INRIA

Communication-avoiding LU and QR factorizations for multicore architectures

DONFACK Simplice

INRIA Saclay

Joint work with

Laura Grigori
INRIA Saclay

Alok Kumar Gupta
BCCS,Norway-5075

16th April 2010

INRIA

1. Introduction
2. CALU and CAQR factorization
3. Multithreaded CALU and CAQR
4. Experimental section
5. Conclusion

Introduction

- Architectural trends show an increasing communication cost compared to the time it takes to perform arithmetic operations
  - Motivated the design of communication avoiding algorithms that minimize communication
  - First results are CAQR [Demmel, Grigori, Hoemmen, Langou '08] and CALU [Grigori, Demmel, Xiang '08], implemented for distributed memory.
- Our goal is to design multithreaded QR and LU factorizations for multicores based on communication avoiding algorithms.

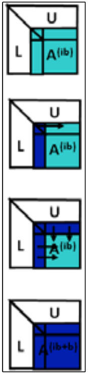
LU factorization with partial pivoting

Factorization on Pr by Pc grid of processors as implemented in SCALAPACK:

For ib = 1 to n-1 step b

A(ib) = A(ib:n, ib:n)

1. Compute panel factorization (pdgetf2) $O(nlog_2 P_r)$
   - find pivot in each column, swap rows
2. Apply all row permutations (pdlaswp) $O(n/b(log_2 P_c + log_2 P_r))$
   - broadcast pivot information along the rows
   - swap rows at left and right
3. Compute block row of U (pdtrsm) $O(n/blog_2 P_c)$
   - broadcast right diagonal block of L of current panel
4. Update trailing matrix (pdgemm) $O(n/b(log_2 P_c + log_2 P_r))$
   - broadcast right block column of L
   - broadcast down block row of U



Pivoting requires communication among processors on distributed memory and synchronisation between threads on multicores.

CALU and CAQR approach

Communication avoiding algorithms [Demmel, Grigori, Hoemmen, Langou, Xiang '08] approach:

- Decrease communication required for pivoting and overcome the latency bottleneck of classic algorithms by
  - performing the factorization of a block column (a tall and skinny matrix) as a reduction operation
  - and doing some redundant computations
- They are communication optimal in terms of both latency and bandwidth
- They lead to important speedups on distributed memory computers

Goal

Our goal

Combine the main ideas to reduce communication in CALU and CAQR with :

- appropriate blocking
- task identification
- dynamic scheduling

The reduction operation to use for a block-column factorization is based on a binary tree with asynchronous tasks :

- reduces synchronisation between threads (only $O(log_2(Pr))$)
- avoids bus contention

INRIA

1 Introduction

2 CALU and CAQR factorization

3 Multithreaded CALU and CAQR

4 Experimental section

5 Conclusion

CAQR

- Each panel factorization is computed as a reduction operation where at each node a QR factorization is performed.
- The reduction tree is chosen depending on the underlying architecture.
- For a binary tree $log_2(Pr)$ steps are used.

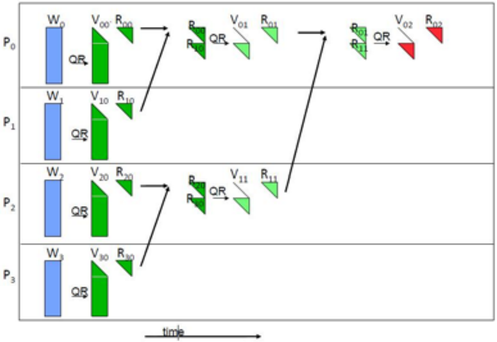


Figure: Parallel TSQR

CAQR

- Update the submatrix using the tree in $log_2(Pr)$ steps

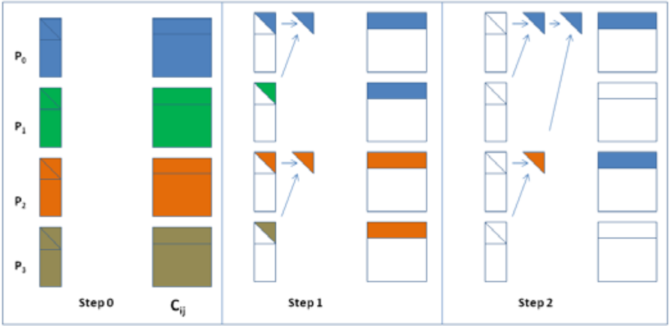


Figure: The update of the trailing submatrix is triggered by the reduction tree used during panel factorization

CALU[Grigori, Demmel, Xiang '08]

The panel factorization is performed in two steps:

- A preprocessing steps aims at identifying at low communication cost good pivot rows
- The pivot rows are permuted in the first positions of the panel and LU without pivoting of the panel is performed

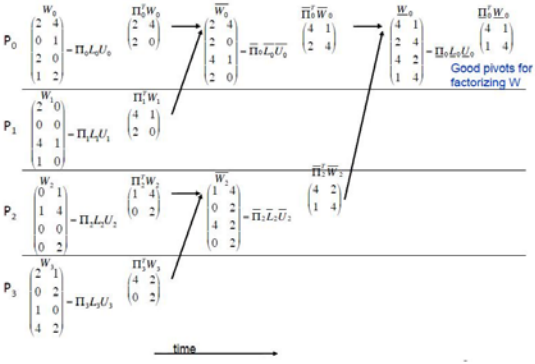


Figure: Stable parallel panel factorization

CALU (Stability)

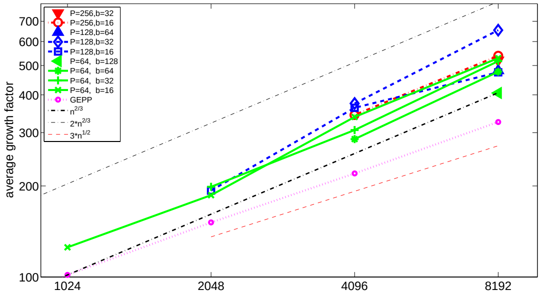


Figure: Stability of binary tree based CALU factorization for random matrices

- Extensive tests performed on random matrices and a set of special matrices using binary tree and flat tree show that CALU is as stable as GEPP in practice.

INRIA

1 Introduction

2 CALU and CAQR factorization

3 Multithreaded CALU and CAQR

4 Experimental section

5 Conclusion

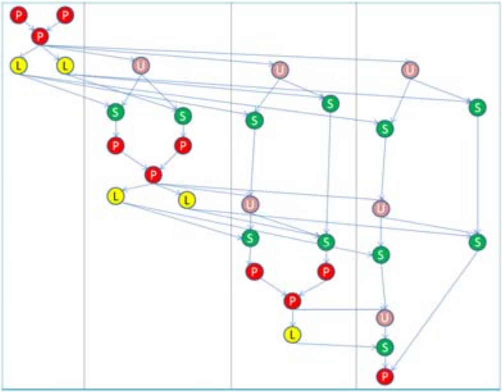
Multithreaded CALU

- The matrix is partitioned in blocks of size Tr x b
- The computation of each block is associated with a task
- The task dependency graph is scheduled using a dynamic scheduler

Figure: Matrix 4×4 blocks and $T_r = 2$ and Corresponding task dependency graph

Multithreaded CALU

Panel factorization is performed in two steps: find good pivots at low communication cost, permute them and compute LU factorization of the panel without pivoting.



The panel factorization stays on the critical path but it is done more efficiently

Multithreaded CALU (Execution)

Figure: Example of execution of CALU for a $10^5 \times 1000$ tall skinny matrix, using $b = 100$ and $T_r = 1$, on 8-core

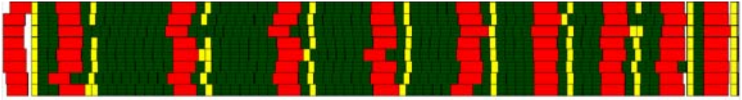


Figure: Example of execution of CALU for a $10^5 \times 1000$ tall skinny matrix, using $b = 100$ and $T_r = 8$, on 8-core

Multithreaded CAQR

Same approach as CALU but:

- Panel factorization is performed only once
- The update of the trailing matrix is triggered by the binary tree used for the panel factorization.

Environments

- Tests performed on: two-socket, quad-core machine based on Intel Xeon EMT64 processor running on Linux and on a four-socket, quad-core machine based on AMD Opteron processor
- Comparison with MKL-10.0.4.23 and PLASMA 2.0 (with default parameters)
- $b = MIN(n, 100)$ has been chosen as block size

Performance of CALU

Performance of CALU, MKL_dgetrf, PLASMA_dgetrf on 8 cores

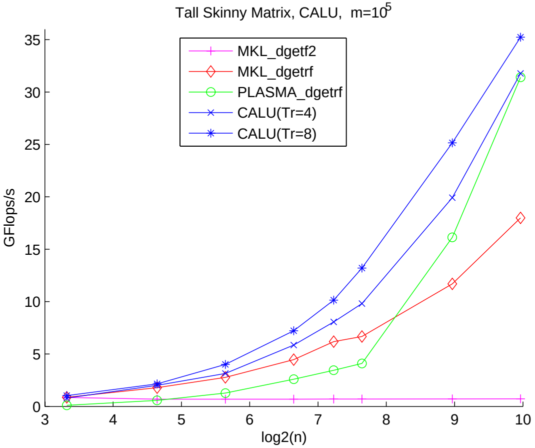


Figure: m=10^5 and varying n from 10 to 1000.

Performance of CALU

Performance of CALU, MKL_dgetrf, PLASMA_dgetrf on 16 cores

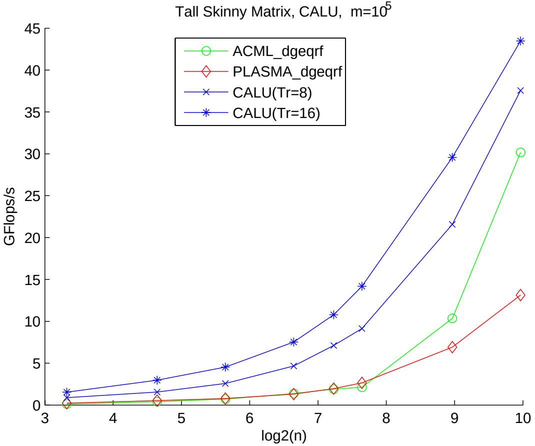


Figure: m=10^5 and varying n from 10 to 1000.

Performance of CAQR, MKL_dgeqrf, PLASMA_dgeqrf on 8 cores

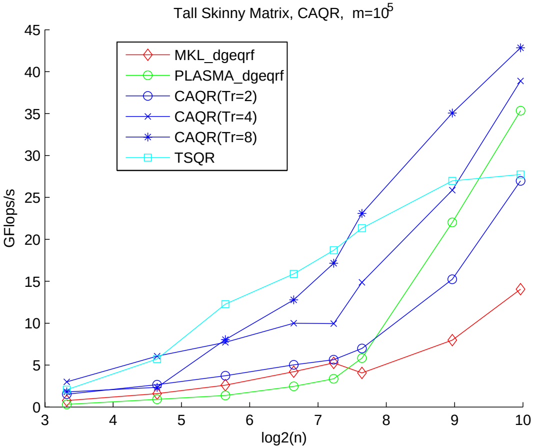


Figure: m=10^5 and varying n from 10 to 1000.

Conclusion

- Multithreaded CALU and CAQR lead to important improvements for tall and skinny matrices with respect to the corresponding routines in MKL and PLASMA.
- PLASMA becomes more efficient with increasing number of columns.
- No significant improvements obtained so far for square matrices.

Prospects:

- Improve the performance of the trailing matrix update by increasing the block size to optimize BLAS3 operations.
- Compare with the recent approach of [Hadri, Ltaief, Agullo, Dongarra'09] for QR factorization, which uses a different reduction tree during panel factorization.

Thank you

Thank you