Future of High Performance Linear Algebra Libraries

Jim Demmel
CScADS
31 July 2007

Collaborators

- Dense:
  - Jack Dongarra, Julien Langou, Julie Langou, Laura Grigori, Jessica Schoen, Yozo Hida, Jason Riedy, LAPACK groups at UTK, UCB
  - Ioana Dumitriu, Olga Holtz, Robert Kleinberg,
- Sparse:
  - Kathy Yelick, Mark Hoemmen, Marghoob Mohiyuddin, BEBOP group

Outline

- Common Challenges
- Dense Linear Algebra
  - Sca/LAPACK Goals
    - What is still missing
  - Novel Algorithms
    - With optimal complexity
  - Autotuning Space
    - Need for automation
- Sparse Linear Algebra
  - OSKI Goals
  - Novel Algorithms
  - Autotuning space

Common Challenges / Research Opportunities

- Increasing parallelism
  - From multicore in your laptop up to Petascale
- Exponentially growing gaps between
  - Floating point time << 1/Memory BW << Memory Latency
    - **Improving 59%/year vs 23%/year vs 5.5%/year**
  - Floating point time << 1/Network BW << Network Latency
    - **Improving 59%/year vs 26%/year vs 15%/year**
- Heterogeneity (performance and semantics)
- Asynchrony
- Unreliability

What *could* go into Sca/LAPACK?

For all linear algebra problems

For all matrix structures

For all data types

For all architectures and networks

For all programming interfaces

Produce best algorithm(s) w.r.t. performance and accuracy (including condition estimates, etc)

Need to prioritize, automate, enlist volunteers!

Sca/LAPACK Participants

- **UC Berkeley**:
  - Jim Demmel, Ming Gu, W. Kahan, Beresford Parlett, Xiaoye Li, Osni Marques, Christof Voemel, David Bindel, Yozo Hida, Jason Riedy, undergrads…
- **U Tennessee, Knoxville**
  - Jack Dongarra, Julien Langou, Julie Langou, Piotr Luszczek, Stan Tomov, Alfredo Buttari, Jakub Kurzak
- **Other Academic Institutions**
  - UT Austin, U Colorado, UC Davis, Florida IT, U Kansas, U Maryland, North Carolina SU, San Jose SU, UC Santa Barbara
  - TU Berlin, U Electrocomm. (Japan), FU Hagen, U Carlos III Madrid, U Manchester, U Umeå, U Wuppertal, U Zagreb
- **Research Institutions**
  - CERFACS, LBL
- **Industrial Partners**
  - Cray, HP, Intel, Interactive Supercomputing, MathWorks, NAG, SGI

Goals of next Sca/LAPACK

1. Expand contents
   - More functions, more parallel implementations
2. Better algorithms
   - Faster, more accurate
3. Automate performance tuning
4. Better software engineering
5. Improve ease of use
6. Increased community involvement

Goal 2: Better Algorithms

- Faster
  - But provide “usual” accuracy, stability
  - … Or accurate for an important subclass
- More accurate
  - But provide “usual” speed
  - … Or at any cost

Goal 2 – Faster Algorithms (1)

1. **MRRR algorithm for symmetric eigenproblem / SVD**:
   – Parlett / Dhillon / Voemel / Marques / Willems
2. **Up to 10x faster HQR for nonsymmetric eigenproblem**
   – Byers / Mathias / Braman
3. **Extensions to QZ for generalized eigenproblem**
   – Kågström / Kressner
4. **Reduce HQR from $O(n^3)$ to $O(n^2)$ for roots()**
   – Gu/Chandrasekaran/Zhu/Xia/Bindel/Garmire/D
5. **Faster Hessenberg, tridiagonal, bidiagonal reductions**
   – First steps for dense eigenvalue, SVD algorithm
   – Halve memory traffic in BLAS 2 part of bidiagonal reduction
   – van de Geijn/Quintana, Howell / Fulton / Hammarling/ D, Bischof / Lang
6. **Recursive blocked layouts for packed formats**:
   – Gustavson / Kågström / Elmroth / Jonsson/
7. **Mixed single/double precision (factor/refinement)**
   – 8x faster on Cell, 2x with SSE
   – Dongarra/Langou/Langou/Luszczek / Kurzak / Buttari
8. **Bisection on GPU**
   – Up to 43 GFlops by doing redundant work, 6.8x speedup

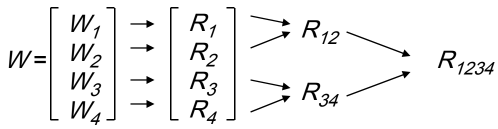
Goal 2 – Faster Algorithms (2)

- Thm (D., Dumitriu, Holtz, Kleinberg): If it is possible to multiply n-by-n matrices in $O(n^w)$ arithmetic operations, stably or not, then it is possible to
  – Solve Ax=b
  – Solve least squares problems
  – Compute Schur form / SVD

  in $O(n^{w+\varepsilon})$ operations, for any $\varepsilon > 0$, and stably!
- Ex: Coppersmith-Winograd => $w \approx 2.38$
- Practicality?
  – Elmroth/Gustavson used by Langou too

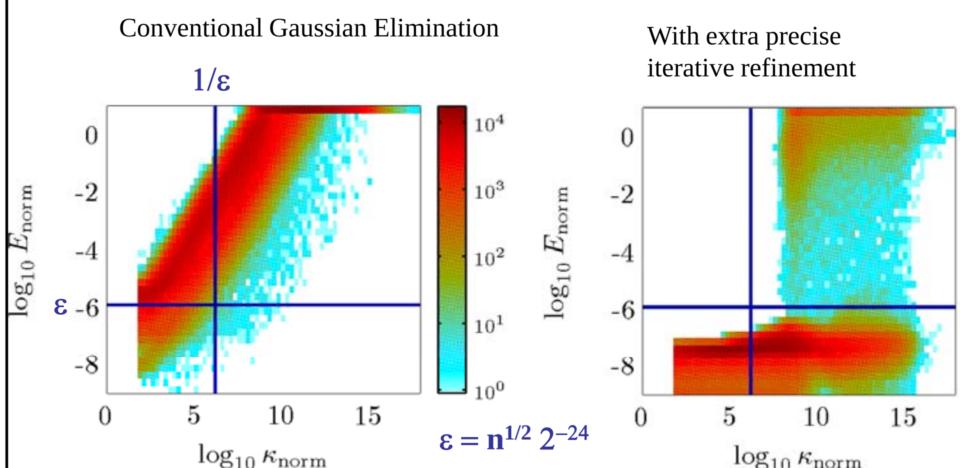
Goal 2 – Faster Algorithms (3)

- What is latency cost of factorization in parallel?
  - 2D block cyclic layout
  - QR and LU: O(N log p)
  - Cholesky: O(N log P / b), b = block size
  - Goal: reduce latency cost of QR and LU by factor b
- Idea (Details in Langou's talk)



- QR when N = b
  - TSQR = "Tall Skinny QR"
  - Latency cost = O(log P)

Goal 2: More Accurate Algorithms



- Arbitrary precision versions of all of LAPACK

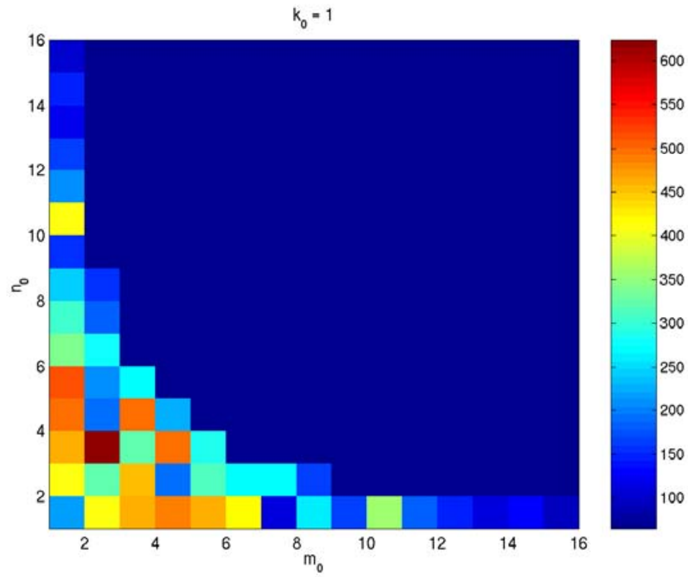
Goal 3: Automatic Performance Tuning

- Writing high performance software is hard
- Ideal: get high fraction of peak performance from one algorithm
- Reality: Best algorithm (and its implementation) can depend strongly on the problem, computer architecture, compiler,…
  - Changes with each new hardware, compiler release
- How much of this can we teach?
- How much of this can we automate?

Impact of Automatic Performance Tuning

- Widely used in performance tuning of Kernels
  - ATLAS (PhiPAC) - www.netlib.org/atlas
    - Dense BLAS, now in Matlab, many other releases
  - FFTW – www.fftw.org
    - Fast Fourier Transform and similar transforms, Wilkinson Software Prize
  - Spiral - www.spiral.net
    - Digital Signal Processing
- Talks from CScADS Autotuning Workshop
- Sparse tuning (second half of this talk)

Optimizing blocksizes for mat-mul



Finding a Needle in a Haystack – So Automate

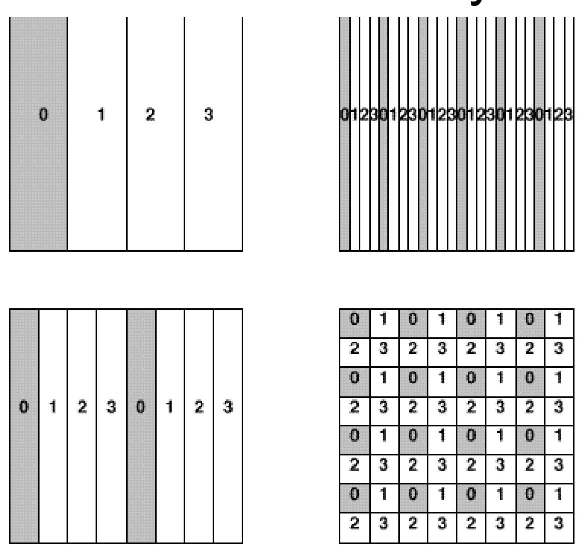
ScaLAPACK Data Layouts

1D Block

1D Cyclic

1D Block Cyclic

2D Block Cyclic



Speedups for using 2D block cyclic range from 2x to 8x
Cost of redistributing from 1D to best 2D layout 1% - 10%

How do we best explore this large tuning space?

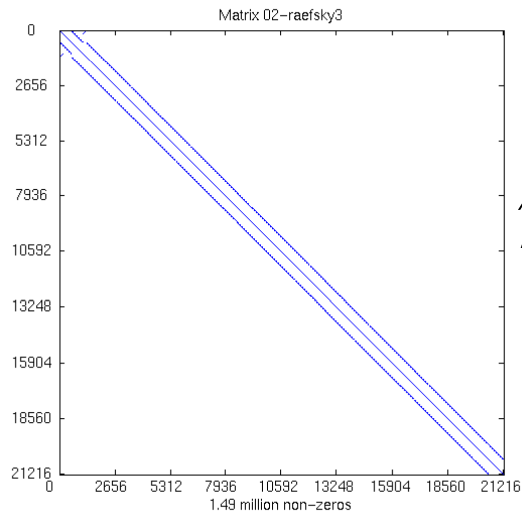
- **Algorithm tuning space includes**
  - **Numerous block sizes, not just in underlying BLAS (1300 ILAENV calls)**
  - **Many possible layers of parallelism, many mappings to HW**
  - **Different traversals of underlying DAGs**
    - **Left and right looking two of many; asynchronous algorithms (Buttari)**
  - **"Redundant" algorithms for GPUs**
  - **Recursive, parallel layouts and algorithms**
  - **New "optimal" algorithms for variations on standard factorizations**
  - **New and old eigenvalue algorithms**
  - **Mixed precision (for speed or accuracy)**
- **Is there a concise set of abstractions to describe, generate tuning space?**
  - **Block matrices, factorizations (partial, tree, …), DAGs, …**
  - **PLASMA, GCO, FLAME, Spiral, Telescoping languages, Bernoulli, Rose, …**
- **Question: What fraction of dense linear algebra can be generated/tuned?**
  - **Lots more than when we started**
    - **Sequential BLAS -> Parallel BLAS -> LU -> other factorizations -> …**
  - **Most of dense linear algebra?**
    - **Not eigenvalue algorithms (on compact forms)**
    - **What fraction of LAPACK can be done?**
    - **Rest of loop "for all linear algebra problems…"**
  - **For all interesting architectures…?**

Automatic Performance Tuning for Sparse Matrix Algorithms

- Tuning techniques for sparse matrix kernels
- OSKI = Optimized Sparse Kernel Interface
- Tuning on Multicore
- Berkeley Benchmarking and OPtimization (BeBOP)
  - bebop.cs.berkeley.edu
  - Codirected by Katherine Yelick

NASA structural analysis matrix

- y = y + A*x



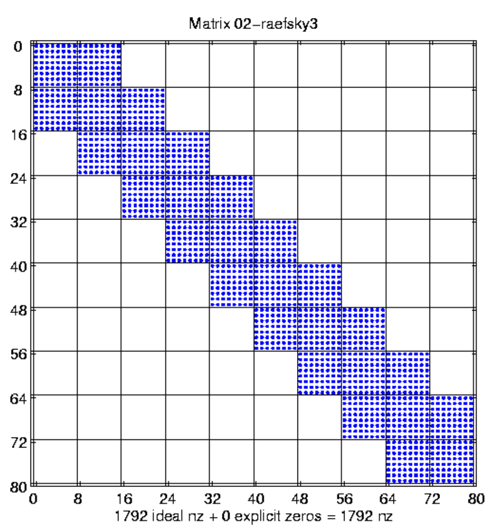
for all nonzero A(i,j)
*y(i) += A(i,j) * x(j)*

// Compressed Sparse Row
For each row i
s = 0
for k = rptr[i] to rptr[i+1]-1
*s += A[k] * x(* ***col(k)*** *)*
y(i) = s

Indirect access

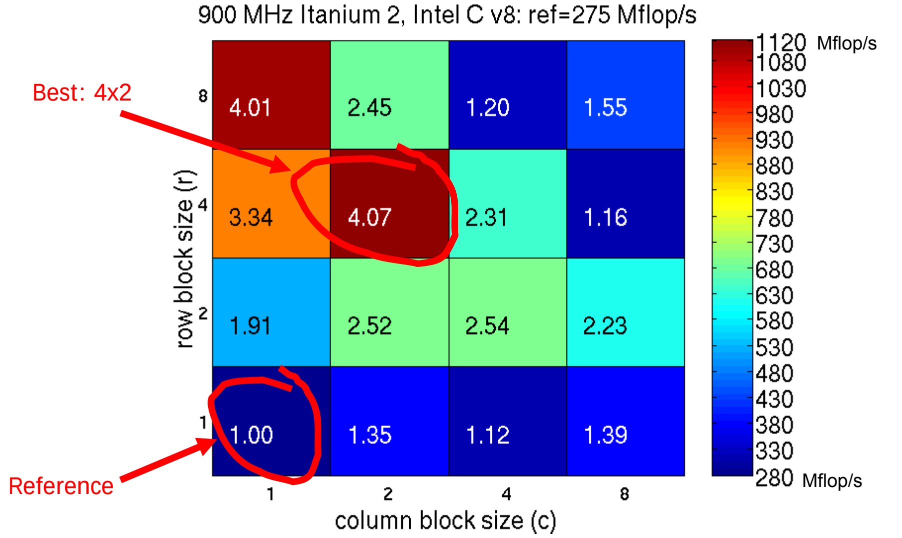
Memory bound

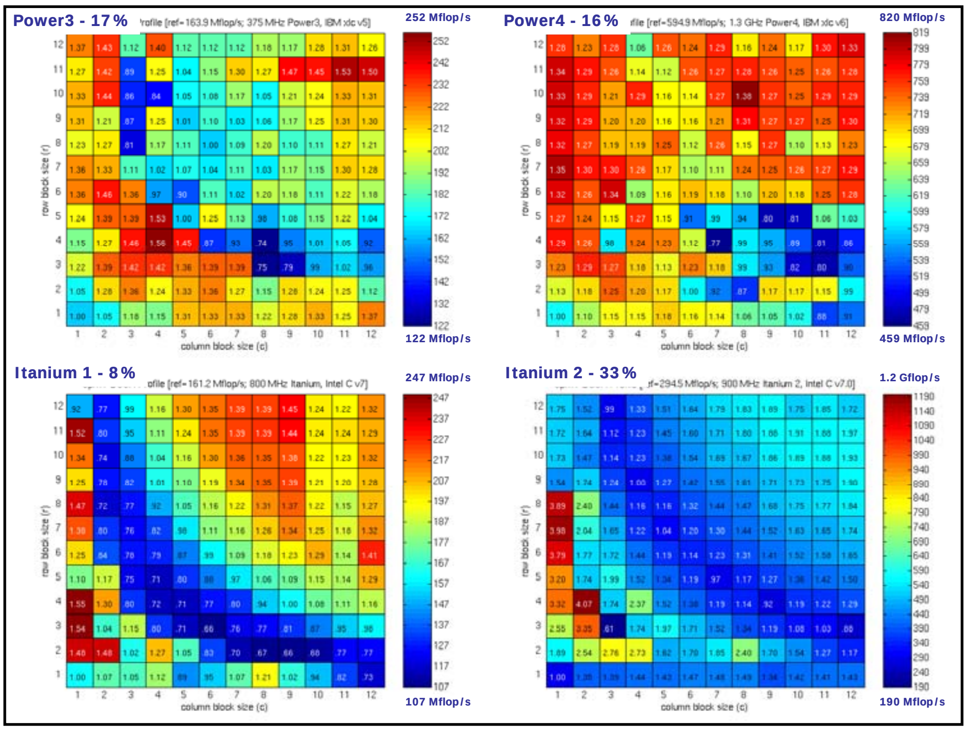
NASA matrix (zoom in)



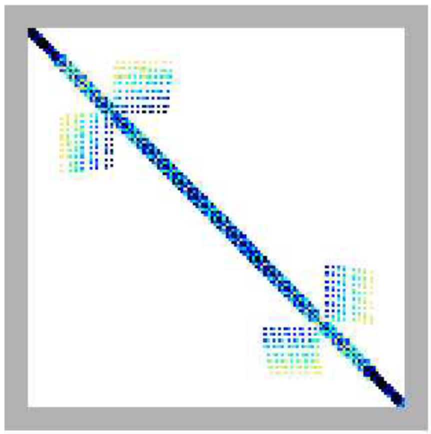
- One index per block
- Memory accesses minimized by storing 8x8 dense blocks

Speedups on Itanium 2: The Need for Search



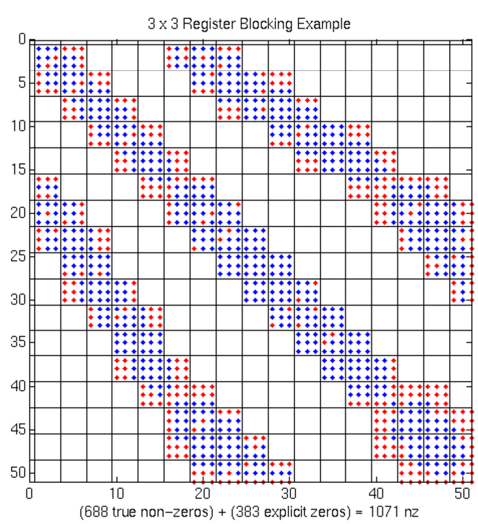


Another tuning challenge



- More complicated non-zero structure in general
- N = 16614
- NNZ = 1.1M

3x3 blocks create a lot of "fill-in"



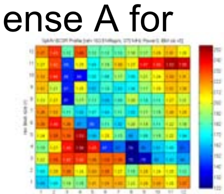
- **Need to store explicit zeros**
- **50% more entries**
- **1.5x as much arithmetic**
- **1.5x as much memory traffic**
- **Take 1.5x less time!**
- **Flop rate is $(1.5)^2 = 2.25$x higher on Pentium III**

Optimized Sparse Kernel Interface: OSKI

- Provides sparse kernels automatically tuned for user's function, matrix & machine (lots more to do)
  - Many functions, optimizations
  - Searches over large set of possible data structures and algorithms
  - Some search at install time, some at run-time
  - Performance models to prune search
  - Learns from past optimizations
  - Hides details from user
  - Used at DOE, Clearstream (lithography)
- Bebop.cs.berkeley.edu

Automatic Selection of Register Block Size (r x c)

- **Off-line benchmark**
  - Precompute **Mflops(r,c)** using dense A for each r x c
  - Once per machine/architecture



- **Run-time "search"**
  - Sample *A* to estimate **Fill(r,c)** for each r x c
- **Run-time heuristic model**
  - Choose r, c to minimize **time ~ Fill(r,c) / Mflops(r,c)**

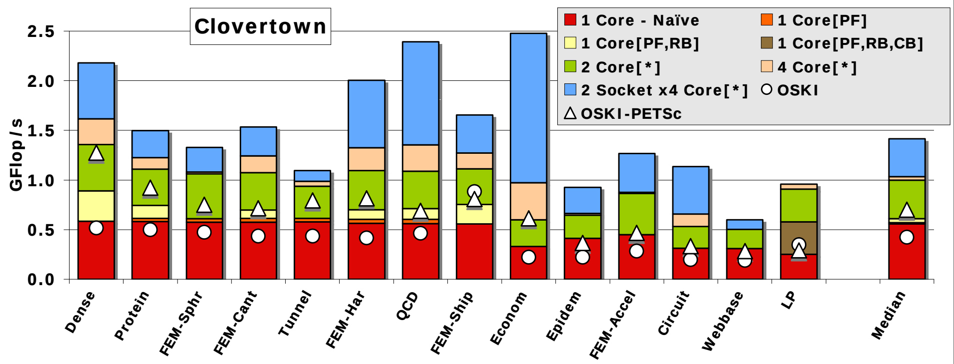
Summary of Other Performance Optimizations

- **Optimizations for SpMV**
  - **Register blocking (RB)**: up to **4x** over CSR
  - **Variable block splitting**: **1.8x** over RB
  - **Diagonals**: **2x** over CSR
  - **Reordering** to create dense structure + **splitting**: **2x** over CSR
  - **Symmetry**: **2.6x** over RB
  - **Cache blocking**: **2.8x** over CSR
  - **Multiple vectors (SpMM)**: **7x** over CSR
  - And combinations…
- **Sparse triangular solve**
  - **Hybrid sparse/dense data structure**: **1.8x** over CSR
- **Higher-level kernels**
  - **AA^T*x**, **A^TA*x**: **4x** over CSR, **1.8x** over RB
  - **A^2*x**: **2x** over CSR, **1.5x** over RB
- **Example** – Omega3P

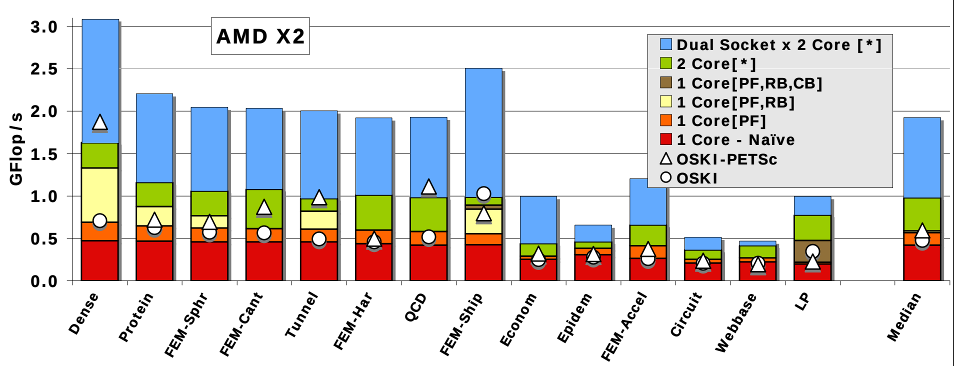
Tuning SpMV on Multicore

- **Larger search space for optimization:**
  - **Data structure optimization**
    - **Register, cache and TLB blocking**
    - **16 or 32 bit indices, block coordinate storage**
  - **Code optimization**
    - **Pipelining, no branches, SIMD, pointer arithmetic, prefetching**
  - **Parallelization optimizations**
    - **Threading, row parallelism**
    - **Process and memory affinity**
- **Different optimizations effective on different platforms**
  - **AMD X2, Intel Clovertown, Sun Niagra, STI Cell**

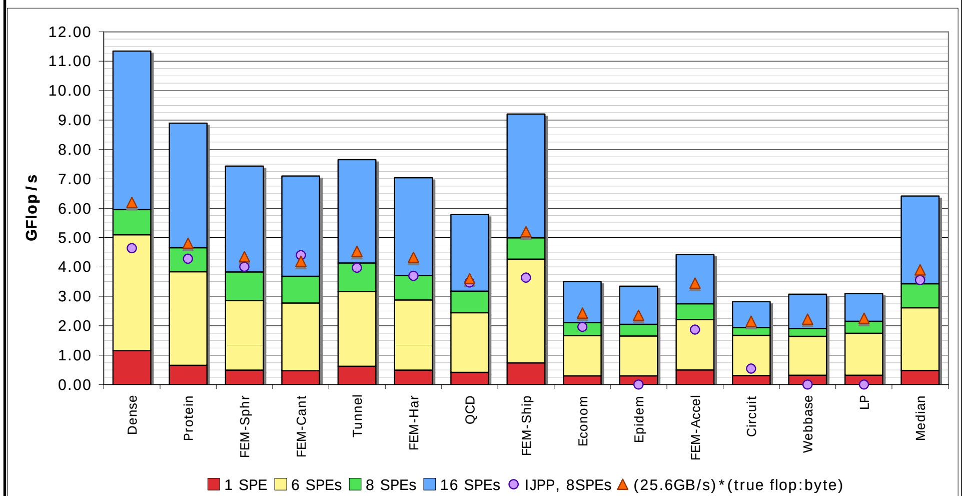
SpMV on Intel Clovertown



SpMV on AMD Opteron



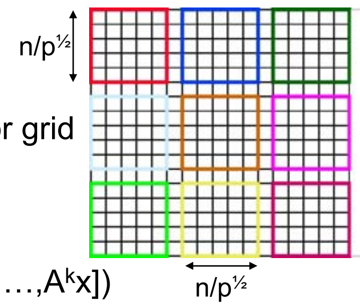
SpMV on Cell



Tuning at a Higher Level than SpMV

- Tuning (x,A,k) → $[x, Ax, A^2x, \ldots A^kx]$
- Optimal communication complexity algorithms for Krylov subspace methods
  - Latency of communication, for a parallel machine
  - Latency and bandwidth, for a memory hierarchy
- Example: GMRES for *Ax=b* on "2D Mesh"
- Performance modeling, measurements

Parallel Complexity

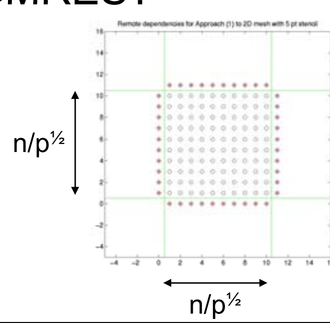


- Example matrix – “2D mesh”
  - *x* lives on n-by-n mesh
  - Partitioned on $p^{1/2}$ -by- $p^{1/2}$ processor grid
  - *A* has “5 point stencil” (Laplacian)
  - Ex: 18-by-18 mesh on 3-by-3 grid
- Cost = (flops, #words, #messages)
- Cost(conventional algorithm for $[x,Ax,\ldots,A^kx]$)
  $= (9kn^2 /p,\ 4kn / p^{1/2},\ 4k)$
- Cost(new algorithm for $[x,Ax,\ldots,A^kx]$)
  $= (9kn^2 /p + 9k^2n / p^{1/2},\ 4kn / p^{1/2} + 2k^2,\ 8)$
- Latency cost of new algorithm is O(1), optimal

Minimizing Communication

- What is the cost = (#flops, #words, #mess) of k steps of standard GMRES?

```
GMRES, v1:
  for i=1 to k
     w = A * v(i-1)
     MGS(w, v(0),…,v(i-1))
     update v(i), H
  endfor
  solve LSQ problem with H
```



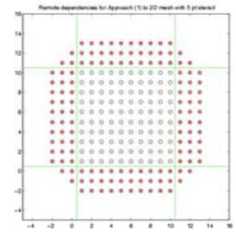
- Cost(A * v) = k * $(9n^2 /p,\ 4n / p^{1/2},\ 4)$
- Cost(MGS) = $k^2/2$ * $(4n^2 /p,\ \log p,\ \log p)$
- Total cost ~ Cost(A * v) + Cost (MGS)
- Can we reduce the latency?

Minimizing Communication

- Cost(GMRES, v1) = Cost(A^*v) + Cost(MGS)
  = ($9kn^2/p$, $4kn / p^{½}$, 4k) + ($2k^2n^2/p$, $k^2 \log p / 2$, $k^2 \log p / 2$)

- How much latency cost from A^*v can you avoid? Almost all

GMRES, v2:
$W = [\ v, Av, A^2v, \dots, A^kv\ ]$
$[Q,R]$ = MGS(W)
Build H from R, solve LSQ problem



s = 3

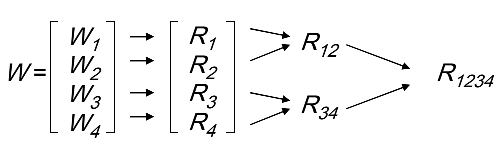
- Cost(W) = (~ same, ~ same , 8)
  - Latency cost *independent* of k – ***optimal***
- Cost (MGS) unchanged
- Can we reduce the latency more?

Minimizing Communication

- Cost(GMRES, v2) = Cost(W) + Cost(MGS)
  = ($9kn^2/p$, $4kn / p^{½}$, 8) + ($2k^2n^2/p$, $k^2 \log p / 2$, $k^2 \log p / 2$)

- How much latency cost from MGS can you avoid? Almost all

GMRES, v3:
$W = [\ v, Av, A^2v, \dots, A^kv\ ]$
$[Q,R]$ = TSQR(W) … "Tall Skinny QR"
Build H from R, solve LSQ problem

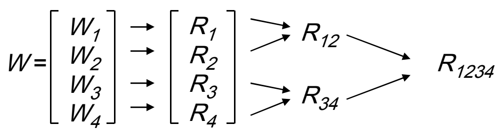


- Cost(TSQR) = (~ same, ~ same , log p)
  - Latency cost *independent* of k - ***optimal***

Minimizing Communication

- Cost(GMRES, v2) = Cost(W) + Cost(MGS)
  = ($9kn^2/p$, $4kn / p^{1/2}$, 8) + ($2k^2n^2/p$, $k^2 \log p / 2$, $k^2 \log p / 2$)
- How much latency cost from MGS can you avoid? Almost all

GMRES, v3:
  $W = [\ v, Av, A^2v, \ldots, A^kv\ ]$
  $[Q,R]$ = TSQR(W) … "Tall Skinny QR"
  Build H from R, solve LSQ problem

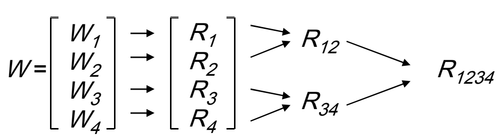


- Cost(TSQR) = (~ same, ~ same , log p)
- ***Oops***

Minimizing Communication

- Cost(GMRES, v2) = Cost(W) + Cost(MGS)
  = ($9kn^2/p$, $4kn / p^{1/2}$, 8) + ($2k^2n^2/p$, $k^2 \log p / 2$, $k^2 \log p / 2$)
- How much latency cost from MGS can you avoid? Almost all

GMRES, v3:
  $W = [\ v, Av, A^2v, \ldots, A^kv\ ]$
  $[Q,R]$ = TSQR(W) … "Tall Skinny QR"
  Build H from R, solve LSQ problem



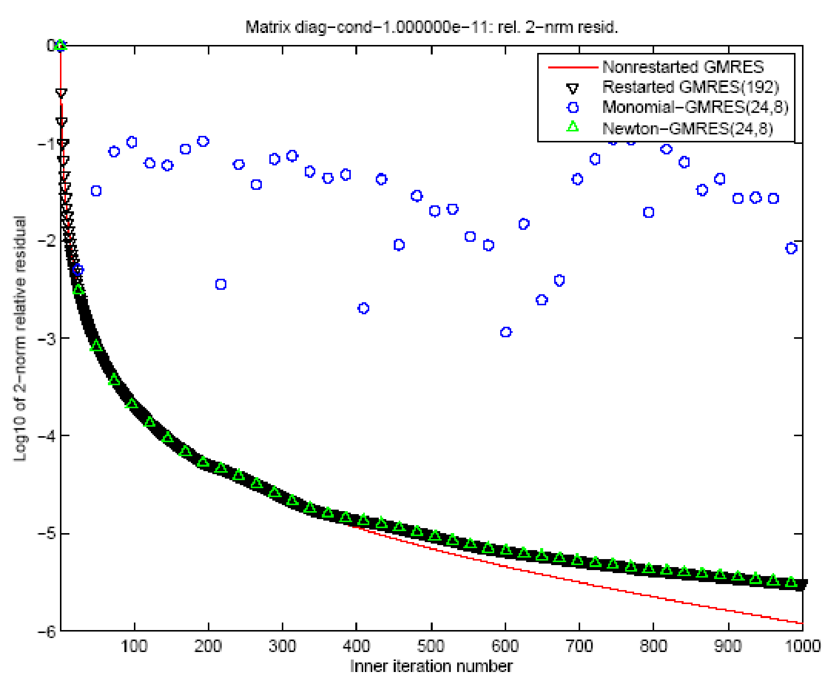
- Cost(TSQR) = (~ same, ~ same , log p)
- ***Oops – W from power method, precision lost!***

Minimizing Communication

- Cost(GMRES, v3) = Cost(W) + Cost(TSQR)
  = ($9kn^2/p$, $4kn/p^{1/2}$, 8) + ($2k^2n^2/p$, $k^2 \log p / 2$, $\log p$)

- Latency cost independent of k, just log p – optimal
- Oops – W from power method, so precision lost – What to do?

- Use a different polynomial basis
- Not Monomial basis $W = [v, Av, A^2 v, \ldots]$, instead …
- Newton Basis $W_N = [v, (A - \theta_1 I) v, (A - \theta_2 I)(A - \theta_1 I) v, \ldots]$ or
- Chebyshev Basis $W_C = [v, T_1(v), T_2(v), \ldots]$

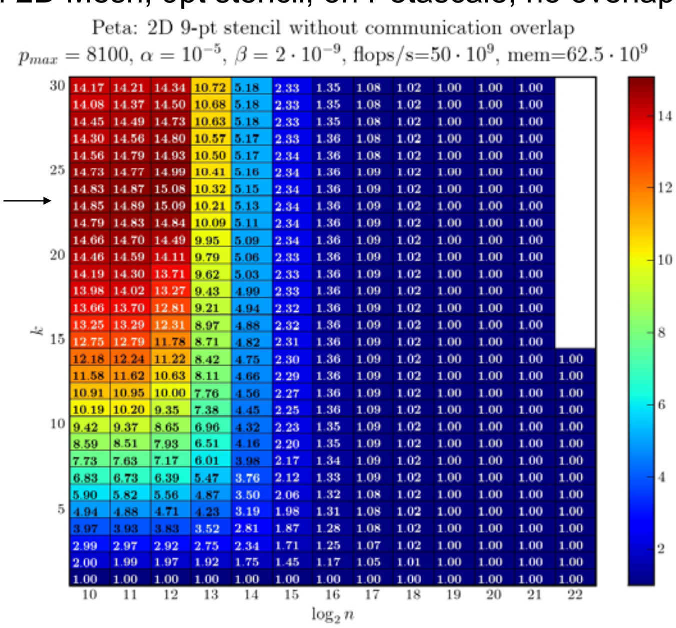


Performance Modeling

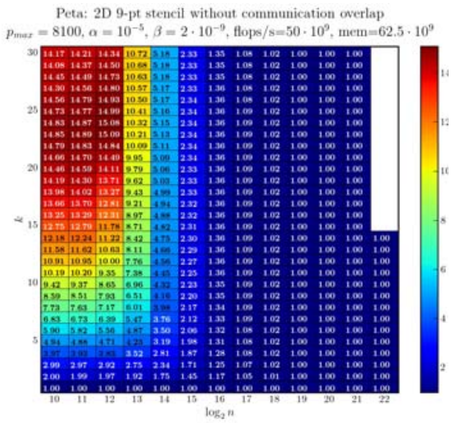
- Petascale
  - Max # processor =8100
  - Memory/processor = 6.25 $\cdot 10^{9}$ words
  - Flop time = 2 $\cdot 10^{-11}$ secs (50 GFlops/s)
  - Latency = 10^{-5} secs
  - 1/Bandwidth = 2 $\cdot 10^{-9}$ secs (4 GB/s)
- Grid
  - Max # processor = 125
  - Memory/processor = 1.2 $\cdot 10^{12}$ words
  - Flop time = 10^{-12} secs (1 TFlops/s)
  - Latency = .1 secs
  - 1/Bandwidth = 25 $\cdot 10^{-9}$ secs (.32 GB/s)

Speedup of 2D Mesh, 9pt stencil, on Petascale, no overlap

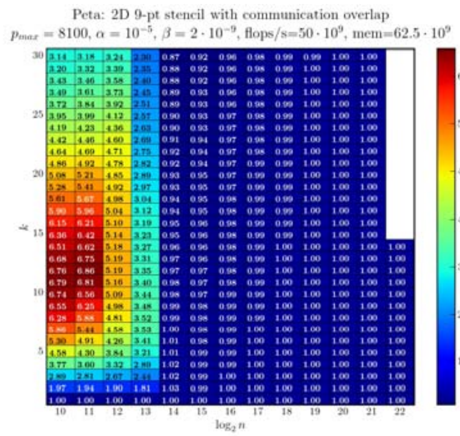
Peak speedup



Speedup of 2D Mesh, 9pt stencil, on Petascale

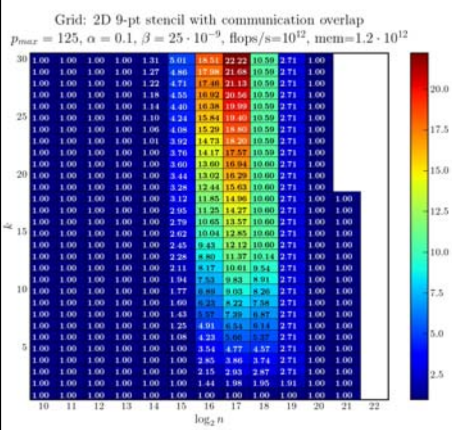


Without overlap

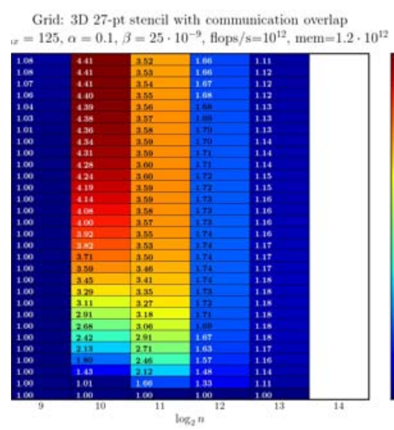


With overlap

Speedup on Grid, with overlap



2D, 9 point stencil



3D, 27 point stencil

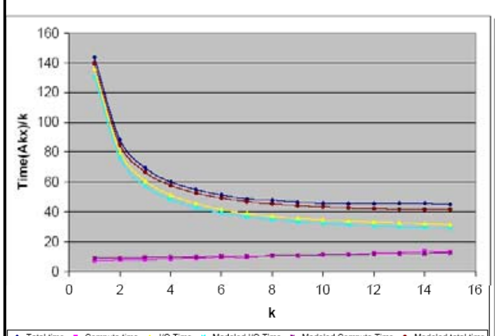
Latency *and Bandwidth* Avoiding Sequential Kernel for [x, Ax, … , A^kx]

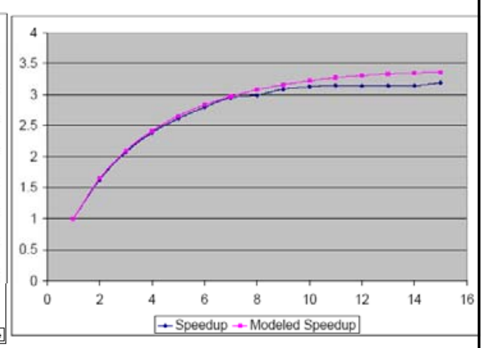
- Mimic parallel algorithm:
  - For i = 1 to #blocks of x
    - Load rows of A needed to compute block i of [Ax,…,A^kx] (including remotely dependent entries)
    - Load block i of x and parts of x from neighboring blocks needed to compute remotely dependent entries of [Ax,…,A^kx]
    - Compute block i of [Ax,…,A^kx]
- #Blocks chosen to fit as much of A and [x,Ax,…,A^kx] in fast memory as possible
  - Double buffering, other optimizations possible
- Optimal in sense that all data moved between fast and slow memory ≈once
  - 1 + (k·surface/volume) times
  - Increase computational intensity k-fold

Measured and Modeled Performance

5.2 GFlop Itanium2, 4GB memory, Disk

1/f= 300MFlops/s, BW_{read} = 140 MB/s, BW_{write} = 30 MB/s, disk latency irrelevant





3D mesh, 27-pt stencil, n = 368, p = 64 blocks,

Measured Speedup up to 3.2x (flop time ≈ ½ bandwidth time)

Summary of Optimal Sparse Algorithms

- Tuning and algorithmic design interact
- Can eliminate latency from GMRES, CG, … maintaining stability
  - Ideas go back to Van Rosendale (1983), Chronopoulos & Gear (1989), Toledo (1991) many others, but without simultaneous stability & optimality
- Extends to preconditioned methods
  - Kernel becomes [x,Ax,MAx,AMAx,MAMAx,…,$(MA)^k x$]
  - But only some preconditioners let us eliminate latency, not raise flop count a lot (work in progress)
- Lots of tuning opportunities
  - All SpMV techniques, plus choosing k, polynomial in kernel, partitioning, overlapping communication and computation, …

Summary

- Tuning spaces for Dense and Sparse Libraries large and promising
  - New algorithms for new architectures
  - Need to break old interfaces to make progress
    - BLAS2, BLAS3 -> BLAS2.5 in SVD
    - SpMV -> Krylov method
  - New minimal communication algorithms
- SW problem moves from managing complexity of libraries to managing autotuners