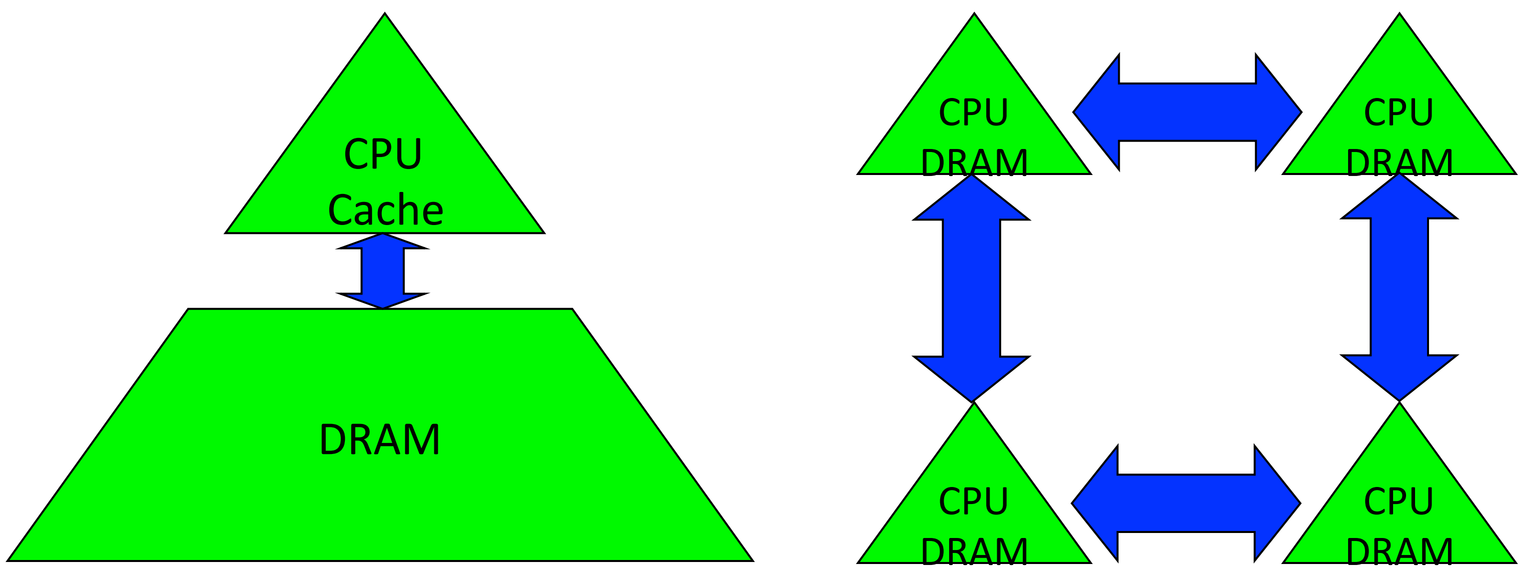
Communication-Avoiding Algorithms for Linear Algebra and Beyond

Jim Demmel
EECS & Math Departments
UC Berkeley

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

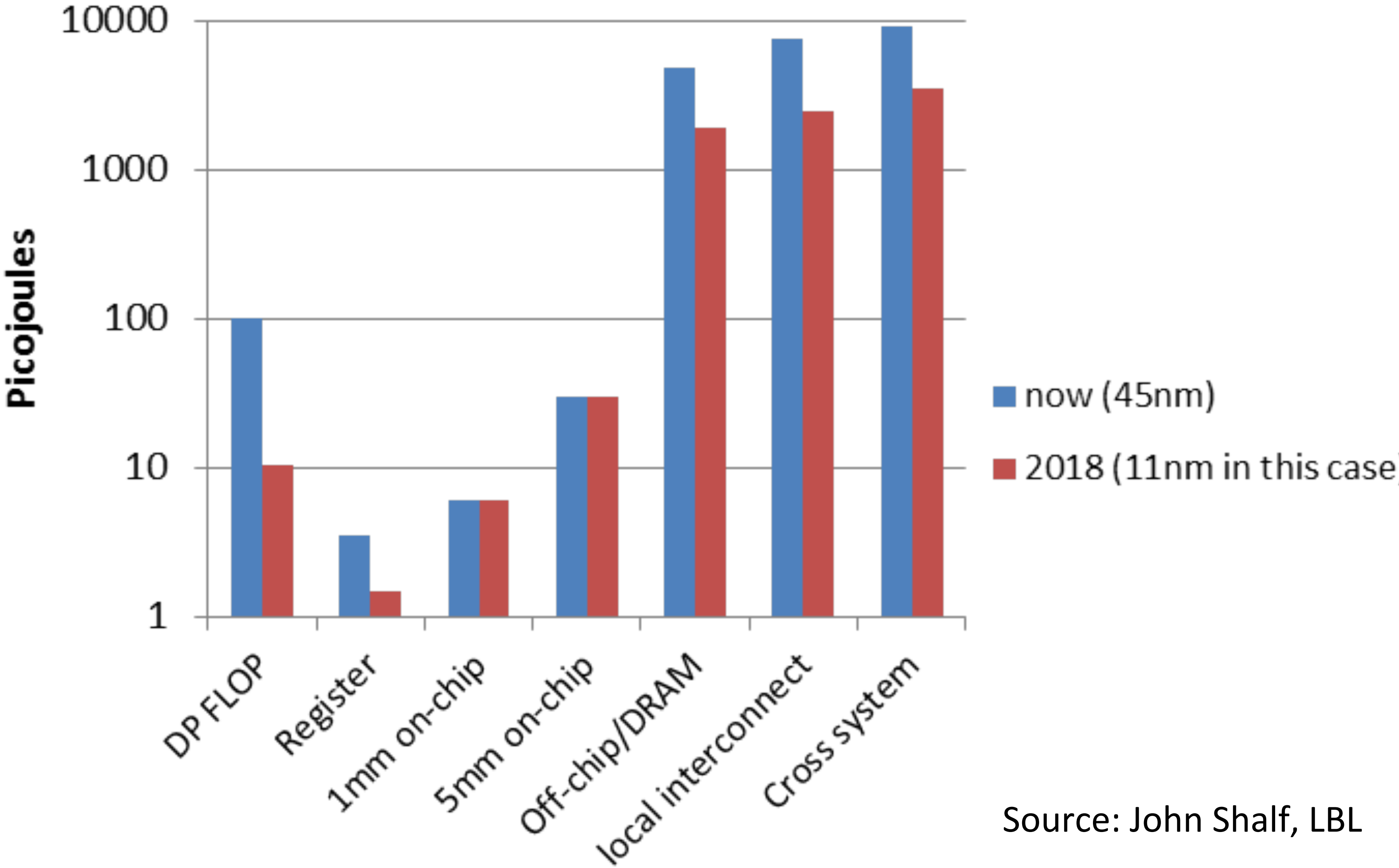
communication

- 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

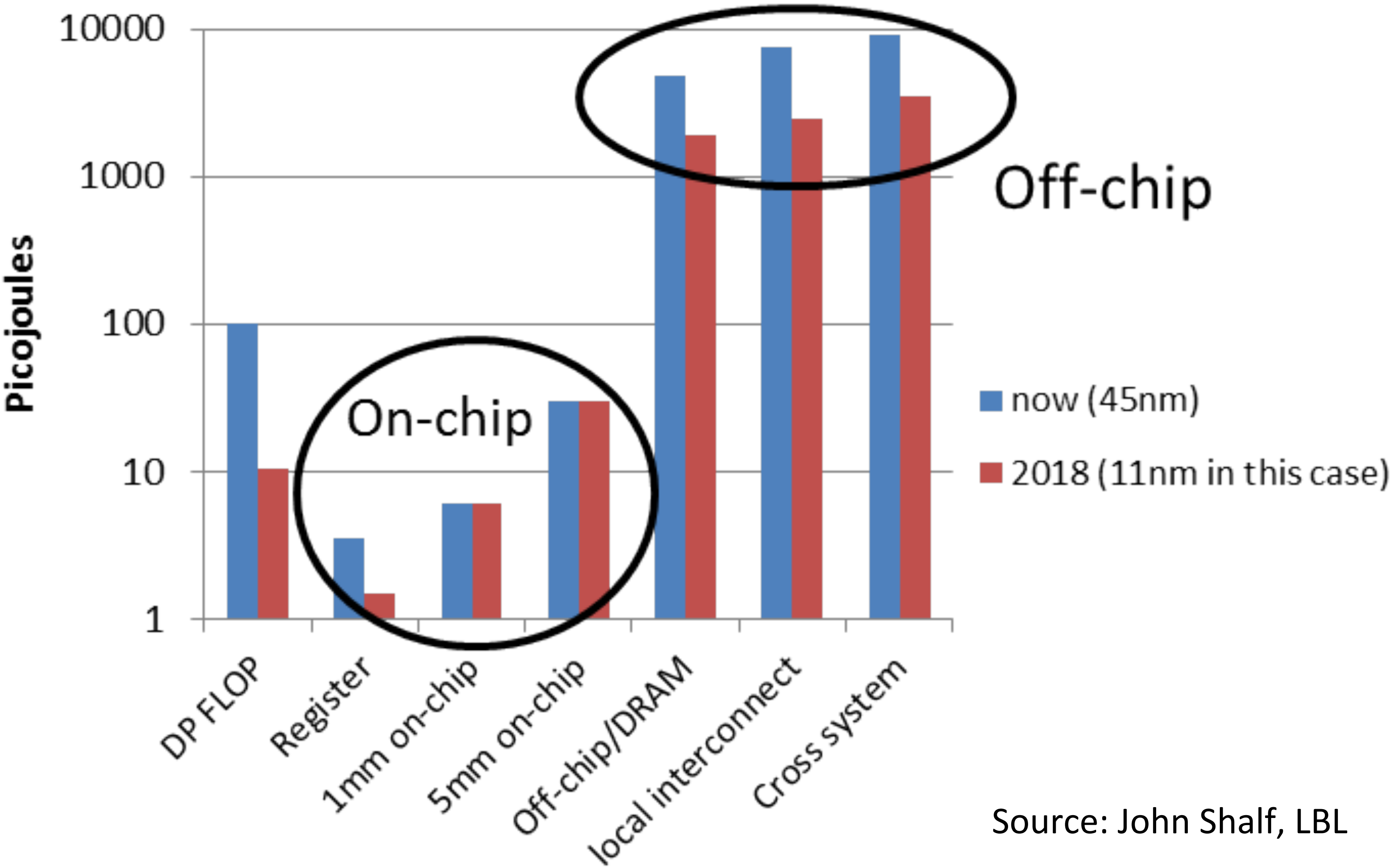
Why Minimize Communication? (3/3)



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 $\leftrightarrow$ L2 $\leftrightarrow$ DRAM $\leftrightarrow$ network, etc
- Attain lower bounds if possible
  - Current algorithms often far from lower bounds
  - Large speedups and energy savings possible
- New lower bounds, optimal algorithms, for
  - Direct linear algebra (matmul, LU, QR, SVD, etc.)
  - Iterative linear algebra (conjugate grad., etc.)
  - Other algorithms that access arrays (n-body, etc.)

President Obama cites Communication-Avoiding Algorithms in the FY 2012 Department of Energy Budget Request to Congress:

"New Algorithm Improves Performance and Accuracy on Extreme-Scale Computing Systems. **On modern computer architectures, communication between processors takes longer than the performance of a floating point arithmetic operation by a given processor.** ASCR researchers have developed a new method, derived from commonly used linear algebra methods, to **minimize communications between processors and the memory hierarchy, by reformulating the communication patterns specified within the algorithm**. This method has been implemented in the TRILINOS framework, a highly-regarded suite of software, which provides functionality for researchers around the world to solve large scale, complex multi-physics problems."

FY 2010 Congressional Budget, Volume 4, FY2010 Accomplishments, Advanced Scientific Computing Research (ASCR), pages 65-67.

CA-GMRES (Hoemmen, Mohiyuddin, Yelick, JD)
"Tall-Skinny" QR (Grigori, Hoemmen, Langou, JD)

Outline

- Survey state of the art of CA (Comm-Avoiding) algorithms
  - CA $O(n^3)$ 2.5D Matmul
  - *APSP: All-Pairs-Shortest-Pahts (Buluc, Monday)*
  - *TSQR: Tall-Skinny QR (Gleich, Tuesday)*
  - *CA Strassen Matmul (Ballard, Tuesday)*
- Beyond linear algebra
  - Extending lower bounds to any algorithm with arrays
  - Communication-optimal N-body algorithm, join (?)
- CA-Krylov methods

Outline

- Survey state of the art of CA (Comm-Avoiding) algorithms
  - CA $O(n^3)$ 2.5D Matmul
  - *APSP: All-Pairs-Shortest-Pahts (Buluc, Monday)*
  - *TSQR: Tall-Skinny QR (Gleich, Tuesday)*
  - *CA Strassen Matmul (Ballard, Tuesday)*
- Beyond linear algebra
  - Extending lower bounds to any algorithm with arrays
  - Communication-optimal N-body algorithm, join (?)
- CA-Krylov methods

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

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
  - New algorithms, with new numerical properties, new ways to encode answers, new data structures
  - Not just loop transformations (need those too!)
  - Being added to Sca/LAPACK, PLASMA, MAGMA, …
- Only a few sparse algorithms so far
- Lots of work in progress
- Ditto for iterative linear algebra

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?

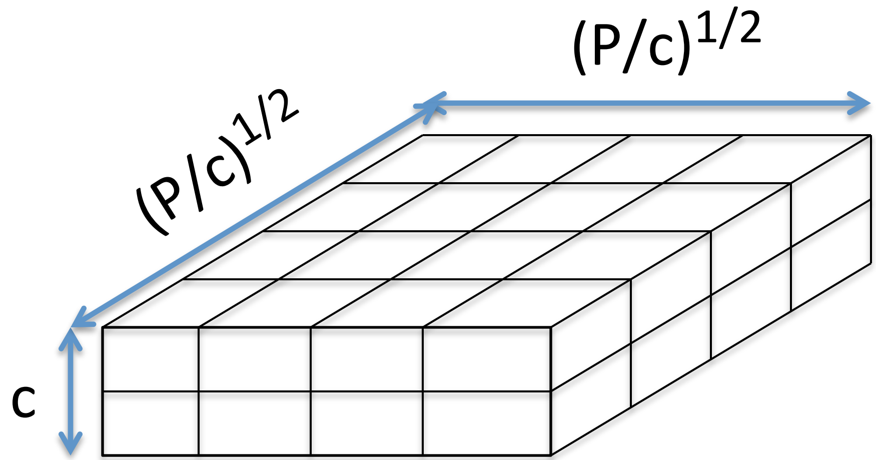
- Aren't we already optimal?
- Why assume $M = O(n^2/p)$, i.e. minimal?
  - Lower bound still true if more memory
  - Can we attain it?

Outline

- Survey state of the art of CA (Comm-Avoiding) algorithms
  - CA $O(n^3)$ 2.5D Matmul
  - *APSP: All-Pairs-Shortest-Pahts (Buluc, Monday)*
  - *TSQR: Tall-Skinny QR (Gleich, Tuesday)*
  - *CA Strassen Matmul (Ballard, Tuesday)*
- Beyond linear algebra
  - Extending lower bounds to any algorithm with arrays
  - Communication-optimal N-body algorithm, join (?)
- CA-Krylov methods

2.5D Matrix Multiplication

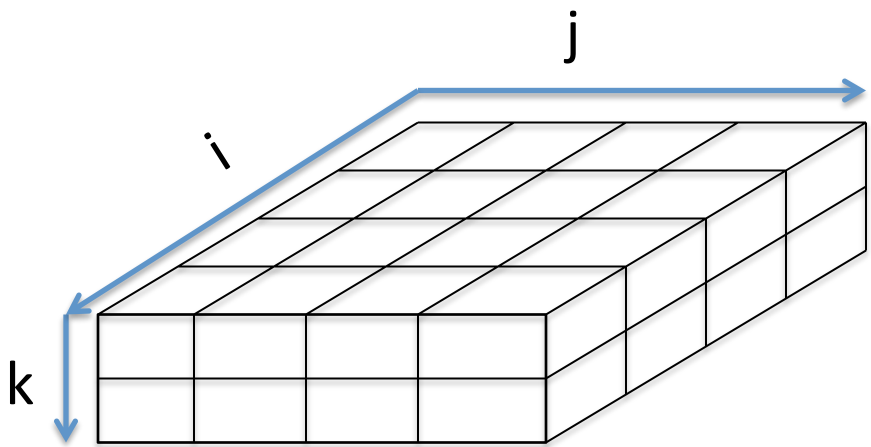
- Assume can fit cn^2/P data per processor, $c > 1$
- Processors form $(P/c)^{1/2}$ x $(P/c)^{1/2}$ x c grid



Example: P = 32, c = 2

2.5D Matrix Multiplication

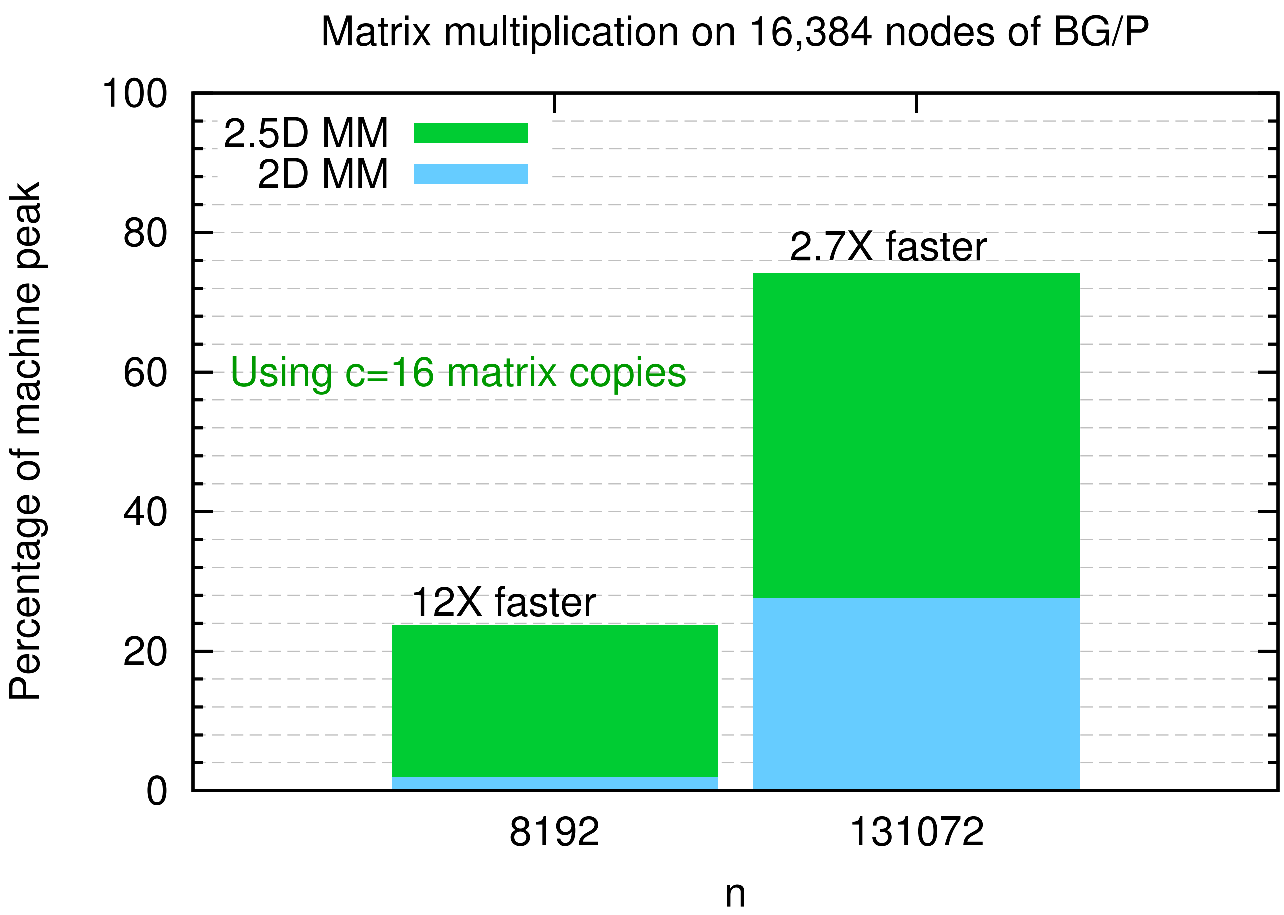
- Assume can fit cn^2/P data per processor, $c > 1$
- Processors form $(P/c)^{1/2}$ x $(P/c)^{1/2}$ x c grid



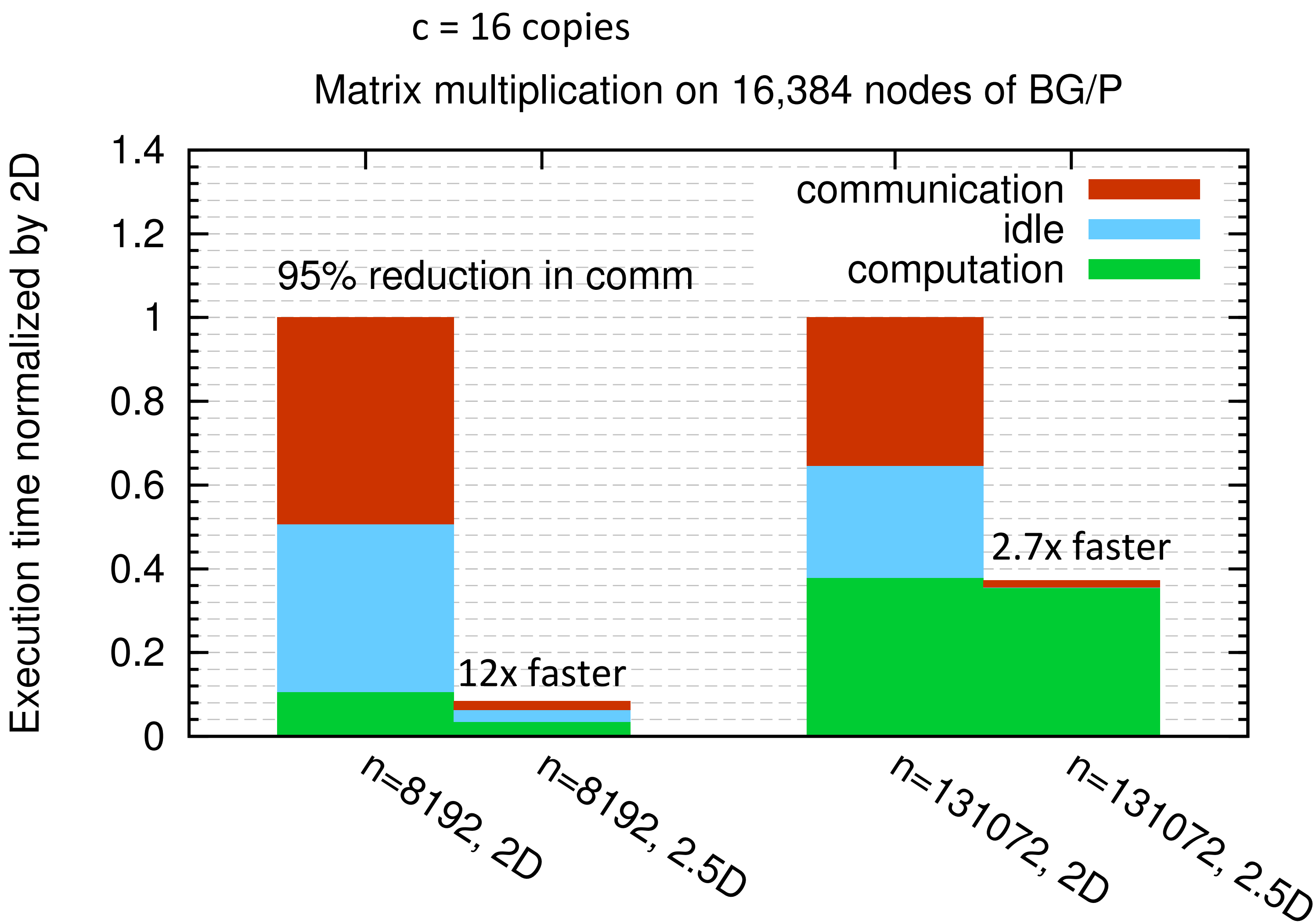
Initially P(i,j,0) owns A(i,j) and B(i,j) each of size $n(c/P)^{1/2} \times n(c/P)^{1/2}$

(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



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.
- $E(cP) = cP\ \{\ n^3/(cP)\ [\ \gamma_E + \beta_E/M^{1/2} + \alpha_E/(mM^{1/2})\ ] + \delta_E M T(cP) + \varepsilon_E T(cP)\ \}$

  $= E(P)$
- Perfect scaling extends to N-body, Strassen, …

Ongoing Work

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

Outline

- Survey state of the art of CA (Comm-Avoiding) algorithms
  - CA $O(n^3)$ 2.5D Matmul
  - *APSP: All-Pairs-Shortest-Pahts (Buluc, Monday)*
  - *TSQR: Tall-Skinny QR (Gleich, Tuesday)*
  - *CA Strassen Matmul (Ballard, Tuesday)*
- Beyond linear algebra
  - Extending lower bounds to any algorithm with arrays
  - Communication-optimal N-body algorithm, join (?)
- CA-Krylov methods

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:
  ... write A as n/b-by-n/b matrix of b-by-b blocks A[i,j]
  ... ditto for B, C
  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-by-b matrix multiply

- 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

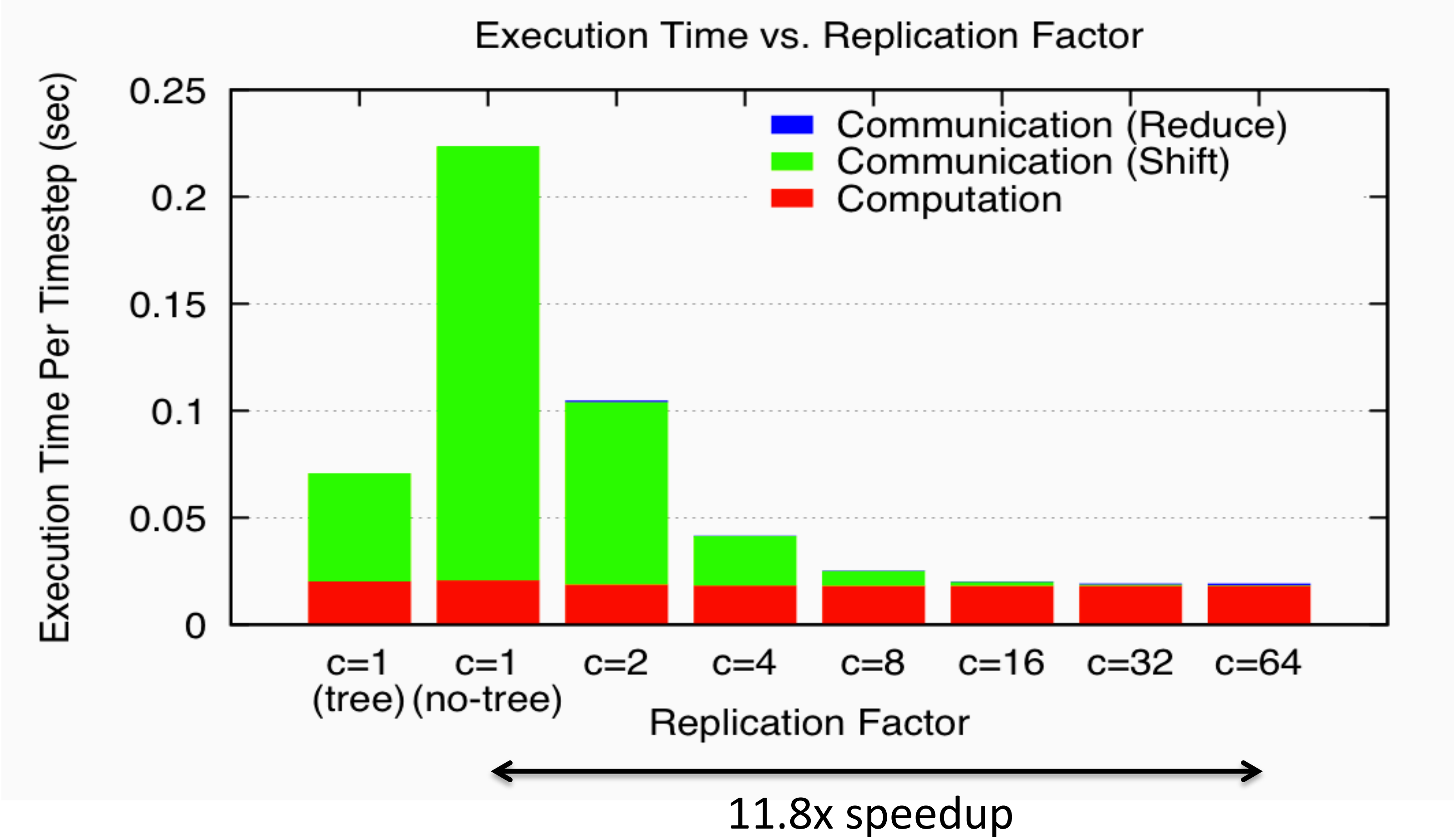
$$\Delta = \begin{array}{c} \\ \\ \\ \\ \end{array}\begin{array}{cc} i & j \\ \left(\begin{array}{c} 1 \\ 1 \\ 0 \end{array}\right. & \left.\begin{array}{c} 0 \\ 0 \\ 1 \end{array}\right) \end{array}\begin{array}{l} \\ F \\ P(i) \\ P(j) \end{array}$$

- 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



New Thm applied to Random Code

- for i1=1:n, for i2=1:n, … , for i6=1:n
  A1(i1,i3,i6) += func1(A2(i1,i2,i4),A3(i2,i3,i5),A4(i3,i4,i6))
  A5(i2,i6) += func2(A6(i1,i4,i5),A3(i3,i4,i6))
- Record array indices
  in matrix Δ

$$\Delta = \begin{array}{cccccccc} & i1 & i2 & i3 & i4 & i5 & i6 & \\ & \left(\begin{array}{c} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{array}\right. & \begin{array}{c} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{array} & \begin{array}{c} 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \end{array} & \begin{array}{c} 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \end{array} & \begin{array}{c} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{array} & \left.\begin{array}{c} 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{array}\right) & \begin{array}{c} A1 \\ A2 \\ A3 \\ A3,A4 \\ A5 \\ A6 \end{array} \end{array}$$

- Solve LP for $x = [x1,\ldots,x7]^T$: max $\mathbf{1}^T x$ s.t. $\Delta\, x \le \mathbf{1}$
  – Result: $x = [2/7,3/7,1/7,2/7,3/7,4/7]$, $\mathbf{1}^T x = 15/7 = s_{HBL}$
- Thm: #words_moved = $\Omega(n^6/M^{s_{HBL}-1}) = \Omega(n^6/M^{8/7})$
  Attained by block sizes $M^{2/7}, M^{3/7}, M^{1/7}, M^{2/7}, M^{3/7}, M^{4/7}$

Summary of Results Beyond LA (1/3)

- Extend communication lower bound proof from linear algebra to any program with
  - Inner loop iterations indexed by $(i_1,\ldots,i_d)$
  - Arrays in inner loop subscripted by linear functions of indices
  - Ex: $A(i_1,i_2-i_1,3i_1-4i_2+7i_4,\ldots)$, $B(ptr(i_5+6i_6))$, …
  - Can be dense or sparse, sequential or parallel, …
- Based on recent result of Bennett/Carbery/Christ/Tao
  - Generalizes **H**older, **B**rascamp-**L**ieb, Loomis-Whitney
  - Get linear program with one inequality per subgroup $H < Z^d$
  - Solution of linear program is s_{HBL}
  - Thm: #words_moved = $\Omega(\#loop\_iterations/M^{s_{HBL}-1})$

Summary of Results Beyond LA (2/3)

- Can we write down the lower bound?
  - One inequality per subgroup $H < Z^d$, but still finitely many
  - Thm (bad news): Writing down all the inequalities equivalent to solving Hilbert's 10^{th} Problem over Q
  - Thm (good news): Another LP has same solution, is decidable
  - Thm (better news): Easy to write down the LP explicitly in many cases of interest (eg when subscripts are just subsets of indices)
  - Also easy to get upper/lower bounds on s_{HBL}

Summary of Results Beyond LA (3/3)

- Can we attain the lower bound?
  - Depends on loop dependencies
  - Best case: none, or reductions (like matmul)
  - Thm: When subscripts are just subsets of indices, the solution x of the dual LP tells us the optimal tile sizes M^{x1}, M^{x2},…,M^{xd}
  - Ex: linear algebra, n-body, join, "random code", …
  - Conjecture: always attainable (modulo dependencies)

Outline

- Survey state of the art of CA (Comm-Avoiding) algorithms
  - CA $O(n^3)$ 2.5D Matmul
  - *APSP: All-Pairs-Shortest-Pahts (Buluc, Monday)*
  - *TSQR: Tall-Skinny QR (Gleich, Tuesday)*
  - *CA Strassen Matmul (Ballard, Tuesday)*
- Beyond linear algebra
  - Extending lower bounds to any algorithm with arrays
  - Communication-optimal N-body algorithm, join (?)
- CA-Krylov methods

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

For more details

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

Collaborators and Supporters

- **James Demmel**, **Kathy Yelick**, Michael Anderson, Grey Ballard, Erin Carson, Aditya Devarakonda, Michael Driscoll, David Eliahu, Andrew Gearhart, Evangelos Georganas, Nicholas Knight, Penporn Koanantakool, Ben Lipshitz, Oded Schwartz, Edgar Solomonik, Omer Spillinger
- Austin Benson, Maryam Dehnavi, Mark Hoemmen, Shoaib Kamil, Marghoob Mohiyuddin
- Abhinav Bhatele, Aydin Buluc, Michael Christ, Ioana Dumitriu, Armando Fox, David Gleich, Ming Gu, Jeff Hammond, Mike Heroux, Olga Holtz, Kurt Keutzer, Julien Langou, Devin Matthews, Tom Scanlon, Michelle Strout, Sam Williams, Hua Xiang
- Jack Dongarra, Dulceneia Becker, Ichitaro Yamazaki
- Sivan Toledo, Alex Druinsky, Inon Peled
- Laura Grigori**,** Sebastien Cayrols, Simplice Donfack, Mathias Jacquelin, Amal Khabou, Sophie Moufawad, Mikolaj Szydlarski
- Members of ParLab, ASPIRE, BEBOP, 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

Summary

Time to redesign all linear algebra, n-body, …
algorithms and software
(and compilers)

Don't Communic…