A communication-avoiding parallel algorithm for the symmetric eigenvalue problem

Edgar Solomonik[1], Grey Ballard[2], James Demmel[3], Torsten Hoefler[4]

(1) Department of Computer Science, University of Illinois at Urbana-Champaign
(2) Department of Computer Science, Wake Forest University
(3) Department of Computer Science and Department of Mathematics, University of California, Berkeley
(4) Department of Computer Science, ETH Zurich

Symposium for Parallel Architectures and Algorithms (SPAA), 2017

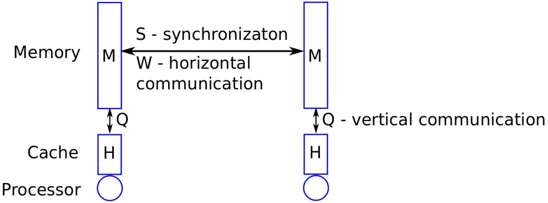
July 24, 2017

L P N A @CS@Illinois

Beyond computational complexity

Algorithms should minimize communication, not just computation

- communication and synchronization cost more energy than flops
- two types of communication (data movement):



  - vertical (intranode memory–cache)
  - horizontal (internode network transfers)
- parallel algorithm design involves tradeoffs: computation vs communication vs synchronization
- parameterized algorithms provide optimality and flexibility

Cost model for parallel algorithms

We use the Bulk Synchronous Parallel (BSP) model (L.G. Valiant 1990)

- execution is subdivided into S supersteps, each associated with a global synchronization (cost α)
- at the start of each superstep, processors interchange messages, then they perform local computation
- if the maximum amount of data sent or received by any process is m_i (and work done is f_i) at superstep i then the BSP time is

$$T = \sum_{i=1}^{S} \alpha + m_i \cdot \beta + f_i \cdot \gamma = O(S \cdot \alpha + W \cdot \beta + F \cdot \gamma)$$

We additionally consider vertical communication cost

- F – computation cost (local computation)
- Q – vertical communication cost (memory–cache traffic)
- W – horizontal communication cost (interprocessor communication)
- S – synchronization cost (number of supersteps)

Symmetric eigenvalue problem

Given a dense symmetric matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ find diagonal matrix $\boldsymbol{D}$ so

$$\boldsymbol{AX} = \boldsymbol{XD}$$

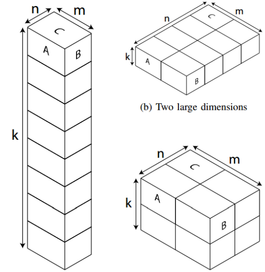
where $\boldsymbol{X}$ is an orthogonal matrix composed of eigenvectors of $\boldsymbol{A}$

- diagonalization – reduction of $\boldsymbol{A}$ to diagonal matrix $\boldsymbol{D}$
- computing the SVD has very similar computational structure
- we focus on tridiagonalization (bidiagonalization for SVD), from which standard approaches (e.g. MRRR, see Dhillon, Parlett, Vömel, 2006) can be used
- core building blocks:
  - matrix multiplication
  - QR factorization
- QR, SVD, diagonalization of lare matrices are needed for appications in scientific computing, data analysis and beyond

Communication complexity of matrix multiplication

Multiplication of $\boldsymbol{A} \in \mathbb{R}^{m \times k}$ and $\boldsymbol{B} \in \mathbb{R}^{k \times n}$ can be done in $O(1)$ supersteps with communication cost $W = O\Big(\big(\frac{mnk}{p}\big)^{2/3}\Big)$ provided sufficiently memory and sufficiently large p

- when $m = n = k$, 3D blocking gets $O(p^{1/6})$ improvement over 2D[1]
- when m, n, k are unequal, need appropriate processor grid[2]



(a) One large dimension

(b) Two large dimensions

(c) Three large dimensions

[1] J. Berntsen, Par. Comp., 1989; A. Aggarwal, A. Chandra, M. Snir, TCS, 1990; R.C. Agarwal, S.M. Balle, F.G. Gustavson, M. Joshi, P. Palkar, IBM, 1995; F.W. McColl, A. Tiskin, Algorithmica, 1999; ...

[2] J. Demmel, D. Eliahu, A. Fox, S. Kamil, B. Lipshitz, O. Schwartz, O. Spillinger 2013

Bandwidth-efficient QR and diagonalization

Goal: achieve the same communication complexity for QR and diagonalization as for matrix multiplication

- synchronization complexity expected to be higher

$$W \cdot S = \Omega(n^2)$$

  product of communication and synchronization cost must be greater than the square of the number of columns[3]

- general strategy
  1. use communication-efficient matrix-multiplication for QR
  2. use communication-efficient QR for diagonalization

[3] E.S., E. Carson, N. Knight, J. Demmel, SPAA 2014 (TOPC 2016)

Communication complexity of dense matrix kernels

For $n \times n$ Cholesky with p processors, optimal parallel schedules attain

$$F = O(n^3/p), \quad W = O(n^2/p^\delta), \quad S = O(p^\delta)$$

for any $\delta = [1/2, 2/3]$.
Achieving similar costs for LU, QR, and the symmetric eigenvalue problem requires some algorithmic tweaks.

triangular solve	square TRSM ✓[4]	rectangular TRSM ✓[5]
LU with pivoting	pairwise pivoting ✓[6]	tournament pivoting ✓[7]
QR factorization	Givens on square ✓[3]	Householder on rect. ✓[8]
SVD (sym. eig.)	singular values only ✓[8]	singular vectors X

✓ means costs attained (synchronization within polylogarithmic factors).

[4] B. Lipshitz, MS thesis 2013
[5] T. Wicky, E.S., T. Hoefler, IPDPS 2017
[6] A. Tiskin, FGCS 2007
[7] E.S., J. Demmel, EuroPar 2011
[8] E.S., G. Ballard, T. Hoefler, J. Demmel, SPAA 2017

QR factorization of tall-and-skinny matrices

Consider the reduced factorization $\boldsymbol{A} = \boldsymbol{QR}$ with $\boldsymbol{A}, \boldsymbol{Q} \in \mathbb{R}^{m\times n}$ and $\boldsymbol{R} \in \mathbb{R}^{n\times n}$ when $m \gg n$ (in particular $m \geq np$)

- $\boldsymbol{A}$ is tall-and-skinny, each processor owns a block of rows
- Householder-QR requires $S = \Theta(n)$ supersteps, $W = O(n^2)$
- Cholesky-QR2, TSQR, and HR-TSQR only $S = \Theta(\log(p))$ supersteps
- TSQR[9]: row-recursive divide-and-conquer, $W = O(n^2 \log(p))$

$$\begin{bmatrix} \boldsymbol{Q}_1\boldsymbol{R}_1 \\ \boldsymbol{Q}_2\boldsymbol{R}_2 \end{bmatrix} = \begin{bmatrix} \text{TSQR}(\boldsymbol{A}_1) \\ \text{TSQR}(\boldsymbol{A}_2) \end{bmatrix}, [\boldsymbol{Q}_{12}, \boldsymbol{R}] = \text{QR}\left(\begin{bmatrix} \boldsymbol{R}_1 \\ \boldsymbol{R}_2 \end{bmatrix} \right), \boldsymbol{Q} = \begin{bmatrix} \boldsymbol{Q}_1 & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{Q}_2 \end{bmatrix} \boldsymbol{Q}_{12}$$

- TSQR-HR[10]: TSQR + Householder-reconstruction, $W = O(n^2 \log(p))$
- Cholesky-QR2[11]: stable so long as $\kappa(\boldsymbol{A}) \leq 1/\sqrt{\epsilon}$, achieves $W = O(n^2)$

---

[9] J. Demmel, L. Grigori, M. Hoemmen, J. Langou 2012

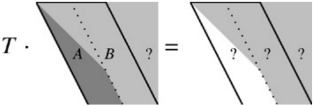
[10] G. Ballard, J. Demmel, L. Grigori, M. Jacquelin, H.-D. Nguyen, E.S. 2014

[11] Y. Yamamoto, Y. Nakatsukasa, Y. Yanagisawa, T. Fukaya 2015

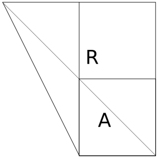
QR factorization of square matrices

Square matrix QR algorithms generally use 1D QR for panel factorization

- algorithms in ScaLAPACK, Elemental, DPLASMA use 2D layout, generally achieve $W = O(n^2/\sqrt{p})$ cost
- Tiskin's 3D QR algorithm[12] achieves $W = O(n^2/p^{2/3})$ communication



- however, requires slanted-panel matrix embedding



which is highly inefficient for rectangular (tall-and-skinny) matrices

[12] A. Tiskin 2007, "Communication-efficient generic pairwise elimination"

Communication-avoiding rectangular QR

For $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ existing algorithms are optimal when $m = n$ and $m \gg n$

- cases with $n < m < np$ underdetermined equations are important
- new algorithm
  - subdivide p processors into m/n groups of pn/m processors
  - perform row-recursive QR (TSQR) with tree of height $\log_2(m/n)$
  - compute each tree-node elimination $\text{QR}\left(\begin{bmatrix}\boldsymbol{R_1}\\ \boldsymbol{R_2}\end{bmatrix}\right)$ using Tiskin's QR with pn/m or more processors
- note: interleaving rows of $\boldsymbol{R_1}$ and $\boldsymbol{R_2}$ gives a slanted panel!
- obtains ideal communication cost for any m, n, generally

$$W = O\left(\left(\frac{mn^2}{p}\right)^{2/3}\right)$$

Tridiagonalization

Reducing the symmetric matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$ to a tridiagonal matrix

$$\boldsymbol{T} = \boldsymbol{Q}^T \boldsymbol{A} \boldsymbol{Q}$$

via a two-sided orthogonal transformation is most costly in diagonalization

- can be done by successive column QR factorizations

$$\boldsymbol{T} = \underbrace{\boldsymbol{Q}_1^T \cdots \boldsymbol{Q}_n^T}_{\boldsymbol{Q}^T} \boldsymbol{A} \underbrace{\boldsymbol{Q}_1 \cdots \boldsymbol{Q}_n}_{\boldsymbol{Q}}$$

- two-sided updates harder to manage than one-sided
- can use n/b QRs on panels of b columns to go to band-width $b+1$
- $b = 1$ gives direct tridiagonalization

Multi-stage tridiagonalization

Writing the orthogonal transformation in Householder form, we get

$$\underbrace{(\boldsymbol{I} - \boldsymbol{U}\boldsymbol{T}\boldsymbol{U}^T)^T}_{\boldsymbol{Q}^T} \boldsymbol{A} \underbrace{(\boldsymbol{I} - \boldsymbol{U}\boldsymbol{T}\boldsymbol{U}^T)}_{\boldsymbol{Q}} = \boldsymbol{A} - \boldsymbol{U}\boldsymbol{V}^T - \boldsymbol{V}\boldsymbol{U}^T$$

where $\boldsymbol{U}$ are Householder vectors and $\boldsymbol{V}$ is

$$\boldsymbol{V}^T = \boldsymbol{T}\boldsymbol{U}^T + \frac{1}{2}\boldsymbol{T}^T\boldsymbol{U}^T \underbrace{\boldsymbol{A}\boldsymbol{U}}_{\text{challenge}} \boldsymbol{T}\boldsymbol{U}^T$$

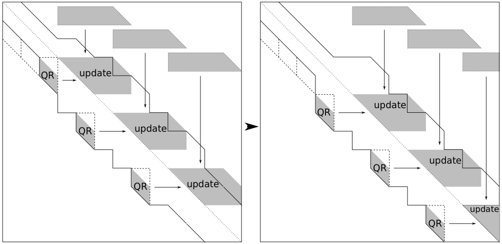
- when performing two-sided updates, computing $\boldsymbol{AU}$ dominates cost
- if $b = 1$, $\boldsymbol{U}$ is a column-vector, and $\boldsymbol{AU}$ is dominated by vertical communication cost (moving $\boldsymbol{A}$ between memory and cache)
- idea: reduce to banded matrix ($b \gg 1$) first[13]

[13] T. Auckenthaler, H.-J. Bungartz, T. Huckle, L. Krämer, B. Lang, P. Willems 2011

Successive band reduction (SBR)

After reducing to a banded matrix, we need to transform the banded matrix to a tridiagonal one

- fewer nonzeros lead to lower computational cost, $F = O(n^2b/p)$
- however, transformations introduce fill/bulges
- bulges must be chased down the band[14]



- communication- and synchronization-efficient 1D SBR algorithm known for small band-width[15]

---

[14] B. Lang 1993; C. Bischof, B. Lang, X. Sun 2000

[15] G. Ballard, J. Demmel, N. Knight 2012

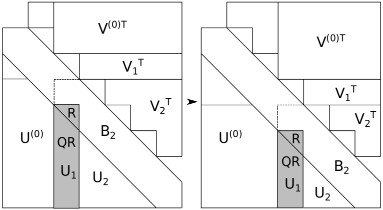
Communication-efficient eigenvalue computation

Previous work (start-of-the-art): two-stage tridiagonalization

- implemented in ELPA, can outperform ScaLAPACK[16]
- with $n = n/\sqrt{p}$, 1D SBR gives $W = O(n^2/\sqrt{p})$, $S = O(\sqrt{p}\log^2(p))$[17]

We show the benefits of many-stage tridiagonalization

- use $\Theta(\log(p))$ intermediate band-widths to achieve $W = O(n^2/p^{2/3})$
- leverage communication-efficient rectangular QR with processor groups



- 3D SBR (each QR and matrix multiplication update parallelized)

[16] T. Auckenthaler, H.-J. Bungartz, T. Huckle, L. Krämer, B. Lang, P. Willems 2011

[17] G. Ballard, J. Demmel, N. Knight 2012

Symmetric eigensolver results summary

Algorithm	W	Q	S
ScaLAPACK	$n^2/\sqrt{p}$	n^3/p	$n\log(p)$
ELPA	$n^2/\sqrt{p}$	-	$n\log(p)$
two-stage + 1D-SBR	$n^2/\sqrt{p}$	$n^2\log(n)/\sqrt{p}$	$\sqrt{p}(\log^2(p)+\log(n))$
many-stage	$n^2/p^{2/3}$	$n^2\log p/p^{2/3}$	$p^{2/3}\log^2 p$

- costs are asymptotic (same computational cost F for eigenvalues)
- W – horizontal (interprocessor) communication
- Q – vertical (memory–cache) communication, excluding $W + F/\sqrt{H}$
- S – synchronization cost (number of supersteps)

Conclusion

Summary of contributions

- communication-efficient QR factorization algorithm
  - optimal communication cost for any matrix dimensions
  - variants that trade-off some accuracy guarantees for performance
- communication-efficient symmetric eigensolver algorithm
  - reduce matrix to successively smaller band-width
  - uses concurrent executions of 3D matrix multiplication and 3D QR

Practical implications

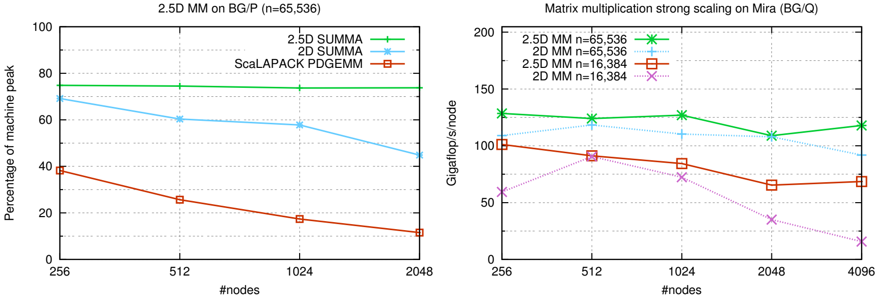
- ELPA demonstrated efficacy of two-stage approach, our work motivates 3+ stages
- partial parallel implementation is competitive but no speed-up

Future work

- back-transformations to compute eigenvectors in less computational complexity than $F = O(n^3 \log(p)/p)$
- QR with column pivoting / low-rank SVD

Backup slides

Communication-efficient matrix multiplication



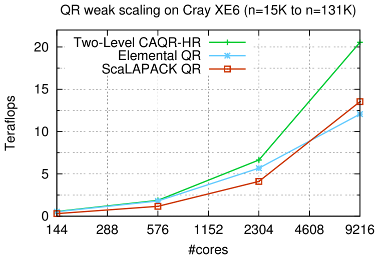
12X speed-up, 95% reduction in comm. for $n = 8\text{K}$ on 16K nodes of BG/P

Communication-efficient QR factorization

- Householder form can be reconstructed quickly from TSQR[18]

$$\boldsymbol{Q} = \boldsymbol{I} - \boldsymbol{Y}\boldsymbol{T}\boldsymbol{Y}^T \quad \Rightarrow \quad \text{LU}(\boldsymbol{I} - \boldsymbol{Q}) \to (\boldsymbol{Y}, \boldsymbol{T}\boldsymbol{Y}^T)$$

- Householder aggregation yields performance improvements

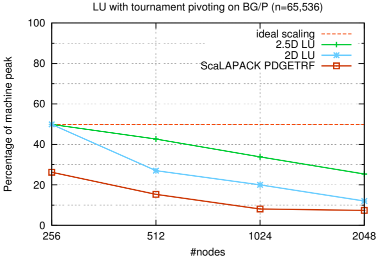


[18] Ballard, Demmel, Grigori, Jacquelin, Nguyen, S., IPDPS, 2014

Communication-efficient LU factorization

For any $c \in [1, p^{1/3}]$, use cn^2/p memory per processor and obtain

$$W_{\text{LU}} = O(n^2/\sqrt{cp}), \qquad S_{\text{LU}} = O(\sqrt{cp})$$



- LU with pairwise pivoting[19] extended to tournament pivoting[20]
- first implementation of a communication-optimal LU algorithm[11]

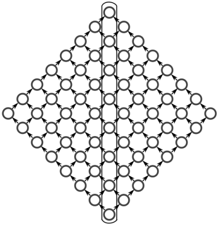
[19] Tiskin, FGCS, 2007

[20] S., Demmel, Euro-Par, 2011

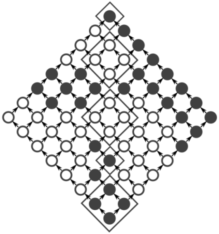
Tradeoffs in the diamond DAG

Computation vs synchronization tradeoff for the $n \times n$ diamond DAG,[21]

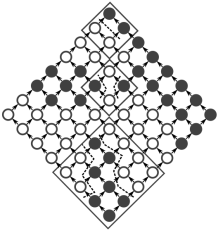
$$F \cdot S = \Omega(n^2)$$



Dependency chain P



Monochrome dependency intervals



Multicolored dependency intervals

We generalize this idea[22]

- additionally consider horizontal communication
- allow arbitrary (polynomial or exponential) interval expansion

[21] Papadimitriou, Ullman, SIAM JC, 1987

[22] S., Carson, Knight, Demmel, SPAA 2014 (extended version, JPDC 2016)

Tradeoffs involving synchronization

We apply tradeoff lower bounds to dense linear algebra algorithms, represented via dependency hypergraphs:[23]
For triangular solve with an $n \times n$ matrix,

$$F_{\text{TRSV}} \cdot S_{\text{TRSV}} = \Omega\left(n^2\right)$$

For Cholesky of an $n \times n$ matrix,

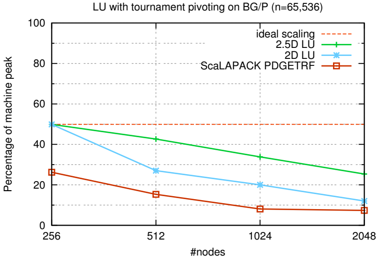
$$F_{\text{CHOL}} \cdot S_{\text{CHOL}}^2 = \Omega\left(n^3\right) \qquad W_{\text{CHOL}} \cdot S_{\text{CHOL}} = \Omega\left(n^2\right)$$

[23] S., Carson, Knight, Demmel, SPAA 2014 (extended version, JPDC 2016)

Communication-efficient LU factorization

For any $c \in [1, p^{1/3}]$, use cn^2/p memory per processor and obtain

$$W_{\text{LU}} = O(n^2/\sqrt{cp}), \qquad S_{\text{LU}} = O(\sqrt{cp})$$



- LU with pairwise pivoting[24] extended to tournament pivoting[25]
- first implementation of a communication-optimal LU algorithm[10]

[24] Tiskin, FGCS, 2007
[25] S., Demmel, Euro-Par, 2011