Communication-avoiding factorization algorithms

Edgar Solomonik

Department of Computer Science, University of Illinois at Urbana-Champaign

Conference on Fast Direct Solvers, Purdue University

November 10, 2018

L P N A @CS@Illinois

Beyond computational complexity

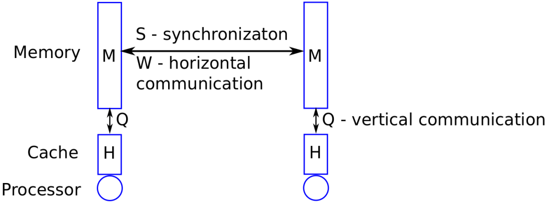
Algorithms should minimize communication, not just computation

- communication and synchronization cost more energy than flops

Beyond computational complexity

Algorithms should minimize communication, not just computation

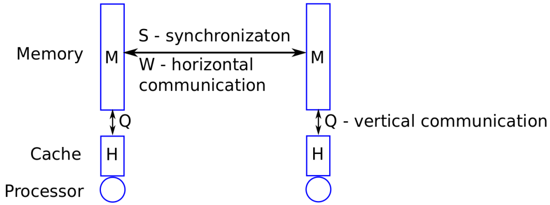
- communication and synchronization cost more energy than flops
- two types of communication (data movement):



Beyond computational complexity

Algorithms should minimize communication, not just computation

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

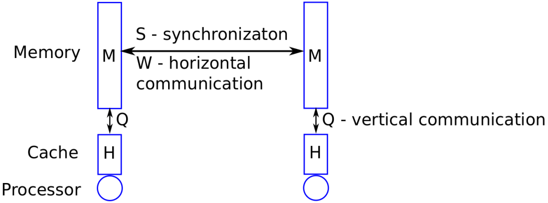


  - vertical (intranode memory–cache)

Beyond computational complexity

Algorithms should minimize communication, not just computation

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

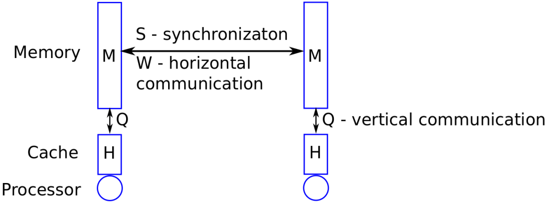


  - vertical (intranode memory–cache)
  - horizontal (internode network transfers)

Beyond computational complexity

Algorithms should minimize communication, not just computation

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



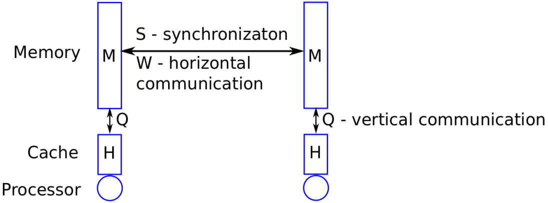
  - vertical (intranode memory–cache)
  - horizontal (internode network transfers)

- parallel algorithm design involves tradeoffs: computation vs communication vs synchronization

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)

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

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 w_i (work done is f_i and amount of memory traffic is q_i) at superstep i then the BSP time is

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

where typically $\alpha \gg \beta \gg \nu \gg \gamma$

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 w_i (work done is f_i and amount of memory traffic is q_i) at superstep i then the BSP time is

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

  where typically $\alpha \gg \beta \gg \nu \gg \gamma$
- we mention vertical communication cost only when it exceeds $Q = O(F/\sqrt{H} + W)$ where H is cache size

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 $W = O\left(\left(\frac{mnk}{p}\right)^{2/3}\right)$ provided sufficient memory and sufficiently large p

[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

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 sufficient memory and sufficiently large p

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

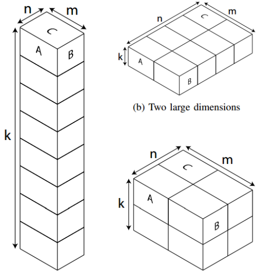
[1] J. Berntsen, Par. Comp., 1989; A. Aggarwal, A. Chandra, M. Snir, TCS, 1990; R.C. Agarwal, S.M. Balle, F.G. Gustavson, M. Joshi, P. Palkar, IBM, 1995; F.W. McColl, A. Tiskin, Algorithmica, 1999; ...

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

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 sufficient 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

Communication complexity of dense matrix kernels

For $n \times n$ Cholesky with p processors

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

given memory to store $p^{2\delta-1}$ copies of the matrix for any $\delta = [1/2, 2/3]$.

[3] B. Lipshitz, MS thesis 2013

[4] T. Wicky, E.S., T. Hoefler, IPDPS 2017

[5] A. Tiskin, FGCS 2007

[6] E.S., J. Demmel, EuroPar 2011

[7] E.S., G. Ballard, T. Hoefler, J. Demmel, SPAA 2017

Communication complexity of dense matrix kernels

For $n \times n$ Cholesky with p processors

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

given memory to store $p^{2\delta-1}$ copies of the matrix for any $\delta = [1/2, 2/3]$.

Can achieve similar costs for LU, QR, and the symmetric eigenvalue problem (modulo logarithmic factors on synchronization), but algorithmic changes (as opposed to parallel schedules) are necessary.

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

[3] B. Lipshitz, MS thesis 2013

[4] T. Wicky, E.S., T. Hoefler, IPDPS 2017

[5] A. Tiskin, FGCS 2007

[6] E.S., J. Demmel, EuroPar 2011

[7] E.S., G. Ballard, T. Hoefler, J. Demmel, SPAA 2017

Tradeoffs between costs based on dependency graphs

Definition ((ϵ, σ)-path-expander)

Graph $G = (V, E)$ is a (ϵ, σ)-**path-expander** if there exists a path $(u_1, \ldots u_n) \subset V$, such that the dependency interval $[u_i, u_{i+b}]_G$ for each i, b has size $\Theta(\sigma(b))$ and a minimum cut of size $\Omega(\epsilon(b))$.

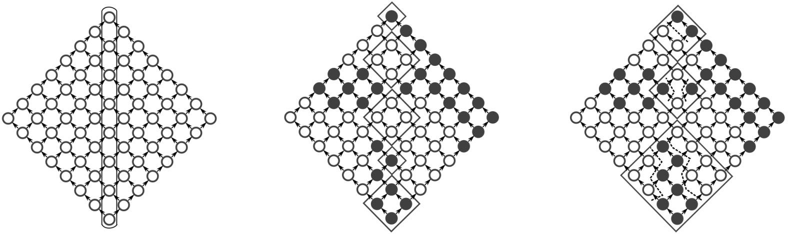
[8] C.H. Papadimitriou, J.D. Ullman, SIAM JC, 1987

[9] E.S., E. Carson, N. Knight, J. Demmel, JPDC 2017

Tradeoffs between costs based on dependency graphs

Definition ((ϵ,σ)-path-expander)

Graph $G=(V,E)$ is a (ϵ,σ)-**path-expander** if there exists a path $(u_1,\ldots u_n)\subset V$, such that the dependency interval $[u_i,u_{i+b}]_G$ for each i,b has size $\Theta(\sigma(b))$ and a minimum cut of size $\Omega(\epsilon(b))$.



Dependency chain P   Monochrome dependency intervals   Multicolored dependency intervals

- computation-synchronizaton tradeoff in diamond DAG[8]: $F\cdot S=\Omega(n^2)$
- extends to triangular solve, matrix factorization, and iterative methods[9]

[8] C.H. Papadimitriou, J.D. Ullman, SIAM JC, 1987

[9] E.S., E. Carson, N. Knight, J. Demmel, JPDC 2017

Tradeoffs between costs

Definition ((ϵ, σ)-path-expander)

Graph $G = (V, E)$ is a (ϵ, σ)-**path-expander** if there exists a path $(u_1, \ldots u_n) \subset V$, such that the dependency interval $[u_i, u_{i+b}]_G$ for each i, b has size $\Theta(\sigma(b))$ and a minimum cut of size $\Omega(\epsilon(b))$.

Theorem (Path-expander communication lower bound)

*Any parallel schedule of an algorithm with a (ϵ, σ)-**path-expander** dependency graph about a path of length n and some $b \in [1, n]$ incurs computation (F), communication (W), and synchronization (S) costs:*

$$F = \Omega\left(\sigma(b) \cdot n/b\right), \quad W = \Omega\left(\epsilon(b) \cdot n/b\right), \quad S = \Omega\left(n/b\right).$$

Tradeoffs between costs

Definition ((ϵ, σ)-path-expander)

Graph $G = (V, E)$ is a (ϵ, σ)-**path-expander** if there exists a path $(u_1, \ldots u_n) \subset V$, such that the dependency interval $[u_i, u_{i+b}]_G$ for each i, b has size $\Theta(\sigma(b))$ and a minimum cut of size $\Omega(\epsilon(b))$.

Theorem (Path-expander communication lower bound)

*Any parallel schedule of an algorithm with a (ϵ, σ)-**path-expander** dependency graph about a path of length n and some $b \in [1, n]$ incurs computation (F), communication (W), and synchronization (S) costs:*

$$F = \Omega\left(\sigma(b) \cdot n/b\right), \quad W = \Omega\left(\epsilon(b) \cdot n/b\right), \quad S = \Omega\left(n/b\right).$$

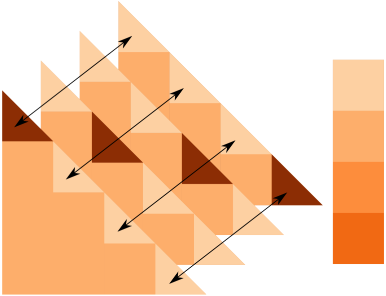
Corollary (Computation-sync. and bandwidth-sync. tradeoffs)

If $\sigma(b) = b^d$ and $\epsilon(b) = b^{d-1}$, the above theorem yields,

$$F \cdot S^{d-1} = \Omega(n^d), \;\; W \cdot S^{d-2} = \Omega(n^{d-1}).$$

New algorithms can circumvent lower bounds

For TRSM, we can achieve a lower synchronization/communication cost by performing triangular inversion on diagonal blocks

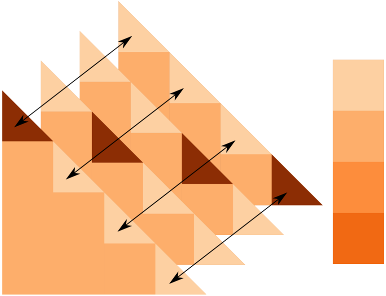


- MS thesis work by Tobias Wicky[10]

[10] T. Wicky, E.S., T. Hoefler, IPDPS 2017

New algorithms can circumvent lower bounds

For TRSM, we can achieve a lower synchronization/communication cost by performing triangular inversion on diagonal blocks

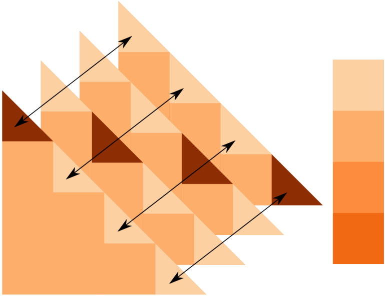


- MS thesis work by Tobias Wicky[10]
- decreases synchronization cost by $O(p^{2/3})$ on p processors with respect to known algorithms

[10] T. Wicky, E.S., T. Hoefler, IPDPS 2017

New algorithms can circumvent lower bounds

For TRSM, we can achieve a lower synchronization/communication cost by performing triangular inversion on diagonal blocks



- MS thesis work by Tobias Wicky[10]
- decreases synchronization cost by $O(p^{2/3})$ on p processors with respect to known algorithms
- optimal communication for any number of right-hand sides

[10] T. Wicky, E.S., T. Hoefler, IPDPS 2017

QR factorization of tall-and-skinny matrices

Consider the reduced factorization $\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{R}$ 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

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

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

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

[14] T. Fukaya, R. Kannan, Y. Nakatsukasa, Y. Yamamoto, Y. Yanagisawa 2018

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, $W = O(n^2)$ comm.

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

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

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

[14] T. Fukaya, R. Kannan, Y. Nakatsukasa, Y. Yamamoto, Y. Yanagisawa 2018

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, $W = O(n^2)$ comm.
- TSQR[11] row-wise divide-and-conquer, $W{=}O(n^2 \log p)$, $S{=}O(\log p)$

$$\begin{bmatrix} \boldsymbol{Q_1 R_1} \\ \boldsymbol{Q_2 R_2} \end{bmatrix} = \begin{bmatrix} \text{TSQR}(\boldsymbol{A_1}) \\ \text{TSQR}(\boldsymbol{A_2}) \end{bmatrix}, \boldsymbol{Q_{12} R} = \begin{bmatrix} \boldsymbol{R_1} \\ \boldsymbol{R_2} \end{bmatrix}, \boldsymbol{Q} = \begin{bmatrix} \boldsymbol{Q_1} & \\ & \boldsymbol{Q_2} \end{bmatrix} \boldsymbol{Q_{12}}$$

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

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

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

[14] T. Fukaya, R. Kannan, Y. Nakatsukasa, Y. Yamamoto, Y. Yanagisawa 2018

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, $W = O(n^2)$ comm.
- TSQR[11] row-wise divide-and-conquer, $W = O(n^2 \log p)$, $S = O(\log p)$

$$\begin{bmatrix} \boldsymbol{Q_1 R_1} \\ \boldsymbol{Q_2 R_2} \end{bmatrix} = \begin{bmatrix} \text{TSQR}(\boldsymbol{A_1}) \\ \text{TSQR}(\boldsymbol{A_2}) \end{bmatrix}, \boldsymbol{Q_{12} R} = \begin{bmatrix} \boldsymbol{R_1} \\ \boldsymbol{R_2} \end{bmatrix}, \boldsymbol{Q} = \begin{bmatrix} \boldsymbol{Q_1} & \\ & \boldsymbol{Q_2} \end{bmatrix} \boldsymbol{Q_{12}}$$

- TSQR-HR[12] Householder rep. $\boldsymbol{I} - \boldsymbol{YTY}$, $W = O(n^2 \log p)$, $S = O(\log p)$

[11] J. Demmel, L. Grigori, M. Hoemmen, J. Langou 2012
[12] G. Ballard, J. Demmel, L. Grigori, M. Jacquelin, H.-D. Nguyen, E.S. 2014
[13] Y. Yamamoto, Y. Nakatsukasa, Y. Yanagisawa, T. Fukaya 2015
[14] T. Fukaya, R. Kannan, Y. Nakatsukasa, Y. Yamamoto, Y. Yanagisawa 2018

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, $W = O(n^2)$ comm.
- TSQR[11] row-wise divide-and-conquer, $W{=}O(n^2 \log p)$, $S{=}O(\log p)$

$$\begin{bmatrix} \boldsymbol{Q_1 R_1} \\ \boldsymbol{Q_2 R_2} \end{bmatrix} = \begin{bmatrix} \text{TSQR}(\boldsymbol{A_1}) \\ \text{TSQR}(\boldsymbol{A_2}) \end{bmatrix}, \boldsymbol{Q_{12} R} = \begin{bmatrix} \boldsymbol{R_1} \\ \boldsymbol{R_2} \end{bmatrix}, \boldsymbol{Q} = \begin{bmatrix} \boldsymbol{Q_1} & \\ & \boldsymbol{Q_2} \end{bmatrix} \boldsymbol{Q_{12}}$$

- TSQR-HR[12] Householder rep. $\boldsymbol{I} - \boldsymbol{YTY}$, $W{=}O(n^2 \log p)$, $S{=}O(\log p)$
- Cholesky-QR2[13] stable so long as $\kappa(\boldsymbol{A}) \leq 1/\sqrt{\epsilon}$, achieves $W{=}O(n^2)$, $S{=}O(1)$, Cholesky-QR3[14] gets same and is unconditionally stable

[11] J. Demmel, L. Grigori, M. Hoemmen, J. Langou 2012
[12] G. Ballard, J. Demmel, L. Grigori, M. Jacquelin, H.-D. Nguyen, E.S. 2014
[13] Y. Yamamoto, Y. Nakatsukasa, Y. Yanagisawa, T. Fukaya 2015
[14] T. Fukaya, R. Kannan, Y. Nakatsukasa, Y. Yamamoto, Y. Yanagisawa 2018

QR factorization of square matrices

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

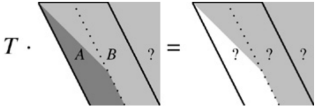
- algorithms in ScaLAPACK, Elemental, DPLASMA use 2D layout, generally achieve $W = O(n^2/\sqrt{p})$ cost

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

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[15] achieves $W = O(n^2/p^{2/3})$ communication

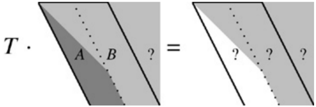


15 A. Tiskin 2007, "Communication-efficient generic pairwise elimination"

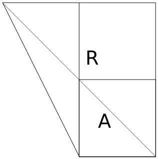
QR factorization of square matrices

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[15] 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

[15] 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

[16] E.S., G. Ballard, J. Demmel, and T. Hoefler, SPAA 2017

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[16]

[16] E.S., G. Ballard, J. Demmel, and T. Hoefler, SPAA 2017

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[16]
  - subdivide p processors into m/n groups of pn/m processors

[16] E.S., G. Ballard, J. Demmel, and T. Hoefler, SPAA 2017

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[16]
  - subdivide p processors into m/n groups of pn/m processors
  - perform row-recursive QR (TSQR) with tree of height $\log_2(m/n)$

[16] E.S., G. Ballard, J. Demmel, and T. Hoefler, SPAA 2017

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[16]
  - 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 $\boldsymbol{Q_{12}R} = \begin{bmatrix} \boldsymbol{R_1} \\ \boldsymbol{R_2} \end{bmatrix}$ using Tiskin's QR with pn/m or more processors

[16] E.S., G. Ballard, J. Demmel, and T. Hoefler, SPAA 2017

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[16]
  - 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 $\boldsymbol{Q_{12}R} = \begin{bmatrix} \boldsymbol{R_1} \\ \boldsymbol{R_2} \end{bmatrix}$ 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

[16] E.S., G. Ballard, J. Demmel, and T. Hoefler, SPAA 2017

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[16]
  - 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 $\boldsymbol{Q_{12}R} = \begin{bmatrix}\boldsymbol{R_1}\\ \boldsymbol{R_2}\end{bmatrix}$ 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

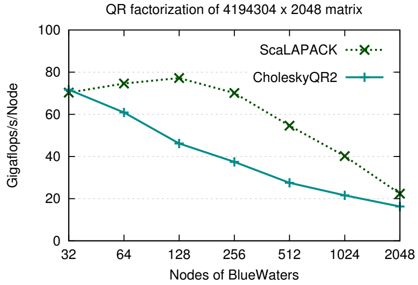
$$W = O\left(\left(\frac{mn^2}{p}\right)^{2/3}\right)$$

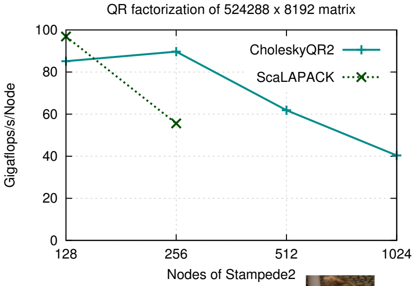
[16] E.S., G. Ballard, J. Demmel, and T. Hoefler, SPAA 2017

Cholesky-QR2 for rectangular matrices

Cholesky-QR2[17] with 3D Cholesky gives a practical 3D QR algorithm[18]

- Compute $\boldsymbol{A} = \hat{\boldsymbol{Q}}\hat{\boldsymbol{R}}$ using Cholesky-QR $\boldsymbol{A}^T\boldsymbol{A} = \hat{\boldsymbol{R}}^T\hat{\boldsymbol{R}}, \quad \hat{\boldsymbol{Q}} = \boldsymbol{A}\hat{\boldsymbol{R}}^{-1}$
- Correct approximate factorization by Cholesky-QR $\boldsymbol{Q}\bar{\boldsymbol{R}} = \hat{\boldsymbol{Q}}$, $\boldsymbol{R} = \bar{\boldsymbol{R}}\hat{\boldsymbol{R}}$
- Simple algorithm to achieve minimize comm. and sync. for any m, n, p





Analysis and implementation by PhD student Edward Hutter

[17] T. Fukaya, Y. Nakatsukasa, Y. Yanagisawa, Y. Yamamoto 2014

[18] E. Hutter, E.S. 2018

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 (eigenvalue computation, SVD similar)

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 (eigenvalue computation, SVD similar)

- can be done by successive subcolumn QR factorizations

$$\boldsymbol{T} = \underbrace{\boldsymbol{Q}_1^T \cdots \boldsymbol{Q}_{n-2}^T}_{\boldsymbol{Q}^T} \boldsymbol{A} \underbrace{\boldsymbol{Q}_1 \cdots \boldsymbol{Q}_{n-2}}_{\boldsymbol{Q}}$$

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 (eigenvalue computation, SVD similar)

- can be done by successive subcolumn QR factorizations

$$\boldsymbol{T} = \underbrace{\boldsymbol{Q}_1^T \cdots \boldsymbol{Q}_{n-2}^T}_{\boldsymbol{Q}^T} \boldsymbol{A} \underbrace{\boldsymbol{Q}_1 \cdots \boldsymbol{Q}_{n-2}}_{\boldsymbol{Q}}$$

- two-sided updates harder to parallelize than one-sided

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 (eigenvalue computation, SVD similar)

- can be done by successive subcolumn QR factorizations

$$\boldsymbol{T} = \underbrace{\boldsymbol{Q}_1^T \cdots \boldsymbol{Q}_{n-2}^T}_{\boldsymbol{Q}^T} \boldsymbol{A} \underbrace{\boldsymbol{Q}_1 \cdots \boldsymbol{Q}_{n-2}}_{\boldsymbol{Q}}$$

- two-sided updates harder to parallelize than one-sided
- each update requires a BSP superstep and reading $\boldsymbol{A}$ from memory

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 (eigenvalue computation, SVD similar)

- can be done by successive subcolumn QR factorizations

$$\boldsymbol{T} = \underbrace{\boldsymbol{Q}_1^T \cdots \boldsymbol{Q}_{n-2}^T}_{\boldsymbol{Q}^T} \boldsymbol{A} \underbrace{\boldsymbol{Q}_1 \cdots \boldsymbol{Q}_{n-2}}_{\boldsymbol{Q}}$$

- two-sided updates harder to parallelize than one-sided
- each update requires a BSP superstep and reading $\boldsymbol{A}$ from memory
- can use n/b QRs on panels of b subcolumns to go to band-width $b+1$

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 (eigenvalue computation, SVD similar)

- can be done by successive subcolumn QR factorizations

$$\boldsymbol{T} = \underbrace{\boldsymbol{Q}_1^T \cdots \boldsymbol{Q}_{n-2}^T}_{\boldsymbol{Q}^T} \boldsymbol{A} \underbrace{\boldsymbol{Q}_1 \cdots \boldsymbol{Q}_{n-2}}_{\boldsymbol{Q}}$$

- two-sided updates harder to parallelize than one-sided
- each update requires a BSP superstep and reading $\boldsymbol{A}$ from memory
- can use n/b QRs on panels of b subcolumns to go to band-width $b+1$
- $b = 1$ gives direct tridiagonalization

Successive band reduction (SBR)

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

[19] Lang 1993; Bischof, Lang, Sun 2000

[20] Ballard, Demmel, Knight 2012

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, $F = O(n^2 b/p)$

[19] Lang 1993; Bischof, Lang, Sun 2000
[20] Ballard, Demmel, Knight 2012

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, $F = O(n^2 b/p)$
- however, transformations introduce fill/bulges

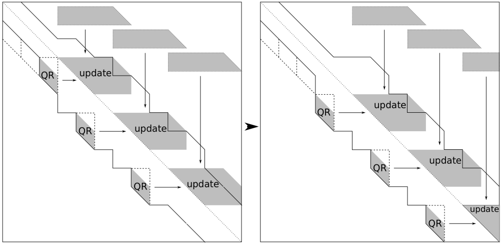
[19] Lang 1993; Bischof, Lang, Sun 2000

[20] Ballard, Demmel, Knight 2012

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, $F = O(n^2 b/p)$
- however, transformations introduce fill/bulges
- bulges must be chased down the band[19]

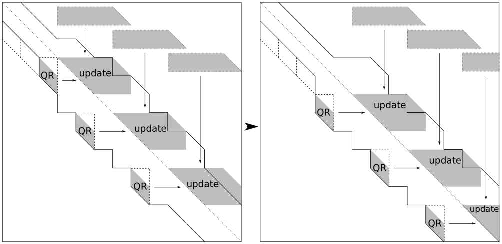


[19] Lang 1993; Bischof, Lang, Sun 2000

[20] Ballard, Demmel, Knight 2012

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[19]



- communication- and synchronization-efficient 1D SBR algorithm known for small band-width[20]

[19] Lang 1993; Bischof, Lang, Sun 2000

[20] Ballard, Demmel, Knight 2012

Communication-efficient eigenvalue computation

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

- implemented in ELPA, can outperform ScaLAPACK[21]

[21] Auckenthaler, Bungartz, Huckle, Krämer, Lang, Willems 2011

[22] Ballard, Demmel, Knight 2012

[23] E.S., G. Ballard, J. Demmel, T. Hoefler, SPAA 2017

Communication-efficient eigenvalue computation

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

- implemented in ELPA, can outperform ScaLAPACK[21]
- with $n = n/\sqrt{p}$, 1D SBR gives $W = O(n^2/\sqrt{p})$, $S = O(\sqrt{p}\log^2(p))$[22]

[21] Auckenthaler, Bungartz, Huckle, Krämer, Lang, Willems 2011

[22] Ballard, Demmel, Knight 2012

[23] E.S., G. Ballard, J. Demmel, T. Hoefler, SPAA 2017

Communication-efficient eigenvalue computation

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

- implemented in ELPA, can outperform ScaLAPACK[21]
- with $n = n/\sqrt{p}$, 1D SBR gives $W = O(n^2/\sqrt{p})$, $S = O(\sqrt{p}\log^2(p))$[22]

[21] Auckenthaler, Bungartz, Huckle, Krämer, Lang, Willems 2011

[22] Ballard, Demmel, Knight 2012

[23] E.S., G. Ballard, J. Demmel, T. Hoefler, SPAA 2017

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

- implemented in ELPA, can outperform ScaLAPACK[21]
- with $n = n/\sqrt{p}$, 1D SBR gives $W = O(n^2/\sqrt{p})$, $S = O(\sqrt{p}\log^2(p))$[22]

New results[23]: many-stage tridiagonalization

- $\Theta(\log(p))$ intermediate band-widths to achieve $W = O(n^2/p^{2/3})$

[21] Auckenthaler, Bungartz, Huckle, Krämer, Lang, Willems 2011

[22] Ballard, Demmel, Knight 2012

[23] E.S., G. Ballard, J. Demmel, T. Hoefler, SPAA 2017

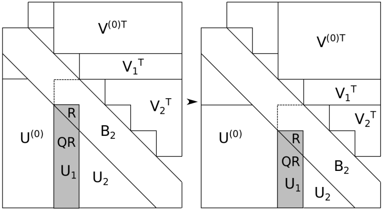
Communication-efficient eigenvalue computation

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

- implemented in ELPA, can outperform ScaLAPACK[21]
- with $n = n/\sqrt{p}$, 1D SBR gives $W = O(n^2/\sqrt{p})$, $S = O(\sqrt{p}\log^2(p))$[22]

New results[23]: many-stage tridiagonalization

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



21 Auckenthaler, Bungartz, Huckle, Krämer, Lang, Willems 2011

22 Ballard, Demmel, Knight 2012

23 E.S., G. Ballard, J. Demmel, T. Hoefler, SPAA 2017

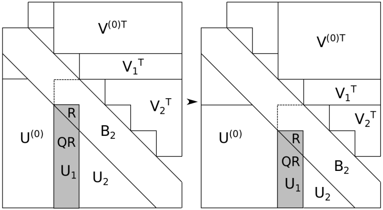
Communication-efficient eigenvalue computation

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

- implemented in ELPA, can outperform ScaLAPACK[21]
- with $n = n/\sqrt{p}$, 1D SBR gives $W = O(n^2/\sqrt{p})$, $S = O(\sqrt{p}\log^2(p))$[22]

New results[23]: many-stage tridiagonalization

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



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

---

[21] Auckenthaler, Bungartz, Huckle, Krämer, Lang, Willems 2011

[22] Ballard, Demmel, Knight 2012

[23] E.S., G. Ballard, J. Demmel, T. Hoefler, SPAA 2017

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 $W + F/\sqrt{H}$ where H is cache size
- S – synchronization cost (number of supersteps)

Conclusion

Summary of new communication avoiding algorithms

- communication-efficient QR factorization algorithm

Conclusion

Summary of new communication avoiding algorithms

- communication-efficient QR factorization algorithm
  - optimal communication cost for any matrix dimensions

Conclusion

Summary of new communication avoiding algorithms

- communication-efficient QR factorization algorithm
  - optimal communication cost for any matrix dimensions
  - variants that trade-off some accuracy guarantees for performance

Conclusion

Summary of new communication avoiding algorithms

- 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

Conclusion

Summary of new communication avoiding algorithms

- 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

Conclusion

Summary of new communication avoiding algorithms

- 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

Conclusion

Summary of new communication avoiding algorithms

- 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

Conclusion

Summary of new communication avoiding algorithms

- 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

Conclusion

Summary of new communication avoiding algorithms

- 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

Conclusion

Summary of new communication avoiding algorithms

- 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

Conclusion

Summary of new communication avoiding algorithms

- 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 $F = O(n^3 \log(p)/p)$

Conclusion

Summary of new communication avoiding algorithms

- 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 / sparse factorization

Acknowledgements

Collaborators on this work

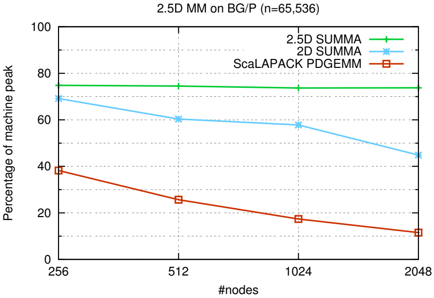
- Edward Hutter (Department of Computer Science, University of Illinois at Urbana-Champaign)
- Grey Ballard (Department of Computer Science, Wake Forest University)
- James Demmel (Department of Computer Science and Department of Mathematics, University of California, Berkeley)
- Tobias Wicky (Department of Computer Science, ETH Zurich)
- Torsten Hoefler (Department of Computer Science, ETH Zurich)
- Erin Carson (Courant Institute of Mathematical Sciences, NYU)
- Nicholas Knight (Courant Institute of Mathematical Sciences, NYU)

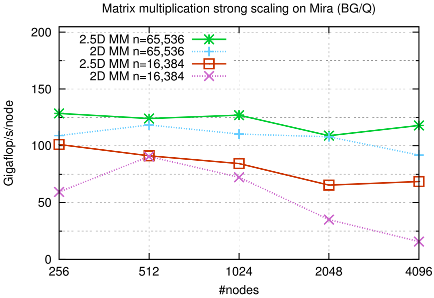
Computational resources and funding

- DOE Computational Science Graduate Fellowship
- ETH Zurich Postdoctoral Fellowship
- XSEDE/TACC (Stampede2) and NCSA (BlueWaters)

Backup slides

Communication-efficient matrix multiplication





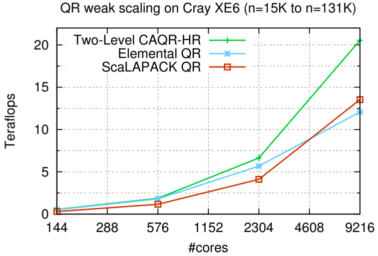
12X speed-up, 95% reduction in comm. for $n = 8$K on 16K nodes of BG/P

Communication-efficient QR factorization

- Householder form can be reconstructed quickly from TSQR[24]

$$\boldsymbol{Q} = \boldsymbol{I} - \boldsymbol{Y}\boldsymbol{T}\boldsymbol{Y}^T \qquad \Rightarrow \qquad \mathrm{LU}(\boldsymbol{I} - \boldsymbol{Q}) \rightarrow (\boldsymbol{Y}, \boldsymbol{T}\boldsymbol{Y}^T)$$

- Householder aggregation yields performance improvements

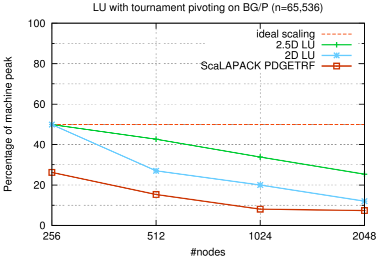


[24] Ballard, Demmel, Grigori, Jacquelin, Nguyen, S., IPDPS, 2014

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[25] extended to tournament pivoting[26]
- first implementation of a communication-optimal LU algorithm[11]

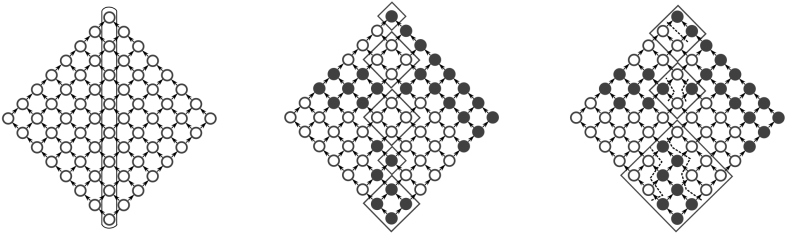
[25] Tiskin, FGCS, 2007

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

Tradeoffs in the diamond DAG

Computation vs synchronization tradeoff for the $n \times n$ diamond DAG,[27]

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



Dependency chain P   Monochrome dependency intervals   Multicolored dependency intervals

We generalize this idea[28]

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

[27] Papadimitriou, Ullman, SIAM JC, 1987

[28] 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:[29]
For triangular solve with an $n \times n$ matrix,

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

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

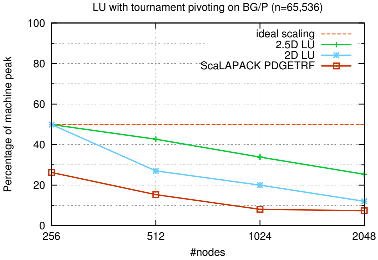
$$F_{\mathsf{CHOL}} \cdot S_{\mathsf{CHOL}}^2 = \Omega\left(n^3\right) \qquad W_{\mathsf{CHOL}} \cdot S_{\mathsf{CHOL}} = \Omega\left(n^2\right)$$

[29] 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[30] extended to tournament pivoting[31]
- first implementation of a communication-optimal LU algorithm[10]

[30] Tiskin, FGCS, 2007

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