Improving data locality in communication avoiding LU and QR factorizations

DONFACK Simplice

INRIA Saclay

Collaborators:

Laura Grigori, Bill Gropp, Dahai Guo, Vivek Kale

November 23, 2010

Introduction

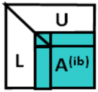
- Architectural trends show an increasing communication cost compared to the time it takes to perform arithmetic operations
  - Motivated the design of communication avoiding algorithms that minimize communication
  - First results are CAQR [Demmel, Grigori, Hoemmen, Langou '08] and CALU [Grigori, Demmel, Xiang '08], implemented for distributed memory.
- Multithreaded CALU and CAQR lead to important improvements for tall and skinny matrices but no significant improvements obtained so far for square matrices.
- Our goal is to evaluate and improve performance of our multithreaded algorithms on petascale machines.

LU factorization with partial pivoting

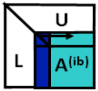
Factorization on Pr by Pc grid of processors as implemented in SCALAPACK:

For ib = 1 to n-1 step b

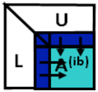
A(ib) = A(ib:n, ib:n)



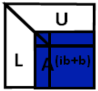
1. Compute panel factorization (pdgetf2) $O(nlog_2 P_r)$
   - find pivot in each column, swap rows



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

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.

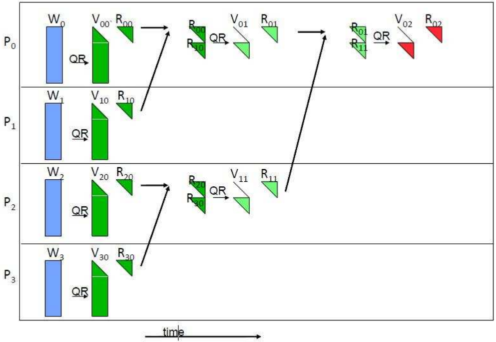


Figure: Parallel TSQR

CAQR

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

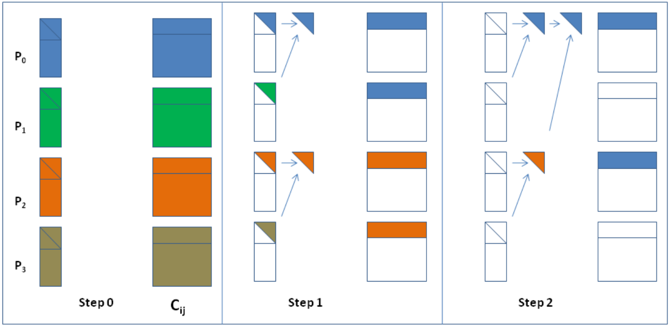
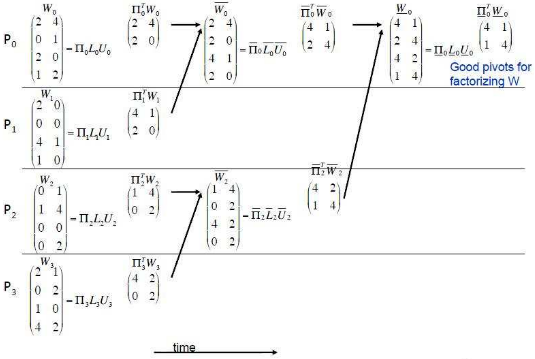


Figure: The update of the trailing submatrix is triggered by the reduction tree used during panel factorization

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



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

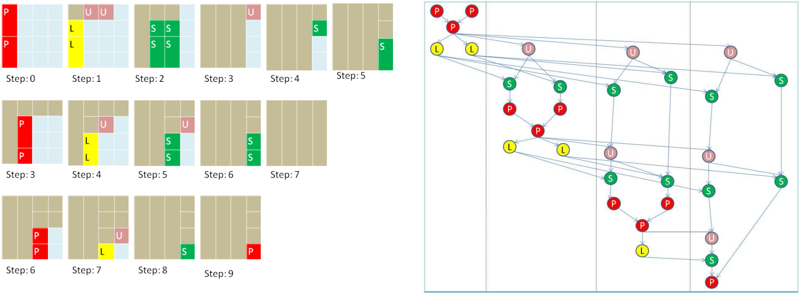
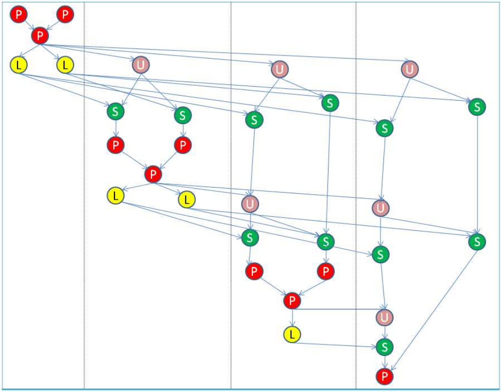


Figure: Matrix 4 $\times$ 4 blocks and $T_r = 2$ and Corresponding task dependency graph

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)

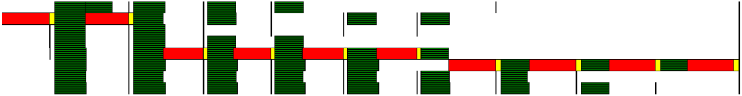


Figure: Example of execution of CALU for a $10^5 \times 1000$ tall skinny matrix, using $b = 100$ and $T_r = 1$, on 8-core

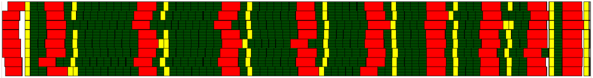


Figure: Example of execution of CALU for a $10^5 \times 1000$ tall skinny matrix, using $b = 100$ and $T_r = 8$, on 8-core

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

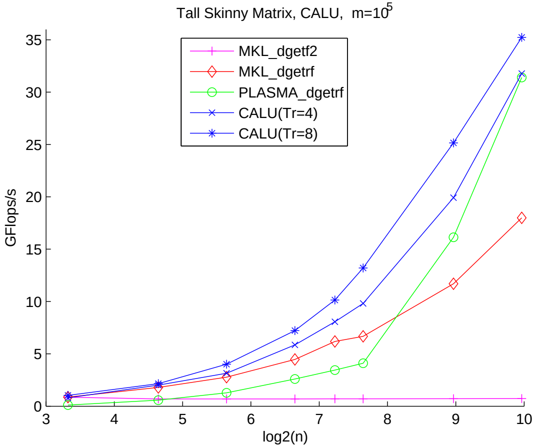


Figure: m=10^5 and varying n from 10 to 1000.

Performance of CAQR

Performance of CAQR, MKL_dgeqrf, PLASMA_dgeqrf on 8 cores

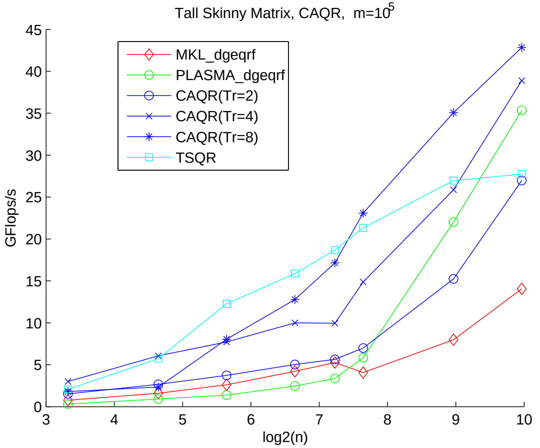


Figure: m=10^5 and varying n from 10 to 1000.

Profiling: CALU with dynamic scheduling

Figure: L2,L3 Cache misses on Bluewaters Power 7. CALU with dynamic scheduling. m=n=5000, b=150, $P = 4 \times 2$

Table: Average

L2 total cache misses	25M
L3 total cache misses	15M
Fetch task time	0.47%

CALU with dynamic scheduling with data locality

- Good mapping of task to processors using dynamic scheduler to improve data locality.
- Schedule a task using a queue of threads associated to recent operation on the corresponding bloc to reuse data.
- Minimize the number of transferts from slow to fast memory.

CALU with dynamic scheduling and strict data locality

- No workstealing
- Bad load balancing

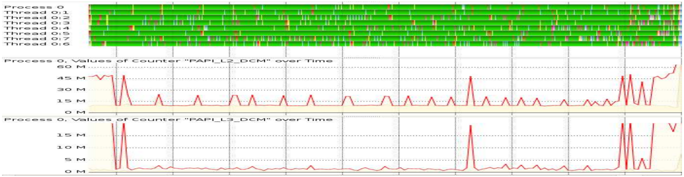


Figure: L2,L3 Cache misses on Bluewaters Power 7. CALU with dynamic scheduling and strict data locality . m=n=5000, b=150, $P = 4 \times 2$

Table: Average

L2 total cache misses	12.5M
L3 total cache misses	3.5M
Fetch task time	2.27%

CALU with dynamic scheduling, locality and workstealing

- Workstealing

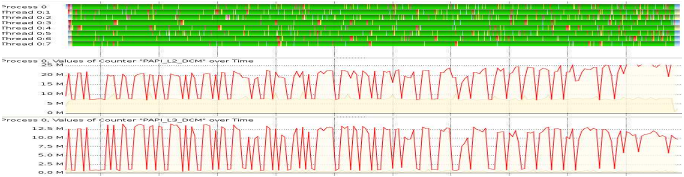


Figure: L2,L3 Cache misses on Bluewaters Power 7. CALU with dynamic scheduling, data locality and workstealing . m=n=5000, b=150, $P = 4 \times 2$

Table: Average

L2 total cache misses	20M
L3 total cache misses	12.5M
Fetch task time	1.58%

First results on Intel X86_64, 16 cores (using mkl blas)

Table: CALU performance, M=N=5000, b=100, values:Gflops/s

	Gflops/s
dgetrf	46.45
CALU (no pivoting, diagonal block as pivot)	66,08
CALU (dynamic scheduling)	59,96
CALU (dynamic + Locality)	60,51
CALU (panel perform by dgetrf)	54,19

First results on Bluewaters P7, (using essl)

Table: CALU performance, M=N=5000, b=100, values:Gflops/s

	2x4	4x4	4x8
CALU (no pivoting, diagonal block as pivot)	147,97	255,10	248,68
CALU (dynamic scheduling)	**134,17**	**206,60**	177,62
CALU (dynamic + Locality)	132,32	206,33	**177,72**

Conclusion

- Multithreaded CALU and CAQR lead to important improvements for tall and skinny matrices with respect to the corresponding routines in MKL and PLASMA.
- Improving data locality in CALU and CAQR is a trade-off between dequeue time and cache misses.

Prospects

- Improve the performance of the trailing matrix update by increasing the block size to optimize BLAS3 operations.
- Combining static/dynamic scheduling
    - Overlap the panel factorization with dynamic small task

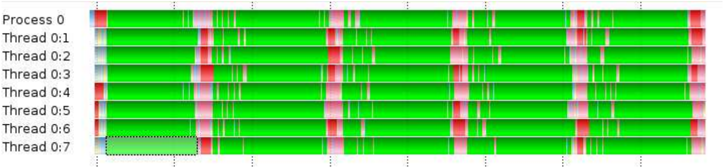


Figure: CAQR prediction

Thank you

Thank you