to synchronizations

Cholesky Solver

Optimization Techniques

- Appropriate choice of NB (synchronization-parallelism trade-off)
- Numerical computations within each UPC thread are parallelized to exploit NUMA systems:
  - Calls to multithreaded BLAS routines
  - Implementation of sequential Cholesky factorization with OpenMP support

1 Introduction

2 Cholesky Solver

3 LU Solver

4 Experimental Evaluation

5 Conclusions

Basic LU Solver

for *i=0;i<NB;i=i+1* **do**
  if *MYTHREAD has affinity to block i* **then**
    Sequential LU Factorization of $A_{i,i..N}$
  end
  Solve $Z * A_{i,i}^T = A_{i+1..N,i} \rightarrow trsm$
  $A_{i+1..N,i} = Z$
  $A_{i+1..N,i+1..N} = A_{i+1..N,i+1..N} - A_{i+1..N,i} * A_{i,i+1..N} \rightarrow gemm$
end
Solve $A * Y = B \rightarrow trsm$
Solve $A * X = Y \rightarrow trsm$

Basic LU Solver

for *i=0;i<NB;i=i+1* **do**
  if *MYTHREAD has affinity to block i* **then**
    Sequential LU Factorization of $A_{i,i..N}$
  end
  Solve $Z * A^T_{i,i} = A_{i+1..N,i} \rightarrow trsm$
  $A_{i+1..N,i} = Z$
  $A_{i+1..N,i+1..N} = A_{i+1..N,i+1..N} - A_{i+1..N,i} * A_{i,i+1..N} \rightarrow gemm$
end
Solve $A * Y = B \rightarrow trsm$
Solve $A * X = Y \rightarrow trsm$

Basic LU Solver

for $i=0;i<NB;i=i+1$ **do**
  if *MYTHREAD has affinity to block i* **then**
    Sequential LU Factorization of $A_{i,i..N}$
  end
  Solve $Z * A_{i,i}^T = A_{i+1..N,i} \rightarrow trsm$
  $A_{i+1..N,i} = Z$
  $A_{i+1..N,i+1..N} = A_{i+1..N,i+1..N} - A_{i+1..N,i} * A_{i,i+1..N} \rightarrow gemm$
end
Solve $A * Y = B \rightarrow trsm$
Solve $A * X = Y \rightarrow trsm$

Basic LU Solver

for $i=0;i<NB;i=i+1$ **do**
  if *MYTHREAD has affinity to block i* **then**
    Sequential LU Factorization of $A_{i,i..N}$
  end
  Solve $Z * A_{i,i}^T = A_{i+1..N,i} \rightarrow trsm$
  $A_{i+1..N,i} = Z$
  $A_{i+1..N,i+1..N} = A_{i+1..N,i+1..N} - A_{i+1..N,i} * A_{i,i+1..N} \rightarrow gemm$
end
Solve $A * Y = B \rightarrow trsm$
Solve $A * X = Y \rightarrow trsm$

Basic LU Solver

for *i=0;i<NB;i=i+1* **do**
  if *MYTHREAD has affinity to block i* **then**
    Sequential LU Factorization of $A_{i,i..N}$
  end
  Solve $Z * A_{i,i}^T = A_{i+1..N,i} \rightarrow trsm$
  $A_{i+1..N,i} = Z$
  $A_{i+1..N,i+1..N} = A_{i+1..N,i+1..N} - A_{i+1..N,i} * A_{i,i+1..N} \rightarrow gemm$
end
Solve $A * Y = B \rightarrow trsm$
Solve $A * X = Y \rightarrow trsm$

LU Solver with Partial Pivoting

Partial Pivoting

- To avoid inconsistencies because of dividing by 0
- If the matrix is distributed by rows the pivoting is performed by columns -> it can be parallelized
- A vector of size N (P) is used to store the information about the pivoting

LU Solver with Partial Pivoting

for *i=0;i<NB;i=i+1* **do**
  if *MYTHREAD has affinity to block i* **then**
    P_i = *Partial Pivoting of* $A_{i,i..N}$
    Swap $A_{i,i..N}$ *according to* P_i
    Sequential LU Factorization of $A_{i,i..N}$
  end
  Swap $A_{0..N,i..N}$ *according to* P_i
  Solve $Z * A_{i,i}^T = A_{i+1..N,i} \rightarrow trsm$
  $A_{i+1..N,i} = Z$
  $A_{i+1..N,i+1..N} = A_{i+1..N,i+1..N} - A_{i+1..N,i} * A_{i,i+1..N} \rightarrow gemm$
end
Swap B according to P
Solve $A * Y = B \rightarrow trsm$
Solve $A * X = Y \rightarrow trsm$

LU Solver with Partial Pivoting

for $i=0;i<NB;i=i+1$ **do**
  if *MYTHREAD has affinity to block i* **then**
    P_i = *Partial Pivoting of* $A_{i,i..N}$
    Swap $A_{i,i..N}$ *according to* P_i
    Sequential LU Factorization of $A_{i,i..N}$
  end
  Swap $A_{0..N,i..N}$ *according to* P_i
  Solve $Z * A_{i,i}^T = A_{i+1..N,i} \rightarrow trsm$
  $A_{i+1..N,i} = Z$
  $A_{i+1..N,i+1..N} = A_{i+1..N,i+1..N} - A_{i+1..N,i} * A_{i,i+1..N} \rightarrow gemm$
end
Swap B according to P
Solve $A * Y = B \rightarrow trsm$
Solve $A * X = Y \rightarrow trsm$

LU Solver with Partial Pivoting

for *i=0;i<NB;i=i+1* **do**
  if *MYTHREAD has affinity to block i* **then**
    $P_i = Partial\ Pivoting\ of\ A_{i,i..N}$
    $Swap\ A_{i,i..N}\ according\ to\ P_i$
    $Sequential\ LU\ Factorization\ of\ A_{i,i..N}$
  end
  $Swap\ A_{0..N,i..N}\ according\ to\ P_i$
  $Solve\ Z * A_{i,i}^T = A_{i+1..N,i} \rightarrow trsm$
  $A_{i+1..N,i} = Z$
  $A_{i+1..N,i+1..N} = A_{i+1..N,i+1..N} - A_{i+1..N,i} * A_{i,i+1..N} \rightarrow gemm$
end
$Swap\ B\ according\ to\ P$
$Solve\ A * Y = B \rightarrow trsm$
$Solve\ A * X = Y \rightarrow trsm$

LU Solver

Optimization Techniques

- Appropriate choice of the block size
- Numerical computations within each UPC thread are parallelized to exploit NUMA systems:
  - Calls to multithreaded BLAS routines
  - Implementation of sequential LU factorization with OpenMP support

1 Introduction

2 Cholesky Solver

3 LU Solver

4 Experimental Evaluation

5 Conclusions

Characteristics of the Testbed

Carver Supercomputer

- Installed at National Energy Research Supercomputing Center (NERSC) of the Lawrence Berkeley National Laboratory
- 320 nodes with 2 quad-core Intel Xeon 5550X
- 8 cores and 24GB per node
- 4X QDR InfiniBand network (32 Gbps of theoretical effective bandwidth)

Characteristics of the Testbed

Software

- Berkeley UPC 2.12.2 compiler (internode communications through GASNet over InfiniBand)
- Multithreaded Intel Math Kernel Library (MKL) 10.2.2 (8 internal threads per node)
- ScaLAPACK 1.8.0 library

Characteristics of the Experiments

Type of Experiments

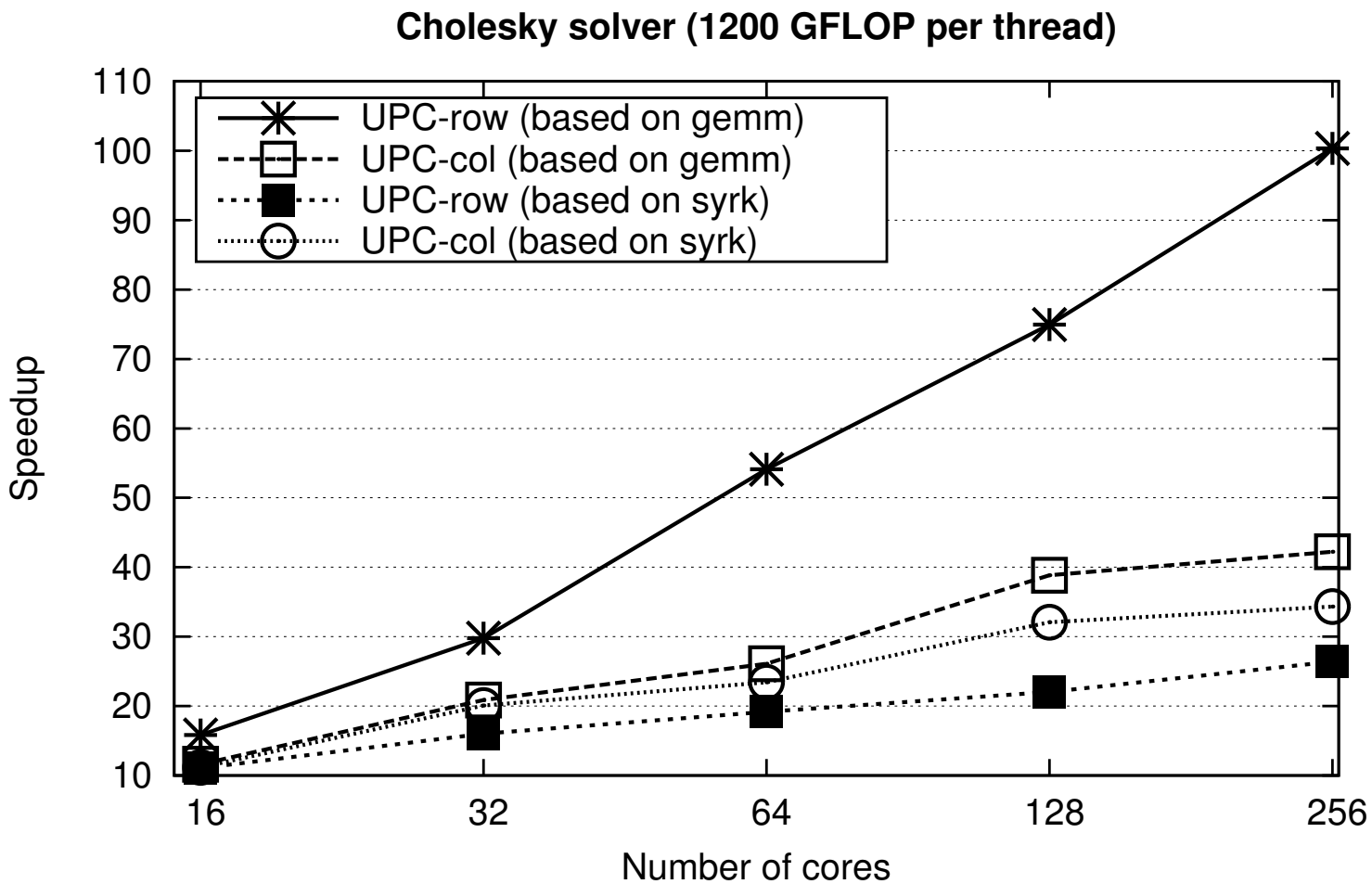
- Weak scaling
- Double precision data
- Best block size
- 1 UPC thread per node and 8 OpenMP threads per UPC thread (one per core)

Characteristics of the Experiments

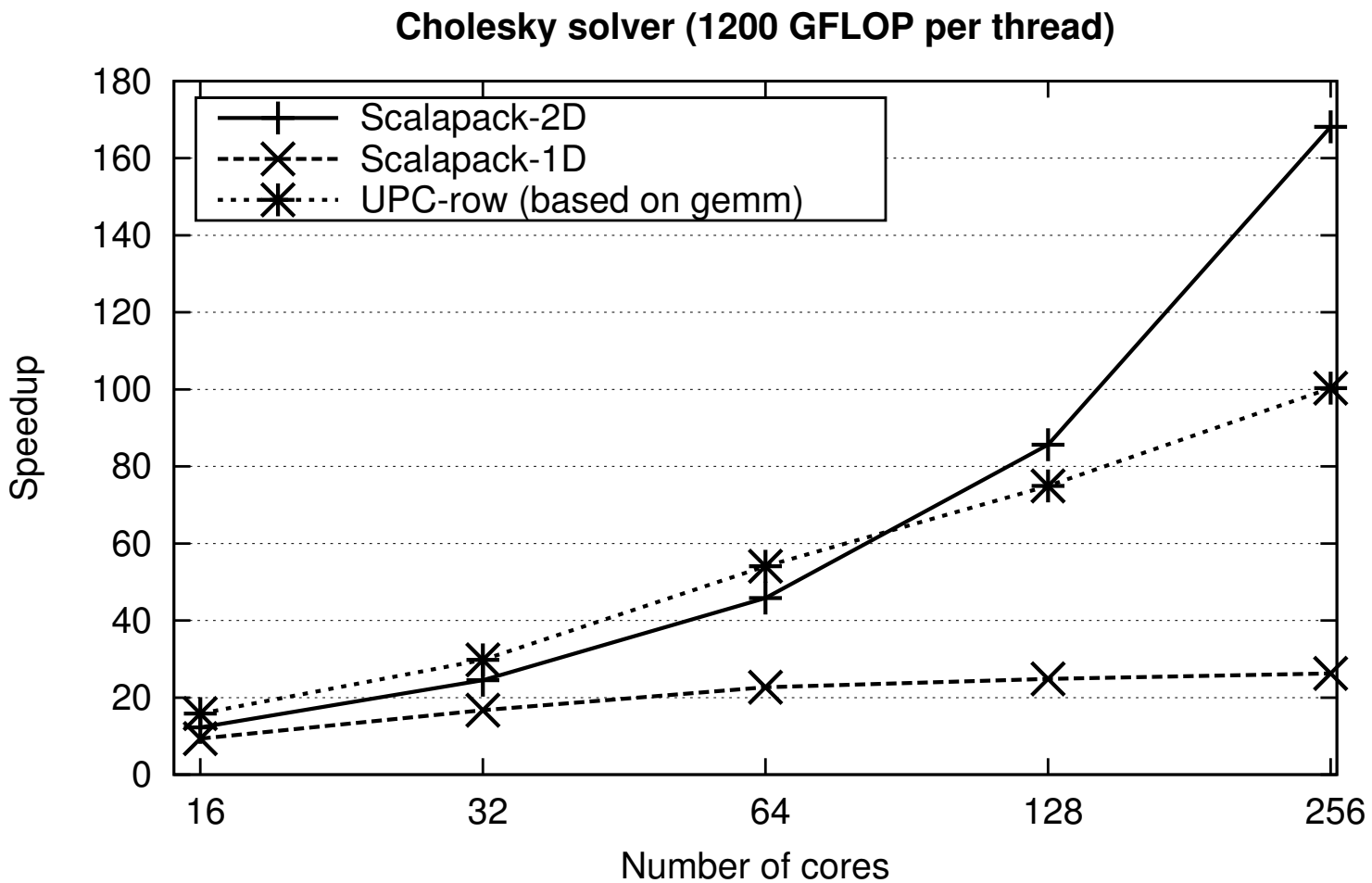
Measures

- Speedups relative to the ScaLAPACK times
- Comparisons with ScaLAPACK
  - ScaLAPACK-2D: results using ScaLAPACK and a 2D distribution
  - ScaLAPACK-1D: results using ScaLAPACK and a 1D distribution

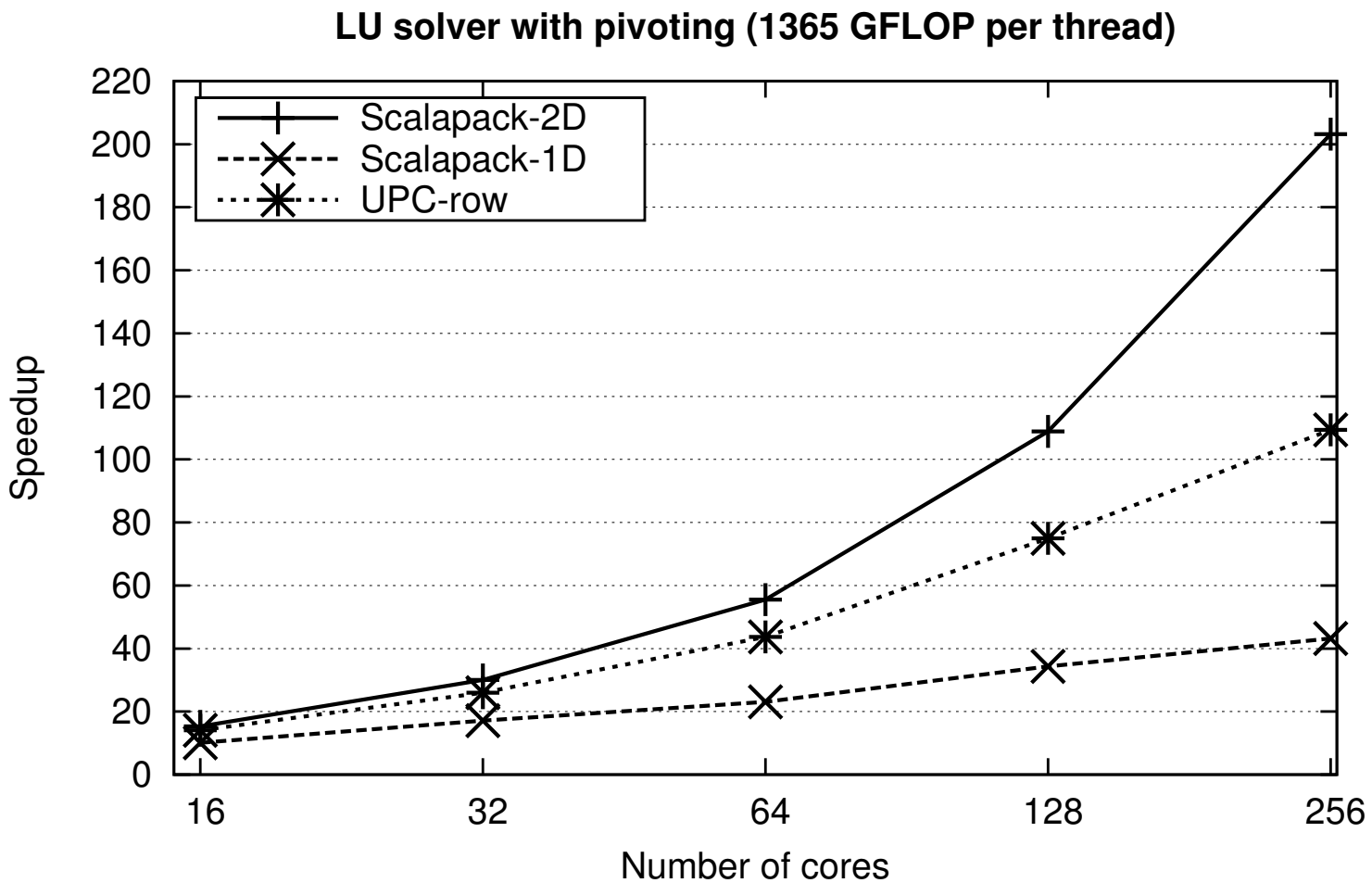
Experimental Evaluation of Cholesky



Experimental Evaluation of Cholesky



Experimental Evaluation of LU with Partial Pivoting



1 Introduction

2 Cholesky Solver

3 LU Solver

4 Experimental Evaluation

5 Conclusions

Summary

- UPC implementations of Cholesky and LU solvers using UPCBLAS
- **UPCBLAS main advantage:** Easier to use than MPI-based libraries
- **UPCBLAS main drawback:** Performance limited as shared arrays can only work with 1D distributions

Conclusions

- UPCBLAS is a good trade-off between programmability and performance
- UPCBLAS is a good alternative for increasing productivity

Design and Performance Issues of Cholesky and LU Solvers using UPCBLAS

Jorge González-Domínguez*, Osni A. Marques**,
María J. Martín*, Guillermo L. Taboada*, Juan Touriño*

*Computer Architecture Group, University of A Coruña, Spain
{jgonzalezd,mariam,taboada,juan}@udc.es
**Computational Research Division, Lawrence Berkeley National Laboratory, CA, USA
OAMarques@lbl.gov

10th IEEE International Symposium on Parallel and Distributed Processing with Applications
ISPA 2012