III.1 Re-design of Higher level Matrix Algorithms for Multicore and Heterogeneous Architectures

Based on the presentation at UC Berkeley, October 7, 2009

Background and motivation

Running time of an algorithm is the sum of three items:

1. # flops $\times$ time-per-flop
2. # words moved/bandwith (memory communication)
3. # messages $\times$ latency (network communication)

Hardward trend: exponentially growing gaps

- Time-per-flop $\ll$ 1/Memory BW $\ll$ Memory Latency

  improving 59%/year vs 23%/year vs 5.5%/year

- Time-per-flop $\ll$ 1/Network BW $\ll$ Network Latency

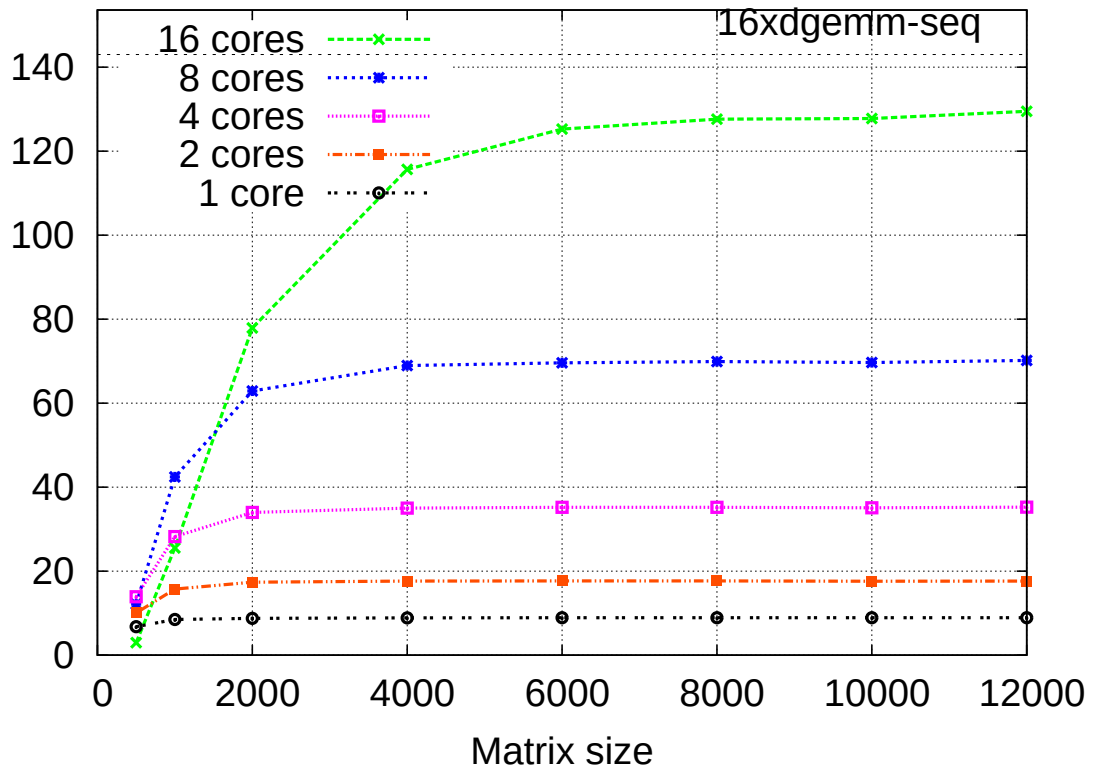
  improving 59%/year vs 26%/year vs 15%/year

Goal: re-design matrix algorithms to reduce communication and optimize the performance

Recent work by Dongarra, Demmel, *et al*, ...

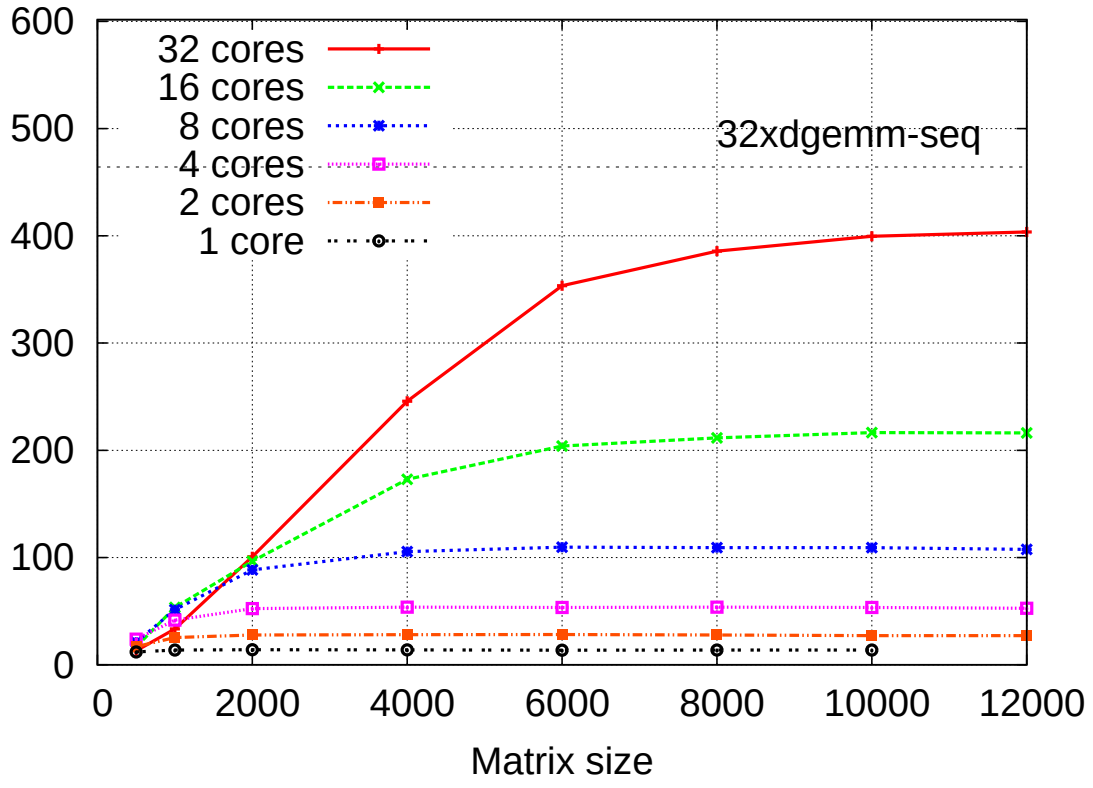
Background and motivation – Experts' benchmark

Matrix multiplication of MKL library on Intel Xeon EMT64:



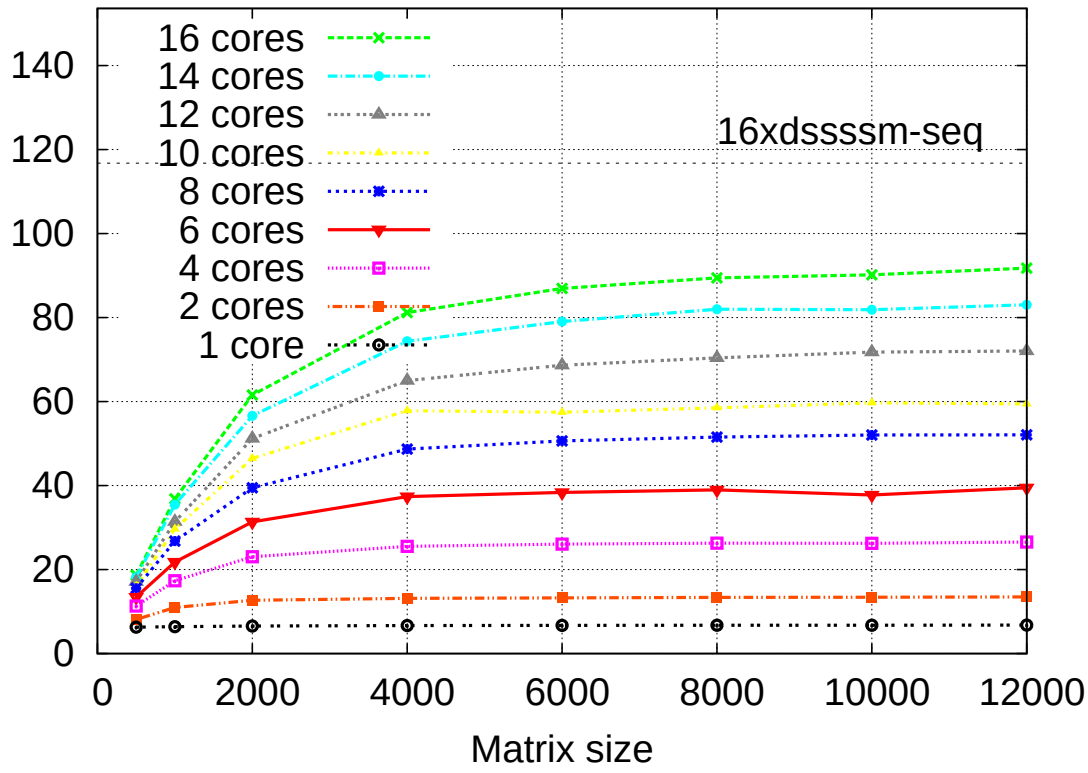
matrix size and Gflops

Matrix multiplication of ESSL library on an IBM Power 6:



matrix size and Gflops

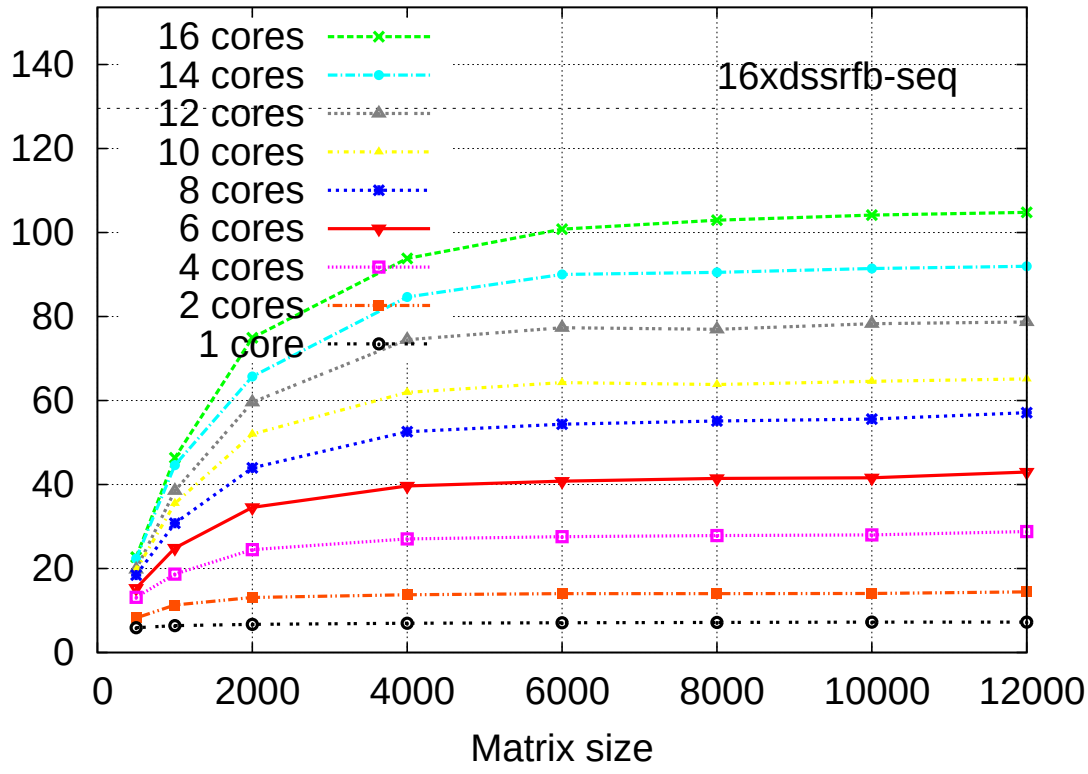
PLASMA_DGETRF on Intel Xeon EMT64



matrix size and Gflops

Courtesy of Innovative Computing Laboratory at the University of Tennessee (J. Dongarra *et al*)

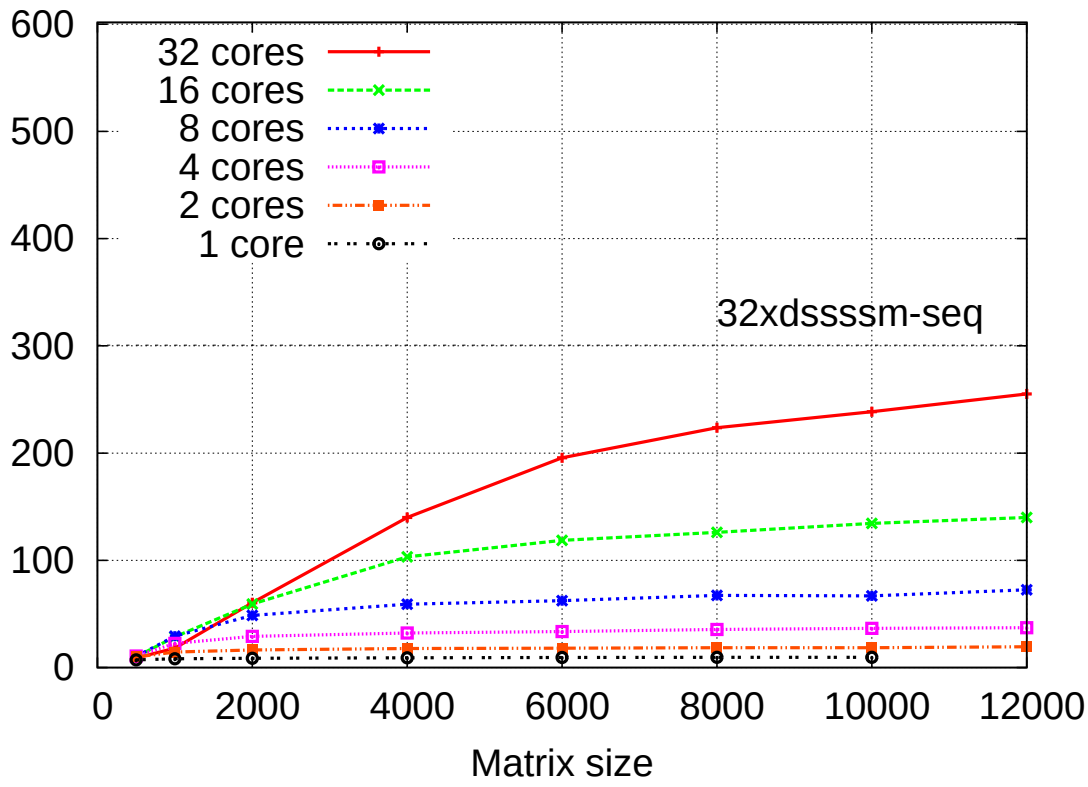
PLASMA_DGEQRF on Intel Xeon EMT64



matrix size and Gflops

Courtesy of Innovative Computing Laboratory at the University of Tennessee (J. Dongarra *et al*)

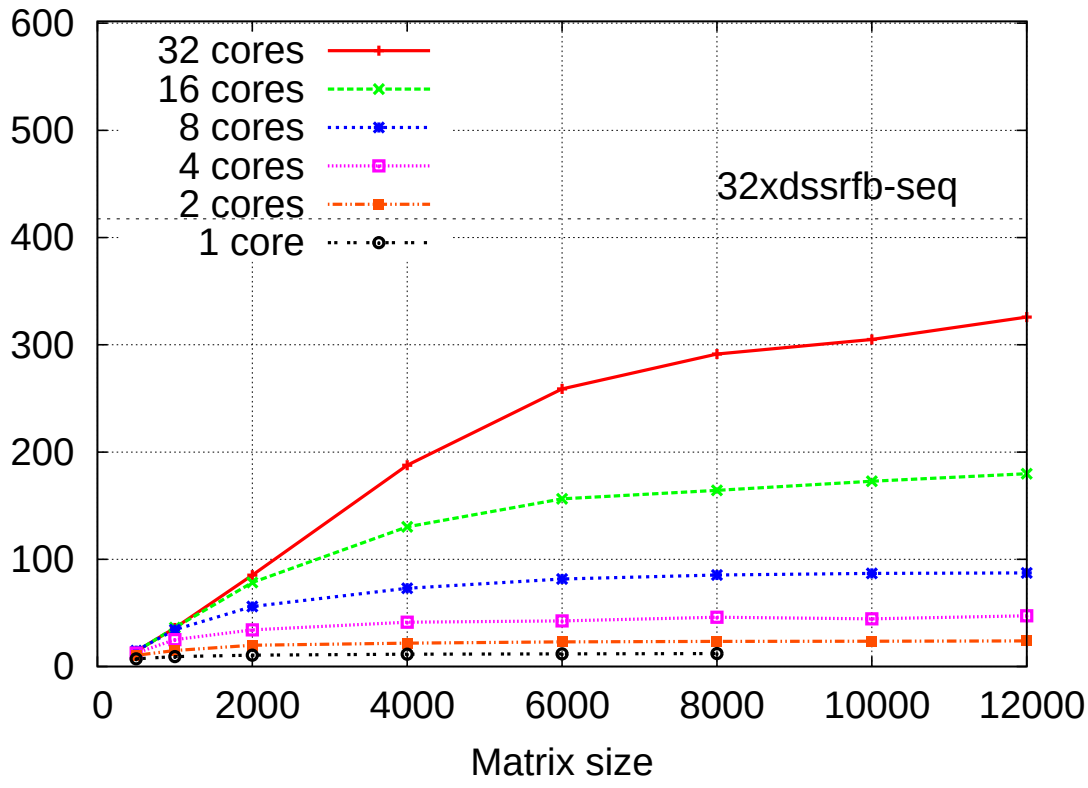
PLASMA_DGETRF on IBM Power 6:



matrix size and Gflops

Courtesy of Innovative Computing Laboratory at the University of Tennessee (J. Dongarra *et al*)

PLASMA_DGEQRF on IBM Power 6:

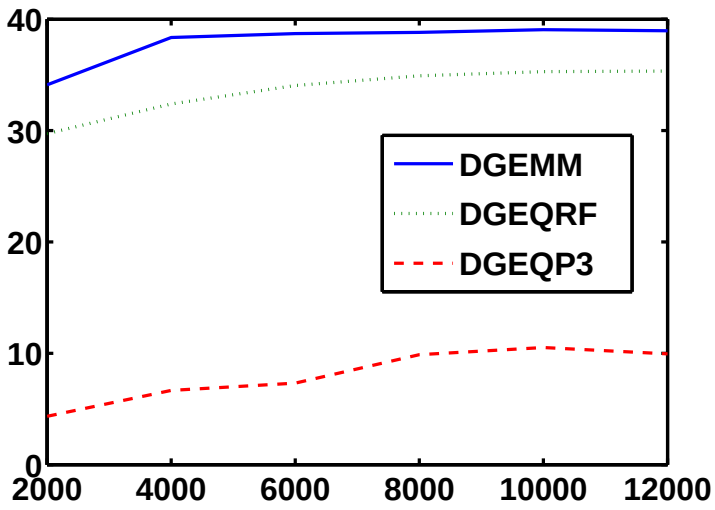


matrix size and Gflops

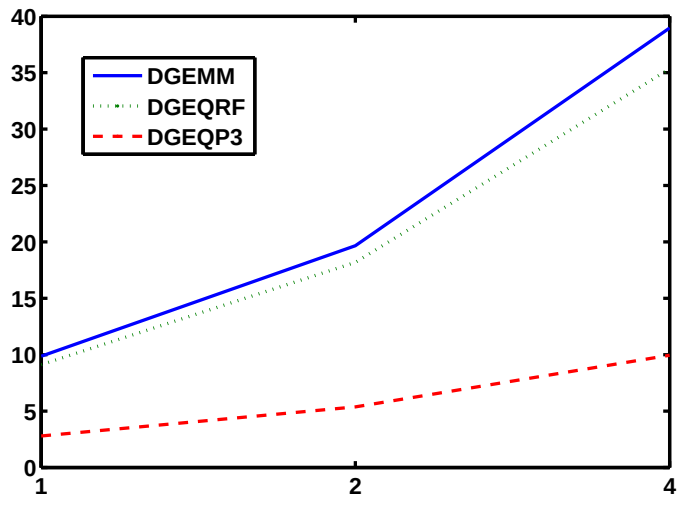
Innovative Computing Laboratory at the University of Tennessee
(J. Dongarra *et al*)

Background and motivation – Poor man's benchmark data

`DGEMM`, `DGEQRF` and `DGEQP3` of MKL 10.2 on Intel Core™ i7 Quad 2.66GHz



Speed (Gflop/s) (quad-core)



scalability ($n = 12,000$)

Message: pivoting is expensive!

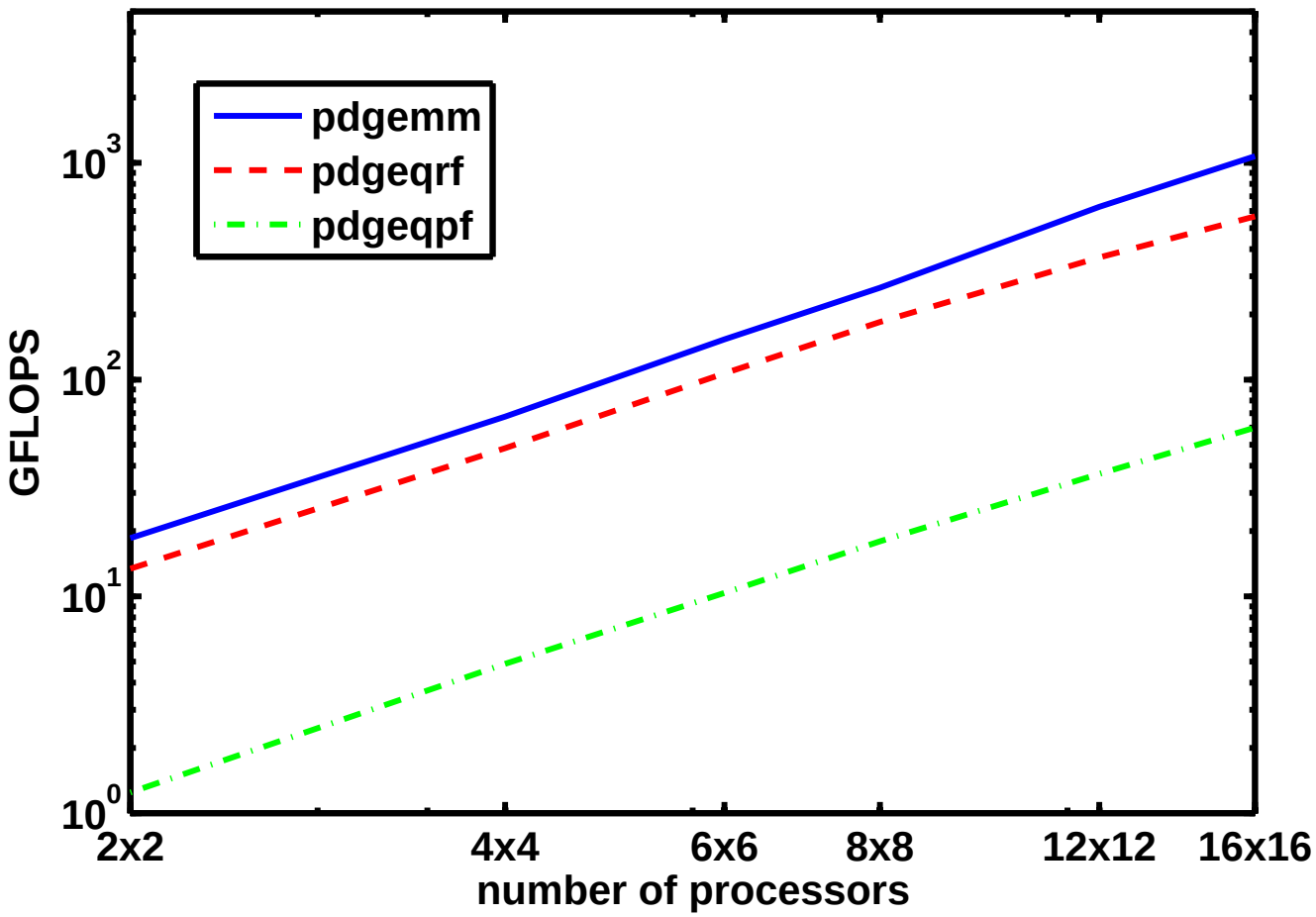
Background and motivation – Poor man's benchmark data

`PDGEMM`, `PDGEQRF` and `PDGEQPF` (pivoted QR) of PBLAS and ScaLAPACK on Cray XT4 (Franklin of NERSC):

GFlop/s ($n = 12000$)

N. Proc.	PDGEMM	PDGEQRF	PDGEQPF
2x2	18.55	13.40	1.25
4x4	67.51	48.23	4.89
6x6	153.39	106.86	10.38
8x8	265.40	184.52	17.93
12x12	626.06	365.80	36.76
16x16	1072.93	566.41	59.97

Message: pivoting is even more expensive on hybrid systems!


pdgemm
pdgeqrf
pdgeqpf
GFLOPS
10^3
10^2
10^1
10^0
2x2
4x4
6x6
8x8
12x12
16x16
number of processors

Question and the rest of my talk

Question:

- How to re-design higher-level matrix algorithms using fastest matrix opertions?

Two Case Studies:

1. Communication-reducing algorithm for Green's function calculation and application in Quantum Monte Carlo (QMC) simulations
2. Optimizing Halley's iteration for computing matrix polar decomposition and application in First Principal Molecular Dynamics (FPMD)

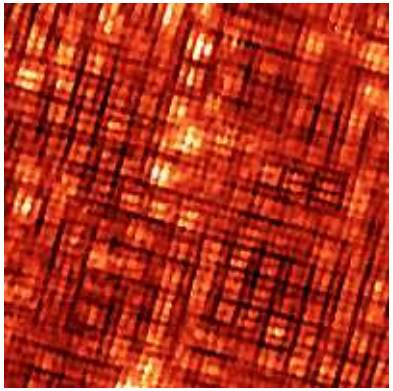
Communication reducing algorithm for Green's function calculation

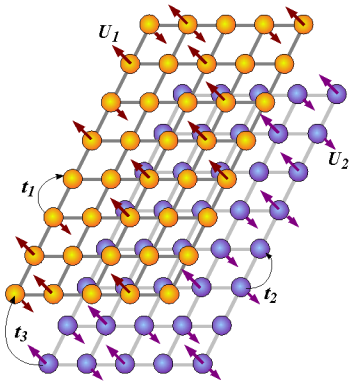
joint with

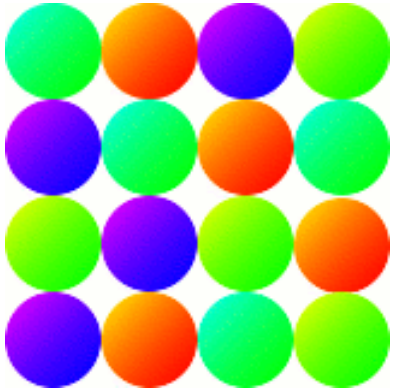
Roger Lee	National Tsinghua Univ. Taiwan
Wenbin Chen	Fudan University, China
Richard Scalettar	Physics, UC Davis

Computational Material Science

- Simulation and understanding properties of solid-state materials: magnetism, metal-insulator transition, high temperature superconductivity, ...
- Many body simulation on multi-layer lattice using Hubbard model and quantum Monte Carlo simulation
- QUEST (QUantum Electron Simulation Toolbox): Fortran 90 package for determinant (and hybrid) Monte Carlo simulations







QUEST kernel: Green's function (matrix) calcuation

$$G = (I + B_L B_{L-1} \cdots B_2 B_1)^{-1}$$

where

$$B_i = B \cdot V_i = e^{\Delta\tau K} \cdot \text{diag}(e^{\pm\lambda}),$$

- K is the *hopping matrix*, an adjance matrix of the lattice
- $\Delta\tau$ is the time discretization parameter.
- L is the time slice
- $\lambda = \cosh^{-1}(e^{U\Delta\tau/2})$.
- U is a potential energy parameter for local repulsion between electrons

Challenge: $B_L B_{L-1} \cdots B_2 B_1$ is extremely ill-conditioned!

Physicist's method [Loh et al '89, ...'05]

1. Compute an "UDT decomposition" of the product

$$B_L B_{L-1} \cdots B_2 B_1 = U_L D_L T_L$$

where D is stratified (graded)

$$D = \begin{bmatrix} \mathrm{X} & & & \\ & \mathrm{x} & & \\ & & \ddots & \\ & & & \mathrm{x} \end{bmatrix}$$

2. Compute

$$G = T_L^{-1}(U_L^T T_L^{-1} + D_L)^{-1} U_L^T.$$

The pivoted QR decomposition is used for the stratification.

There are a number of work on the product of matrices by numerical linear algebraists [Stewart, ...]

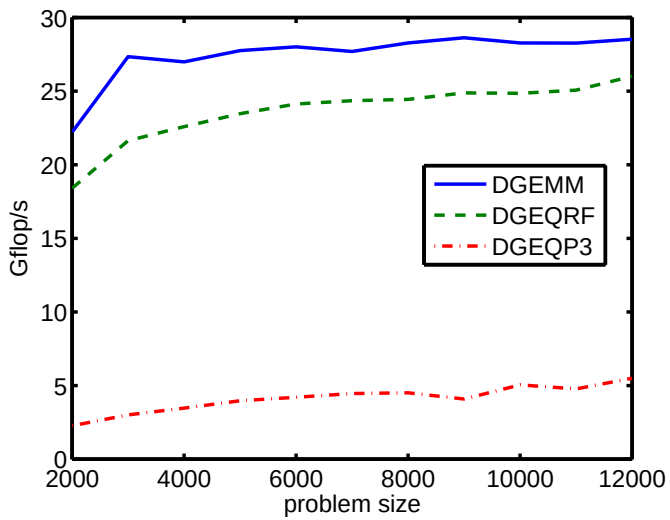
Physicist's method [Loh et al]

1. Compute pivoted QR decomposition: $B_1 = Q_1 R_1 P_1^T$
2. Set
$$\begin{aligned} U_1 &= Q_1 \\ D_1 &= \text{diag}(R_1) \\ T_1 &= D_1^{-1} R_1 P_1^T \end{aligned}$$
3. For $i = 2, 3, \cdots, L$.
   - Compute $C_i = (B_i U_{i-1}) D_{i-1}$. (*culumn scaling*)
   - Compute pivoted QR decomposition: $C_i = Q_i R_i P_i^T$.
   - Set
$$\begin{aligned} U_i &= Q_i \\ D_i &= \text{diag}(R_i) \\ T_i &= (D_i^{-1} R_i)(P_i^T T_{i-1}) \end{aligned}$$
4. Compute $G = T_L^{-1}(U_L^T T_L^{-1} + D_L)^{-1} U_L^T$.

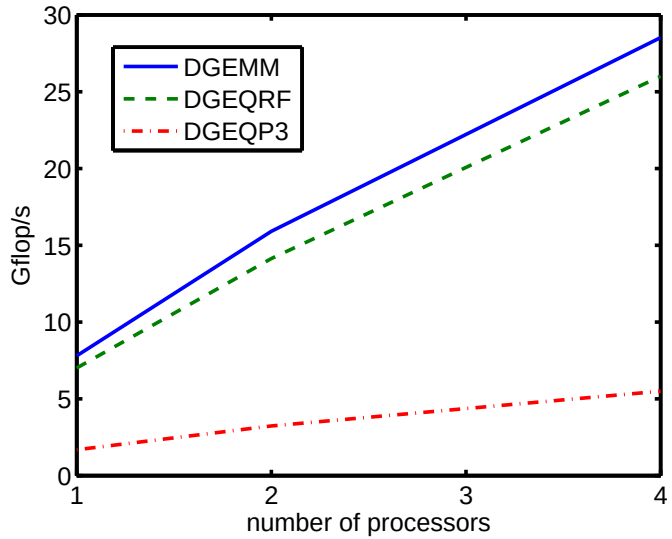
However, *pivoting* is very expensive on multicore

Intel Core™ Quad 2.4GHz,

- `DGEMM`: matrix-matrix multiplication
- `DGEQRF`: QR decomposition
- `DGEQP3`: pivoted QR decomposition



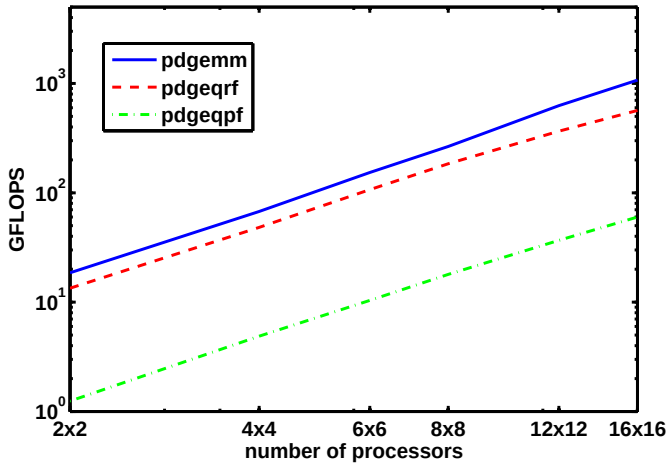
Speed (Gflop/s)



scalability

Pivoting is even more expensive on distributed systems

Cray XT4 (Franklin of NERSC)



SOF algorithm - without the pivoting

Structured orthogonal factorization (SOF) algorithm

1. Set $M_2 = I$, $A_2 = B_1$
2. For $i = 2, 3, \cdots, L$
   (a) compute QR decomposition:

$$\begin{bmatrix} M_i \\ -B_i \end{bmatrix} = \begin{bmatrix} Q_{11}^{(i)} & Q_{12}^{(i)} \\ Q_{21}^{(i)} & Q_{22}^{(i)} \end{bmatrix} \begin{bmatrix} R_i \\ 0 \end{bmatrix}$$

   (b) update

$$\begin{bmatrix} A_{i+1} & M_{i+1} \end{bmatrix} = \begin{bmatrix} (Q_{12}^{(i)})^T & (Q_{22}^{(i)})^T \end{bmatrix} \begin{bmatrix} A^{(i)} \\ I \end{bmatrix}$$

3. Compute $G = (M_L + A_L)^{-1} M_L$

Computational kernel: QR decomposition *without pivoting*.
Price: SOF takes about $3\times$ more flops than Physicist' method.

SOF algorithm – motiviation and correctness

- Motiviation: Connection betweenthe Green's function and the inverse of the following block p-cyclic matrix:

$$\begin{aligned} G &= (I + B_L B_{L-1} \cdots B_2 B_1)^{-1} \\ &= (L, L)\text{-subblock of } M^{-1} \end{aligned}$$

where

$$M = \begin{bmatrix} I & & & & B_1 \\ -B_2 & I & & & \\ & -B_3 & I & & \\ & & \ddots & \ddots & \\ & & & -B_L & I \end{bmatrix}$$

- Correctness: by direct verification

SOF algorithm – correctness

By the QR decomposition, we have

$$\left(Q_{12}^{(i)}\right)^T M_i - \left(Q_{22}^{(i)}\right)^T B_i = 0$$

Since $M_2 = I$ and $M_i = \left(Q_{22}^{(i-1)}\right)^T$ for $i > 2$, the above equations can be rewritten as

$$\begin{cases} \left(Q_{12}^{(2)}\right)^T = \left(Q_{22}^{(2)}\right)^T B_2 \\ \left(Q_{12}^{(i)}\right)^T \left(Q_{22}^{(i-1)}\right)^T = \left(Q_{22}^{(i)}\right)^T B_i \quad \text{for} \quad i \geq 3 \end{cases}$$

Note that $Q_{22}^{(i)}$ are invertible,

$$\begin{cases} \left(Q_{12}^{(2)}\right)^T = \left(Q_{22}^{(2)}\right)^T B_2 \\ \left(Q_{12}^{(i)}\right)^T = \left(Q_{22}^{(i)}\right)^T B_i \left(Q_{22}^{(i-1)}\right)^{-T} \quad \text{for} \quad i \geq 3 \end{cases}$$

Hence

$$\left(Q_{12}^{(L)}\right)^T \cdots \left(Q_{12}^{(3)}\right)^T \left(Q_{12}^{(2)}\right)^T = \left(Q_{22}^{(L)}\right)^T B_L \cdots B_3 B_2.$$

By steps 1 and 2(b) of SOF,

$$A_L = \left(Q_{12}^{(L)}\right)^T \cdots \left(Q_{12}^{(3)}\right)^T \left(Q_{12}^{(2)}\right)^T B_1.$$

Hence

$$A_L = \left(Q_{22}^{(L)}\right)^T B_L \cdots B_3 B_2 B_1.$$

In summary,

$$\begin{aligned}(M_L + A_L)^{-1} M_L &= \left[\left(Q_{22}^{(L)}\right)^T + \left(Q_{22}^{(L)}\right)^T B_L \cdots B_2 B_1\right]^{-1} \left(Q_{22}^{(L)}\right)^T \\ &= (I + B_L \cdots B_2 B_1)^{-1} = G.\end{aligned}$$

SOF algorithm – Implementation

- QR decomposition in LAPACK:
  1. `DGEQRF`: compute the QR decomposition, Q is stored implicitly
  2. `DORGQR`: forming the full Q-factor explicitly

$$Q = I - VTV^T,$$

T is an $n \times n$ upper triangular, and V is $2n \times n$ (`DLARFT`).

SOF algorithm – Implementation 1

- Call `DGEQRF` for QR decomposition
- explicitly form the full Q-factor by `DORGQR`.

SOF algorithm – Implementation 2

Modify `DORGQR` to compute $Q_{12}^{(i)}$ and $Q_{22}^{(i)}$ directly:

- Let V be comformally partitioned

$$V = \begin{bmatrix} V_u \\ V_d \end{bmatrix}, \tag{1}$$

- Then the right half of the Q-factor can be computed by

$$\begin{aligned} Q_{12}^{(i)} &= -V_u T V_d^T \\ Q_{22}^{(i)} &= I - V_d T V_d^T. \end{aligned}$$

- Note that V_u is a lower triangular matrix, used to reduce the cost of matrix multiplication.

SOF algorithm – Implementation 3

Modify `DGEQRF` to compute the T matrix recursively.

- If
$$T = \begin{bmatrix} T_{11} & T_{12} & \cdots & T_{1b} \\ & T_{22} & & T_{2b} \\ & & \ddots & \vdots \\ & & & T_{bb} \end{bmatrix},$$
`DGEQRF` produces diagonal blocks T_{ii} in order explicitly.
- Consider two consecutive block Householder transformations H_1 and H_2. Then
$$\begin{aligned} H_1 H_2 &= (I - V_1 T_1 V_1^T)(I - V_2 T_2 V_2^T) \\ &= I - (V_1 \; V_2) \begin{bmatrix} T_1 & -T_1 V_1^T V_2 T_2 \\ 0 & T_2 \end{bmatrix} \begin{bmatrix} V_1^T \\ V_2^T \end{bmatrix}, \end{aligned}$$
- The "T" matrix of the merged Householder transformation can be computed with additional computation of $-T_1 V_1^T V_2 T_2$.

Performance – I

Green's matrix

$$G = (I + B_{96}B_{95} \cdots B_2B_1)^{-1}$$

on a 32×32 lattice ($n = 1024$)

CPU elapsed time

	1 core	2 cores	4 cores
Loh's	95.93	65.56	51.23
Impl 1	195.96	110.91	69.66
Impl 2	139.68	74.34	43.94

GFlop/s rate:

	1 core	2 cores	4 cores
Loh's	5.76	8.42	10.78
Impl 1	7.65	13.51	21.52
Impl 2	7.59	14.25	24.12

Intel Core™ i7 Quad 2.66GHz, `ifort` Fortran and MKL

Performance – II

Green's matrix

$$G = (I + B_{12}B_{11} \cdots B_2 B_1)^{-1}$$

on a 64×64 lattice ($n = 4096$)

CPU elapsed time

No. of Proc.	16 (4x4)	64 (8x8)	256 (16x16)	1024 (32x32)
Loh's	354.36	127.13	41.12	53.23
Impl 4	249.29	79.36	29.21	21.29
Impl 5	158.35	57.75	28.55	26.79

GFlop/s rate:

No. of Proc.	16 (4x4)	64 (8x8)	256 (16x16)	1024 (32x32)
Loh's	12.64	35.23	108.92	84.14
Impl 4	61.20	192.26	522.34	716.65
Impl 5	70.68	193.80	392.02	417.77

Intel Core™ i7 Quad 2.66GHz, `ifort` Fortran and MKL

Further reading - I

Green's function calculation, may appear in different form, is one of fundamental problems in nano-scale simulations. The physical background and related numerical linear algebra problems are discussed in detail in the following reference

- Z. Bai, W. Chen, R. Scalettar, I. Yamazaki, *Numerical Methods for Quantum Monte Carlo Simulations of the Hubbard Model* In "Multi-Scale Phenomena in Complex Fluids" edited by T. Y. Hou *et al*, Higher Education Press, 2009 (114 pages)

A pre-print of the reference is available at

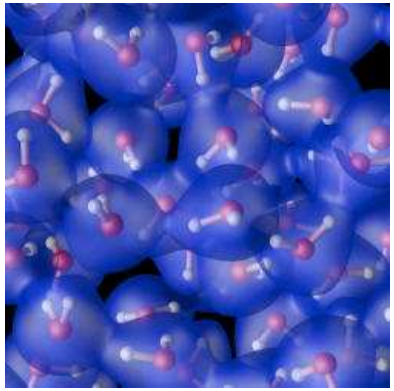
- http://www.cs.ucdavis.edu/∼bai/bcsy09.pdf

Optimized Halley's iteration for computing matrix polar decomposition

joint with
Yuji Nakatsukasa Applied Math, UC Davis
Francois Gygi Applied Science/CS, UC Davis

Electronic structure calculations

- Investigate the properties of solids, liquids, biomolecules, and nanoparticles .....
- First-principles molecular dynamics (FPMD) based on the plane-wave, pseudopotential formalism
- Qbox: a C++/MPI implementation of FPMD for massively parallel computers (F. Gygi et al)



Subspace alignment in Self-consistent iteration

Given wave functions $Y(t-1)$ and $Y(t)$ at time steps $t-1$ and t:

- Compute QR decomposition $Y(t) = \widehat{Y}(t)R$ (tall and skinny QR)
- Solve the Procrustes problem

$$\min_{Q} \|Y(t-1) - Y(t)Q\|_F$$

- Compute $\widetilde{Y} = 2\widehat{Y}(t)Q - Y(t-1)$
- Obtain trial wavefunction $\widehat{Y}(t+1)$ for the time step $t+1$ by QR: $\widetilde{Y}(t) = \widehat{Y}(t+1)R$.

Ref: [Arias et al'92, Edelman et al'99, Fattebert and Gygi'04]

Matrix polar decomposition

- Given $A \in \mathbb{C}^{n\times n}$, compute the polar decomposition

$$A = UH,$$

where

$$U^H U = I \qquad \text{and} \qquad H^H = H \geq 0$$

- Application: orthogonal procrustes problem in *subspace alignment* of SCF iteration for solving Kohn-Sham equation in electronic structure calculation:

$$\min_U \|X_i - \widehat{X}_i U\|$$

Newton's iteration

- Newton's iteration:

$$X_{k+1} = \frac{1}{2}\left(X_k + X_k^{-H}\right), \quad X_0 = A.$$

- $X_k \to U$ as $k \to \infty$

  Singular values $\sigma_i(X_k) \to 1$ as $k \to \infty$

- The convergence is slow when A is ill-conditioned.

- Scaled Newton's iteration

$$X_{k+1} = \frac{1}{2}\left(\zeta_k X_k + (\zeta_k X_k)^{-H}\right), \quad X_0 = A,$$

- The inverse X_k^{-1} has to be computed explicitly. ....

Newton's iteration variant

- Scaled Newton's iteration variant (SNV):

$$X_{k+1} = 2\eta_k X_k \left(I + \eta_k^2 X_k^H X_k\right)^{-1}, \quad Y_0 = \eta_0 A,$$

- Inverse-free implementation: SNV can be implemented using an QR decomposition (discussed later).
- Unfortunately, even the inverse-free implementation is not numerical stability?

  discovered by [Crudge '98] and [Byers and Xu '01], independently

Halley's iteration

- Halley's iteration

$$X_{k+1} = X_k(3I + X_k^H X_k)(I + 3X_k^H X_k)^{-1}, \quad X_0 = A.$$

- There is a family of iterations based on rational function approximation
- **Theorem**. $X_k \to U$ and the asymptotic rate is cubic.
- The initial steps could be *still* slow

Consider the 2×2 matrix: $X_0 = A = \begin{bmatrix} 1 & \\ & 10^{-10} \end{bmatrix}$.

The kth Halley's iterate is given by

$$X_k = \begin{bmatrix} 1 & \\ & x_k \end{bmatrix}, \quad x_k = \frac{x_{k-1}(3 + x_{k-1}^2)}{1 + 3x_{k-1}^2}.$$

It takes 24 iterations to satisfy $\|1 - X_{24}\| < 10^{-16}$.

Dynamically weighted Halley's iteration

- Dynamically weighted Halley's (DWH) iteration

$$X_{k+1} = X_k(a_k I + b_k X_k^H X_k)(I + c_k X_k^H X_k)^{-1},$$

where $X_0 = A/\alpha$ and $\alpha \geq ||A||_2$.

- Question: how to dynamically choose a_k, b_k and c_k to optimize the convergence rate?

How to dynamically determine a_k, b_k and c_k? - first step

- Let l_0 be chosen such that
$$[\sigma_{\min}(X_0), \sigma_{\max}(X_0)] \subseteq [l_0, 1] \subset (0, 1].$$
- $\sigma_i(X_1) = g_0(\sigma_i(X_0))$, and
$$[\sigma_{\min}(X_1), \sigma_{\max}(X_1)] \subseteq [\min g_0(x), \max g_0(x)].$$
where
$$g_0(x) = x\frac{a_0 + b_0 x^2}{1 + c_0 x^2}.$$
- Rational optimization problem: find a_0, b_0, c_0 such that
(1) $0 < g_0(x) \leq 1$ for $l_0 < x \leq 1$.
(2) the max-min property $\max\limits_{a_0,b_0,c_0} \left\{ \min\limits_{l_0 \leq x \leq 1} g_0(x) \right\}$.
- Update $l_1 = \min_{l_0 \leq x \leq 1} g_0(x)$, such that
$$[\sigma_{\min}(X_1), \sigma_{\max}(X_1)] \subseteq [l_1, 1] \subset (0, 1].$$

How to dynamically determine a_k, b_k and c_k? – general case

- At the $(k+1)$st iteration, we have

$$[\sigma_{\min}(X_k), \sigma_{\max}(X_k)] \subseteq [l_k, 1] \subset (0, 1].$$

and

$$[\sigma_{\min}(X_{k+1}), \sigma_{\max}(X_{k+1})] \subseteq \left[\min_{l_k \le x \le 1} g_k(x), \max_{l_k \le x \le 1} g_k(x)\right],$$

where

$$g_k(x) = x\frac{a_k + b_k x^2}{1 + c_k x^2}.$$

- Rational optimization problem: find a_k, b_k and c_k such that:
  (1) $0 < g_k(x) \le 1 \quad \text{for} \quad l_k < x \le 1.$
  (2) the max-min property $\max_{a_k, b_k, c_k} \left\{ \min_{l_k \le x \le 1} g_k(x) \right\}.$
- Update $l_{k+1} = \min_{l_k \le x \le 1} g_k(x)$

Solving rational optimization problem

- For the special three-parameter case, we have developed a direct method to obtain analytic expressions for a_k, b_k and c_k

  – *tedious, but doable*

- In general, for high-order Halley's iteration we can first reformulate the problem, and then use a SDP solver, say SeDuMi or YALMIP

- Applications of rational optimization problem in shape optimization of transfer functions

  – *[Nie et al '09]*

Convergence theorem and number of of iterations

- **Theorem**. $X_k \to U$ and the asymptotic rate is cubic.
- For the 2×2 example, 5 itertions such that $\|1 - X_5\| < 10^{-16}$.
- In general, we have the following theoretical prediction for the number of iterations

$\kappa(A)$	10^1	10^2	10^5	10^8	10^{10}	10^{12}	10^{16}
SN	5	6	7	8	8	9	9
DWH	3	4	5	5	5	5	6

Inverse-free implementation

- **Theorem.** Given the QR decomposition

$$\begin{bmatrix} \eta A \\ I \end{bmatrix} = \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} R.$$

Then

$$Q_1 Q_2^H = \eta A (I + \eta^2 A^H A)^{-1}.$$

[Zha & Zhang'96, Higham'07]

Inverse-free implementation - DWH

- Dynamically weighted Halley's (DWH) iteration

$$
\begin{aligned}
X_{k+1} &= X_k(a_k I + b_k X_k^H X_k)(I + c_k X_k^H X_k)^{-1} \\
&= \frac{b_k}{c_k} X_k + (a_k - \frac{b_k}{c_k}) \underbrace{X_k(I + c_k X_k^H X_k)^{-1}}
\end{aligned}
$$

- QR based implementation of DHW:

$$
\begin{cases}
\begin{bmatrix} \sqrt{c_k} X_k \\ I \end{bmatrix} = \begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} R, \\
X_{k+1} = \frac{b_k}{c_k} X_k + \frac{1}{\sqrt{c_k}} \left(a_k - \frac{b_k}{c_k} \right) Q_1 Q_2^H,
\end{cases}
$$

Example: verification of theoretical iteration count

Diagonal matrices A. Iter numbers and $\|X_k - I\|_F$

$\kappa(A)$	10	10^2	10^5	10^{10}	10^{15}	10^{20}
Halley	5	7	14	24	35	45
Gander	6	7	9	14	18	24
DWH	4	4	5	5	6	6
Halley	4.7e-16	5.4e-16	2.4e-16	1.1e-16	1.0e-16	1.1e-16
Gander	7.6e-16	7.5e-16	8.0e-16	7.4e-16	8.0e-16	6.4e-16
DWH	4.9e-16	3.8e-16	3.1e-16	5.7e-16	6.6e-16	5.4e-16

Example: Q-DWH and scaled Newton's iteration

Random matrices with varying condition numbers:

kappa		10^2		10^8		10^{15}	
		min	max	min	max	min	max
iter	Q-DWH	4	4	5	5	5	6
	SN	6	6	8	8	8	9
res	Q-DWH	5.3e-16	7.3e-16	4.7e-16	8.7e-16	4.7e-16	8.4e-16
	SN	7.4e-16	9.8e-16	7.2e-16	1.0e-15	6.0e-16	1.1e-15

Multicore implementation is progress.

Further reading - II

The inverse-free implementation of the Halley's iteration is described in the following paper

- Y. Nakatsukasa, Z. Bai and F. Gygi, *Optimizing Halley's iteration for computing the matrix polar decomposition*, submitted, 2009

It is available on our class website.

Concluding remarks

1. Our case studies illustrate the ideas of designing communication-reducing numerical linear algebra algorithms to balance flops, communication, flops, stability and optimize the performance on modern computer architecture.
2. Hardware trend means the time has come to do!
3. Lots of prior work in numerical linear algebra community, but more open problems
4. Why stops at numerical linear algebra?