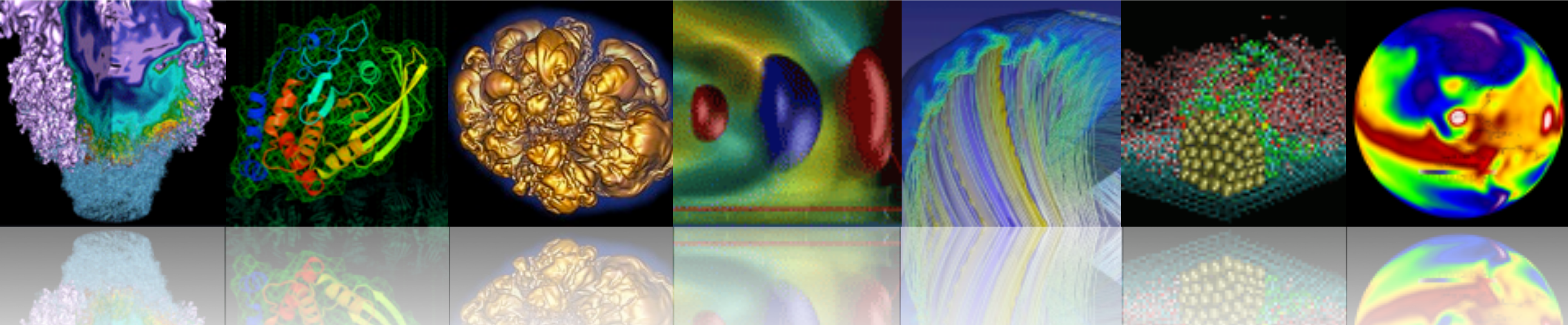


Antisocial Parallelism:
Avoiding, Hiding and Managing Communication

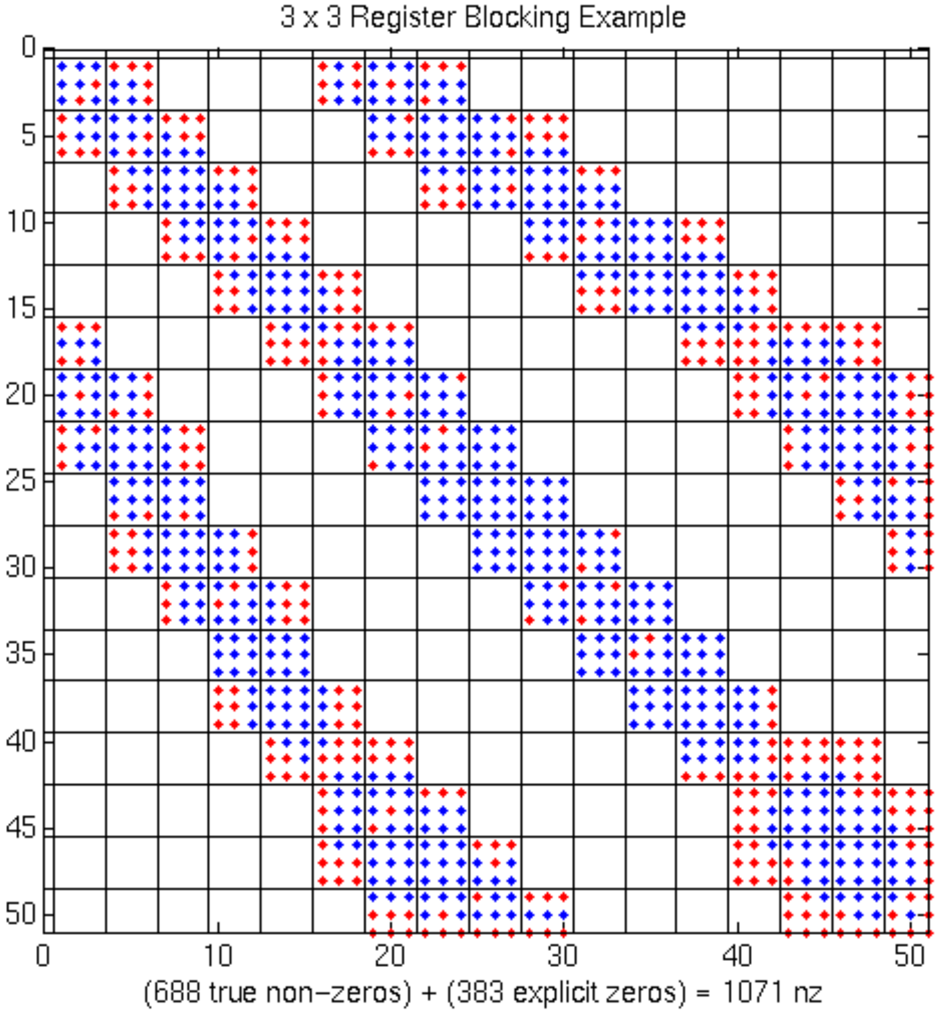
Kathy Yelick

Associate Laboratory Director of Computing Sciences

Lawrence Berkeley National Laboratory

EECS Professor, UC Berkeley

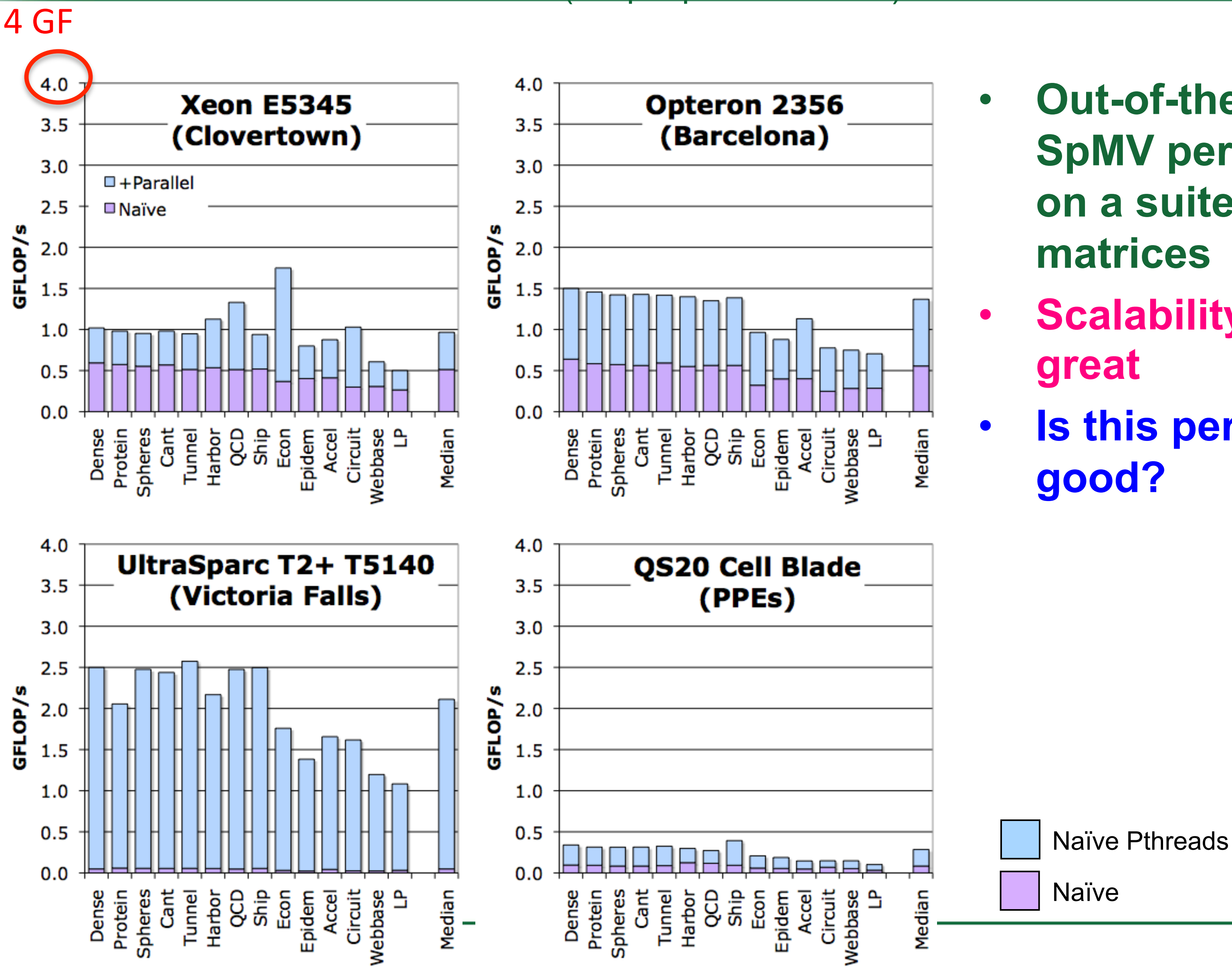
Extra Work Can Improve Efficiency!



- **Optimizing Sparse Matrix Vectory Multiply (fill)**
- **Example: 3x3 blocking**
  - Logical grid of 3x3 cells
  - Fill-in explicit zeros
  - Unroll 3x3 block multiplies
  - "Fill ratio" = 1.5
  - Takes advantage of registers
- **On Pentium III: 1.5x speedup!**
  - Actual mflop rate $1.5^2 = 2.25$ higher

See Eun-Jin Im PhD Thesis (Sparsity Library) and Rich Vuduc PhD thesis (OSKI Library)

SpMV Performance
(simple parallelization)

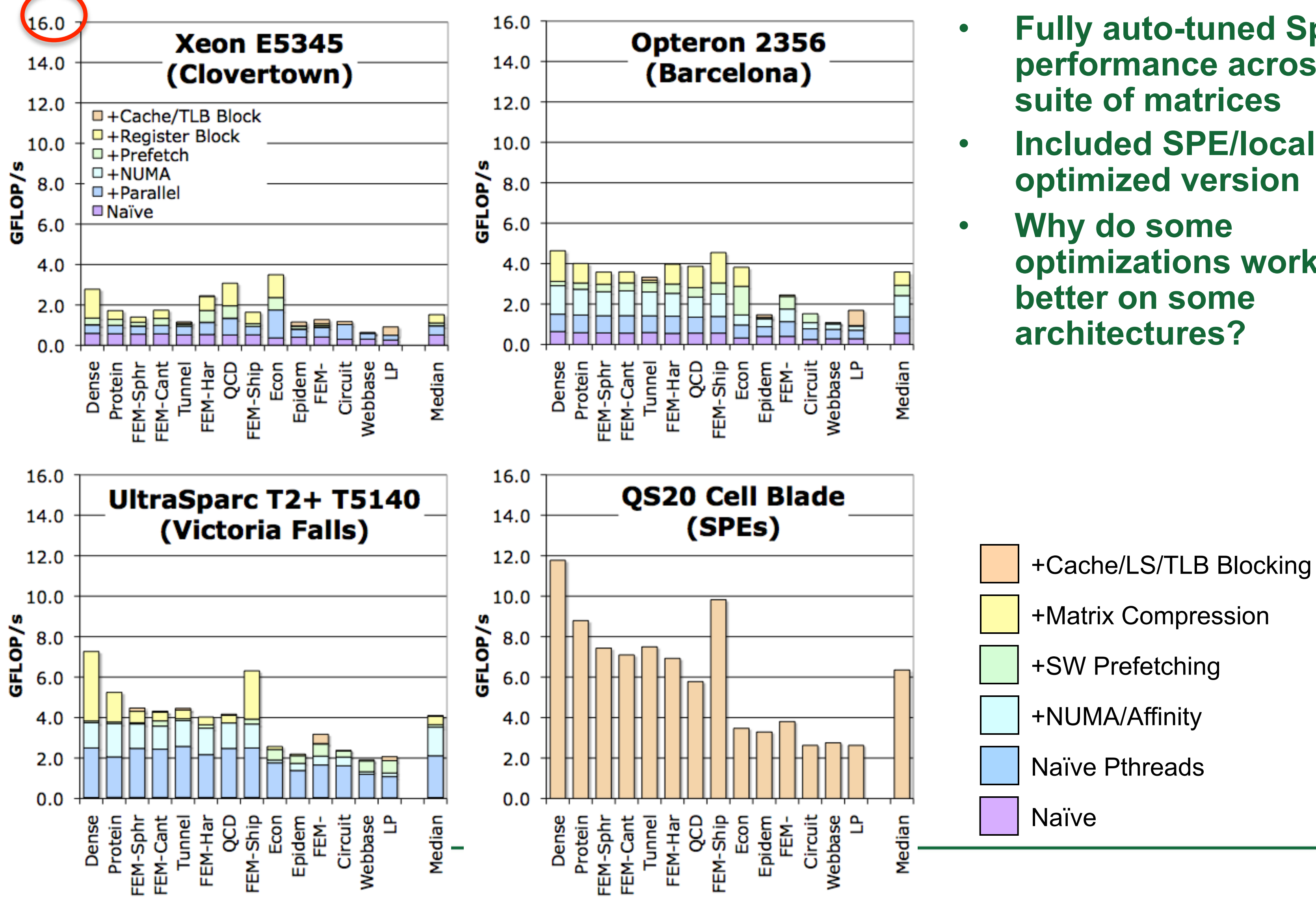


- Out-of-the box SpMV performance on a suite of 14 matrices
- Scalability isn't great
- Is this performance good?

See Sam Williams PhD thesis + papers

Auto-tuned SpMV Performance
(architecture specific optimizations)

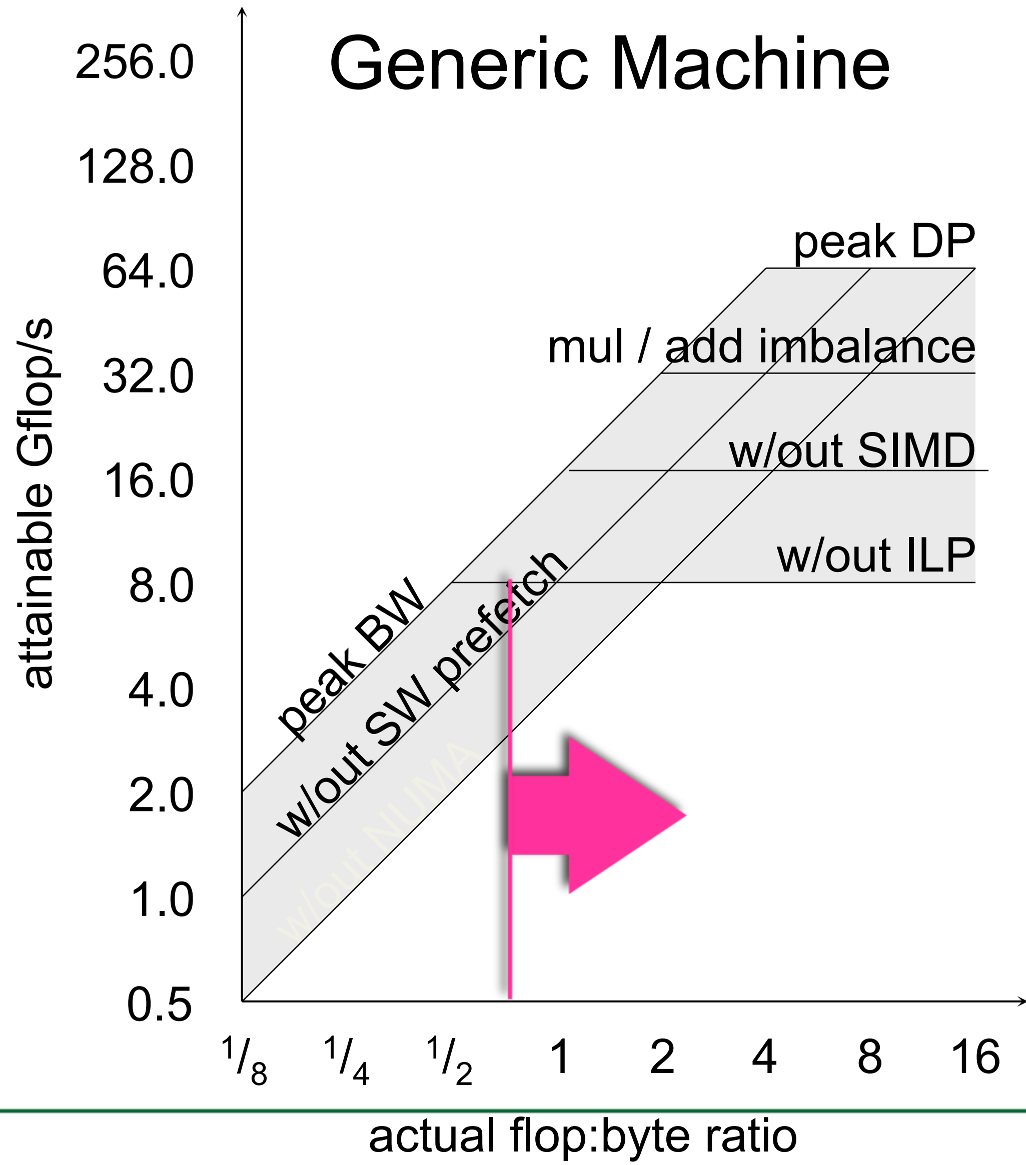
16 GF



- **Fully auto-tuned SpMV performance across the suite of matrices**
- **Included SPE/local store optimized version**
- **Why do some optimizations work better on some architectures?**

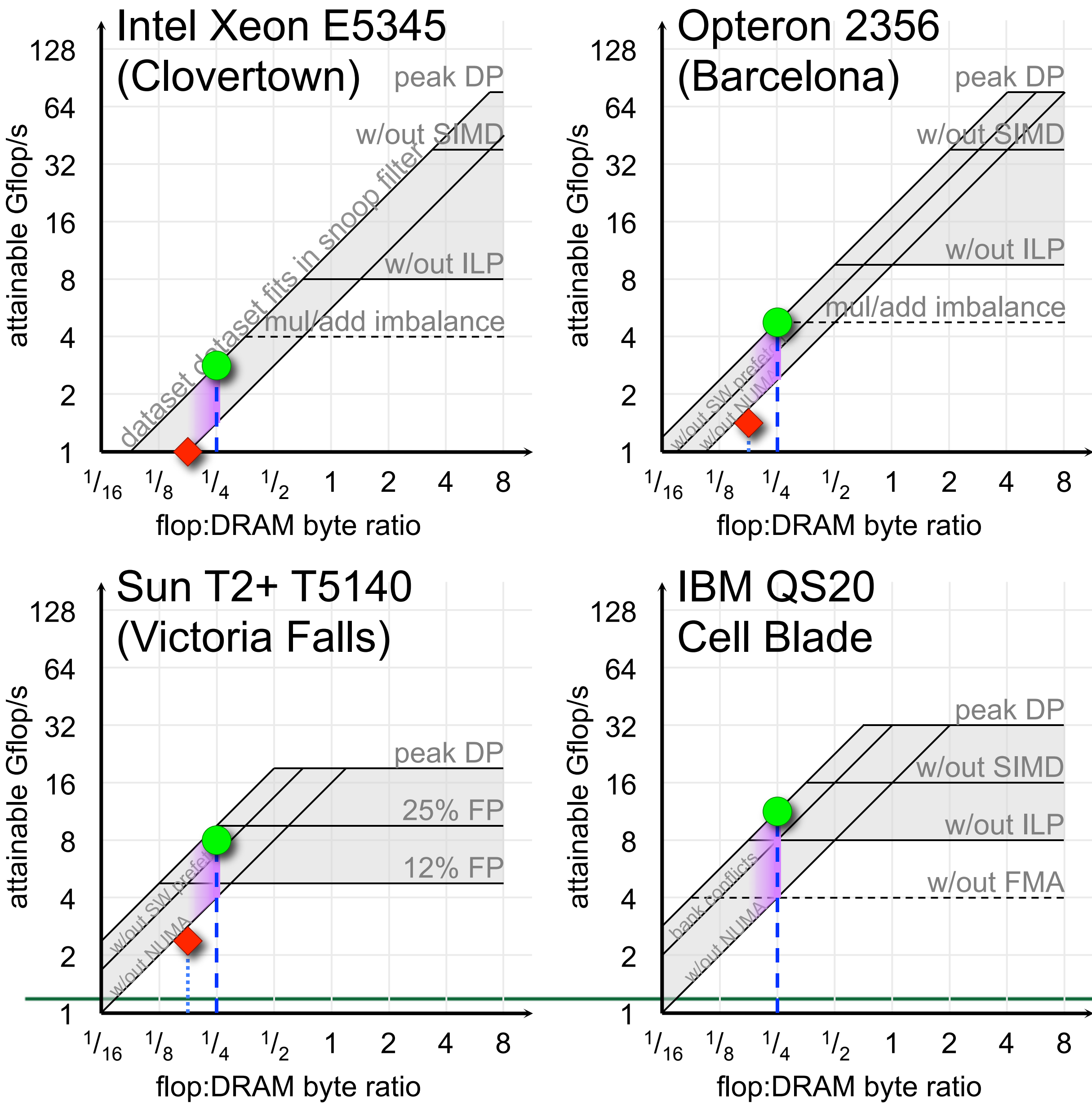
The Roofline Performance Model

See Sam Williams PhD Thesis



- Roof structure determined by machine
- Locations of posts in the building are determined by algorithmic intensity
- Will vary across algorithms and with bandwidth-reducing optimizations, such as better cache re-use (tiling), compression techniques
- Can use DRAM, network, disk,…

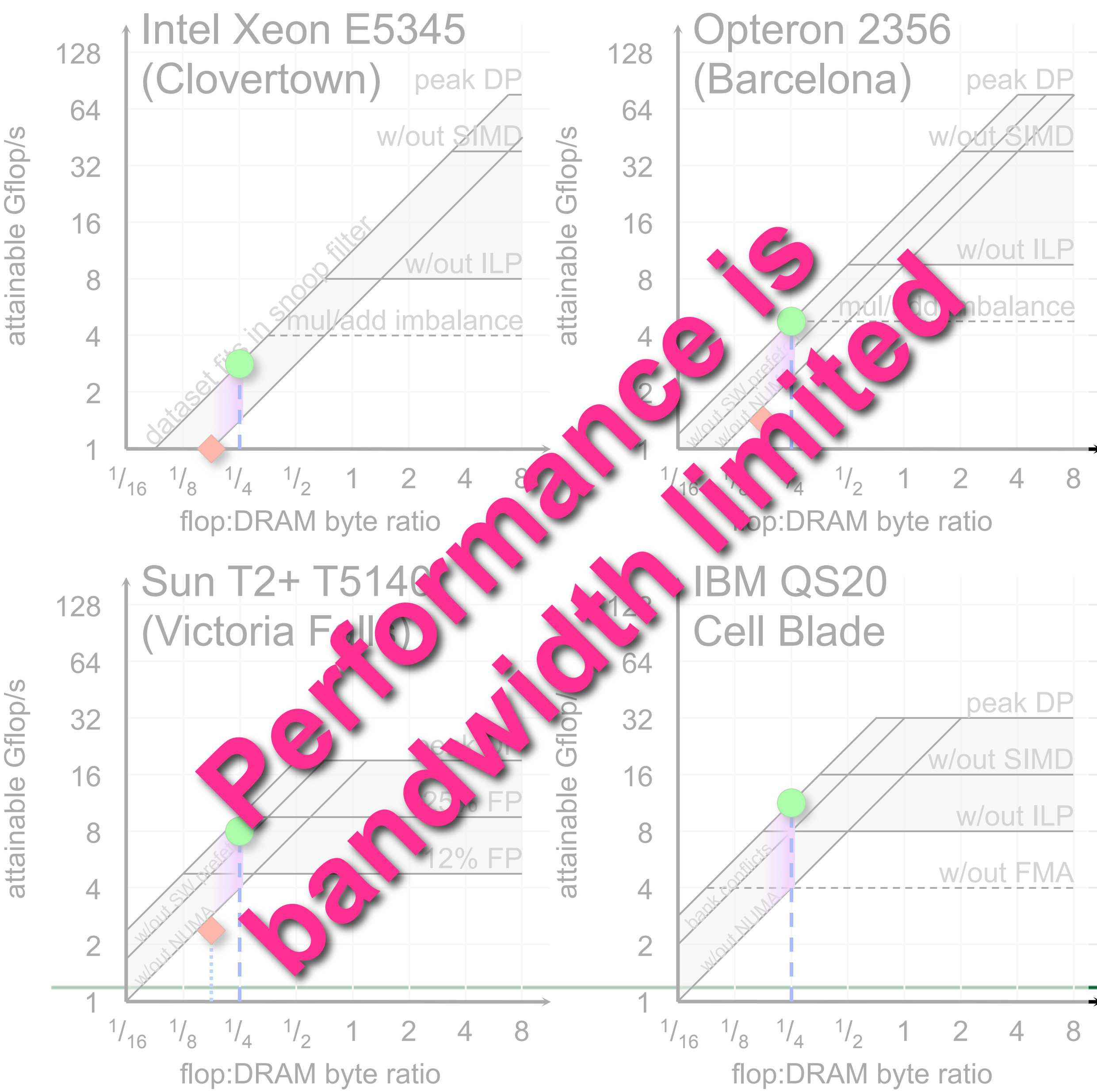
Roofline model for SpMV
(matrix compression)



- Inherent FMA
- Register blocking improves ILP, DLP, flop:byte ratio, and FP% of instructions

Roofline model for SpMV

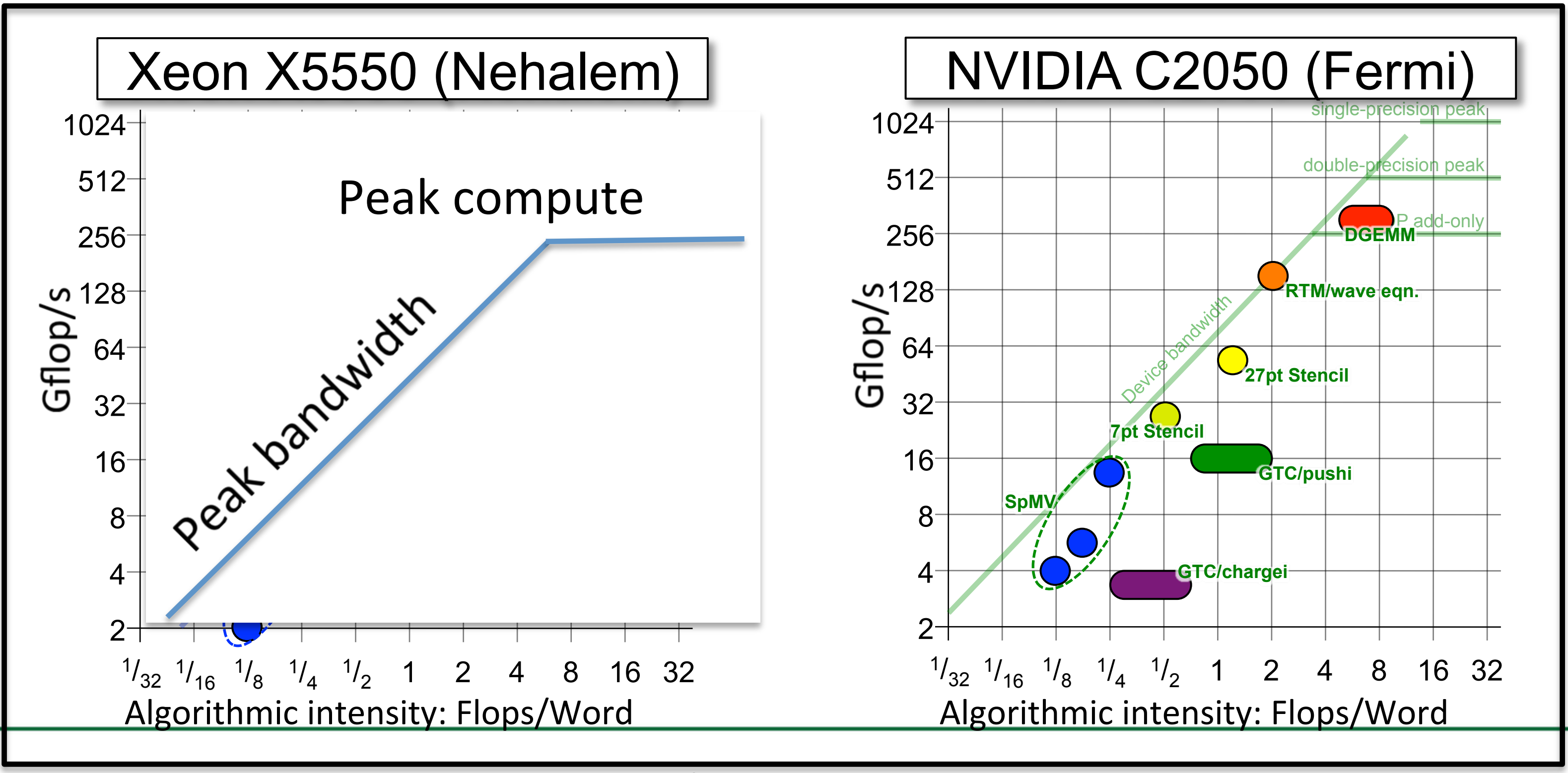
(matrix compression)



- SpMV *should* run close to memory bandwidth
  - Time to read matrix is major cost
- Can we do better?
- Can we compute A^k*x with one read of A?
- If so, this would
  - Reduce # messages
  - Reduce memory bandwidth

Autotuning: Write Code Generators

- Autotuners are code generators plus search
- Avoids two unsolved compiler problems: dependence analysis and accurate performance models
- Popular in libraries: Atlas, FFTW, OSKI,…

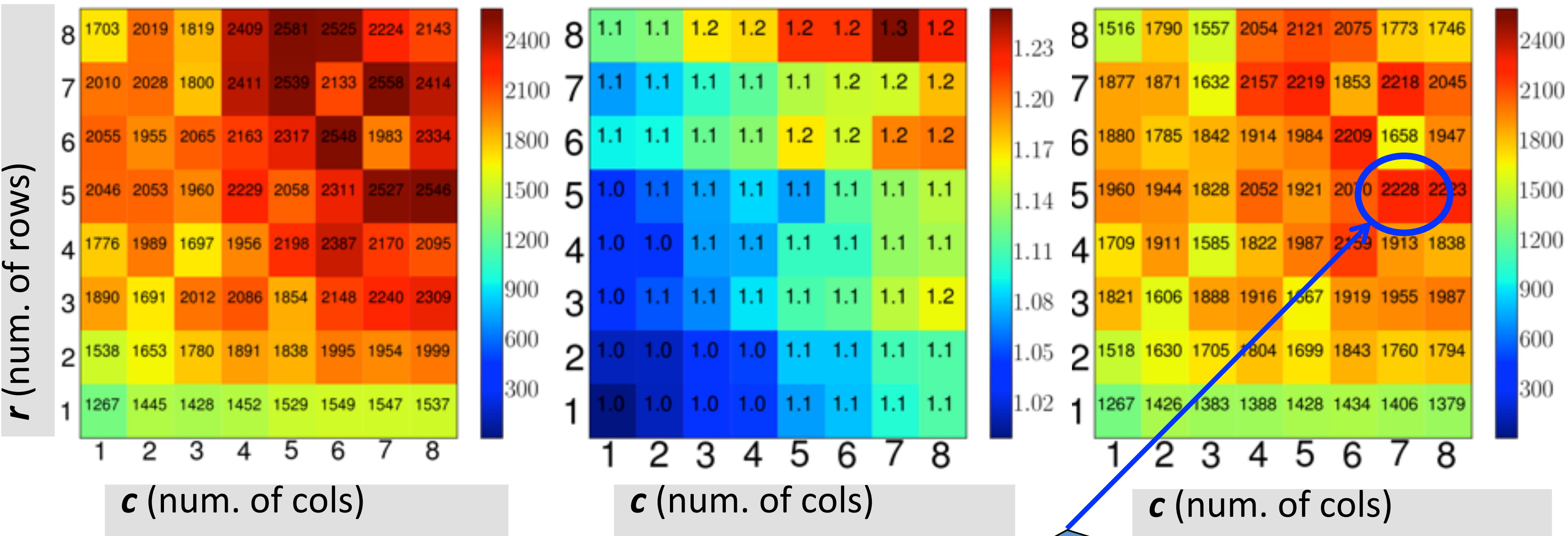


Work by Williams, Oliker, Shalf, Madduri, Kamil, Im, Ethier,…

Finding Good Performance is like finding the Needle in a Haystack

OSKI sparse matrix library: offline search + online evaluation: adding zeros can reduce storage in blocked **format**

Dense: ***MFlops(r,c)*** / Tsopf: ***Fill(r,c)*** = ***Effective_MFlops(r,c)***

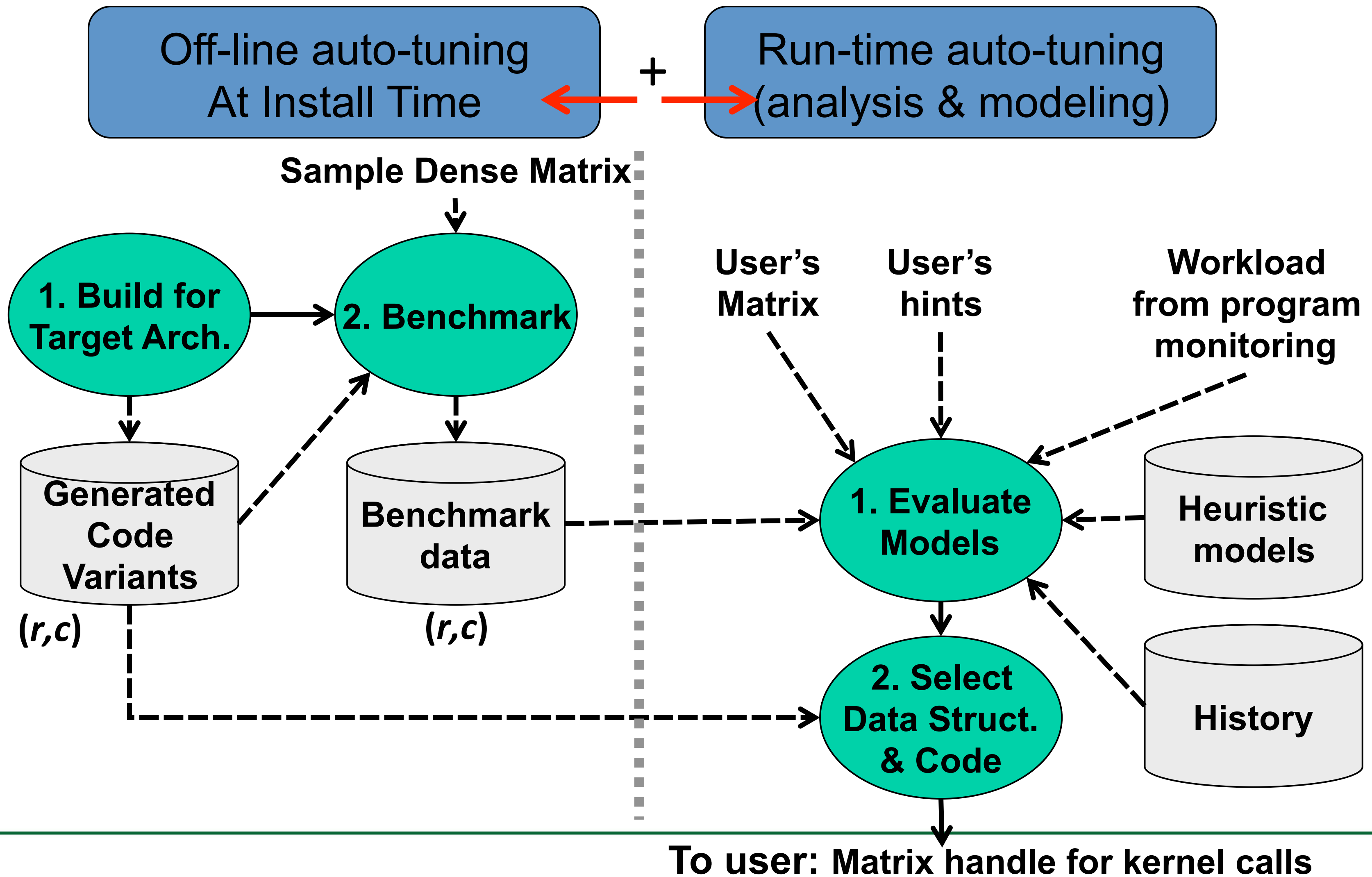


Selected **RB(5x7)** with a sample dense matrix

Work by Im, Vuduc, Williams, Kamil, Ho, Demmel, Yelick…

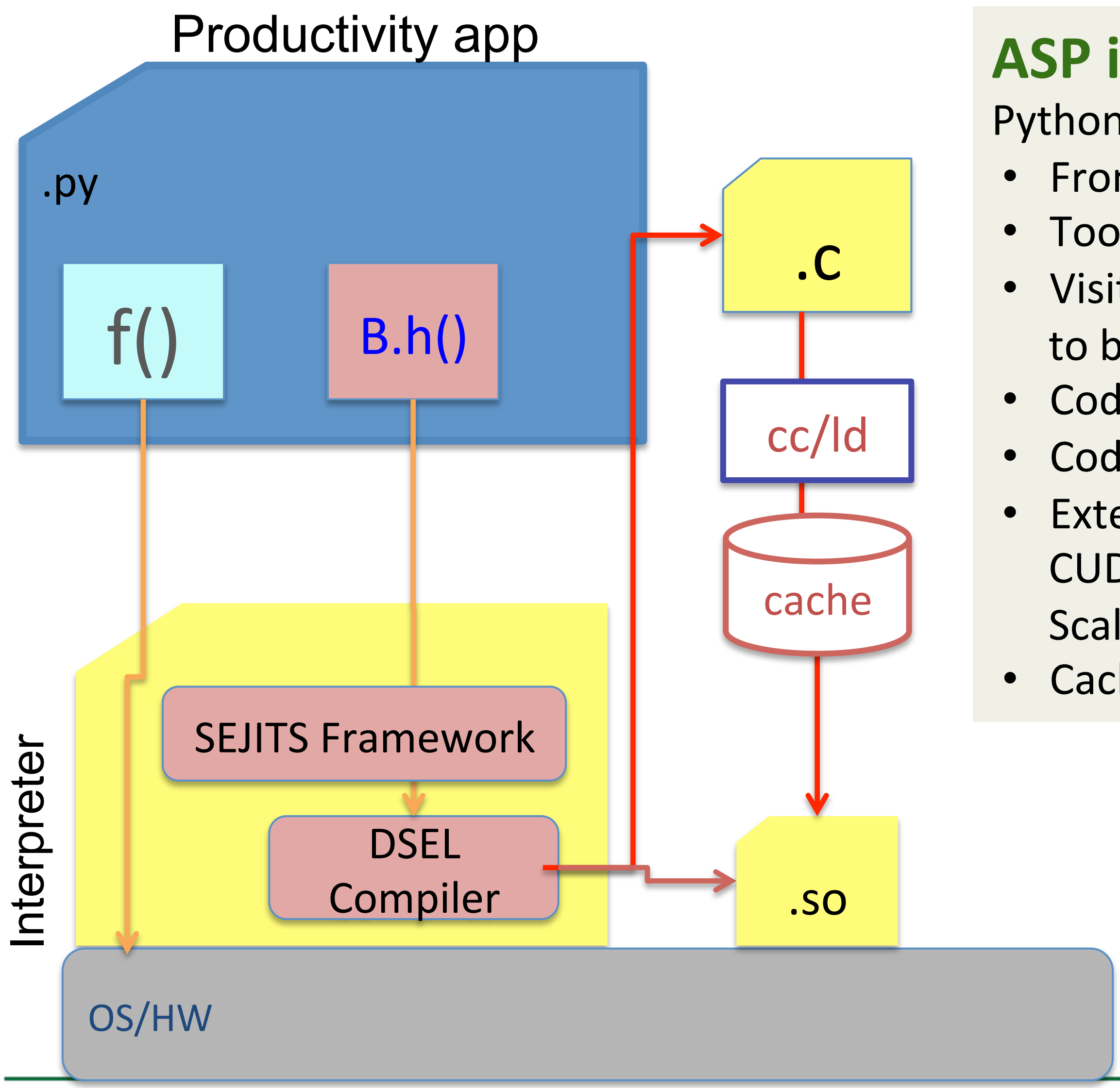
OSKI and pOSKI: Auto-tuning Sparse Matrix Kernels

- Our approach uses Empirical modeling and search



SEJITS: Selective Embedded Just-in-Time Specialization

(beyond Perl code generators)



ASP is SEJITS for Python:

Python provides

- Front-end parsing
- Tools for building strongly-typed IR
- Visitor pattern for IR transformation to backend AST
- Code "templates" (like webapps)
- Code generation
- External compiler support (C++, CUDA, OpenMP, pthreads, MPI, Scala)
- Caching

Armando Fox's group, including Shoaib Kamil and Michael Driscoll

Lessons Learned

- **Optimizations (not all in OSKI)**
  - Register blocking, loop unrolling, cache blocking, thread blocking, reordering, index compression, SIMDization, manual prefetch, NUMA (“PGAS” on node), matrix splitting, switch-to-dense, sparse/bit-masked register blocks
  - See http://bebop.berkeley.edu for papers
  - Straight line code failed to work on Spice ~10 years ago
    - 64-bit instructions: 1 load (x), 1 store (y), 1 op
    - Vs 1 op and fraction of load/store depending on reuse
- **Good news**
  - Autotuning helps save programmer time
- **But the operation is bandwidth limited**
  - With hardware optimizations (NUMA, prefetch, SIMDization, threading)
  - The rest is about matrix compression
- **A problem for local memory and network**

Avoiding Communication in Iterative Solvers

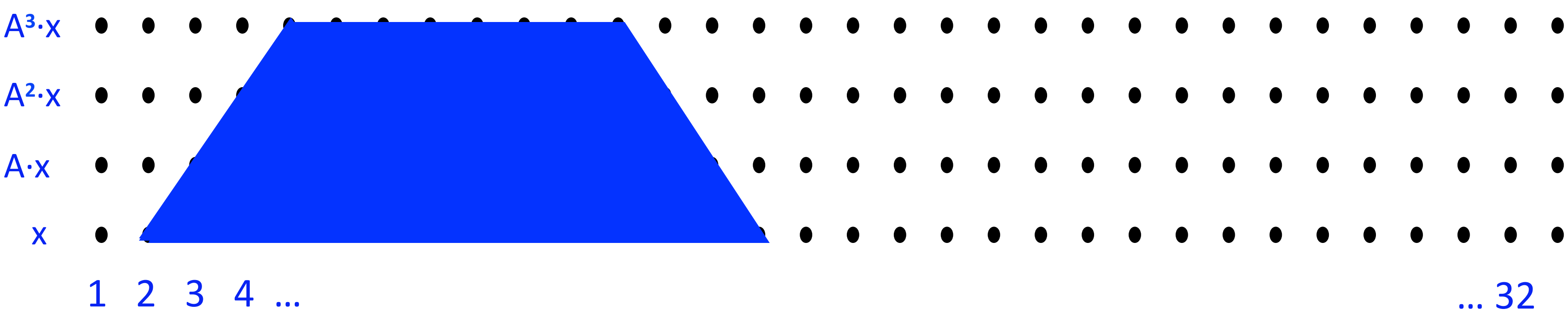
- **Consider Sparse Iterative Methods for *Ax*=*b***
  - Krylov Subspace Methods: GMRES, CG,…
- **Solve time dominated by:**
  - Sparse matrix-vector multiple (SPMV)
    - Which even on one processor is dominated by "communication" time to read the matrix
  - Global collectives (reductions)
    - Global latency-limited
- **Can we lower the communication costs?**
  - Latency: reduce # messages by computing multiple reductions at once
  - Bandwidth to memory, i.e., compute Ax, A^2x, … A^kx with one read of A

Joint work with Jim Demmel, Mark Hoemmen, Marghoob Mohiyuddin; See 2 PhD thesis for details

Communication Avoiding Kernels

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

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

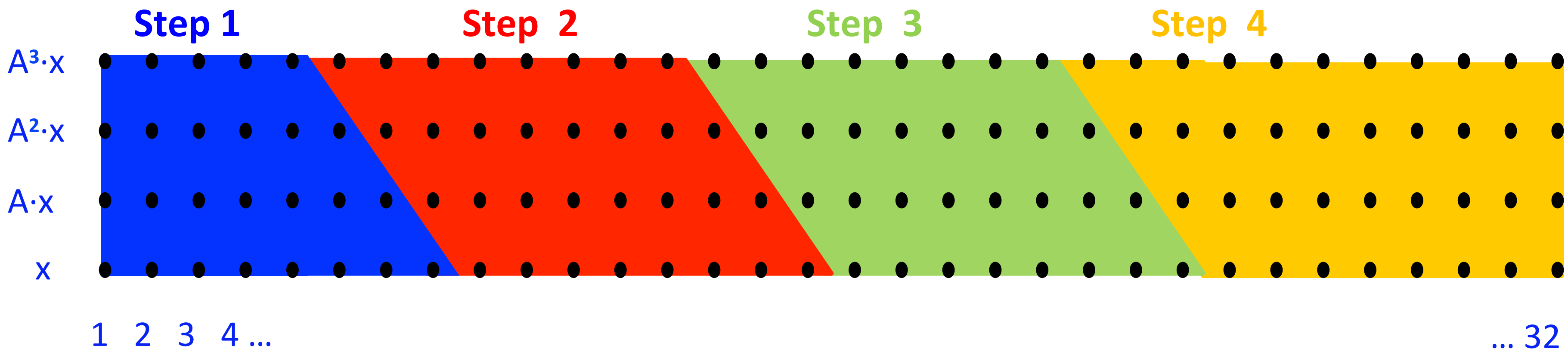


- Idea: pick up part of A and x that fit in fast memory, compute each of k products
- Example: A tridiagonal matrix (a 1D "grid"), n=32, k=3
- General idea works for any "well-partitioned" A

Communication Avoiding Kernels (Sequential case)

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**

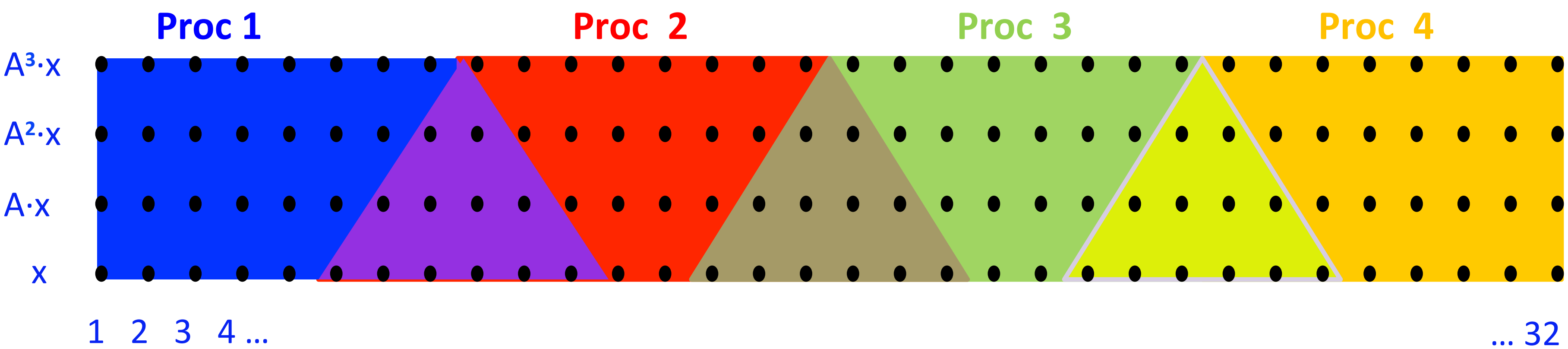


- Example: A tridiagonal, n=32, k=3
- Saves bandwidth (one read of A&x for k steps)
- Saves latency (number of independent read events)

Communication Avoiding Kernels: (Parallel case)

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

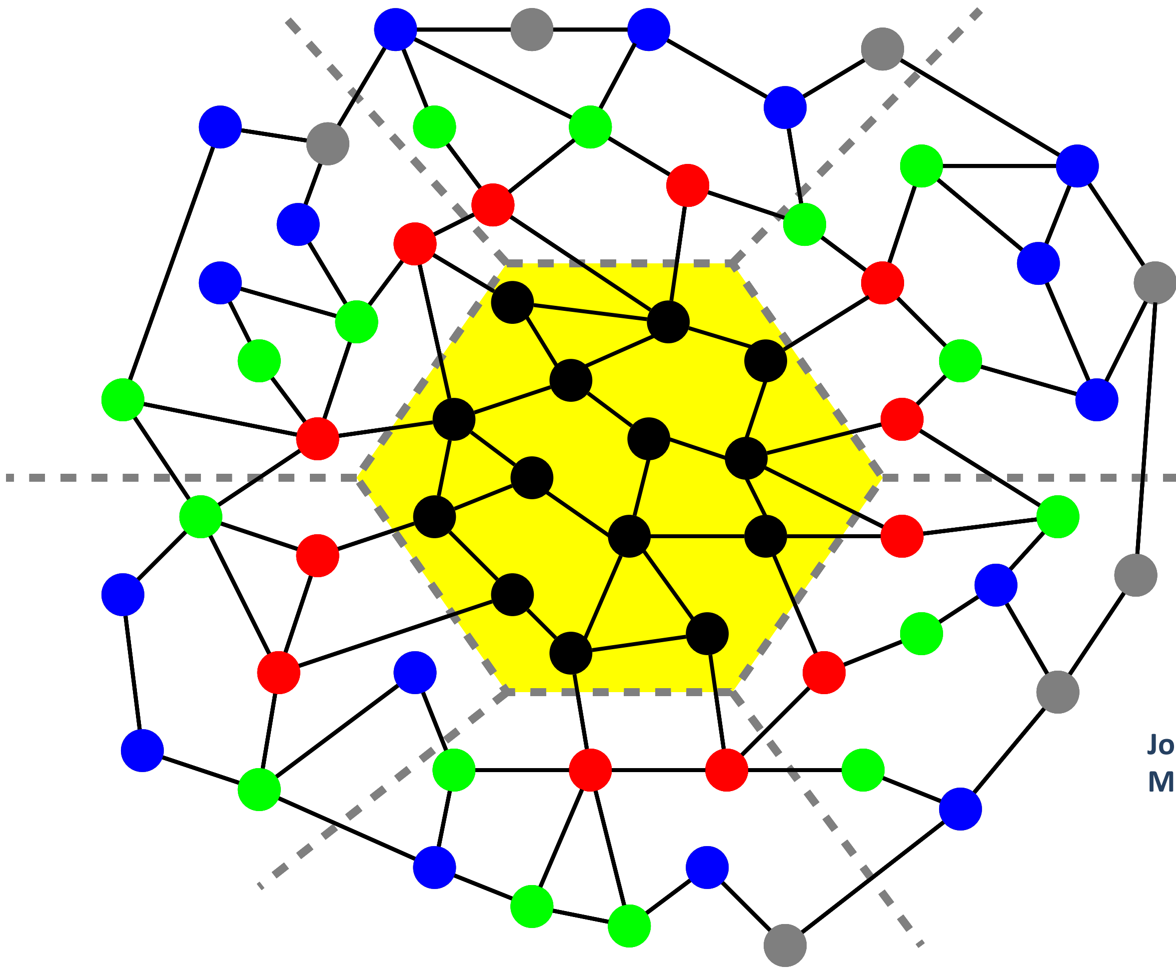
- Replace k iterations of $y = A \cdot x$ with $[Ax, A^2x, \ldots, A^kx]$
- **Parallel Algorithm**



- Example: A tridiagonal, n=32, k=3
- Each processor works on (overlapping) trapezoid
- Saves latency (# of messages); Not bandwidth

But adds redundant computation

Matrix Powers Kernel on a General Matrix

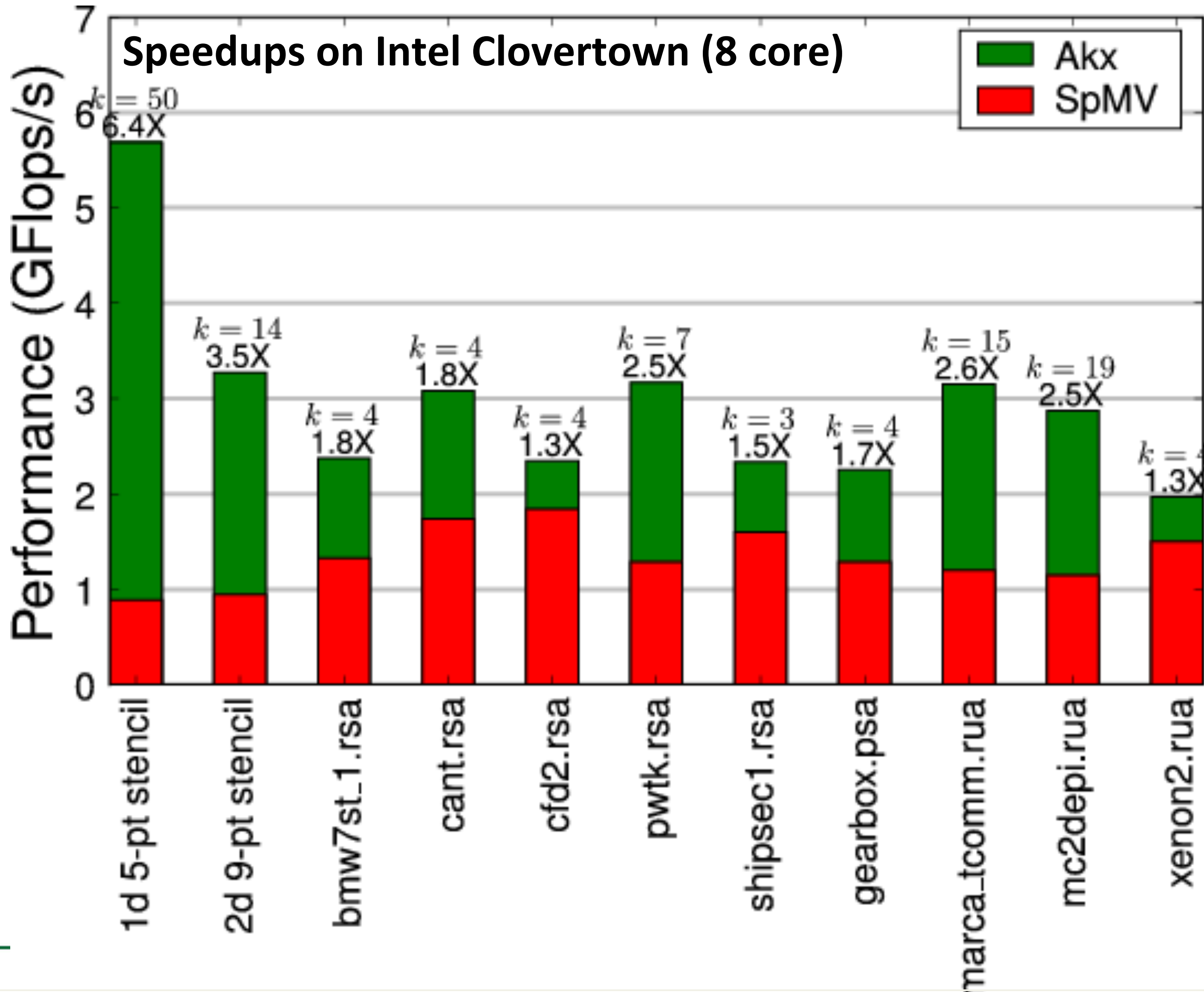


For implicit memory management (caches) uses a TSP algorithm for layout

Joint work with Jim Demmel, Mark Hoemman, Marghoob Mohiyuddin

- **Saves communication for "well partitioned" matrices**
  - Serial: O(1) moves of data moves vs. O(k)
  - Parallel: O(log p) messages vs. O(k log p)

$A^k x$ has **higher performance than Ax**



Jim Demmel, Mark Hoemmen, Marghoob Mohiyuddin, Kathy Yelick

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

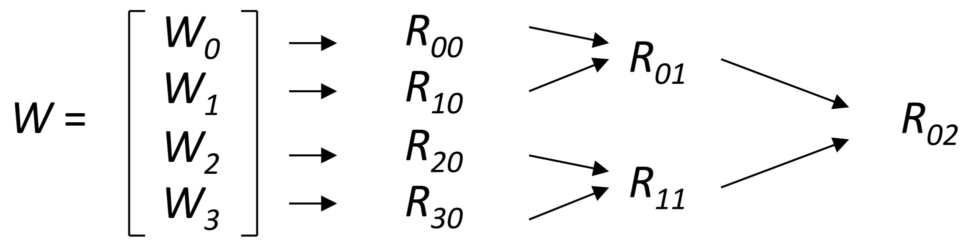
Sequential case: #words moved decreases by a factor of k
Parallel case: #messages decreases by a factor of k

- Oops – W from power method, precision lost!

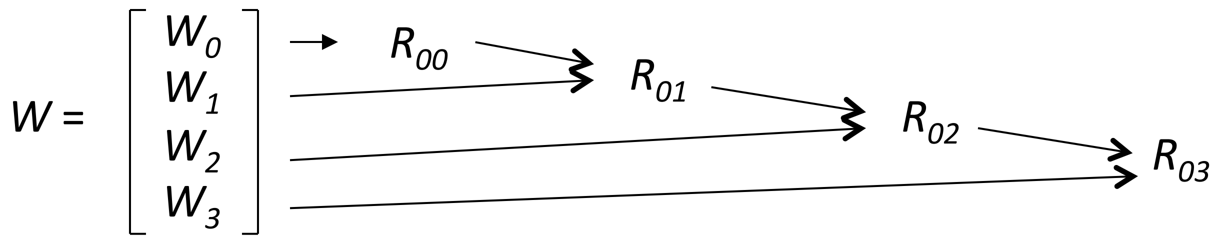
Mark Hoemmen, PhD thesis

TSQR: An Architecture-Dependent Algorithm

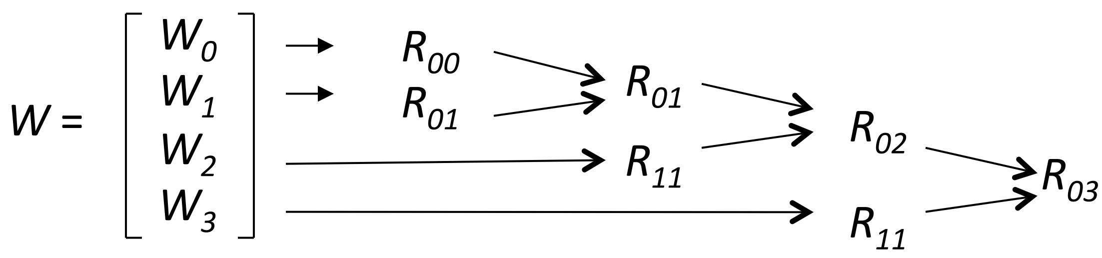
Parallel:



Sequential:



Dual Core:

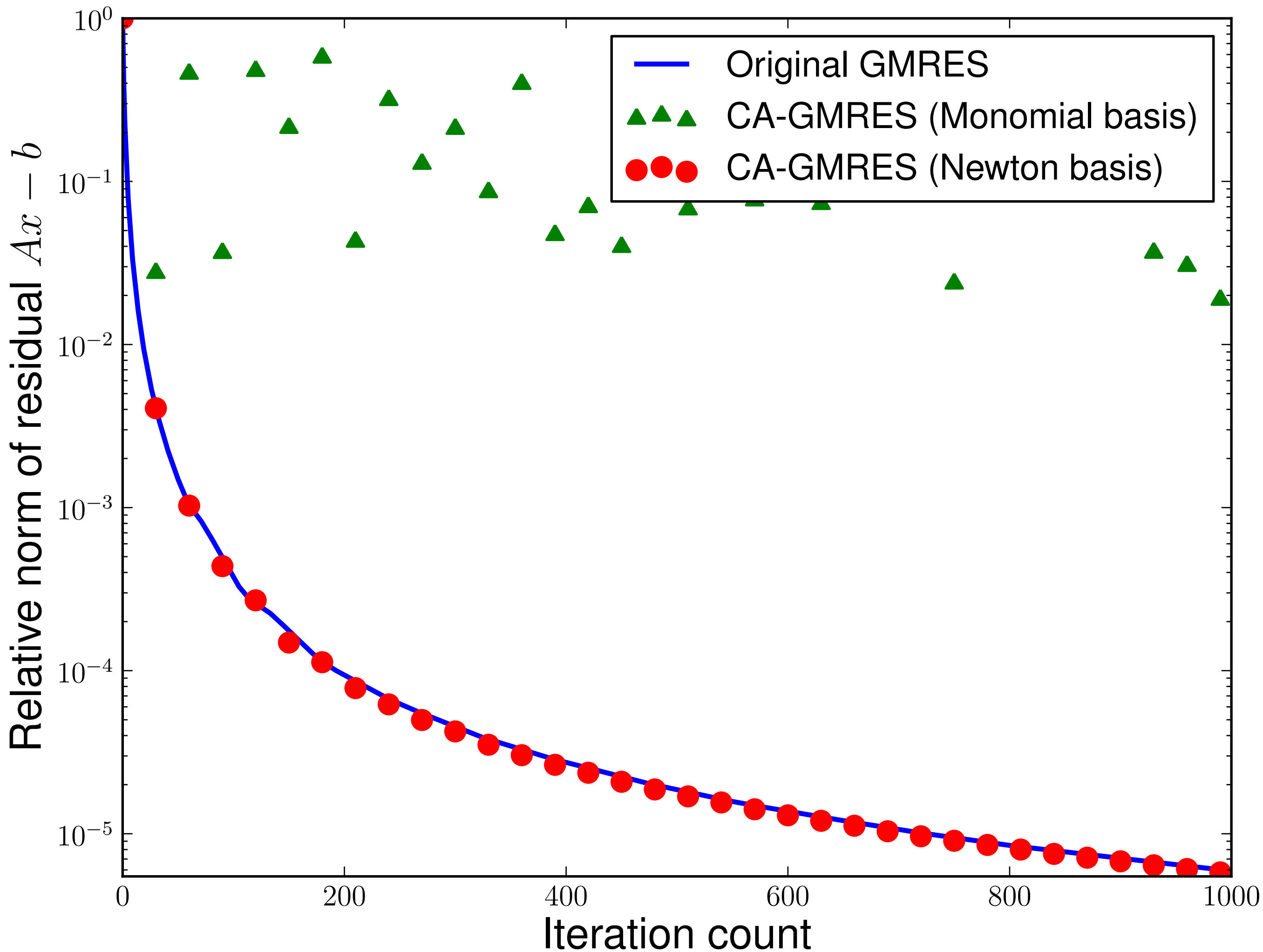


Multicore / Multisocket / Multirack / Multisite / Out-of-core: ?

Can choose reduction tree dynamically

Work by Laura Grigori, Jim Demmel, Mark Hoemmen, Julien Langou

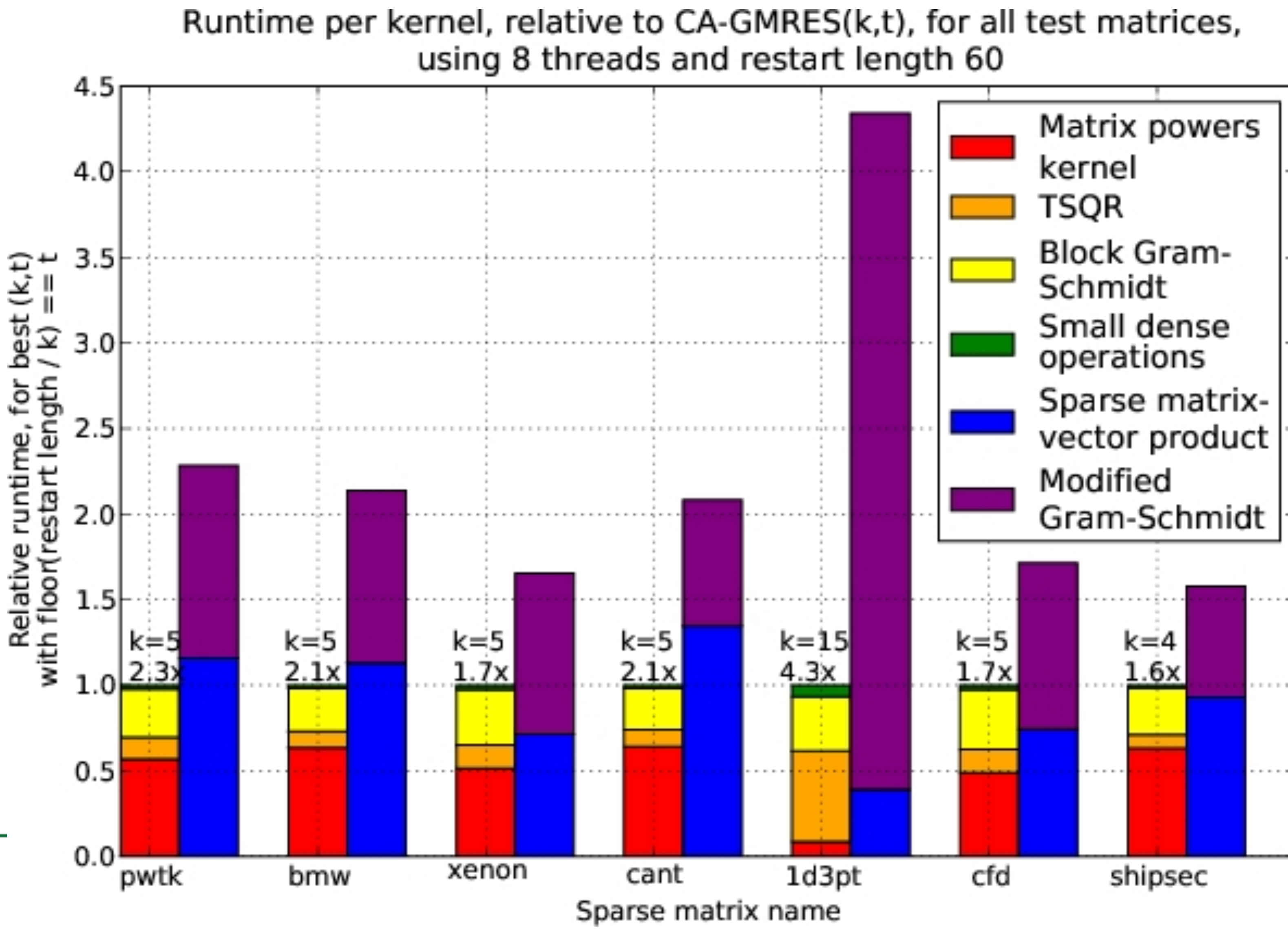
Matrix Powers Kernel (and TSQR) in GMRES



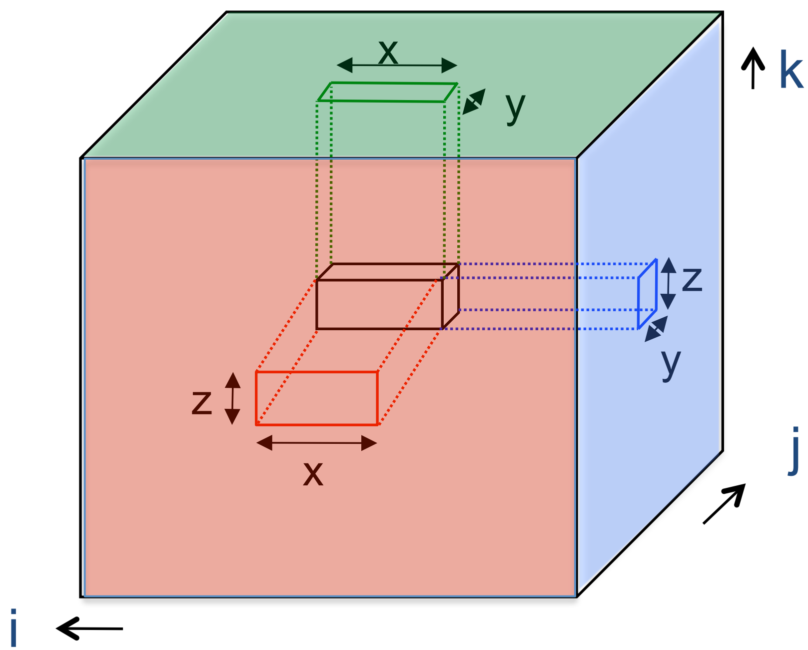
Jim Demmel, Mark Hoemmen, Marghoob Mohiyuddin, Kathy Yelick

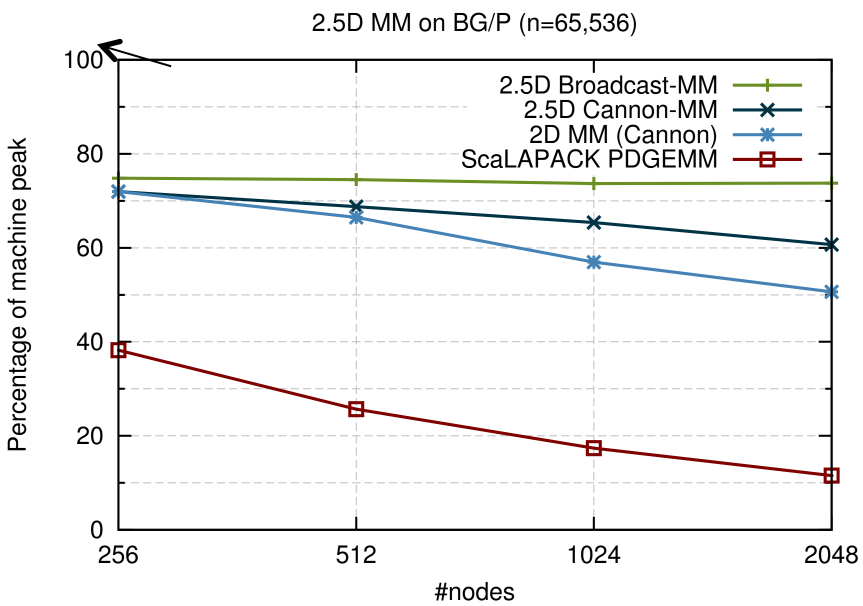
Communication-Avoiding Krylov Method (GMRES)

Performance on 8 core Clovertown



Towards Communication-Avoiding Compilers: Deconstructing 2.5D Matrix Multiply





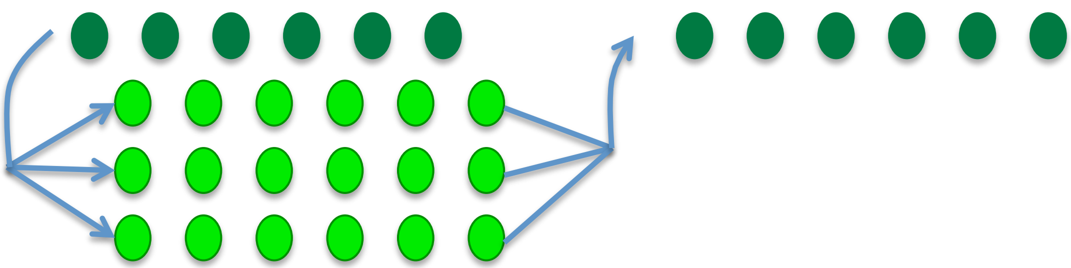
Matrix Multiplication code has a 3D iteration space
Each point in the space is a constant computation (*/+)

for i, for j, for k C[i,j] … A[i,k] … B[k,j] …

These are not just "avoiding," they are "communication-optimal"

Generalizing Communication Optimal Transformations to Arbitrary Loop Nests

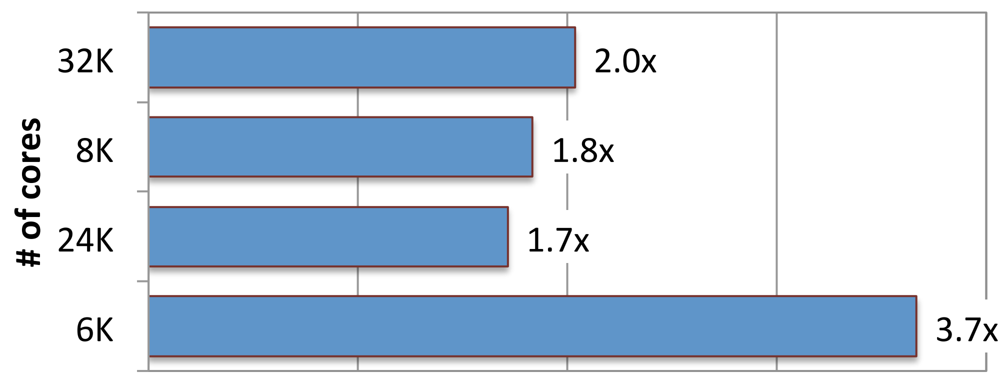
1.5D N-Body: Replicate and Reduce



The same idea (replicate and reduce) can be used on (direct) N-Body code:

1D decomposition → "1.5D"

Speedup of 1.5D N-Body over 1D



Does this work in general?

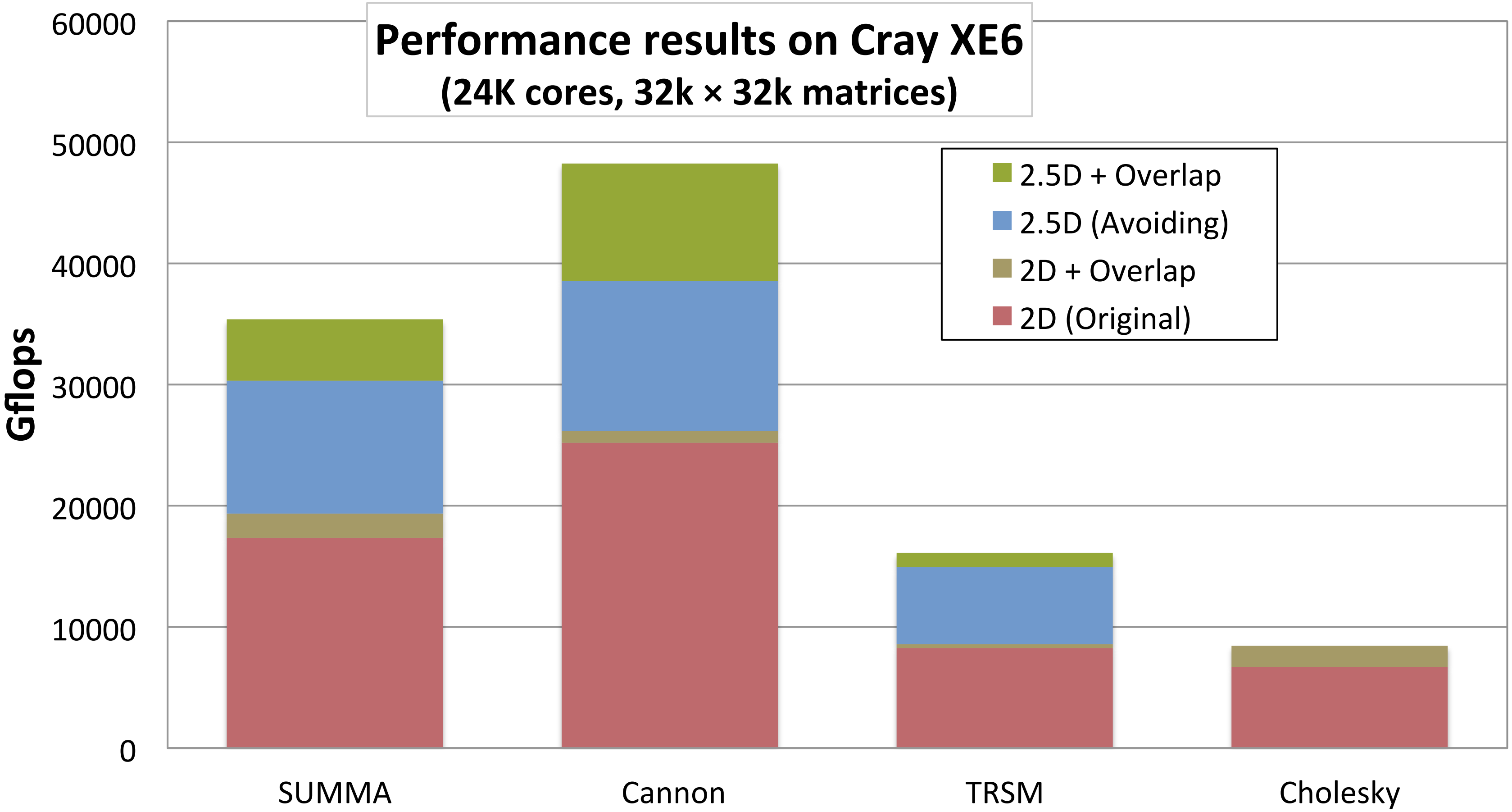
- *Yes, for certain loops and array expressions*
- *Relies on basic result in group theory*
- *Compiler work TBD*

A Communication-Optimal N-Body Algorithm for Direct Interactions, *Driscoll et al, IPDPS'13*

Generalizing Communication Lower Bounds and Optimal Algorithms

- **For serial matmul, we know #words_moved = $\Omega(n^3/M^{1/2})$, attained by tile sizes $M^{1/2}$ x $M^{1/2}$**
  - Where do all the ½'s come from?
- **Thm (Christ,Demmel,Knight,Scanlon,Yelick): For any program that "smells like" nested loops, accessing arrays with subscripts that are linear functions of the loop indices, #words_moved = $\Omega(\#\text{iterations}/M^{e})$, for some e we can determine**
- **Thm (C/D/K/S/Y): Under some assumptions, we can determine the optimal tiles sizes**
- **Long term goal: All compilers should generate communication optimal code from nested loops**

Communication Overlap Complements Avoidance



- **Even with communication-optimal algorithms (minimized bandwidth) there are still benefits to overlap and other things that speed up networks**
- ***Communication Avoiding and Overlapping for Numerical Linear Algebra*, Georganas et al, SC12**

Optimality of Communication

Lower bounds, (matching) upper bounds (algorithms) and a question:

Can we train compilers to do this?

See: http://www.eecs.berkeley.edu/Pubs/TechRpts/2013/EECS-2013-61.pdf

Beyond Domain Decomposition

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

c = 16 copies

Matrix multiplication on 16,384 nodes of BG/P

Surprises:

- Even Matrix Multiply had room for improvement
- Idea: make copies of C matrix (as in prior 3D algorithm, but not as many)
- Result is provably optimal in communication

Lesson: Never waste fast memory

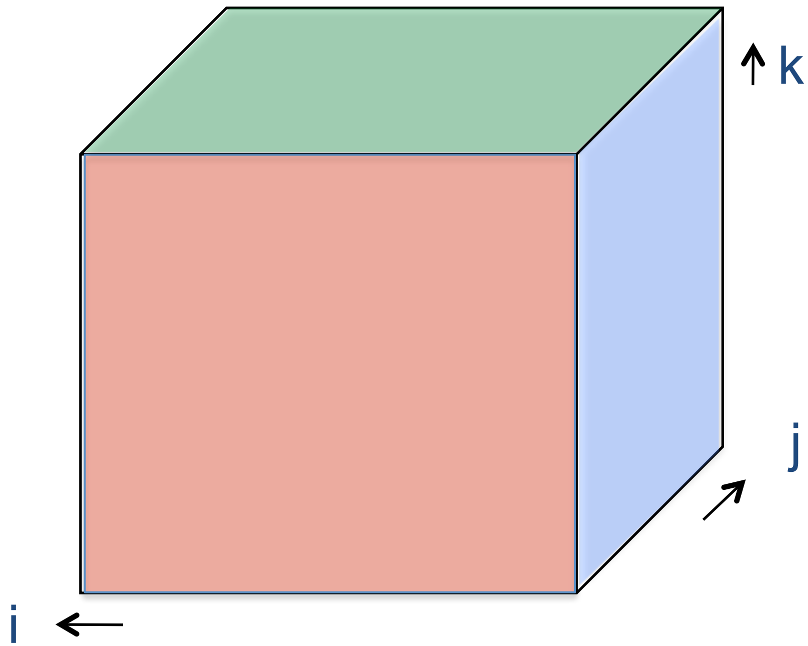
Can we generalize for compiler writers?

2D 2.5D 2, 2D 2, 2.5D

EuroPar'11 (Solomonik, Demmel)

SC'11 paper (Solomonik, Bhatele, Demmel)

Towards Communication-Avoiding Compilers: Deconstructing 2.5D Matrix Multiply



Tiling the iteration space

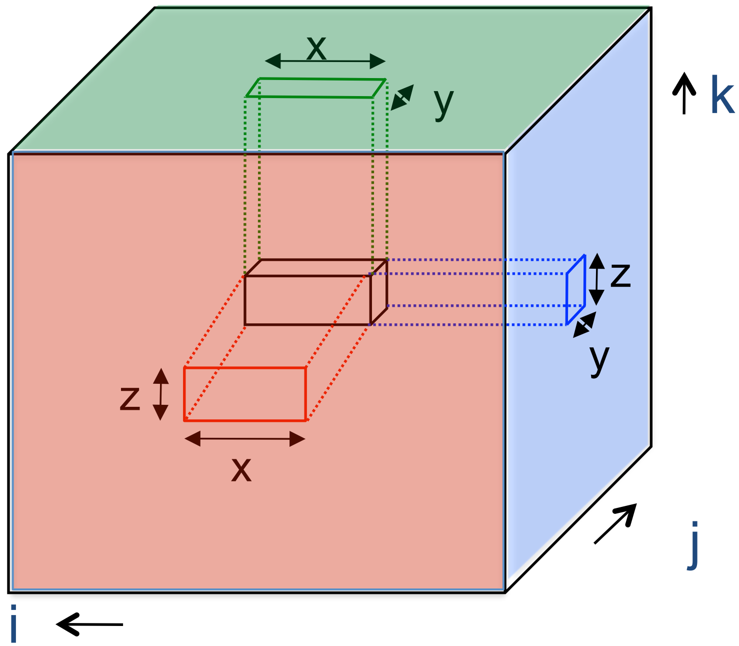
- **Compute a subcube**
- **Will need data on faces (projection of cube, subarrays)**
- **For s loops in the nest ➔ s dimensional space**
- **For x dimensional arrays, project to x dim space**

Matrix Multiplication code has a 3D iteration space
Each unit cube in the space is a constant computation (*/+)

```
for i
  for j
    for k
      C[i,j] … A[i,k] … B[k,j] …
```

Deconstructing 2.5D Matrix Multiply

Solomonik & Demmel



Tiling in the k dimension

- **k loop has dependencies because C (on the top) is a Left-Hand-Side variable**
  C += ..
- **Advantages to tiling in k:**
  - **More parallelism → Less synchronization**
  - **Less communication**

What happens to these dependencies?

- **All dependencies are vertical k dim (updating C matrix)**
- **Serial case: compute vertical block column in order**
- **Parallel case:**
  - **2D algorithm (and compilers): never chop k dim**
  - **2.5 or 3D: Assume + is associative; chop k, which implies replication of C matrix**

Beyond Domain Decomposition

x += …

x += …

x += …

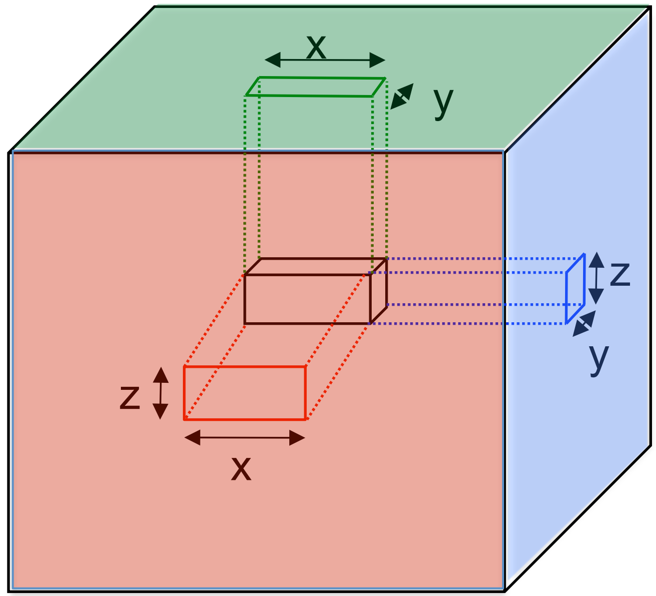
x += …

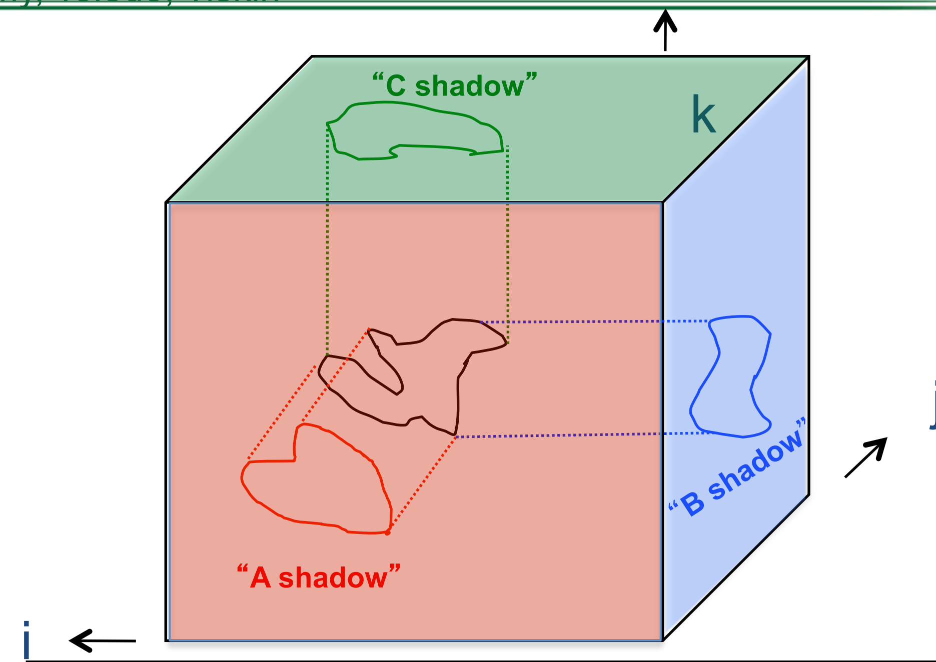
Using x for C[i,j] here

- **Much of the work on compilers is based on owner-computes**
  - For MM: Divide C into chunks, schedule movement of A/B
  - In this case domain decomposition becomes replication
- **Ways to compute C "pencil"**
  1. Serially
  2. Parallel reduction *Standard vectorization trick*
  3. Parallel asynchronous (atomic) updates
  4. Or any hybrid of these
- **For what types / operators does this work?**
  - "+" is associative for 1,2 rest of RHS is "simple"
  - and commutative for 3

Lower Bound Idea on C = A*B

Iromy, Toledo, Tiskin



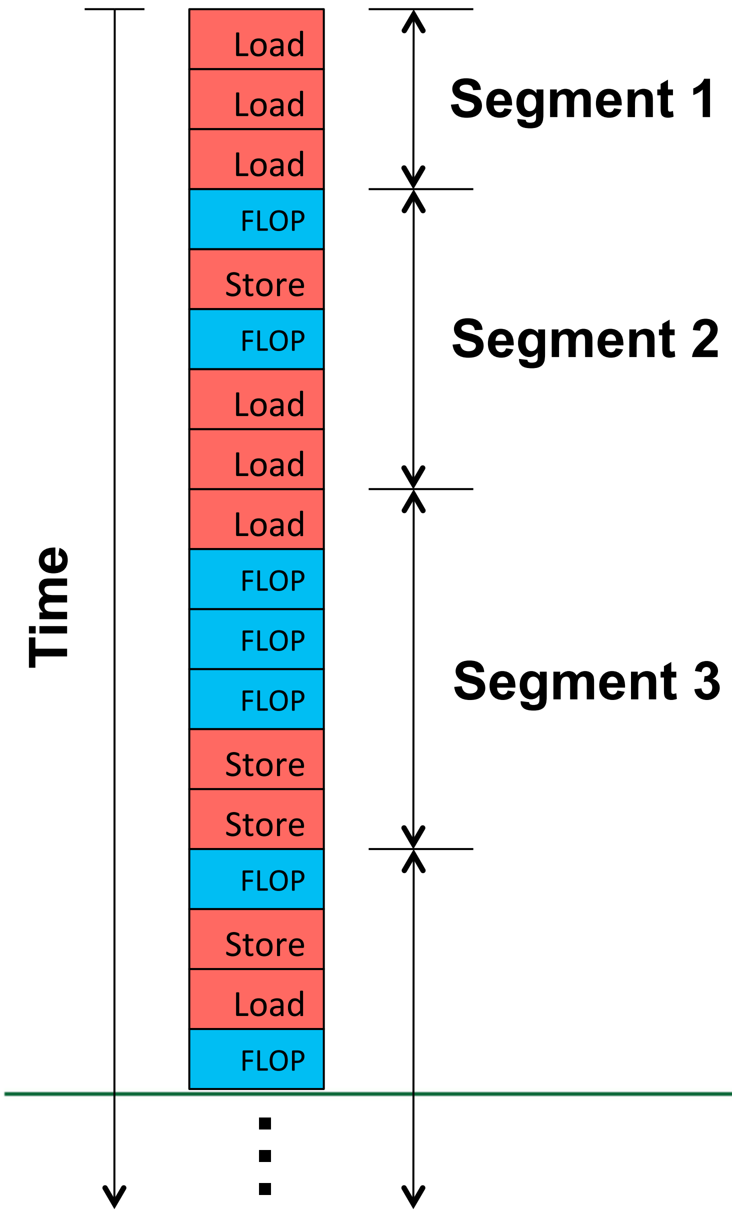


"Unit cubes" in black box with side lengths x, y and z
= Volume of black box
= x*y*z
= (#A□s * #B□s * #C□s)$^{1/2}$
= (xz * zy * yx)$^{1/2}$

(i,k) is in "A shadow" if (i,j,k) in 3D set
(j,k) is in "B shadow" if (i,j,k) in 3D set
(i,j) is in "C shadow" if (i,j,k) in 3D set

Thm (Loomis & Whitney, 1949)
cubes in 3D set = Volume of 3D set
≤ (area(A shadow) * area(B shadow) * area(C shadow)) $^{1/2}$

Lower Bound: What is the minimum amount of communication required?



- Proof from Irony/Toledo/Tiskin (2004)
- Assume fast memory of size M
- Outline (big-O reasoning):
  - Segment instruction stream, each with M loads/stores
  - Somehow bound the maximum number of flops that can be done in each segment, call it F
  - So F · # segments ≥ T = total flops = $2 \cdot n^3$, so # segments ≥ T / F
  - So # loads & stores = M · #segments ≥ M · T / F
- How much work (F) can we do with O(M) data?

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? Can we compute these for arbitrary programs?**

Generalizing Communication Lower Bounds and Optimal Algorithms

- **For serial matmul, we know #words_moved = $\Omega(n^3/M^{1/2})$, attained by tile sizes $M^{1/2}$ x $M^{1/2}$**

- **Thm (Christ,Demmel,Knight,Scanlon,Yelick):** *For any program that "smells like" nested loops, accessing arrays with subscripts that are linear functions of the loop indices*

  #words_moved = Ω (#iterations/M^e)

  for some e we can determine

- **Thm (C/D/K/S/Y): Under some assumptions, we can determine the optimal tiles sizes**
  - E.g., index expressions are just subsets of indices
- **Long term goal: All compilers should generate communication optimal code from nested loops**

New Theorem 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**

$$\Delta = \begin{array}{c} \\ \\ \\ \\ \end{array}\begin{array}{cccc} i & j & k & \\ \end{array}$$

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

- **Solve LP for x = [xi,xj,xk]^T: max 1^Tx s.t. Δ x ≤ 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 Theorem 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**

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

- **Solve LP for x = [xi,xj]^T: max 1^Tx s.t. Δ x ≤ 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

New Theorem 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**

$$\Delta = \begin{array}{c} \\ \\ \\ \\ \\ \\ \\ \end{array}\begin{array}{c} \begin{array}{cccccc} i1 & i2 & i3 & i4 & i5 & i6 \end{array} \\ \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \end{pmatrix} \end{array}\begin{array}{l} \\ A1 \\ A2 \\ A3 \\ A3,A4 \\ A5 \\ A6 \end{array}$$

- **Solve LP for $x = [x1,\ldots,x7]^T$: max $1^T x$ s.t. $\Delta\, x \le 1$**
  - 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 $M^{2/7}, M^{3/7}, M^{1/7}, M^{2/7}, M^{3/7}, M^{4/7}$

General Communication Bound

- Given S subset of Z^k, group homomorphisms ϕ_1, ϕ_2, …, bound $|S|$ in terms of $|\phi_1(S)|$, $|\phi_2(S)|$, … , $|\phi_m(S)|$
- Def: Hölder-Brascamp-Lieb LP (HBL-LP) for $s_1,\ldots,s_m$:

  for all subgroups $H < Z^k$, $\quad \mathrm{rank}(H) \le \Sigma_j\, s_j{*}\mathrm{rank}(\phi_j(H))$
- Thm (Christ/Tao/Carbery/Bennett): Given $s_1,\ldots,s_m$

$$|S| \le \Pi_j\, |\phi_j(S)|^{s_j}$$

- Thm: Given a program with array refs given by ϕ_j, choose s_j to minimize $s_{HBL} = \Sigma_j\, s_j$ subject to HBL-LP. Then

$$\#\text{words\_moved} = \Omega\,(\#\text{iterations}/M^{s_{HBL}-1})$$

Comments

- **Attainability depends on loop dependencies**

  Best case: none, or associative operators (matmul, nbody)

- **Thm: When all ϕ_j = {subset of indices}, dual of HBL-LP gives optimal tile sizes:**

  HBL-LP: minimize 1^T*s s.t. $s^T*\Delta \geq 1^T$

  Dual-HBL-LP: maximize 1^T*x s.t. $\Delta*x \leq 1$

  Then for sequential algorithm, tile i_j by M^{xj}

- **Ex: Matmul: $s = [\ 1/2\ ,\ 1/2\ ,\ 1/2\ ]^T = x$**
- **Generality:**
  - Extends to unimodular transforms of indices
  - Does not require arrays (as long as the data structures are injective containers)
  - Does not require loops as long as they can model computation

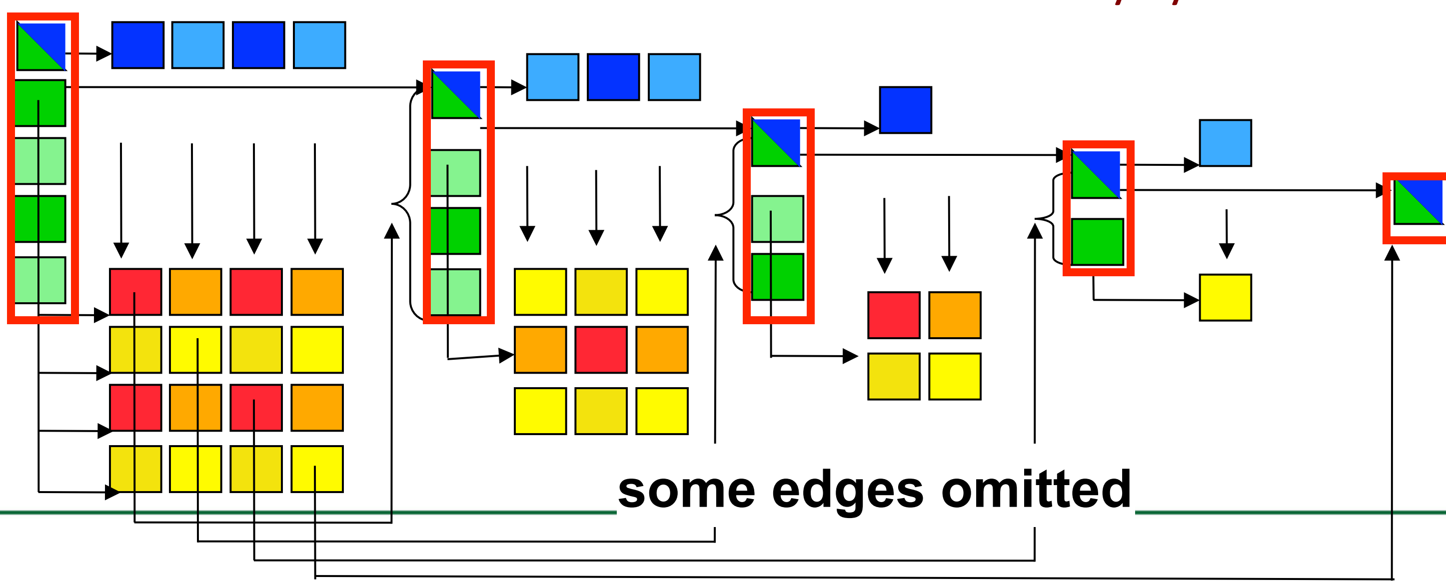
Conclusions

- **Communication is expensive and (relative) cost is growing**
  - Avoid bandwidth (data volume)
  - Hide latency or reduce number of messages
- **Conceptual model**
  - Think of computation a set of points in a volume in d-space (d = # loops in nest)
  - What is maximum amount you can do for a fixed surface area
- **Theory**
  - Lower bounds are useful to understand limits
  - Many programs (index expressions) still open for upper bounds

Bonus Slide #1: Beyond UPC

- **DAG Scheduling in a distributed (partitioned) memory context**
- **Assignment of work is static; schedule is dynamic**
- **Ordering needs to be imposed on the schedule**
  - Critical path operation: Panel Factorization
- **General issue: dynamic scheduling in partitioned memory**
  - Can deadlock in memory allocation
  - "memory constrained" lookahead

Uses a Berkeley extension to UPC to remotely synchronize



Bonus slide #2: Emerging Fast Forward Exascale Node Architecture

System on Chip (SoC) design coming into focus

Slide from John Shalf