

Why are GPUs so hard to program – or are they?

Wen-mei Hwu

University of Illinois, Urbana-Champaign
MulticoreWare

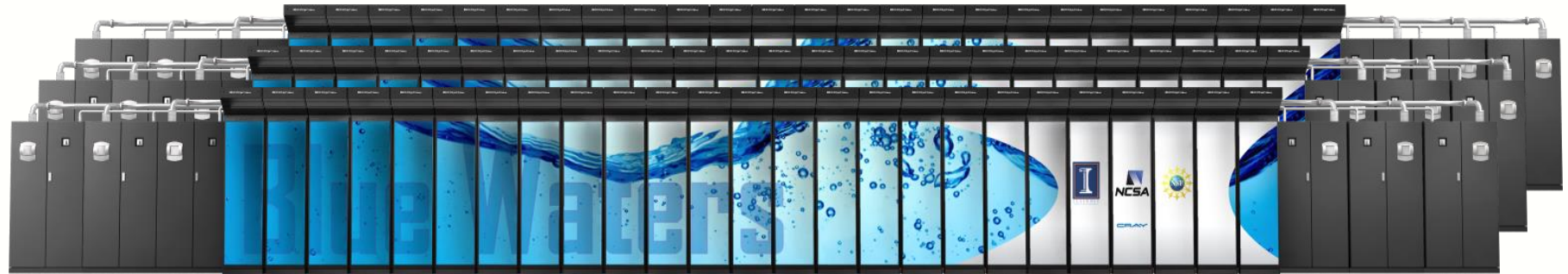


Agenda

- GPU Computing in Blue Waters
- Library Algorithms
 - Scalability, performance, and numerical stability
- Programming Interfaces
- Conclusion and Outlook

Blue Waters – the Most Powerful Computer for the NSF Community

Operational at Illinois since 11/2012



11.1 PF
1.5 PB DRAM

**10/40/100 Gb
Ethernet Switch**

IB Switch

>1 TB/sec

120+ Gb/sec

100 GB/sec

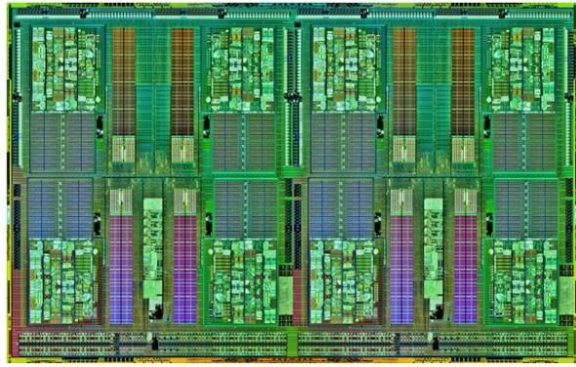


Spectra Logic: 300 PBs



Sonexion: 26 PBs

Heart of Blue Waters: Two New Chips



AMD Interlagos

157 GF peak performance

Features:

2.3-2.6 GHz

8 core modules, 16 threads

On-chip Caches

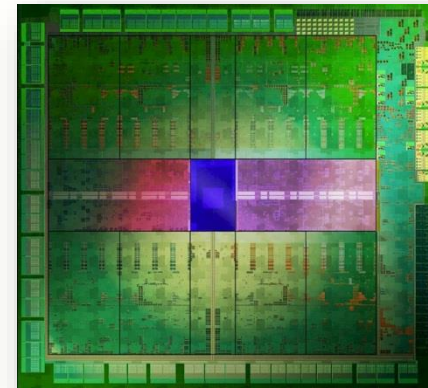
L1 (I:8x64KB; D:16x16KB)

L2 (8x2MB)

Memory Subsystem

Four memory channels

51.2 GB/s bandwidth



NVIDIA Kepler

1,300 GF peak performance

Features:

15 Streaming multiprocessors (SMX)

SMX: 192 CUDA SPs, 64 dp units,

32 special function units

L1 caches/shared mem (64KB, 48KB)

L2 cache (1,536KB)

Memory subsystem

Six memory channels

250 GB/s bandwidth

Blue Waters vs. Titan

System Attribute	NCSA Blue Waters	ORNL Titan
Vendors	Cray/AMD/NVIDIA	Cray/AMD/NVIDIA
Processors	Interlagos/Kepler	Interlagos/Kepler
Total Peak Performance (PF)	11.1	27.1
Total Peak Performance (CPU/GPU)	7.1/4	2.6/24.5
Number of CPU Chips	48,352	18,688
Number of GPU Chips	3,072	18,688
Amount of CPU Memory (TB)	1511	584
Interconnect	3D Torus	3D Torus
Amount of On-line Disk Storage (PB)	26	13.6
Sustained Disk Transfer (TB/sec)	>1	0.4-0.7
Amount of Archival Storage	300	15-30
Sustained Tape Transfer (GB/sec)	100	7



Why did we have only 3,072 GPUs in Blue Waters?

- Blue Waters will be the leadership machine for the U.S. science community for at least two years
 - Must minimize risk for petascale application teams
- The NSF review panel was very concerned about GPU usability in 2011 (Blue Waters redesign)
 - Hard to program for application teams
 - Small DRAM – 6GB
 - Lack of at-scale experience
 - Lack of begin-to-end production use experience

Science Area	Number of Teams	Codes	Struct Grids	Unstruct Grids	Dense Matrix	Sparse Matrix	N-Body	Monte Carlo	FFT	PIC	Sig I/O
Climate and Weather	3	CESM, GCRM, CM1/WRF, HOMME	X	X		X		X			X
Plasmas/ Magnetosphere	2	H3D(M),VPIC, OSIRIS, Magtail/UPIC	X				X		X		X
Stellar Atmospheres and Supernovae	5	PPM, MAESTRO, CASTRO, SEDONA, ChaNGa, MS-FLUKSS	X			X	X	X		X	X
Cosmology	2	Enzo, pGADGET	X			X	X				
Combustion/ Turbulence	2	PSDNS, DISTUF	X						X		
General Relativity	2	Cactus, Harm3D, LazEV	X			X					
Molecular Dynamics	4	AMBER, Gromacs, NAMD , LAMMPS	X			X	X		X		
Quantum Chemistry	2	SIAL, GAMESS , NWChem			X	X	X	X			X
Material Science	3	NEMOS, OMEN, GW, QMCPACK			X	X	X	X			
Earthquakes/ Seismology	2	AWP-ODC , HERCULES, PLSQR , SPECFEM3D	X	X			X				X
Quantum Chromo Dynamics	1	Chroma , MILC, USQCD	X		X	X					
Social Networks	1	EPISIMDEMICS									
Evolution	1	Eve									
Engineering/System of Systems	1	GRIPS,Revisit						X			
Computer Science	1			X	X	X			X		X

Production Use Tests

from launch to finish, all I/O included

- **NAMD** - The 100M atom "chromataphore" benchmark was run for 60,000 time steps with Langevin dynamics and PME once every 4 steps. A 2-femtosecond time-step was used, with output of atom positions to DCD files with parallel writers.
- **Chroma** - Solution for all 12 spin-color components of the quark propagator. Two GPU Dirac equation solvers: (1) BiCGStab with algorithmic improvement to allow mixed precision, (2) a GCR algorithm with a domain-decomposed Additive-Schwarz solver. Lattice QCD parameters: grid size of $48^3 \times 512$ running at the physical values of the quark masses
- **QMCPACK** - Graphite 4x4x1 (256 electrons), VMC followed by DMC with 179,200 DMC Walkers. The scientific result of each run is an energy value with a computed error bar.
- **GAMESS** - A many-body expansion to estimate the full CCSD(T) correlation energy for a system of 32 water molecules by calculating 1-, 2-, and 3-body terms. The monomer, dimer and trimer CCSD(T) calculations are performed in parallel with 384 concurrent calculations.



Initial Performance Comparison

	NAMD	QMCPACK	Chroma	GAMESS
# of nodes used	768	700	768	1536
XK7 CPU-only time (sec)	11833.7	4477.0	1244.5	14637.5
XK7 CPU+GPU time (sec)	3484.5	908.3	320.2	4682.7
Ratio	3.4	4.9	3.9	3.1
XE6 time (sec)	6620.6	2452.4	1244.5	To be confirmed
XK7 time (sec)	3484.5	908.3	320.2	4682.7
Ratio	1.8	2.7	2.4	To be confirmed

Chroma GPU Work

	Calculation	Lines of Code	In main distribution?	Comments
Dslash	Application of the finite-difference operator to the 4-d lattice	2705 lines of CUDA code	Yes (in QUDA)	Memory bound. Utilizes L2 cache and texture cache for bandwidth aggregation.
BLAS Kernels	Addition and scaling of vectors	22 lines of CUDA code, 171 lines of C++ driver code	Yes (in QUDA)	Memory bound. Generic kernel with C++ functor approach fully defines any BLAS1 kernel with arbitrary precision
Reduction Kernels	Norm and dot product of vectors	127 lines of CUDA code, 369 of C++ driver code	Yes (in QUDA)	Memory and CPU-GPU latency bound. Generic kernel with C++ functor approach fully defines any reduction kernel with arbitrary precision.

LIBRARY ALGORITHMS

Scalable GPU Libraries

- Dense Linear algebra—BLAS, LU, Cholesky, Eigen solvers (CUBLAS, CULA, MAGMA)
- Sparse Matrix Vector Multiplication, Tridiagonal solvers (CUSPARSE, QUDA, ViennaCL, Parboil)
- FFTs, Convolutions (CUFFT, ViennaCL, Parboil)
- N-Body (NAMD/VMD, FMM BU, Parboil)
- Histograms (Parboil)
- Some PDE solvers (CURRENT, QUDA, Parboil)
- Graphs – Breadth-First Search (Parboil, CUSPARSE)
- Curve Fitting – Spline (Parboil)
- ...

Example of Library Needs

- Sparse linear algebra
 - Sparse LU, Cholesky factorization(?)
 - Sparse Eigen solvers
- Graph algorithms
 - Graph partitioning
 - Depth first search
 - ...
- ...

Algorithm Design Challenges

Parallelism

- Parallelism to fill growing HW parallelism

Data Scalability

- Operations should grow linearly with data size

Locality

- DRAM burst and cache space utilization

Regularity

- SIMD utilization and load balance

Numerical Stability

- Pivoting for linear system solvers



Example: Tridiagonal Solver

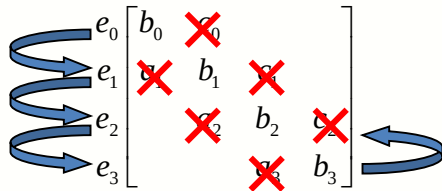
- Implicit finite difference methods, cubic spline interpolation, preconditioners
- An algorithm to find a solution of $\mathbf{Ax} = \mathbf{d}$, where A is an n -by- n tridiagonal matrix and d is an n -element vector

$$A = \begin{bmatrix} b_1 & c_1 & & \\ a_2 & b_2 & c_2 & 0 \\ & \ddots & \ddots & \ddots \\ 0 & a_{n-1} & b_{n-1} & c_{n-1} \\ & & a_n & b_n \end{bmatrix} \quad d = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_{n-1} \\ d_n \end{bmatrix}$$

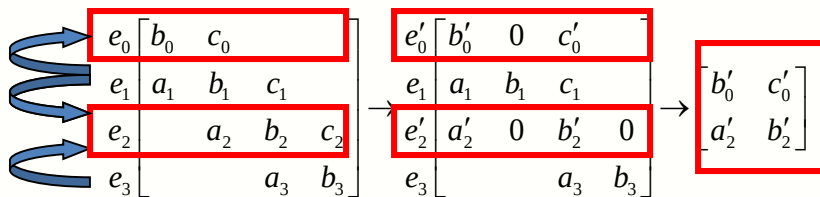
GPU Tridiagonal System Solver

Case Study

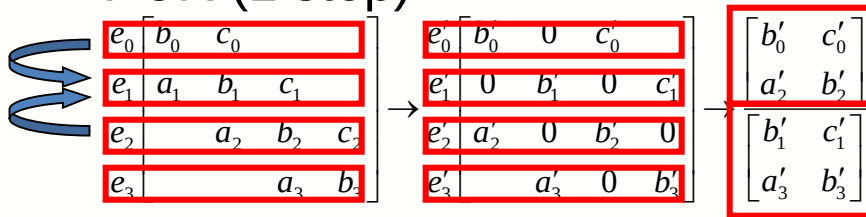
- Thomas (sequential)



- Cyclic Reduction (1 step)



- PCR (1 step)



- Hybrid Methods

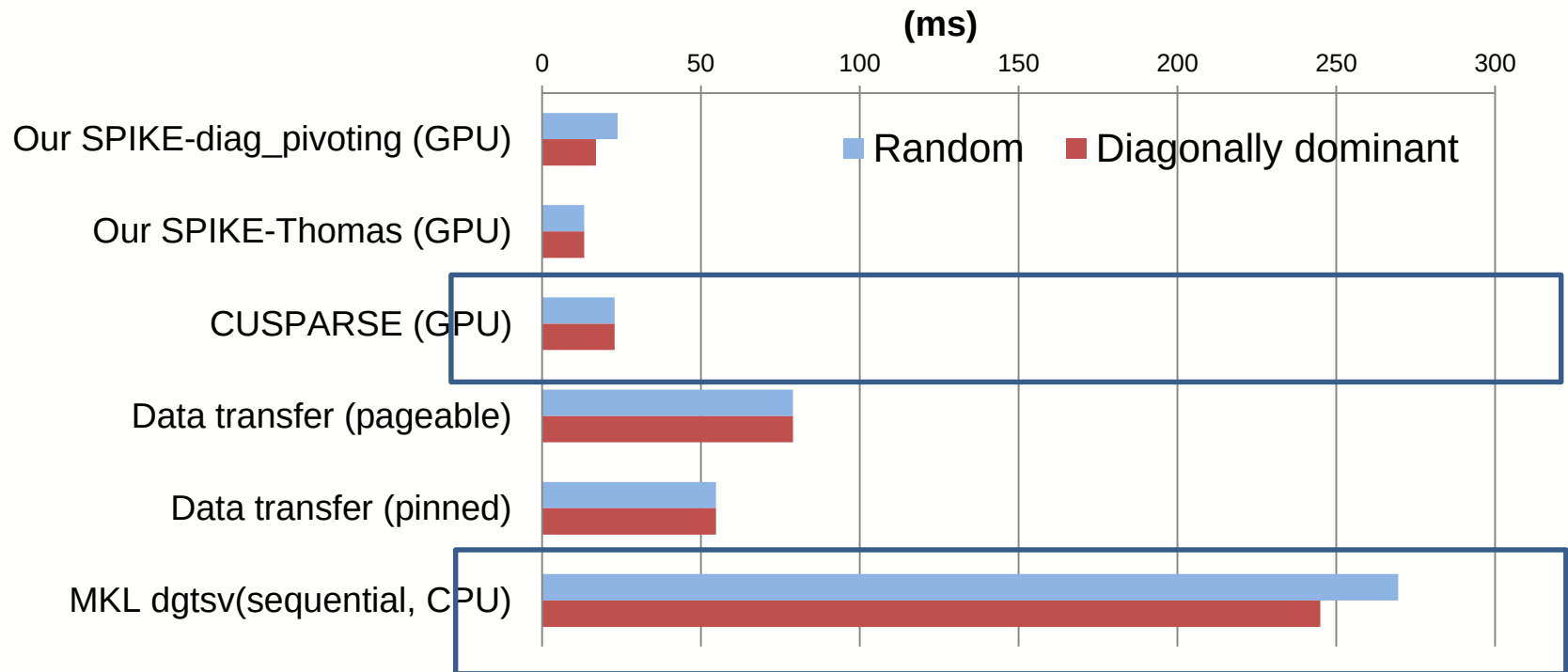
- PCR-Thomas (Kim 2011, Davidson 2011)
- PCR-CR (CUSPARSE 2012)
- Etc.

- CUSPARSE is supported by NVIDIA

Interleaved layout after partitioning

GPU Performance Advantage

Runtime of solving an 8M-row matrix



Numerical Error and Stability

Relative Backward Error

Matrix type	SPIKE-diag_pivoting	SPIKE-Thoma	CUSPARSE	MKL	ntel SPIKE	Matlab
1	1.82E-14	1.97E-14	7.14E-12	1.88E-14	1.39E-15	1.96E-14
2	1.27E-16	1.27E-16	1.69E-16	1.03E-16	1.02E-16	1.03E-16
3	1.55E-16	1.52E-16	2.57E-16	1.35E-16	1.29E-16	1.35E-16
4	1.37E-14	1.22E-14	1.39E-12	3.10E-15	1.69E-15	2.78E-15
5	1.07E-14	1.13E-14	1.82E-14	1.56E-14	4.62E-15	2.93E-14
6	1.05E-16	1.06E-16	1.57E-16	9.34E-17	9.51E-17	9.34E-17
7	2.42E-16	2.46E-16	5.13E-16	2.52E-16	2.55E-16	2.27E-16
8	2.14E-04	2.14E-04	1.50E+10	3.76E-04	2.32E-16	2.14E-04
9	2.32E-05	3.90E-04	1.93E+08	3.15E-05	9.07E-16	1.19E-05
10	4.27E-05	4.83E-05	2.74E+05	3.21E-05	4.72E-16	3.21E-05
11	7.52E-04	6.59E-02	4.54E+11	2.99E-04	2.20E-15	2.28E-04
12	5.58E-05	7.95E-05	5.55E-04	2.24E-05	5.52E-05	2.24E-05
13	5.51E-01	5.45E-01	1.12E+16	3.34E-01	3.92E-15	3.08E-01
14	2.86E+49	4.49E+49	2.92E+51	1.77E+48	3.86E+54	1.77E+48
15	2.09E+60	Nan	Nan	1.47E+59	Fail	3.69E+58
16	Inf	Nan	Nan	Inf	Fail	4.68E+171

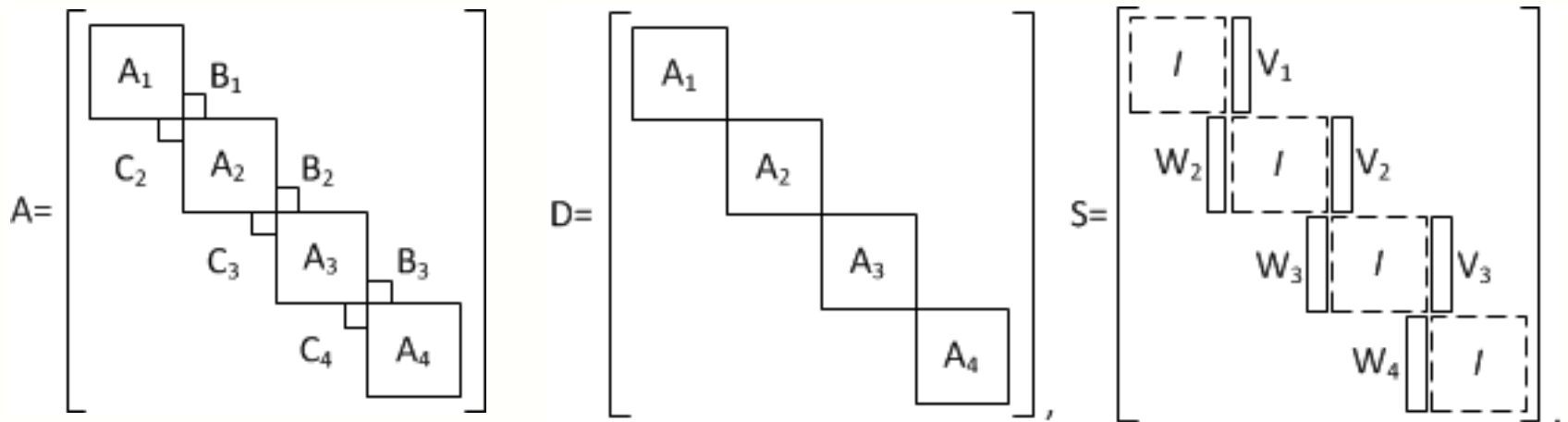
SPIKE Algorithm

- SPIKE algorithm decomposes a tridiagonal matrix A into several blocks

$$A = \begin{bmatrix} A_1 & B_1 & & \\ C_2 & A_2 & B_2 & \\ & C_3 & A_3 & B_3 \\ & & C_4 & A_4 \end{bmatrix}, \quad X = \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix}, \quad F = \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{bmatrix}.$$

SPIKE Algorithm

- A can be factored as $A = DS$

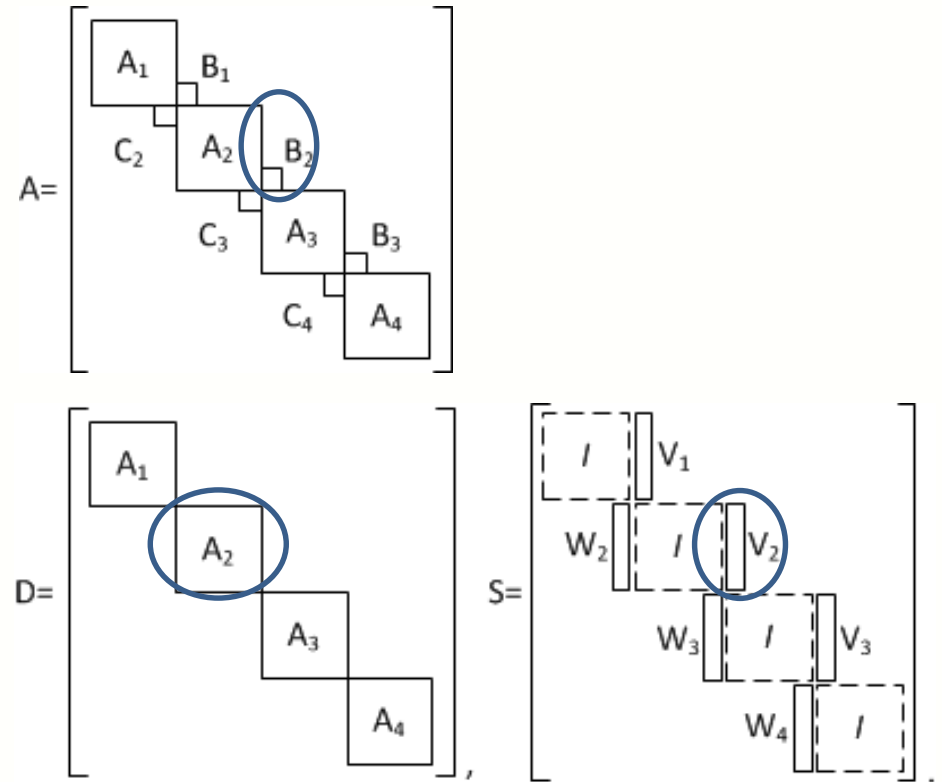


- $AX = D SX = F$ can be solved by solving $DY = F$ (parallel), and $SX = Y$

SPIKE Algorithm

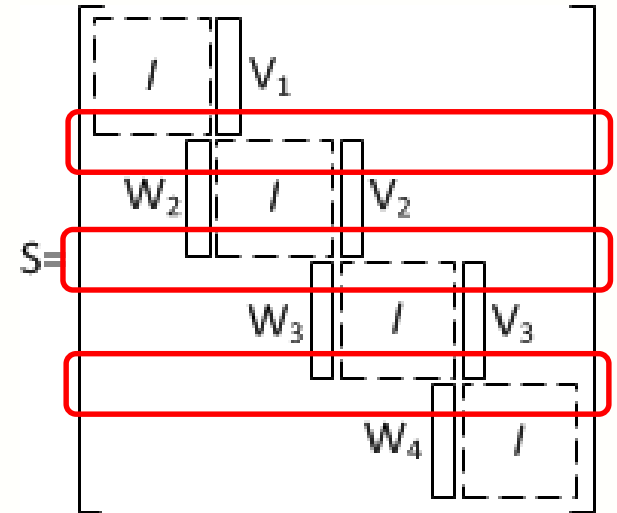
- How to build S?

$$\textcircled{A_i} V_i = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ B_i \end{bmatrix} \quad \textcircled{A_i} W_i = \begin{bmatrix} C_i \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$



SPIKE Algorithm

- How to solve $SX = Y$?
 - Solving the collection of the first and latest rows in all blocks
 - Backward substitution



$$\begin{bmatrix}
 1 & v_{1L} & & & \\
 w_{21} & 1 & 0 & v_{21} & \\
 w_{2L} & 0 & 1 & v_{2L} & \\
 & & w_{31} & 1 & 0 & v_{31} \\
 & & w_{3L} & 0 & 1 & v_{3L} \\
 & & & & w_{41} & 1
 \end{bmatrix}$$

*E. Polizzi and A. H. Sameh, "A parallel hybrid banded system solver: The SPIKE algorithm," Parallel Computing, vol. 32, no. 2, pp. 177–194, 2006.

Diagonal Pivoting

- Solving each block A_i in a numerically stable way
- A tridiagonal matrix A can be decomposed to LBM^T
 - B is a block diagonal matrix with 1-by-1 or 2-by-2 blocks

*J. B. Erway, R. F. Marcia, and J. Tyson, “Generalized diagonal pivoting methods for tridiagonal systems without interchanges,” IAENG International Journal of Applied Mathematics, vol. 4, no. 40, pp. 269–275, 2010.



LBM^T Decomposition for Pivoting

$$A = \begin{bmatrix} \boxed{b_1} & c_1 & 0 & \cdots & 0 \\ a_2 & \boxed{b_2} & c_2 & \ddots & \vdots \\ 0 & a_3 & b_3 & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & c_{n-1} \\ 0 & \cdots & 0 & a_n & b_n \end{bmatrix}$$

$$A = \underbrace{\begin{bmatrix} B_d & T_{12}^T \\ T_{21} & T_{22} \end{bmatrix}}_L = \begin{bmatrix} I_d & 0 \\ T_{21}B_d^{-1} & I_{n-d} \end{bmatrix} \underbrace{\begin{bmatrix} B_d & 0 \\ 0 & \boxed{A_s} \end{bmatrix}}_{M^T} \begin{bmatrix} I_d & B_d^{-1}T_{12}^T \\ 0 & I_{n-d} \end{bmatrix}$$

- L and M are unit lower triangular matrices
- B_d is 1-by-1 or 2-by-2 block
- A_s is also a tridiagonal matrix
 - A_s is updated by modifying leading elements of T₂₂
 - A_s can be decomposed **recursively**

$$A_s = T_{22} - T_{21}B_d^{-1}T_{12}^T$$

$$= \begin{cases} T_{22} - \frac{a_2c_1}{b_1}e_1^{(n-1)}e_1^{(n-1)T} & \text{if } d=1 \\ T_{22} - \frac{a_3b_1c_2}{\Delta}e_1^{(n-2)}e_1^{(n-2)T} & \text{if } d=2, \end{cases}$$

$$\Delta = b_1b_2 - a_2c_1$$

Pivoting Criteria

- Bunch-Kaufman algorithm for asymmetric case

$k = (\sqrt{5} - 1)$
 $\sigma = \max\{|c_1|, |a_2|, |b_2|, |c_2|, |a_3|\}$
if $|b_1|\sigma \geq k|c_1 a_2|$
 1-by-1 pivoting
else
 2-by-2 pivoting

$$\begin{bmatrix}
 \boxed{b_1} & \boxed{c_1} & 0 & \cdots & 0 \\
 \boxed{a_2} & \boxed{b_2} & \boxed{c_2} & \ddots & \vdots \\
 0 & \boxed{a_3} & \boxed{b_3} & \ddots & 0 \\
 \vdots & \ddots & \ddots & \ddots & c_{n-1} \\
 0 & \cdots & 0 & a_n & b_n
 \end{bmatrix}$$

An Example

$$\left[\begin{array}{cc|cc} 0 & 2 & 1 & 1 \\ 0.5 & 0 & 0 & 0 \\ & 1 & & \end{array} \right] \xrightarrow{d=2} \left[\begin{array}{cc|cc} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0.5 & 0 & 0 & 1 \end{array} \right] \left[\begin{array}{cc|cc} 0 & 2 & 0 & 1 \\ 0.5 & 0 & 1 & 0 \end{array} \right] \left[\begin{array}{cc|cc} 1 & 0 & 2 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \end{array} \right]$$

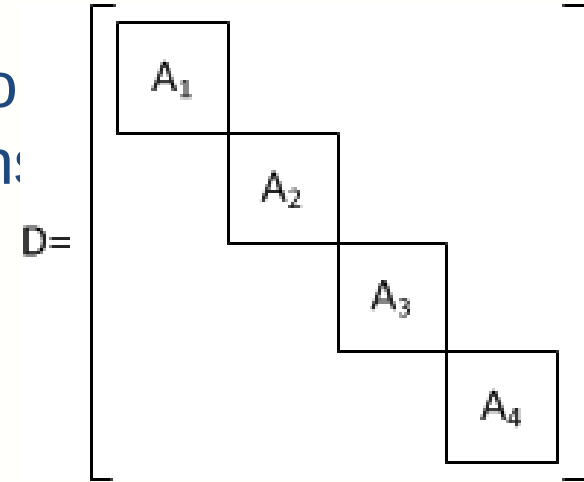
What we really store

$$A = \begin{bmatrix} 0 & 2 & 1 \\ 0.5 & 0 & 0 \\ 1 & 0 & 1 \\ & 1 & 0 \end{bmatrix}$$

condition = 2 0 2 0

Data Layout

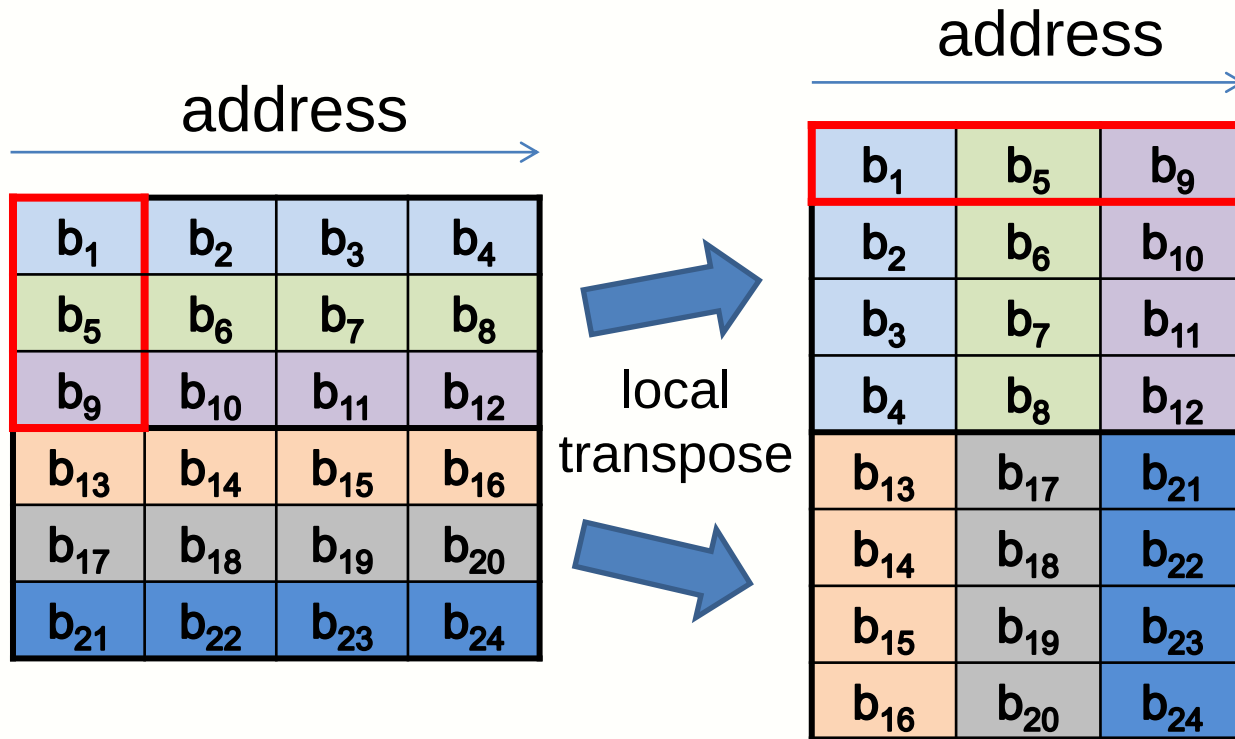
- Observation
 - GPUs require consecutive threads to access consecutive memory locations to fully utilize DRAM burst
- However, in SPIKE
 - Consecutive elements in a diagonal are stored in consecutive memory in gtsv interface
 - Each block is processed by one thread
 - Consecutive threads access locations that are many elements away



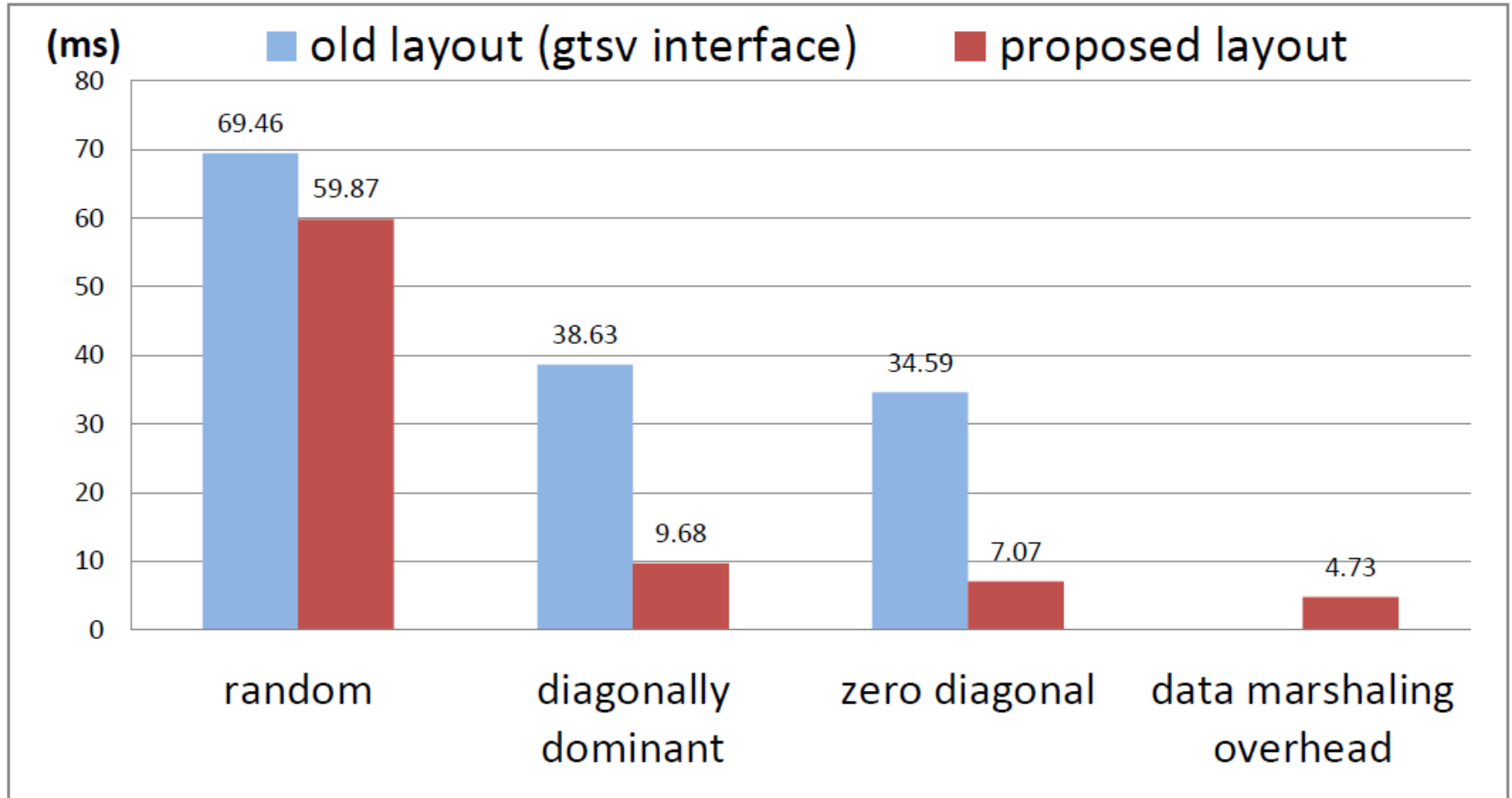
Data Layout Transformation

- Local transpose
 - b_i 's are elements in a diagonal
 - 6 4-elements blocks (block in SPIKE)

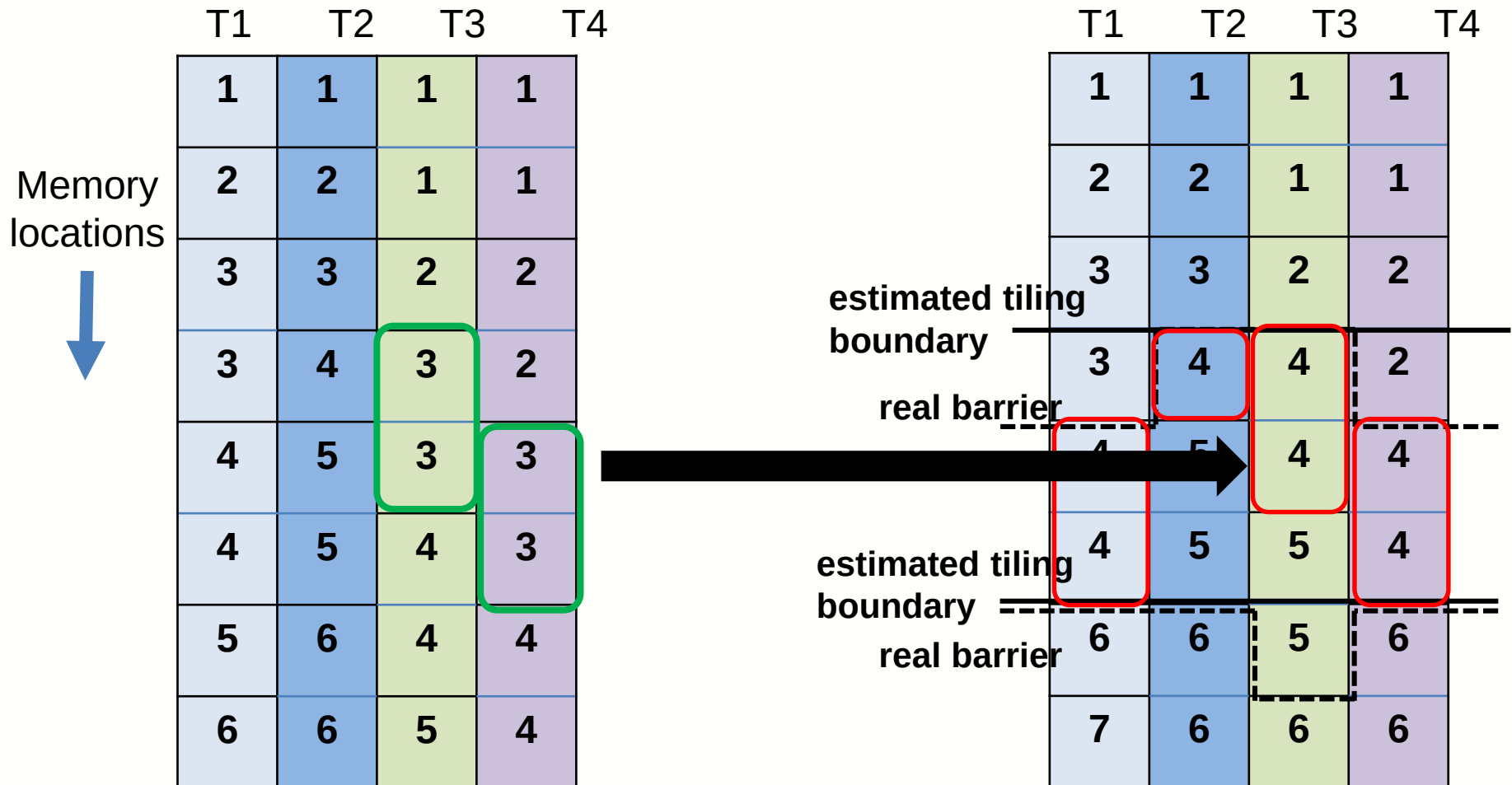
$$D = \begin{bmatrix} A_1 & & & \\ & A_2 & & \\ & & A_3 & \\ & & & A_4 \end{bmatrix}$$



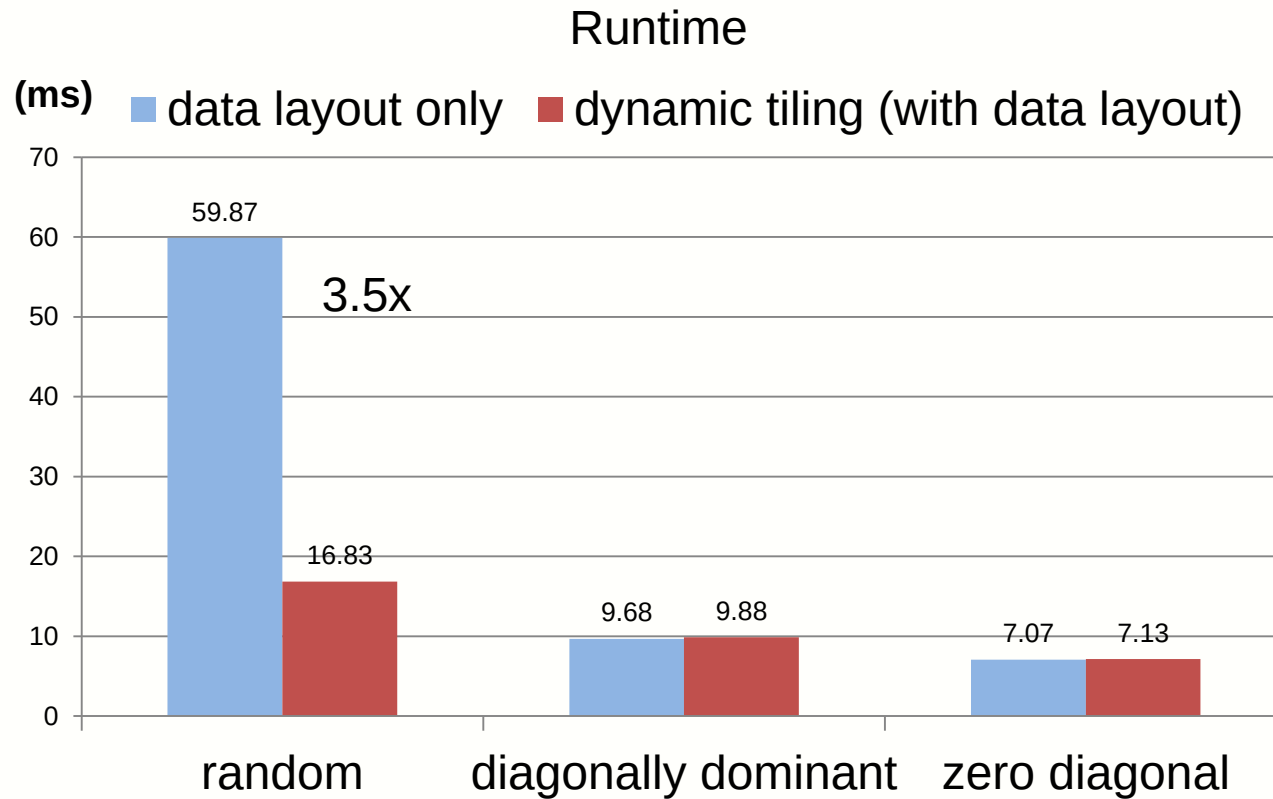
Fast Transposition on GPU



Dynamic Tiling



Dynamic Tiling



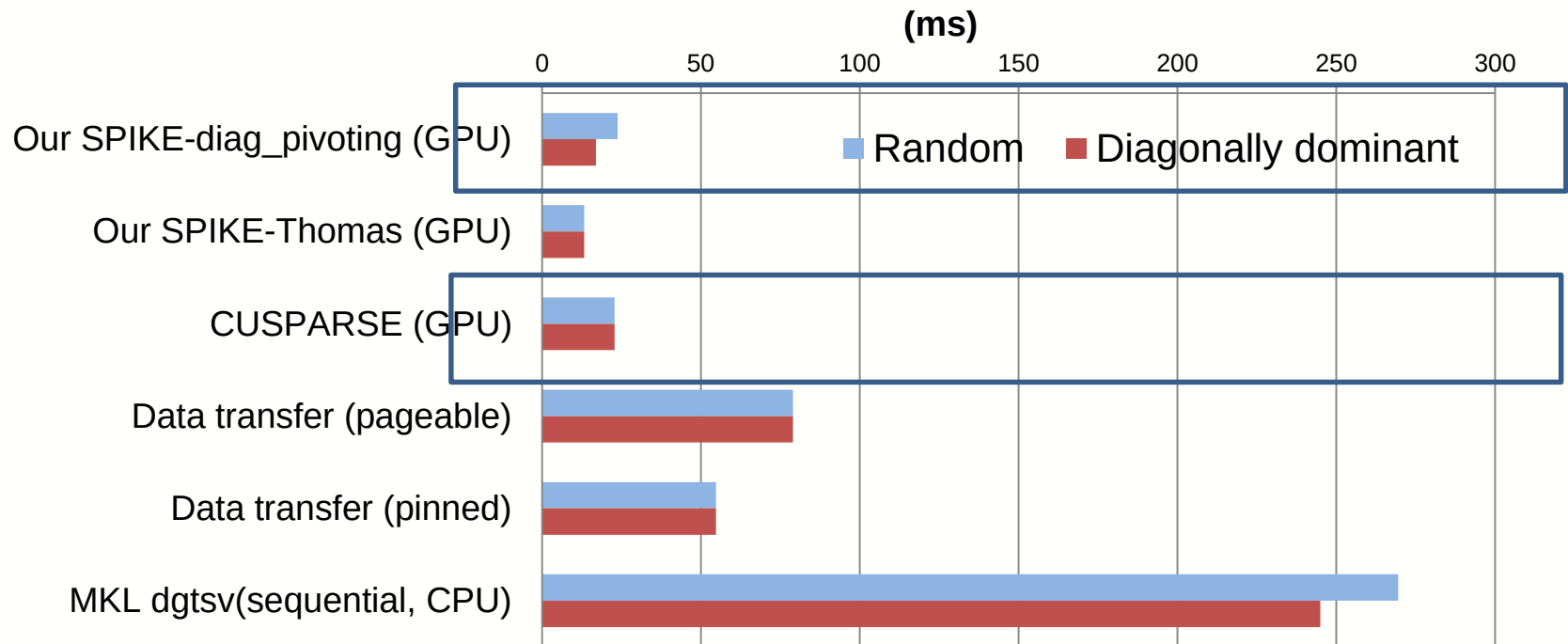
Numerical Error and Stability

Relative Backward Error

Matrix type	SPIKE-diag_pivoting	SPIKE-Thomas	CUSPARSE	MKL	intel SPIKE	Matlab
1	1.82E-14	1.97E-14	7.14E-12	1.88E-14	1.39E-15	1.96E-14
2	1.27E-16	1.27E-16	1.69E-16	1.03E-16	1.02E-16	1.03E-16
3	1.55E-16	1.52E-16	2.57E-16	1.35E-16	1.29E-16	1.35E-16
4	1.37E-14	1.22E-14	1.39E-12	3.10E-15	1.69E-15	2.78E-15
5	1.07E-14	1.13E-14	1.82E-14	1.56E-14	4.62E-15	2.93E-14
6	1.05E-16	1.06E-16	1.57E-16	9.34E-17	9.51E-17	9.34E-17
7	2.42E-16	2.46E-16	5.13E-16	2.52E-16	2.55E-16	2.27E-16
8	2.14E-04	2.14E-04	1.50E+10	3.76E-04	2.32E-16	2.14E-04
9	2.32E-05	3.90E-04	1.93E+08	3.15E-05	9.07E-16	1.19E-05
10	4.27E-05	4.83E-05	2.74E+05	3.21E-05	4.72E-16	3.21E-05
11	7.52E-04	6.59E-02	4.54E+11	2.99E-04	2.20E-15	2.28E-04
12	5.58E-05	7.95E-05	5.55E-04	2.24E-05	5.52E-05	2.24E-05
13	5.51E-01	5.45E-01	1.12E+16	3.34E-01	3.92E-15	3.08E-01
14	2.86E+49	4.49E+49	2.92E+51	1.77E+48	3.86E+54	1.77E+48
15	2.09E+60	Nan	Nan	1.47E+59	Fail	3.69E+58
16	Inf	Nan	Nan	Inf	Fail	4.68E+171

GPU Performance Advantage

Runtime of solving an 8M matrix



Eight Categories of Scalability Algorithm Techniques

	Memory Bandwidth	Update Contention	Load Balance	Regularity	Efficiency
Scatter to Gather		X			
Privatization		X			
Tiling	X				X
Coarsening	X	X			X
Data Layout	X	X			X
Input Binning	X				X
Regularization			X	X	X
Compaction	X		X	X	X

Stratton, et al, IEEE Computer, 8/2012

An Opportunity of a Lifetime

- Scalable and portable software lasts through many hardware generations

Scalable algorithms and libraries could be the best legacy we can leave behind from this era

PROGRAMMING INTERFACES

and Tools



ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Hwu 2013

Levels of GPU Programming Interfaces

Prototype & in development

X10, Chapel, Nesl,
Delite, Triolet, Par4all...

Implementation manages GPU threading and synchronization invisibly to user

Next generation

OpenACC, C++AMP, Thrust, Bolt

Simplifies data movement and kernel launch

Same GPU execution model (but less boilerplate)

Current generation

CUDA, OpenCL, DirectCompute



CUDA Hard? Final MP Grade Distribution for Coursera HPP

students

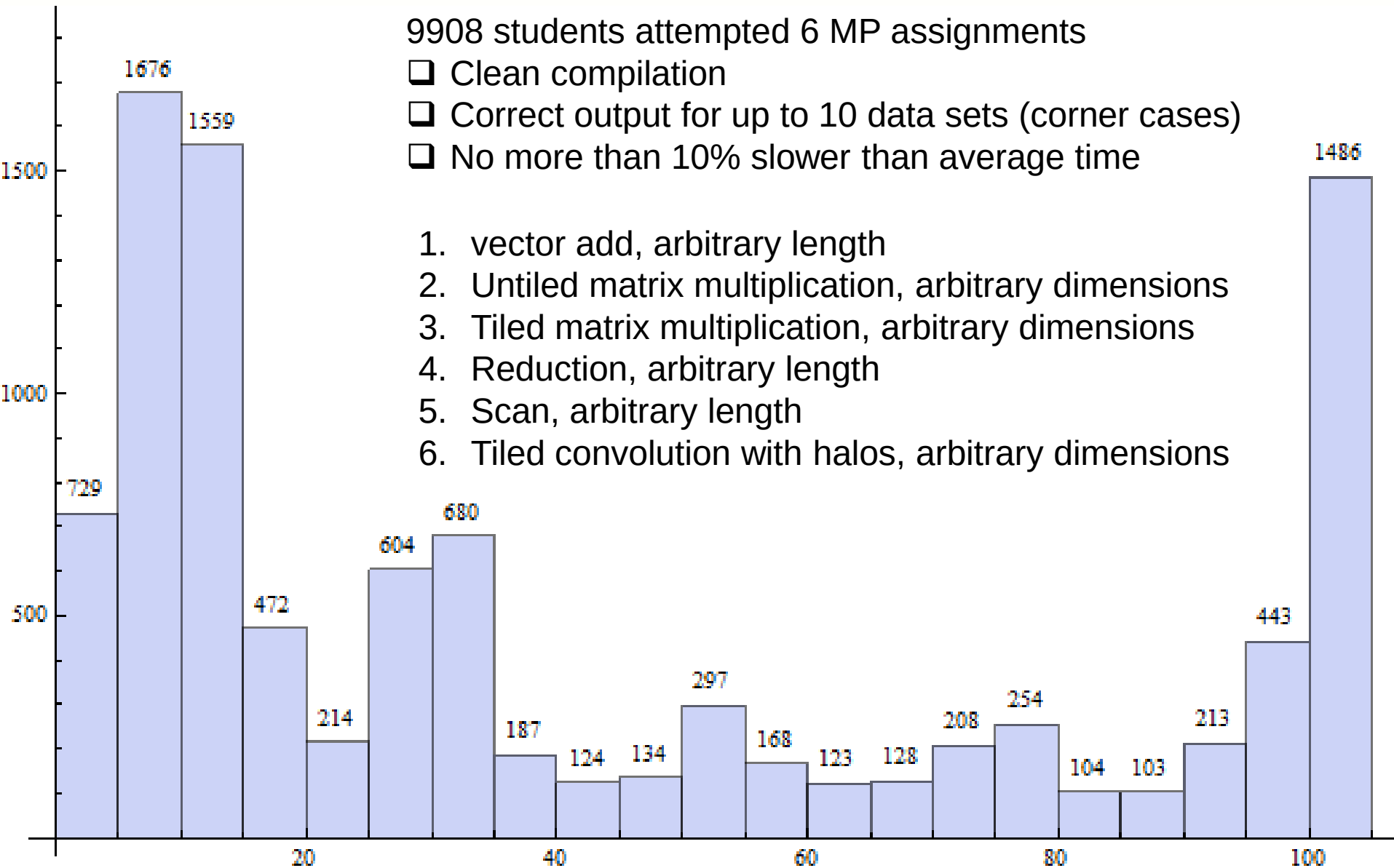
9908 students attempted 6 MP assignments

□ Clean compilation

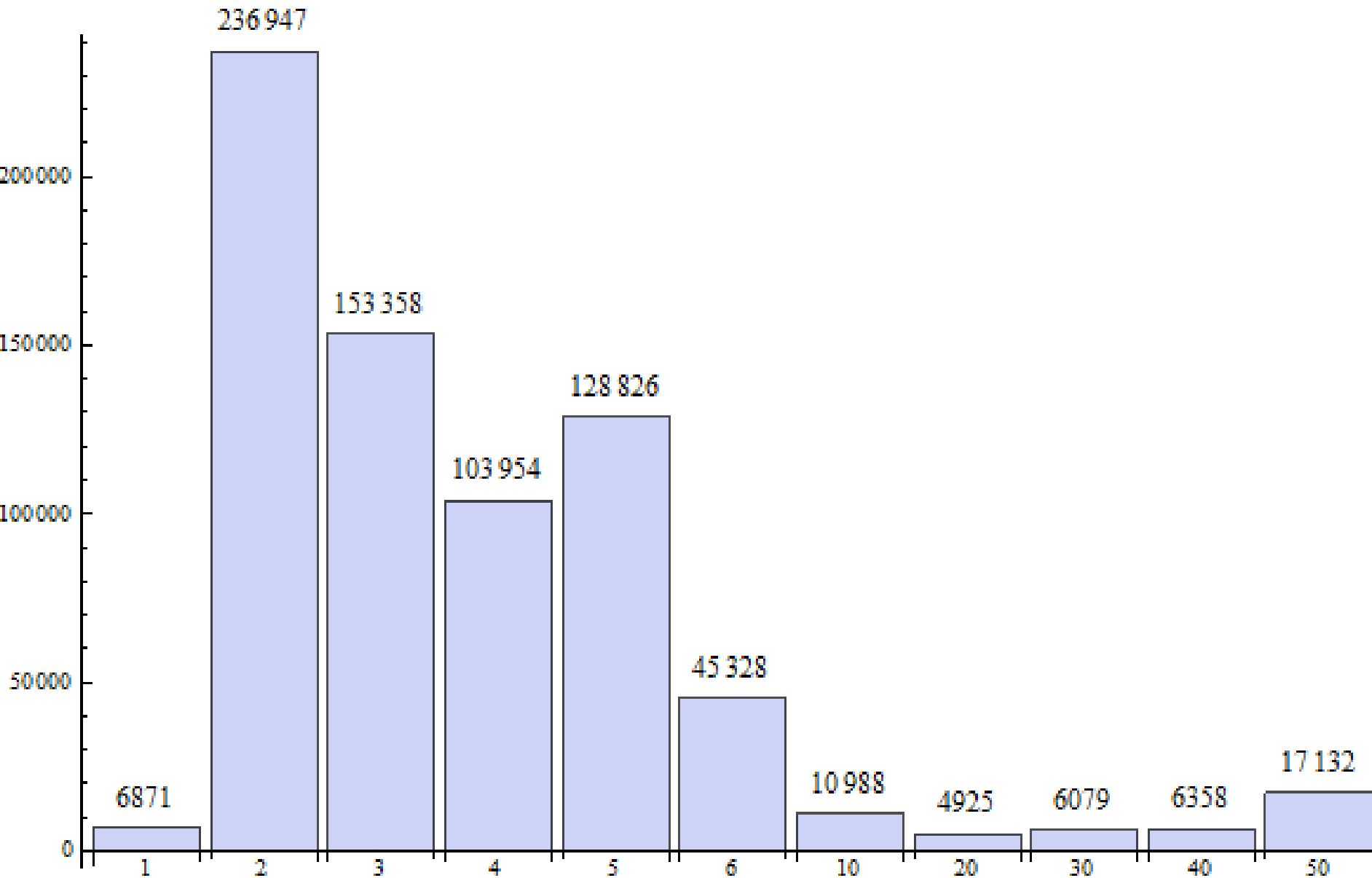
□ Correct output for up to 10 data sets (corner cases)

□ No more than 10% slower than average time

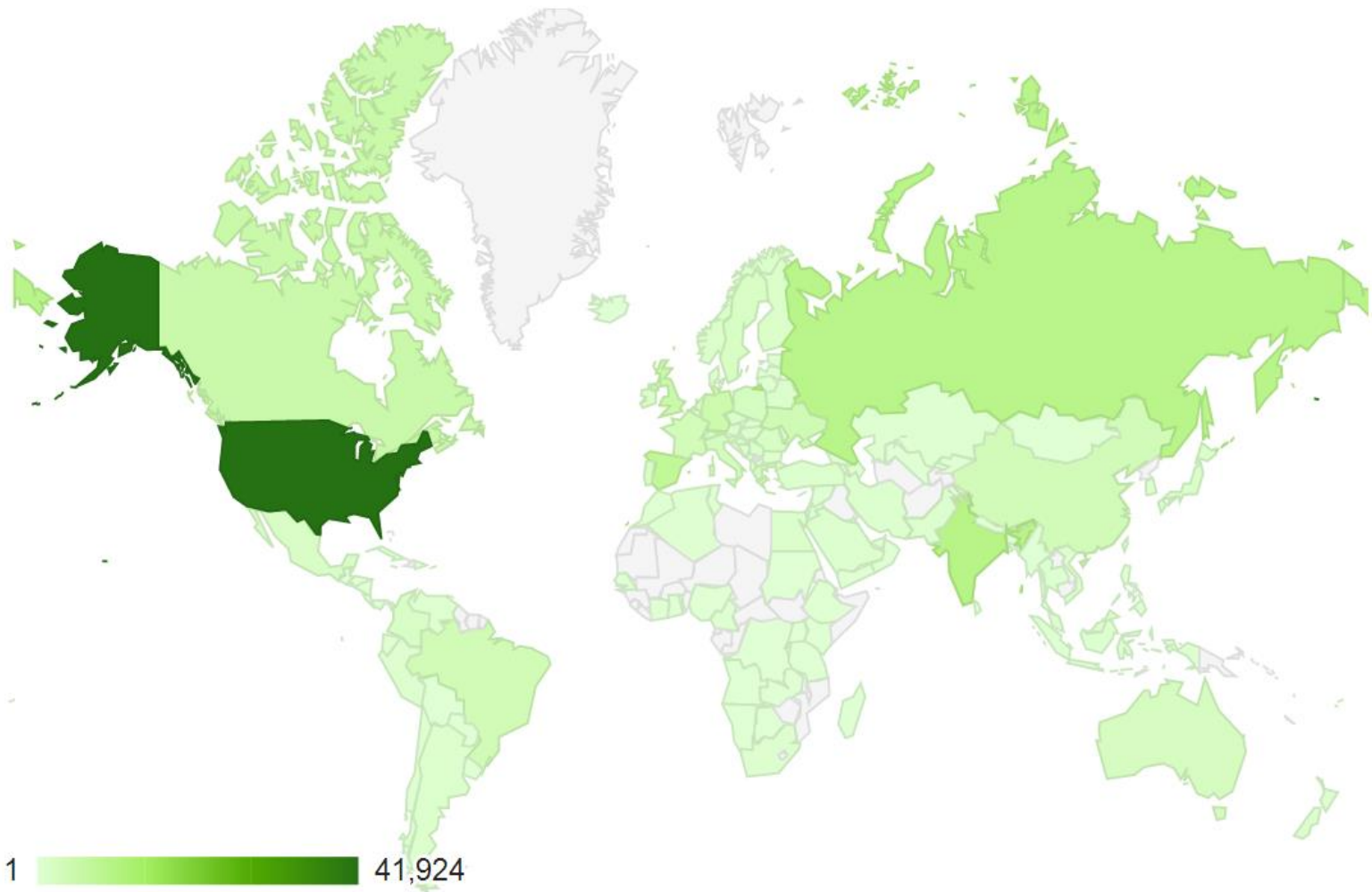
1. vector add, arbitrary length
2. Untiled matrix multiplication, arbitrary dimensions
3. Tiled matrix multiplication, arbitrary dimensions
4. Reduction, arbitrary length
5. Scan, arbitrary length
6. Tiled convolution with halos, arbitrary dimensions



Number of Compile-Runs for each MP



Visitors from each Country



Visits

150,194

% of Total: 100.00% (150,194)

Pages / Visit

4.43

Site Avg: 4.43 (0.00%)

Avg. Visit Duration

00:07:20

Site Avg: 00:07:20 (0.00%)

% New Visits

18.76%

Site Avg: 18.76% (0.00%)

Hwu 2013

Performance is currently hard to predict for high-level tools.

- High-level tools are making strides in usability, generality, and performance
- Performance sometimes comparable to low-level tools
- ...sometimes much worse

Nested data-parallel code vs. CUDA running on a GPU

	Code size (lines)		Run time (ms)		
	CUDA	Nesl	CUDA	Nesl	
Convex Hull	72	25	269	283	<i>On par</i>
Barnes-Hut	414	225	40	4502	<i>100 × slower</i>

Source: Bergstrom and Reppy,
ICFP 2012

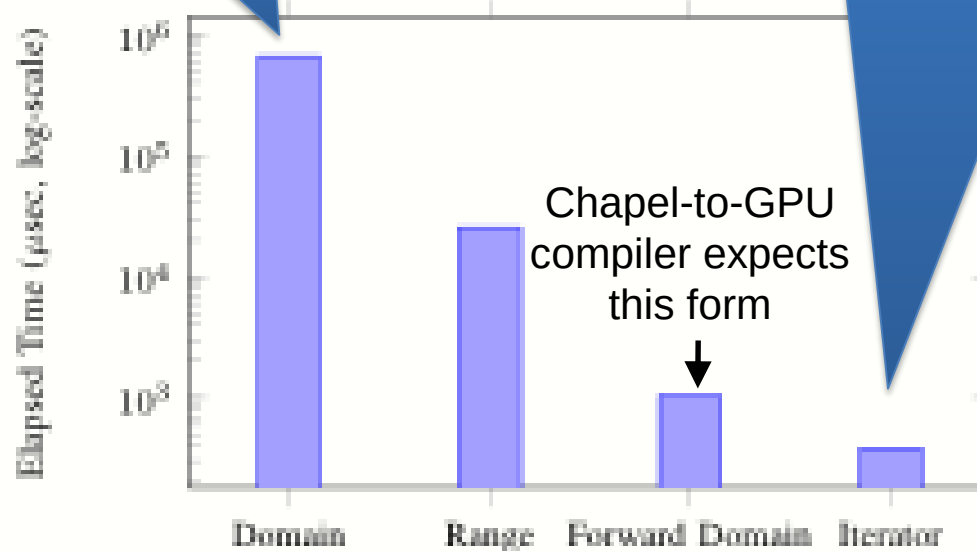
Huge overhead is often caused by the generality of the interface.

- Many performance pitfalls are fixable, but will still cause problems for novices

```
for j in [1..N] do ...;
```

```
iter myIter(min:int, max:int, step:int=1) {  
  while min <= max {  
    yield min;  
    min += step;  
  }  
}  
for j in myIter(1,N) do ...;
```

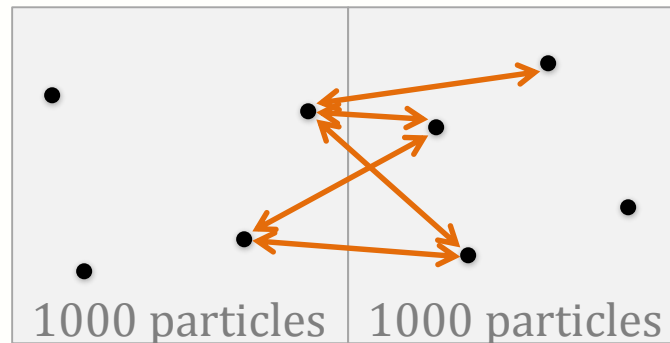
Time to execute a one-iteration loop on CPU in Chapel



Source: Dun and Taura,
IPDPSW 2012

Triolet Parallel Patterns

- Put patterns of work decomposition and communication into a reusable library
 - Also known as algorithmic skeletons
- Advanced compiler transformations



Building neighbor lists

Sequential C

Triolet

```
for (i = 0; i < M; i++)  
  for (j = 0; j < N; j++)  
    if (distance2(A[i], B[j]) < d)  
      C[k++] = (pair){i, j};
```

Sequential run time: **6.5ms**

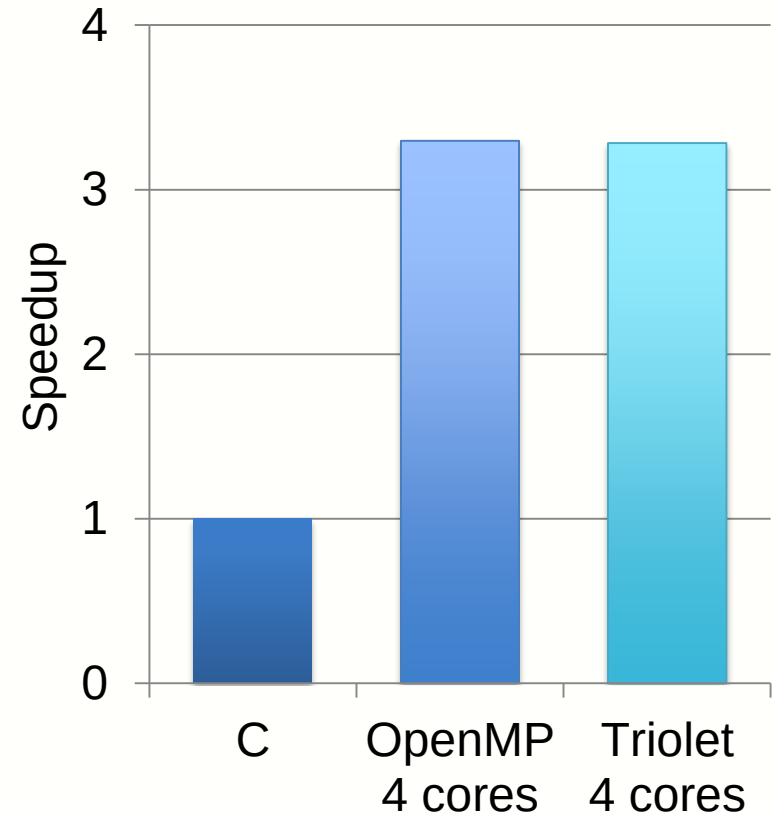
```
C = [(i, j) for (i, a) in enumerate(A)  
              for (j, b) in enumerate(B)  
                if distance2(a, b) < d]
```

Sequential run time: **9.2ms** (140% of C)

4-core parallel run time: **3.8ms** (58% of C)

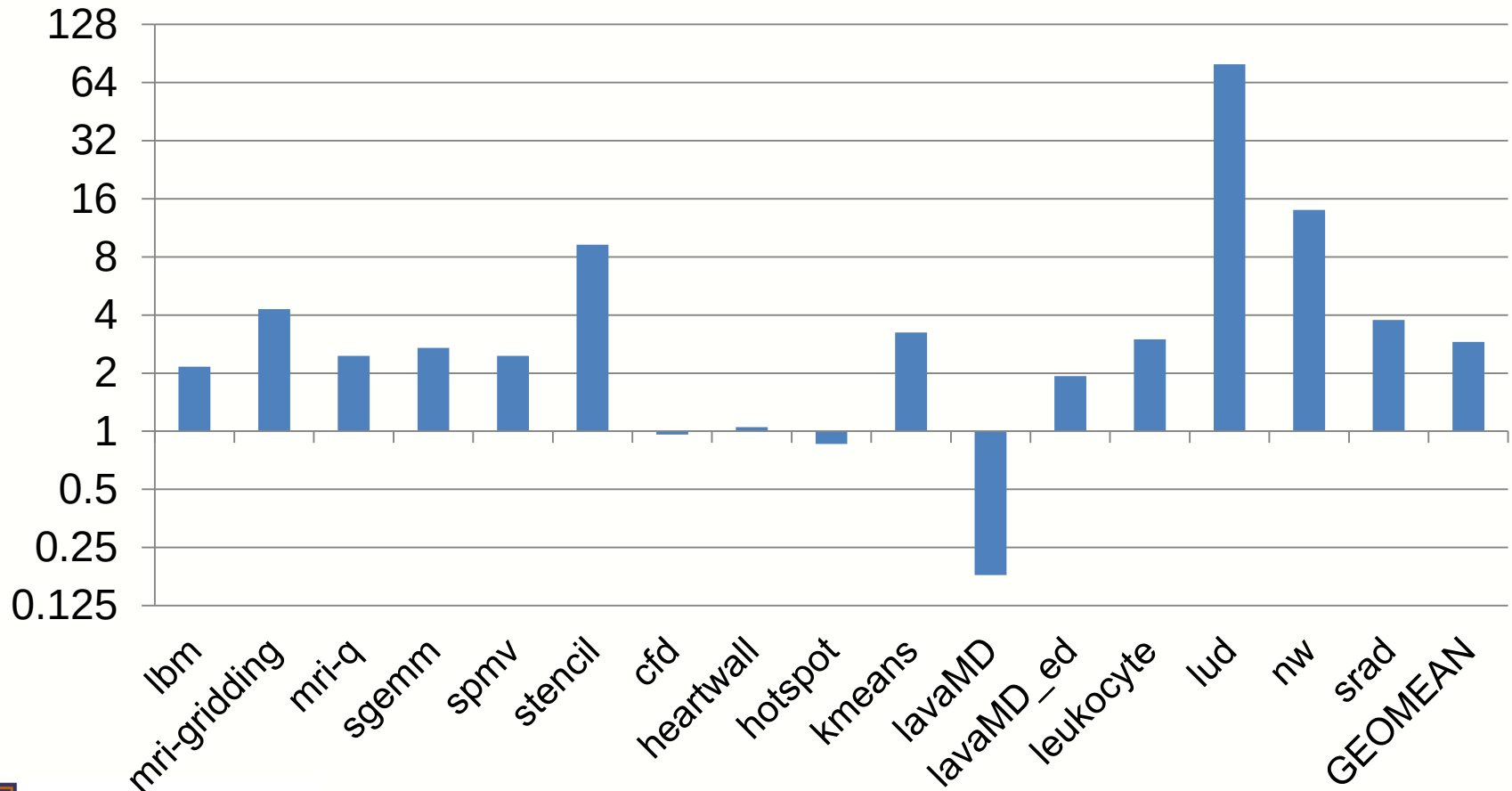
Work in Progress

- TPACF benchmark from Parboil
 - Astrophysics code
 - Generates a histogram showing correlation of two data sets
- Parallelizable with OpenMP
 - Give each thread a private histogram
 - Combine histograms at the end
- Triolet matches OpenMP performance with much less code
- However, steeper learning curve



Even low-level parallel code can be performance portable.

OpenCL-MxPA Speedup Over OpenMP on Multicore CPU



Conclusion and Outlook

- We have made much progress in GPU Computing
 - Solid applications and kernels
 - VPL OpenCL deployment for Olympics 2012
 - Experience with low-level interfaces
 - Coursera HPP with 900,000 captured student programs
 - Educated developers
- We will face more challenges
 - Energy efficiency vs. ease of programming
 - E.g., DARPA PERFECT10X10 ultra customized cores
 - Potential fragmentation of programming interfaces
 - Widening the set of algorithms and kernels

Acknowledgements

- D. August (Princeton), S. Baghsorkhi (Illinois), N. Bell (NVIDIA), I. Buck (NVIDIA), D. Callahan (Microsoft), A. Chien (Chicago), J. Cohen (NVIDIA), B. Colwell (DARPA), B. Dally (Stanford), J. Demmel (Berkeley), P. Dubey (Intel), M. Flynn (Stanford), M. Frank (Intel), M. Garland (NVIDIA), Isaac Gelado (BSC), B. Gropp (Illinois), Gschwind (IBM), R. Hank (Google), J. Hennessy (Stanford), P. Hanrahan (Stanford), M. Horowitz (Stanford), M. Houston (AMD), T. Huang (Illinois), R. Karp (Berkeley), D. Kaeli (NEU), K. Keutzer (Berkeley), D. Kirk (NVIDIA), D. Kuck (Intel), S. Mahlke (Michigan), T. Mattson (Intel), T. Mudge (Michigan), N. Navarro (UPC), J. Owens (Davis), D. Padua (Illinois), S. Patel (Illinois), Y. Patt (Texas), D. Patterson (Berkeley), C. Rodrigues (Illinois), S. Ryoo (ZeroSoft), K. Schulten (Illinois), B. Smith (Microsoft), M. Snir (Illinois), I. Sung (Illinois), P. Stenstrom (Chalmers), J. Stone (Illinois), S. Stone (Harvard), J. Stratton (Illinois), H. Takizawa (Tohoku), M. Valero (UPC)
- And many others!

There is always hope.

— Aragorn in the eve of the Battle of Pelennor
Minas Tirith

THANK YOU!

Research Opportunities

- Most important scalability goals are still achieved by hand today
 - Lack of effective compiler tools and generic building blocks
 - Too many code versions and too much programming effort
- Compiler transformations have been built on dependence analysis that require array accesses based on affine expression
 - Most real data structures do not have affine access expressions
 - Need programming interfaces to alleviate this
- Current directive-based runtime systems do not provide enough value



Top Five Working Environments

OS	Visits	Percentage
Windows	87,526	58.27%
Linux	29,761	19.81%
Macintosh	29,350	19.54%
iOS	1,866	0.92%
Android	1,387	0.06%



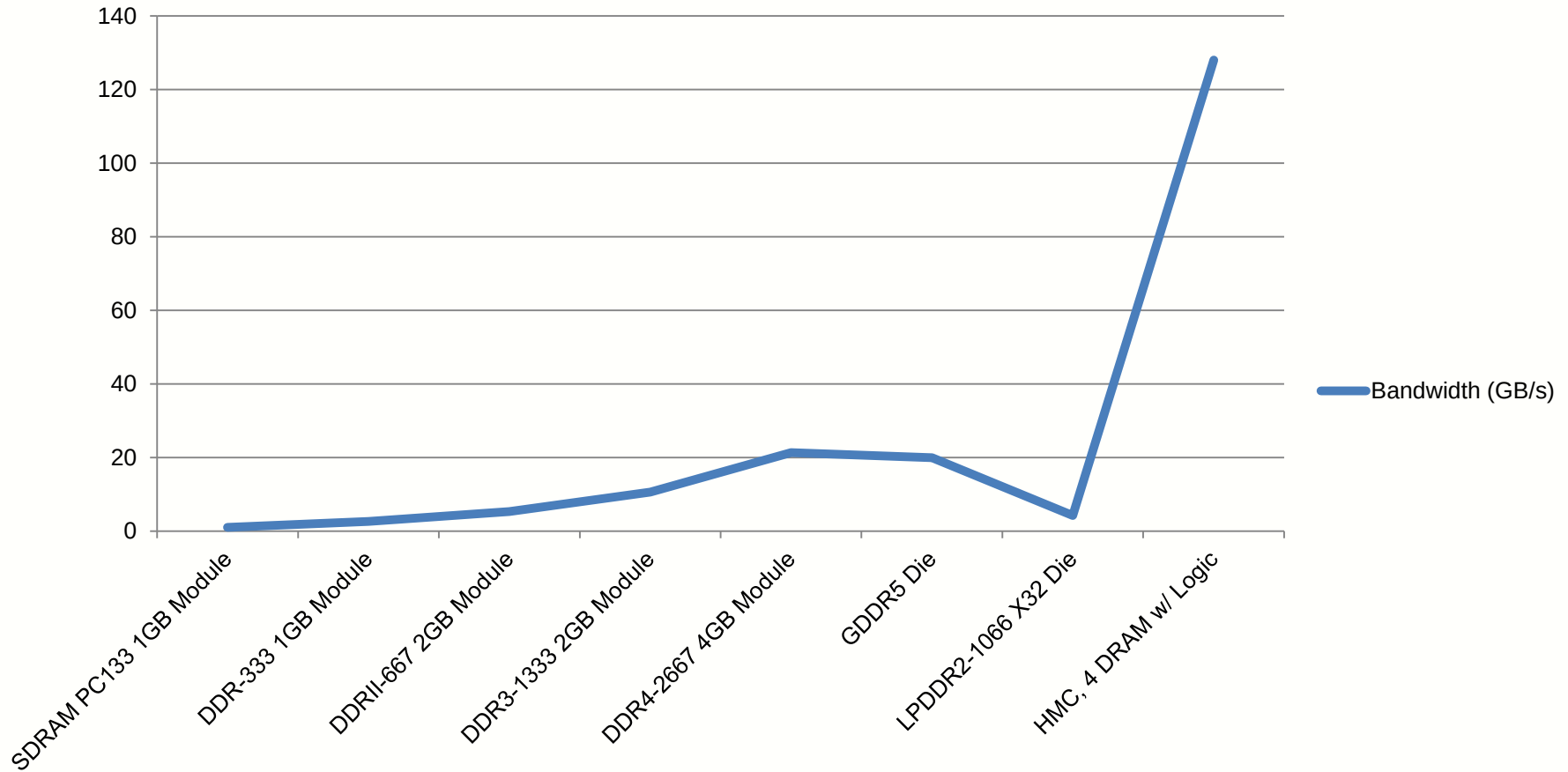
SOME IMPORTANT TRENDS

Exascale Energy Pressure

- Pressure for higher energy efficiency will likely make processors more difficult to program
 - More specialized processor data path (width, connectivity, etc.)
 - Wider SIMD
 - More system-level data movement control
 - Smaller on-chip storage per thread
 - ...

DRAM trends in bandwidth

Bandwidth in GB/s



More Optimization

