Architectural Implications of Communication-Avoiding Algorithms

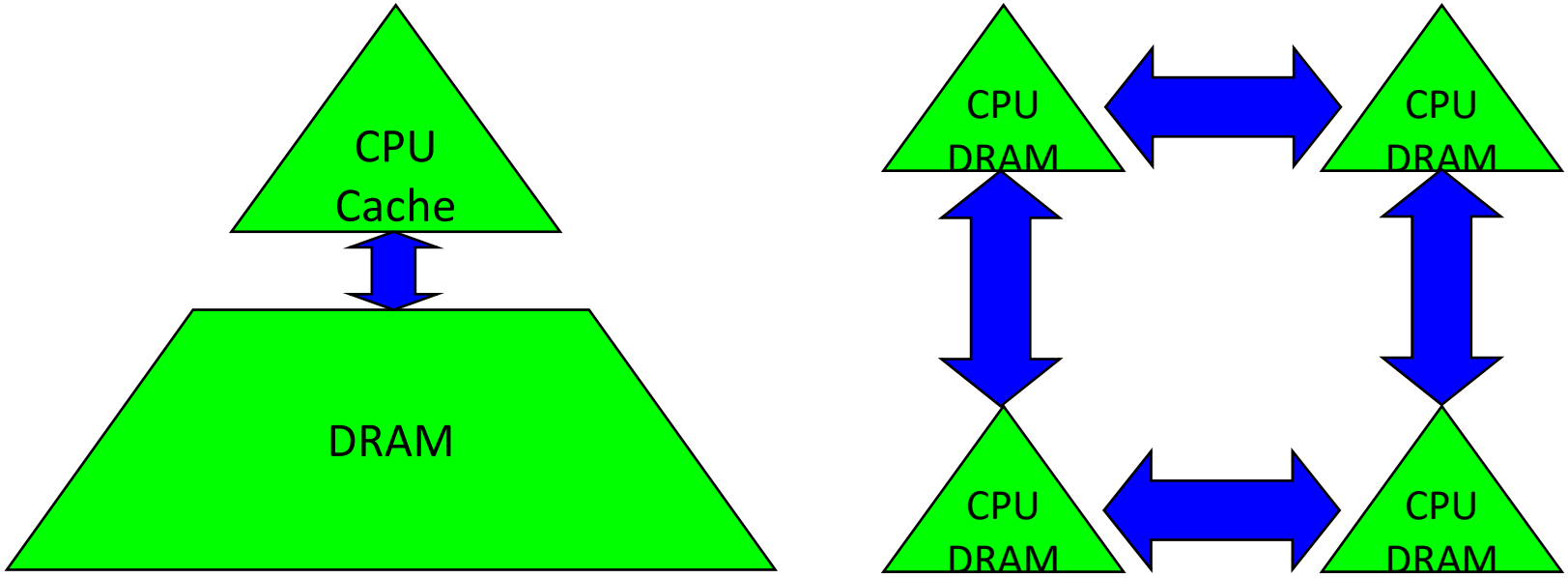
Jim Demmel, EECS & Math Depts., UC Berkeley

And many, many others …

Why avoid communication? (1/3)

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? (2/3)

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

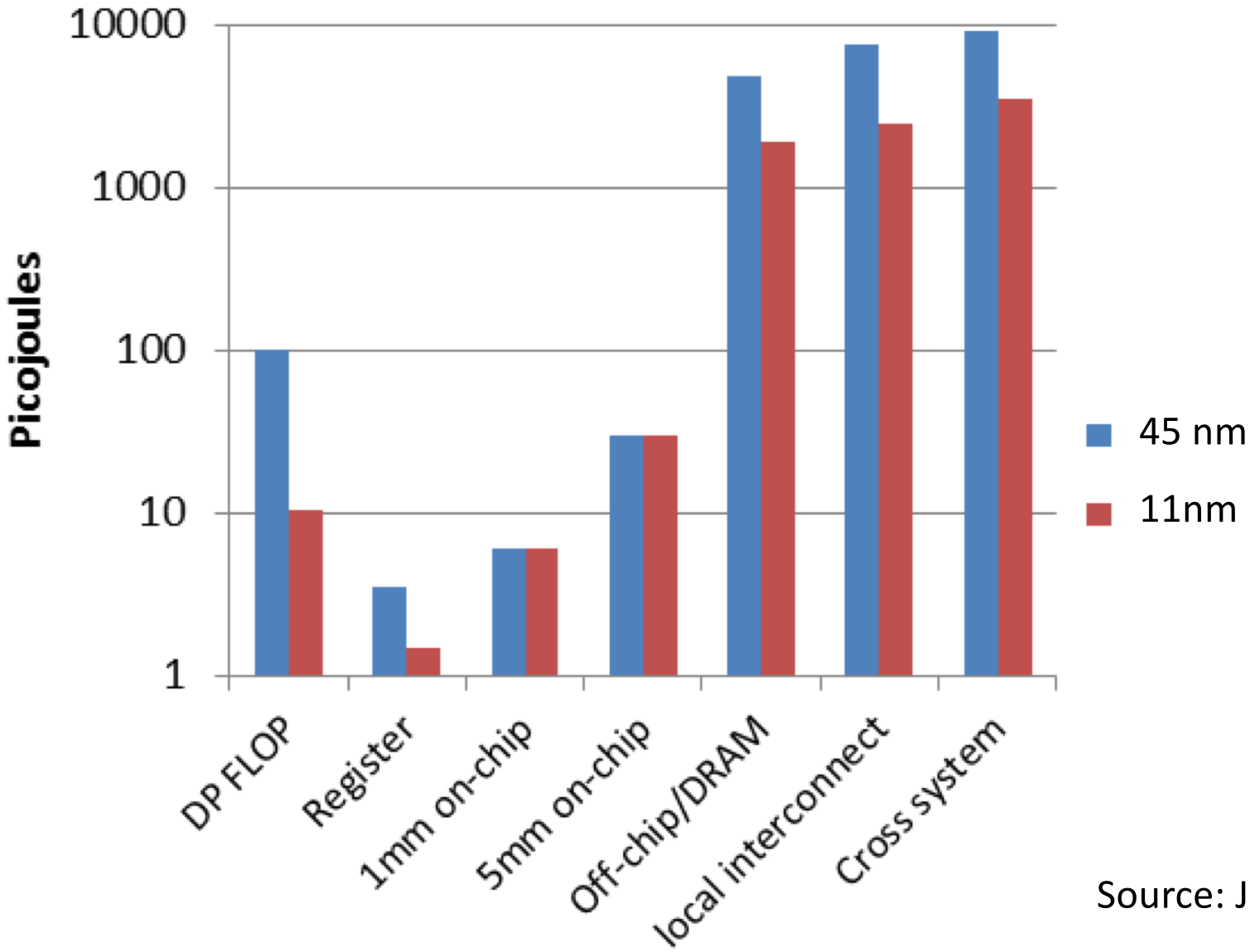
  } communication

- 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

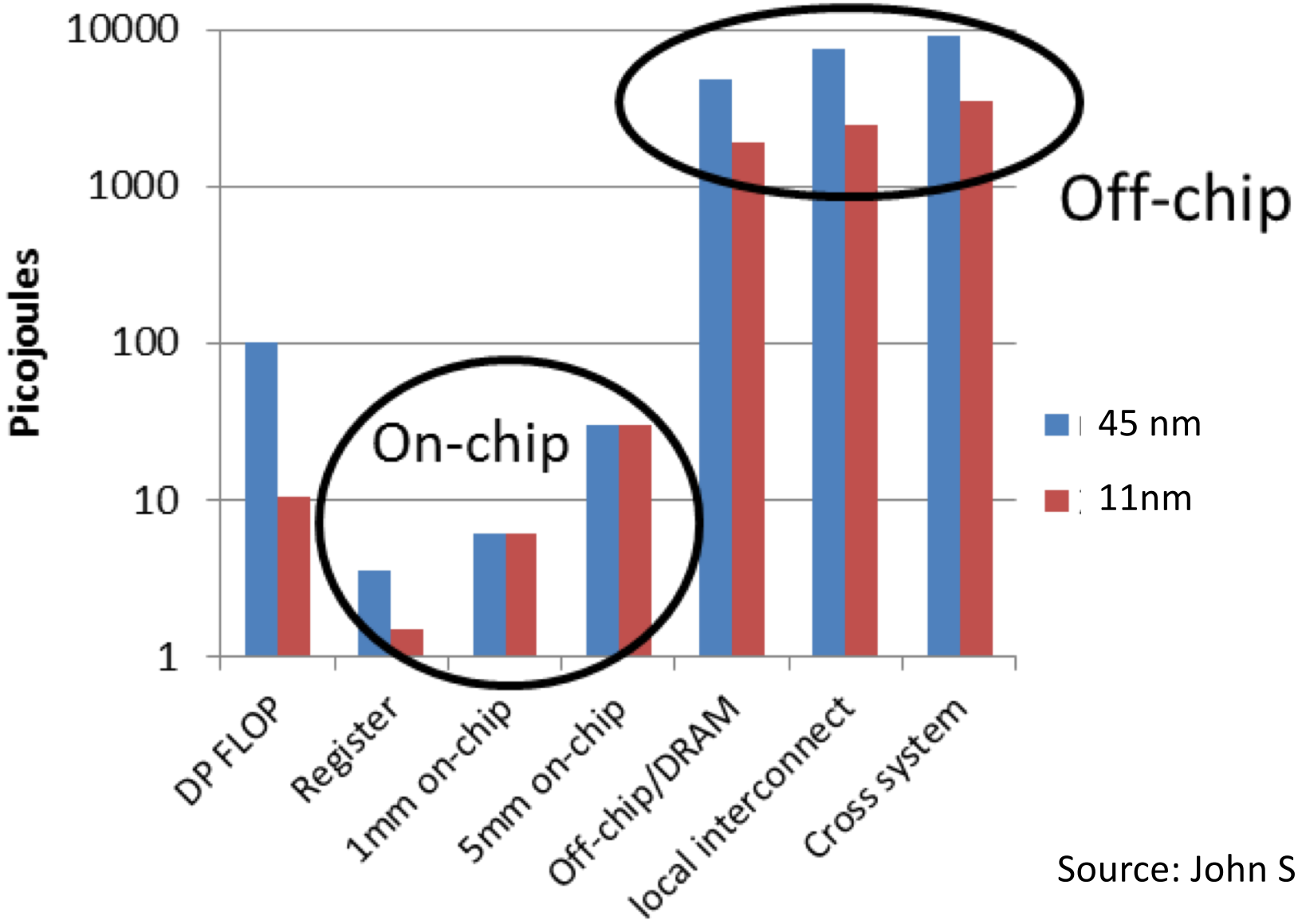
Why Minimize Communication? (3/3)



Source: John Shalf, LBL

Why Minimize Communication? (3/3)

Minimize communication to save energy



Source: John Shalf, LBL

Goals

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

Sample Speedups

- Doing same operations, just in a different order
  - Up to **12x** faster for 2.5D dense matmul on 64K core IBM BG/P
  - Up to **100x** faster for 1.5D sparse-dense matmul on 1536 core Cray XC30
  - Up to **6.2x** faster for 2.5D All-Pairs-Shortest-Path on 24K core Cray XE6
  - Up to **11.8x** faster for direct N-body on 32K core IBM BG/P
- Mathematically identical answer, but different algorithm
  - 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 **4.2x** faster for BiCGStab (MiniGMG bottom solver) on 24K core Cray XE6
  - Up to **5.1x** faster for coordinate descent LASSO on 3K core Cray XC30
- Different algorithm, different approximate answer
  - Up to **16x** faster for SVM on a 1536 core Cray XC30
  - Up to **135x** faster for ImageNet training on 2K Intel KNL nodes

Sample Speedups

- Doing same operations, just in a different order

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

  - Up to **6.2x** faster for 2.5D All-Pairs-Shortest-Path on 24K core Cray XE6
  - Up to **11.8x** faster for direct N-body on 32K core IBM BG/P

- Mathematically identical answer, but different algorithm

SIAG on Supercomputing Best Paper Prize, 2016

(D., Grigori, Hoemmen, Langou)

Released in LAPACK 3.7, Dec 2016

  - Up to **5.1x** faster for coordinate descent LASSO on 3K core Cray XC30

- Different algorithm, different approximate answer

IPDPS 2015 Best Paper Prize (You, D. Czechowski, Song, Vuduc)

ICPP 2018 Best Paper Prize (You, Zhang, Hsieh, D., Keutzer)

Outline

- Recall lower bounds for Linear Algebra
- Perfect Strong Scaling, in time and energy
- Write-Avoiding Algorithms
- Accommodating Reproducibility
- Extensions to Nested Loops Accessing Arrays

Outline

- Recall lower bounds for Linear Algebra
- Perfect Strong Scaling, in time and energy
- Write-Avoiding Algorithms
- Accommodating Reproducibility
- Extensions to Nested Loops Accessing Arrays

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
    - LAPACK, ScaLAPACK, …
  - New algorithms needed to attain these lower bounds
    - New numerical properties, ways to encode answers, data structures, not just loop transformations
  - Autotuning to find optimal implementation
  - Sparse matrices: depends on sparsity structure
- 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 ≥ #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)

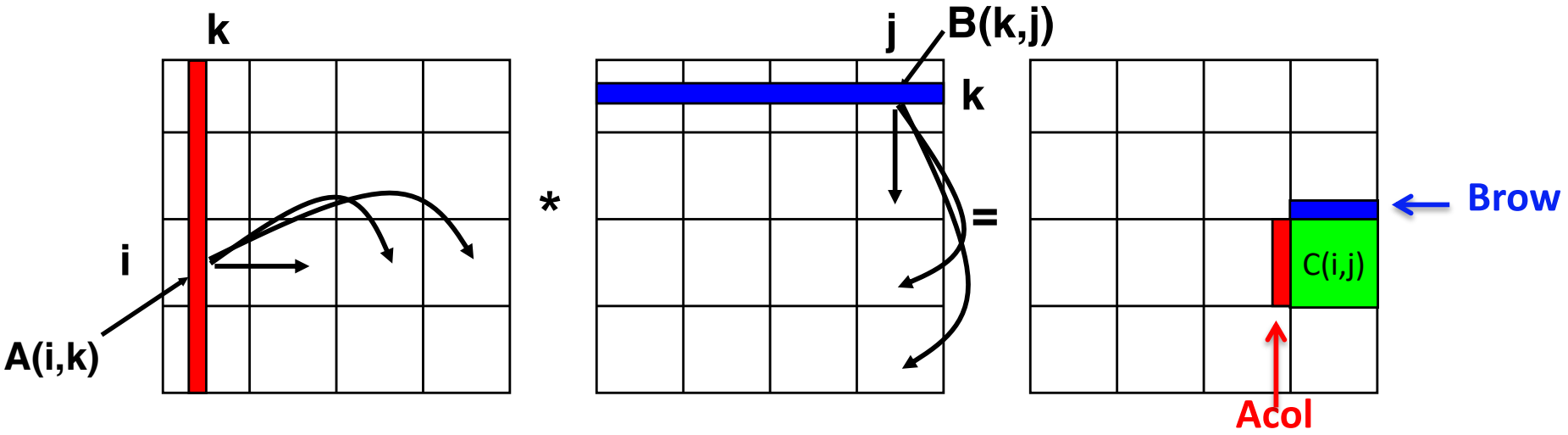
Later: Extend to nested loops accessing arrays

Outline

- Recall lower bounds for Linear Algebra
- Perfect Strong Scaling, in time and energy
- Write-Avoiding Algorithms
- Accommodating Reproducibility
- Extensions to Nested Loops Accessing Arrays

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 = $\Omega((n^3/ P) / M^{3/2}) = \Omega(P^{1/2})$
- SUMMA attains this lower bound
- 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

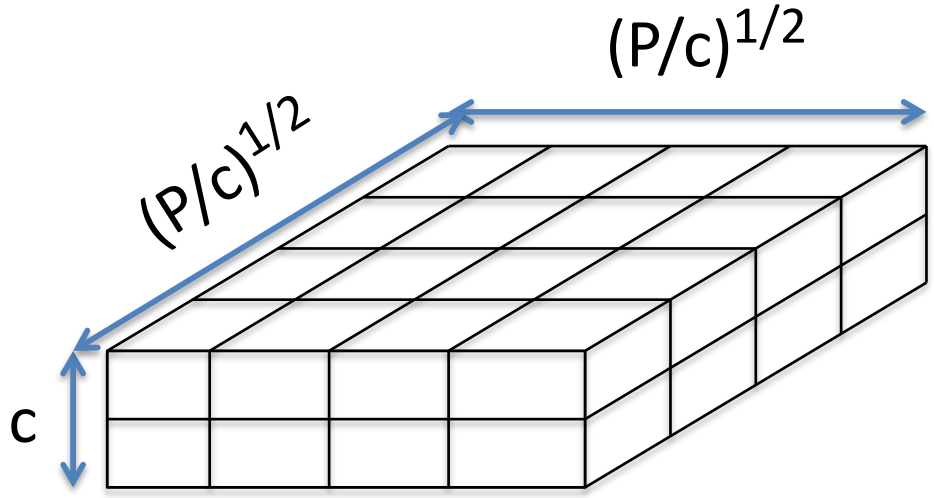
Can we do Better?

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…

2.5D Matrix Multiplication

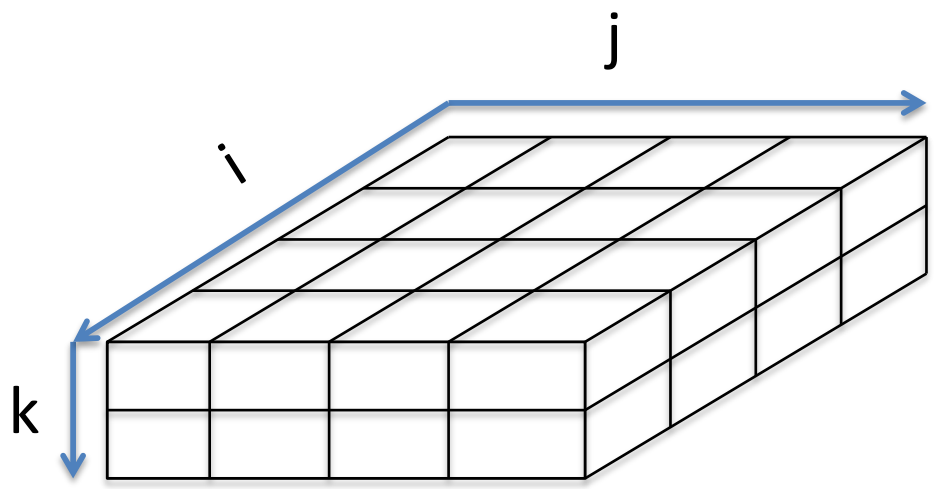
- Assume can fit cn^2/P data per processor, $c > 1$
- Processors form $(P/c)^{1/2}$ x $(P/c)^{1/2}$ x c grid



Example: P = 32, c = 2

2.5D Matrix Multiplication

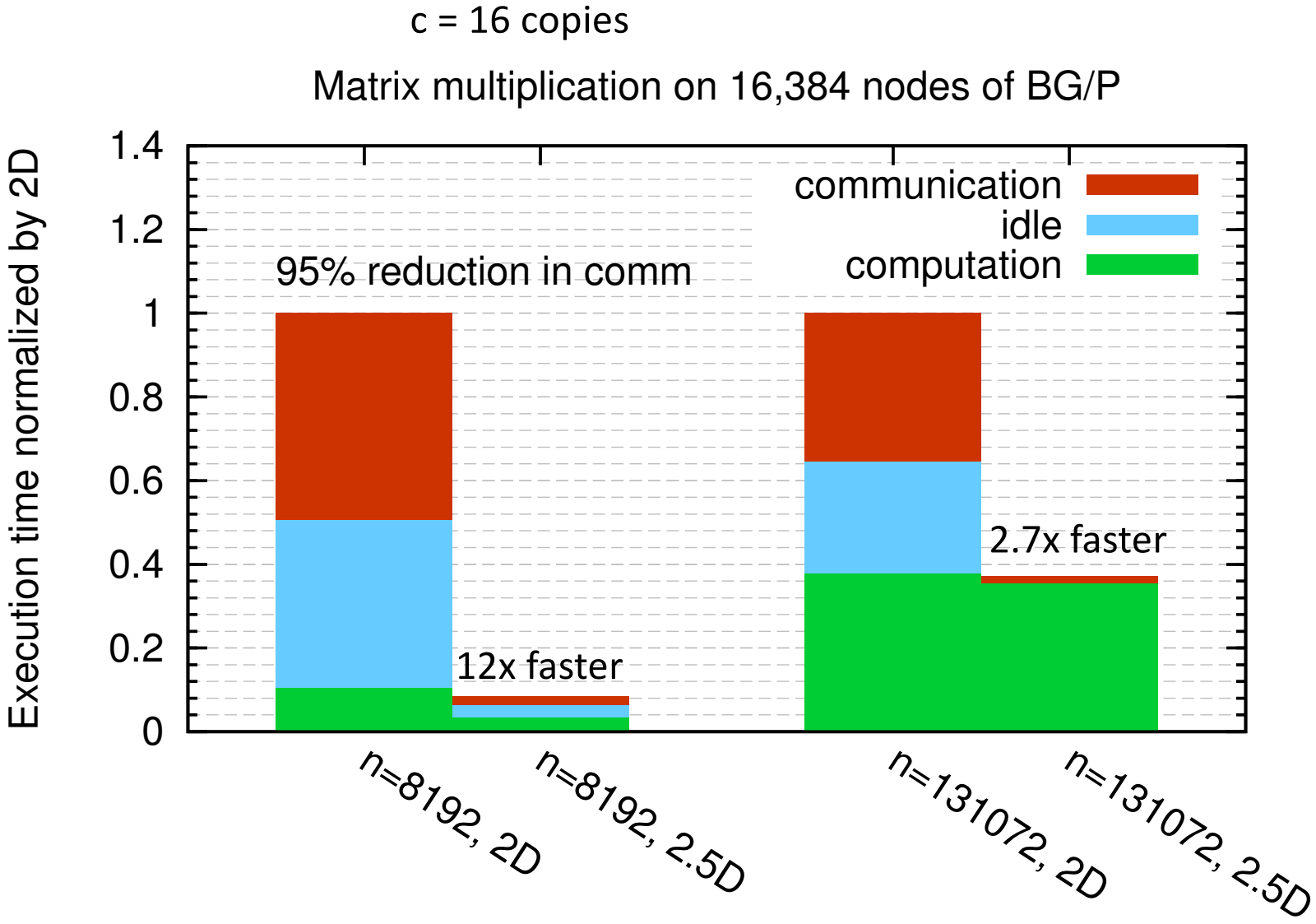
- Assume can fit cn^2/P data per processor, $c > 1$
- Processors form $(P/c)^{1/2}$ x $(P/c)^{1/2}$ x c grid



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



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

Perfect Strong Scaling – in Time and Energy (1/2)

- 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.
- $E(cP) = cP \{ n^3/(cP) [\gamma_E + \beta_E/M^{1/2} + \alpha_E/(mM^{1/2})] + \delta_E MT(cP) + \varepsilon_E T(cP) \}$
  $= E(P)$
- Limit: $c \leq P^{1/3}$ (3D algorithm), if starting with 1 copy of inputs

Perfect Strong Scaling – in Time and Energy (2/2)

- Perfect scaling extends to N-body, Strassen, …
- Thm: Perfect Strong Scaling Impossible for LU, QR
- We can use these models to answer many questions, including:
- What is the minimum energy required for a computation?
- Given a maximum allowed runtime **T** , what is the minimum energy **E** needed to achieve it?
- Given a maximum energy budget **E** , what is the minimum runtime **T** that we can attain?
- The ratio **P** = **E/T** gives us the average power required to run the algorithm. Can we minimize the average power consumed?
- See Andrew Gearhart's PhD thesis

How the Network Can Limit Scaling

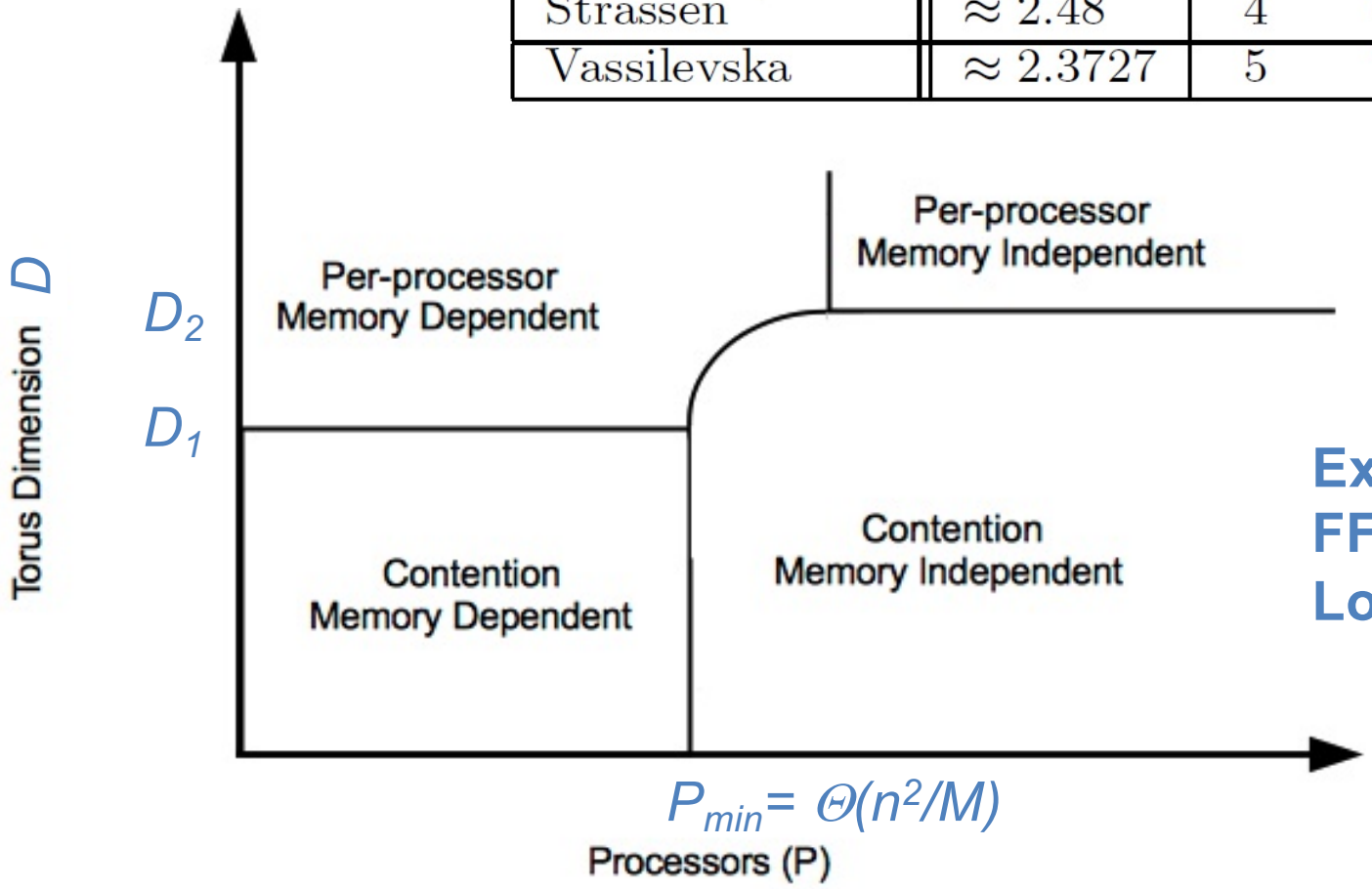
- To attain lower bound, all processors need to communicate simultaneously
  - Depends on network topology
  - Ex: If can only communicate in 1D Torus, may not be able to do 2D or 2.5D Matmul
- Approach:
  - Apply same lower bound to arbitrary subset of processors, compare to “small set expansion” of network graph

Network scaling limits for Matmul On D-dimensional Tori

$D \leq D_1$: network contention dominates

$D \geq D_2$: per-processor dominates

Algorithm	ω_0	D_1	D_2
Classical	3	2	3
Strassen	≈ 2.81	2	4
Schönhage	≈ 2.55	3	5
Strassen	≈ 2.48	4	6
Vassilevska	≈ 2.3727	5	7



Extends to FFT, Sorting, Loops accessing arrays

Outline

- Recall lower bounds for Linear Algebra
- Perfect Strong Scaling, in time and energy
- Write-Avoiding Algorithms
- Accommodating Reproducibility
- Extensions to Nested Loops Accessing Arrays

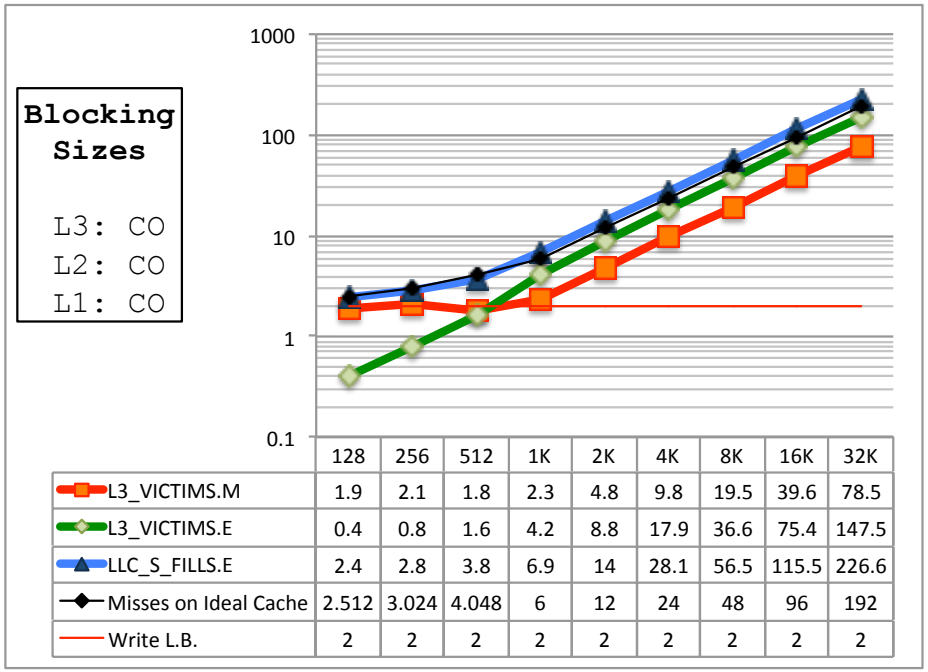
Write-Avoiding Algorithms

- What if writes are more expensive than reads?
  - Nonvolatile Memory (Flash, PCM, …)
  - Saving intermediates to disk in cloud (eg Spark)
  - Extra coherency traffic in shared memory
- Can we design “write-avoiding (WA)” algorithms?
  - Goal: find and attain better lower bound for writes
  - Thm: For classical matmul, possible to do asymptotically fewer writes than reads to given layer of memory hierarchy
  - Thm: Cache-oblivious algorithms cannot be write-avoiding
  - Thm: Strassen and FFT cannot be write-avoiding
  - Distributed Memory: More complicated

Measured L3-DRAM traffic on Intel Nehalem Xeon-7560

Optimal #DRAM reads = $O(n^3/M^{1/2})$

Optimal #DRAM writes = n^2

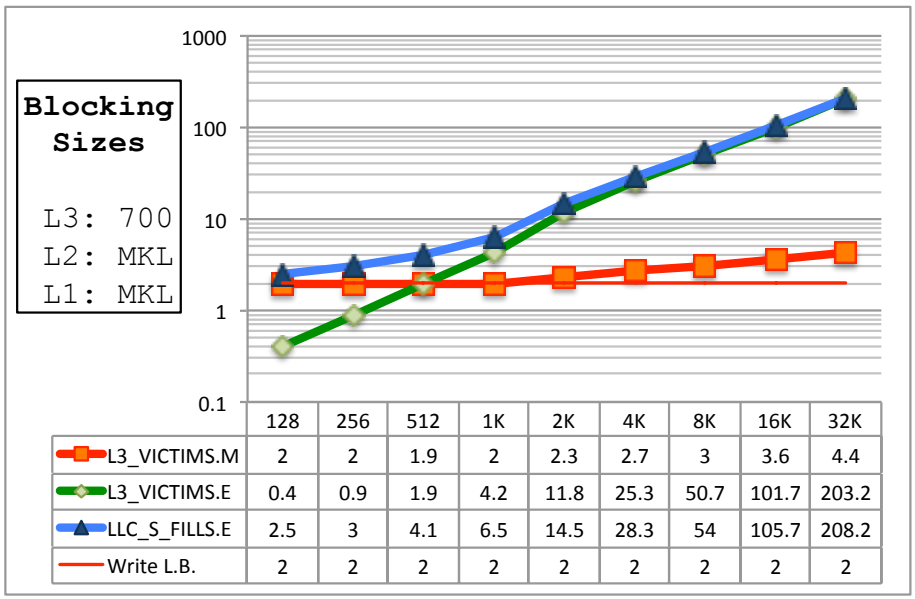


	128	256	512	1K	2K	4K	8K	16K	32K
L3_VICTIMS.M	1.9	2.1	1.8	2.3	4.8	9.8	19.5	39.6	78.5
L3_VICTIMS.E	0.4	0.8	1.6	4.2	8.8	17.9	36.6	75.4	147.5
LLC_S_FILLS.E	2.4	2.8	3.8	6.9	14	28.1	56.5	115.5	226.6
Misses on Ideal Cache	2.512	3.024	4.048	6	12	24	48	96	192
Write L.B.	2	2	2	2	2	2	2	2	2

Cache-Oblivious Matmul

#DRAM reads close to optimal

#DRAM writes much larger



	128	256	512	1K	2K	4K	8K	16K	32K
L3_VICTIMS.M	2	2	1.9	2	2.3	2.7	3	3.6	4.4
L3_VICTIMS.E	0.4	0.9	1.9	4.2	11.8	25.3	50.7	101.7	203.2
LLC_S_FILLS.E	2.5	3	4.1	6.5	14.5	28.3	54	105.7	208.2
Write L.B.	2	2	2	2	2	2	2	2	2

Write-Avoiding Matmul

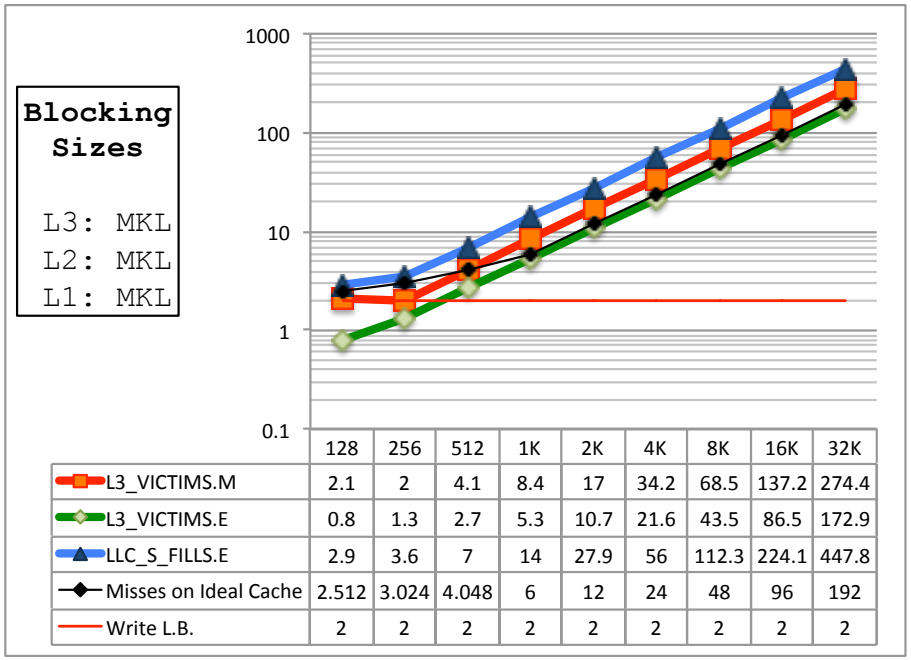
#L3 misses close to optimal

#DRAM writes close to optimal

Measured L3-DRAM traffic on Intel Nehalem Xeon-7560

Optimal #DRAM reads = $O(n^3/M^{1/2})$

Optimal #DRAM writes = n^2

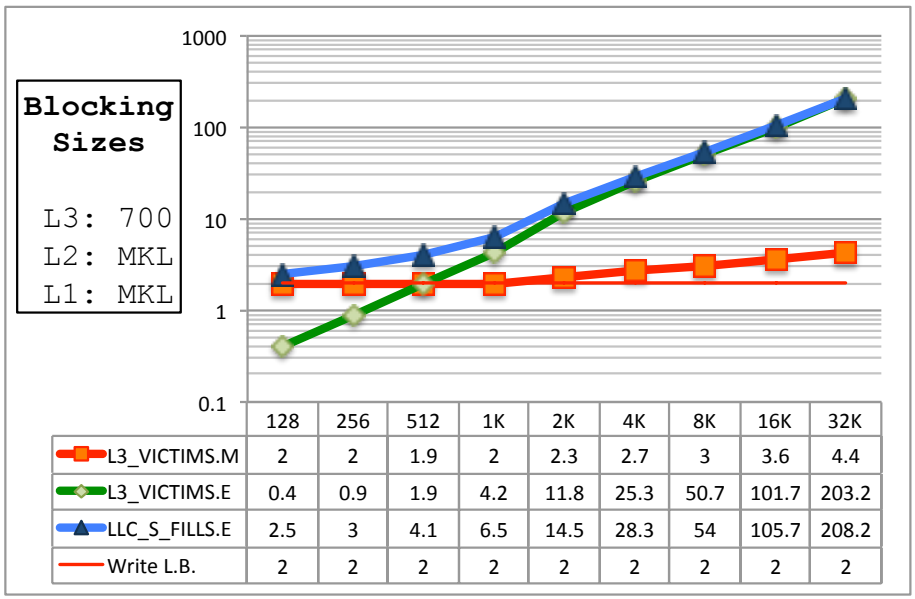


	128	256	512	1K	2K	4K	8K	16K	32K
L3_VICTIMS.M	2.1	2	4.1	8.4	17	34.2	68.5	137.2	274.4
L3_VICTIMS.E	0.8	1.3	2.7	5.3	10.7	21.6	43.5	86.5	172.9
LLC_S_FILLS.E	2.9	3.6	7	14	27.9	56	112.3	224.1	447.8
Misses on Ideal Cache	2.512	3.024	4.048	6	12	24	48	96	192
Write L.B.	2	2	2	2	2	2	2	2	2

Intel MKL Matmul

#DRAM reads >2x optimal

#DRAM writes much larger



	128	256	512	1K	2K	4K	8K	16K	32K
L3_VICTIMS.M	2	2	1.9	2	2.3	2.7	3	3.6	4.4
L3_VICTIMS.E	0.4	0.9	1.9	4.2	11.8	25.3	50.7	101.7	203.2
LLC_S_FILLS.E	2.5	3	4.1	6.5	14.5	28.3	54	105.7	208.2
Write L.B.	2	2	2	2	2	2	2	2	2

Write-Avoiding Matmul

#L3 misses close to optimal

#DRAM writes close to optimal

Outline

- Recall lower bounds for Linear Algebra
- Perfect Strong Scaling, in time and energy
- Write-Avoiding Algorithms
- Accommodating Reproducibility
- Extensions to Nested Loops Accessing Arrays

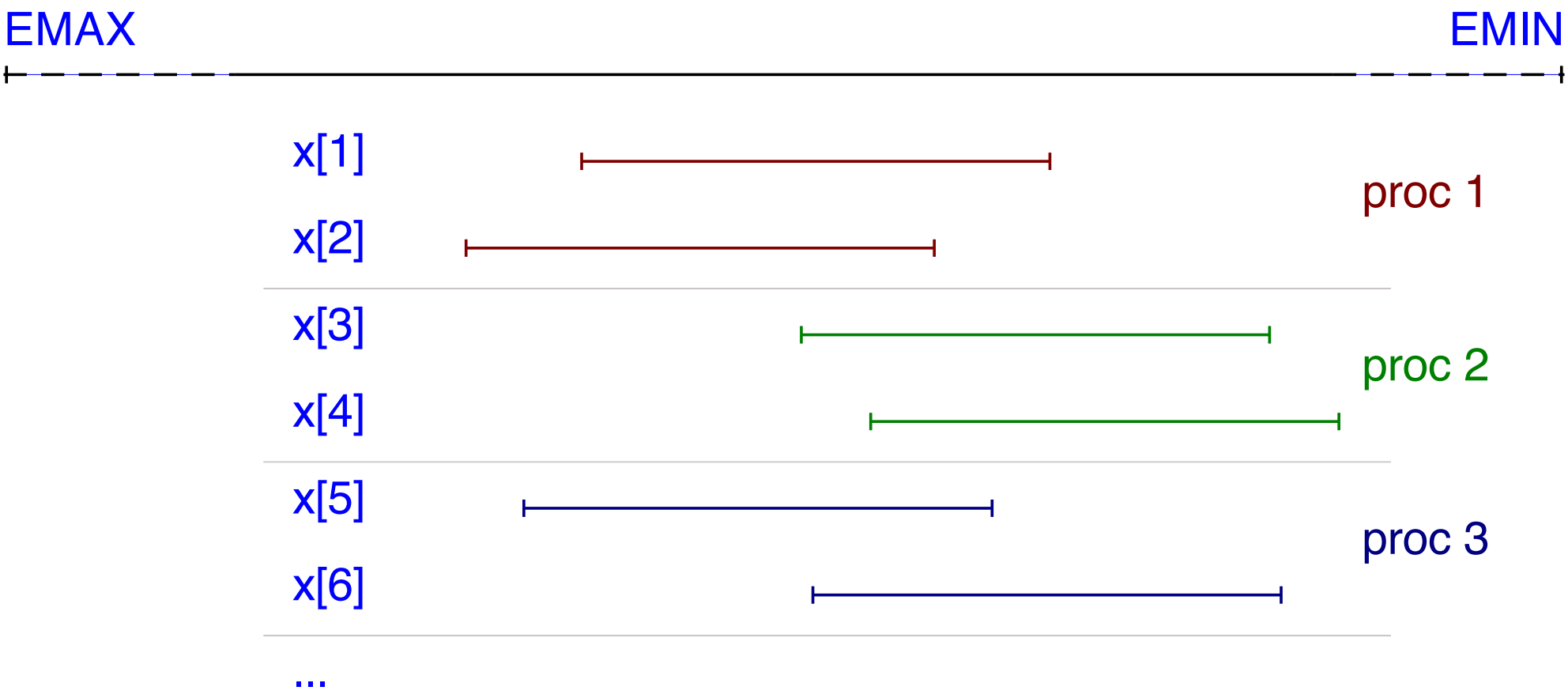
Reproducibility

- Motivation for reproducibility
  - Debugging, correctness, contractual requirements …
  - https://gcl.cis.udel.edu/sc15bof.php
- Reproducible floating point summation
  - Challenge: addition not associative, summation order can vary because of parallelism, changing hardware resources
  - Goal: bitwise reproducible summation
- BLAS Standard Committee
  - Working on adding reproducible BLAS to next standard
- IEEE 754 Floating Point Standard Committee
  - Voted to add new instructions to support both higher precision arithmetic and reproducible summation

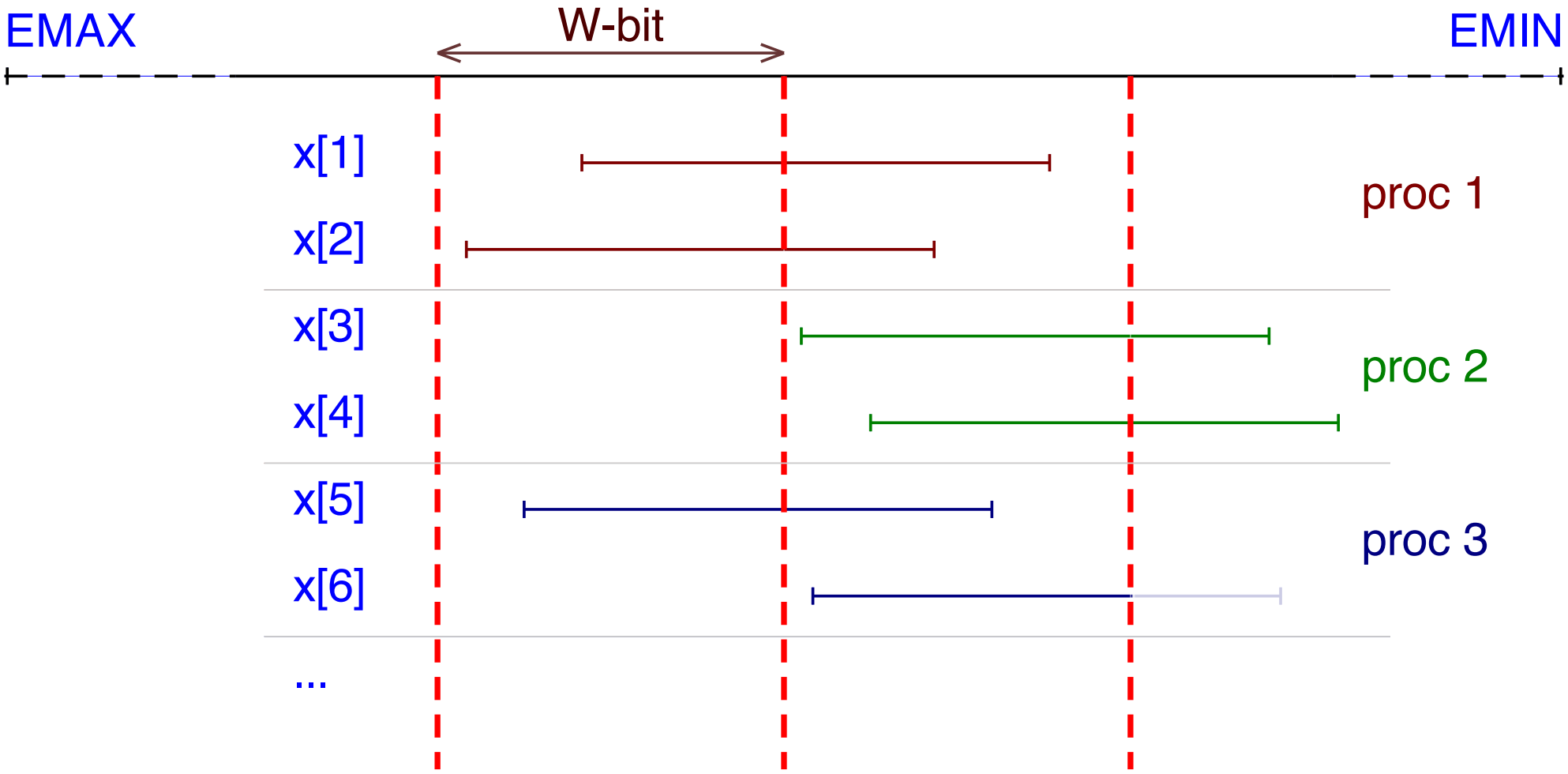
Design Goals for Reproducible Sum (high level)

1. Reproducible sum, independent of order, assuming a subset of IEEE 754
   - Limits: #summands at most 2^{64} in double, 2^{33} in single
2. Accuracy at least as good as conventional, and tunable
   - Default: $\geq$ 80 bit accuracy
3. Handle exceptions reproducibly
4. One read-only pass over summands
5. One reduction
6. Use as little memory as possible, to enable tiling BLAS
   - Default: one "reproducible accumulator" is 6 floats
7. Modular design, for various use cases

Binned Summation



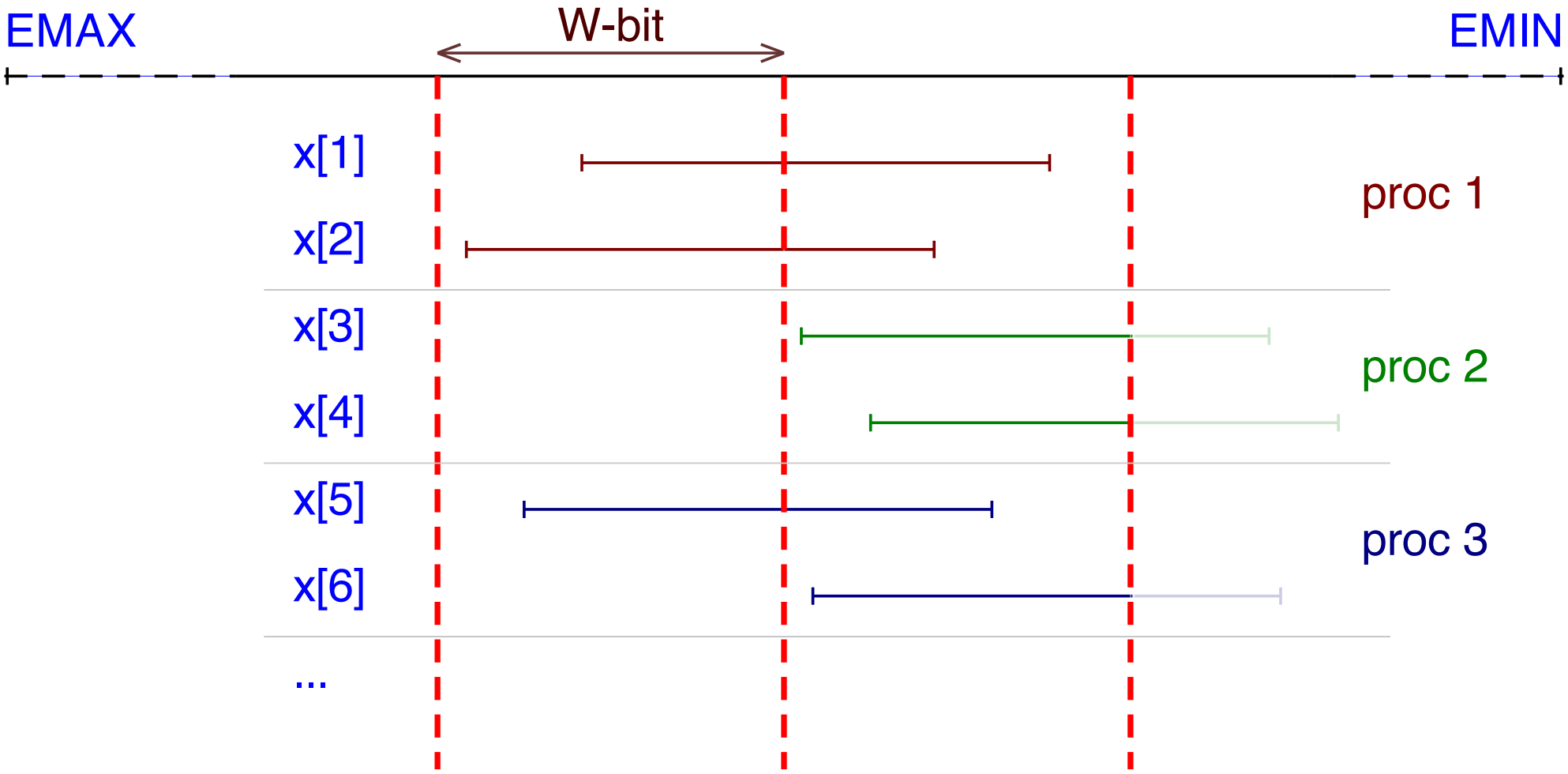
Binned Summation



- Boundaries predetermined

K = 2 bins

Binned Summation



- Only keep top K bins, don't compute, or discard, the rest

Cost to sum n numbers reproducibly (1/2)

- Depends on K = # bins
  - Use K = 3, min value at least as accurate as usual sum
- Arithmetic Costs
  - Using existing 754 standard: (3K-2)n = 7n ops
  - Using new operation in next 754: Kn = 3n ops
  - Additional common operations: n abs, n max
  - Same cost (plus O(1)) for higher precision too
    - Needs larger reproducible accumulator
- How does communication cost depend on K?

Cost to sum n numbers reproducibly (2/2)

- How does communication cost depend on K?
  - Let R = 2K = size of "reproducible accumulator"
- Goal: Lower bound and optimal tiling that depends on R
- Thm: For sequential n x n matmul, lower bound = $\Omega(n^3\ R^{1/2}\ /M^{1/2}\ )$
- Thm: Attainable by tiling $(\frac{M}{R})^{\frac{1}{2}}$ x $(MR)^{\frac{1}{2}}$ x $(\frac{M}{R})^{\frac{1}{2}}$

Outline

- Recall lower bounds for Linear Algebra
- Perfect Strong Scaling, in time and energy
- Write-Avoiding Algorithms
- Accommodating Reproducibility
- Extensions to Nested Loops Accessing Arrays

Recall optimal sequential Matmul

- For simplicity, assume 2 level memory hierarchy, cache size = M, only consider # words moved between cache & main memory

- 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 i = 1:n/b, for j = 1:n/b, for k = 1:n/b

  C[i,j]+=A[i,k]*B[k,j] … b x b matmul

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

Goal

- Generalize matmul lower bound to any code that “resembles” nested loops accessing arrays, whose subscripts are affine functions of the loop indices
  - Some technical assumptions…
- Lower bound is always attainable
  - Some other technical assumptions...
- Open problems…

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 affine functions, 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 and 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

  $$\#words\_moved = \Omega(|S| / M^{e-1})$$

  where $|S|$ = #loop iterations, M = cache size,

  and e is the the value of a linear program:

  $$\text{minimize } e = \Sigma_j e_j \text{ subject to}$$

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

- Proof depends on recent result in pure mathematics by Christ/Tao/Carbery/Bennett
  - Generalization of Hölder-Brascamp-Lieb inequality to Abelian groups

Some Technical Assumptions

- No interleaving of inner loops
  - Loop splitting can potentially reduce lower bound
- No "unbounded reuse" possible, for example:

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

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

  looks like matmul, with lower bound $\Omega(n^3/M^{1/2})$

  but could be rewritten as

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

  t = A(i,k)*B(k,j), s = s + t, if (k=n), C(i,j)=t, endif

  with lower bound $\Omega(n^3/M)$
- More formal model in publications

Is this bound attainable (1/2)?

- But first: Can we write it down?
  - One inequality per subgroup $H < Z^d$, but still finitely many!
  - Thm (bad news): Writing down all inequalities in LP reduces to Hilbert's 10^{th} problem over Q
    - Could be undecidable: open question
  - Thm (good news): Another LP has same solution, is decidable (but could be expensive)
  - Thm: (better news) Easy to write LP down explicitly in many cases of interest:
    - When at most 3 arrays
    - When at most 5 loop indices
    - When subscripts are subsets of indices
    - "Compositions" of the above (eg CNNs)

Is this bound attainable (2/2)?

- Thm: We can always construct an optimal tiling, that attains the lower bound
- Some technical assumptions:
  - Attains lower bound Ω (#iterations/M^{e-1}) in O() sense
    - Can determine and attain hidden constant in some cases
  - Depends on loop dependencies
    - Not all tilings may compute the right answer
    - Easiest case: no dependencies, or reductions (like matmul)
  - Assumes loop bounds are large enough to fit tile
    - Ex: same lower bound for matmul applies to matrix-vector-multiply, but not attainable
- Based on dual of linear program in lower bound

Open Problems

- What happens when some loop bounds too small to fit optimal tile?
  - Ex: CNNs
- Extend to multiple levels of memory hierarchy
  - Depends on previous question
- Generalize 2.5D approach
- Deal with loop-carried dependencies

Collaborators and Supporters

- **James Demmel**, **Kathy Yelick**, Aditya Devarakonda, Grace Dinh, Michael Driscoll, Penporn Koanantakool, Alex Rusciano, Yang You
- 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, Kimon Fountoulakis, Armando Fox, David Gleich, Ming Gu, Jeff Hammond, Mike Heroux, Olga Holtz, Kurt Keutzer, Julien Langou, Xiaoye Li, Michael Mahoney, Devin Matthews, Tom Scanlon, Michelle Strout, Sam Williams, Hua Xiang, Zhao Zhang, Cho-Jui Hsieh,
- 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 linear algebra survey in Acta Numerica (2014)
- CS267 – Berkeley's Parallel Computing Course
  - Live broadcast in Spring 2018, next in 2019
    - 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

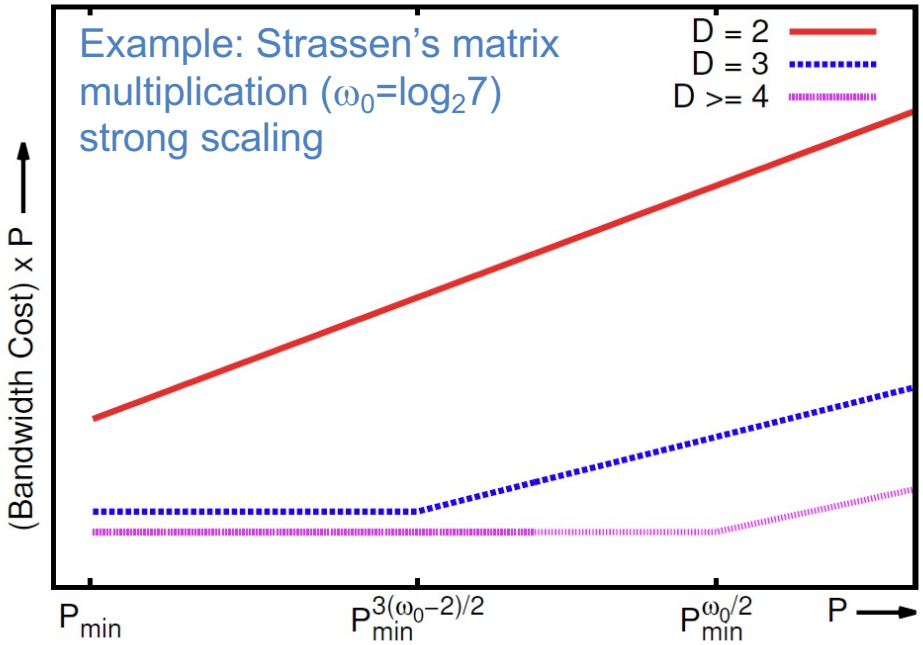
linear algebra, machine learning, n-body, …
algorithms and software (and compilers)

Don’t Communic…

49

EXTRA SLIDES

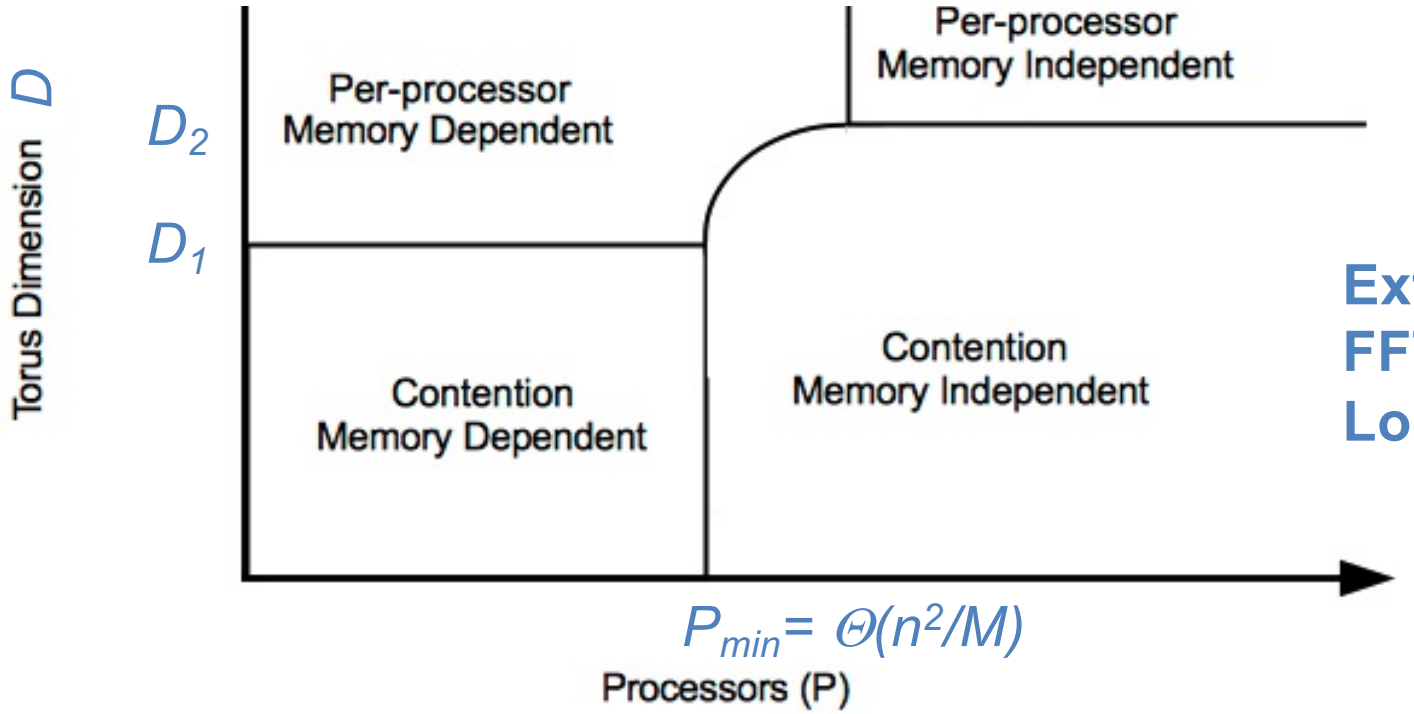
Network contention vs. per-processor bounds



$D \leq D_1$: network contention dominates

$D \geq D_2$: per-processor dominates

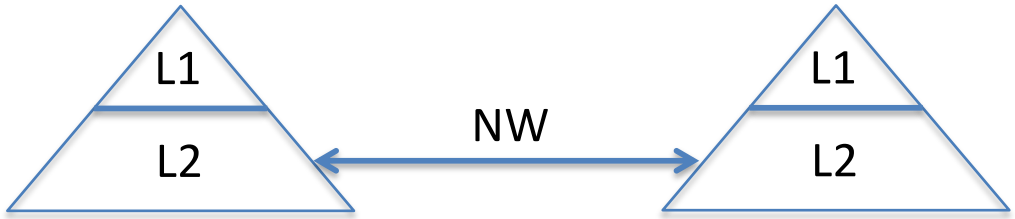
Algorithm	ω_0	D_1	D_2
Classical	3	2	3
Strassen	≈ 2.81	2	4
Schönhage	≈ 2.55	3	5
Strassen	≈ 2.48	4	6
Vassilevska	≈ 2.3727	5	7



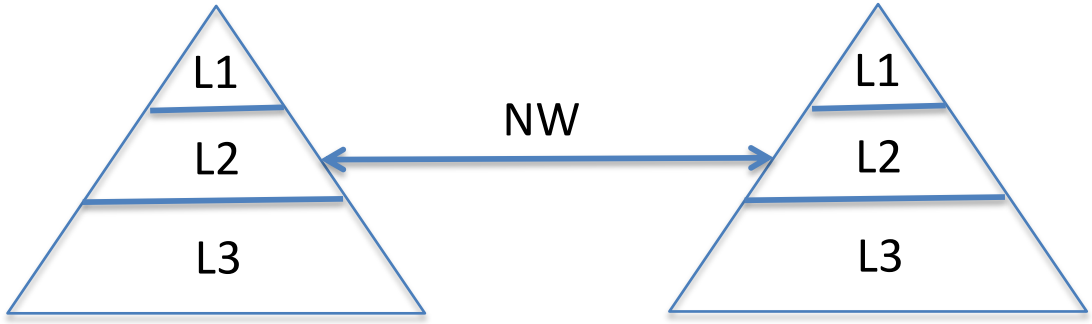
Extends to FFT, Sorting, Loops accessing arrays

Parallel WA Algorithms

- Model 1:
  - L1 = cache
  - L2 = DRAM
  - Network (NW) connects L2s
  - Can we minimize NW communication and L2 writes from L1?



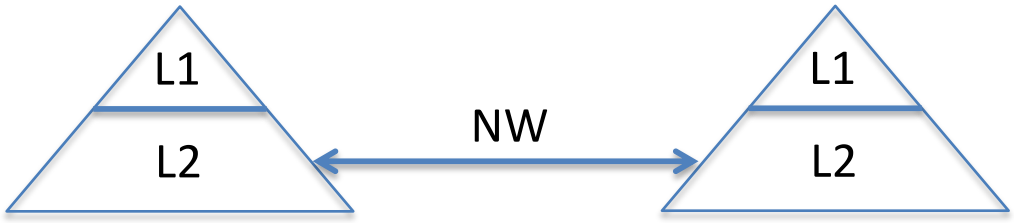
- Model 2.1
  - L3 = NVM
  - Data fits in L2
  - Is it worth using L3?



- Model 2.2
  - Same architecture as Model 2.1
  - Data fits in L3, not L2
  - Can we minimize NW communication and L3 writes from L2?

Parallel WA Algorithms – Model 1

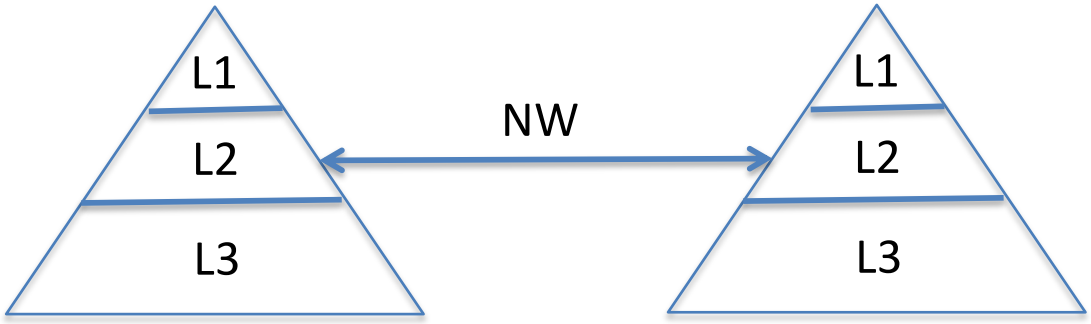
- Model 1: L1 and L2, network connects L2s



- Can we minimize NW communication and L2 writes from L1?
- Natural idea:
  - Use CA algorithm to minimize NW communication
  - Use WA algorithm locally on each processor
  - Applies to Matmul, TRSM, Cholesky, N-Body, …
- Does it work, for Matmul?
  - Goals: $n^2/P^{1/2}$ words moved on NW, n^2/P writes to L2 from L1
  - Yes for NW, but $n^2/P^{1/2}$ writes to L2 from L1
    - Probably ok, since dominated by NW costs
  - Can attain both lower bounds, but with $P^{1/2}$ times as much L2
    - Probably not worth it

Parallel WA Algorithms – Model 2.1

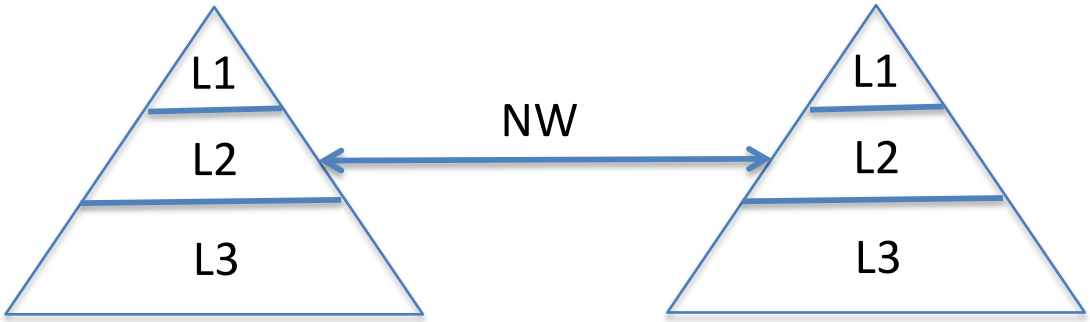
- Model 2.1: L1, L2 and L3 (NVM), network connects L2s
  - Data fits in L2, so don't need L3 – Is it worth using L3?



- Idea: use 2.5D algorithms that replicate data to reduce NW traffic
  - Ex: 2.5DMM moves $n^2/(cP)^{1/2}$ words over NW if enough memory for c copies of data $(1 \le c \le P^{1/3})$
  - Using L3 may let us increase c, at cost of L3 writes
- Performance model
  - c_2 = #copies using L2, c_3 = #copies using L3, so $c_3 > c_2$
  - β_{NW} = network BW, β_{23} = L3 write BW >> β_{32} = L3 read BW
  - Potential speedup = $(c_3/c_2)^{1/2}$ [β_{NW} / (β_{NW} + 1.5 β_{23} + β_{32})]

Parallel WA Algorithms – Model 2.2

- Model 2.2: L1, L2 and L3 (NVM), network connects L2s
  - Data fits in L3, not L2



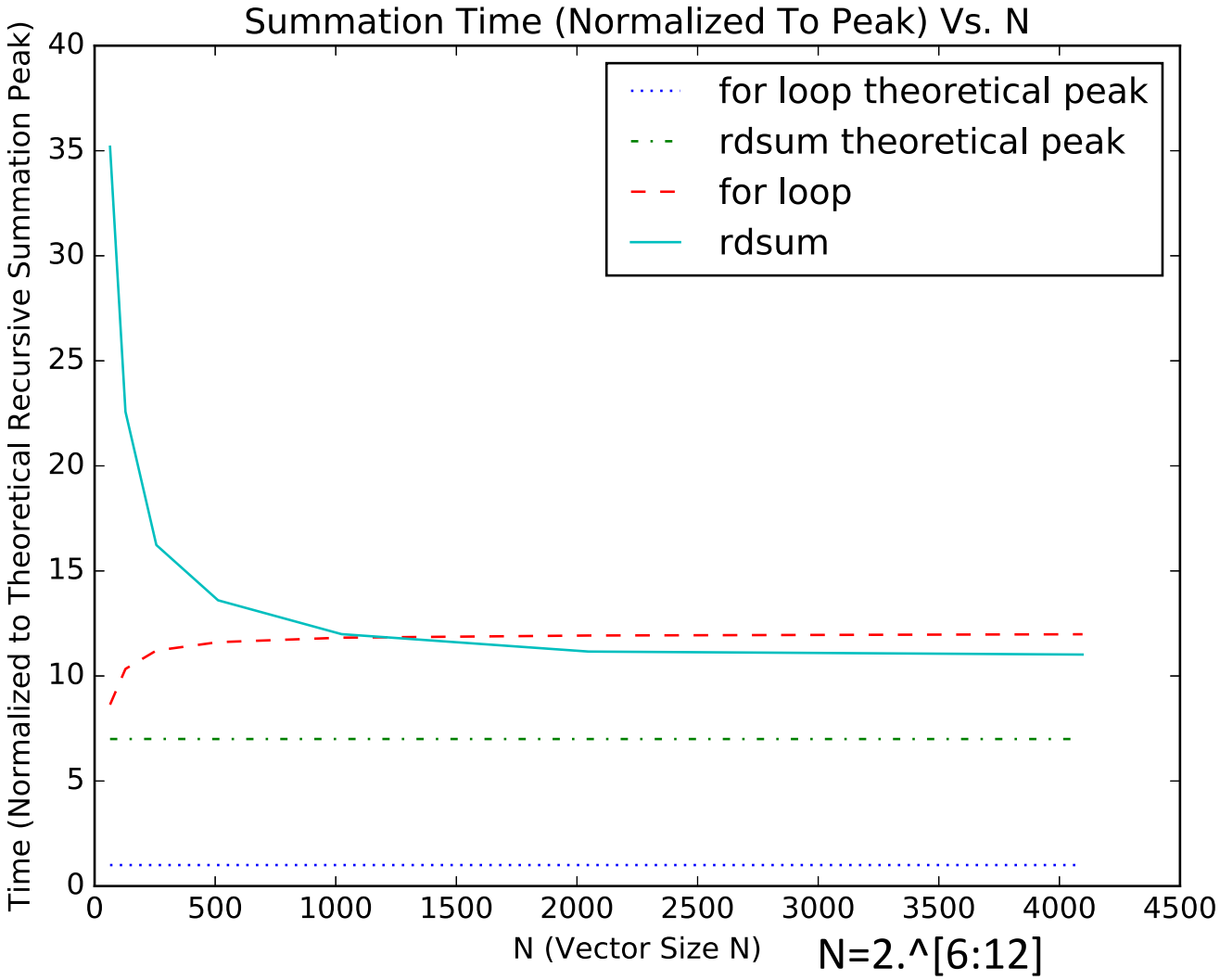
- Can we attain all lower bounds?
  - $W_{NW} = \Omega\ (n^2/(cP)^{1/2})$ words communicated over network
  - $W_{23} = \Omega\ (n^2/P)$ words written to L3 from L2
- Thm 4 (bad news): It is impossible to attain both lower bounds
  - See paper for details
- Good news: There are algorithms that can attain either bound (but not the other)
  - Alg 1: $W_{NW} = O\ (n^2/(cP)^{1/2})$ but $W_{23} = W_{NW}$
  - Alg 2: $W_{23} = O\ (n^2/P)$ but $W_{NW} = O\ (n^3/(P(\text{size\_of\_L2})^{1/2})$
  - Which one is best depends on algorithmic & HW parameters

Summation Performance

- Compare to gcc –O3 applied to:

  res=0; for (j=0; j<N; j++) { res += X[j]; }

- Reproducible sum faster for large N !



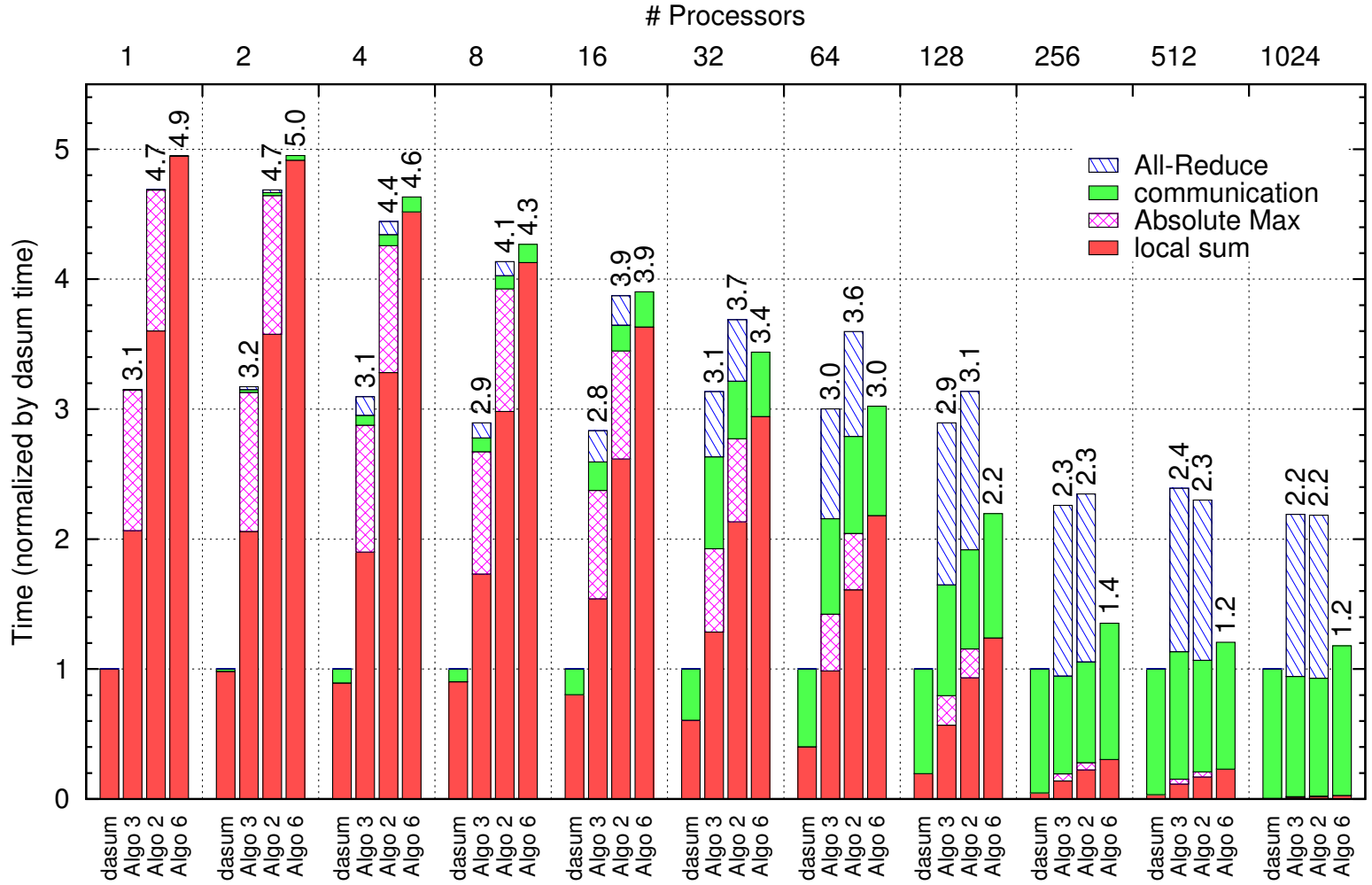
Performance results on 1024 proc Cray XC30

1.2x to 3.2x slowdown vs fastest (nonreproducible) code dasum

data for n=1M summands on up to p=1024 processors

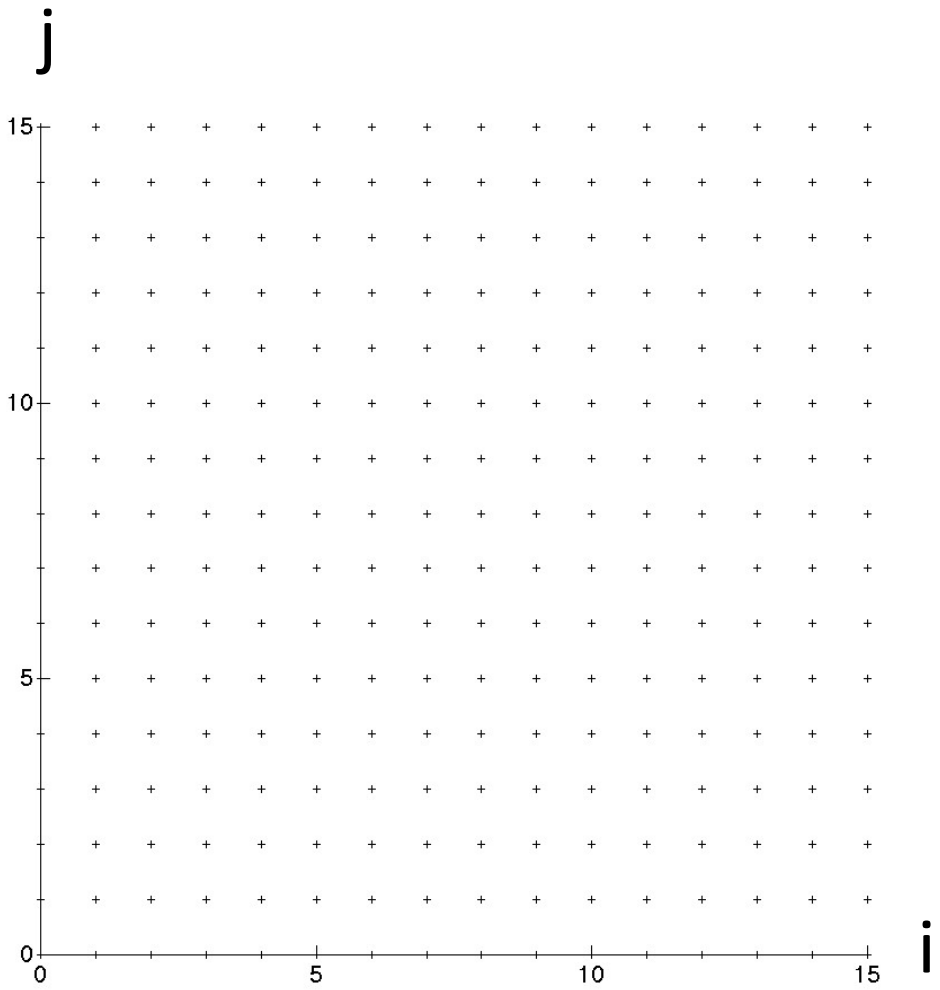
3 reproducible sum algorithms compared, best one depends on n, p

code and papers at bebop.cs.berkeley.edu/reproblas



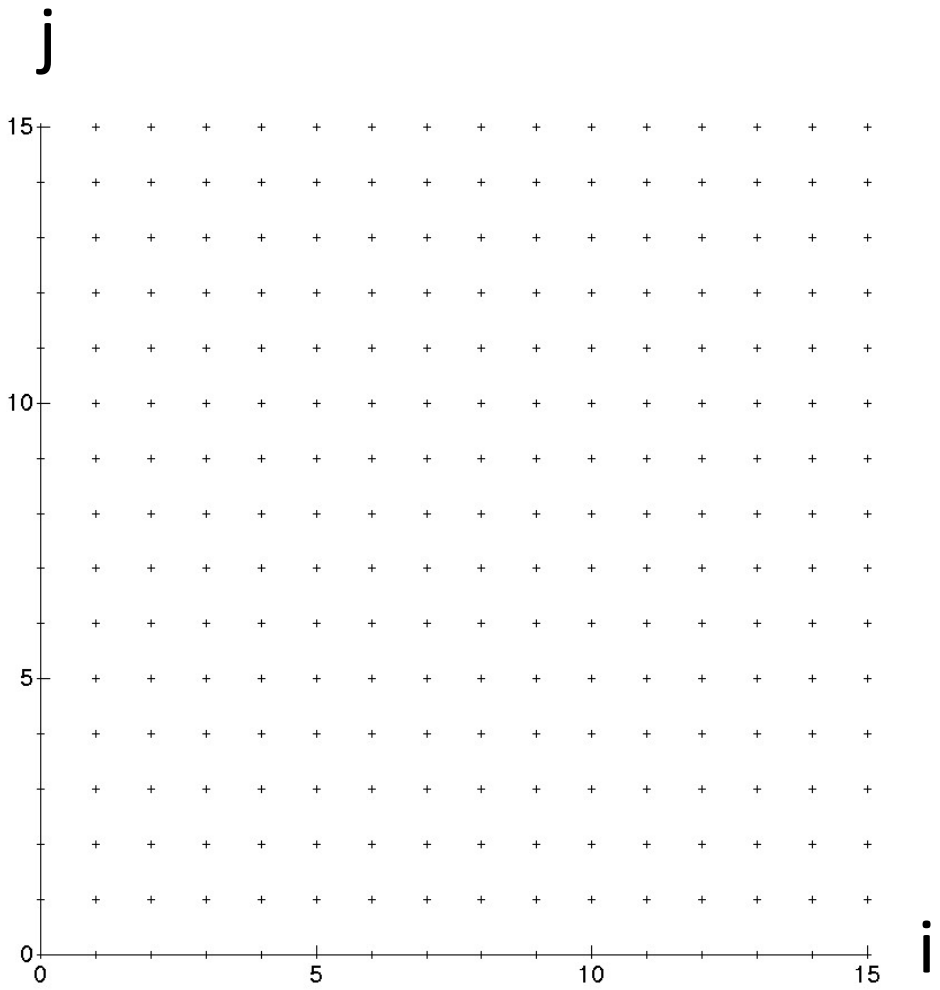
Optimal tiling for usual n-body

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



Optimal tiling for usual n-body

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

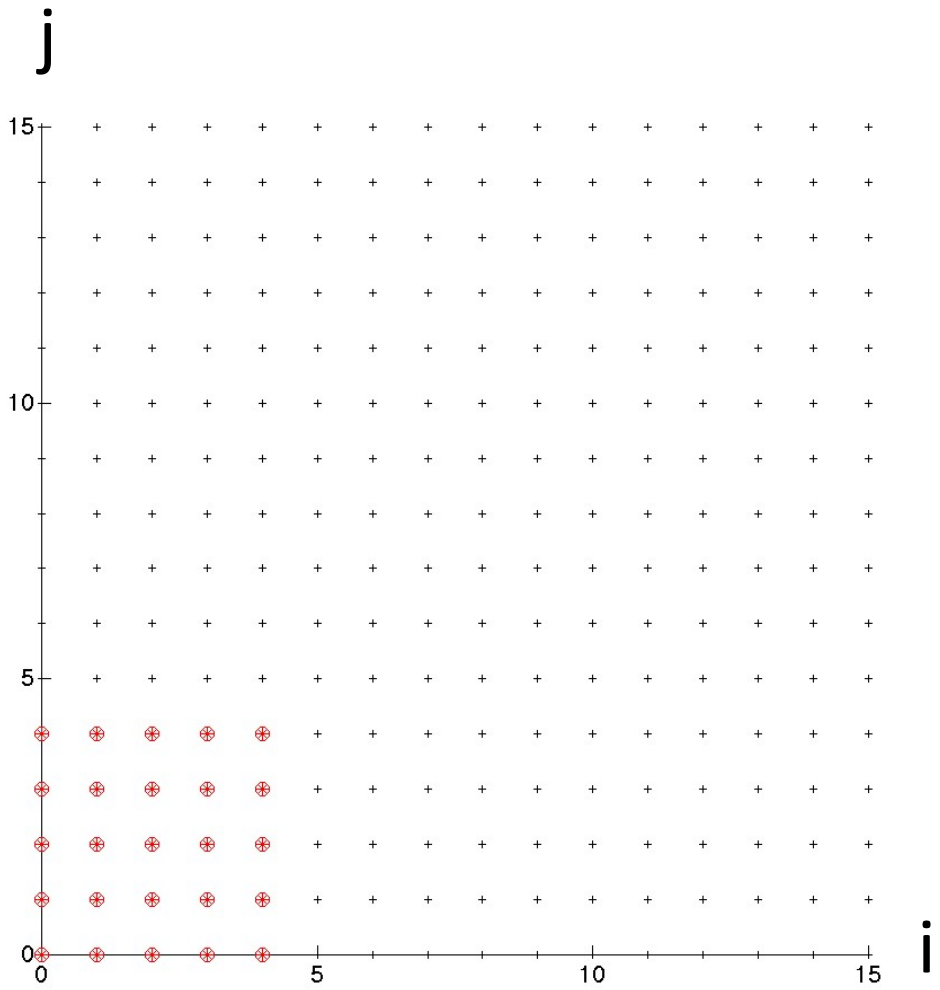


Optimal tiling for usual n-body

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

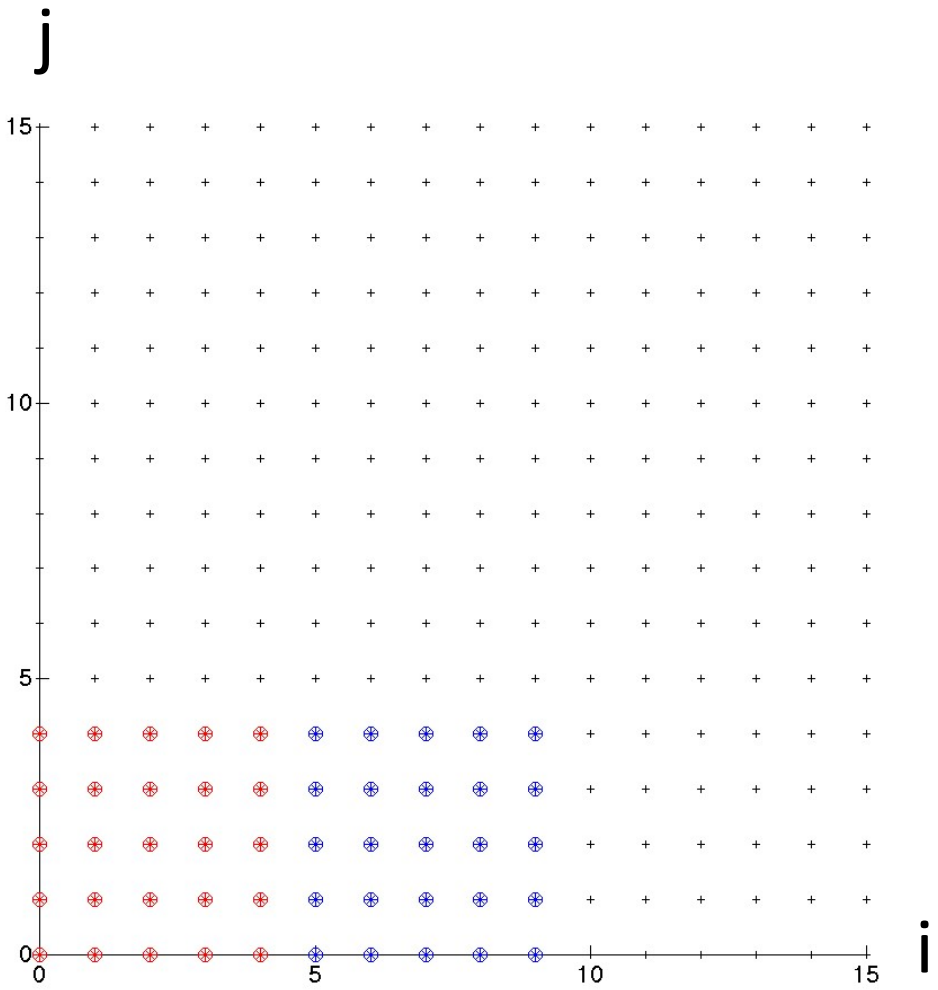
Tiling, with M = 10:

- 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



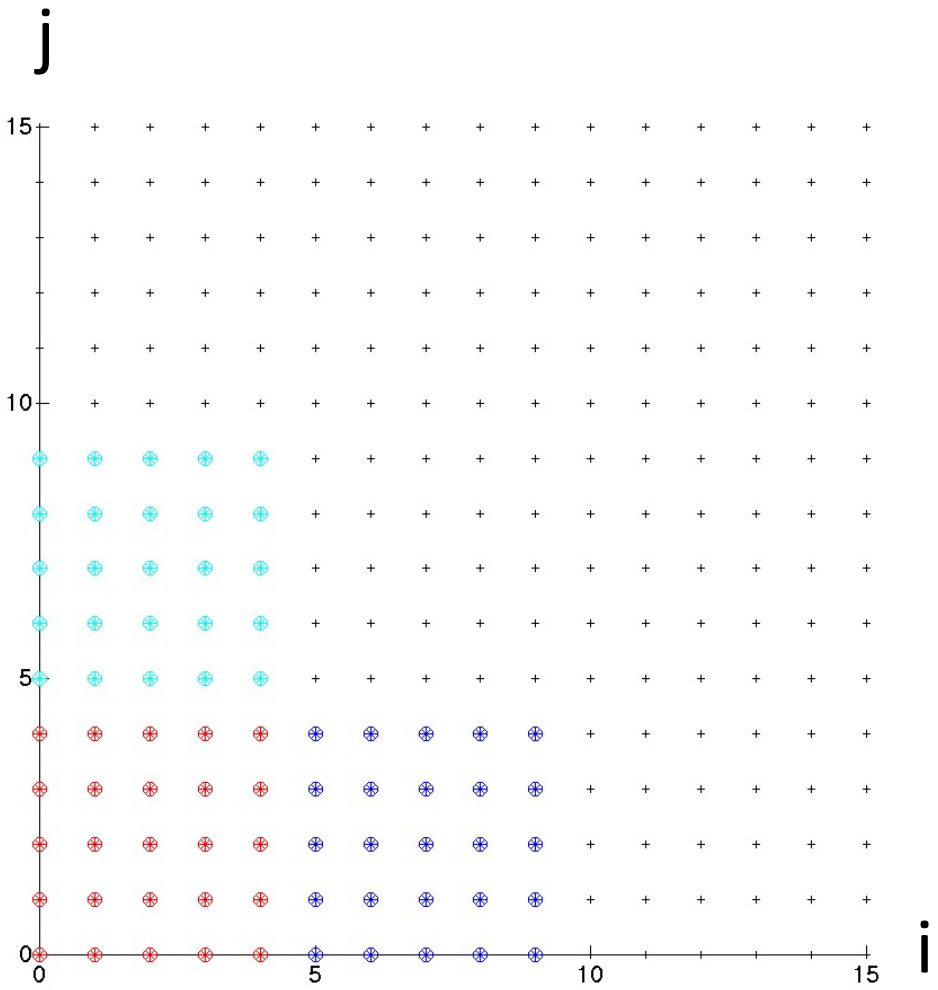
Optimal tiling for usual n-body

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



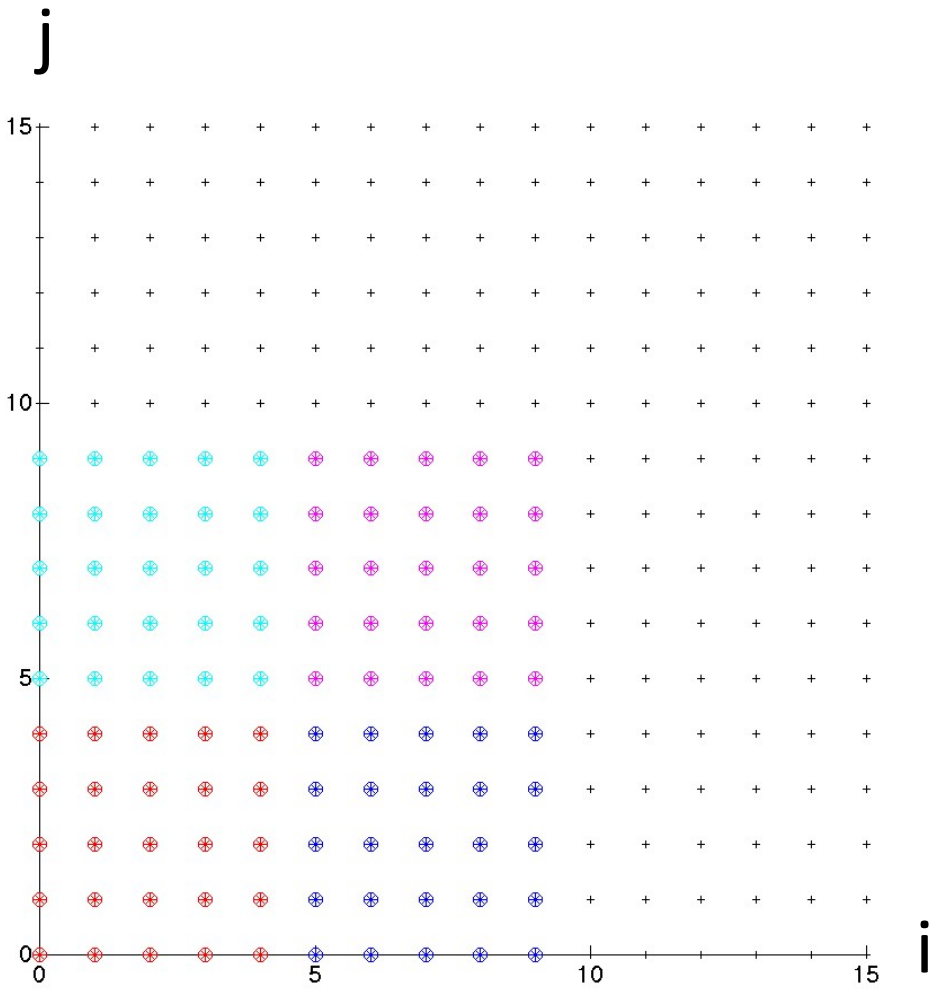
Optimal tiling for usual n-body

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



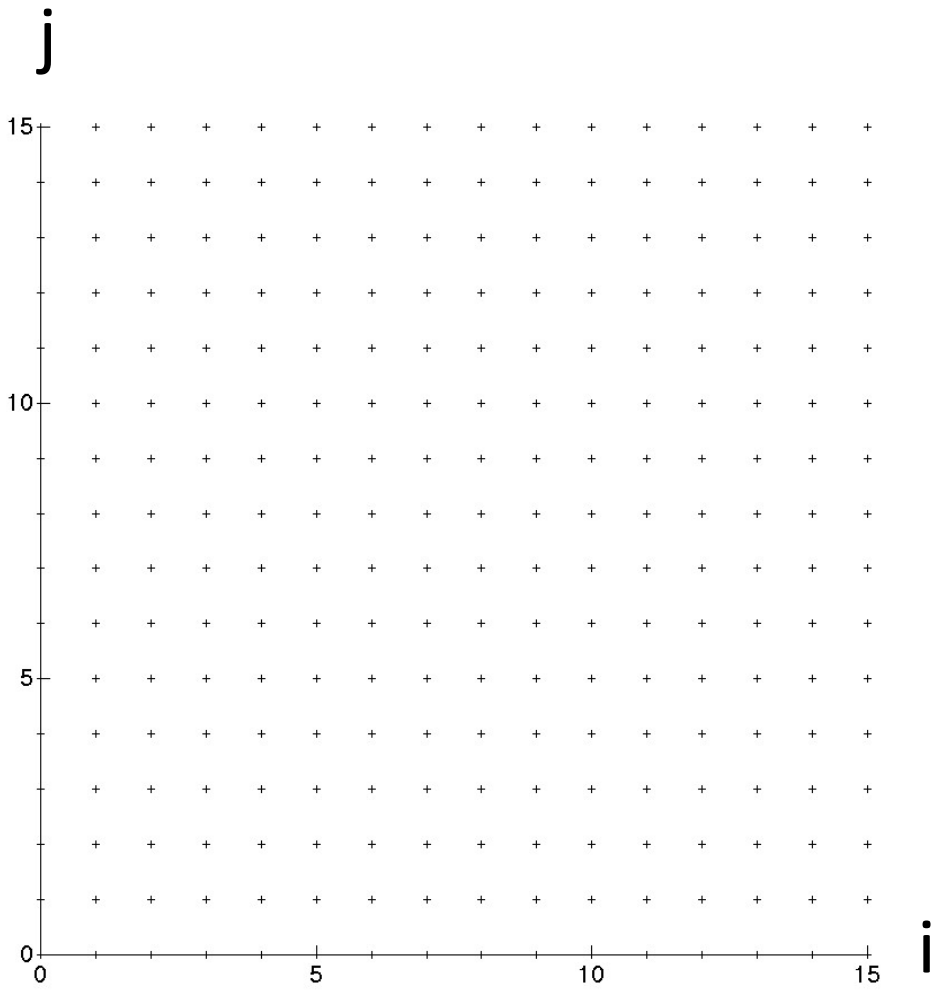
Optimal tiling for usual n-body

```
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(i+j)
```

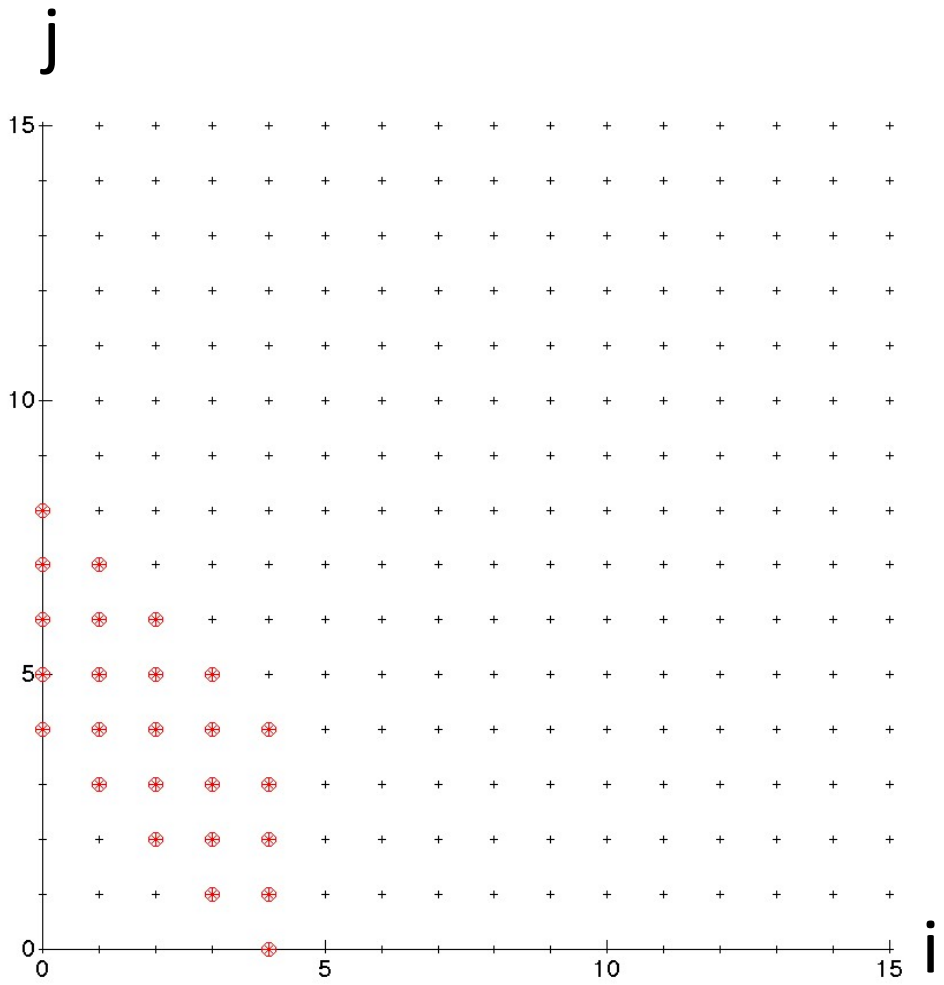


Optimal tiling for “slanted” n-body

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

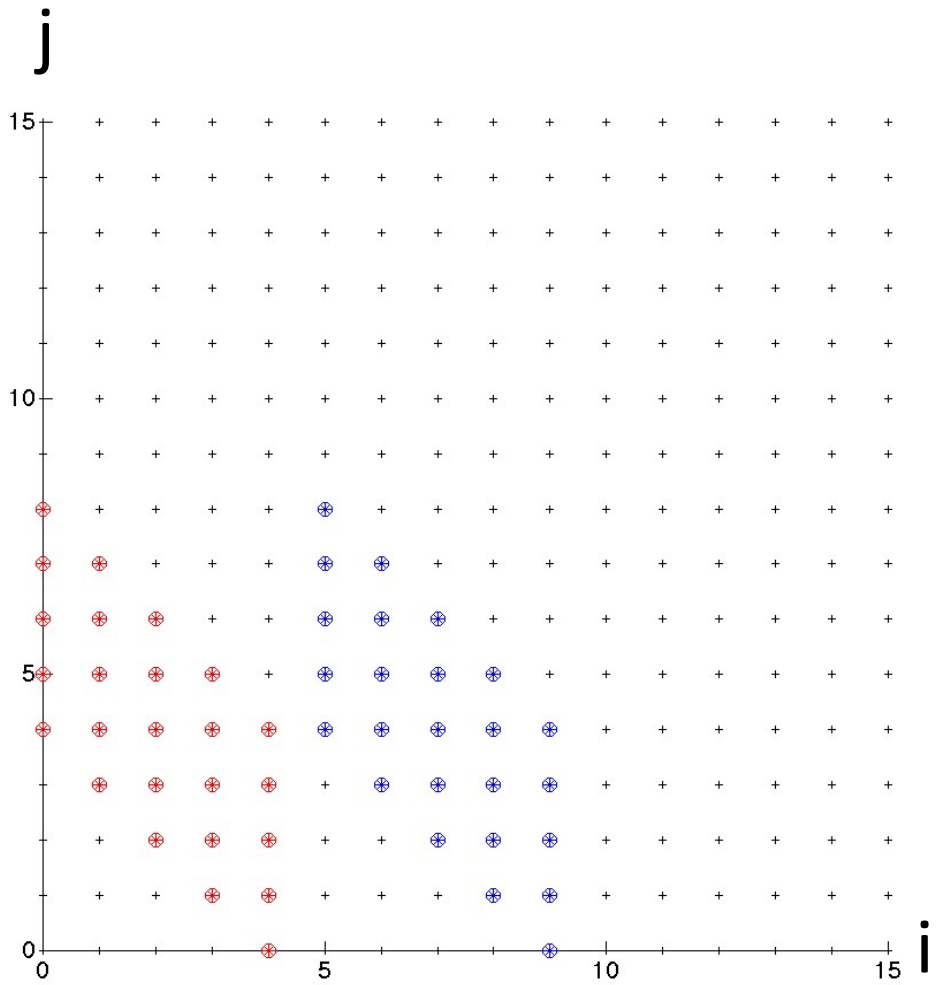
Tiling, with M = 10:

- 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



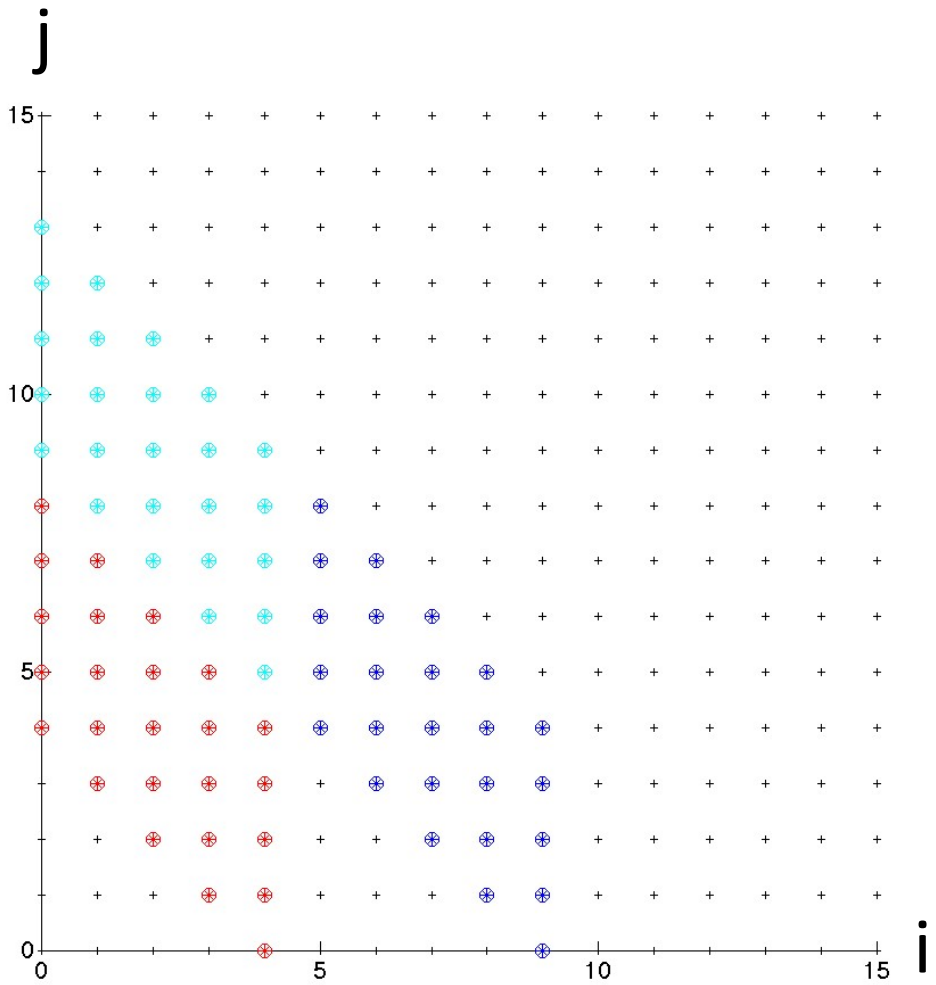
Optimal tiling for “slanted” n-body

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



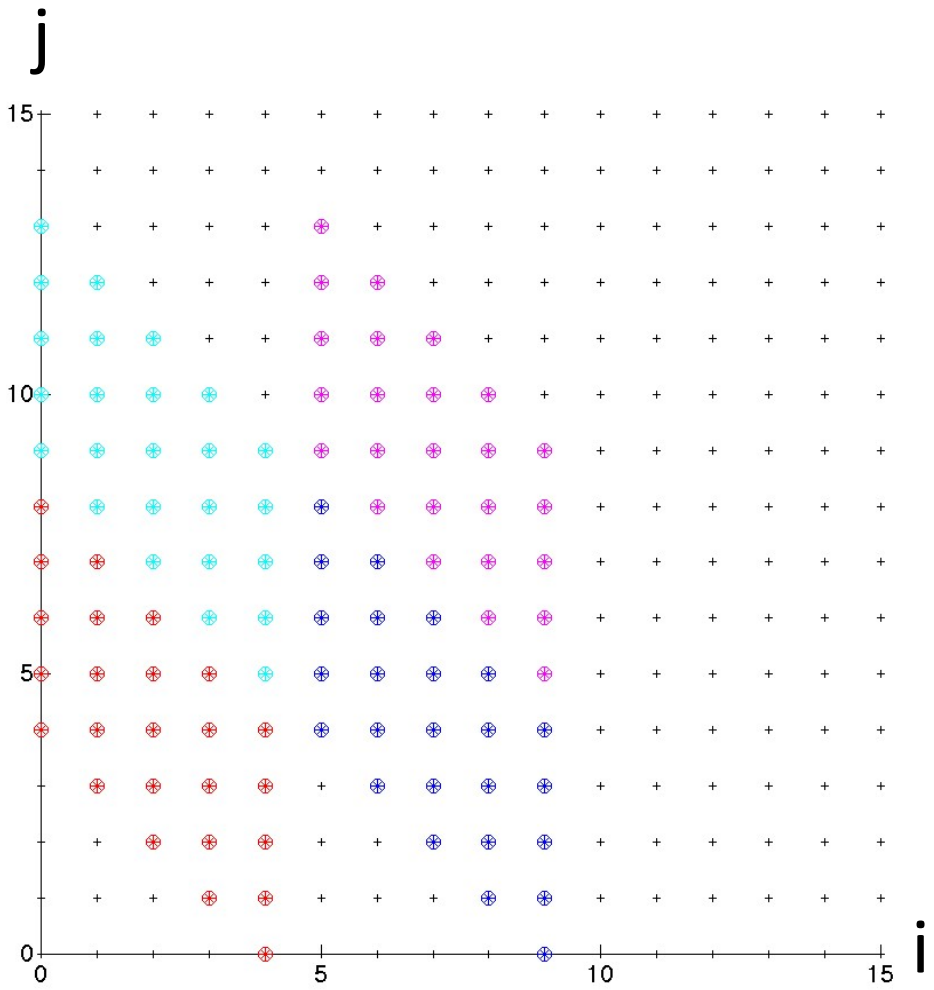
Optimal tiling for “slanted” n-body

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



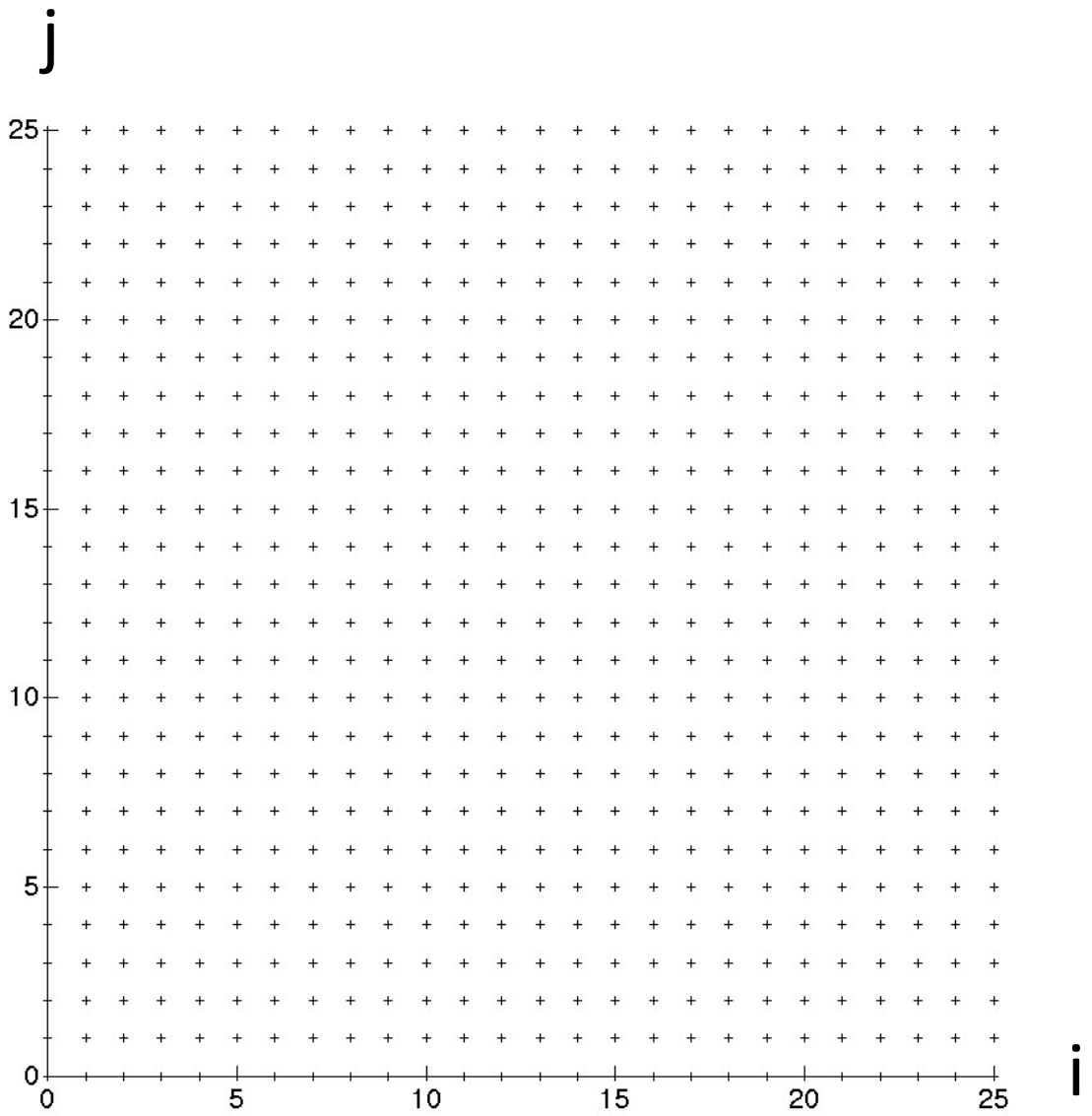
Optimal tiling for “slanted” n-body

```
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)
```

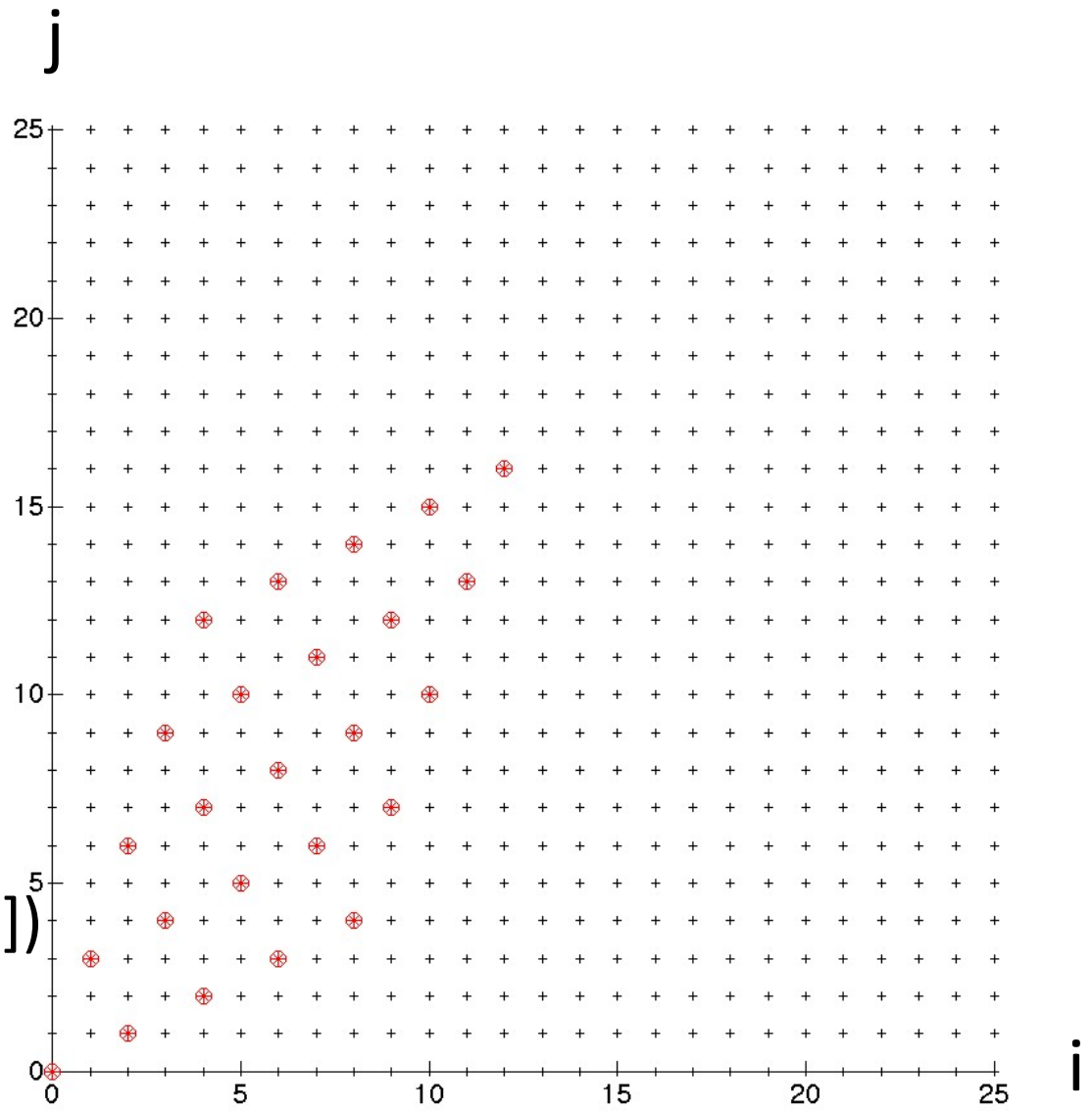


Optimal tiling for “twisted” n-body

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

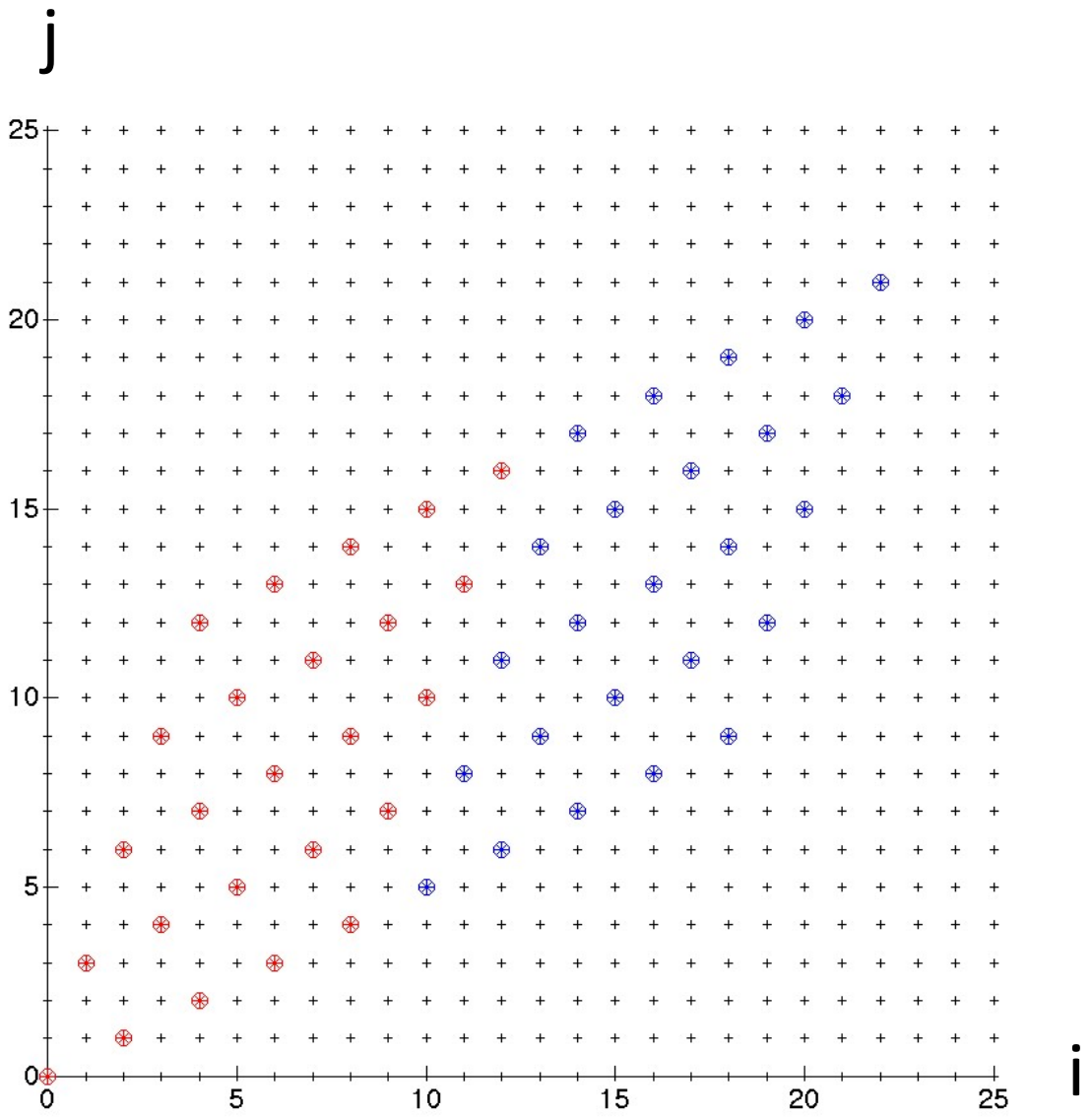
Tiling, with M = 10:

- 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



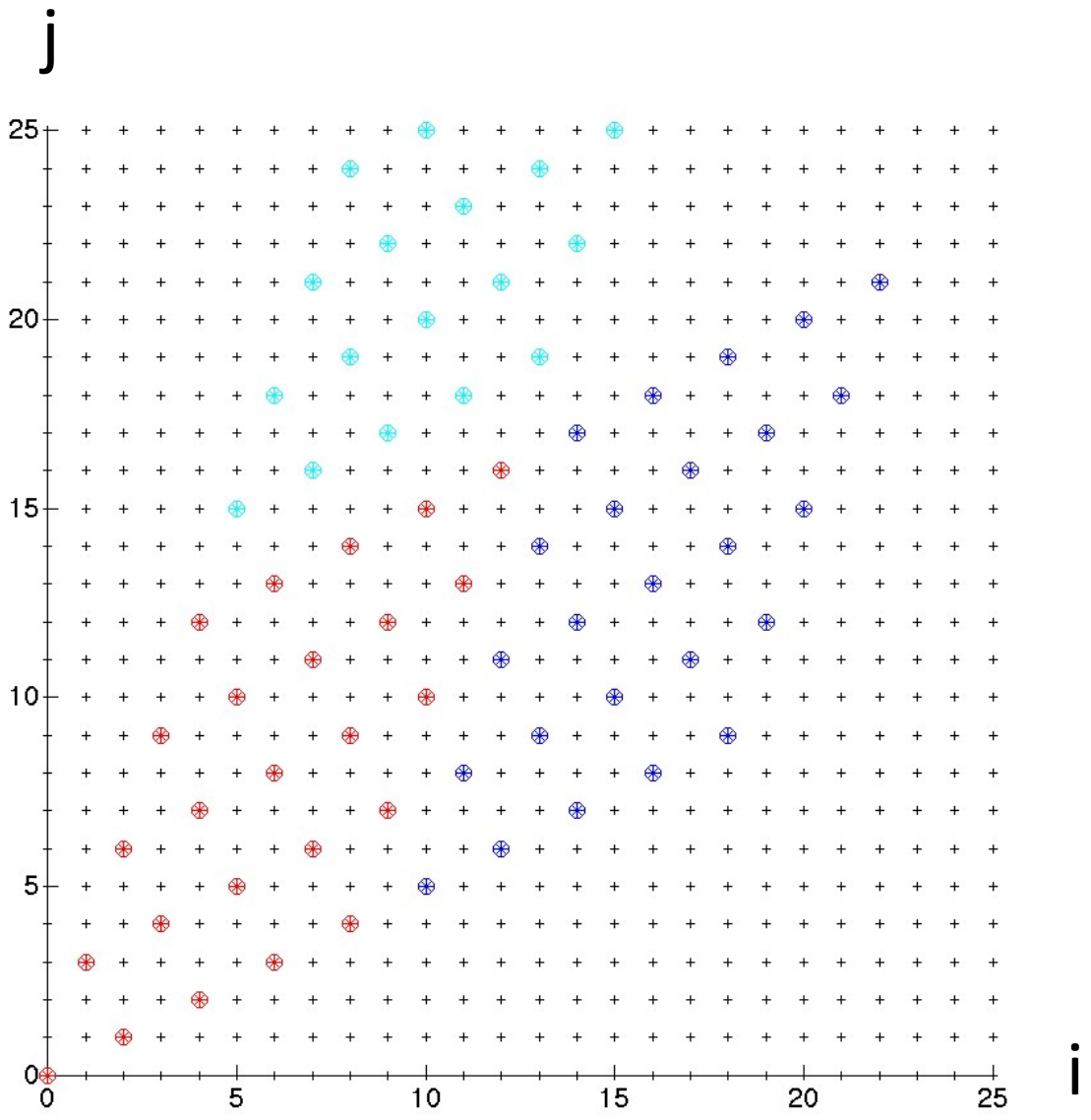
Optimal tiling for “twisted” n-body

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



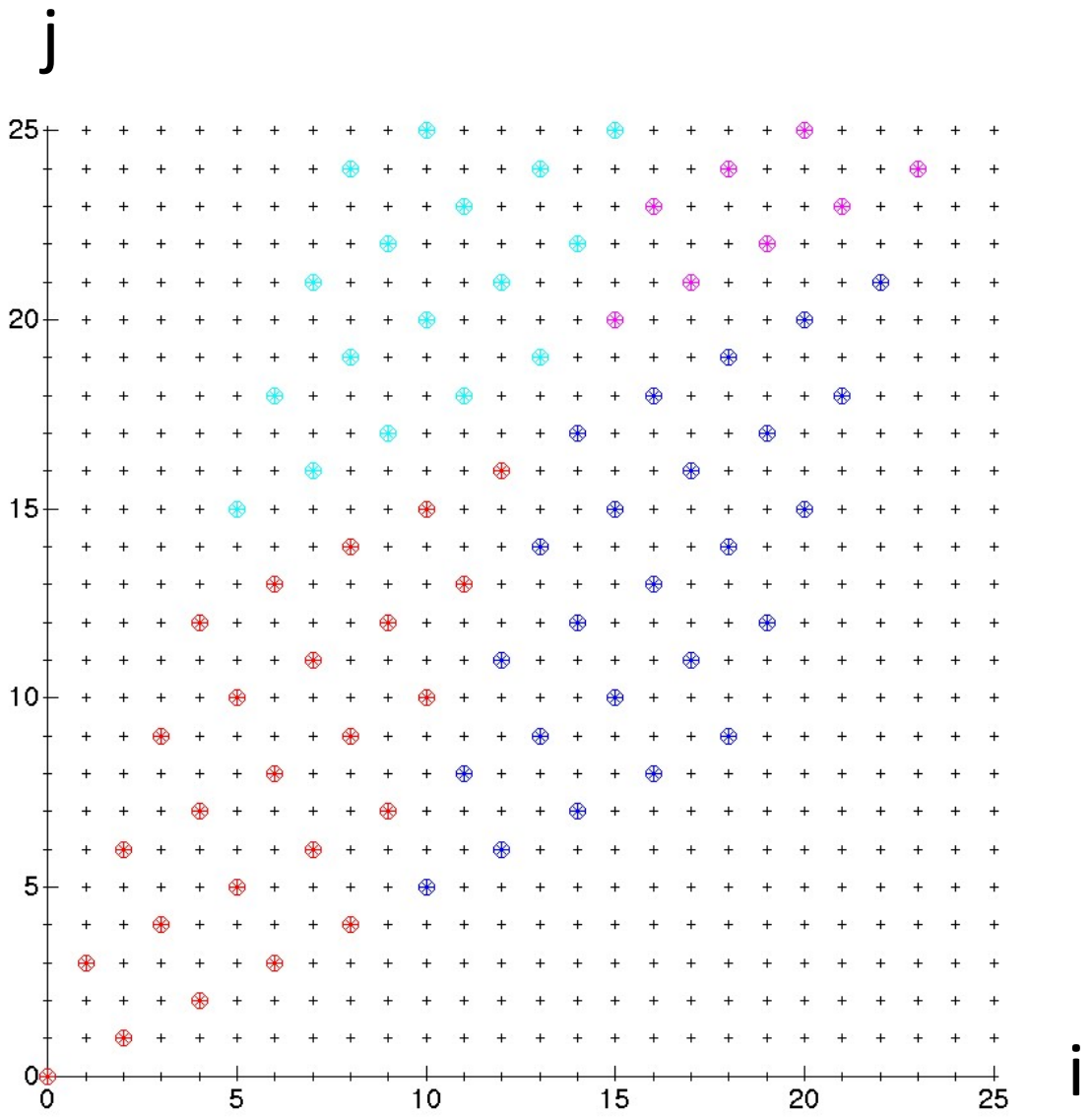
Optimal tiling for “twisted” n-body

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



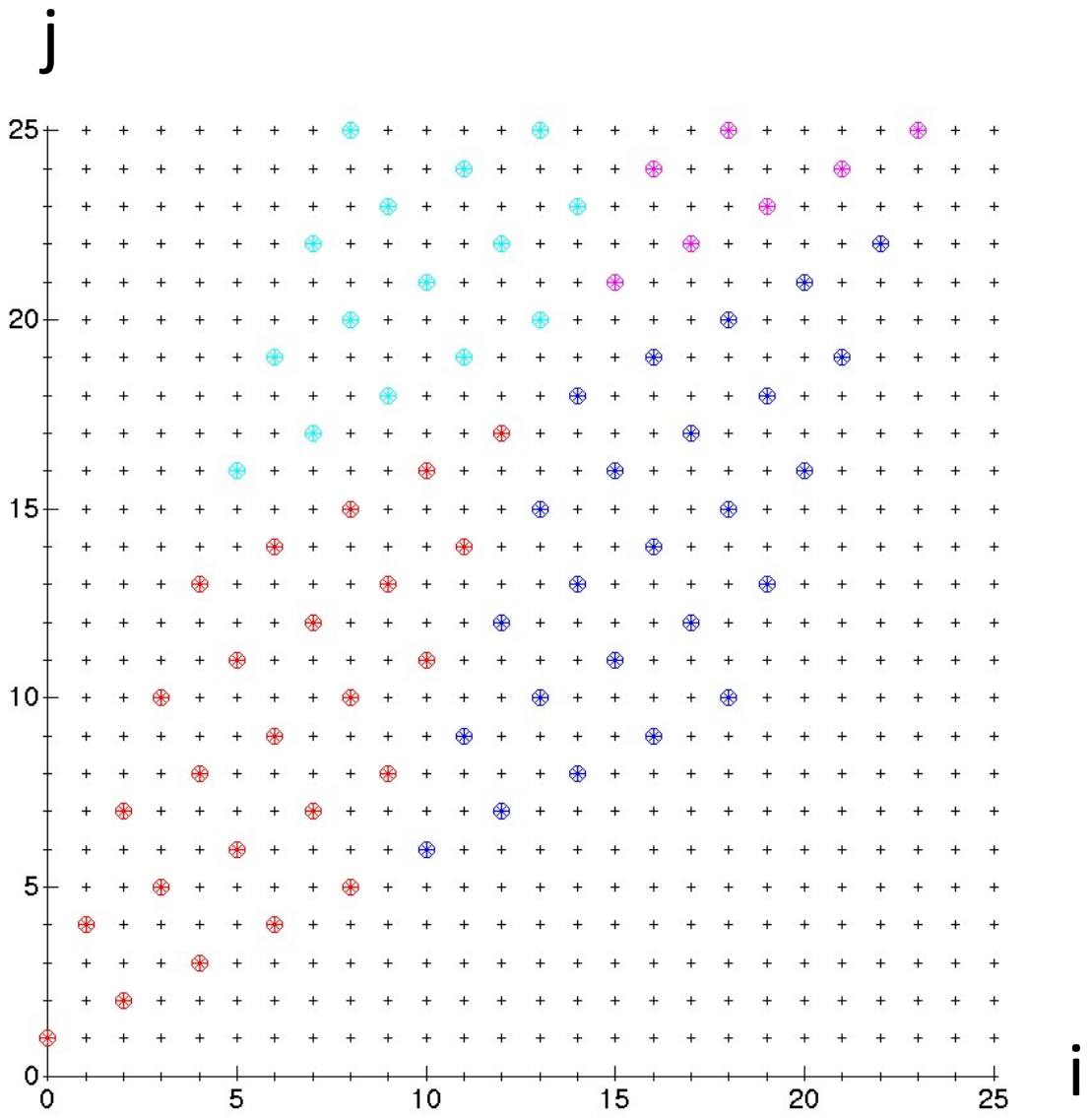
Optimal tiling for “twisted” n-body

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



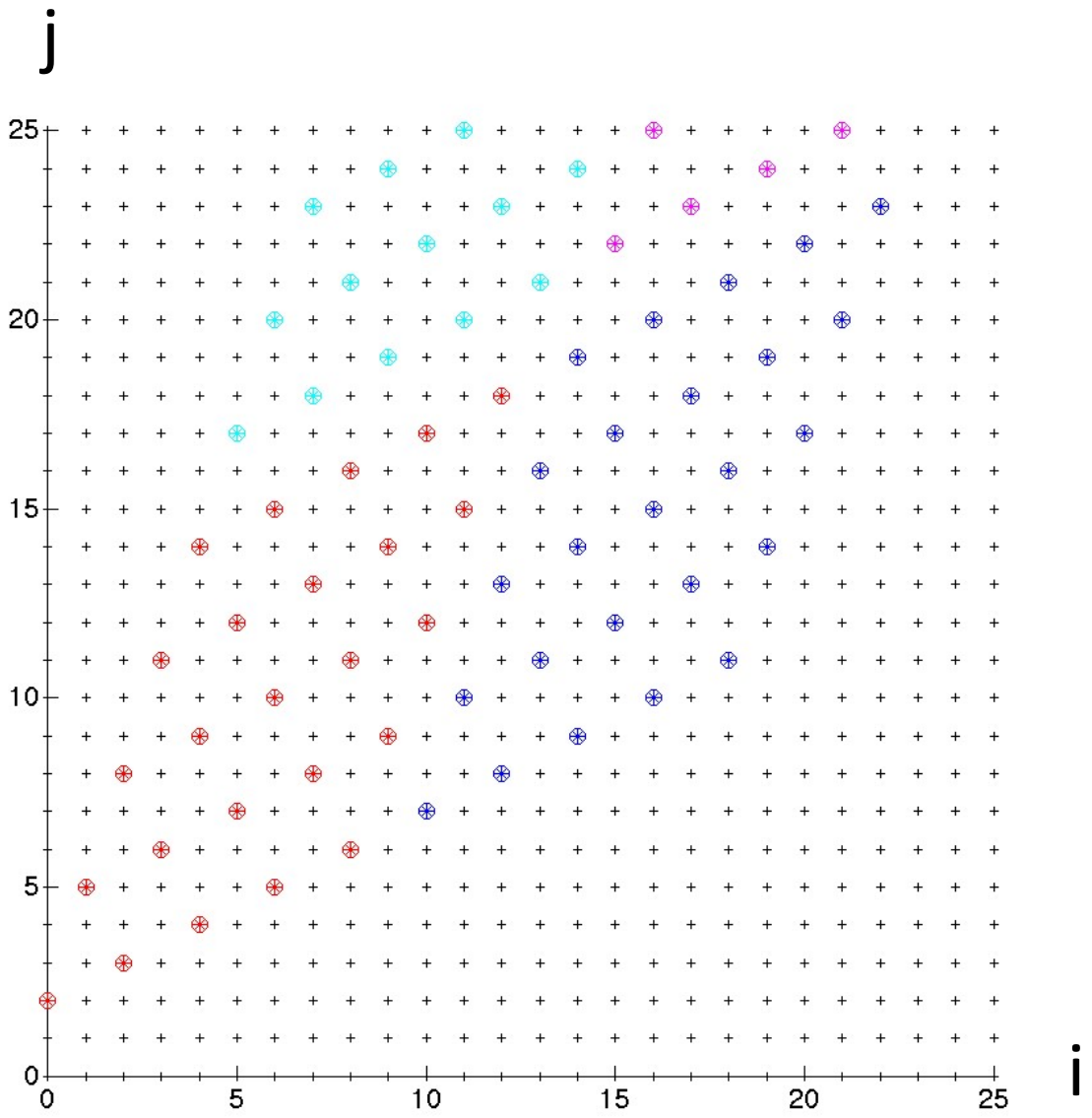
Optimal tiling for “twisted” n-body

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



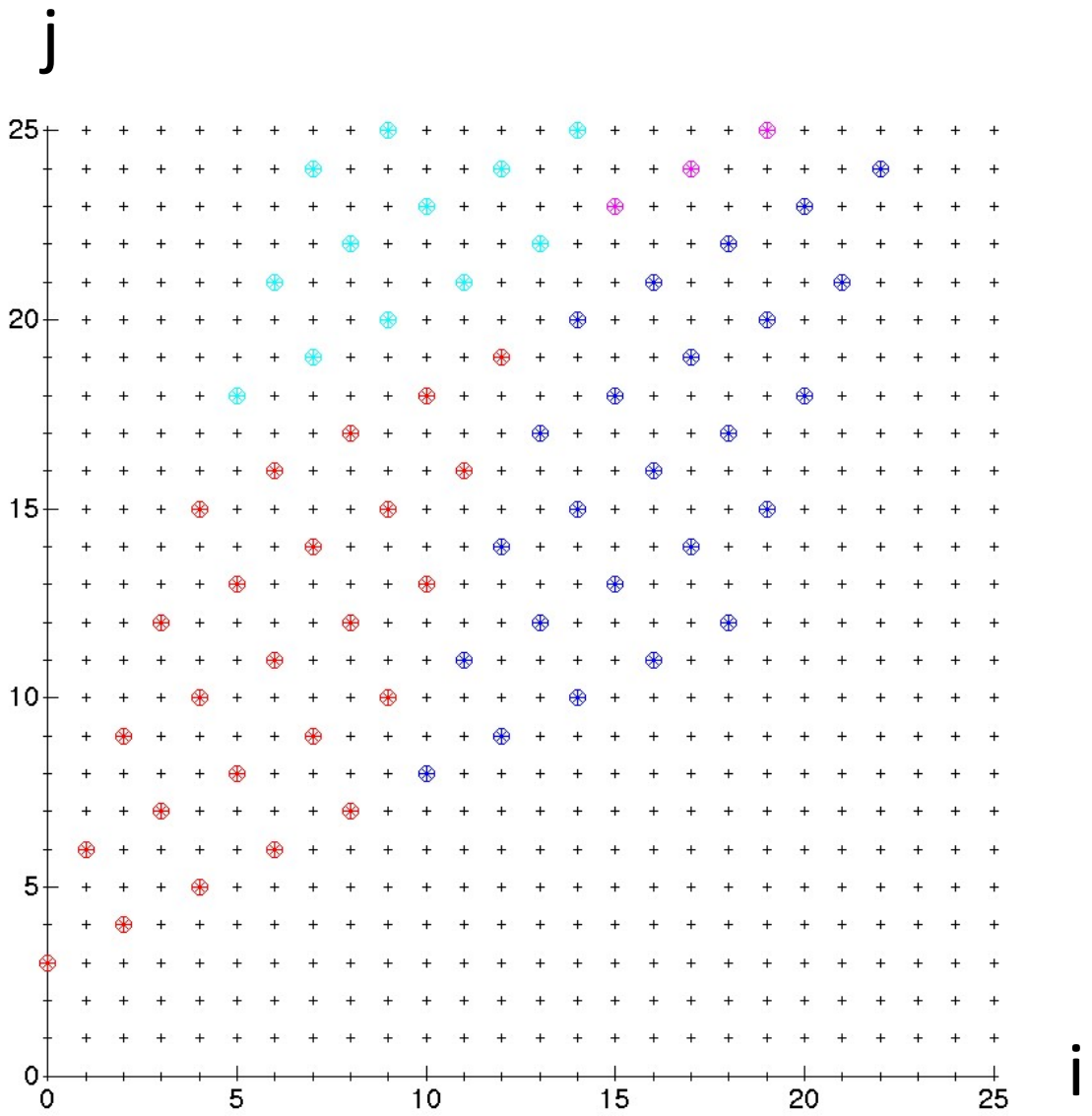
Optimal tiling for “twisted” n-body

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



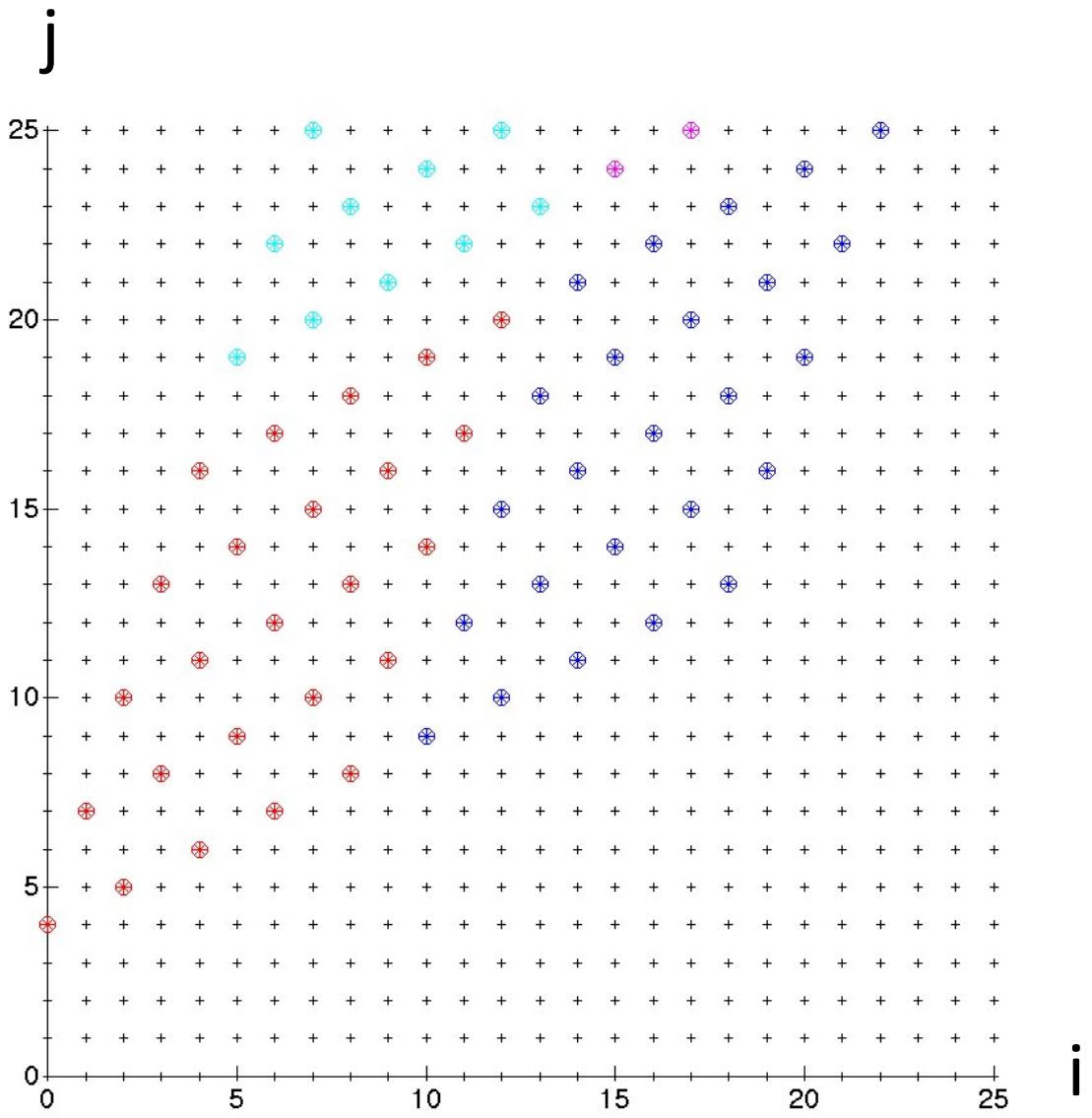
Optimal tiling for “twisted” n-body

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



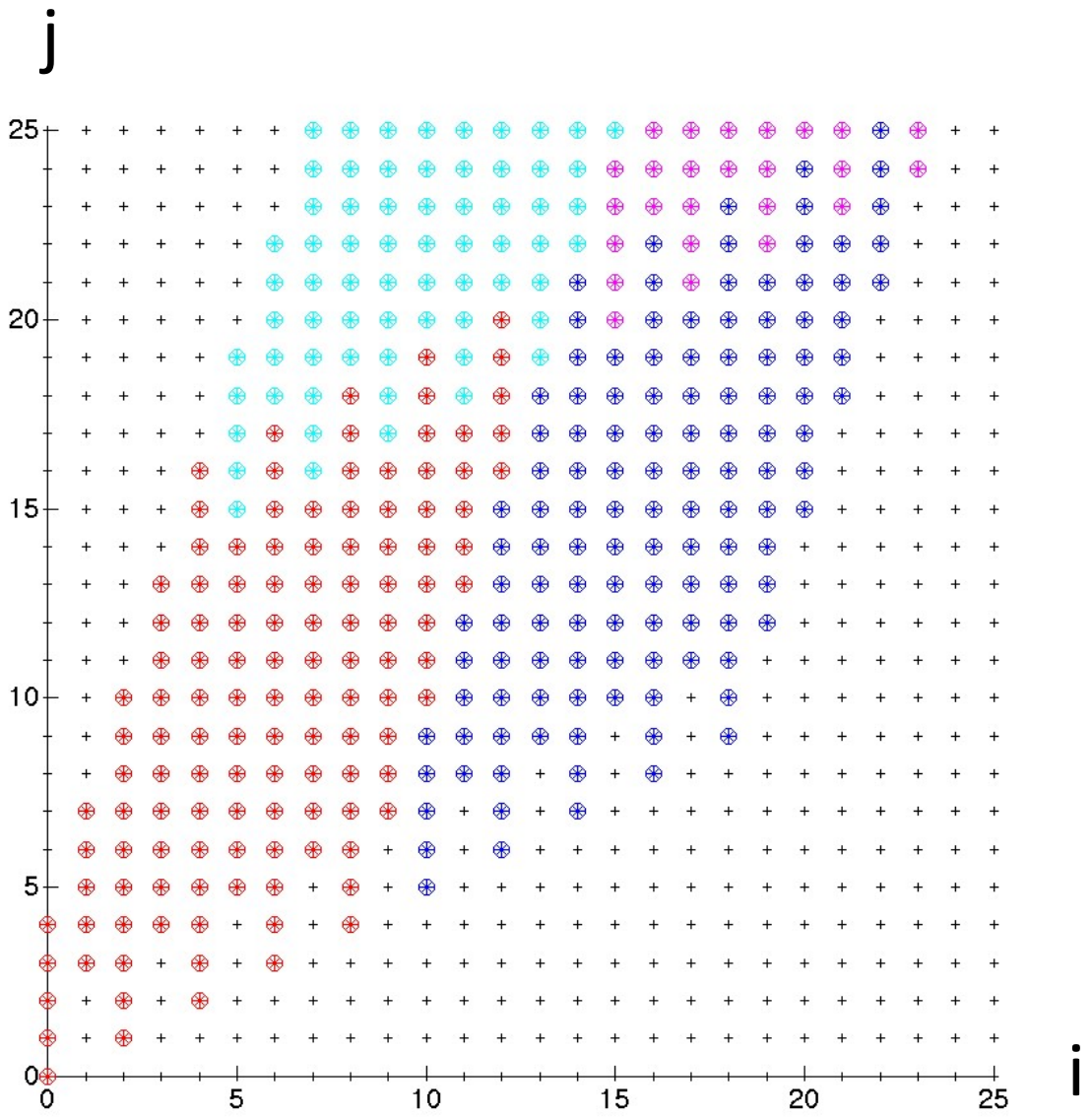
Optimal tiling for “twisted” n-body

```
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)
```



Future Work

- BLAS Standard
  - Get community input
  - Converge on final version
  - Reference implementations (using current IEEE 754 standard)
- Reproducible summation
  - Shepherd 754 standard through voting process
  - Provide reference (software) implementation of new operations
  - Use to provide new ReproBLAS
    - Partial implementation at bebop.cs.berkeley.edu/reproblas
- Consistent exception handling for other libraries
  - Sca/LAPACK, ,…
  - Develop tools to help automate consistency analyses: Abstract Interpretation?

Outline

- Linear Algebra
  - Communication Lower Bounds for classical direct linear algebra
  - CA 2.5D Matmul
  - **TSQR - Tall-Skinny QR**
  - Iterative Methods for linear algebra (GMRES)
- Machine Learning
  - Coordinate Descent (LASSO)
  - Training Neural Nets – “ImageNet training in minutes”
- And Beyond
  - Extending communication lower bounds and optimal algorithms to general loop nests
- Summary

TSQR: QR of a Tall, Skinny matrix

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

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

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

TSQR: QR of a Tall, Skinny matrix

$$W = \begin{pmatrix} W_0 \\ W_1 \\ W_2 \\ W_3 \end{pmatrix} = \begin{pmatrix} Q_{00}\,R_{00} \\ Q_{10}\,R_{10} \\ Q_{20}\,R_{20} \\ Q_{30}\,R_{30} \end{pmatrix} = \begin{pmatrix} Q_{00} & & & \\ & Q_{10} & & \\ & & Q_{20} & \\ & & & Q_{30} \end{pmatrix} \cdot \begin{pmatrix} R_{00} \\ R_{10} \\ R_{20} \\ R_{30} \end{pmatrix}$$

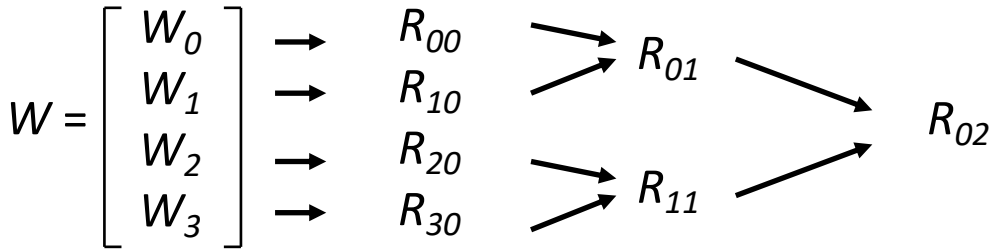
$$\begin{pmatrix} R_{00} \\ R_{10} \\ R_{20} \\ R_{30} \end{pmatrix} = \begin{pmatrix} Q_{01}\,R_{01} \\ Q_{11}\,R_{11} \end{pmatrix} = \begin{pmatrix} Q_{01} & \\ & Q_{11} \end{pmatrix} \cdot \begin{pmatrix} R_{01} \\ R_{11} \end{pmatrix}$$

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

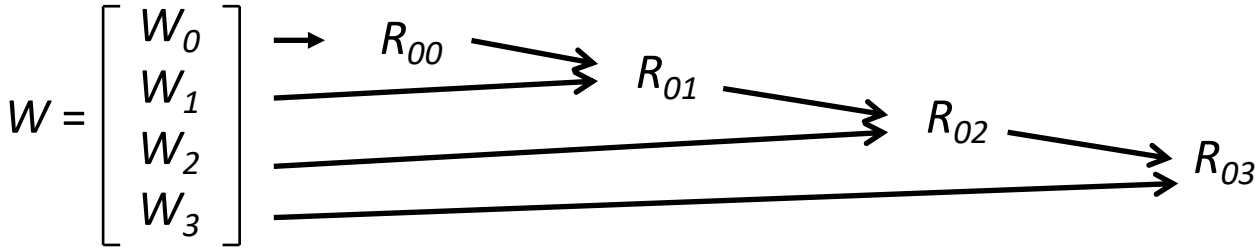
Output = { Q_{00}, Q_{10}, Q_{20}, Q_{30}, Q_{01}, Q_{11}, Q_{02}, R_{02} }

TSQR: An Architecture-Dependent Algorithm

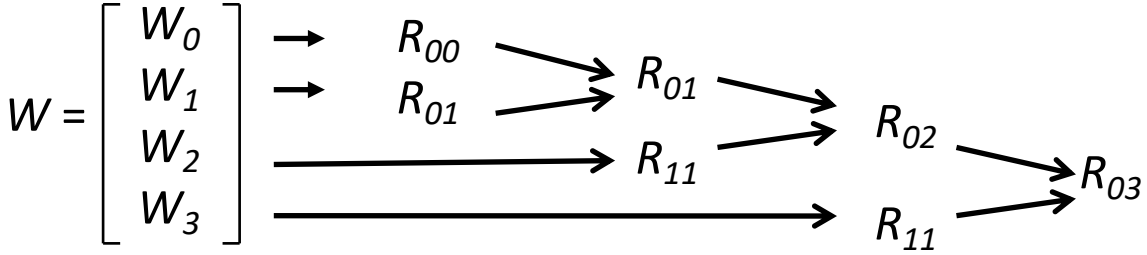
Parallel:



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

(D., Grigori, Hoemmen, Langou)

Outline

- Linear Algebra
  - Communication Lower Bounds for classical direct linear algebra
  - CA 2.5D Matmul
  - TSQR - Tall-Skinny QR
  - **Iterative Methods for linear algebra (GMRES)**
- Machine Learning
  - Coordinate Descent (LASSO)
  - Training Neural Nets – “ImageNet training in minutes”
- And Beyond
  - Extending communication lower bounds and optimal algorithms to general loop nests
- Summary

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

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!
- Fix: replace W by [v, $p_1(A)$v, $p_2(A)$v, …, $p_k(A)$v] (Hoemmen)
- Up to **2.3x** speedup for GMRES on 8 core Intel Clovertown
- Up to **4.2x** speedup for BiCGStab on 24K core Cray XE6 (Carson)

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 $\left\lfloor \frac{s}{2} \right\rfloor - 1$, 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. $\quad \alpha_j := (r_j, r_0^*)/(Ap_j, r_0^*)$
5. $\quad s_j := r_j - \alpha_j Ap_j$
6. $\quad \omega_j := (As_j, s_j)/(As_j, As_j)$
7. $\quad x_{j+1} := x_j + \alpha_j p_j + \omega_j s_j$
8. $\quad r_{j+1} := s_j - \omega_j As_j$
9. $\quad \beta_j := \frac{(r_{j+1}, r_0^*)}{(r_j, r_0^*)} \frac{\alpha_j}{\omega_j}$
10. $\quad p_{j+1} := r_{j+1} + \beta_j(p_j - \omega_j Ap_j)$
11. EndDo

Outline

- Linear Algebra
  - Communication Lower Bounds for classical direct linear algebra
  - CA 2.5D Matmul
  - TSQR - Tall-Skinny QR
  - Iterative Methods for linear algebra (GMRES)
- **Machine Learning**
  - **Coordinate Descent (LASSO)**
  - Training Neural Nets – "ImageNet training in minutes"
- And Beyond
  - Extending communication lower bounds and optimal algorithms to general loop nests
- Summary

Communication-Avoiding ML (1/2)

- Apply "unrolling" idea from Krylov subspace methods to (block) coordinate descent
- Illustrate with LASSO:

$$\underset{x}{\operatorname{argmin}} \parallel Ax - b \parallel_2^2 + \lambda \parallel x \parallel_1$$

  - Applies to ridge regression, proximal least squares, SVMs, kernel methods
  - Works as long as nonlinearity just in inner loop

Communication-Avoiding ML (2/2)

- Coordinate Descent (CD)

Until convergence do (H times):
  Randomly select a data point A_i
  Solve minimization problem for A_i
  Update solution vector

Vector ops

Flops = O(Hm/P)
Messages = O(H log P)
Words = O(H)

- Communication-Avoiding CD

Until convergence do:
  Randomly select s data points $\mathcal{A}$
  Compute Gram matrix $\mathcal{A}^T\mathcal{A}$
  Solve minimization problem
    for all data points in $\mathcal{A}$
  Update solution vector

Matmul,
Vector ops

Flops = O(Hms/P + Hs)
Messages = O(H/s log P)
Words = O(Hs)

– Up to **5.1x** speedup on 3K core Cray XC30

(Devarakonda, D., Fountoulakis, Mahoney, IPDPS 18)

Outline

- Linear Algebra
  - Communication Lower Bounds for classical direct linear algebra
  - CA 2.5D Matmul
  - TSQR - Tall-Skinny QR
  - Iterative Methods for linear algebra (GMRES)
- Machine Learning
  - Coordinate Descent (LASSO)
  - **Training Neural Nets – "ImageNet training in minutes"**
- And Beyond
  - Extending communication lower bounds and optimal algorithms to general loop nests
- Summary

Training Neural Nets by Mini-Batch Stochastic Gradient Descent (SGD)

(You, Zhang, Hsieh, D., Keutzer, IPDPS 18)

- Iterate:
  - Pick a mini-batch of B data points
  - Update weights $W = W - \eta \cdot \nabla L(W)$
    - η = learning rate
    - $\nabla L(W)$ = gradient
- Data parallel version on P processors
  - Data partitioned, each processor gets B/P points
  - W_i replicated
  - Each processor computes $\nabla L(W)_i$ wrt its data
  - All-reduce: each processor computes

    $W_i = W_i - (\eta/\mathrm{P}) \cdot \Sigma_{i=1}^{P} \nabla L(W)_i$

SGD: $W_i = W_i - (\eta/\mathrm{P}) \cdot \Sigma_{i=1}^{P} \nabla L(W)_i$

- Increase P to go faster: What are the bottlenecks?
- B/P decreases ⇒ less work per processor
  - Small matrix operations ⇒ locally communication bound
- Cost of each reduction $\Sigma_i \nabla L(W)_i$ grows
- Solution: increase B along with P
  - Maintain B/P ⇒ maintain processor efficiency
  - Try to converge in same #epochs (passes over data)
    - Same overall work, fewer reductions
- Oops: Convergence can be much worse
  - Convergence rate, test accuracy

Improving SGD convergence as B grows

- Facebook's strategy: adjust learning rate η
  - Increase B to kB $\Rightarrow$ increase η to $\mathrm{k}\eta$
  - Warmup rule: Start with smaller η, then increase
- Only worked up to B=1K for AlexNet (tried lots of tuning)
- Fix: Add Layer-wise Adaptive Rate Scaling (LARS)
  - $\| W \| / \| \nabla L(W) \|$ can vary by 233x between AlexNet layers
  - Let η be proportional to $\| W \| / \| \nabla L(W) \|$
  - (You, Gitman, Ginsburg, 2017)
  - Also need momentum, weight decay

ImageNet Training in Minutes

Speedup for AlexNet (for batchsize = 32K, changed LRN to BN)

Batch Size	Epochs	Top-1 Accuracy	Platform	Time
256	100	58.7%	8-core + K20 GPU	144 hrs
512	100	58.8%	DGX-1 station	6h 10m
4096	100	58.4%	DGX-1 station	2h 19m
32k	100	58.6%	512 KNLs	24m
32k	100	58.6%	1024 CPUs	**11m**

Speedup for ResNet50

Batch Size	Epochs	Top-1 Accuracy	Platform	Time
32	90	75.3%	CPU + M40 GPU	336h
256	90	75.3%	16 KNLs	45h
32K	90	75.4%	512 KNLs	60m
32K	90	75.4%	1600 CPUs	32m
32K	90	75.4%	2048 KNLs	**20m**

135x

ImageNet Training in Minutes

- Best Paper Prize at ICPP 2018
- Open Source in Caffe, NVIDIA Caffe, Facebook Caffe 2 (PyTorch)
- Media coverage by CACM, EureKalert, Intel, NSF, Science Daily, Science NewsLine, etc.
- Subsequent work at Tencent reached 4 minutes

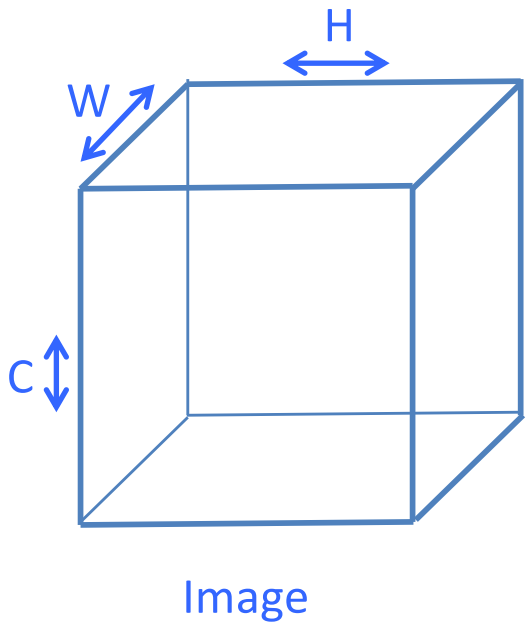
Outline

- Linear Algebra
  - Communication Lower Bounds for classical direct linear algebra
  - CA 2.5D Matmul
  - TSQR - Tall-Skinny QR
  - Iterative Methods for linear algebra (GMRES)
- Machine Learning
  - Coordinate Descent (LASSO)
  - Training Neural Nets – "ImageNet training in minutes"
- **And Beyond**
  - **Extending communication lower bounds and optimal algorithms to general loop nests**
- Summary

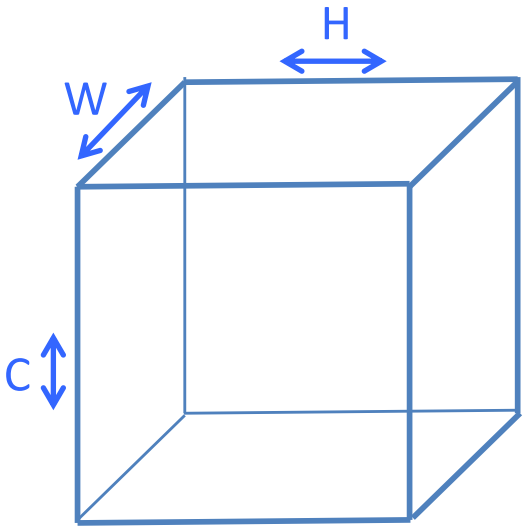
Communication lower bounds and optimal algorithms for general loop nests

- for i = 1:n, for j=1:n, for k = 1:n
  C(i,j) = C(i,j) + A(i,k)*B(k,j)
- #Words moved between main memory and cache of size M = $\Omega(n^3 / M^{1/2})$, attainable
- For $(i_1,i_2,\ldots i_k) \in S \subseteq \mathbb{Z}^k$, do something with
  - A1(i_1), A2(i_2, i_3+i_4), A3(i_1-i_2, i_2+3*i_3- 5*i_4, …), …
- Thm: #Words moved = $\Omega(|S| / M^{e_{HBL}})$
  - HBL = Holder / Brascamp / Lieb
  - Uses recent results by Christ, Tao, others
- Thm: There exists an optimal algorithm that attain this lower bound (D. Rusciano)
- Ex: Convolutional Neural Nets (D., Dinh)

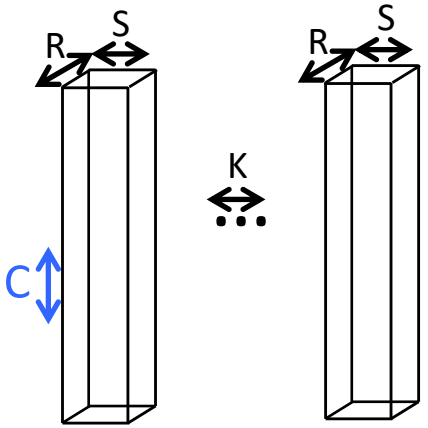
What CNNs compute



What CNNs compute

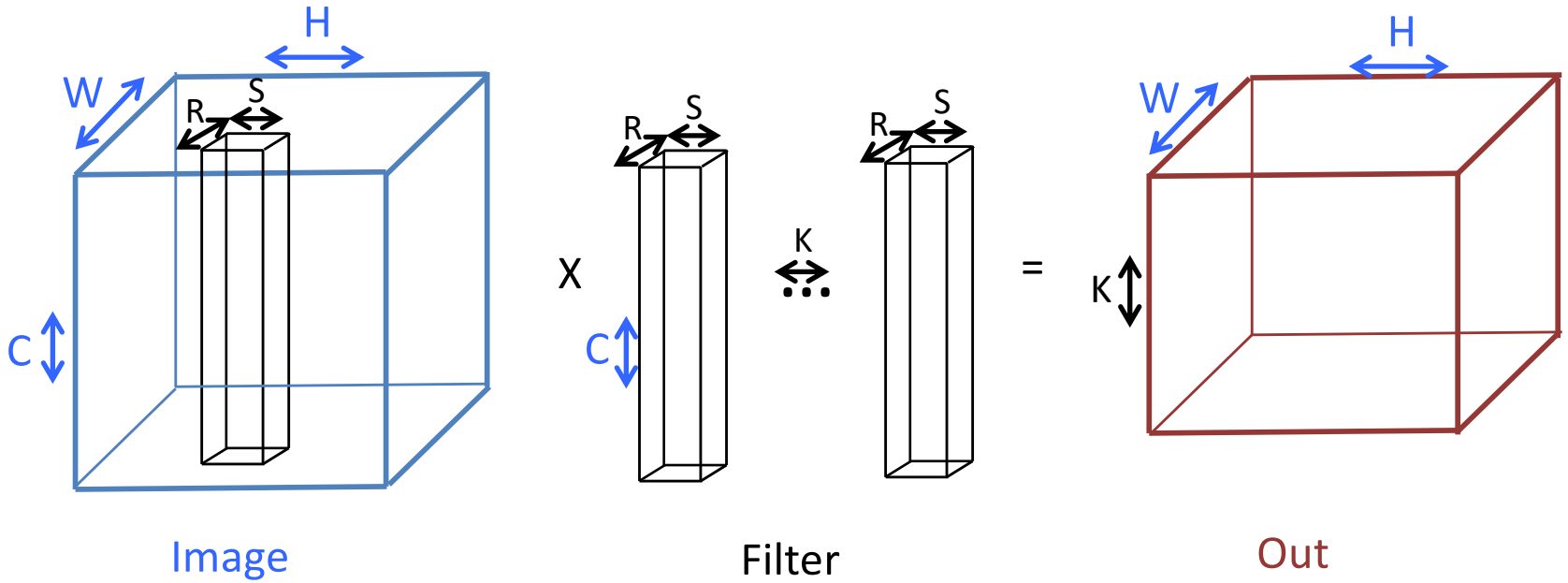


Image

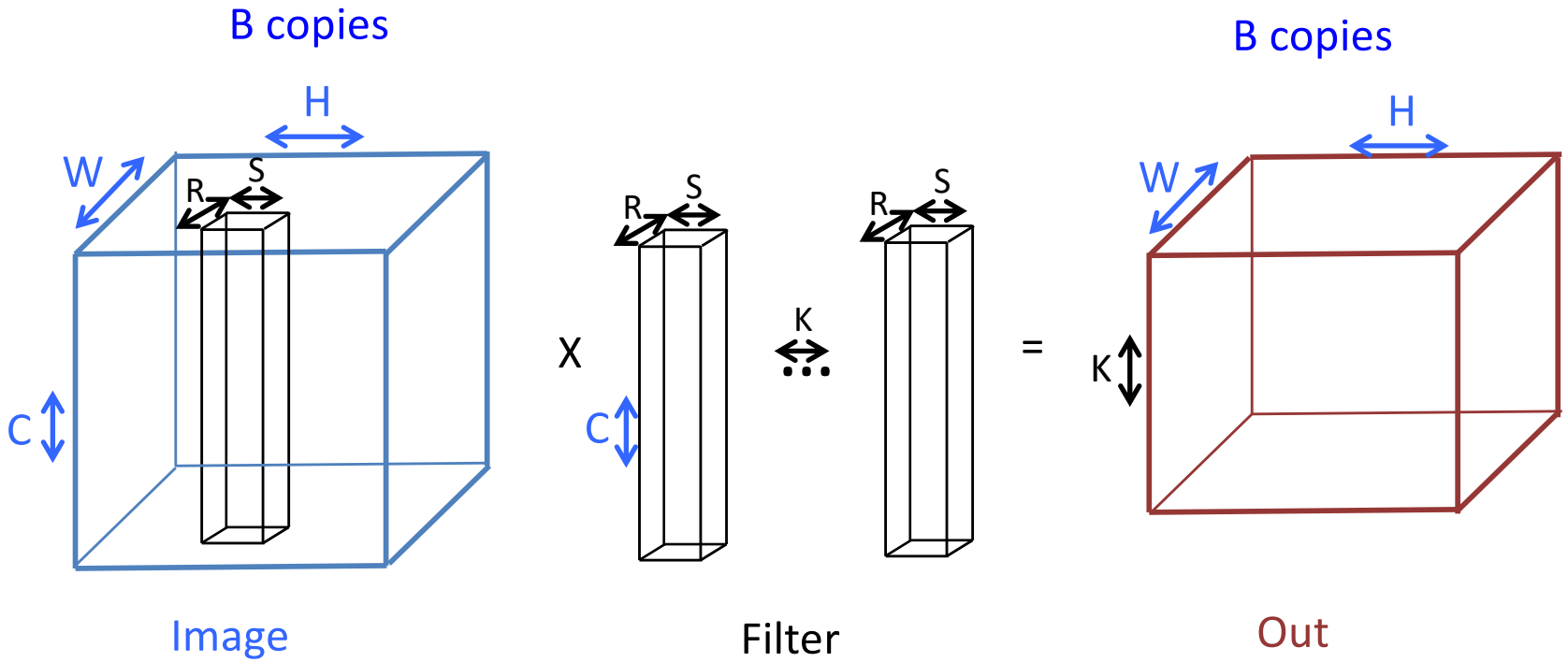


Filter

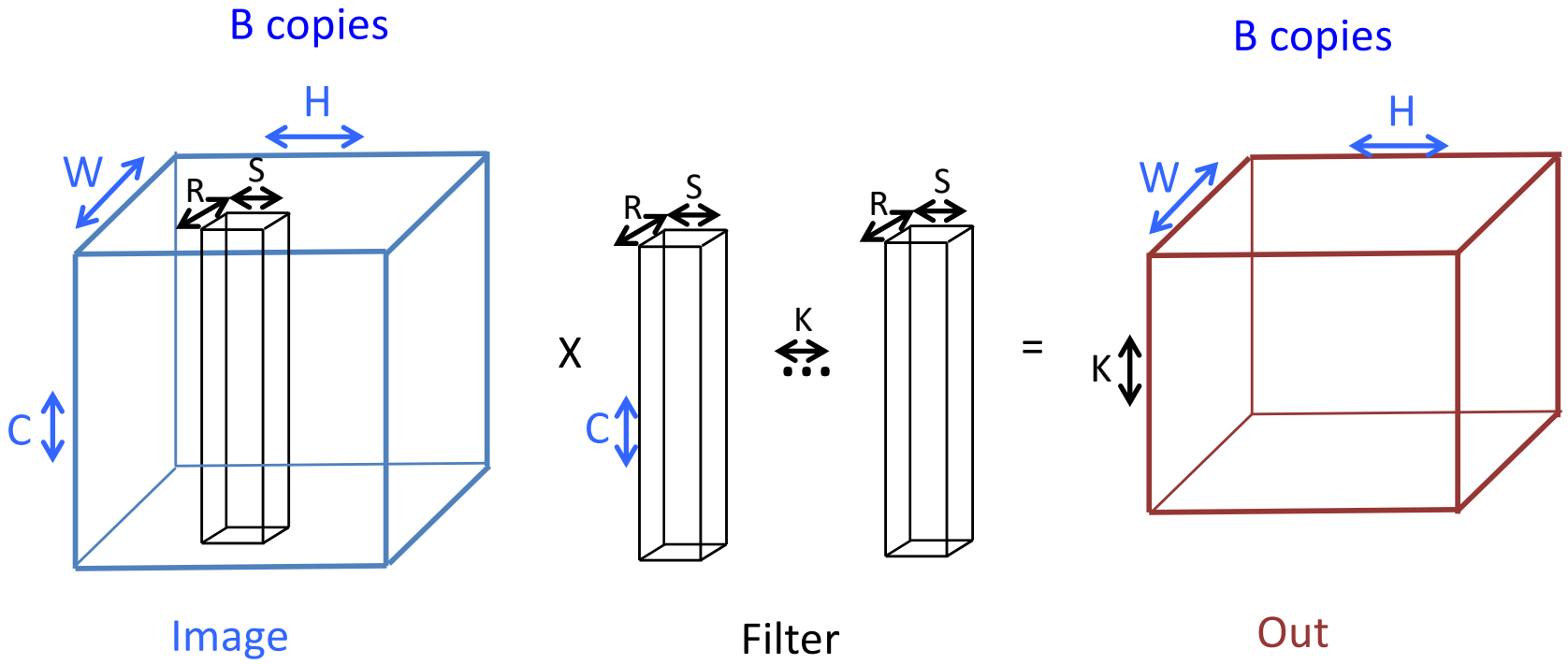
What CNNs compute



What CNNs compute



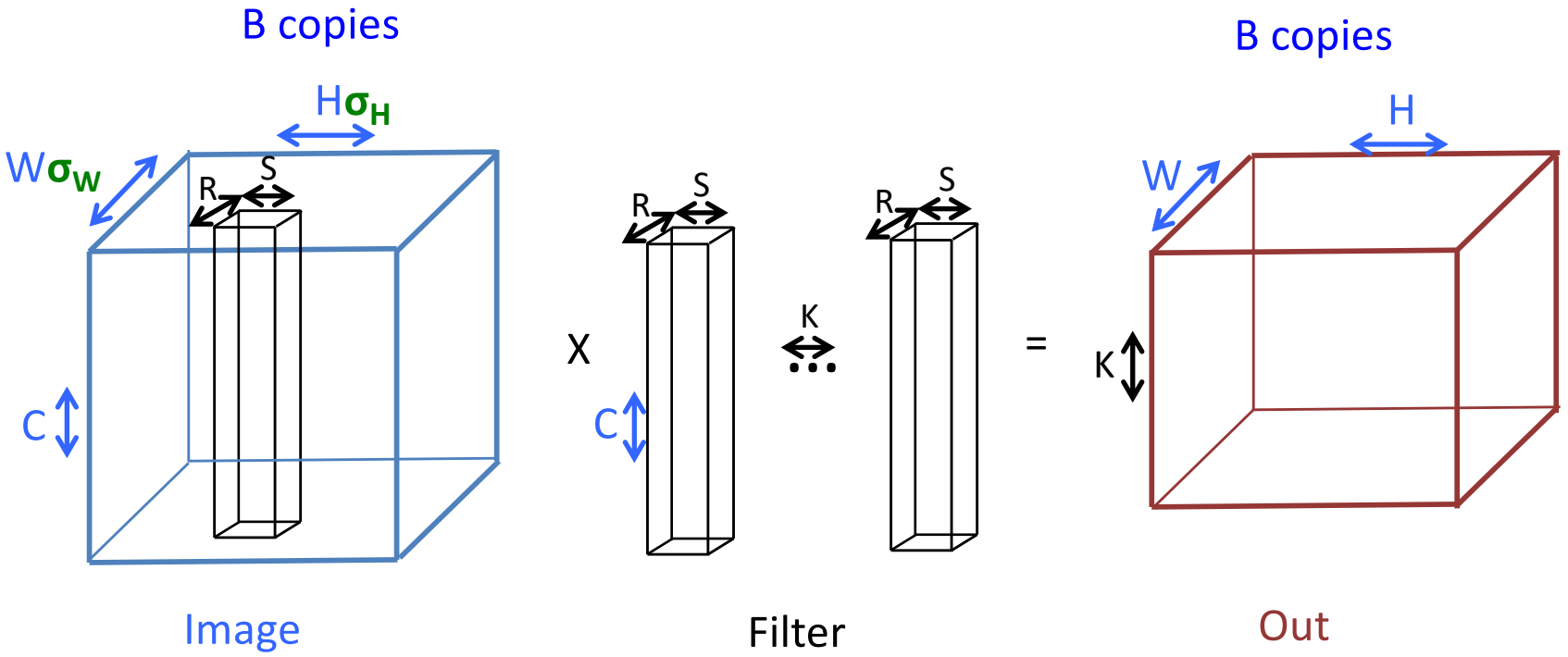
What CNNs compute



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) += Image(r+σ_Ww, s+σ_Hh, 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 for large loop bounds

  $N/(M^{1/2} (RS/(\sigma_H\sigma_W))^{1/2})$ ... lower bound for small filters)
- Any one of 5 terms may be largest
- Bottommost 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
- Thm: Always attainable! (computer generated proof)

Backup slides

Architectural Trends: Time

time per flop << time per word << time per message

	Petascale System* (2017)	Predicted Exascale System^	Amazon EC2 c5.18XL (est.)
Node Flops Time	$0.3\ ps$	$0.1\ -1\ ps$	$>1\ ps$
Node Memory Bandwidth	$132\ GB/s$	$0.4-4\ TB/s$	$<100\ GB/s$
Node Interconnect Bandwidth	$\mathbf{16\ GB/s}$	$\mathbf{100-400\ GB/s}$	$\mathbf{<3\ GB/s}$
Memory Latency	$\sim 100\ ns$	$50\ ns$	$>100\ ns$
Interconnect Latency	$\mathbf{1\ \mu s}$	$\mathbf{0.5\ \mu s}$	$\mathbf{>10\ \mu s}$

* Sunway TaihuLight Report (Dongarra 2016)

^ Source P. Beckman (ANL), J. Shalf (LBL), D. Unat (LBL)

Architectural Trends: Energy