Communication-Avoiding Algorithms for Linear Algebra and Beyond

Jim Demmel

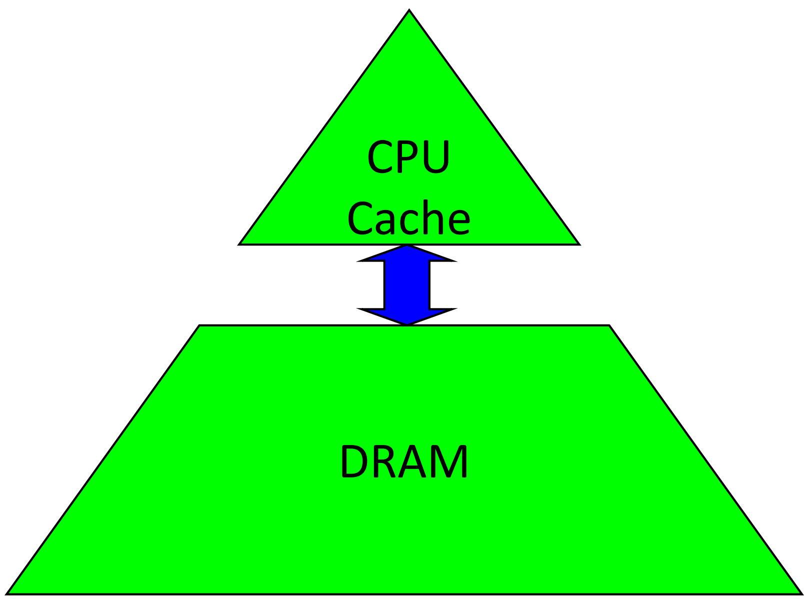
EECS & Math Departments

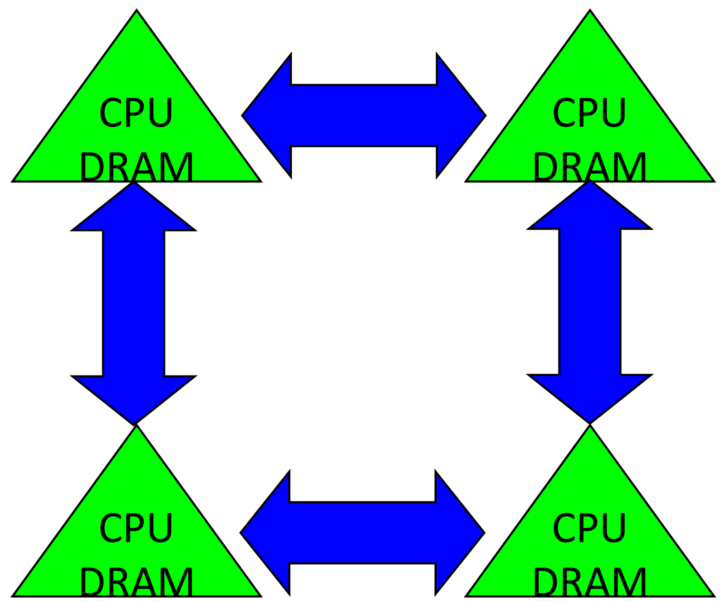
UC Berkeley

Why avoid communication? (1/2)

Algorithms have two costs (measured in time or energy):

1. Arithmetic (FLOPS)
2. Communication: moving data between
   - levels of a memory hierarchy (sequential case)
   - processors over a network (parallel case).





Why avoid communication? (2/2)

- Running time of an algorithm is sum of 3 terms:
  - # flops * time_per_flop
  - # words moved / bandwidth
  - # messages * latency

communication

- Time_per_flop << 1/ bandwidth << latency
  - Gaps growing exponentially with time [FOSC]

Annual improvements			
Time_per_flop		Bandwidth	Latency
59%	Network	26%	15%
	DRAM	23%	5%

- Avoid communication to save time
- Same story for saving energy

Goals

- Redesign algorithms to *avoid* communication
  - Between all memory hierarchy levels
    - L1 $\longleftrightarrow$ L2 $\longleftrightarrow$ DRAM $\longleftrightarrow$ network, etc
- Attain lower bounds if possible
  - Current algorithms often far from lower bounds
  - Large speedups and energy savings possible

Sample Speedups

- Up to **12x** faster for 2.5D matmul on 64K core IBM BG/P
- Up to **3x** faster for tensor contractions on 2K core Cray XE/6
- Up to **6.2x** faster for All-Pairs-Shortest-Path on 24K core Cray CE6
- Up to **2.1x** faster for 2.5D LU on 64K core IBM BG/P
- Up to **11.8x** faster for direct N-body on 32K core IBM BG/P
- Up to **13x** faster for Tall Skinny QR on Tesla C2050 Fermi NVIDIA GPU
- Up to **6.7x** faster for symeig(band A) on 10 core Intel Westmere
- Up to **2x** faster for 2.5D Strassen on 38K core Cray XT4
- Up to **4.2x** faster for MiniGMG benchmark bottom solver, using CA-BiCGStab (**2.5x** for overall solve) on 32K core Cray XE6
  - **2.5x** / **1.5x** for combustion simulation code
- Up to **5.1x** faster for coordinate descent LASSO on 3K core Cray XC30

Sample Speedups

- Up to **12x** faster for 2.5D matmul on 64K core IBM BG/P

Ideas adopted by Nervana, “deep learning” startup, acquired by Intel in August 2016

- Up to **2.1x** faster for 2.5D LU on 64K core IBM BG/P
- Up to **11.8x** faster for direct N-body on 32K core IBM BG/P
- Up to **13x** faster for Tall Skinny QR on Tesla C2050 Fermi NVIDIA GPU

SIAG on Supercomputing Best Paper Prize, 2016
Released in LAPACK 3.7, Dec 2016

- Up to **4.2x** faster for MiniGMG benchmark bottom solver, using CA-BiCGStab (**2.5x** for overall solve) on 32K core Cray XE6
  - **2.5x** / **1.5x** for combustion simulation code
- Up to **5.1x** faster for coordinate descent LASSO on 3K core Cray XC30

Outline

- Survey state of the art of CA (Comm-Avoiding) algorithms
  - Review previous Matmul algorithms
  - CA $O(n^3)$ 2.5D Matmul and LU
  - TSQR: Tall-Skinny QR
  - CA Strassen Matmul
- Beyond linear algebra
  - Extending lower bounds to any algorithm with arrays
  - Communication-optimal N-body and CNN algorithms
- CA-Krylov methods, and ML
- Related Topics

Outline

- Survey state of the art of CA (Comm-Avoiding) algorithms
  - Review previous Matmul algorithms
  - CA $O(n^3)$ 2.5D Matmul and LU
  - TSQR: Tall-Skinny QR
  - CA Strassen Matmul
- Beyond linear algebra
  - Extending lower bounds to any algorithm with arrays
  - Communication-optimal N-body and CNN algorithms
- CA-Krylov methods, and ML
- Related Topics

Summary of CA Linear Algebra

- “Direct” Linear Algebra
  - Lower bounds on communication for linear algebra problems like Ax=b, least squares, Ax = λx, SVD, etc
  - Mostly not attained by algorithms in standard libraries
  - New algorithms that attain these lower bounds
    - Being added to libraries: Sca/LAPACK, PLASMA, MAGMA
    - Large speed-ups possible
  - Autotuning to find optimal implementation
- Ditto for “Iterative” Linear Algebra

Lower bound for all "n^3-like" linear algebra

- Let M = "fast" memory size (per processor)

#words_moved (per processor) = Ω(#flops (per processor) / $M^{1/2}$)

- Parallel case: assume either load or memory balanced

- Holds for
  - Matmul

Lower bound for all "n^3-like" linear algebra

- Let M = "fast" memory size (per processor)

#words_moved (per processor) = Ω(#flops (per processor) / $M^{1/2}$)

#messages_sent $\geq$ #words_moved / largest_message_size

- Parallel case: assume either load or memory balanced

- Holds for
  - Matmul, BLAS, LU, QR, eig, SVD, tensor contractions, …
  - Some whole programs (sequences of these operations, no matter how individual ops are interleaved, eg A^k)
  - Dense and sparse matrices (where #flops << n^3)
  - Sequential and parallel algorithms
  - Some graph-theoretic algorithms (eg Floyd-Warshall)

Lower bound for all "n^3-like" linear algebra

- Let M = "fast" memory size (per processor)

#words_moved (per processor) = Ω(#flops (per processor) / $M^{1/2}$)

#messages_sent (per processor) = Ω(#flops (per processor) / $M^{3/2}$)

- Parallel case: assume either load or memory balanced

- Holds for
  - Matmul, BLAS, LU, QR, eig, SVD, tensor contractions, …
  - Some whole programs (sequences of these operations, no matter how individual ops are interleaved, eg A^k)

SIAM SIAG/Linear Algebra Prize, 2012

Ballard, D., Holtz, Schwartz

Can we attain these lower bounds?

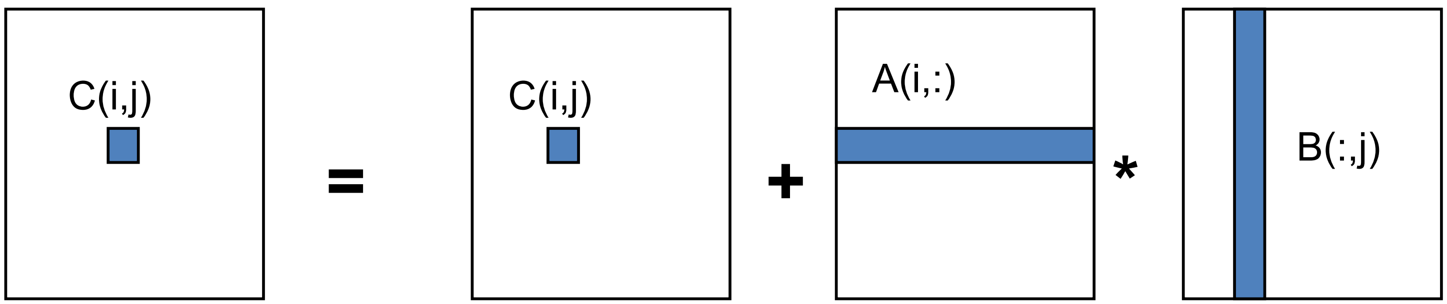
- Do conventional dense algorithms as implemented in LAPACK and ScaLAPACK attain these bounds?
  - Often not
- If not, are there other algorithms that do?
  - Yes, for much of dense linear algebra, APSP
  - New algorithms, with new numerical properties, new ways to encode answers, new data structures
  - Not just loop transformations (need those too!)
- Sparse algorithms: depends on sparsity structure
  - Ex: Sparse Cholesky of matrices with "large" separators
  - Ex: Matmul of "random" sparse matrices
  - Ex: Matmul of dense x sparse matrices – up to 100x faster
- Lots of work in progress

Outline

- Survey state of the art of CA (Comm-Avoiding) algorithms
  - Review previous Matmul algorithms
  - CA $O(n^3)$ 2.5D Matmul and LU
  - TSQR: Tall-Skinny QR
  - CA Strassen Matmul
- Beyond linear algebra
  - Extending lower bounds to any algorithm with arrays
  - Communication-optimal N-body and CNN algorithms
- CA-Krylov methods, and ML
- Related Topics

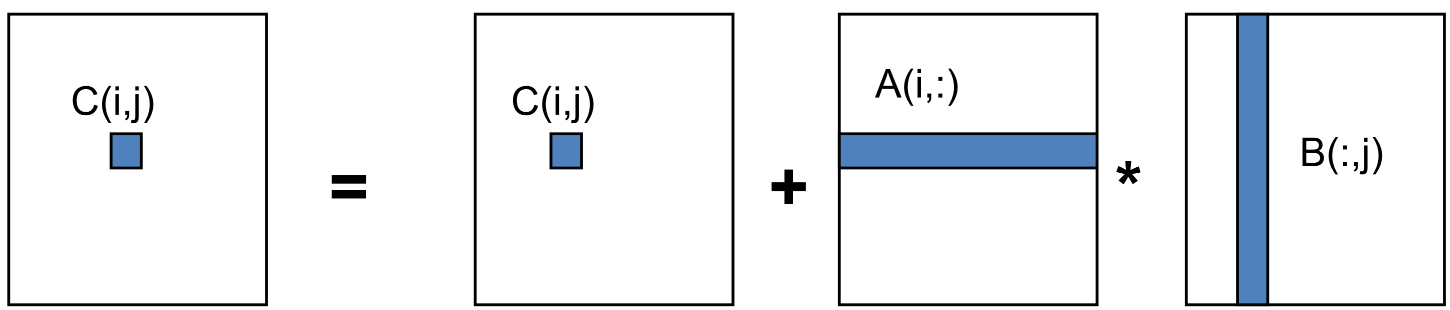
Naïve Matrix Multiply

```
{implements C = C + A*B}
for i = 1 to n

  for j = 1 to n



     for k = 1 to n
        C(i,j) = C(i,j) + A(i,k) * B(k,j)
```



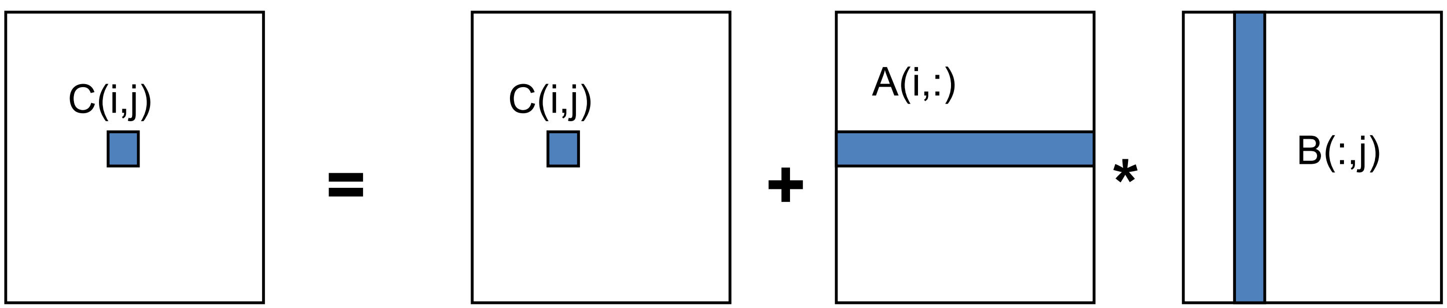
Naïve Matrix Multiply

```
{implements C = C + A*B}
for i = 1 to n
  {read row i of A into fast memory}
   for j = 1 to n
      {read C(i,j) into fast memory}
      {read column j of B into fast memory}
      for k = 1 to n
         C(i,j) = C(i,j) + A(i,k) * B(k,j)
      {write C(i,j) back to slow memory}
```



Naïve Matrix Multiply

```
{implements C = C + A*B}
for i = 1 to n
  {read row i of A into fast memory}               … n^2 reads altogether
  for j = 1 to n
    {read C(i,j) into fast memory}                 … n^2 reads altogether
    {read column j of B into fast memory}          … n^3 reads altogether
    for k = 1 to n
      C(i,j) = C(i,j) + A(i,k) * B(k,j)
    {write C(i,j) back to slow memory}             … n^2 writes altogether
```

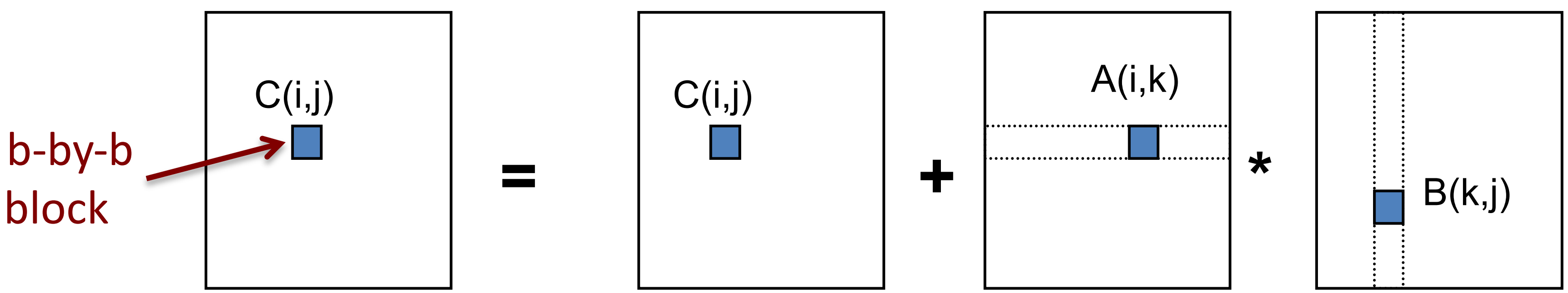


$n^3 + 3n^2$ reads/writes altogether – dominates $2n^3$ arithmetic

Blocked (Tiled) Matrix Multiply

Consider A,B,C to be n/b-by-n/b matrices of b-by-b subblocks where
b is called the block size; assume 3 b-by-b blocks fit in fast memory

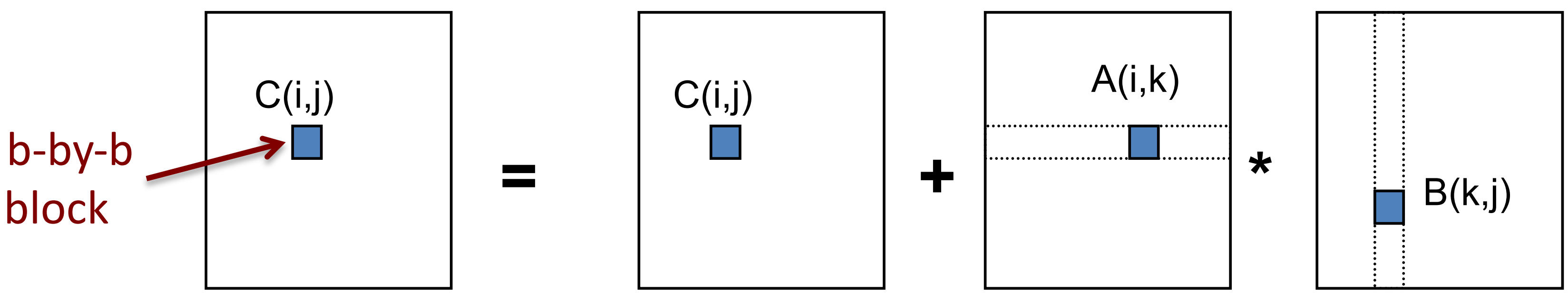
```
for i = 1 to n/b
  for j = 1 to n/b
    {read block C(i,j) into fast memory}
    for k = 1 to n/b
        {read block A(i,k) into fast memory}
        {read block B(k,j) into fast memory}
        C(i,j) = C(i,j) + A(i,k) * B(k,j) {do a matrix multiply on blocks}
    {write block C(i,j) back to slow memory}
```



Blocked (Tiled) Matrix Multiply

Consider A,B,C to be n/b-by-n/b matrices of b-by-b subblocks where
b is called the block size; assume 3 b-by-b blocks fit in fast memory

for i = 1 to n/b
  for j = 1 to n/b
    {read block C(i,j) into fast memory} ... $b^2 \times (n/b)^2 = n^2$ reads
    for k = 1 to n/b
      {read block A(i,k) into fast memory} ... $b^2 \times (n/b)^3 = n^3/b$ reads
      {read block B(k,j) into fast memory} ... $b^2 \times (n/b)^3 = n^3/b$ reads
      C(i,j) = C(i,j) + A(i,k) * B(k,j) {do a matrix multiply on blocks}
    {write block C(i,j) back to slow memory} ... $b^2 \times (n/b)^2 = n^2$ writes



$2n^3/b + 2n^2$ reads/writes << $2n^3$ arithmetic - Faster!

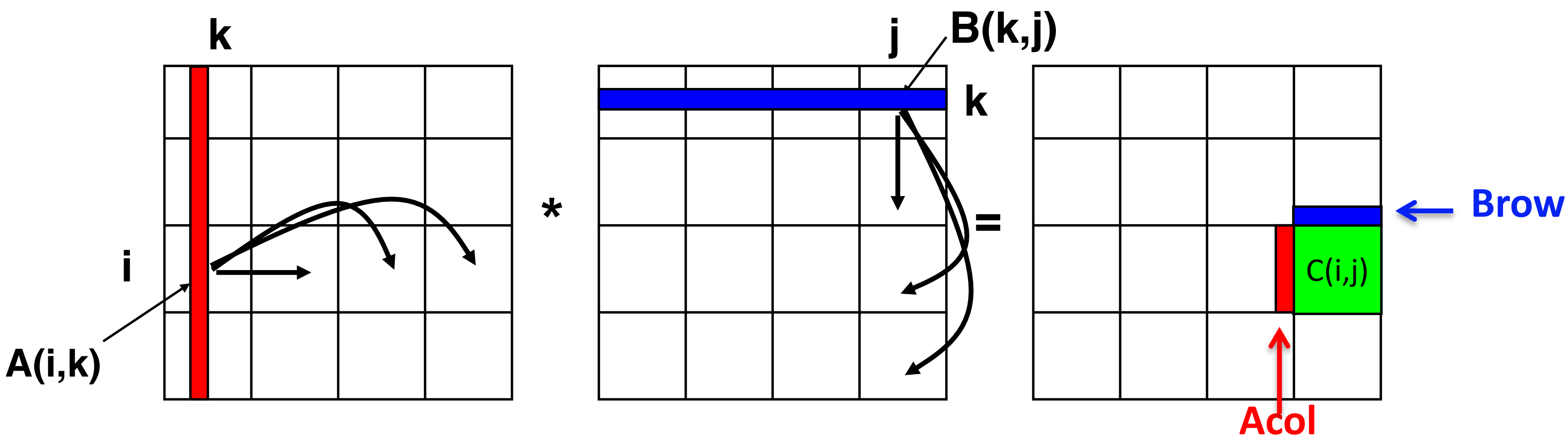
Does blocked matmul attain lower bound?

- Recall: if 3 b-by-b blocks fit in fast memory of size M, then #reads/writes = $2n^3/b + 2n^2$
- Make b as large as possible: $3b^2 \le M$, so #reads/writes $\ge 3^{1/2}n^3/M^{1/2} + 2n^2$
- Attains lower bound $= \Omega$ (#flops / $M^{1/2}$)

- But what if we don't know M?
- Or if there are multiple levels of fast memory?
- Can use "Cache Oblivious" algorithm

SUMMA– n x n matmul on $P^{1/2}$ x $P^{1/2}$ grid

(nearly) optimal using minimum memory $M=O(n^2/P)$



For k=0 to n/b-1 … b = block size = #cols in A(i,k) = #rows in B(k,j)

for all i = 1 to $P^{1/2}$

owner of A(i,k) broadcasts it to whole processor row (using binary tree)

for all j = 1 to $P^{1/2}$

owner of B(k,j) broadcasts it to whole processor column (using bin. tree)

Receive A(i,k) into Acol

Receive B(k,j) into Brow

C_myproc = C_myproc + Acol * Brow

Summary of dense *parallel* algorithms attaining communication lower bounds

- Assume nxn matrices on P processors
- Minimum Memory per processor = $M = O(n^2 / P)$
- Recall lower bounds:

  #words_moved $= \Omega((n^3/ P) / M^{1/2}) = \Omega(n^2 / P^{1/2})$

  #messages $= \Omega((n^3/ P) / M^{3/2}) = \Omega(P^{1/2})$
- Does ScaLAPACK attain these bounds?
  - For #words_moved: mostly, except nonsym. Eigenproblem
  - For #messages: asymptotically worse, except Cholesky
- New algorithms attain all bounds, up to polylog(P) factors
  - Cholesky, LU, QR, Sym. and Nonsym eigenproblems, SVD

Can we do Better?

Can we do better?

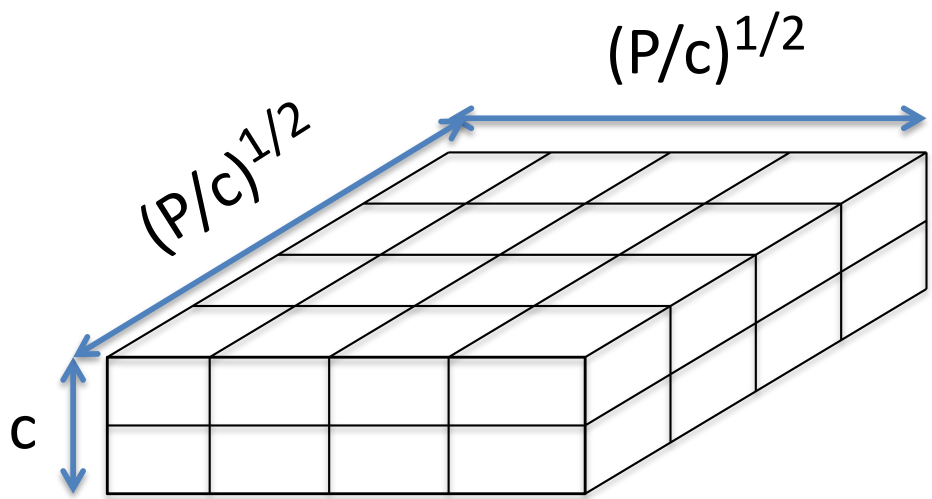
- Aren't we already optimal?
- Why assume $M = O(n^2/p)$, i.e. minimal?
  - Lower bound still true if more memory
  - Can we attain it?
- Special case: "3D Matmul"
  - Uses $M = O(n^2/p^{2/3})$
  - Dekel, Nassimi, Sahni [81], Bernsten [89], Agarwal, Chandra, Snir [90], Johnson [93], Agarwal, Balle, Gustavson, Joshi, Palkar [95]
- Not always $p^{1/3}$ times as much memory available…

Outline

- Survey state of the art of CA (Comm-Avoiding) algorithms
  - Review previous Matmul algorithms
  - CA $O(n^3)$ 2.5D Matmul and LU
  - TSQR: Tall-Skinny QR
  - CA Strassen Matmul
- Beyond linear algebra
  - Extending lower bounds to any algorithm with arrays
  - Communication-optimal N-body and CNN algorithms
- CA-Krylov methods, and ML
- Related Topics

2.5D Matrix Multiplication

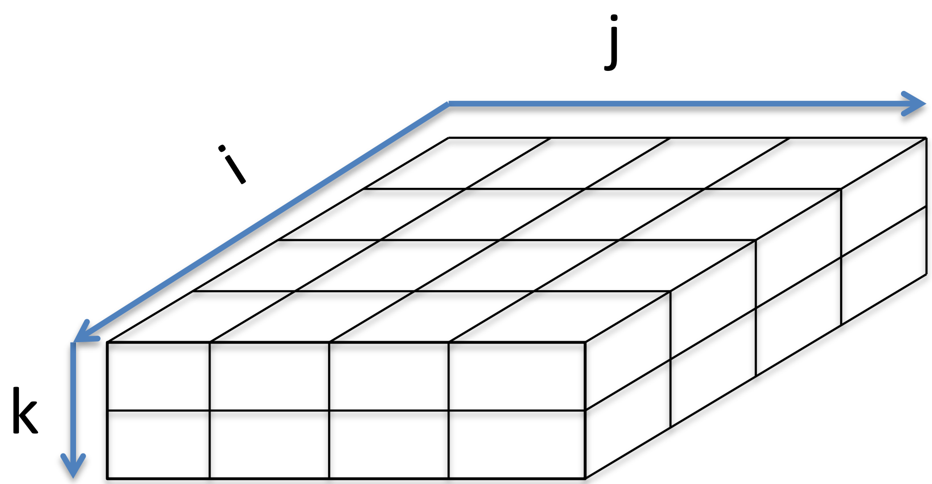
- Assume can fit cn^2/P data per processor, $c > 1$
- Processors form $(P/c)^{1/2}$ x $(P/c)^{1/2}$ x c grid



Example: P = 32, c = 2

2.5D Matrix Multiplication

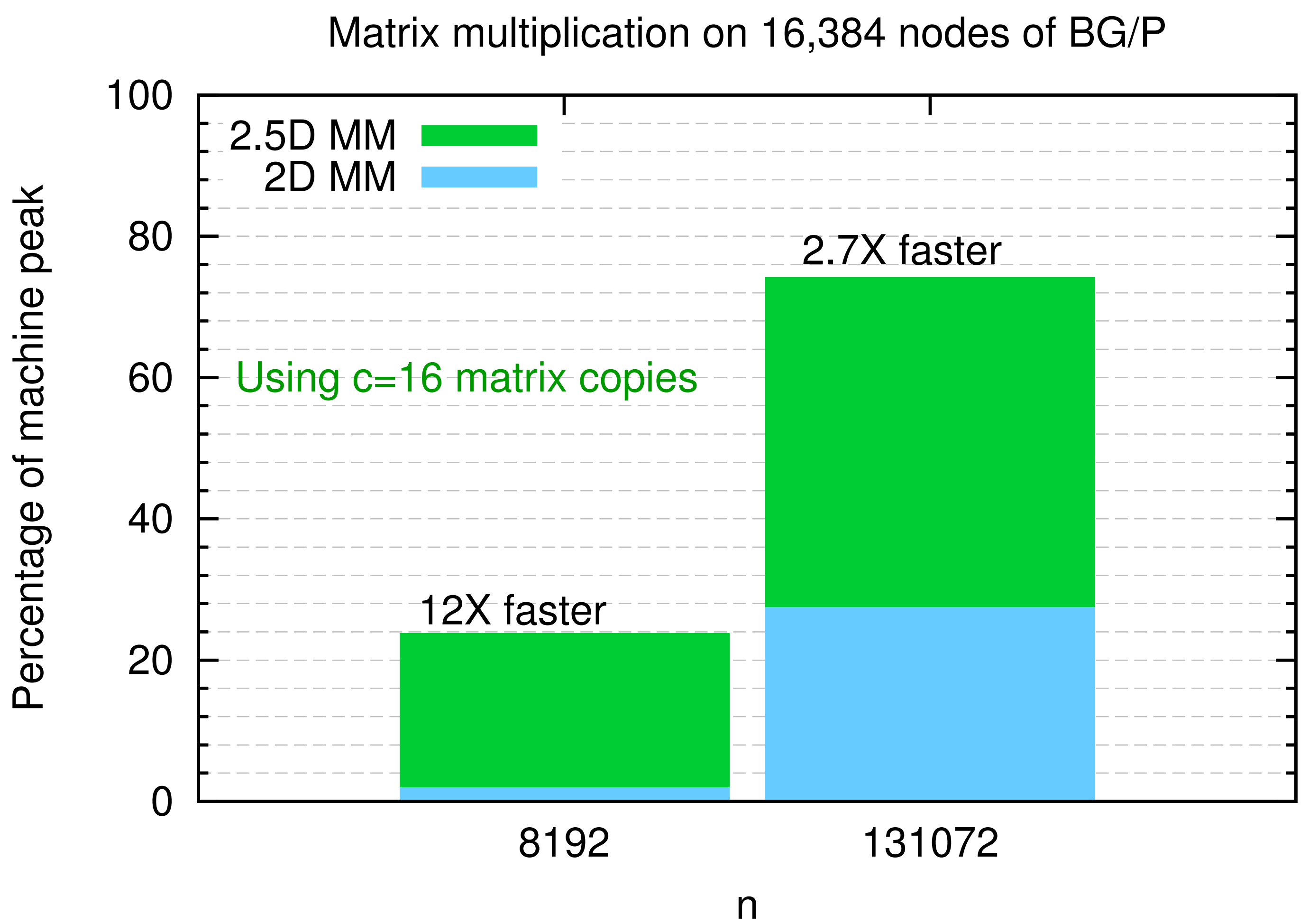
- Assume can fit cn^2/P data per processor, $c > 1$
- Processors form $(P/c)^{1/2}$ x $(P/c)^{1/2}$ x c grid



Initially P(i,j,0) owns A(i,j) and B(i,j) each of size $n(c/P)^{1/2}$ x $n(c/P)^{1/2}$

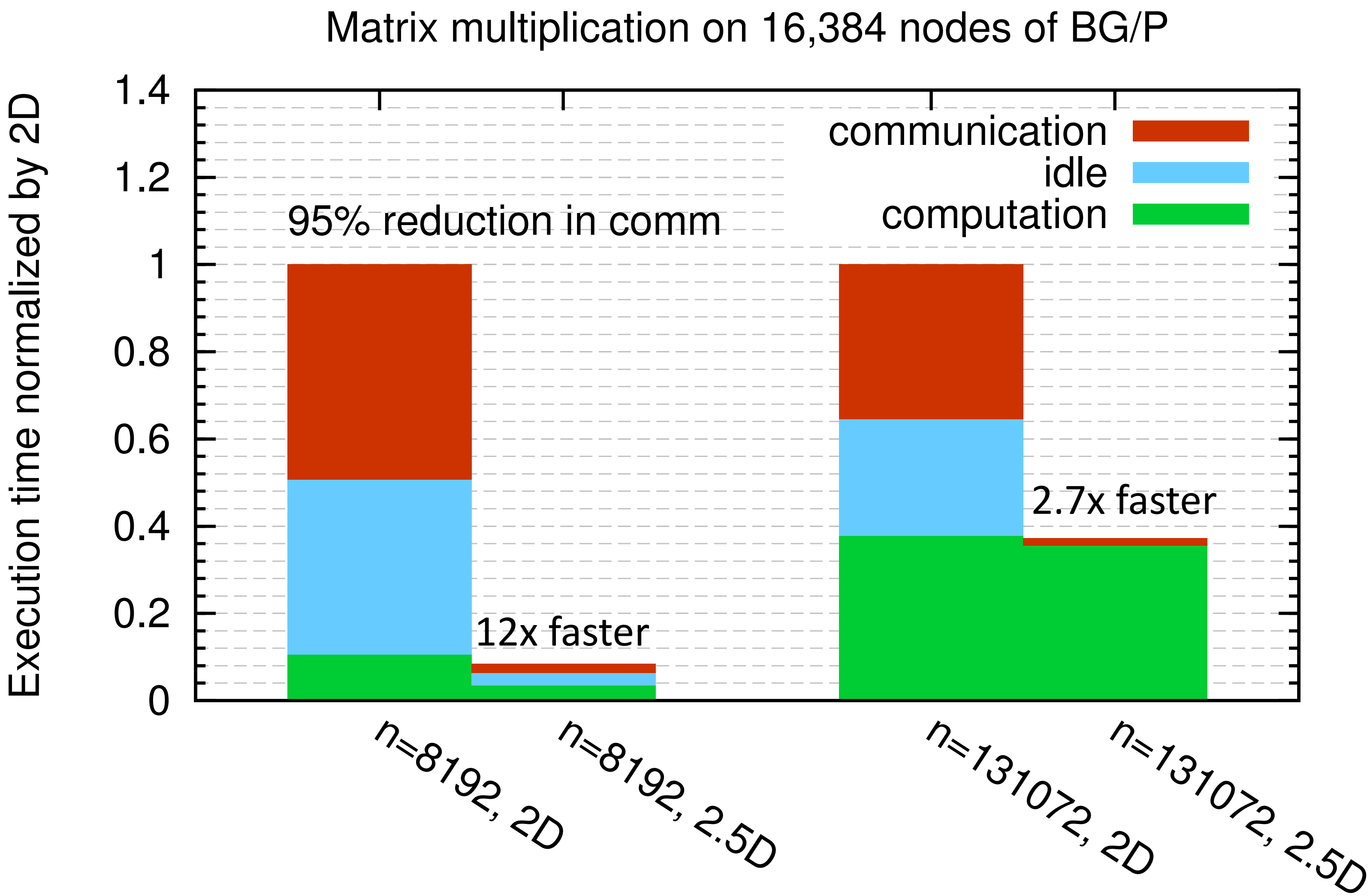
(1) P(i,j,0) broadcasts A(i,j) and B(i,j) to P(i,j,k)
(2) Processors at level k perform 1/c-th of SUMMA, i.e. 1/c-th of Σ_m A(i,m)*B(m,j)
(3) Sum-reduce partial sums Σ_m A(i,m)*B(m,j) along k-axis so P(i,j,0) owns C(i,j)

2.5D Matmul on BG/P, 16K nodes / 64K cores



2.5D Matmul on BG/P, 16K nodes / 64K cores

c = 16 copies



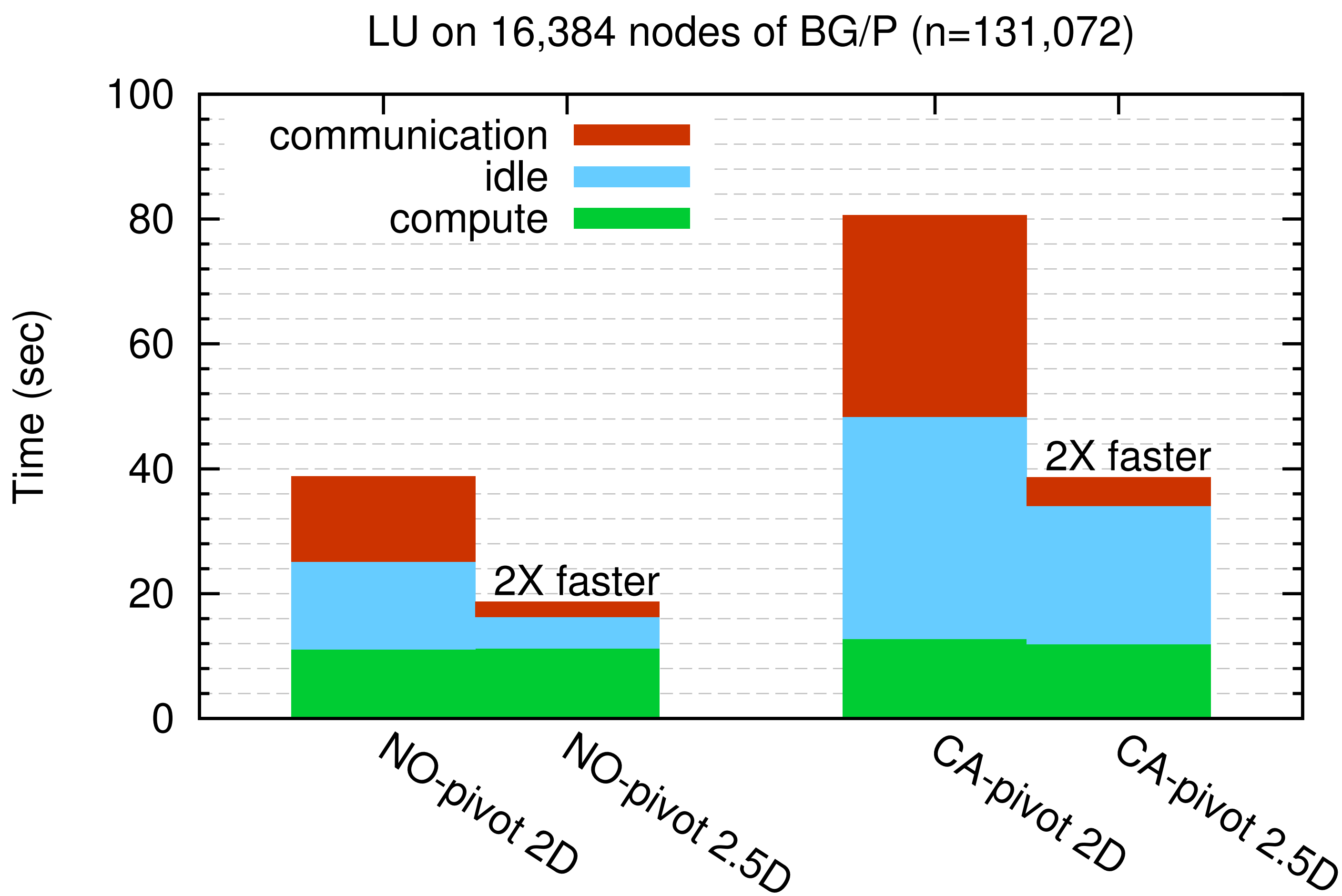
Distinguished Paper Award, EuroPar'11 (Solomonik, D.)

SC'11 paper by Solomonik, Bhatele, D.

Perfect Strong Scaling – in Time and Energy

- Every time you add a processor, you should use its memory M too
- Start with minimal number of procs: $PM = 3n^2$
- Increase P by a factor of c ➔ total memory increases by a factor of c
- Notation for timing model:
  - γ_T , β_T , α_T = secs per flop, per word_moved, per message of size m
- $T(cP) = n^3/(cP)\ [\ \gamma_T + \beta_T/M^{1/2} + \alpha_T/(mM^{1/2})\ ]$
  $= T(P)/c$
- Notation for energy model:
  - γ_E , β_E , α_E = joules for same operations
  - δ_E = joules per word of memory used per sec
  - ε_E = joules per sec for leakage, etc.
- $E(cP) = cP\ \{\ n^3/(cP)\ [\ \gamma_E + \beta_E/M^{1/2} + \alpha_E/(mM^{1/2})\ ] + \delta_E M T(cP) + \varepsilon_E T(cP)\ \}$
  $= E(P)$
- Extends to N-body, Strassen, …
- Can prove lower bounds on needed network (eg 3D torus for matmul)

2.5D vs 2D LU
With and Without Pivoting



Thm: 2.5D version of QR possible too

Thm: Perfect Strong Scaling impossible, because Latency*BW = $\Omega(n^2)$

Outline

- Survey state of the art of CA (Comm-Avoiding) algorithms
  - Review previous Matmul algorithms
  - CA $O(n^3)$ 2.5D Matmul and LU
  - TSQR: Tall-Skinny QR
  - CA Strassen Matmul
- Beyond linear algebra
  - Extending lower bounds to any algorithm with arrays
  - Communication-optimal N-body and CNN algorithms
- CA-Krylov methods, and ML
- Related Topics

TSQR: QR of a Tall, Skinny matrix

$$W = \begin{pmatrix} W_0 \\ \hline W_1 \\ \hline W_2 \\ \hline W_3 \end{pmatrix}$$

$$\begin{pmatrix} R_{00} \\ R_{10} \\ \hline R_{20} \\ R_{30} \end{pmatrix} = \begin{pmatrix} Q_{01}\, R_{01} \\ \hline Q_{11}\, R_{11} \end{pmatrix}$$

$$\begin{pmatrix} R_{01} \\ \hline R_{11} \end{pmatrix} = \begin{pmatrix} Q_{02}\, R_{02} \end{pmatrix}$$

TSQR: QR of a Tall, Skinny matrix

$$
W = \left(\begin{array}{c} W_0 \\ \hline W_1 \\ \hline W_2 \\ \hline W_3 \end{array}\right) = \left(\begin{array}{c} Q_{00}\,R_{00} \\ \hline Q_{10}\,R_{10} \\ \hline Q_{20}\,R_{20} \\ \hline Q_{30}\,R_{30} \end{array}\right) = \left(\begin{array}{cccc} Q_{00} & & & \\ \hline & Q_{10} & & \\ \hline & & Q_{20} & \\ \hline & & & Q_{30} \end{array}\right) \cdot \left(\begin{array}{c} R_{00} \\ \hline R_{10} \\ \hline R_{20} \\ \hline R_{30} \end{array}\right)
$$

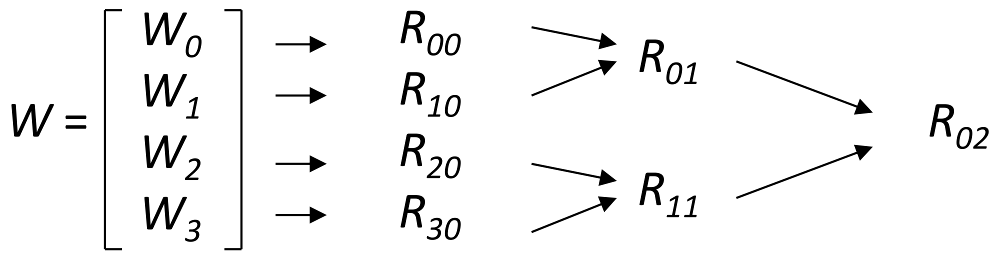
$$
\left(\begin{array}{c} R_{00} \\ R_{10} \\ \hline R_{20} \\ R_{30} \end{array}\right) = \left(\begin{array}{c} Q_{01}\,R_{01} \\ \hline Q_{11}\,R_{11} \end{array}\right) = \left(\begin{array}{cc} Q_{01} & \\ \hline & Q_{11} \end{array}\right) \cdot \left(\begin{array}{c} R_{01} \\ \hline R_{11} \end{array}\right)
$$

$$
\left(\begin{array}{c} R_{01} \\ \hline R_{11} \end{array}\right) = \left(Q_{02}\,R_{02} \right)
$$

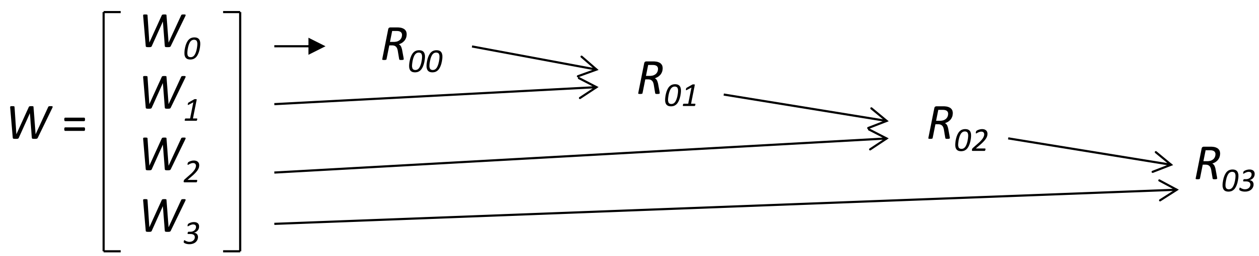
Output = $\{ Q_{00}, Q_{10}, Q_{20}, Q_{30}, Q_{01}, Q_{11}, Q_{02}, R_{02} \}$

TSQR: An Architecture-Dependent Algorithm

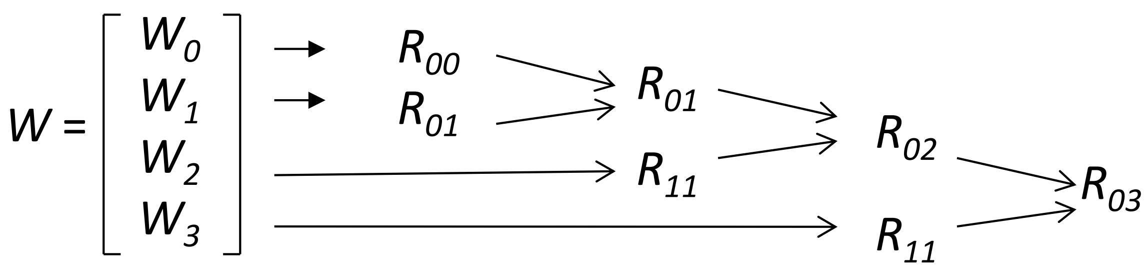
Parallel:



Sequential:



Dual Core:



Multicore / Multisocket / Multirack / Multisite / Out-of-core: ?

Can choose reduction tree dynamically

TSQR Performance Results

- Parallel
  - Intel Clovertown
    - Up to **8x** speedup (8 core, dual socket, 10M x 10)
  - Pentium III cluster, Dolphin Interconnect, MPICH
    - Up to **6.7x** speedup (16 procs, 100K x 200)
  - BlueGene/L
    - Up to **4x** speedup (32 procs, 1M x 50)
  - Tesla C 2050 / Fermi
    - Up to **13x** (110,592 x 100)
  - Grid – **4x** on 4 cities vs 1 city (Dongarra, Langou et al)
  - Cloud – **1.6x slower than just accessing data twice** (Gleich and Benson)
- Sequential
  - "**Infinite speedup**" for out-of-core on PowerPC laptop
    - As little as 2x slowdown vs (predicted) infinite DRAM
    - LAPACK with virtual memory never finished
- SVD costs about the same
- Joint work with Grigori, Hoemmen, Langou, Anderson, Ballard, Keutzer, others

TSQR Performance Results

- Parallel
  - Intel Clovertown
    - Up to **8x** speedup (8 core, dual socket, 10M x 10)
  - Pentium III cluster, Dolphin Interconnect, MPICH
    - Up to **6.7x** speedup (16 procs, 100K x 200)
  - BlueGene/L
    - Up to **4x** speedup (32 procs, 1M x 50)
  - Tesla C 2050 / Fermi
    - Up to **13x** (110,592 x 100)
  - Grid – **4x** on 4 cities vs 1 city (Dongarra, Langou et al)
  - Cloud – **1.6x slower than just accessing data twice** (Gleich and Benson)
- Sequential
  - "**Infinite speedup**" for out-of-core on PowerPC laptop
    - As little as 2x slowdown vs (predicted) infinite DRAM
    - LAPACK with virtual memory never finished
-
- **SIAG on Supercomputing Best Paper Prize, 2016**

Related Work

- Lots more work on
  - Algorithms:
    - BLAS, LDL^T, QR with pivoting, other pivoting schemes, eigenproblems, …
    - Sparse matrices, structured matrices
    - All-pairs-shortest-path, …
    - Both 2D (c=1) and 2.5D (c>1)
    - But only bandwidth may decrease with c>1, not latency (eg LU)
  - Platforms:
    - Multicore, cluster, GPU, cloud, heterogeneous, low-energy, …
  - Software:
    - Integration into Sca/LAPACK, PLASMA, MAGMA,…
- Integration into applications
  - CTF (with ANL): symmetric tensor contractions

Outline

- Survey state of the art of CA (Comm-Avoiding) algorithms
  - Review previous Matmul algorithms
  - CA $O(n^3)$ 2.5D Matmul and LU
  - TSQR: Tall-Skinny QR
  - CA Strassen Matmul
- Beyond linear algebra
  - Extending lower bounds to any algorithm with arrays
  - Communication-optimal N-body and CNN algorithms
- CA-Krylov methods, and ML
- Related Topics

Outline

- Survey state of the art of CA (Comm-Avoiding) algorithms
  - Review previous Matmul algorithms
  - CA $O(n^3)$ 2.5D Matmul and LU
  - TSQR: Tall-Skinny QR
  - CA Strassen Matmul
- Beyond linear algebra
  - Extending lower bounds to any algorithm with arrays
  - Communication-optimal N-body and CNN algorithms
- CA-Krylov methods, and ML
- Related Topics

Recall optimal sequential Matmul

- Naïve code
  for i=1:n, for j=1:n, for k=1:n, C(i,j)+=A(i,k)*B(k,j)

- "Blocked" code
  for i1 = 1:b:n, for j1 = 1:b:n,   for k1 = 1:b:n
    for i2 = 0:b-1, for j2 = 0:b-1,   for k2 = 0:b-1
      i=i1+i2, j = j1+j2, k = k1+k2
      C(i,j)+=A(i,k)*B(k,j)

  b x b matmul

- Thm: Picking b = $M^{1/2}$ attains lower bound:
  #words_moved = $\Omega(n^3/M^{1/2})$
- Where does **1/2** come from?

New Thm applied to Matmul

- for i=1:n, for j=1:n, for k=1:n, C(i,j) += A(i,k)*B(k,j)
- Record array indices in matrix Δ

$$\Delta = \begin{array}{c} \\ \\ \\ \\ \end{array}\begin{array}{c} \begin{array}{ccc} i & j & k \end{array} \\ \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 0 \end{pmatrix} \end{array} \begin{array}{c} \\ A \\ B \\ C \end{array}$$

- Solve LP for $x = [xi,xj,xk]^T$: max $\mathbf{1}^T x$ s.t. $\Delta\, x \le \mathbf{1}$
  - Result: $x = [1/2, 1/2, 1/2]^T$, $\mathbf{1}^T x = 3/2 = s_{HBL}$
- Thm: #words_moved = $\Omega(n^3/M^{s_{HBL}-1}) = \Omega(n^3/M^{1/2})$

  Attained by block sizes $M^{xi}, M^{xj}, M^{xk} = M^{1/2}, M^{1/2}, M^{1/2}$

New Thm applied to Direct N-Body

- for i=1:n, for j=1:n, F(i) += force(P(i) , P(j))
- Record array indices in matrix Δ

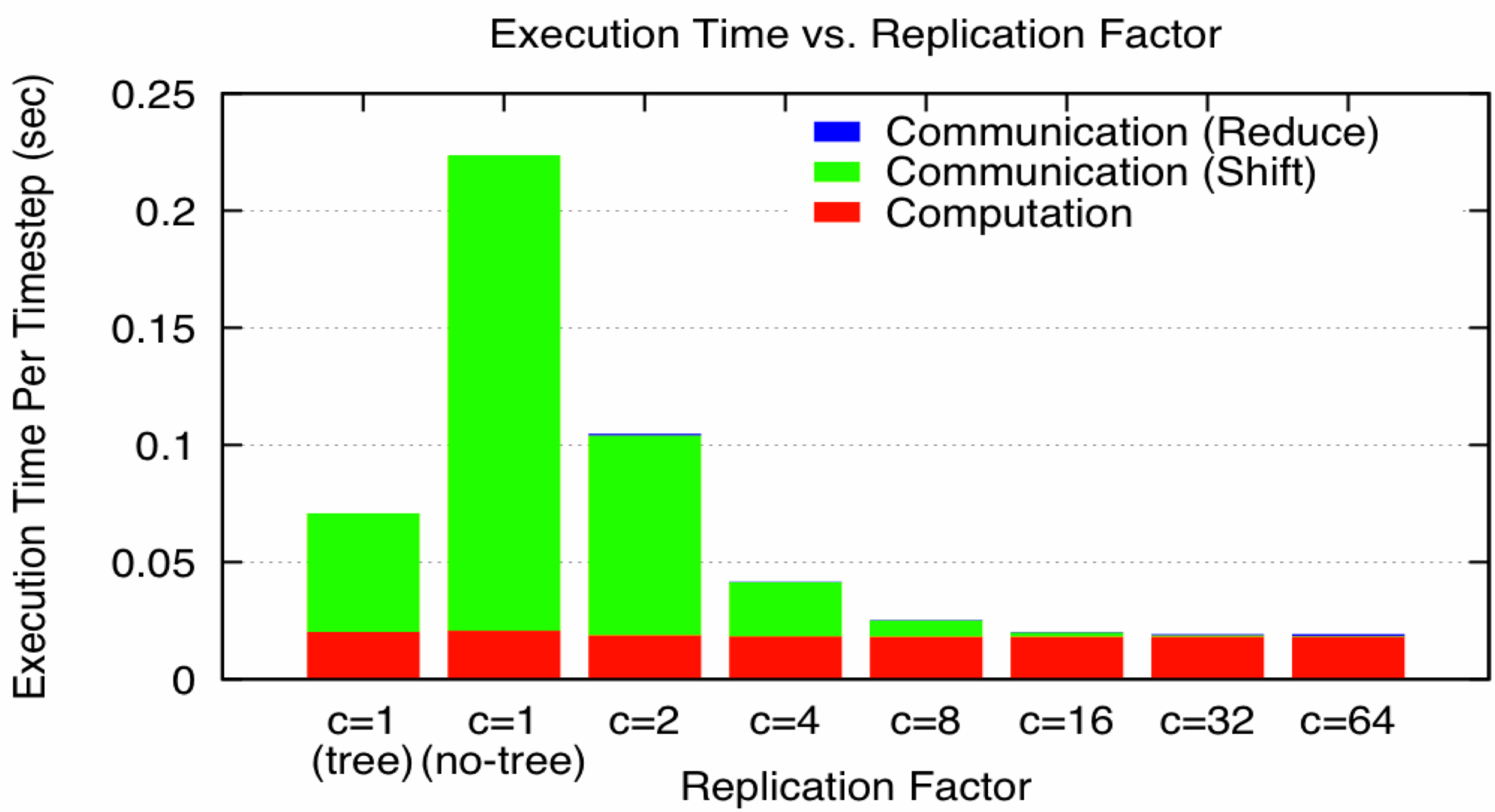
$$\Delta = \begin{array}{cc} \begin{array}{cc} i & j \end{array} & \\ \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix} & \begin{array}{c} F \\ P(i) \\ P(j) \end{array} \end{array}$$

- Solve LP for $x = [xi,xj]^T$: max $\mathbf{1}^T x$ s.t. $\Delta\, x \le \mathbf{1}$
  - Result: $x = [1,1]$, $\mathbf{1}^T x = 2 = s_{HBL}$
- Thm: #words_moved = $\Omega(n^2/M^{s_{HBL}-1}) = \Omega(n^2/M^1)$

  Attained by block sizes $M^{xi}, M^{xj} = M^1, M^1$

N-Body Speedups on IBM-BG/P (Intrepid) 8K cores, 32K particles

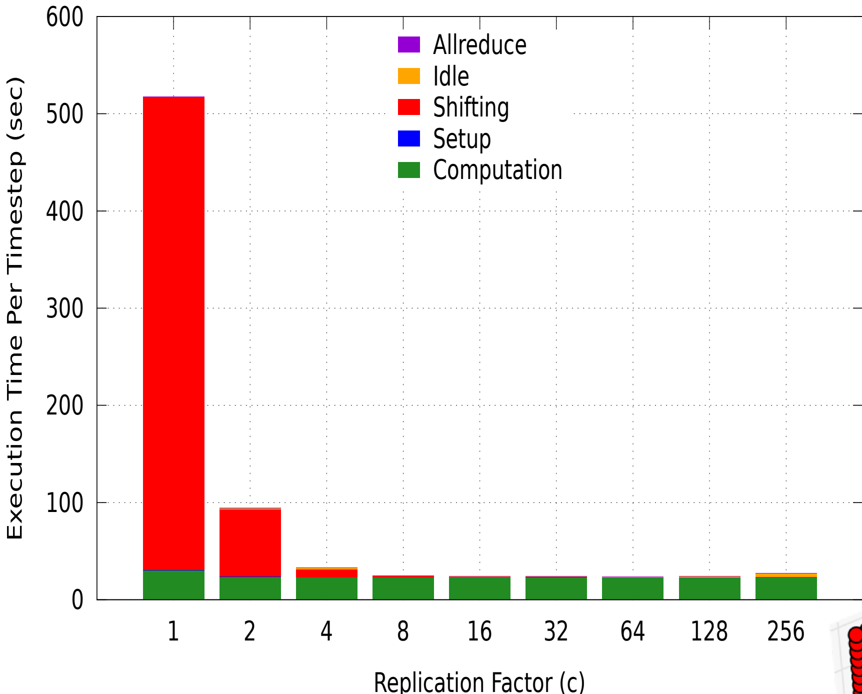
K. Yelick, E. Georganas, M. Driscoll, P. Koanantakool, E. Solomonik

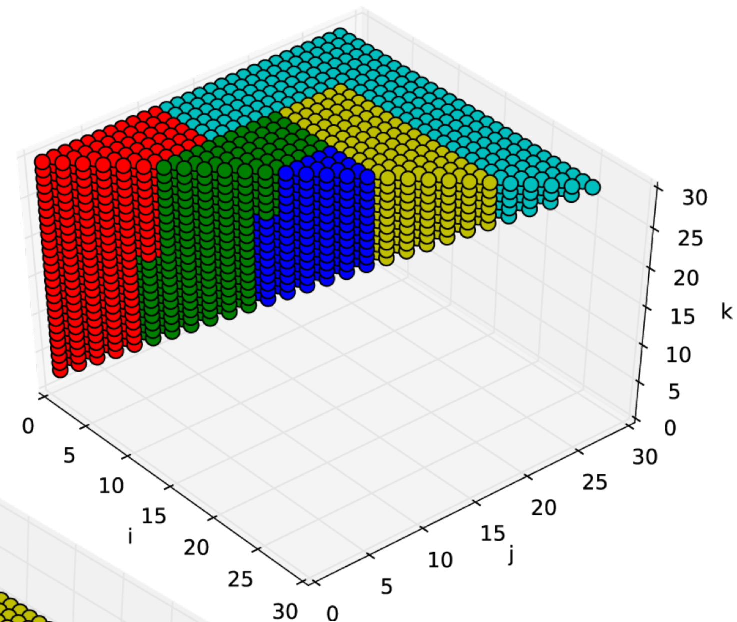


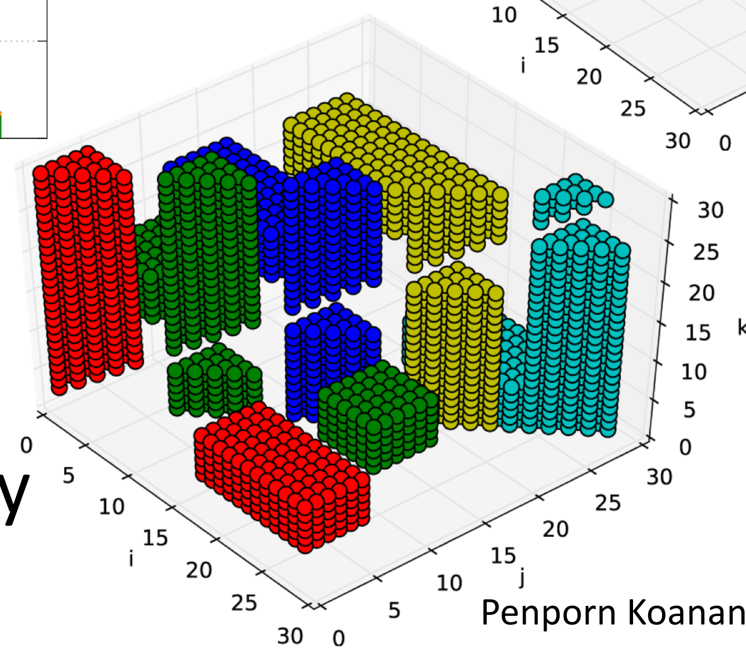
11.8x speedup

Variable Loop Bounds are More Complicated

- Redundancy in n-body calculation f(i,j), f(j,i)
- k-way n-body problems (“k-body”) has even more





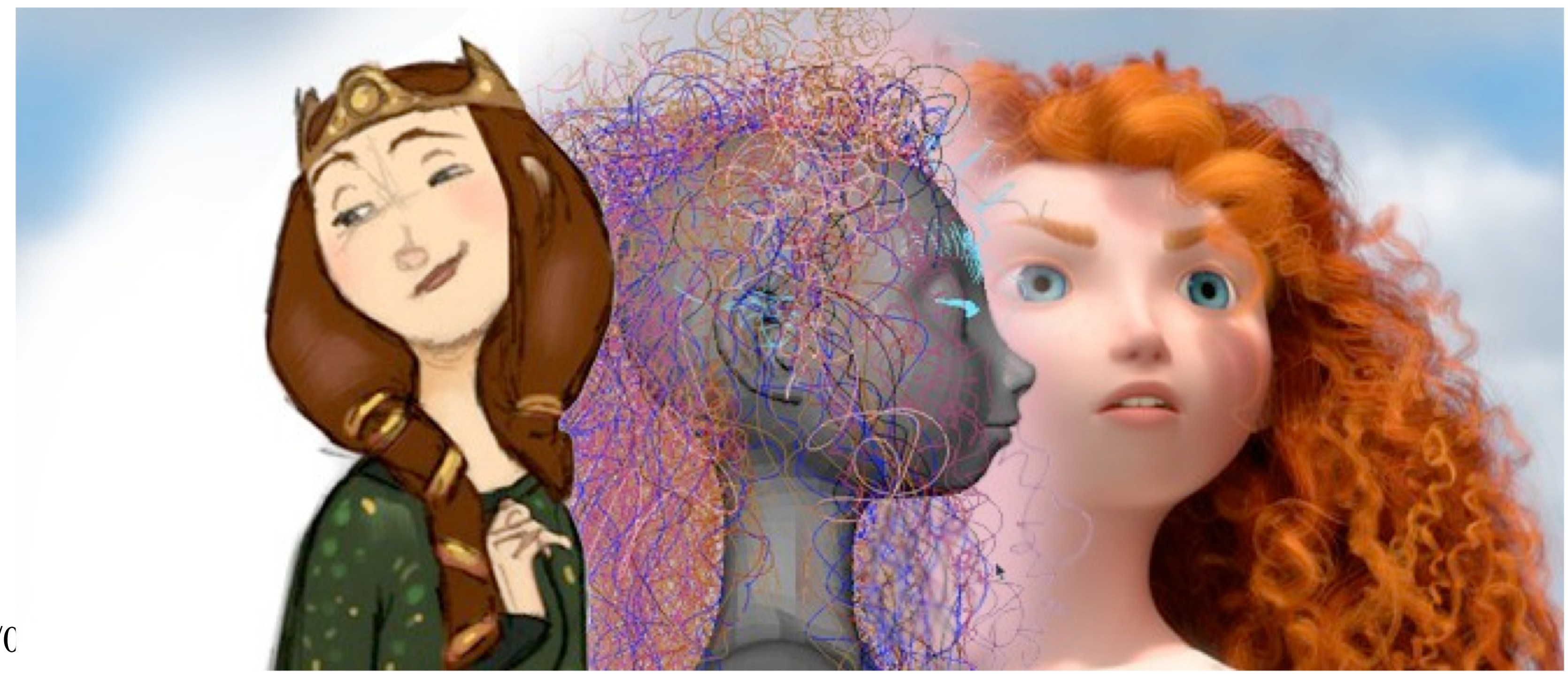


- Can achieve both communication and computation (symmetry exploiting) optimal

Penporn Koanantakool and K. Yelick

Some Applications

- Gravity, Turbulence, Molecular Dynamics, Plasma Simulation, …
- Electron-Beam Lithography Device Simulation
- Hair ...
  - www.fxguide.com/featured/brave-new-hair/
  - graphics.pixar.com/library/CurlyHairA/paper.pdf



Where do lower and matching upper bounds on communication come from?

- Originally for C = A*B by Irony/Tiskin/Toledo (2004)
- Proof idea
  - Suppose we can bound #useful_operations ≤ G doable with data in fast memory of size M
  - So to do F = #total_operations, need to fill fast memory F/G times, and so #words_moved ≥ MF/G
- Hard part: finding G
- Attaining lower bound
  - Need to “block” all operations to perform ~G operations on every chunk of M words of data

Approach to generalizing lower bounds

- Matmul

  for i=1:n, for j=1:n, for k=1:n,

  C(i,j)+=A(i,k)*B(k,j)

=> for (i,j,k) in S = subset of Z^3

  Access locations indexed by (i,j), (i,k), (k,j)

- General case

  for i1=1:n, for i2 = i1:m, … for ik = i3:i4

  C(i1+2*i3-i7) = func(A(i2+3*i4,i1,i2,i1+i2,…),B(pnt(3*i4)),…)

  D(something else) = func(something else), …

=> for (i1,i2,…,ik) in S = subset of Z^k

  Access locations indexed by group homomorphisms, eg

  ϕ_C (i1,i2,…,ik) = (i1+2*i3-i7)

  ϕ_A (i1,i2,…,ik) = (i2+3*i4,i1,i2,i1+i2,…), …

- Goal: Communication lower bounds, optimal algorithms for *any* program that looks like this

General Communication Lower Bound

- Thm: Given a program with array refs given by projections ϕ_j, then there is an $e \geq 1$ such that

  #words_moved = Ω (#iterations/M^{e-1})

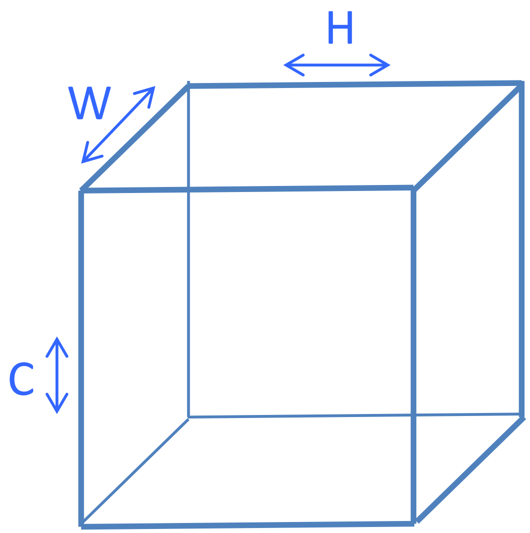
  where e is the the value of a linear program:

  minimize $e = \Sigma_j\, e_j$ subject to

  $\mathrm{rank}(H) \leq \Sigma_j\, e_j * \mathrm{rank}(\phi_j(H))$ for all subgroups $H < Z^k$

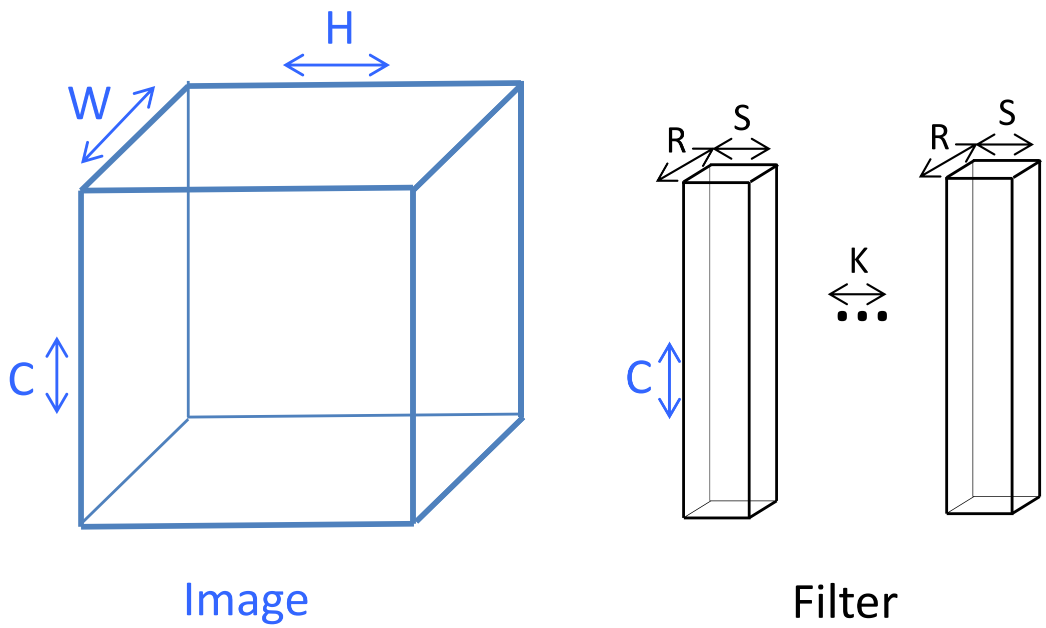
  – Proof depends on recent result in pure mathematics by Christ/Tao/Carbery/Bennett
- Thm: This lower bound is attainable, via loop tiling
  – Assumptions: dependencies permit, and iteration space big enough

What CNNs compute

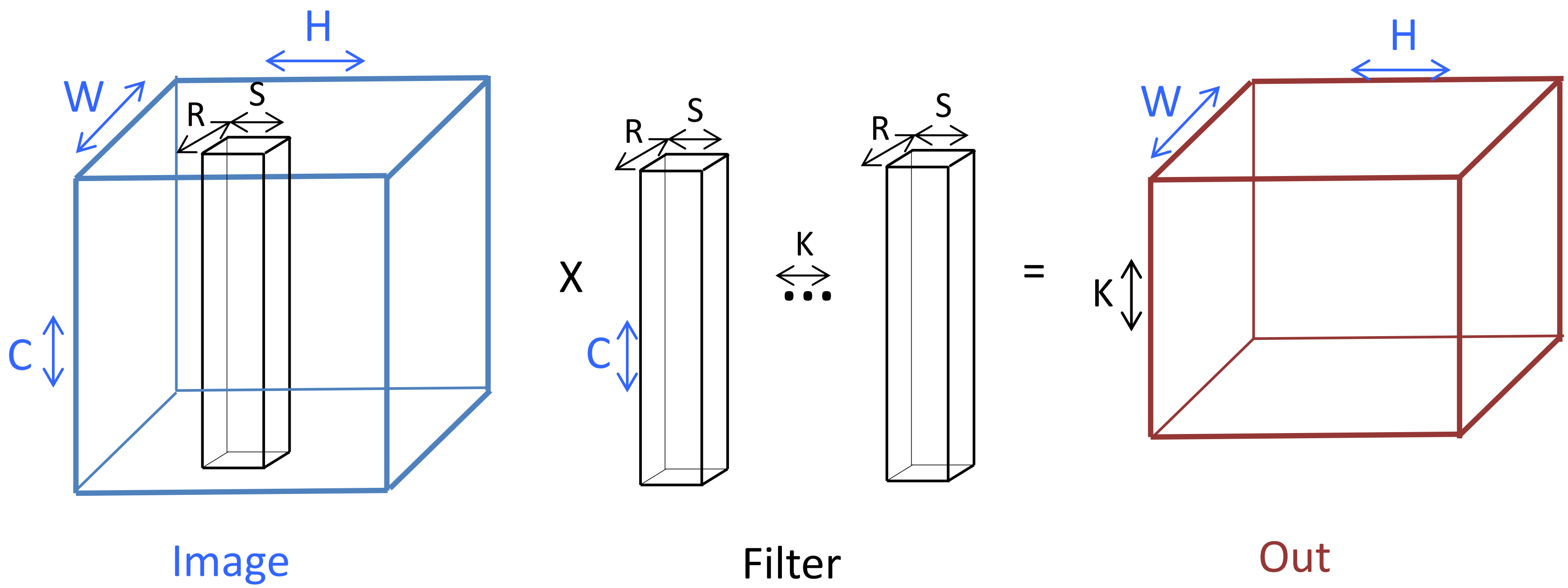


Image

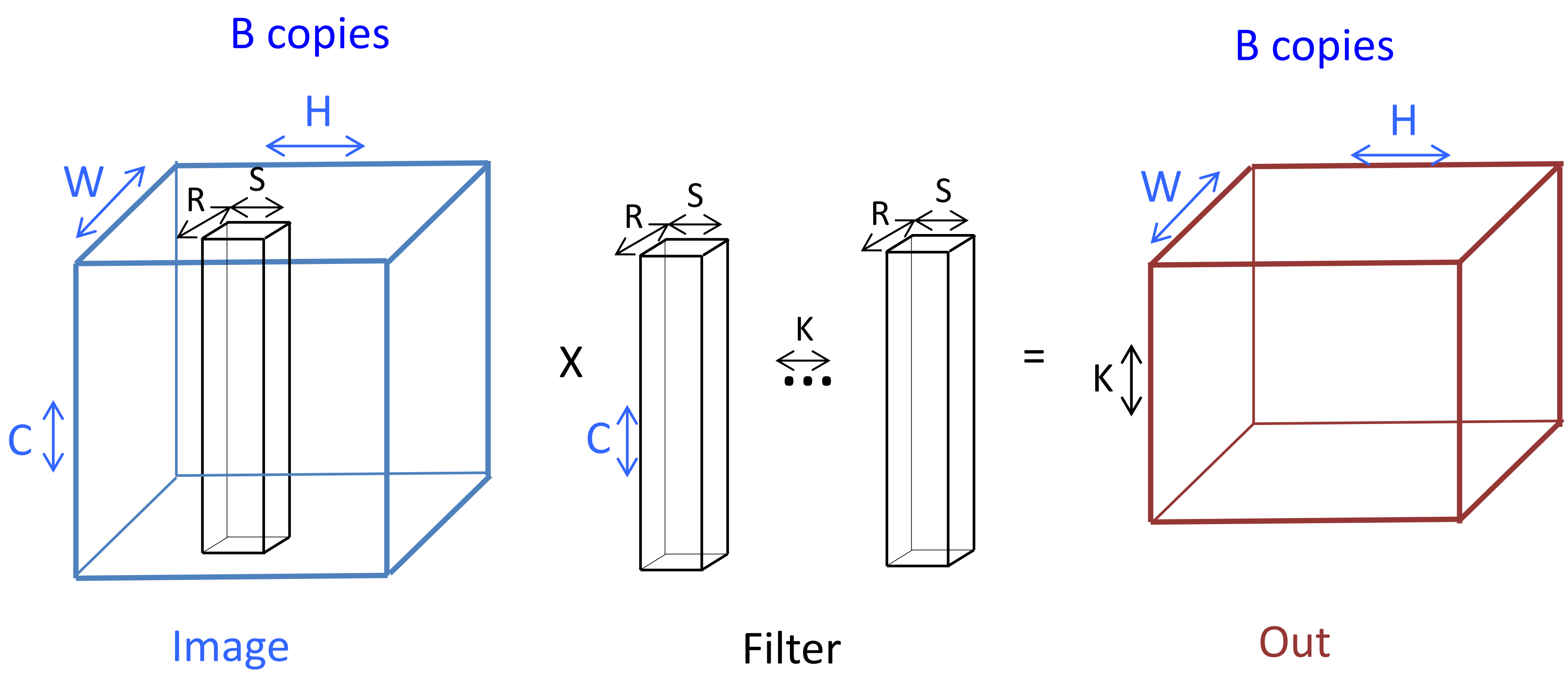
What CNNs compute



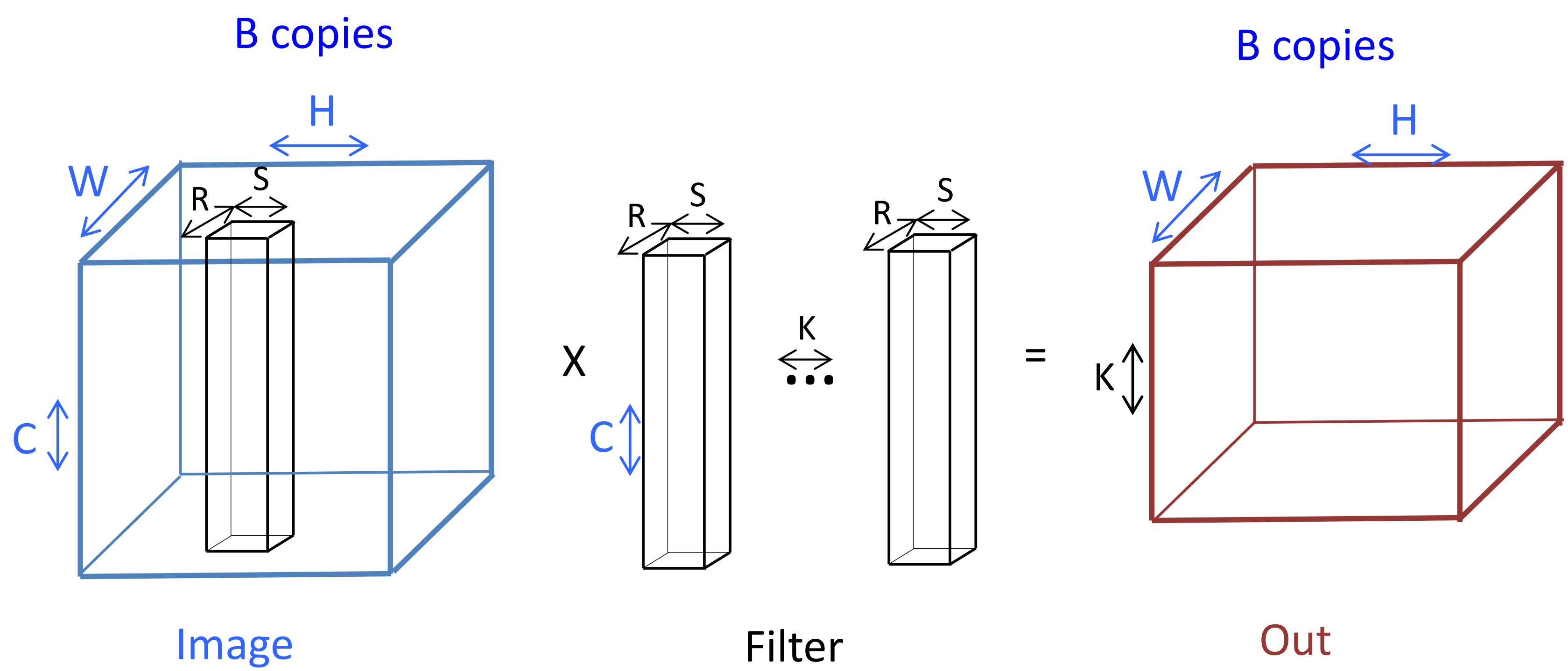
What CNNs compute



What CNNs compute

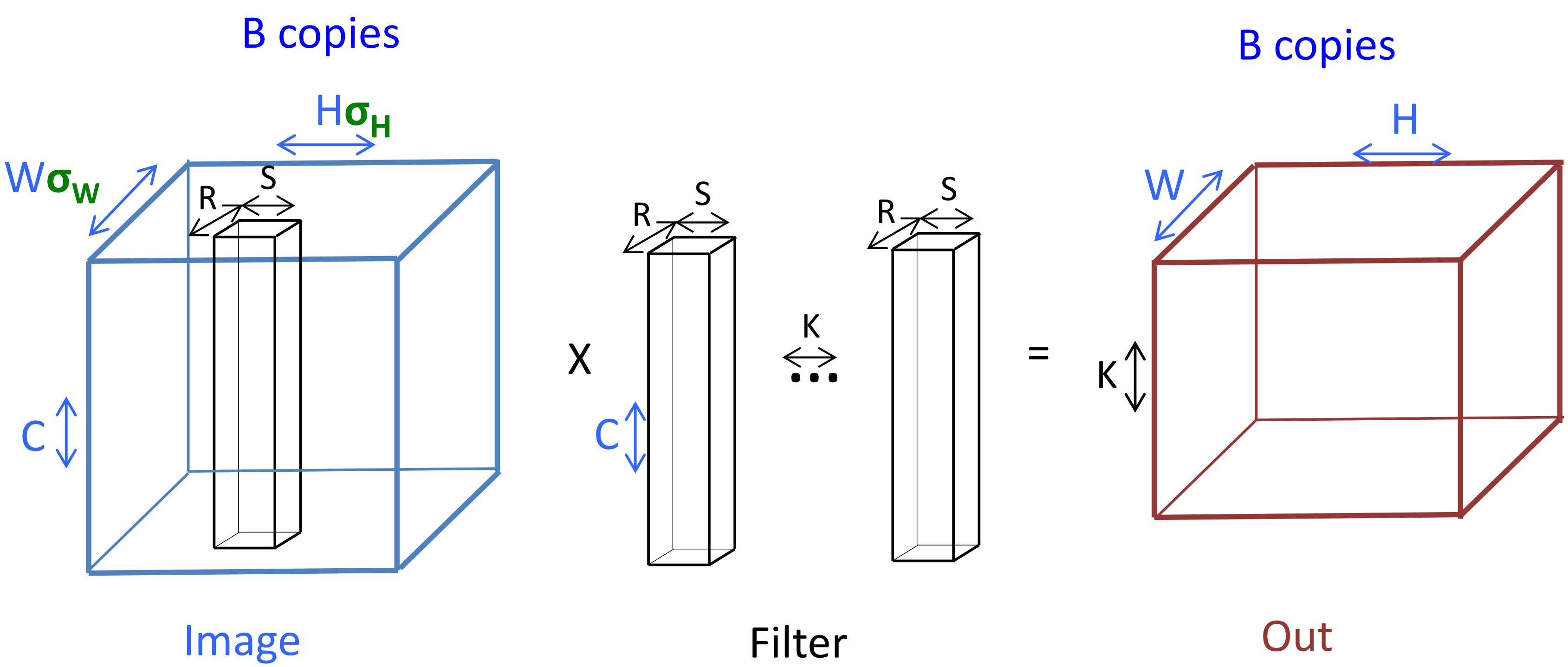


What CNNs compute




```
for k=1:K,    for h=1:H,    for w=1:W,    for r=1:R,
for s=1:S,     for c=1:C,    for b=1:B
      Out(k, h, w, b) += Image(r+w, s+h, c, b) * Filter( k, r, s, c )
```

What CNNs compute



for k=1:K, for h=1:H, for w=1:W, for r=1:R,
for s=1:S, for c=1:C, for b=1:B

$$Out(k, h, w, b) \mathrel{+}= Image(r+\sigma_W w, s+\sigma_H h, c, b) * Filter(k, r, s, c)$$

Communication Lower Bound for CNNs

- Let N = #iterations = KHWRSCB, M = cache size
- #words moved = Ω(max(… 5 terms

  BKHW, … size of Out

  $\sigma_H \sigma_W$BCWH, ... size of Image

  CKRS, ... size of Filter

  N/M, ... lower bound like N-body

  $N/(M^{1/2} (RS/(\sigma_H \sigma_W))^{1/2})$ ... new lower bound)
- New lower bound
  - Beats matmul by factor $(RS/(\sigma_H \sigma_W))^{1/2}$
  - Applies in common case when data does not fit in cache, but one RxS filter does
  - Tile needed to attain N/M too big to fit in loop bounds

Attaining lower bound

- Thm: Lower bound attainable, for all $K,H,W,R,S,C,B,M,\sigma_H,\sigma_W$
- Computer generated proof

  automatic code generation for all cases

Ongoing Work

- Implement/improve algorithms to generate for lower bounds, optimal algorithms
- Dealing with small loop bounds
  - What if "optimal" tile too large to use?
  - Ex: Tighter lower bound possible for CNNs
- Dealing with dependencies
  - What if "optimal" tile does not respect dependencies?
- Extend "perfect scaling" results for time and energy by using extra memory
  - "n.5D algorithms"
- Incorporate into compilers

Outline

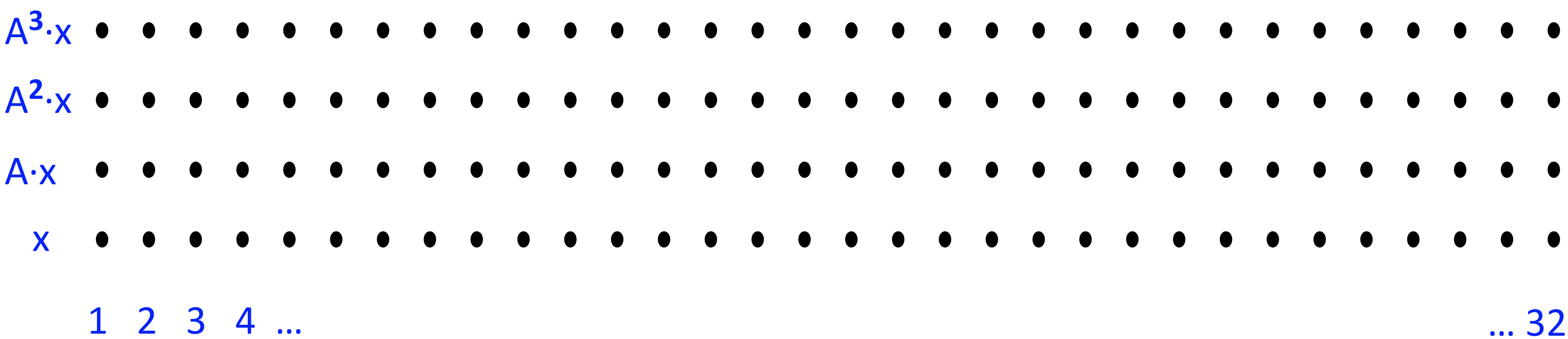
- Survey state of the art of CA (Comm-Avoiding) algorithms
  - Review previous Matmul algorithms
  - CA $O(n^3)$ 2.5D Matmul and LU
  - TSQR: Tall-Skinny QR
  - CA Strassen Matmul
- Beyond linear algebra
  - Extending lower bounds to any algorithm with arrays
  - Communication-optimal N-body and CNN algorithms
- CA-Krylov methods, and ML
- Related Topics

Avoiding Communication in Iterative Linear Algebra

- k-steps of iterative solver for sparse Ax=b or Ax=λx
  - Does k SpMVs with A and starting vector
  - Many such "Krylov Subspace Methods"
    - Conjugate Gradients (CG), GMRES, Lanczos, Arnoldi, …
- Goal: minimize communication
  - Assume matrix "well-partitioned"
  - Serial implementation
    - Conventional: O(k) moves of data from slow to fast memory
    - **New**: **O(1) moves of data – optimal**
  - Parallel implementation on p processors
    - Conventional: O(k log p) messages (k SpMV calls, dot prods)
    - **New: O(log p) messages - optimal**
- Lots of speed up possible (modeled and measured)
  - Price: some redundant computation
  - Challenges: Poor partitioning, Preconditioning, Num. Stability

Communication Avoiding Kernels:
The Matrix Powers Kernel : $[Ax, A^2x, \ldots, A^kx]$

- Replace k iterations of $y = A \cdot x$ with $[Ax, A^2x, \ldots, A^kx]$

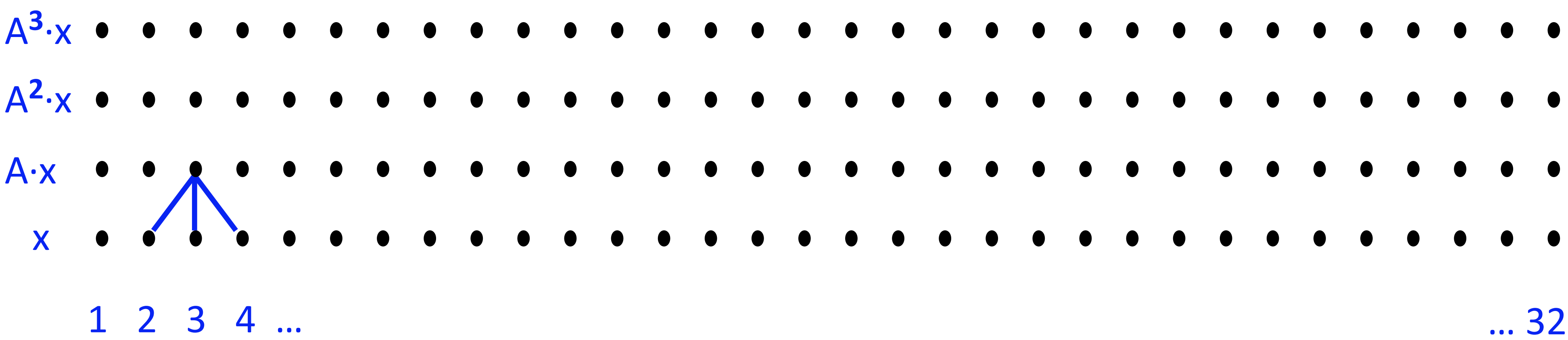


- Example: A tridiagonal, n=32, k=3
- Works for any "well-partitioned" A

Communication Avoiding Kernels:

The Matrix Powers Kernel : $[Ax, A^2x, \ldots, A^kx]$

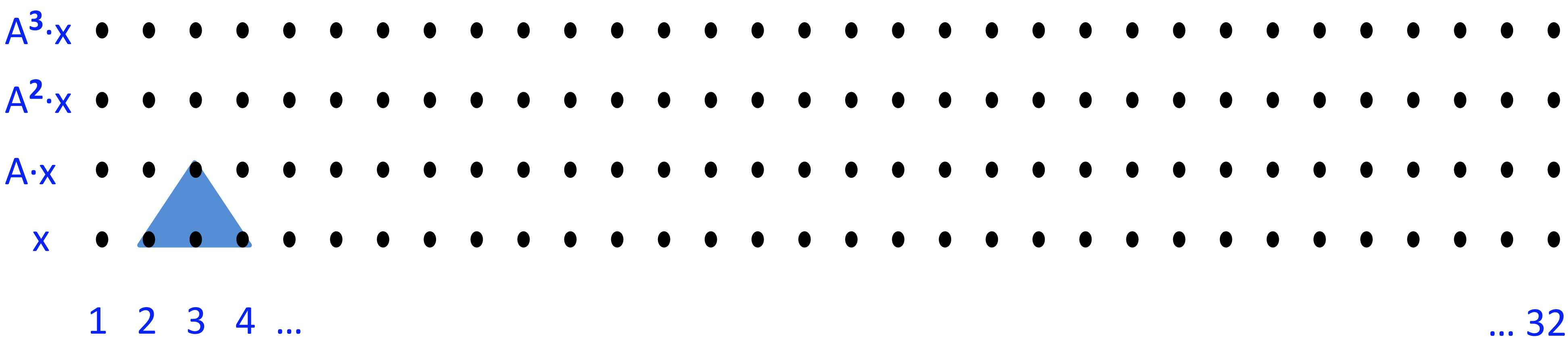
- Replace k iterations of $y = A \cdot x$ with $[Ax, A^2x, \ldots, A^kx]$



- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:
The Matrix Powers Kernel : [Ax, A^2x, …, A^kx]

- Replace k iterations of y = A·x with [Ax, A^2x, …, A^kx]

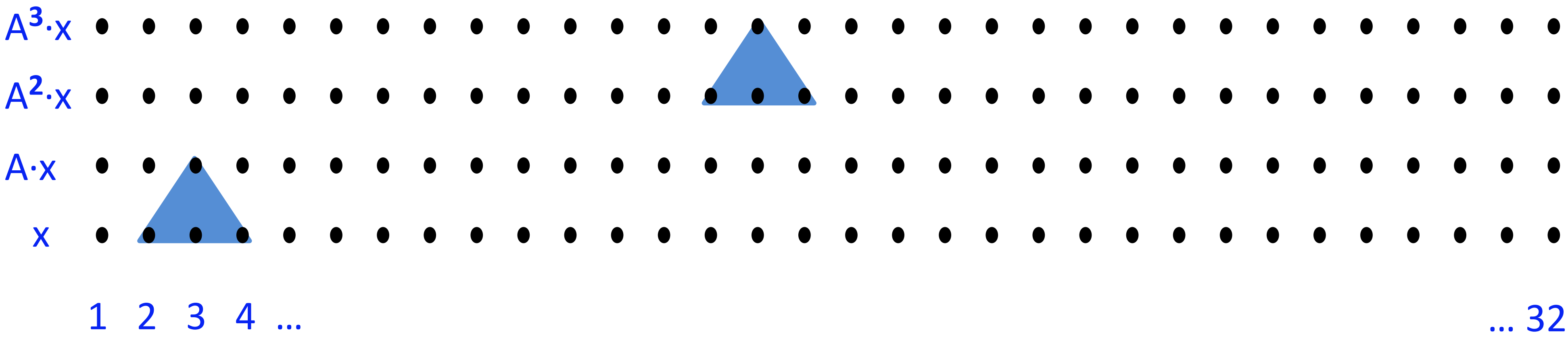


- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:

The Matrix Powers Kernel : $[Ax, A^2x, \ldots, A^kx]$

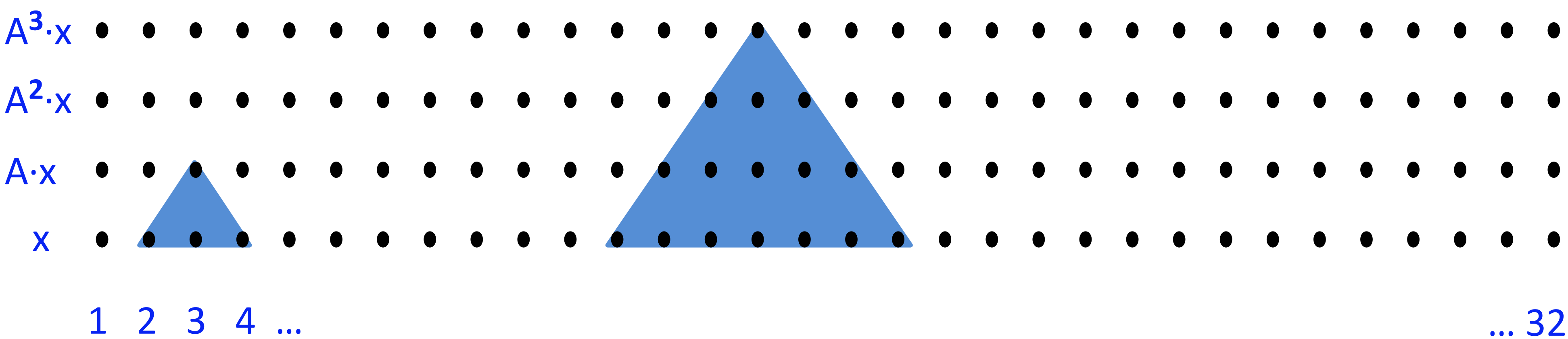
- Replace k iterations of $y = A \cdot x$ with $[Ax, A^2x, \ldots, A^kx]$



- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:
The Matrix Powers Kernel : $[Ax, A^2x, \ldots, A^kx]$

- Replace k iterations of $y = A \cdot x$ with $[Ax, A^2x, \ldots, A^kx]$

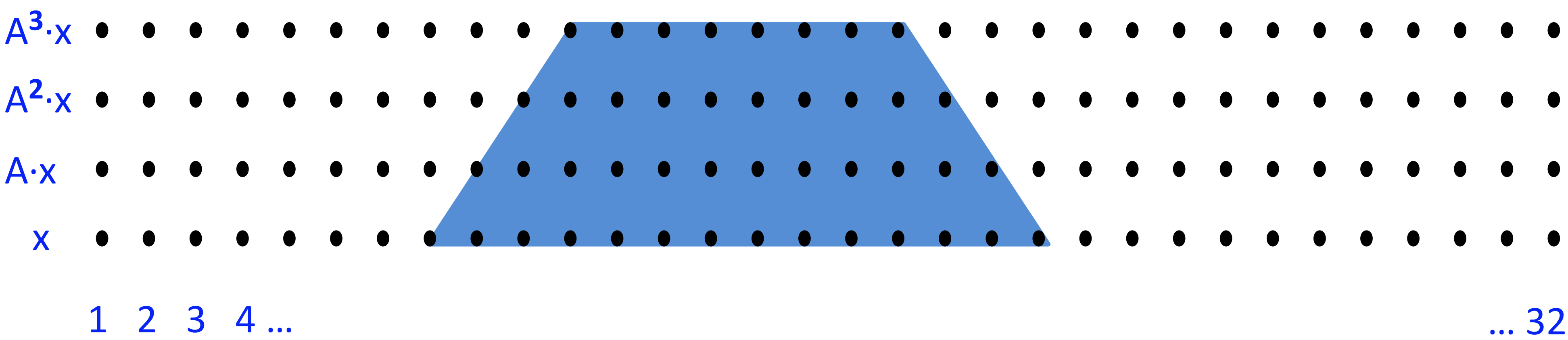


- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:

The Matrix Powers Kernel : $[Ax, A^2x, \ldots, A^kx]$

- Replace k iterations of $y = A \cdot x$ with $[Ax, A^2x, \ldots, A^kx]$

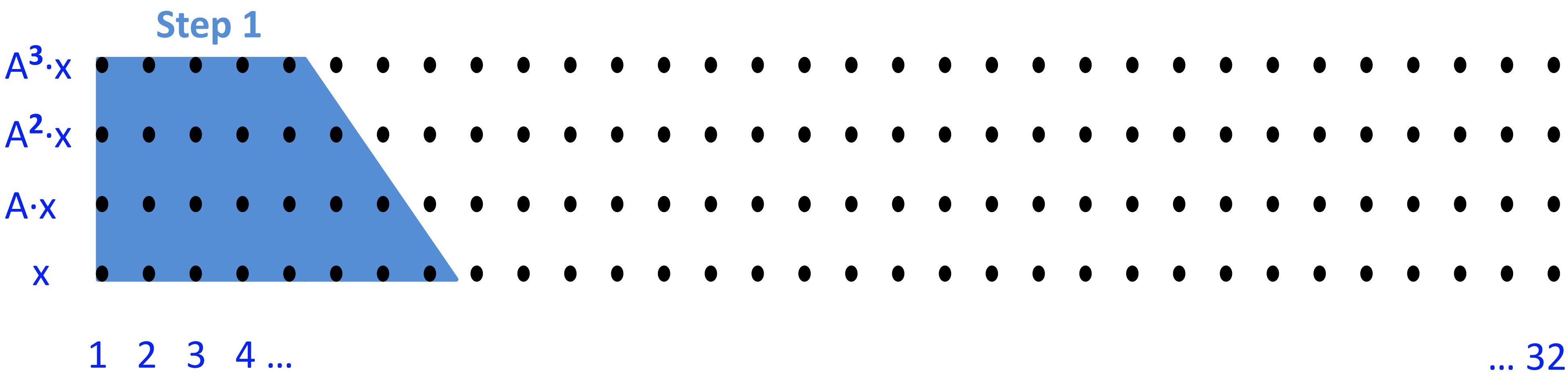


- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:

The Matrix Powers Kernel : $[Ax, A^2x, \ldots, A^kx]$

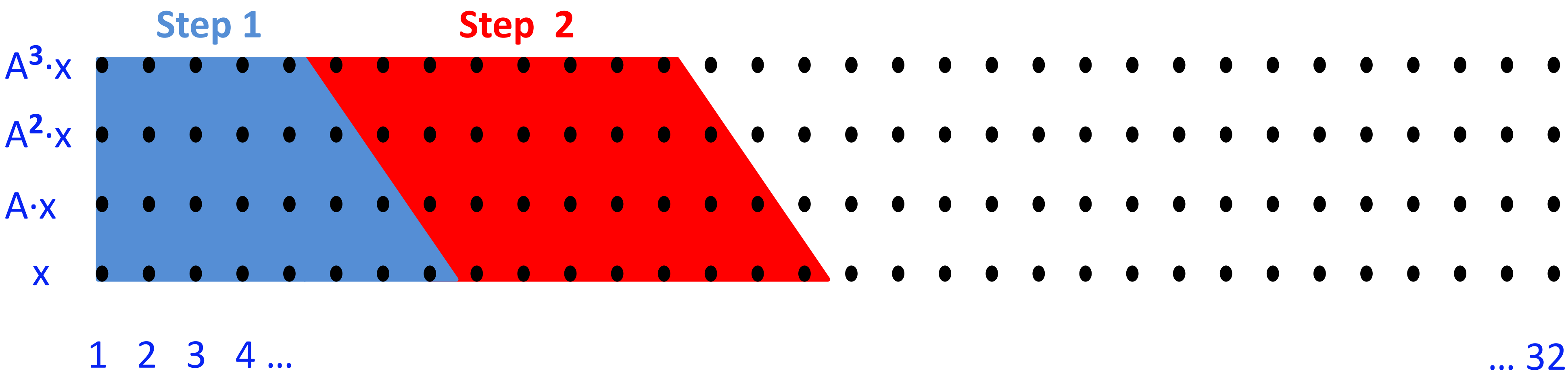
- Replace k iterations of $y = A \cdot x$ with $[Ax, A^2x, \ldots, A^kx]$
- Sequential Algorithm



- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:
The Matrix Powers Kernel : $[Ax, A^2x, \ldots, A^kx]$

- Replace k iterations of $y = A \cdot x$ with $[Ax, A^2x, \ldots, A^kx]$
- Sequential Algorithm

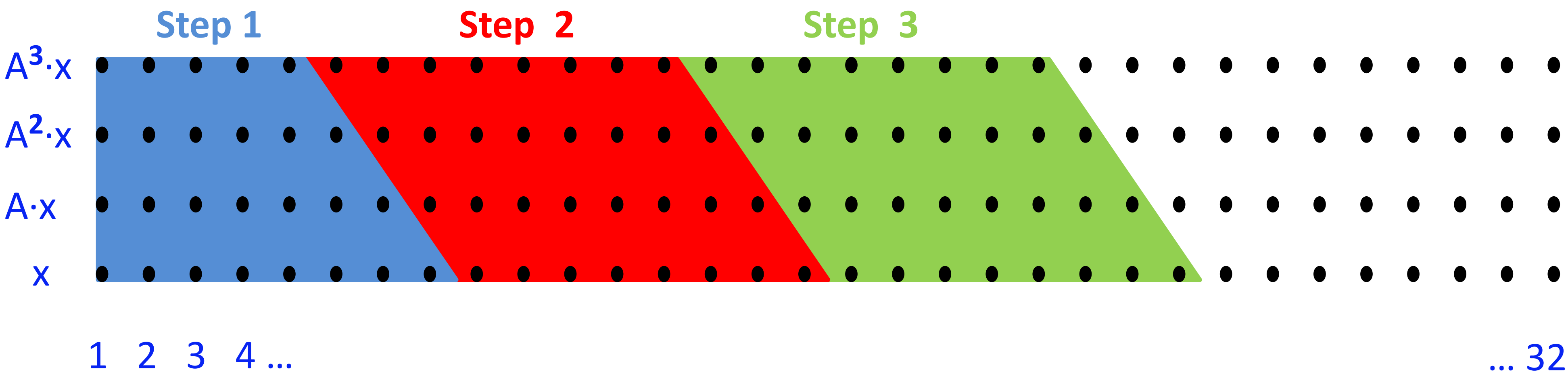


- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:

The Matrix Powers Kernel : $[Ax, A^2x, \ldots, A^kx]$

- Replace k iterations of $y = A \cdot x$ with $[Ax, A^2x, \ldots, A^kx]$
- Sequential Algorithm

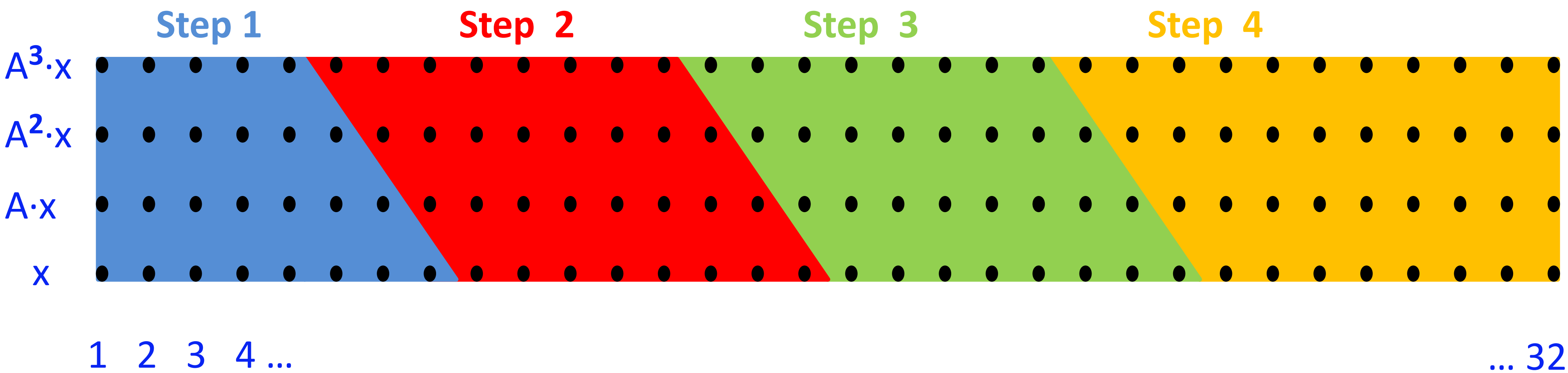


- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:

The Matrix Powers Kernel : $[Ax, A^2x, \ldots, A^kx]$

- Replace k iterations of $y = A \cdot x$ with $[Ax, A^2x, \ldots, A^kx]$
- Sequential Algorithm

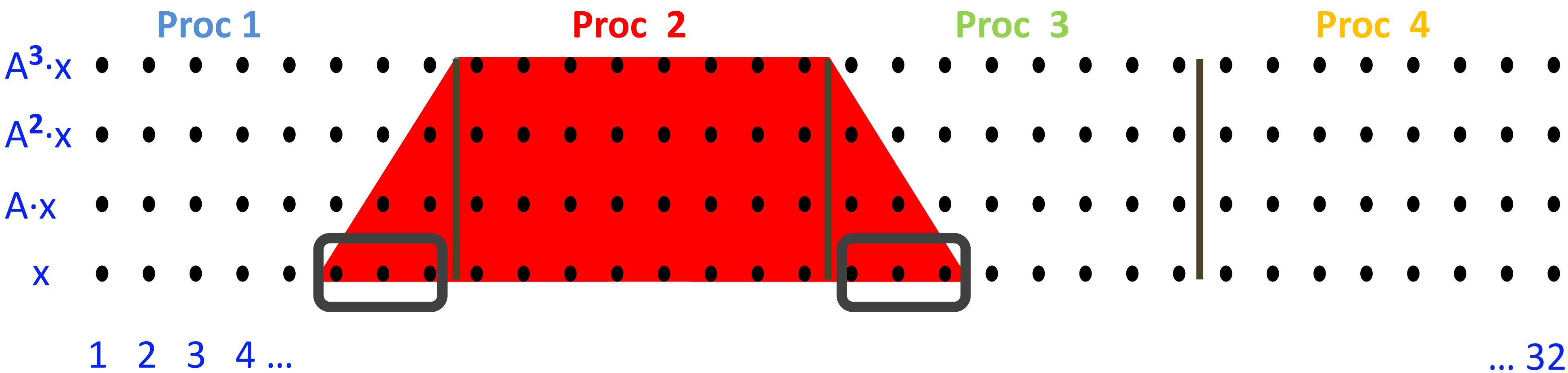


- Example: A tridiagonal, n=32, k=3

Communication Avoiding Kernels:

The Matrix Powers Kernel : $[Ax, A^2x, \ldots, A^kx]$

- Replace k iterations of $y = A \cdot x$ with $[Ax, A^2x, \ldots, A^kx]$
- Parallel Algorithm

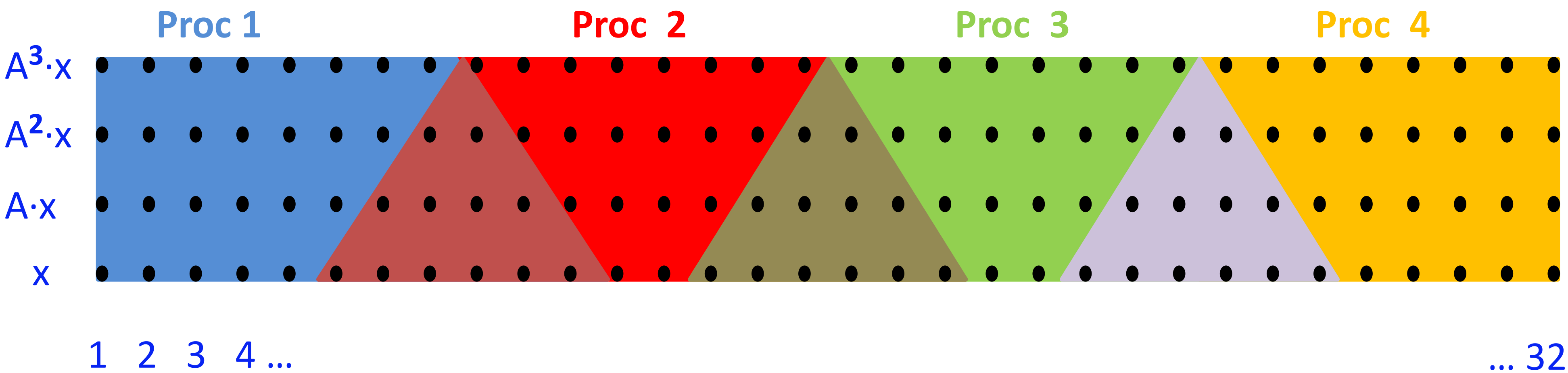


- Example: A tridiagonal, n=32, k=3
- Each processor communicates once with neighbors

Communication Avoiding Kernels:

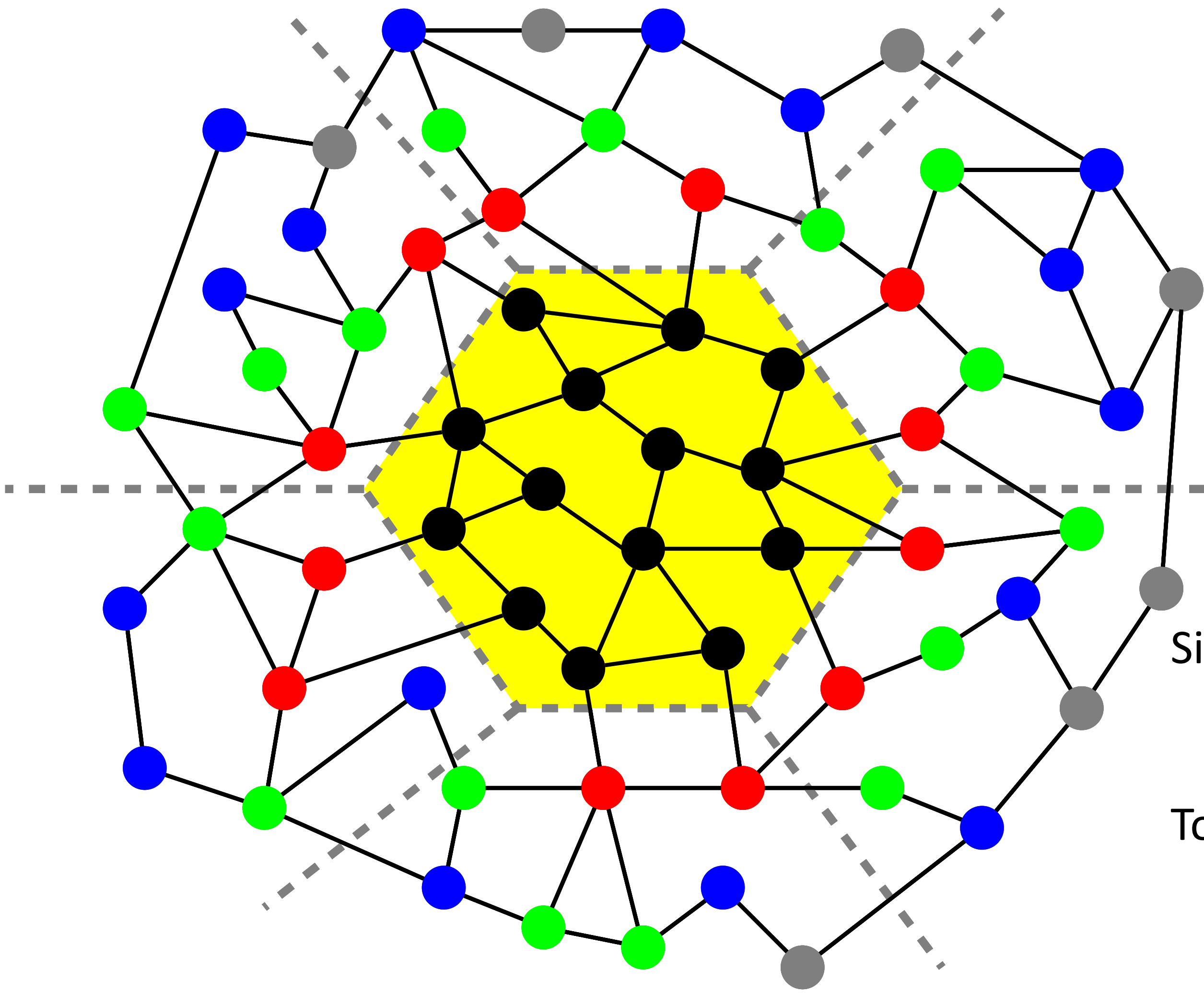
The Matrix Powers Kernel : $[Ax, A^2x, \ldots, A^kx]$

- Replace k iterations of $y = A \cdot x$ with $[Ax, A^2x, \ldots, A^kx]$
- Parallel Algorithm



- Example: A tridiagonal, n=32, k=3
- Each processor works on (overlapping) trapezoid

The Matrix Powers Kernel : $[Ax, A^2x, \ldots, A^kx]$ on a general matrix (nearest k neighbors on a graph)



Simple block-row partitioning → (hyper)graph partitioning

Top-to-bottom processing → Traveling Salesman Problem

Same idea for general sparse matrices: k-wide neighboring region

Minimizing Communication of GMRES to solve Ax=b

- GMRES: find x in span{b,Ab,…,A^kb} minimizing || Ax-b $||_2$

Standard GMRES

for i=1 to k

  w = A · v(i-1)   *… SpMV*

  MGS(w, v(0),…,v(i-1))

  update v(i), H

endfor

solve LSQ problem with H

Communication-avoiding GMRES

W = [v, Av, A^2v, … , A^kv]

[Q,R] = TSQR(W)

  … *"Tall Skinny QR"*

build H from R

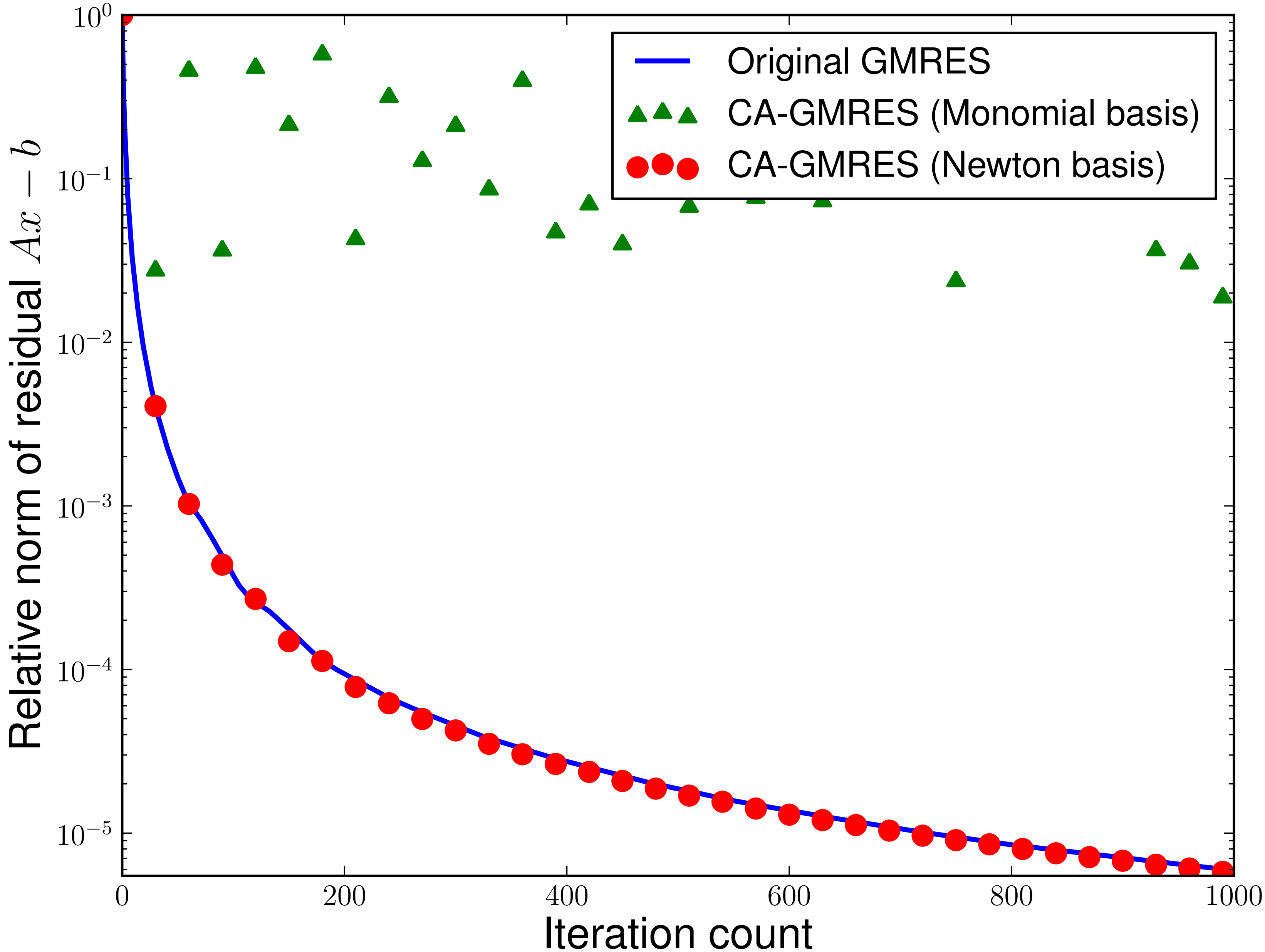
solve LSQ problem with H

Sequential case: #words moved decreases by a factor of k

Parallel case: #messages decreases by a factor of k

- Oops – W from power method, precision lost!

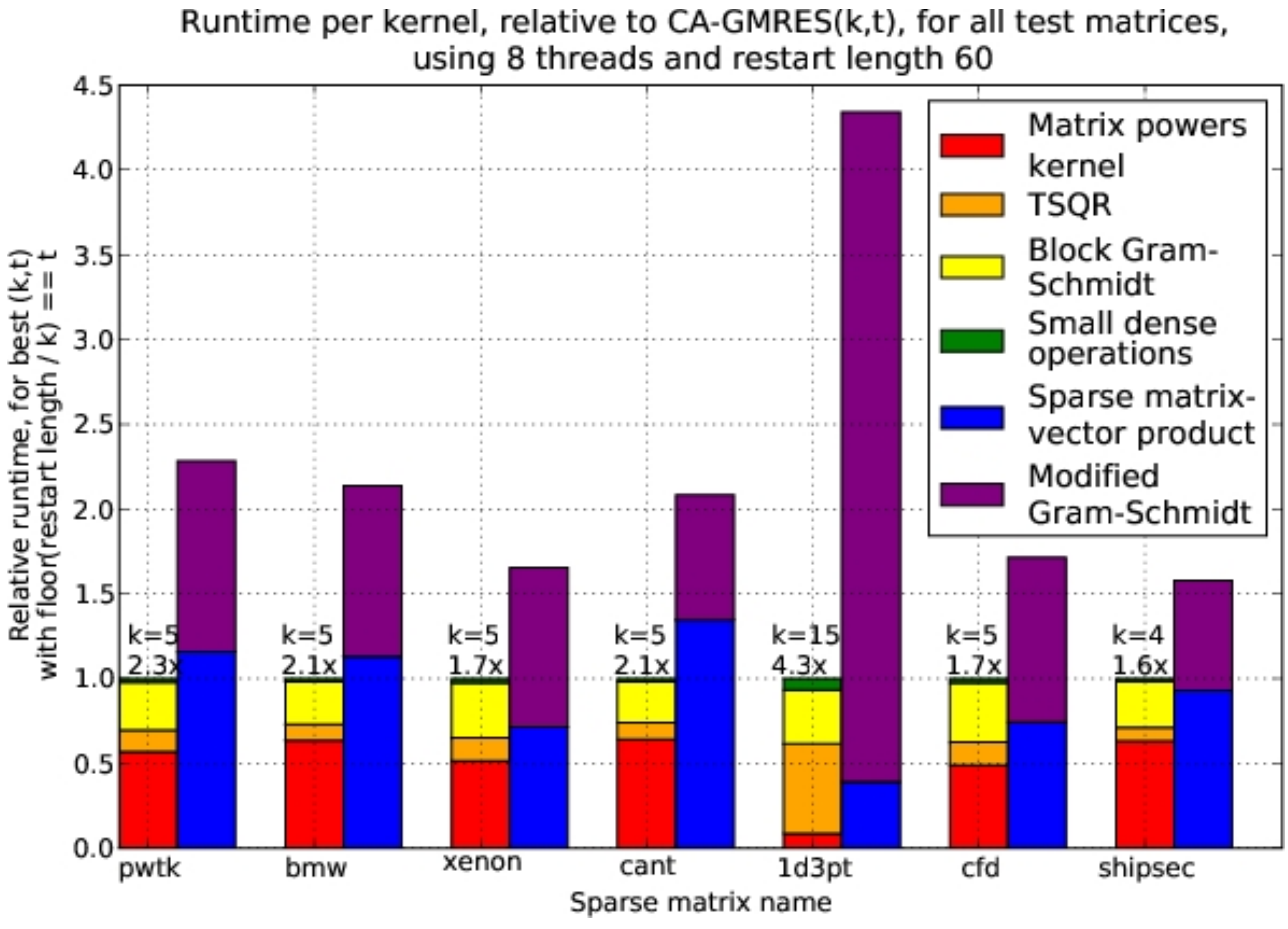
Matrix Powers Kernel + TSQR in GMRES



Speed ups of GMRES on 8-core Intel Clovertown

Requires Co-tuning Kernels

[MHDY09]



CA-BiCGStab

Compute $r_0 = b - Ax_0$. Choose $r_0^\star$ arbitrary.
Set $p_0 = r_0$, $q_{-1} = 0_{N\times 1}$.
For $k = 0, 1, \ldots,$ until convergence, Do

$P = [p_{sk},\ Ap_{sk},\ \ldots,\ A^s p_{sk}]$
$Q = [q_{sk-1},\ Aq_{sk-1},\ \ldots,\ A^s q_{sk-1}]$
$R = [r_{sk},\ Ar_{sk},\ \ldots,\ A^s r_{sk}]$

//Compute the $1 \times (3s+3)$ Gram vector.
$g = (r_0^\star)^T\,[P,\ Q,\ R]$
//Compute the $(3s+3)\times(3s+3)$ Gram matrix

$$G = \begin{bmatrix} P^T \\ Q^T \\ R^T \end{bmatrix} \begin{bmatrix} P & Q & R \end{bmatrix}$$

For $\ell = 0$ to s,

$$b_{sk}^{\ell} = \left[B_1(:,\ell)^T,\ 0_{s+1}^T,\ 0_{s+1}^T\right]^T$$
$$c_{sk-1}^{\ell} = \left[0_{s+1}^T,\ B_2(:,\ell)^T,\ 0_{s+1}^T\right]^T$$
$$d_{sk}^{\ell} = \left[0_{s+1}^T,\ 0_{s+1}^T,\ B_3(:,\ell)^T\right]^T$$

For $j = 0$ to $\lfloor \frac{s}{2} \rfloor - 1$, Do

$$\alpha_{sk+j} = \frac{<g,\ d_{sk+j}^0>}{<g,\ b_{sk+j}^1>}$$
$$q_{sk+j} = r_{sk+j} - \alpha_{sk+j}[P,\ Q,\ R]b_{sk+j}^1$$

For $\ell = 0$ to $s - 2j + 1$, Do

$$c_{sk+j}^{\ell} = d_{sk+j}^{\ell} - \alpha_{sk+j} b_{sk+j-1}^{\ell+1}$$

//such that $[P,\ Q,\ R]\,c_{sk+j}^{\ell} = A^{\ell} q_{sk+j}$

$$\omega_{sk+j} = \frac{<c_{sk+j+1}^1,\ Gc_{sk+j+1}^0>}{<c_{sk+j+1}^1,\ Gc_{sk+j+1}^1>}$$
$$x_{sk+j+1} = x_{sk+j} + \alpha_{sk+j}p_{sk+j} + \omega_{sk+j}q_{sk+j}$$
$$r_{sk+j+1} = q_{sk+j} - \omega_{sk+j}[P,\ Q,\ R]c_{sk+j+1}^1$$

For $\ell = 0$ to $s - 2j$, Do

$$d_{sk+j+1}^{\ell} = c_{sk+j+1}^{\ell} - \omega_{sk+j} c_{sk+j+1}^{\ell+1}$$

//such that $[P,\ Q,\ R]\,d_{sk+j+1}^{\ell} = A^{\ell} r_{sk+j+1}$

$$\beta_{sk+j} = \frac{<g,\ d_{sk+j+1}^0>}{<g,\ d_{sk+j}^0>} \times \frac{\alpha}{\omega}$$
$$p_{sk+j+1} = r_{sk+j+1} + \beta_{sk+j}p_{sk+j} - \beta_{sk+j}\omega_{sk+j}[P,\ Q,\ R]b_{sk+j}^1$$

For $\ell = 0$ to $s - 2j$, Do

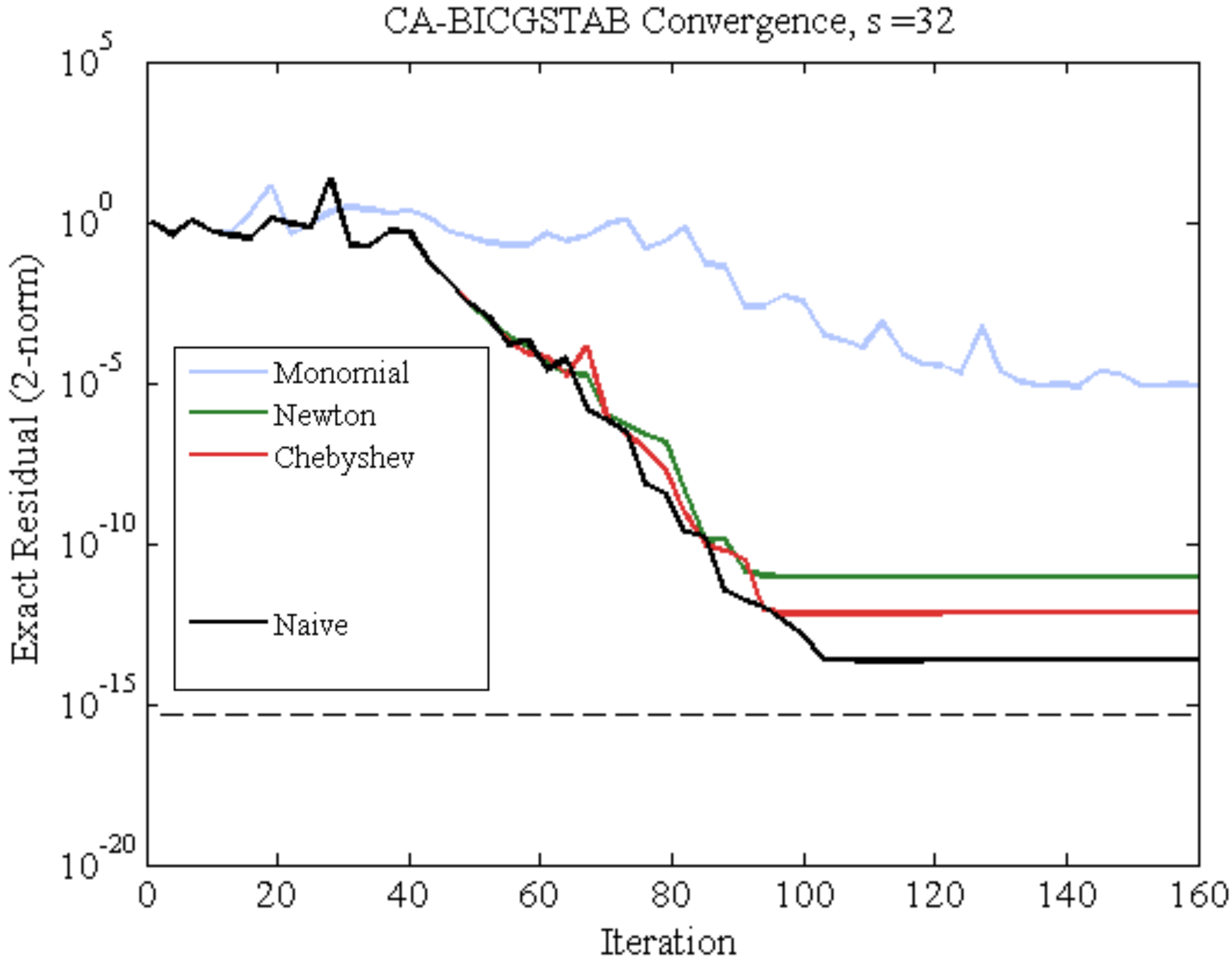
$$b_{sk+j+1}^{\ell} = d_{sk+j+1}^{\ell} + \beta_{sk+j} b_{sk+j}^{\ell} - \beta_{sk+j}\omega_{sk+j} b_{sk+j}^{\ell+1}$$

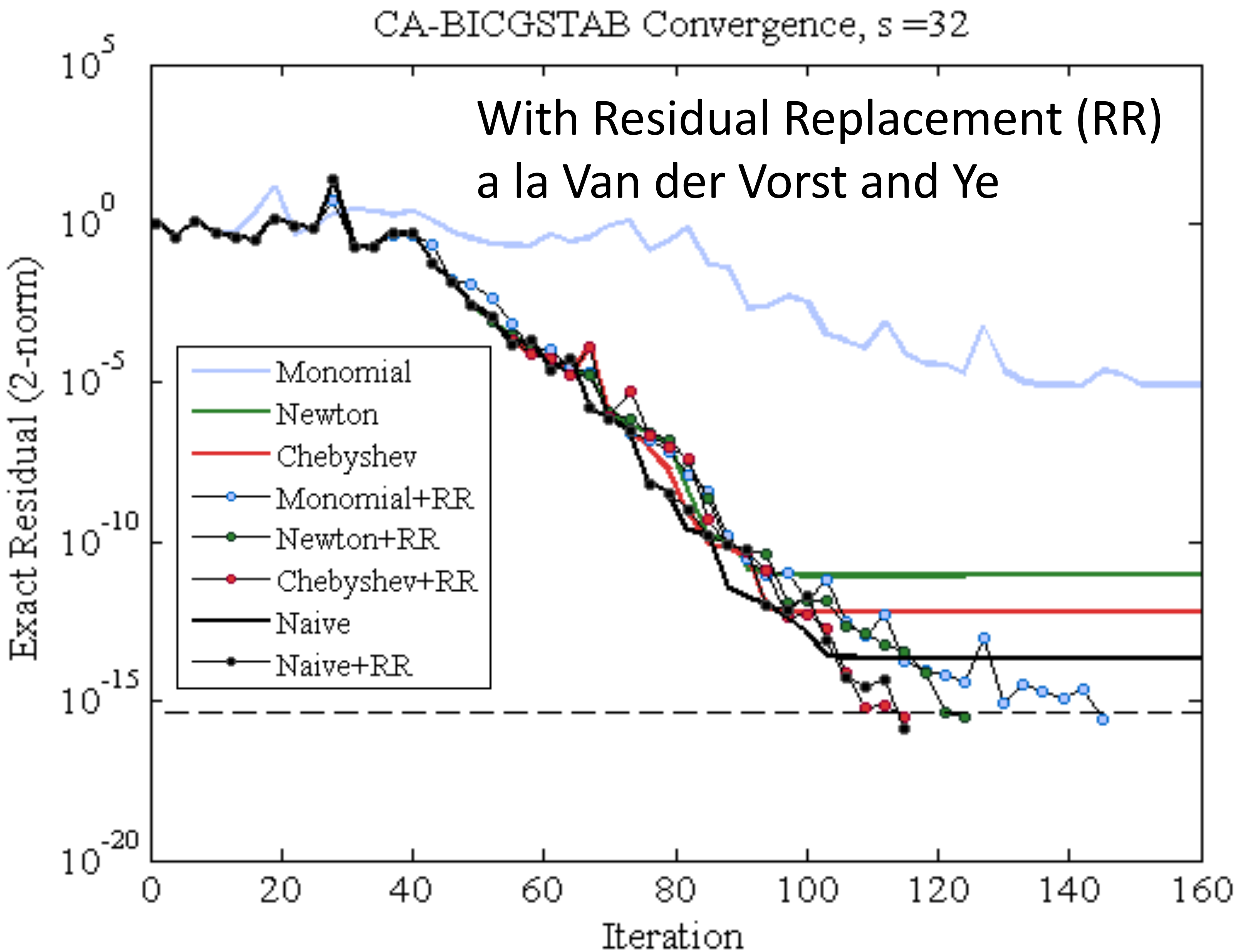
//such that $[P,\ Q,\ R]\,b_{sk+j+1}^{\ell} = A^{\ell} p_{sk+j+1}$.

EndDo

EndDo

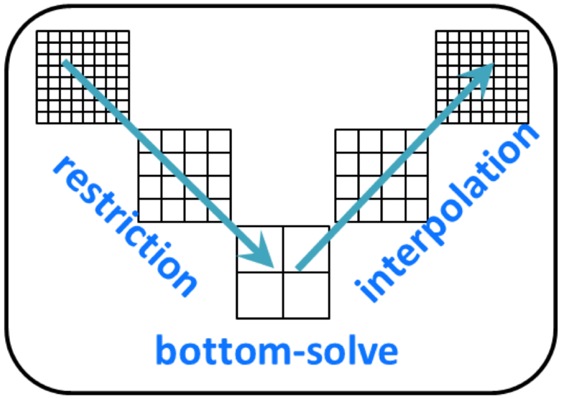
1. Compute $r_0 := b - Ax_0$; r_0^* arbitrary;
2. $p_0 := r_0$.
3. For $j = 0, 1, \ldots,$ until convergence Do:
4. $\alpha_j := \boxed{(r_j, r_0^*)/(\boxed{Ap_j}, r_0^*)}$
5. $s_j := r_j - \alpha_j Ap_j$
6. $\omega_j := \boxed{(\boxed{As_j}, s_j)/(As_j, As_j)}$
7. $x_{j+1} := x_j + \alpha_j p_j + \omega_j s_j$
8. $r_{j+1} := s_j - \omega_j As_j$
9. $\beta_j := \boxed{\frac{(r_{j+1}, r_0^*)}{(r_j, r_0^*)}} \times \frac{\alpha_j}{\omega_j}$
10. $p_{j+1} := r_{j+1} + \beta_j(p_j - \omega_j Ap_j)$
11. EndDo


CA-BICGSTAB Convergence, s =32
10^{5}
10^{0}
10^{-5}
10^{-10}
10^{-15}
10^{-20}
Exact Residual (2-norm)
0
20
40
60
80
100
120
140
160
Iteration
Monomial
Newton
Chebyshev
Naive



	Naive	Monomial	Newton	Chebyshev
Replacement Its.	74 **(1)**	[7, 15, 24, 31, …, 92, 97, 103] **(17)**	[67, 98] **(2)**	68 **(1)**

Speedups for GMG w/CA-KSM Bottom Solve



- Compared **BICGSTAB** vs. **CA-BICGSTAB** with s = 4 (monomial basis)
- Hopper at NERSC (Cray XE6), weak scaling: Up to 4096 MPI processes (1 per chip, 24,576 cores total)
- Speedups for miniGMG benchmark (HPGMG benchmark predecessor)
  – **4.2x** in bottom solve, **2.5x** overall GMG solve
- Implemented as a solver option in BoxLib and CHOMBO AMR frameworks
- Speedups for two BoxLib applications:
  – 3D LMC (a low-mach number combustion code)
    - **2.5x** in bottom solve, **1.5x** overall GMG solve
  – 3D Nyx (an N-body and gas dynamics code)
    - **2x** in bottom solve, **1.15x** overall GMG solve

Communication-Avoiding Machine Learning: CAML

- CA-technique extends to other iterative ML methods
- Coordinate descent of the minimization problem:

$$\underset{\alpha \in \mathbb{R}^n}{\operatorname{argmin}} \frac{1}{2n} \|\alpha + y\|_2^2 + \frac{\lambda}{2} \left\| \frac{1}{\lambda n} X\alpha \right\|_2^2$$

CD algorithm

Until convergence do:

1. Randomly select a data point, x_i
2. Solve minimization problem for x_i
3. Update solution vector

Dot products and axpys

$$Flops = O\left(\frac{Hd}{P}\right),$$
$$Messages = O(H \log P),$$
$$Words = O(H)$$

Communication-Avoiding Coordinate Descent

CA-CD algorithm

Until convergence do:

1. Randomly select s data points
2. Compute Gram matrix
3. Solve minimization problem for all data points
4. Update solution vector

GEMM, dot products, and axpys

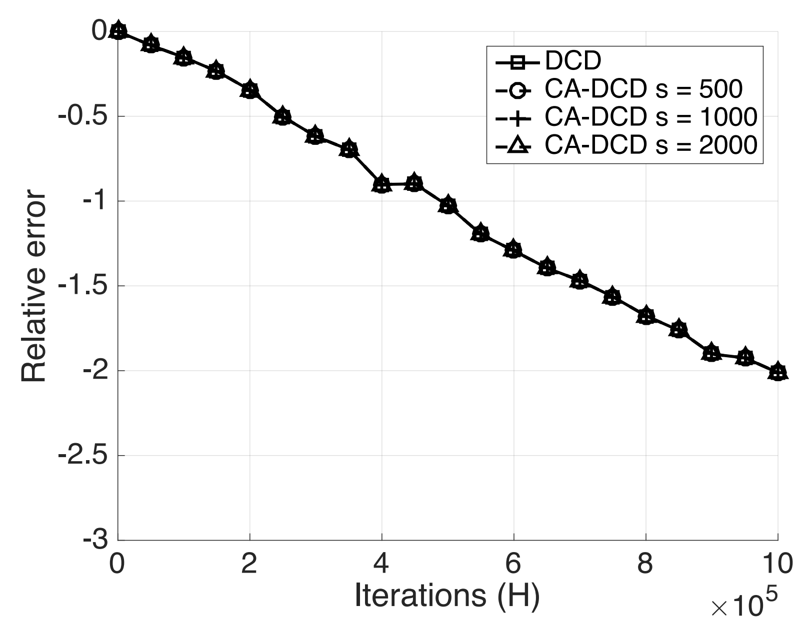
- We expect 1st flops term to dominate

- MPI: choose s that balances cost
- Spark: choose large s to minimize rounds
- Parallel implementations in progress
  - Up to **5.1x** speedup on Cray XC30 for LASSO

$$Flops = O\left(\frac{Hsd}{P} + Hs\right),$$

$$Messages = O\left(\frac{H}{s}\log P\right),$$

$$Words = O(Hs)$$



Numerically stable for (very) large s

Summary of Iterative Linear Algebra

- New lower bounds, optimal algorithms, big speedups in theory and practice
- Lots of other progress, open problems
  - Many different algorithms reorganized
    - More underway, more to be done
  - Need to recognize stable variants more easily
  - Preconditioning
    - Hierarchically Semiseparable Matrices
  - Autotuning and synthesis
    - Different kinds of “sparse matrices”
  - More extensions to Machine Learning

Outline

- Survey state of the art of CA (Comm-Avoiding) algorithms
  - Review previous Matmul algorithms
  - CA $O(n^3)$ 2.5D Matmul and LU
  - TSQR: Tall-Skinny QR
  - CA Strassen Matmul
- Beyond linear algebra
  - Extending lower bounds to any algorithm with arrays
  - Communication-optimal N-body and CNN algorithms
- CA-Krylov methods, and ML
- Related Topics
  - Write-Avoiding Algorithms
  - Reproducibility

Collaborators and Supporters

- **James Demmel**, **Kathy Yelick**, Aditya Devarakonda, Grace Dinh, Michael Driscoll, Penporn Koanantakool, Alex Rusciano
- Peter Ahrens, Michael Anderson, Grey Ballard, Austin Benson, Erin Carson, Maryam Dehnavi, David Eliahu, Andrew Gearhart, Evangelos Georganas, Mark Hoemmen, Shoaib Kamil, , Nicholas Knight, Ben Lipshitz, Marghoob Mohiyuddin, Hong Diep Nguyen, Jason Riedy, Oded Schwartz, Edgar Solomonik, Omer Spillinger
- Abhinav Bhatele, Aydin Buluc, Michael Christ, Ioana Dumitriu, Armando Fox, David Gleich, Ming Gu, Jeff Hammond, Mike Heroux, Olga Holtz, Kurt Keutzer, Julien Langou, Xiaoye Li, Devin Matthews, Tom Scanlon, Michelle Strout, Sam Williams, Hua Xiang
- Jack Dongarra, Mark Gates, Jakub Kurzak, Dulceneia Becker, Ichitaro Yamazaki, …
- Sivan Toledo, Alex Druinsky, Inon Peled
- Greg Henry, Peter Tang,
- Laura Grigori**,** Sebastien Cayrols, Simplice Donfack, Mathias Jacquelin, Amal Khabou, Sophie Moufawad, Mikolaj Szydlarski
- Members of ASPIRE, BEBOP, ParLab, CACHE, EASI, FASTMath, MAGMA, PLASMA
- Thanks to DOE, NSF, UC Discovery, INRIA, Intel, Microsoft, Mathworks, National Instruments, NEC, Nokia, NVIDIA, Samsung, Oracle
- bebop.cs.berkeley.edu

For more details

- Bebop.cs.berkeley.edu
  - 155 page survey in Acta Numerica (2014)
- CS267 – Berkeley's Parallel Computing Course
  - Live broadcast in Spring 2018
    - www.cs.berkeley.edu/~demmel
    - All slides, video available
  - Prerecorded version broadcast since Spring 2013
    - www.xsede.org
    - Free supercomputer accounts to do homework
    - Free autograding of homework

Summary

Time to redesign all linear algebra, n-body, …

algorithms and software

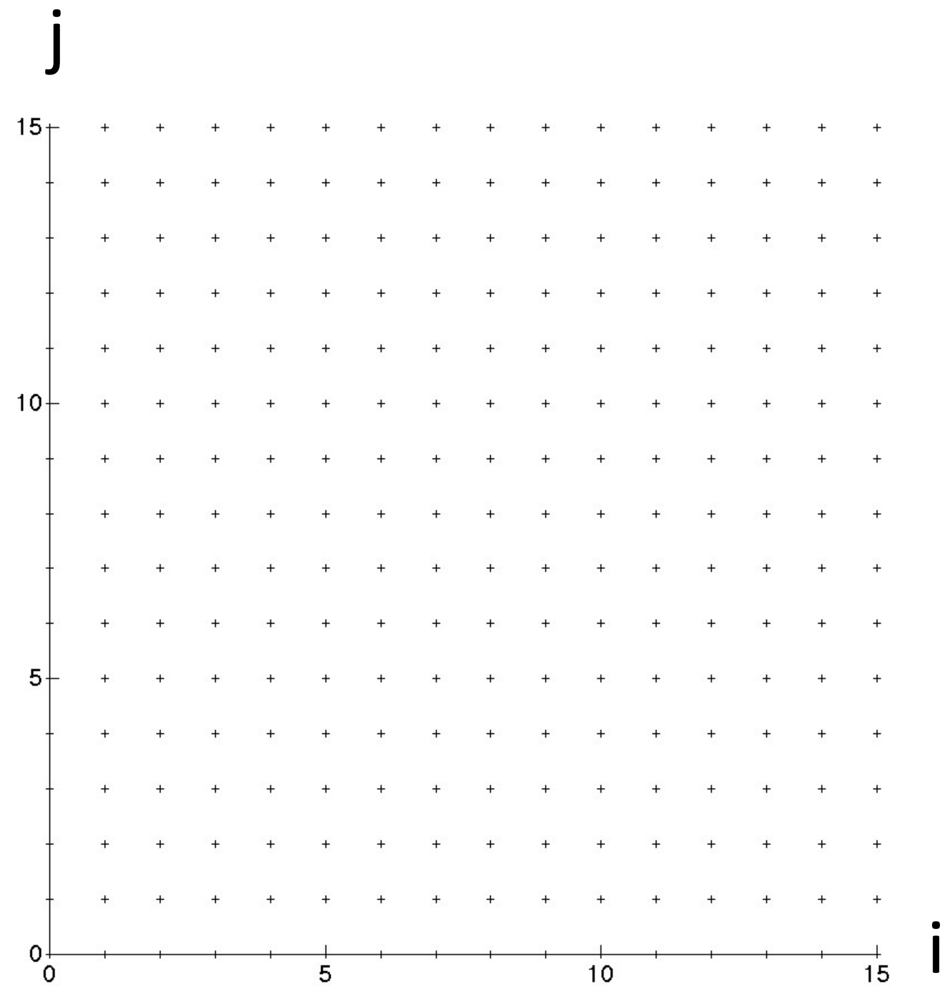
(and compilers)

Don’t Communic…

Back up slides

Optimal tiling for usual n-body

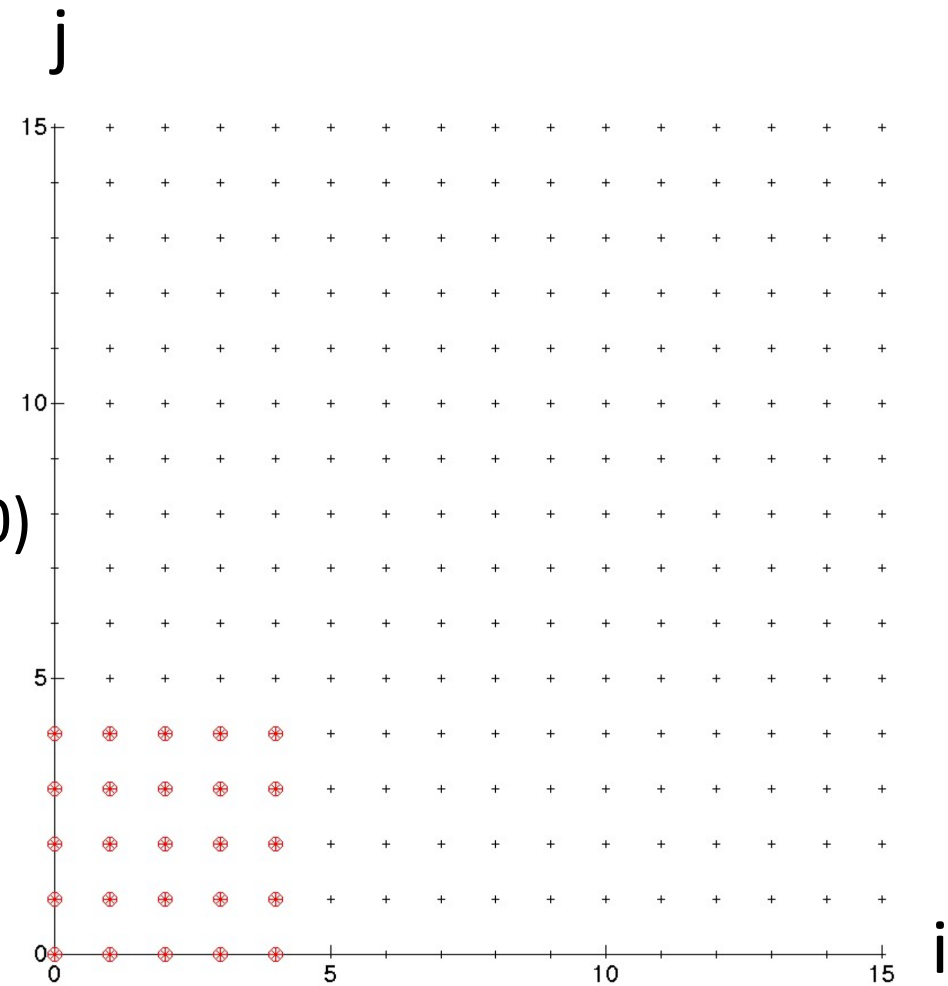
```
for i = 0:n
  for j = 0:n
    access A(i), B(j)
```



Optimal tiling for usual n-body

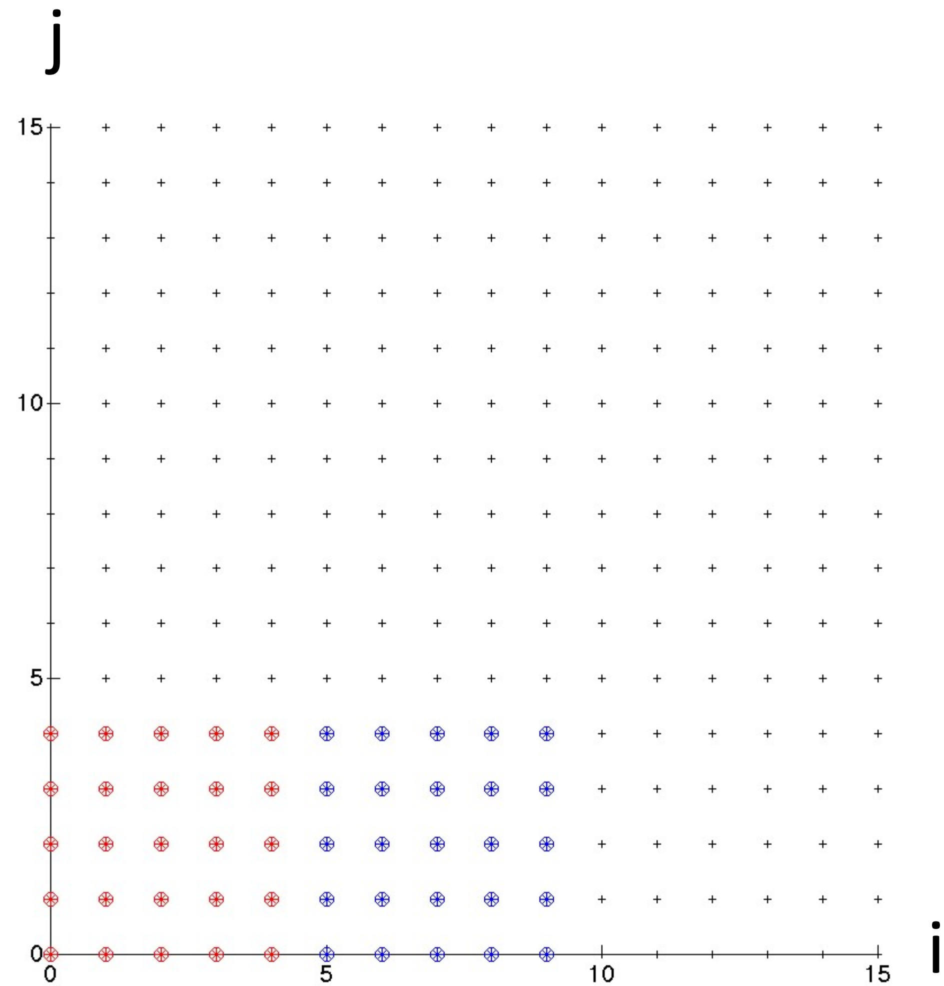
for i = 0:n
  for j = 0:n
    access A(i), B(j)

Tiling (for cache size 10)
  Read 5 entries of A:
    A([0,1,2,3,4])
  Read 5 entries of B:
    B([0,1,2,3,4])
  Perform 5^2 = 25
    loop iterations



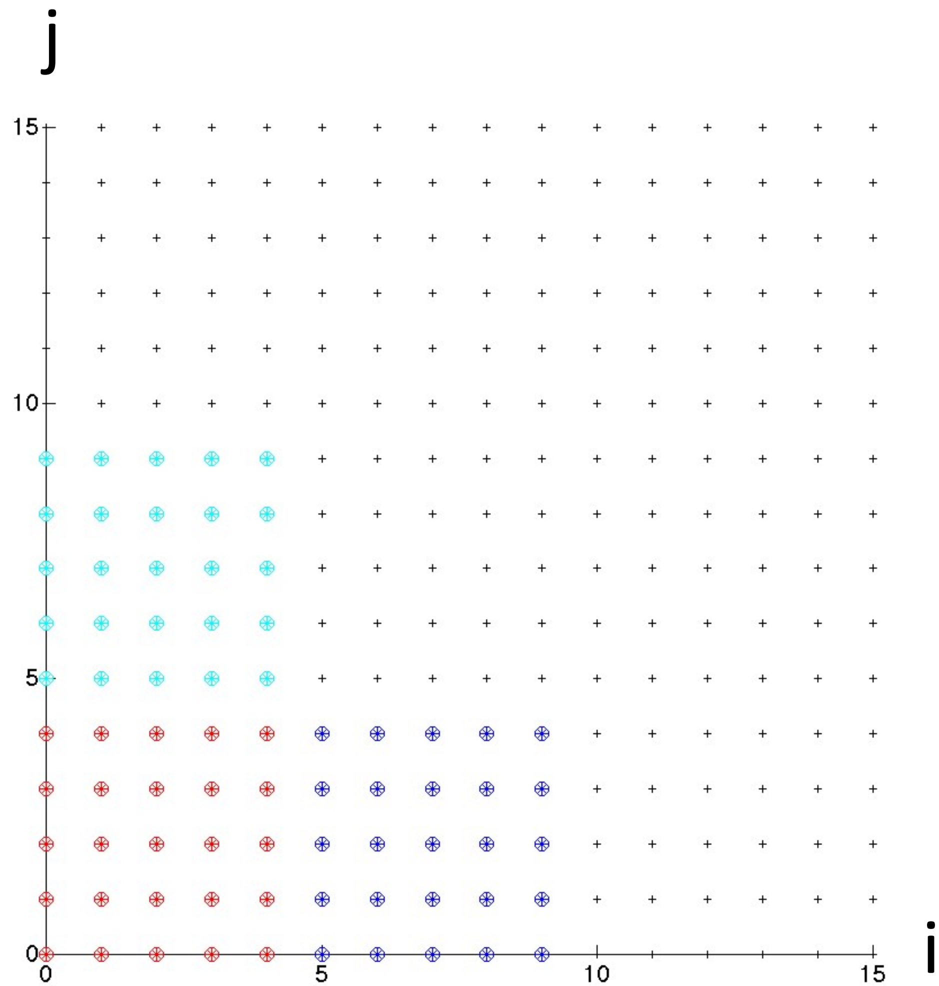
Optimal tiling for usual n-body

```
for i = 0:n
  for j = 0:n
    access A(i), B(j)
```



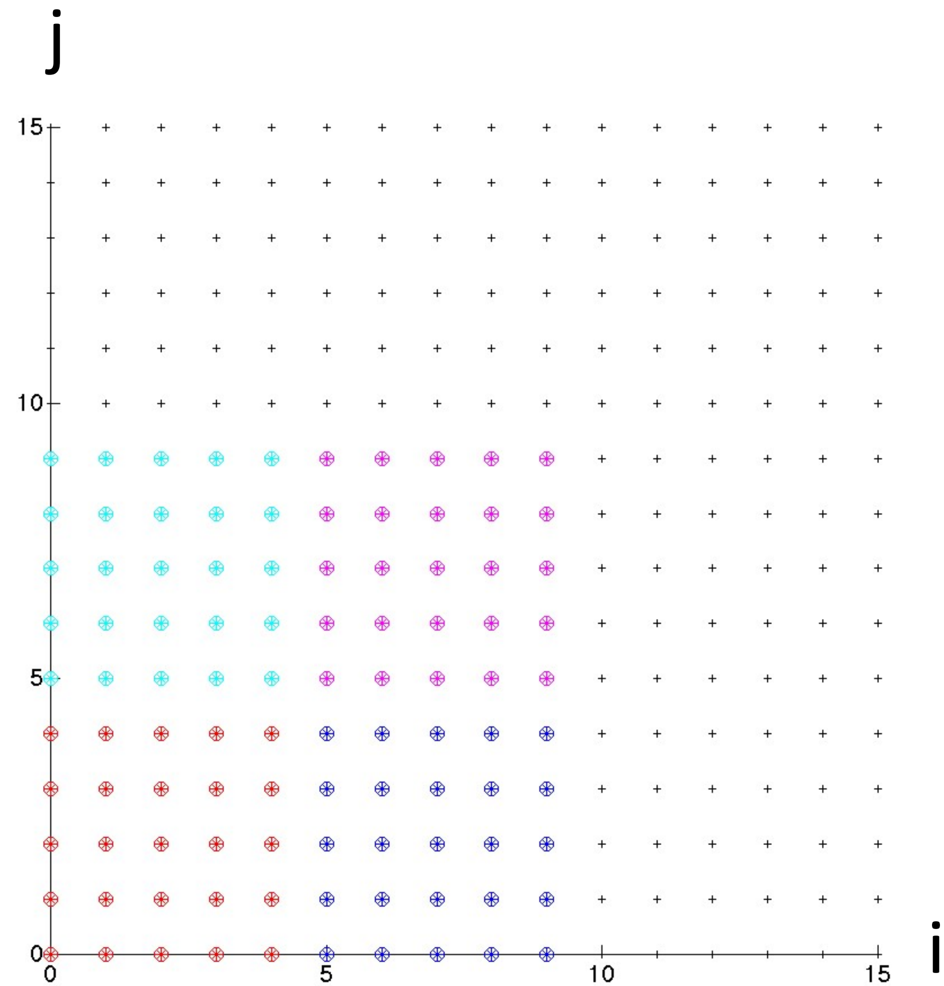
Optimal tiling for usual n-body

```
for i = 0:n
   for j = 0:n
      access A(i), B(j)
```



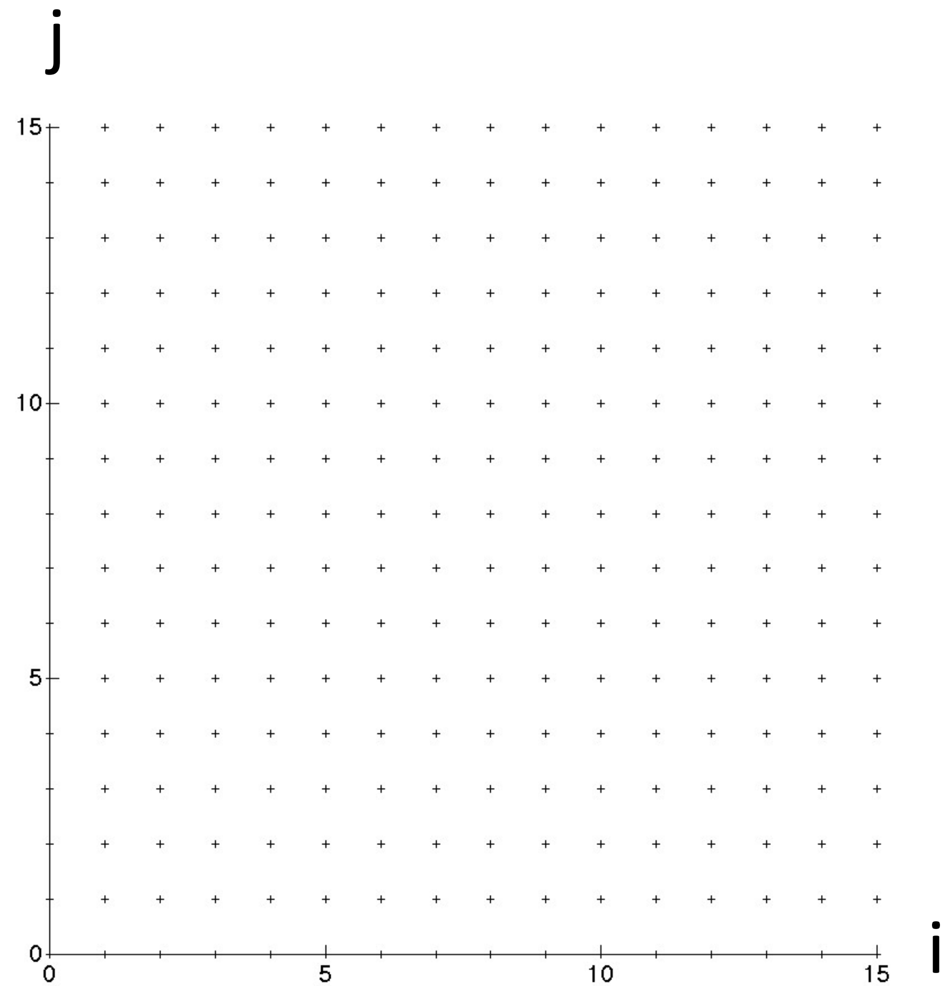
Optimal tiling for usual n-body

```
for i = 0:n
  for j = 0:n
    access A(i), B(j)
```



Optimal tiling for "slanted" n-body

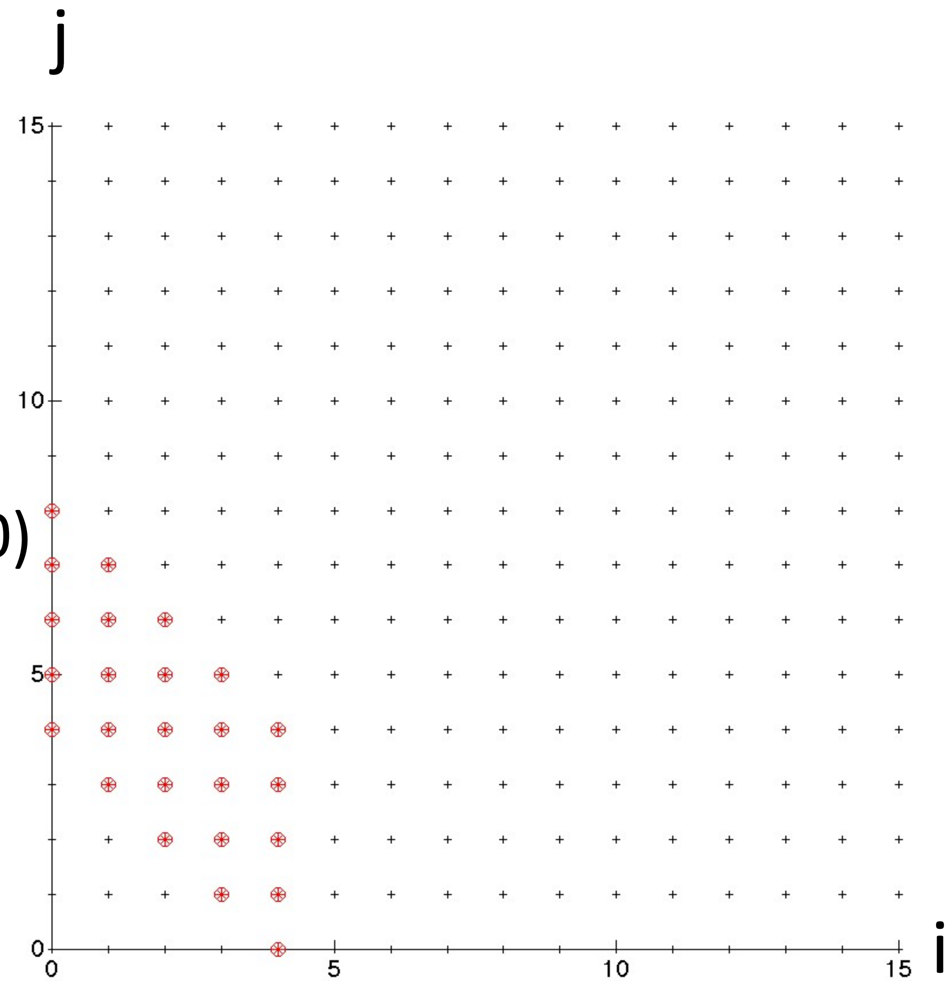
for i = 0:n
  for j = 0:n
    access A(i), B(**<u>i+j</u>**)



Optimal tiling for “slanted” n-body

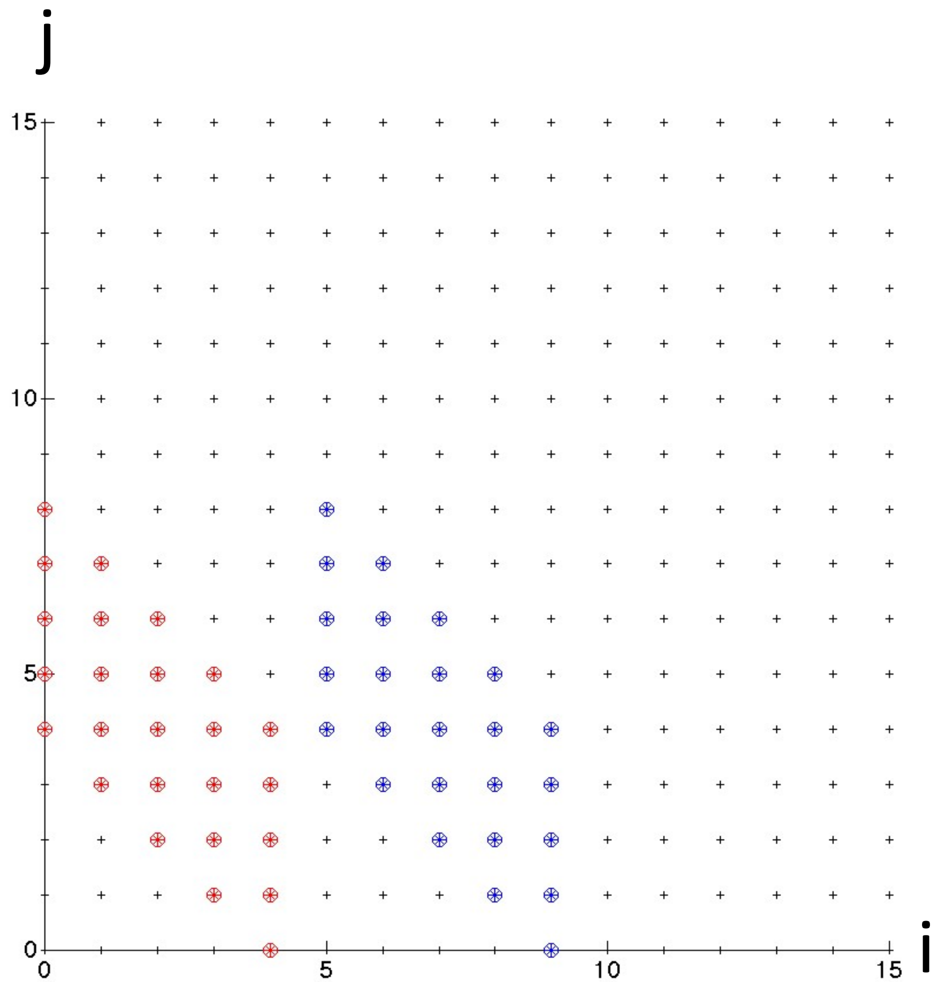
for i = 0:n
  for j = 0:n
    access A(i), B(i+j)

Tiling (for cache size 10)
  Read 5 entries of A:
    A([0,1,2,3,4])
  Read 5 entries of B:
    B([4,5,6,7,8])
  Perform 5^2 = 25
    loop iterations



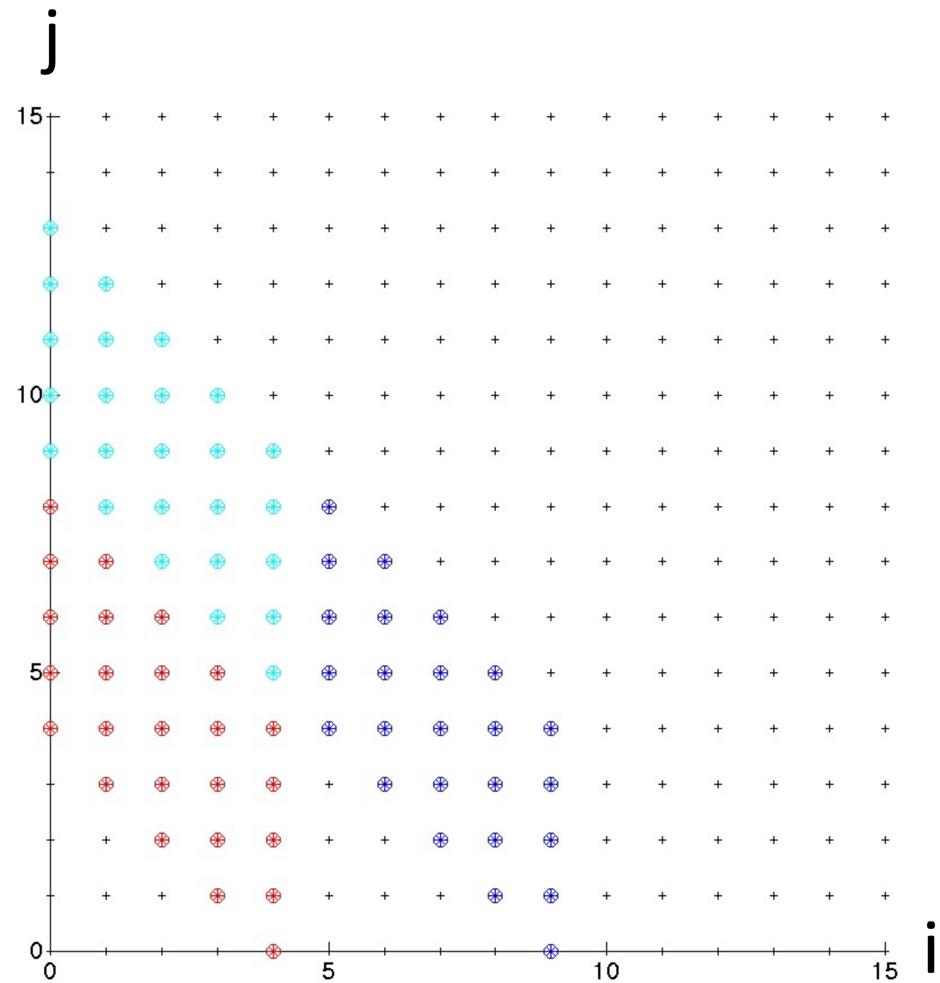
Optimal tiling for “slanted” n-body

```
for i = 0:n
  for j = 0:n
    access A(i), B(i+j)
```



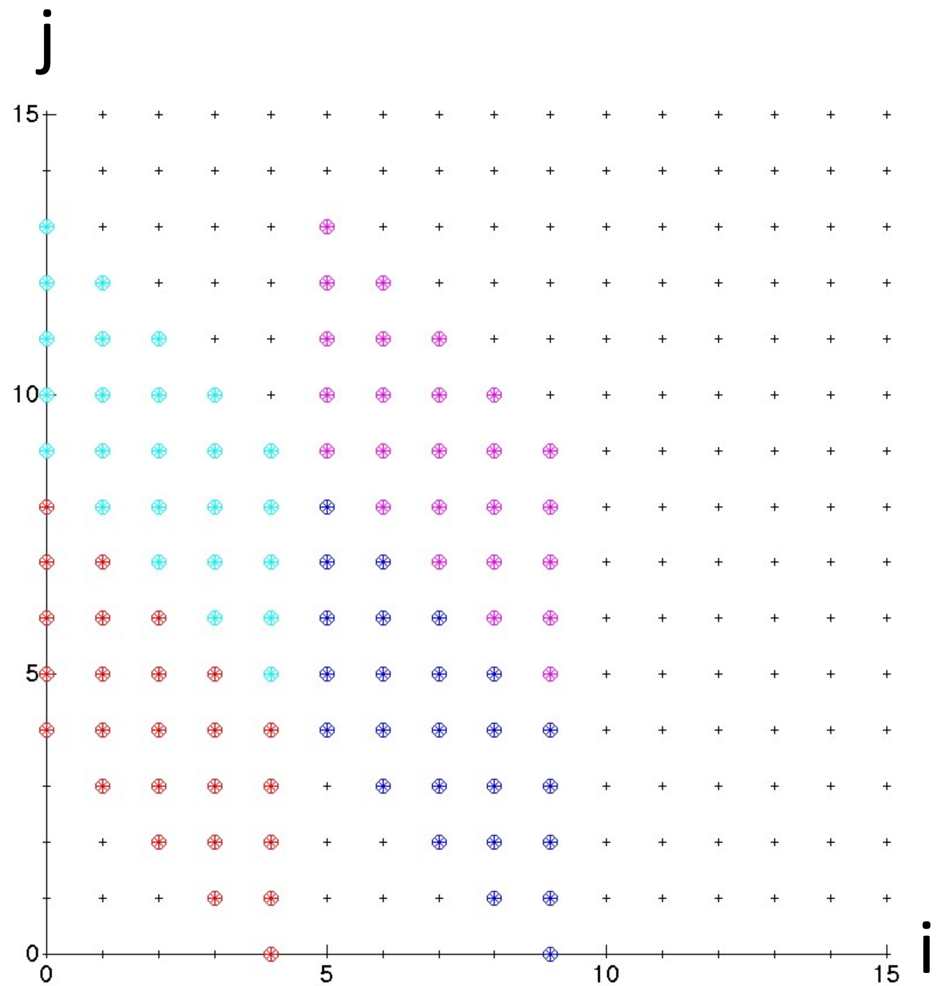
Optimal tiling for “slanted” n-body

```
for i = 0:n
  for j = 0:n
    access A(i), B(i+j)
```



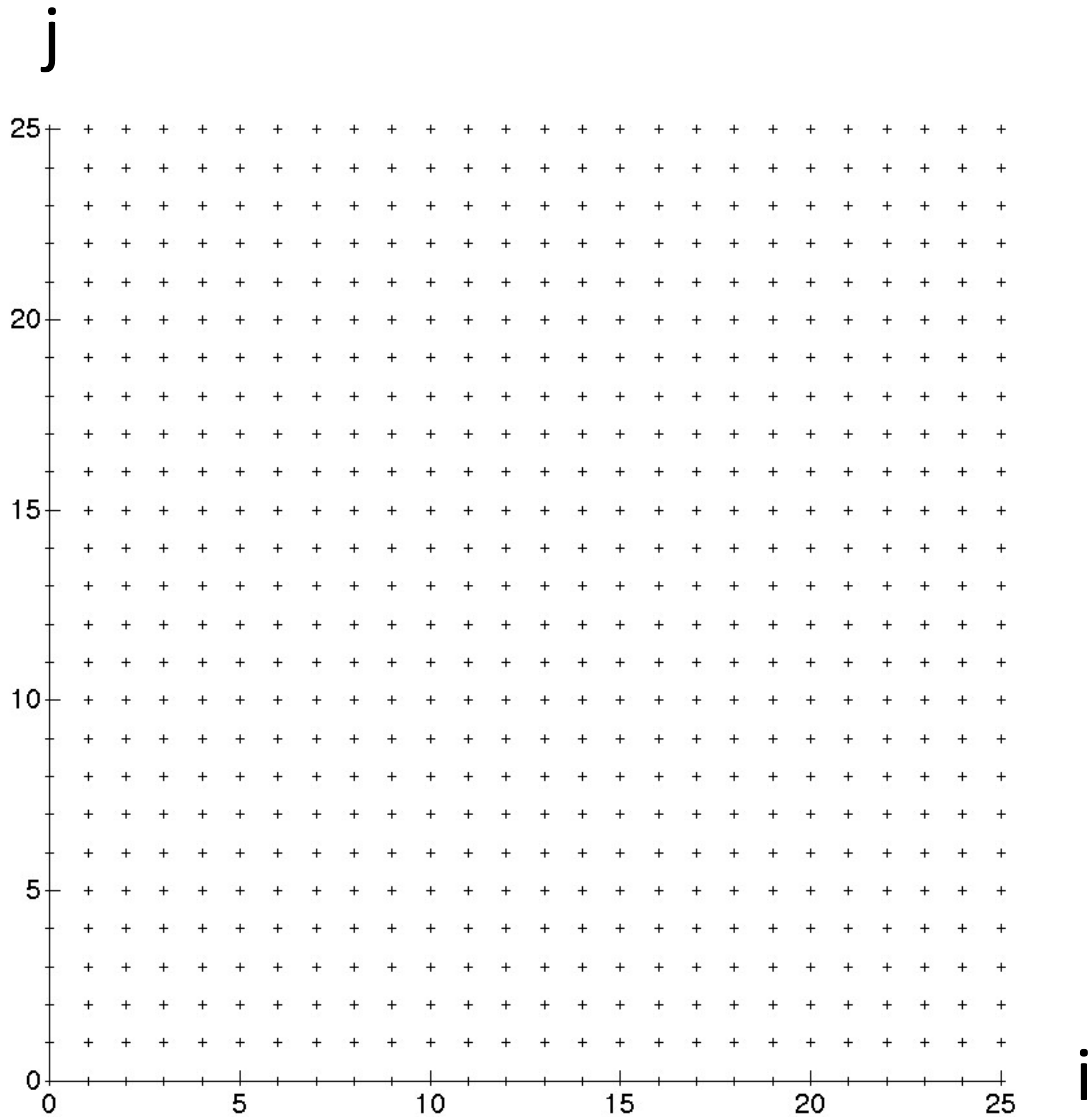
Optimal tiling for “slanted” n-body

```
for i = 0:n
  for j = 0:n
    access A(i), B(i+j)
```



Optimal tiling for “twisted” n-body

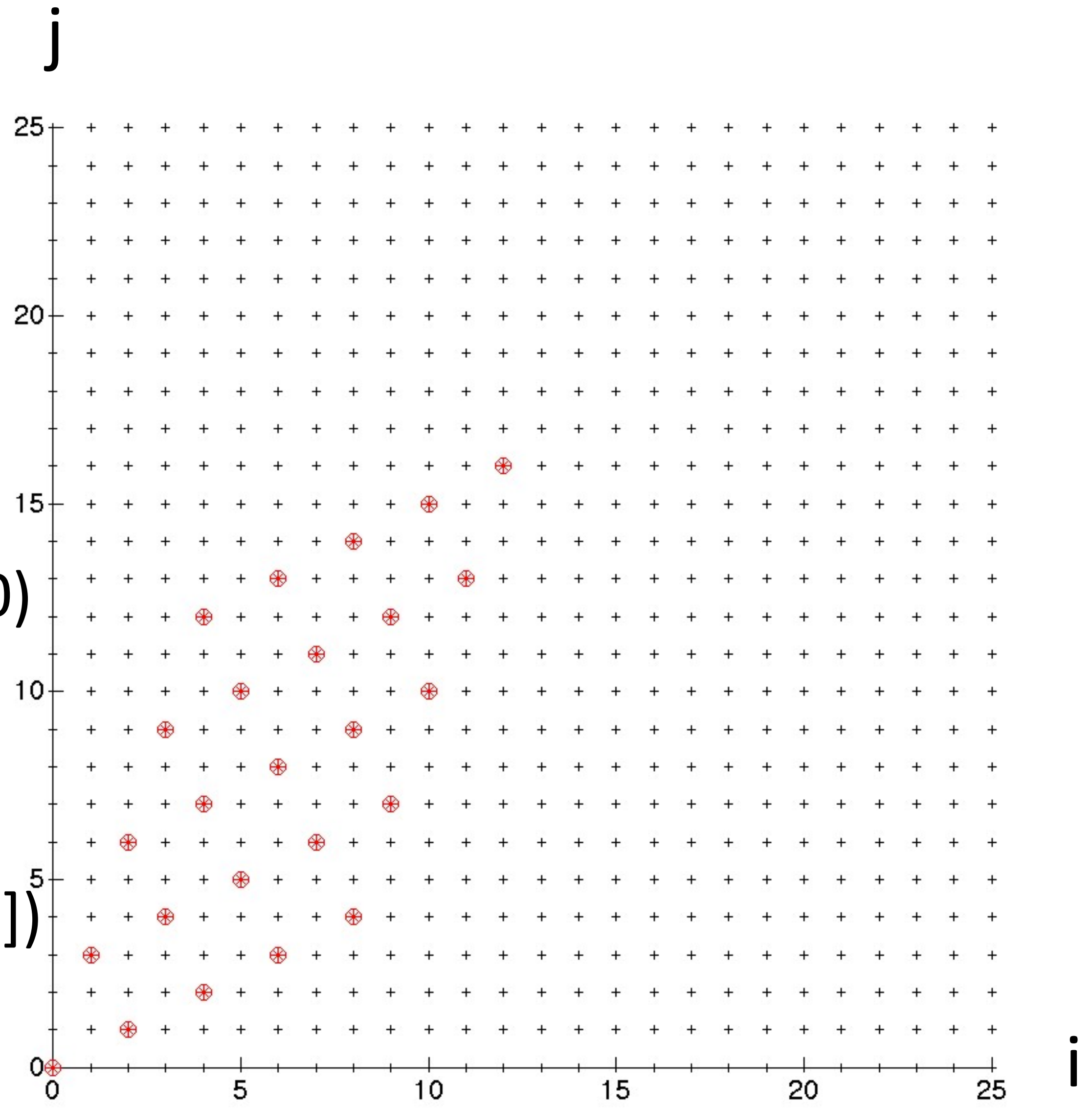
```
for i = 0:n
  for j = 0:n
    access A(3*i-j),
           B(i-2*j)
```



Optimal tiling for "twisted" n-body

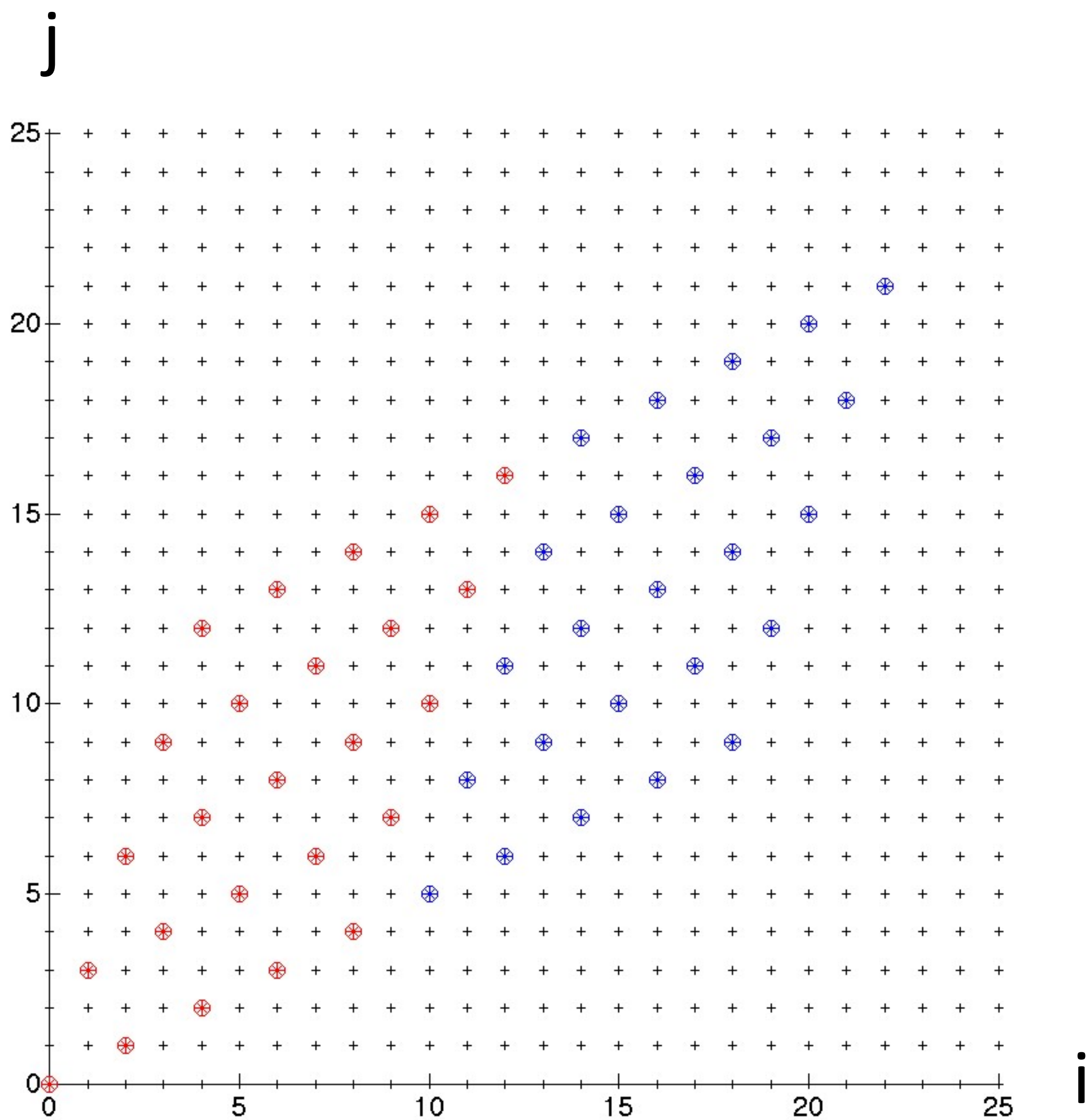
for i = 0:n
  for j = 0:n
    access A(3*i-j),
        B(i-2*j)

Tiling (for cache size 10)
  Read 5 entries of A:
    A([0,5,10,15,20])
  Read 5 entries of B:
    B([0,-5,-10,-15,-20])
  Perform 5^2 = 25
    loop iterations



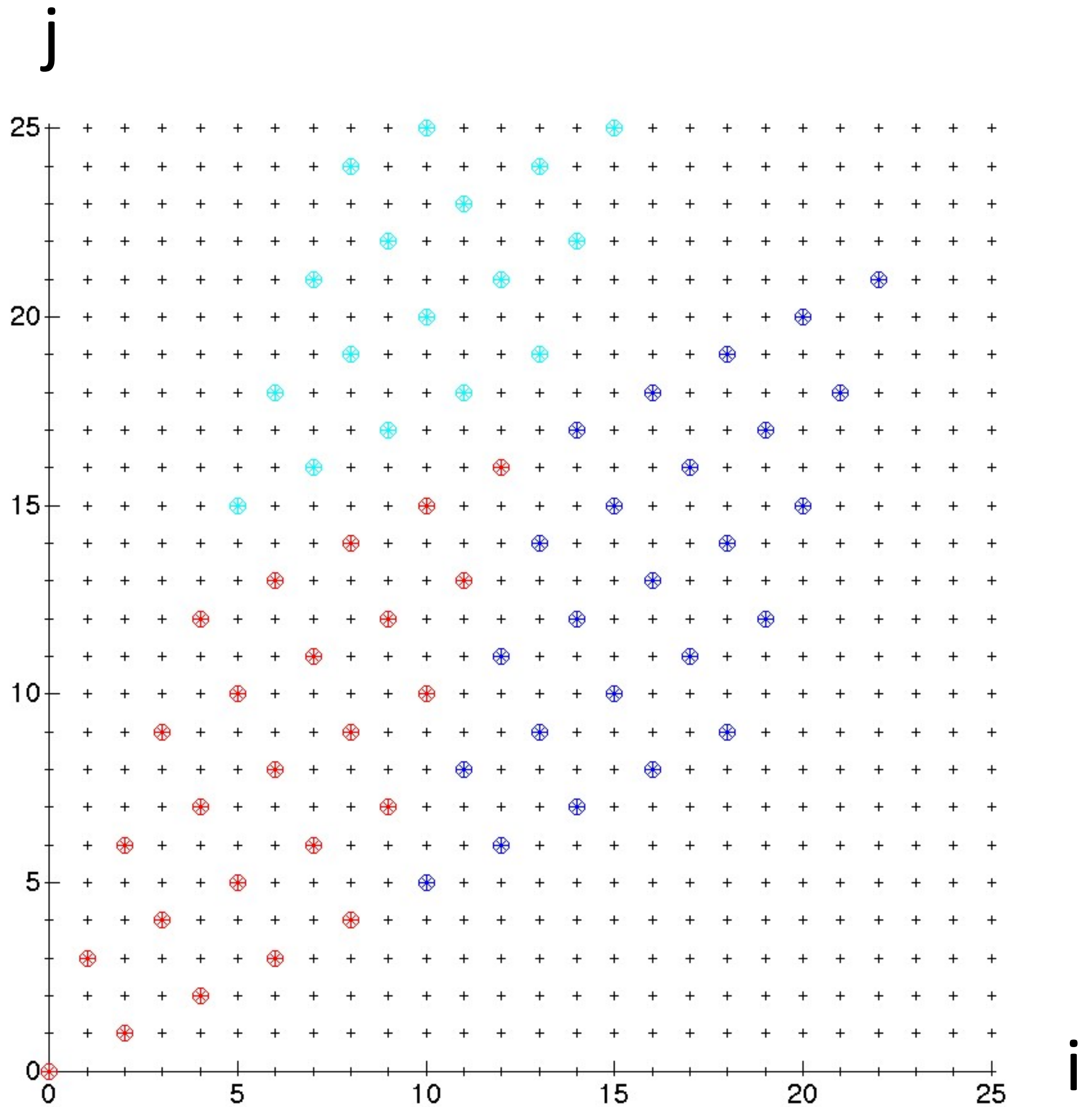
Optimal tiling for "twisted" n-body

for i = 0:n
  for j = 0:n
    access A(3*i-j),
      B(i-2*j)



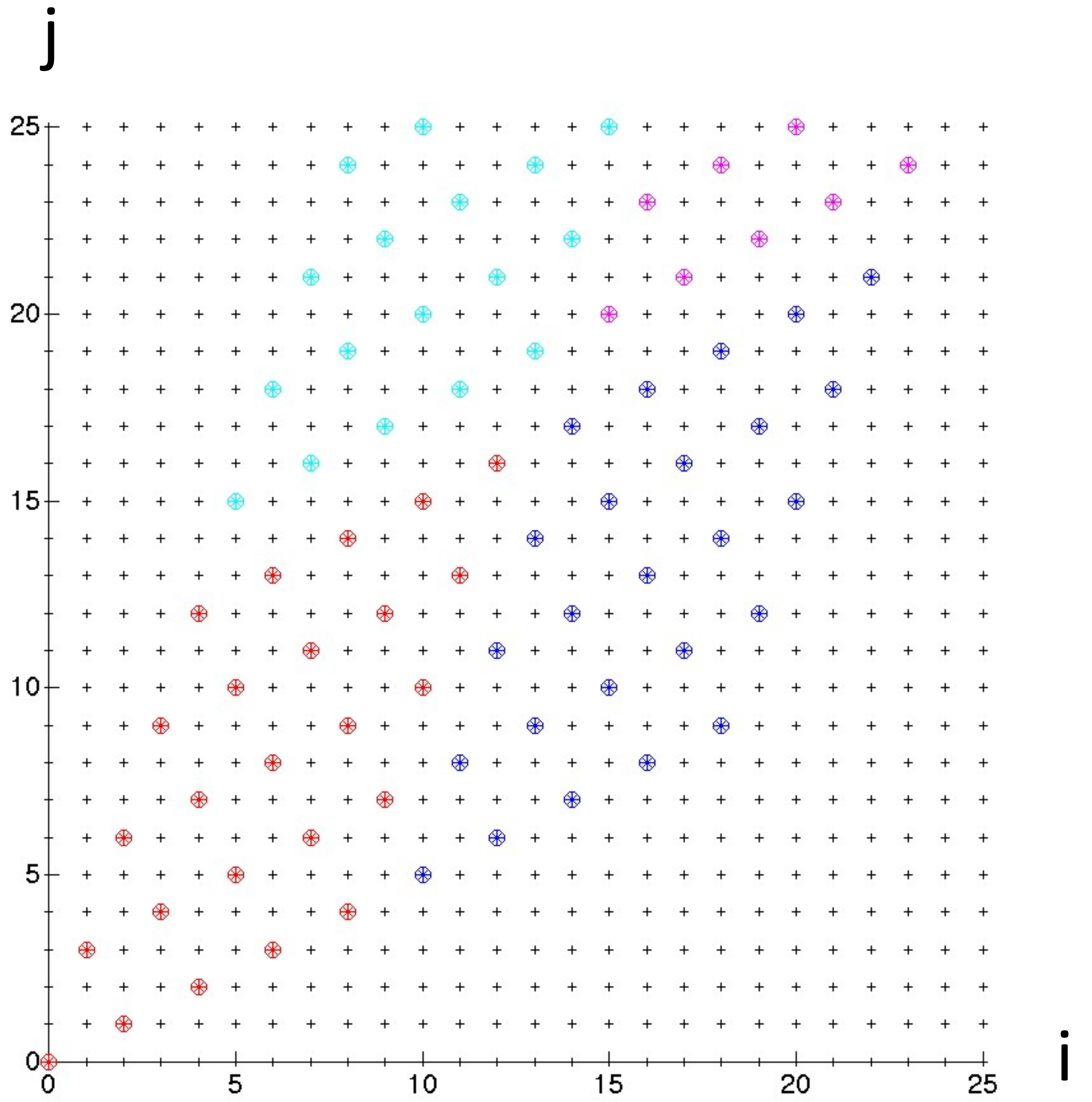
Optimal tiling for "twisted" n-body

```
for i = 0:n
  for j = 0:n
    access A(3*i-j),
           B(i-2*j)
```



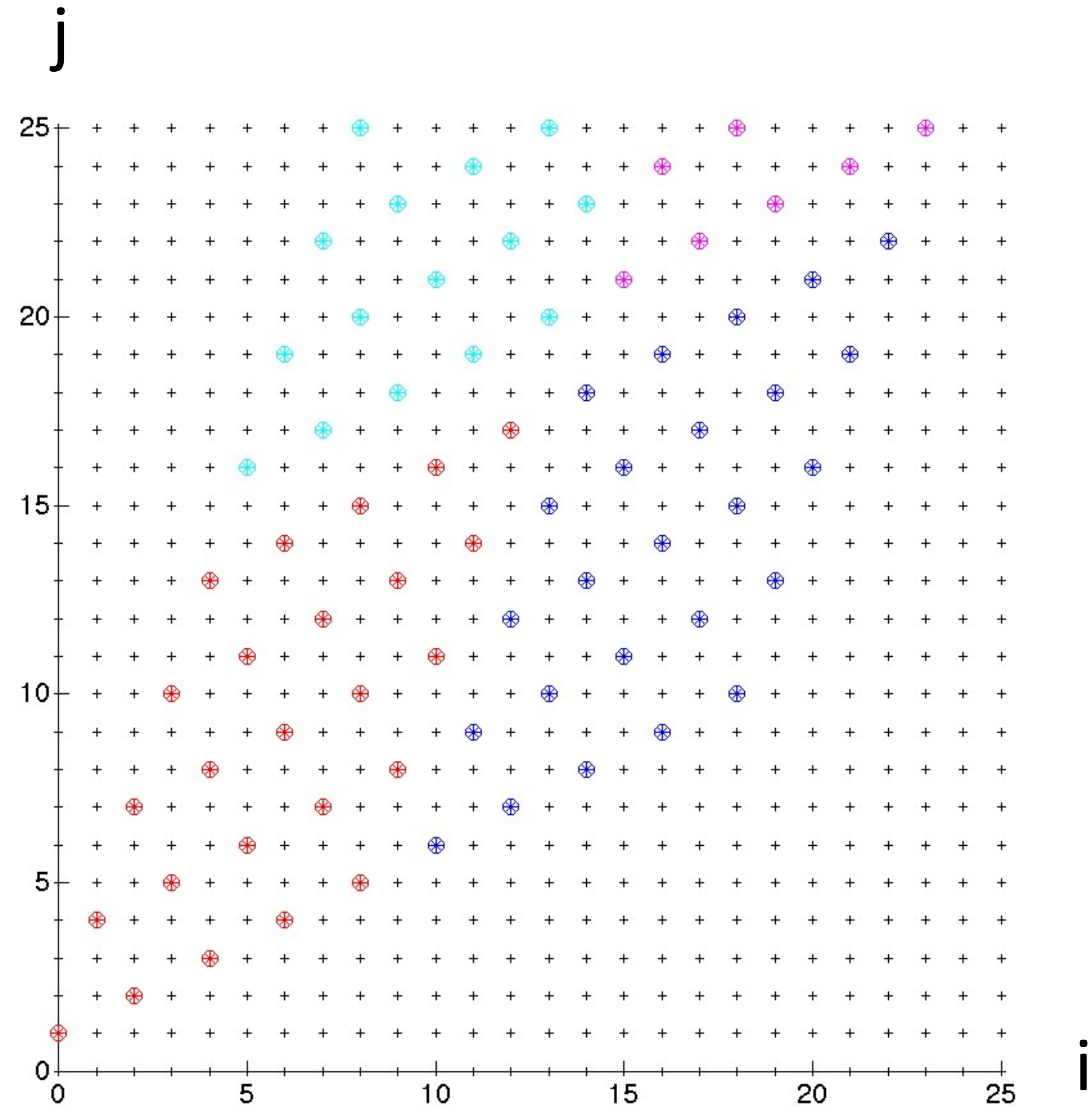
Optimal tiling for "twisted" n-body

```
for i = 0:n
  for j = 0:n
    access A(3*i-j),
           B(i-2*j)
```



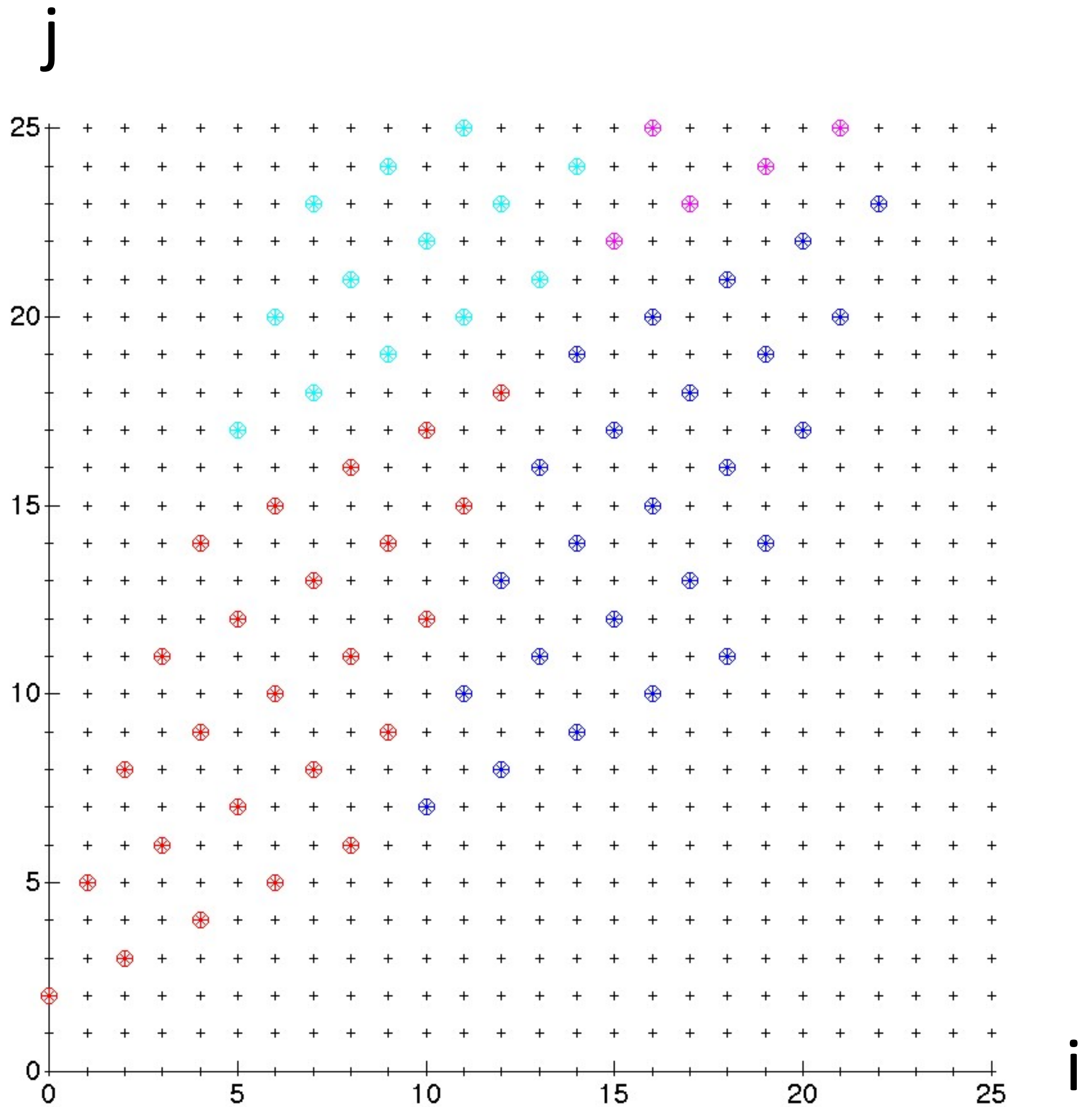
Optimal tiling for "twisted" n-body

```
for i = 0:n
  for j = 0:n
    access A(3*i-j),
           B(i-2*j)
```



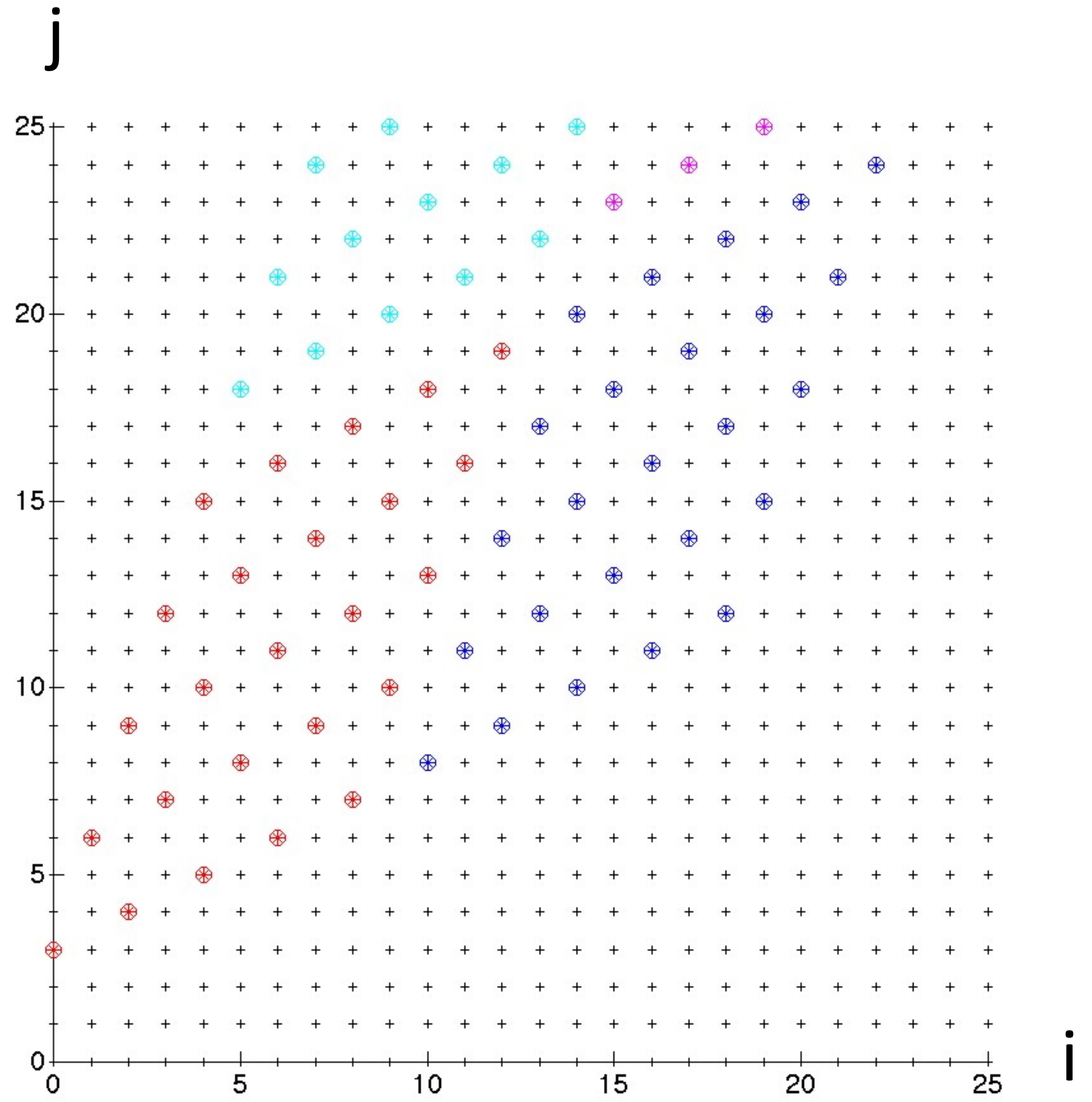
Optimal tiling for "twisted" n-body

```
for i = 0:n
  for j = 0:n
    access A(3*i-j),
           B(i-2*j)
```



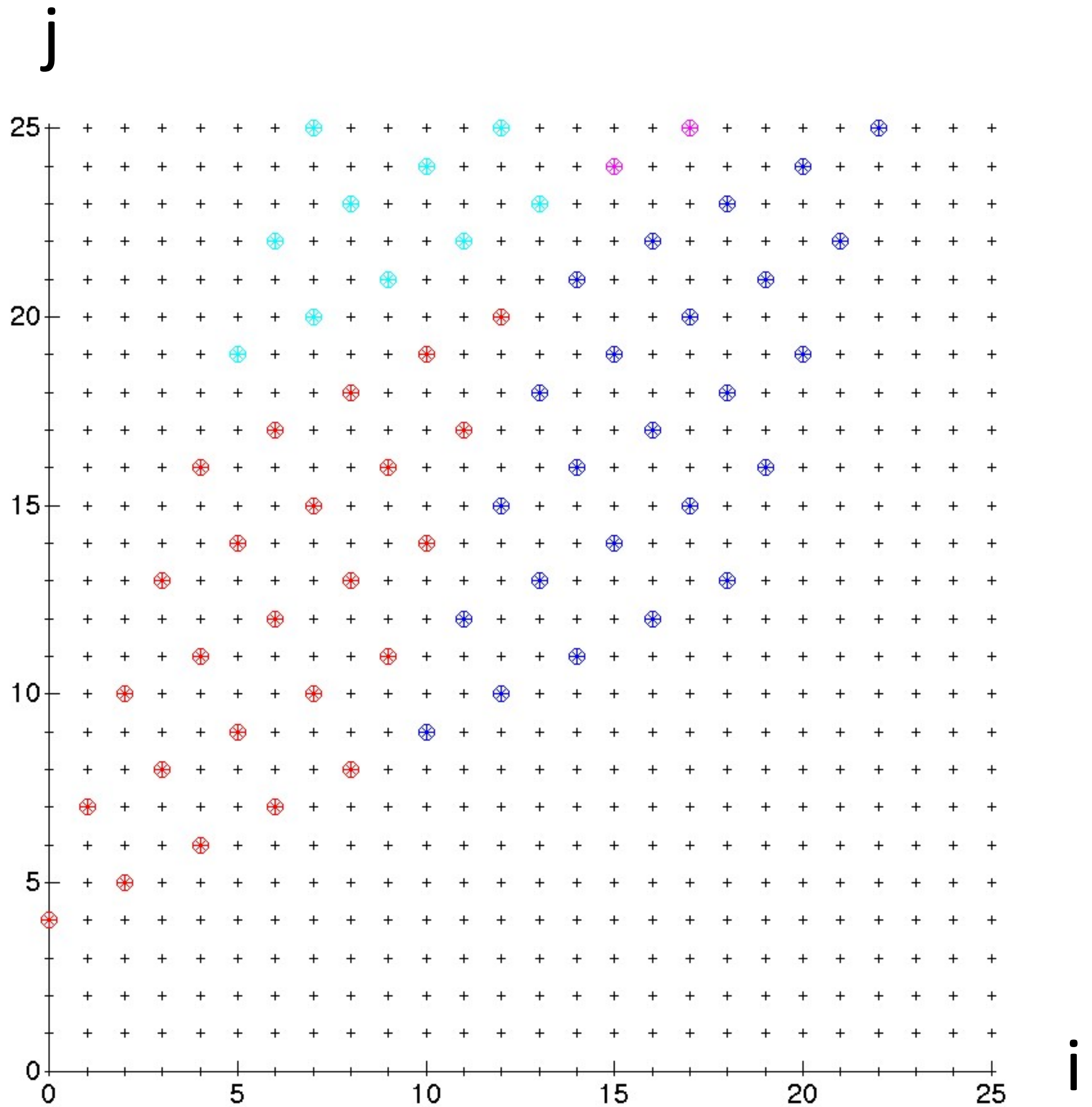
Optimal tiling for "twisted" n-body

```
for i = 0:n
  for j = 0:n
    access A(3*i-j),
           B(i-2*j)
```



Optimal tiling for “twisted” n-body

```
for i = 0:n
  for j = 0:n
    access A(3*i-j),
           B(i-2*j)
```



Optimal tiling for “twisted” n-body

```
for i = 0:n
  for j = 0:n
    access A(3*i-j),
            B(i-2*j)
```