第４回先進スーパーコンピューティング環境研究会（ASE 研究会）発表資料

東京大学情報基盤センター　特任准教授　片桐孝洋

2009 年 3 月 27 日 14 時から 17 時まで、東京大学情報基盤センター大会議室にて、第 4 回先進スーパーコンピューティング環境研究会（ASE 研究会）が開催されました。

本号では、Julien Langou 博士（University of Colorado, Denver）の基調講演

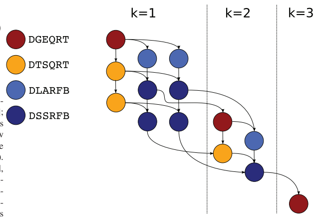
「Communication Optimal and Tiled Algorithms for Dense Linear Algebra: Auto-Tuning Opportunities in this new Design Space」

の発表資料を掲載させていただきます。

以上

Julien Langou, University of Colorado Denver.

ASE seminar (Advanced Supercomputing Environment)
Friday March 27th, 2008.



In this talk, we will first present recent communication-optimal and tiled algorithms for the LU factorization and the QR factorization introduced in [1,2,3,4]; then, we will motivate the need for performance auto-tuning in these algorithms and give some examples of opportunities in software auto-tuning. Our new communication-optimal and tiled algorithms represent a radical change with the current generation of linear algebra software (e.g. LAPACK and ScaLAPACK). They offer considerable advantage on a wide variety of platforms: sequential, multicore, parallel distributed, GPU acceleration. While we know that these algorithms are optimal in the big O sense, while we have proof-of-concept implementations of these algorithms, a lot remains to be done in finely tune these algorithms for a given (possibly heterogeneous) architecture. We believe the answer is in software auto-tuning.

For more information:

- Alfredo Buttari, Julien Langou, Jakub Kurzak and Jack Dongarra. A class of parallel tiled linear algebra algorithms for multicore architectures. *Parallel Computing*, 35:38-53, 2009.
- Alfredo Buttari, Julien Langou, Jakub Kurzak and Jack Dongarra. Parallel tiled QR factorization for multicore architectures. *Concurrency Computat.: Pract. Exper.*, 20(13):1573-1590, 2008.
- James W. Demmel, Laura Grigori, Mark F. Hoemmen, and Julien Langou. Communication-optimal parallel and sequential QR and LU factorizations. arXiv:0808.2664.
- James W. Demmel, Laura Grigori, Mark F. Hoemmen, and Julien Langou. Implementing Communication-Optimal Parallel and Sequential QR Factorizations. arXiv:0809.2407.

1. **TSQR: Tall Skinny QR**
2. **CAQR: Communication Avoiding QR**

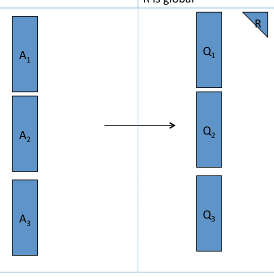
AllReduce Algorithms: Application to Householder QR Factorization

Jim Demmel, University of California, Berkeley;
Laura Grigori, INRIA, France;
Mark Hoemmen, University of California, Berkeley;
Julien Langou, University of Colorado, Denver

Reduce Algorithms: Introduction

The QR factorization of a long and skinny matrix with its data partitioned vertically across several processors arises in a wide range of applications.

Input: A is block distributed by rows	Output: Q is block distributed by rows R is global



Example of applications: in block iterative methods.

a) **in iterative methods with multiple right-hand sides** (block iterative methods:)
   1) Trilinos (Sandia National Lab.) through Belos (R. Lehoucq, H. Thornquist, U. Hetmaniuk).
   2) BlockGMRES, BlockGCR, BlockCG, BlockQMR, …
b) **in iterative methods with a single right-hand side**
   1) s-step methods for linear systems of equations (e.g. A. Chronopoulos),
   2) LGMRES (Jessup, Baker, Dennis, U. Colorado at Boulder) implemented in PETSc,
   3) Recent work from M. Hoemmen and J. Demmel (U. California at Berkeley).
e) **in iterative eigenvalue solvers,**
   1) PETSc (Argonne National Lab.) through BLOPEX (A. Knyazev, UCDHSC),
   2) HYPRE (Lawrence Livermore National Lab.) through BLOPEX,
   3) Trilinos (Sandia National Lab.) through Anasazi (R. Lehoucq, H. Thornquist, U. Hetmaniuk),
   4) PRIMME (A. Stathopoulos, Coll. William & Mary),
   5) And also TRLAN, BLZPACK, IRBLEIGS.

Reduce Algorithms: Introduction

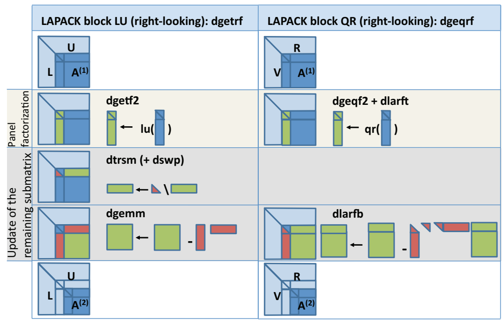
Example of applications:

a) in linear least squares problems which the number of equations is extremely larger than the number of unknowns

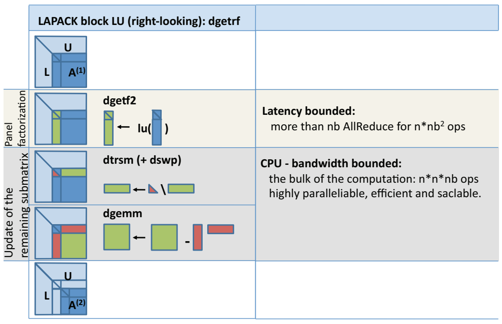
b) in block iterative methods (iterative methods with multiple right-hand sides or iterative eigenvalue solvers)

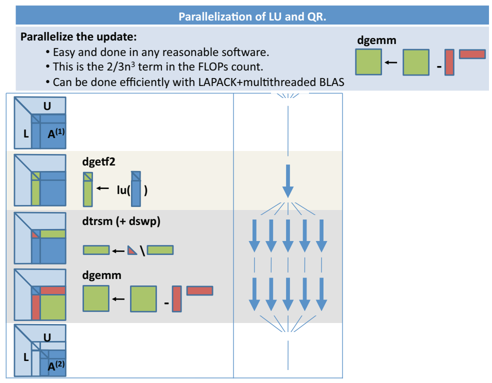
c) in dense large and more square QR factorization where they are used as the panel factorization step

Blocked LU and QR algorithms (LAPACK)



Blocked LU and QR algorithms (LAPACK)





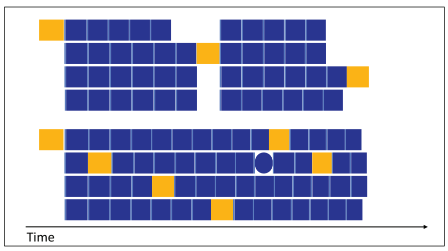
Parallelization of LU and QR.

Parallelize the update: (dgemm)
- Easy and done in any reasonable software.
- This is the $2/3n^3$ term in the FLOPs count.
- Can be done efficiently with LAPACK+multithreaded BLAS

Parallelize the panel factorization: (dgetf2, lu( ))
- Not an option in multicore context ($p < 16$)
- See e.g. ScaLAPACK or HPL but still by far the slowest and the bottleneck of the computation.

Hide the panel factorization: (dgetf2, lu( ))
- Lookahead (see e.g. High Performance LINPACK)
- Dynamic Scheduling

Hiding the panel factorization with dynamic scheduling.



Courtesy from Alfredo Buttari, UTennessee

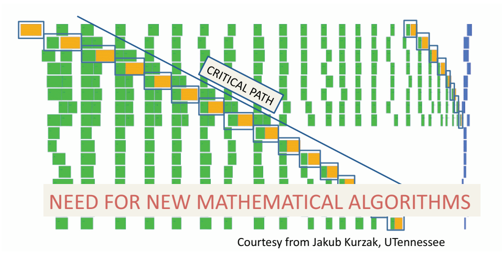
What about strong scalability?

Communication Optimal and Tiled Algorithms for Dense Linear Algebra:
Auto-Tuning Opportunities in this new Design Space.

What about strong scalability?

N = 1536
NB = 64
procs = 16

We can not hide the panel factorization (n^2) with the MM(n^3) , actually it is the MMs that are hidden by the panel factorizations!



Courtesy from Jakub Kurzak, UTennessee

Communication Optimal and Tiled Algorithms for Dense Linear Algebra:
Auto-Tuning Opportunities in this new Design Space.

A new generation of algorithms?

Algorithms follow hardware evolution along time.		
LINPACK (80's) (Vector operations)		Rely on - Level-1 BLAS operations
LAPACK (90's) (Blocking, cache friendly)		Rely on - Level-3 BLAS operations
New Algorithms (00's) (multicore friendly)		Rely on - a DAG/scheduler - block data layout - some extra kernels

Those new algorithms
- have a very low granularity, they scale very well (multicore, petascale computing, …)
- removes a lots of dependencies among the tasks, (multicore, distributed computing)
- avoid latency (distributed computing, out-of-core)
- rely on fast kernels

Those new algorithms need new kernels and rely on efficient scheduling algorithms.

Communication Optimal and Tiled Algorithms for Dense Linear Algebra:
Auto-Tuning Opportunities in this new Design Space.

2005-2007: New algorithms based on 2D partitionning:

- **UTexas (van de Geijn):** SYRK, CHOL (multicore), LU, QR (out-of-core)
- **UTennessee (Dongarra):** CHOL (multicore)
- **HPC2N (Kågström)/IBM (Gustavson):** Chol (Distributed)
- **UCBerkeley (Demmel)/INRIA(Grigori):** LU/QR (distributed)
- **UCDenver (Langou):** LU/QR (distributed)

A 3rd revolution for dense linear algebra?

Communication Optimal and Tiled Algorithms for Dense Linear Algebra:
Auto-Tuning Opportunities in this new Design Space.

Reduce Algorithms: Introduction

Example of applications:

a) in block iterative methods (iterative methods with multiple right-hand sides or iterative eigenvalue solvers),

b) in dense large and more square QR factorization where they are used as the panel factorization step, or more simply

c) in linear least squares problems which the number of equations is extremely larger than the number of unknowns.

The main characteristics of those three examples are that

a) there is **only one column of processors involved** but several processor rows,

b) **all the data is known from the beginning**,

c) and **the matrix is dense**.

Various methods already exist to perform the QR factorization of such matrices:

a) Gram-Schmidt (**mgs(row)**,**cgs**),

b) Householder (**qr2**, **qrf**),

c) or **CholeskyQR**.

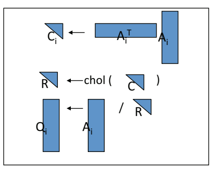
We present a new method:

Allreduce Householder (**rhh_qr3**, **rhh_qrf**).

Communication Optimal and Tiled Algorithms for Dense Linear Algebra:
Auto-Tuning Opportunities in this new Design Space.

The CholeskyQR Algorithm

SYRK:	$C := A^TA$	(mn^2)
CHOL:	$R :=$ chol(C)	($n^3/3$)
TRSM:	$Q := A/R$	(mn^2)



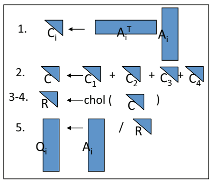
Bibligraphy

- A. Stathopoulos and K. Wu, A block orthogonalization procedure with constant synchronization requirements, *SIAM Journal on Scientific Computing*, 23(6):2165-2182, 2002.
- Popularized by iterative eigensolver libraries:
  1) PETSc (Argonne National Lab.) through BLOPEX (A. Knyazev, UCDHSC),
  2) HYPRE (Lawrence Livermore National Lab.) through BLOPEX,
  3) Trilinos (Sandia National Lab.) through Anasazi (R. Lehoucq, H. Thornquist, U. Hetmaniuk),
  4) PRIMME (A. Stathopoulos, Coll. William & Mary).

Parallel distributed CholeskyQR

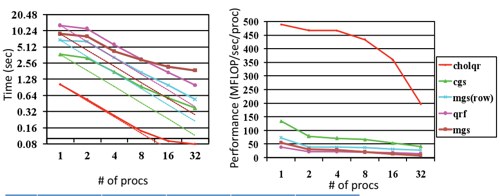
The CholeskyQR method in the parallel distributed context can be described as follows:

1: SYRK:	$C := A^TA$	(mn^2)
2: MPI_Reduce:	$C := \mathrm{sum}_{procs}\, C$	(on proc 0)
3: CHOL:	$R := \mathrm{chol}(C)$	($n^3/3$)
4: MPI_Bdcast	Broadcast the R factor on proc 0 to all the other processors	
5: TRSM:	$Q := A/R$	(mn^2)



In this experiment, we fix the problem: **m=100,000** and **n=50**.

Efficient enough?

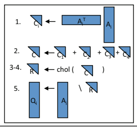


# of procs	cholqr		cgs		mgs(row)		qrf		mgs	
1	489.2	(1.02)	134.1	(3.73)	73.5	(6.81)	39.1	(12.78)	56.18	(8.90)
2	467.3	(0.54)	78.9	(3.17)	39.0	(6.41)	22.3	(11.21)	31.21	(8.01)
4	466.4	(0.27)	71.3	(1.75)	38.7	(3.23)	22.2	(5.63)	29.58	(4.23)
8	434.0	(0.14)	67.4	(0.93)	36.7	(1.70)	20.8	(3.01)	21.15	(2.96)
16	359.2	(0.09)	54.2	(0.58)	31.6	(0.99)	18.3	(1.71)	14.44	(2.16)
32	197.8	(0.08)	41.9	(0.37)	29.0	(0.54)	15.8	(0.99)	8.38	(1.87)

MFLOP/sec/proc

Time in sec

Simple enough?

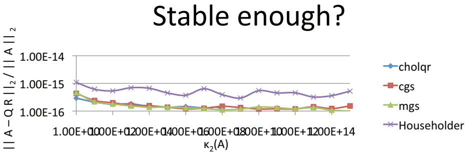


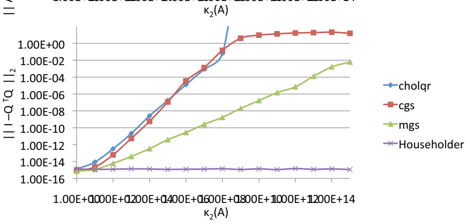

```
int choleskyqr_A_v0(int mloc, int n, double *A, int lda, double *R, int ldr,
          MPI_Comm mpi_comm){

          int info;
          cblas_dsyrk( CblasColMajor, CblasUpper, CblasTrans, n, mloc,
                    1.0e+00, A, lda, 0e+00, R, ldr );
          MPI_Allreduce( MPI_IN_PLACE, R, n*n, MPI_DOUBLE, MPI_SUM, mpi_comm );
          lapack_dpotrf( lapack_upper, n, R, ldr, &info );
          cblas_dtrsm( CblasColMajor, CblasRight, CblasUpper, CblasNoTrans, CblasNonUnit,
                    mloc, n, 1.0e+00, R, ldr, A, lda );
          return 0;

}
```

(… and, OK, you might want to add an MPI user defined datatype to send only the upper part of R)

m=100, n=50

Stable enough?

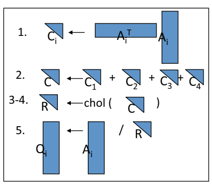




Parallel distributed CholeskyQR

The CholeskyQR method in the parallel distributed context can be described as follows:

1: SYRK:	$C := A^TA$	(mn^2)
2: MPI_Reduce:	$C := \mathrm{sum}_{procs}\, C$	(on proc 0)
3: CHOL:	$R := \mathrm{chol}(C)$	($n^3/3$)
4: MPI_Bdcast	Broadcast the R factor on proc 0 to all the other processors	
5: TRSM:	$Q := A/R$	(mn^2)



This method is extremely fast. For two reasons:

1. first, there is **only one or two communications phase**,
2. second, **the local computations are performed with fast operations**.

Another advantage of this method is that the resulting code is exactly four lines,

3. so the method is **simple** and relies heavily on other libraries.

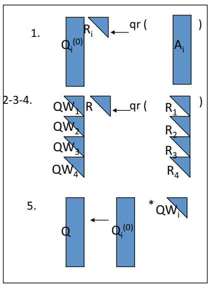
Despite all those advantages,

4. this method is **highly unstable**.

Reduce Algorithms

The gather-scatter variant of our algorithm can be summarized as follows:

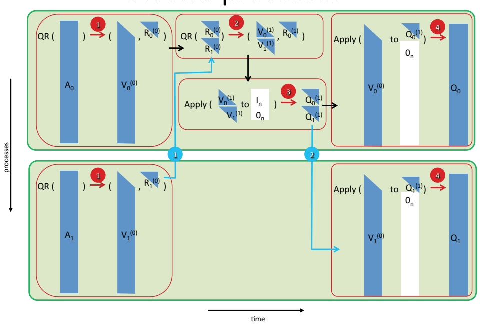
1. perform local QR factorization of the matrix A
2. gather the p R factors on processor 0
3. perform a QR factorization of all the R put the ones on top of the others, the R factor obtained is the R factor
4. scatter the the Q factors from processor 0 to all the processors
5. multiply locally the two Q factors together, done.



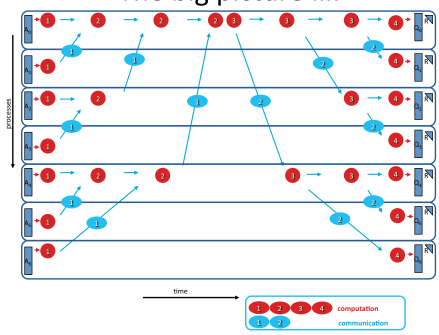
Reduce Algorithms

- This is the scatter-gather version of our algorithm.
- This variant is not very efficient for two reasons:
  - first the communication phases 2 and 4 are highly involving processor 0 ;
  - second the cost of step 3 is $p/3*n^3$, so can get prohibitive for large p.
- Note that the CholeskyQR algorithm can also be implemented in a **scatter-gather** way but **reduce-broadcast**. This leads naturally to the algorithm presented below where a reduce-broadcast version of the previous algorithm is described. This will be our final algorithm.

On two processes



The big picture ….

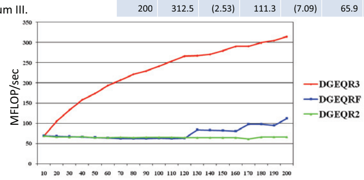


Latency but also possibility of fast panel factorization.

- **DGEQR3** is the recursive algorithm (see Elmroth and Gustavson, 2000), **DGEQRF** and **DGEQR2** are the LAPACK routines.
- Times include QR and DLARFT.
- Run on Pentium III.

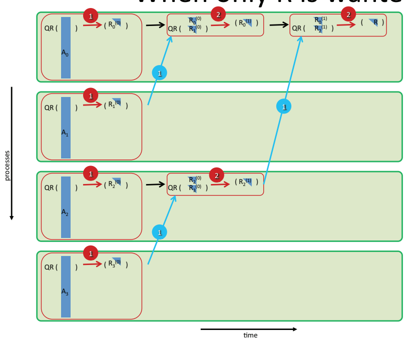
QR factorization and construction of T
m = 10,000
Perf in MFLOP/sec (Times in sec)

n	DGEQR3		DGEQRF		DGEQR2	
50	173.6	(0.29)	65.0	(0.77)	64.6	(0.77)
100	240.5	(0.83)	62.6	(3.17)	65.3	(3.04)
150	277.9	(1.60)	81.6	(5.46)	64.2	(6.94)
200	312.5	(2.53)	111.3	(7.09)	65.9	(11.98)



m=1000,000, the x axis is n

When only R is wanted



When only R is wanted: The MPI_Allreduce

In the case where only R is wanted, instead of constructing our own tree, one can simply use MPI_Allreduce with a user defined operation. The operation we give to MPI is basically the Algorithm 2. It performs the operation:

$$QR\left(\begin{matrix}R_1\\R_2\end{matrix}\right) \longrightarrow R$$

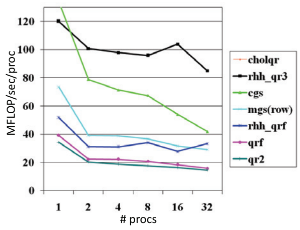
This **binary** operation is **associative** and this is all MPI needs to use a user-defined operation on a user-defined datatype. Moreover, if we change the signs of the elements of R so that the diagonal of R holds positive elements then the binary operation `Rfactor` becomes **commutative**.

The code becomes two lines:

```
lapack_dgeqrf( mloc, n, A, lda, tau, &dlwork, lwork, &info );
MPI_Allreduce( MPI_IN_PLACE, A, 1, MPI_UPPER,
        LILA_MPIOP_QR_UPPER, mpi_comm);
```


Q and R: Strong scalability

- In this experiment, we fix the problem: **m=100,000** and **n=50**. Then we increase the number of processors.
- Once more the algorithm **rhh_qr3** is the second behind **CholeskyQR**. Note that **rhh_qr3** is incondionnally stable while the stability of **CholeskyQR** depends on the square of the condition number of the initial matrix.

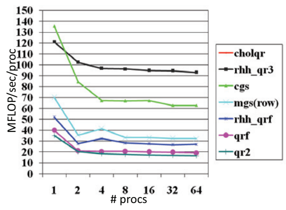


MFLOP/sec/proc — Time in sec

# of procs	cholqr		rhh_qr3		cgs		mgs(row)		rhh_qrf		qrf		qr2	
1	489.2	(1.02)	120.0	(4.17)	134.1	(3.73)	73.5	(6.81)	51.9	(9.64)	39.1	(12.78)	34.3	(14.60)
2	467.3	(0.54)	100.8	(2.48)	78.9	(3.17)	39.0	(6.41)	31.2	(8.02)	22.3	(11.21)	20.2	(12.53)
4	466.4	(0.27)	97.9	(1.28)	71.3	(1.75)	38.7	(3.23)	31.0	(4.03)	22.2	(5.63)	18.8	(6.66)
8	434.0	(0.14)	95.9	(0.65)	67.4	(0.93)	36.7	(1.70)	34.0	(1.84)	20.8	(3.01)	17.7	(3.54)
16	359.2	(0.09)	103.8	(0.30)	54.2	(0.58)	31.6	(0.99)	27.8	(1.12)	18.3	(1.71)	16.3	(1.91)
32	197.8	(0.08)	84.9	(0.18)	41.9	(0.37)	29.0	(0.54)	33.3	(0.47)	15.8	(0.99)	14.5	(1.08)

Q and R: Weak scalability with respect to m

- We fix the local size to be **mloc=100,000** and **n=50**. When we increase the number of processors, the global m grows proportionally.
- **rhh_qr3** is the Allreduce algorithm with recursive panel factorization, **rhh_qrf** is the same with LAPACK Householder QR. We see the obvious benefit of using recursion. See as well (6). **qr2** and **qrf** correspond to the ScaLAPACK Householder QR factorization routines.

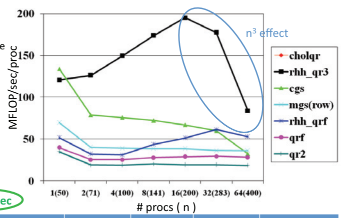


MFLOP/sec/proc — Time in sec

# of procs	cholqr		rhh_qr3		Cgs		mgs(row)		rhh_qrf		qrf		qr2	
1	489.2	(1.02)	121.2	(4.13)	135.7	(3.69)	70.2	(7.13)	51.9	(9.64)	39.8	(12.56)	35.1	(14.23)
2	466.9	(1.07)	102.3	(4.89)	84.4	(5.93)	35.6	(14.04)	27.7	(18.06)	20.9	(23.87)	20.2	(24.80)
4	454.1	(1.10)	96.7	(5.17)	67.2	(7.44)	41.4	(12.09)	32.3	(15.48)	20.6	(24.28)	18.3	(27.29)
8	458.7	(1.09)	96.2	(5.20)	67.1	(7.46)	33.2	(15.06)	28.3	(17.67)	20.5	(24.43)	17.8	(28.07)
16	451.3	(1.11)	94.8	(5.27)	67.2	(7.45)	33.3	(15.04)	27.4	(18.22)	20.0	(24.95)	17.2	(29.10)
32	442.1	(1.13)	94.6	(5.29)	62.8	(7.97)	32.5	(15.38)	26.5	(18.84)	19.8	(25.27)	16.9	(29.61)
64	414.9	(1.21)	93.0	(5.38)	62.8	(7.96)	32.3	(15.46)	27.0	(18.53)	19.4	(25.79)	16.6	(30.13)

Q and R: Weak scalability with respect to n

- We fix the global size **m=100,000** and then we increase n as sqrt(p) so that the workload mn^2 per processor remains constant.
- Due to better performance in the local factorization or SYRK, **CholeskyQR**, **rhh_q3** and **rhh_qrf** exhibit increasing performance at the beginning until the n^3 comes into play

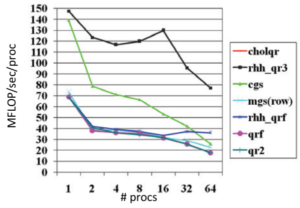


MFLOP/sec/proc — Time in sec

# of procs	cholqr		rhh_qr3		cgs		mgs(row)		rhh_qrf		qrf		qr2	
1	490.7	(1.02)	120.8	(4.14)	134.0	(3.73)	69.7	(7.17)	51.7	(9.68)	39.6	(12.63)	39.9	(14.31)
2	510.2	(0.99)	126.0	(4.00)	78.6	(6.41)	40.1	(12.56)	32.1	(15.71)	25.4	(19.88)	19.0	(26.56)
4	541.1	(0.92)	149.4	(3.35)	75.6	(6.62)	39.1	(12.78)	31.1	(16.07)	25.5	(19.59)	18.9	(26.48)
8	540.2	(0.92)	173.8	(2.86)	72.3	(6.87)	38.5	(12.89)	43.6	(11.41)	27.8	(17.85)	20.2	(24.58)
16	501.5	(1.00)	195.2	(2.56)	66.8	(7.48)	38.4	(13.02)	51.3	(9.75)	28.9	(17.29)	19.3	(25.87)
32	379.2	(1.32)	177.4	(2.82)	59.8	(8.37)	36.2	(13.84)	61.4	(8.15)	29.5	(16.95)	19.3	(25.92)
64	266.4	(1.88)	83.9	(5.96)	32.3	(15.46)	36.1	(13.84)	52.9	(9.46)	28.2	(17.74)	18.4	(27.13)

R only: Strong scalability

- In this experiment, we fix the problem: **m=100,000** and **n=50**. Then we increase the number of processors.

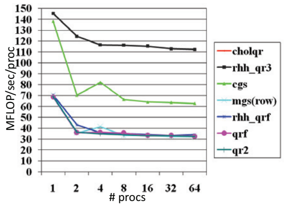


MFLOP/sec/proc — Time in sec

# of procs	cholqr		rhh_qr3		cgs		mgs(row)		rhh_qrf		qrf		qr2	
1	1099.046	(0.45)	147.6	(3.38)	139.309	(3.58)	73.5	(6.81)	69.049	(7.24)	69.108	(7.23)	68.782	(7.27)
2	1067.856	(0.23)	123.424	(2.02)	78.649	(3.17)	39.0	(6.41)	41.837	(5.97)	38.008	(6.57)	40.782	(6.13)
4	1034.203	(0.12)	116.774	(1.07)	71.101	(1.76)	38.7	(3.23)	39.295	(3.18)	36.263	(3.44)	36.046	(3.47)
8	876.724	(0.07)	119.856	(0.52)	66.513	(0.94)	36.7	(1.70)	37.397	(1.67)	35.313	(1.77)	34.081	(1.83)
16	619.02	(0.05)	129.808	(0.24)	53.352	(0.59)	31.6	(0.99)	33.581	(0.93)	31.339	(0.99)	31.697	(0.98)
32	468.332	(0.03)	95.607	(0.16)	42.276	(0.37)	29.0	(0.54)	37.226	(0.42)	25.695	(0.60)	25.971	(0.60)
64	195.885	(0.04)	77.084	(0.10)	25.89	(0.30)	22.8	(0.34)	36.126	(0.22)	17.746	(0.44)	17.725	(0.44)

R only: Weak scalability with respect to m

- We fix the local size to be mloc=100,000 and n=50. When we increase the number of processors, the global m grows proportionally.



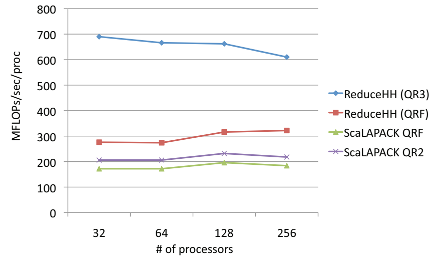
MFLOP/sec/proc — Time in sec

# of procs	cholqr		rhh_qr3		cgs		mgs(row)		rhh_qrf		qrf		qr2	
1	1098.7	(0.45)	145.4	(3.43)	138.2	(3.61)	70.2	(7.13)	70.6	(7.07)	68.7	(7.26)	69.1	(7.22)
2	1048.3	(0.47)	124.3	(4.02)	70.3	(7.11)	35.6	(14.04)	43.1	(11.59)	35.8	(13.95)	36.3	(13.76)
4	1044.0	(0.47)	116.5	(4.29)	82.0	(6.09)	41.4	(12.09)	35.8	(13.94)	36.3	(13.74)	34.7	(14.40)
8	993.9	(0.50)	116.2	(4.30)	66.3	(7.53)	33.2	(15.06)	35.1	(14.21)	35.5	(14.05)	33.8	(14.75)
16	918.7	(0.54)	115.2	(4.33)	64.1	(7.79)	33.3	(15.04)	34.0	(14.66)	33.4	(14.94)	33.0	(15.11)
32	950.7	(0.52)	112.9	(4.42)	63.6	(7.85)	32.5	(15.38)	33.4	(14.95)	33.3	(15.01)	32.9	(15.19)
64	764.6	(0.65)	112.3	(4.45)	62.7	(7.96)	32.3	(15.46)	34.0	(14.66)	32.6	(15.33)	32.3	(15.46)

Blue Gene L
frost.ncar.edu

Q and R: Strong scalability

In this experiment, we fix the problem: m=1,000,000 and n=50. Then we increase the number of processors.



Conclusions

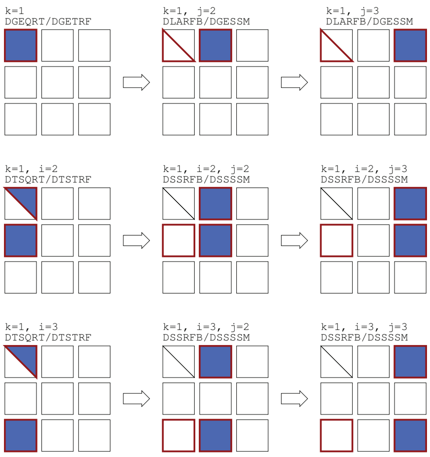
We have described a new method for the Householder QR factorization of skinny matrices. The method is named Allreduce Householder and has four advantages:

1. there is only one synchronization point in the algorithm,
2. the method harvests most of efficiency of the computing unit by large local operations,
3. the method is stable,
4. and finally the method is elegant in particular in the case where only R is needed.

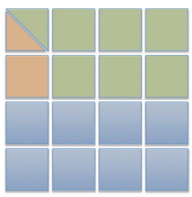
Allreduce algorithms have been depicted here with Householder QR factorization. However it can be applied to *anything* for example Gram-Schmidt or LU.

Current development is in writing a 2D block cyclic QR factorization and LU factorization based on those ideas.

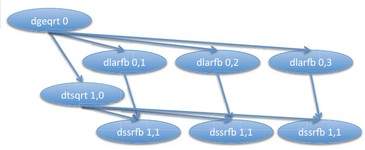
1. TSQR: Tall Skinny QR
2. CAQR: Communication Avoiding QR



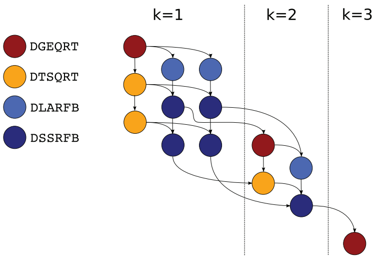
Alfredo Buttari, Julien Langou, Jakub Kurzak, and Jack Dongarra. *LAWN 191 – A Class of Parallel Tiled Linear Algebra Algorithms for Multicore Architectures*, September 2007.



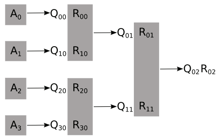
...
4. dlarfb(A[0][0], T[0][0], A[0][3]);
5. dtsqrt(A[0][0], A[1][0], T[1][0]);
6. dssrfb(A[1][0], T[1][0], A[0][1], A[1][1]);
7. dssrfb(A[1][0], T[1][0], A[0][1], A[1][1]);
8. dssrfb(A[1][0], T[1][0], A[0][1], A[1][1]);




```
for (k = 0; k < TILES; k++) {
    dgeqrt(A[k][k], T[k][k]);
    for (n = k+1; n < TILES; n++) {
        dlarfb(A[k][k], T[k][k], A[k][n]);
    for (m = k+1; m < TILES; m++){
        dtsqrt(A[k][k], A[m][k], T[m][k]);
        for (n = k+1; n < TILES; n++)
            dssrfb(A[m][k], T[m][k], A[k][n], A[m][n]);
    }
}
```

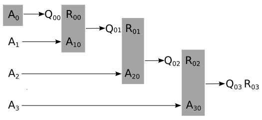


Communication Optimal and Tiled Algorithms for Dense Linear Algebra: Auto-Tuning Opportunities in this new Design Space.



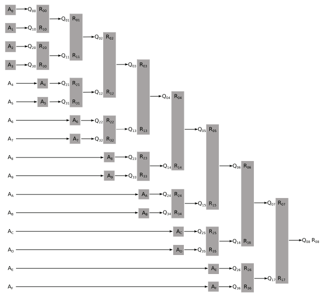
Execution of the parallel TSQR factorization on a binary tree of four processors. The gray boxes indicate where local QR factorizations take place. The Q and R factors each have two subscripts: the first is the sequence number within that stage, and the second is the stage number.

Communication Optimal and Tiled Algorithms for Dense Linear Algebra: Auto-Tuning Opportunities in this new Design Space.



Execution of the sequential TSQR factorization on a flat tree with four submatrices. The gray boxes indicate where local QR factorizations take place The Q and R factors each have two subscripts: the first is the sequence number for that stage, and the second is the stage number.

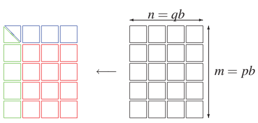
Communication Optimal and Tiled Algorithms for Dense Linear Algebra: Auto-Tuning Opportunities in this new Design Space.



Execution of a hybrid parallel / out-of-core TSQR factorization. The matrix has 16 blocks, and four processors can execute local QR factorizations simultaneously. The gray boxes indicate where local QR factorizations take place. We number the blocks of the input matrix A in hexadecimal to save space (which means that the subscript letter A is the number 10_{10}, but the non-subscript letter A is a matrix block). The Q and R factors each have two subscripts: the first is the sequence number for that stage, and the second is the stage number.

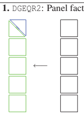
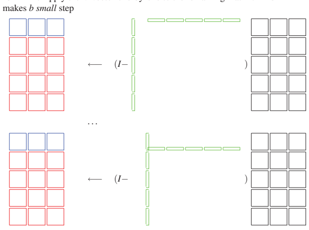
Communication Optimal and Tiled Algorithms for Dense Linear Algebra: Auto-Tuning Opportunities in this new Design Space.

A is m–by–n with $m = pb$ and $n = qb$.
We are only interested in the first step of the Householder QR factorization.



Communication Optimal and Tiled Algorithms for Dense Linear Algebra: Auto-Tuning Opportunities in this new Design Space.

1 Classic unblocked Householder factorization: DGEQF2

1. `DGEQR2`: Panel factorization. 	$2mb^2 - \frac{2}{3}b^3 = 2(p - \frac{1}{3})b^3$
2. `DLARF`: Apply the V vector one-by-one to the remaining matrix. This makes b *small* step 	$\sum_{\ell=(p-1)b+1}^{pb} 4(q-1)b\ell$ $= 4(q-1)b\sum_{i=1}^{b}(p-1)b+i$ $= 4(q-1)b((p-1)b^2 + \frac{1}{2}b^2)$ $= (4p-2)(q-1)b^3$
total	$(\mathbf{4pq - 2p - 2q + \frac{4}{3})b^3}$

2 Classic block Householder factorization: DGEQRF

1. DGEQR2: Panel factorization.	$2mb^2 - \frac{2}{3}b^3 = 2(p-\frac{1}{3})b^3$
2. DLARFT: Construction of the T matrix to apply the Househodler by block	$mb^2 - \frac{1}{3}b^3 = (p-\frac{1}{3})b^3$
3. DLARFB: Apply the V vectors by block to the remaining matrix. $\longleftarrow (I - \quad {}^T \quad)$	$4bm(n-b) - b^2(n-b)$ $= (4p-1)(q-1)b^3$ Or doing it y hand: $(2p-1)(q-1)b^3$ $+(q-1)b^3$ $+(2p-1)(q-1)b^3$ $= (4p-1)(q-1)b^3$
total	$\mathbf{(4pq-p-q)b^3}$

So if we want to compare quickly the unblocked Househodler code with the block Househodler code, there is an overhead of $(p+q)b^3$. There pb^3 overhead due to DLARFT and qb^3 due to DLARFB.

Communication Optimal and Tiled Algorithms for Dense Linear Algebra:
Auto-Tuning Opportunities in this new Design Space.

3 SUQRA

Step 1.	
1.a. First QR factorization.	$\frac{4}{3}b^3$
1.b DLARFT: Construction of the T matrix to apply the Househodler by block	$\frac{2}{3}b^3$
1.c. DLARFB: Apply the V vectors by block to the remaining matrix. $\longleftarrow (I - \quad {}^T \quad)$	$3(q-1)b^3$
Step 2. (repeat this step $(p-1)$ times.	
2.a. TSQR factorization (TS=triangle-square)	$2b^3$
2.b TS-LARFT: Construction of the T matrix to apply the Househodler by block	$\frac{4}{3}b^3$
2.c. TS-LARFB: Apply the V vectors by block to the remaining matrix. $\longleftarrow (I - \quad {}^T \quad)$	$5(q-1)b^3$
total	$\mathbf{(5pq-\frac{5}{3}p-2q+\frac{2}{3})b^3}$

This algorithm is going to perform 20% more FLOPS.⇒ Need for inner blocking.

Communication Optimal and Tiled Algorithms for Dense Linear Algebra:
Auto-Tuning Opportunities in this new Design Space.

4 SUQRA without blocking

Step 1.	
1.a. First QR factorization.	$\frac{4}{3}b^3$
1.c. DLARFB: Apply the V vectors by block to the remaining matrix. $\longleftarrow$ Apply to	$2(q-1)b^3$
Step 2. (repeat this step $(p-1)$ times.	
2.a. TSQR factorization (TS=triangle-square)	$2b^3$
2.c. TS-LARFB: Apply the V vectors by block to the remaining matrix. $\longleftarrow$ Apply to	$4(q-1)b^3$
total	$\mathbf{(4pq-2p-2q+\frac{4}{3})b^3}$

Exactly the same number of FLOPs as for the unblocked QR factorization

Communication Optimal and Tiled Algorithms for Dense Linear Algebra:
Auto-Tuning Opportunities in this new Design Space.

5 comparison

	unblocked QR	unblocked SUQRA	blocked QR	blockedSUQRA
panel	$(2p-\frac{2}{3})b^3$	$(2p-\frac{2}{3})b^3$	$(2p-\frac{2}{3})b^3$	$(2p-\frac{2}{3})b^3$
T			$(p-\frac{1}{3})b^3$	$(\frac{4}{3}p-\frac{2}{3})b^3$
update	$(4pq-2q-4p+2)b^3$	$(4pq-2q-4p+2)b^3$	$(4pq-q-4p+1)b^3$	$(5pq-2q-5p+2)b^3$
total	$(4pq-2q-2p+\frac{4}{3})b^3$	$(4pq-2q-2p+\frac{4}{3})b^3$	$(4pq-q-p)b^3$	$(5pq-2q-\frac{5}{3}p+\frac{2}{3})b^3$

We can integrate those values to know the FLOPS count of the algorithm, we replace q by i when i varies from q to 1 and we replace p with $p-q+i$. We assume that $p>q$, i.e. the matrix is taller than longer.
This gives for the unblocked QR algorithm:

$$\begin{aligned}
& \sum_{i=1}^{q}(4(p-q+i)i-2i-2(p-q+i)+\frac{4}{3})b^3 \\
= & \sum_{i=1}^{q}(4p-4q-4)i+4i^2-2p+2q+\frac{4}{3})b^3 \\
= & (4p-4q-4)\frac{q^2}{2}+4\frac{q^3}{3}-2pq+2q^2+\frac{4}{3}q)b^3 \\
= & (2pq^2-2q^3-2q^2+\frac{4}{3}q^3-2pq+2q^2+\frac{4}{3}q)b^3 \\
= & (2pq^2-\frac{2}{3}q^3-2pq+\frac{4}{3}q)b^3 \\
= & 2mn^2-\frac{2}{3}n^3-2mnb+\frac{4}{3}nb^2
\end{aligned}$$

The order is OK we find: $2mn^2-\frac{2}{3}n^3$ as expected.
This gives for the block SUQRA:

$$\begin{aligned}
& \sum_{i=1}^{q}(5(p-q+i)i-2i-\frac{5}{3}(p-q+i)+\frac{2}{3})b^3 \\
\sim & \sum_{i=1}^{q}(5pi-5qi+5i^2)b^3 \\
\sim & (\frac{5}{2}pq^2-\frac{5}{2}q^3+\frac{5}{3}q^3)b^3 \\
\sim & \frac{5}{2}mn^2-\frac{5}{6}n^3
\end{aligned} \tag{1}$$

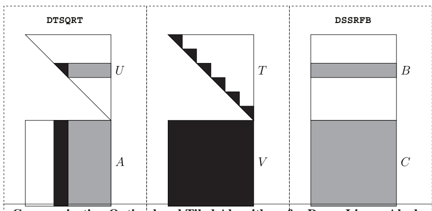
which means 20% overhead.

Communication Optimal and Tiled Algorithms for Dense Linear Algebra:
Auto-Tuning Opportunities in this new Design Space.

6 Remark on the panel factorization

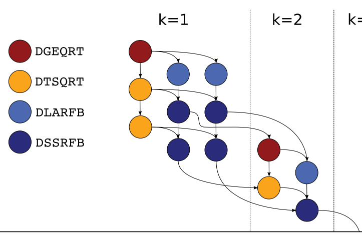
Classical method		
$\longleftarrow$	$2(2b)b^2-\frac{2}{3}b^3$	$\frac{10}{3}b^3$
PUQRA PUQRA-step1	$\frac{4}{3}b^3+\frac{4}{3}b^3$	
PUQRA-step2	$\frac{2}{3}b^3$	$\frac{10}{3}b^3$
SUQRA SUQRA-step1	$\frac{4}{3}b^3$	
SUQRA-step2	$2b^3$	$\frac{10}{3}b^3$

Communication Optimal and Tiled Algorithms for Dense Linear Algebra:
Auto-Tuning Opportunities in this new Design Space.



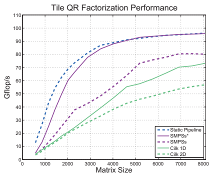
Communication Optimal and Tiled Algorithms for Dense Linear Algebra:
Auto-Tuning Opportunities in this new Design Space.

Alfredo Buttari, Julien Langou, Jakub Kurzak, and Jack Dongarra. *LAWN 191 – A Class of Parallel Tiled Linear Algebra Algorithms for Multicore Architectures*, September 2007.



An interesting middleware: SMPSs

```
#pragma css task \
   inout(RV1[NB][NB]) output(T[NB][NB])
void dgeqrt(double *RV1, double *T);

#pragma css task \
   inout(R[NB][NB], V2[NB][NB]) output(T[NB][NB])
void dtsqrt(double *R, double *V2, double *T);

#pragma css task \
   input(V1[NB][NB], T[NB][NB]) inout(C1[NB][NB])
void dlarfb(double *V1, double *T, double *C1);

#pragma css task \
   input(V2[NB][NB], T[NB][NB]) inout(C1[NB][NB], C2[NB][NB])
void dssrfb(double *V2, double *T, double *C1, double *C2);


#pragma css start
for (k = 0; k < TILES; k++) {

   dgeqrt(A[k][k], T[k][k]);

   for (m = k+1; m < TILES; m++)
      dtsqrt(A[k][k], A[m][k], T[m][k]);

   for (n = k+1; n < TILES; n++) {
      dlarfb(A[k][k], T[k][k], A[k][n]);
      for (m = k+1; m < TILES; m++)
         dssrfb(A[m][k], T[m][k], A[k][n], A[m][n]);
   }
}
#pragma css finish
```

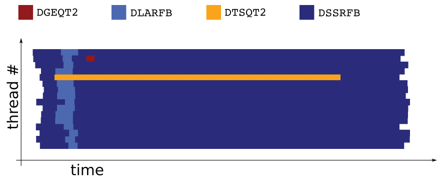


From:

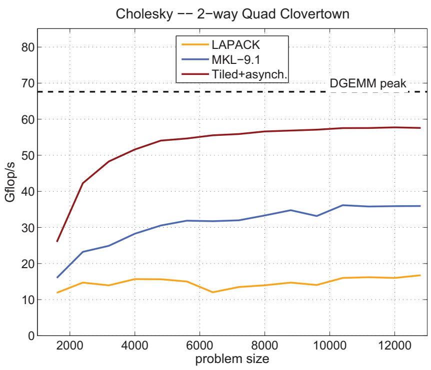
- Jakub Kurzak, Hatem Ltaief, Jack Dongarra, and Rosa M. Badia. *Scheduling Linear Algebra Operations on Multicore Processors.* LAWN 213.

See also:

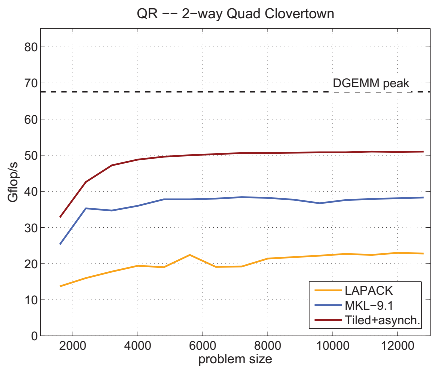
- Rosa M. Badia, José R. Herrero, Jesús Labarta, Josep M. Pérez, Enrique S. Quintana-Ortí, and Gregorio Quintana-Ortí. *Parallelizing dense and banded linear algebra libraries using SMPSs.* UPC-DAC-RR-2008-64.
- Hatem Ltaief, Jakub Kurzak, and Jack Dongarra. *Scheduling Two-sided Transformations using Algorithms-by-Tiles on Multicore Architectures.* LAWN 214.



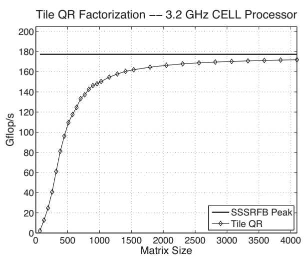
Alfredo Buttari, Julien Langou, Jakub Kurzak, and Jack Dongarra. *LAWN 191 – A Class of Parallel Tiled Linear Algebra Algorithms for Multicore Architectures*, September 2007.



Alfredo Buttari, Julien Langou, Jakub Kurzak, and Jack Dongarra. *LAWN 191 – A Class of Parallel Tiled Linear Algebra Algorithms for Multicore Architectures*, September 2007.



Alfredo Buttari, Julien Langou, Jakub Kurzak, and Jack Dongarra. *LAWN 191 – A Class of Parallel Tiled Linear Algebra Algorithms for Multicore Architectures*, September 2007.



Performance of the tile QR factorization in single precision on a 3.2 GHz CELL processor with eight SPEs. Square matrices were used. Solid horizontal line marks performance of the SSSRFB kernel times the number of SPEs ($22.16 \times 8 = 177$ [Gflop/s]).

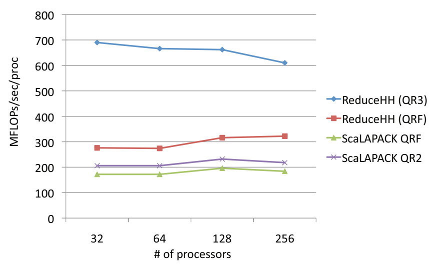
"The presented implementation of tile QR factorization on the CELL processor allows for factorization of a 4000–by–4000 dense matrix in single precision in exactly half of a second. To the author's knowledge, at present, it is the fastest reported time of solving such problem by any semiconductor device implemented on a single semiconductor die."

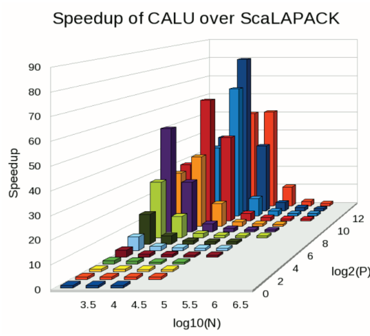
Jakub Kurzak and Jack Dongarra, LAWN 201 – QR Factorization for the CELL Processor, May 2008.

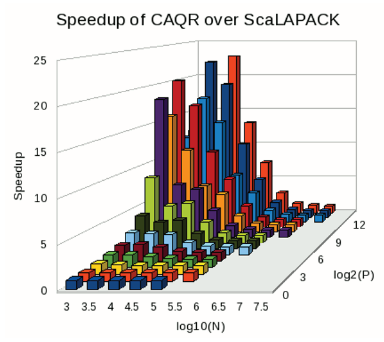
Blue Gene L
frost.ncar.edu

Q and R: Strong scalability

In this experiment, we fix the problem: **m=1,000,000** and **n=50**.
Then we increase the number of processors.







Strategy:

1. obtain some lower bounds for the cost (latency, bandwidth, # of operations) of LU, QR and Cholesky in sequential and parallel distributed
2. compute the costs of our algorithms and compare with the lower bound.

Lower bounds:

1. For LU, observe that:

$$\begin{pmatrix} I & 0 & -B \\ A & I & 0 \\ 0 & 0 & I \end{pmatrix} = \begin{pmatrix} I & & \\ A & I & \\ 0 & 0 & I \end{pmatrix} \begin{pmatrix} I & 0 & -B \\ & I & A\cdot B \\ & & I \end{pmatrix}$$

   therefore lower bound for matrix-matrix multiply (latency, bandwidth and operations) also holds for LU.

2. For Cholesky, observe that:

$$\begin{pmatrix} I & A^T & -B \\ A & I+AA^T & 0 \\ -B^T & 0 & D \end{pmatrix} = \begin{pmatrix} I & & \\ A & I & \\ -B^T & (A\cdot B)^T & X \end{pmatrix} \begin{pmatrix} I & A^T & -B \\ & I & A\cdot B \\ & & X^T \end{pmatrix}$$

   however this gets nasty due to the AA^T term in the initial matrix A. See Grey Ballard, James Demmel, Olga Holtz, and Oded Schwartz. *Communication-optimal Parallel and Sequential Cholesky decomposition.* UCB/EECS-2009-29, February 13th, 2009.

3. For QR, we needed to redo the proof of optimality of matrix-matrix multiply. See James W. Demmel, Laura Grigori, Mark F. Hoemmen, and Julien Langou. *Communication-avoiding parallel and sequential QR factorizations.* arXiv:0902.2537, May 30th, 2008.

	Par. CAQR	PDGEQRF	Lower bound
# flops	$\frac{4n^3}{3P}$	$\frac{4n^3}{3P}$	$O\left(\frac{n^3}{P}\right)$
# words	$\frac{3n^2}{4\sqrt{P}}\log P$	$\frac{3n^2}{4\sqrt{P}}\log P$	$O\left(\frac{n^2}{\sqrt{P}}\right)$
# messages	$\frac{3}{8}\sqrt{P}\log^3 P$	$\frac{5n}{4}\log^2 P$	$O\left(\sqrt{P}\right)$

Performance models of parallel CAQR and ScaLAPACK's parallel QR factorization PDGEQRF on a square $n \times n$ matrix with P processors, along with lower bounds on the number of flops, words, and messages. The matrix is stored in a 2-D $P_r \times P_c$ block cyclic layout with square $b \times b$ blocks. We choose b, P_r, and P_c optimally and independently for each algorithm. Everything (messages, words, and flops) is counted along the critical path.

	Seq. CAQR	Householder QR	Lower bound
# flops	$\frac{4}{3}n^3$	$\frac{4}{3}n^3$	$O(n^3)$
# words	$3\frac{n^3}{\sqrt{W}}$	$\frac{1}{3}\frac{n^4}{W}$	$O(\frac{n^3}{\sqrt{W}})$
# messages	$12\frac{n^3}{W^{3/2}}$	$\frac{1}{2}\frac{n^3}{W}$	$O(\frac{n^3}{W^{3/2}})$

Performance models of sequential CAQR and blocked sequential Householder QR on a square $n \times n$ matrix with fast memory size W, along with lower bounds on the number of flops, words, and messages.

Autotuning opportunities:

kernel tuning: introduction of a lots of new kernels (e.g. QR fact. of a triangle on top of a square). For each kernel:

1. how to optimize the blocking parameter (*nb*)?
2. which algorithmic variants to choose (left looking, recursive, ...) ?
3. the inner blocking parameter (*ib*).

Question 2 and 3 are standard autotuning problems. Choosing *ib* and the algorithmic vairant is done in term of *nb*. Question 1 is more subtle. Choosing *nb* is done at the matrix level (n) since it influences the granularity of the algorithm.

reduction algorithm: Which reduction tree to use? Binary tree? Flat tree? Hybrid tree? Each of these choices represent an algorithm change. No framework to accomodate this yet.

scheduling: How to schedule all these tasks?

1. static scheduling or dynamic scheduling?
2. and in parallel distributed ...