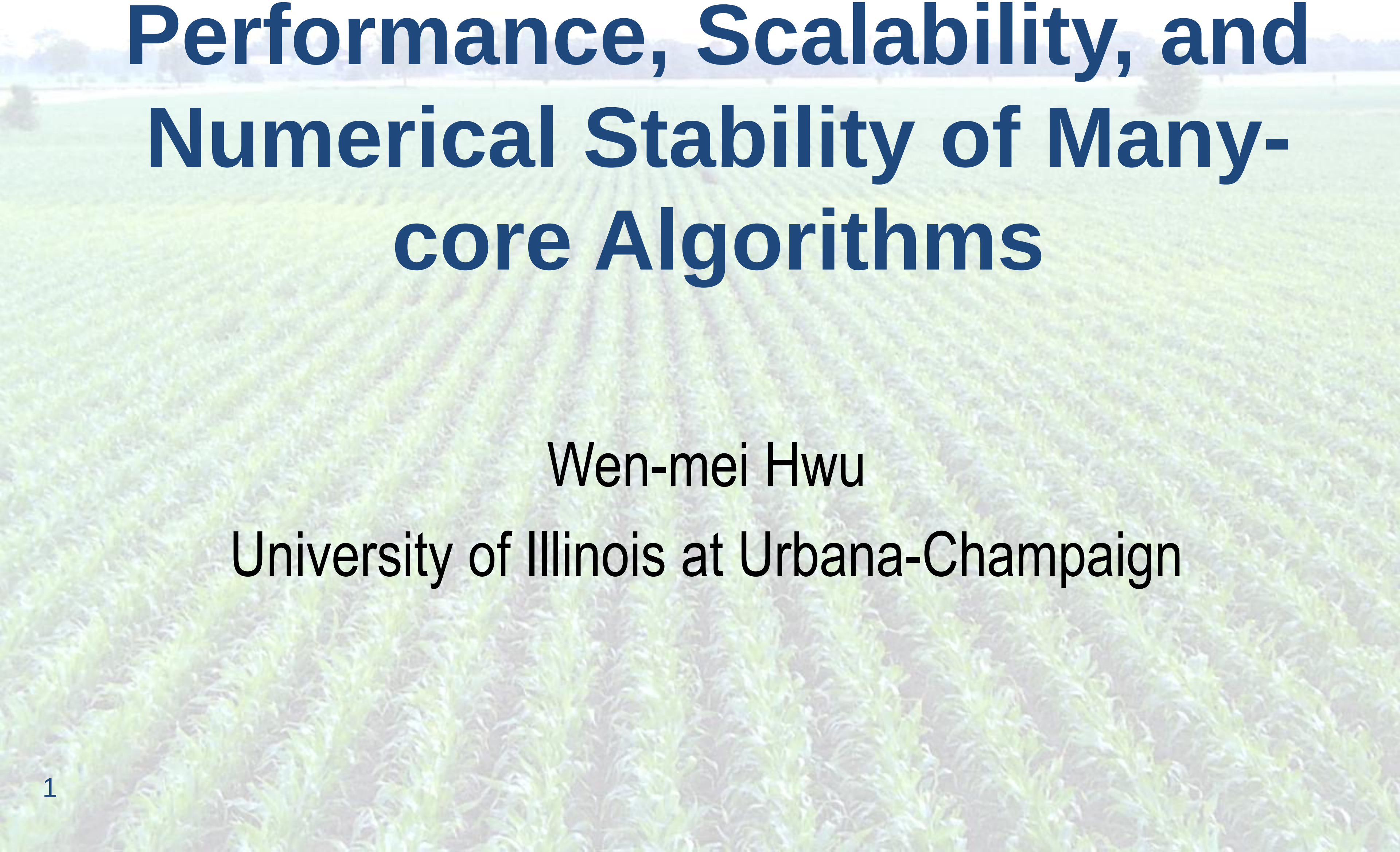
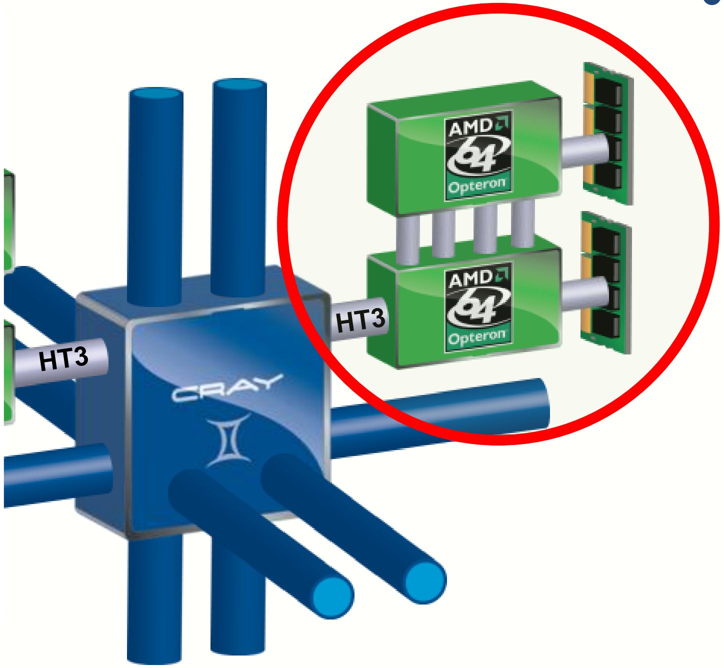
Performance, Scalability, and Numerical Stability of Many-core Algorithms

Wen-mei Hwu

University of Illinois at Urbana-Champaign



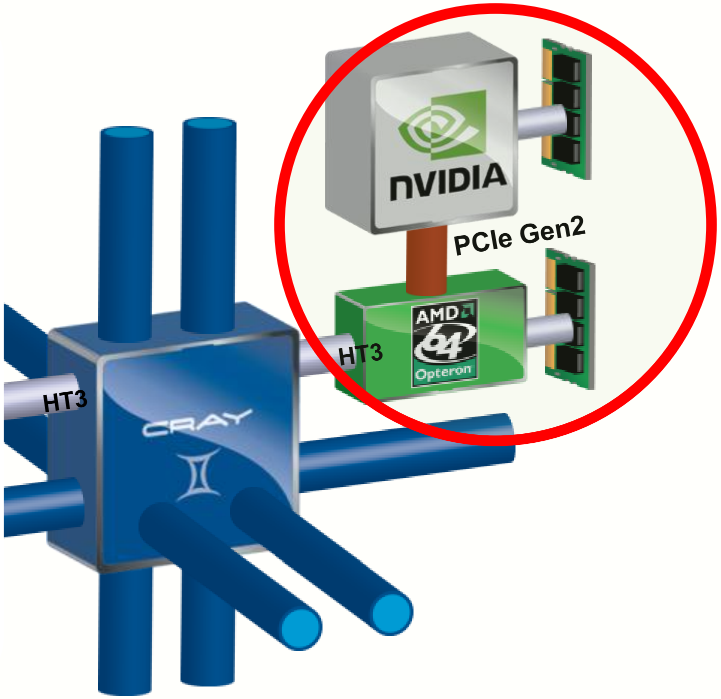
Cray XE6 Nodes



Blue Waters contains 22,640 Cray XE6 compute nodes.

- Dual-socket Node
  - Two AMD Interlagos chips
    - 16 core modules, 64 threads
    - 313 GFs peak performance
    - 64 GBs memory
      - 102 GB/sec memory bandwidth
  - Gemini Interconnect
    - Router chip & network interface
    - Injection Bandwidth (peak)
      - 9.6 GB/sec per direction

Cray XK7 Nodes



Blue Waters contains 3,072 Cray XK7 compute nodes.

- Dual-socket Node
  - One AMD Interlagos chip
    - 8 core modules, 32 threads
    - 156.5 GFs peak performance
    - 32 GBs memory
      - 51 GB/s bandwidth
  - One NVIDIA Kepler chip
    - 1.3 TFs peak performance
    - 6 GBs GDDR5 memory
      - 250 GB/sec bandwidth
  - Gemini Interconnect
    - Same as XE6 nodes

Initial Performance Results

- NAMD
  - 100 million atom benchmark with Langevin dynamics and PME once every 4 steps, from launch to finish, all I/O included
  - 768 nodes, Kepler+Interlagos is 3.9X faster over Interlagos-only
  - 768 nodes, XK7 is 1.8X XE6
- Chroma
  - Lattice QCD parameters: grid size of 48^3 x 512 running at the physical values of the quark masses
  - 768 nodes, Kepler+Interlagos is 4.9X faster over Interlagos-only
  - 768 nodes, XK7 is 2.4X XE6
- QMCPACK
  - Full run Graphite 4x4x1 (256 electrons), QMC followed by VMC
  - 700 nodes, Kepler+Interlagos is 4.9X faster over Interlagos-only
  - 700 nodes, XK7 is 2.7X XE6

Scalability vs. Numerical Stability
A Major Algorithm Design Challenge

Parallelism

- Parallelism to fill growing HW parallelism

Complexity and data scalability

- Operations should grown linearly with data size

Locality

- DRAM bursts and cache space utilization

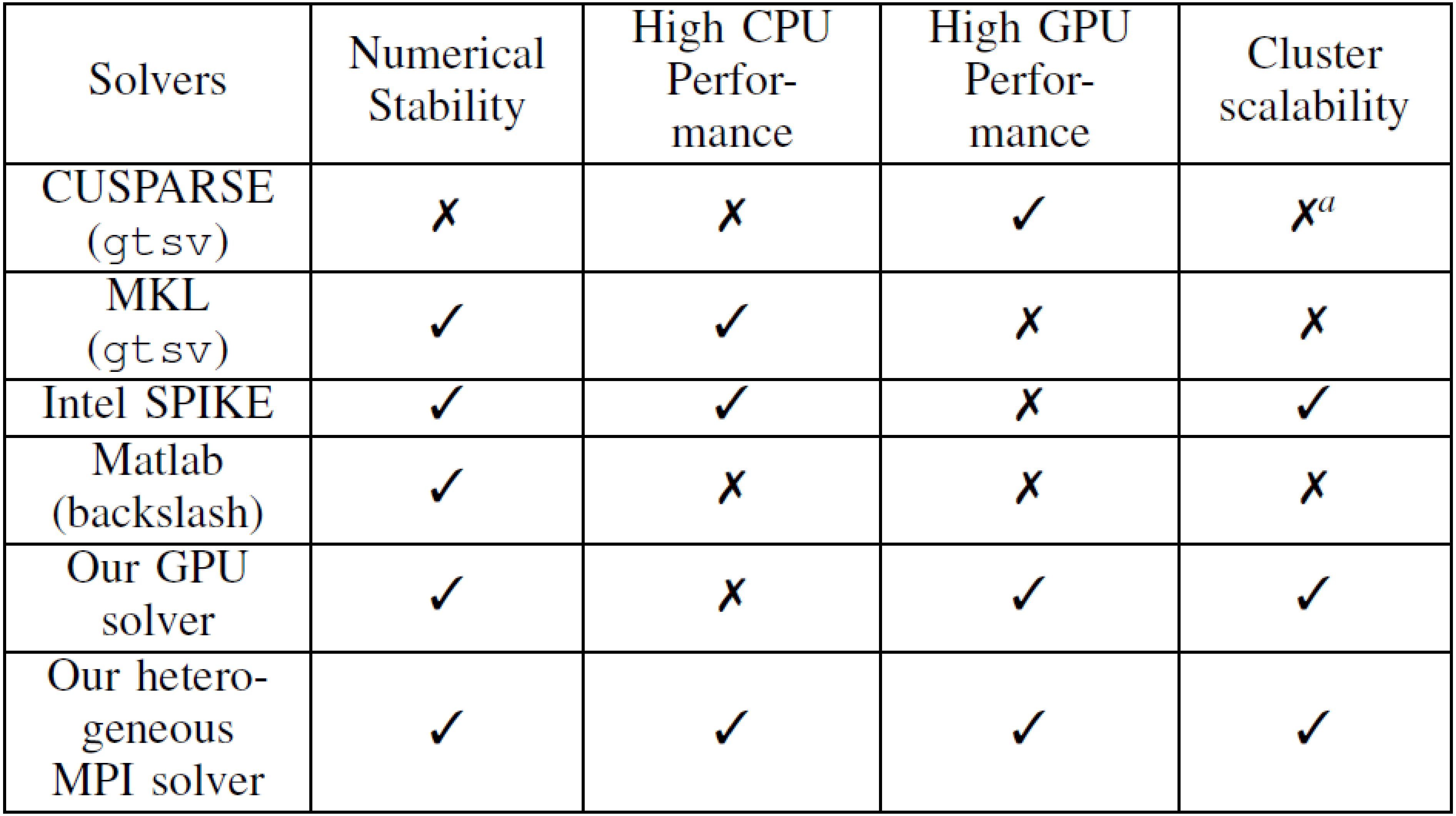
Regularity

- SIMD utilization and load balance

Numerical Stability

- Pivoting for linear system solvers

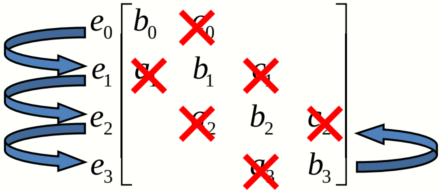
A Comparison of TDS on Major Platforms



Solvers	Numerical Stability	High CPU Performance	High GPU Performance	Cluster scalability
CUSPARSE (gtsv)	✗	✗	✓	✗[a]
MKL (gtsv)	✓	✓	✗	✗
Intel SPIKE	✓	✓	✗	✓
Matlab (backslash)	✓	✗	✗	✗
Our GPU solver	✓	✗	✓	✓
Our heterogeneous MPI solver	✓	✓	✓	✓

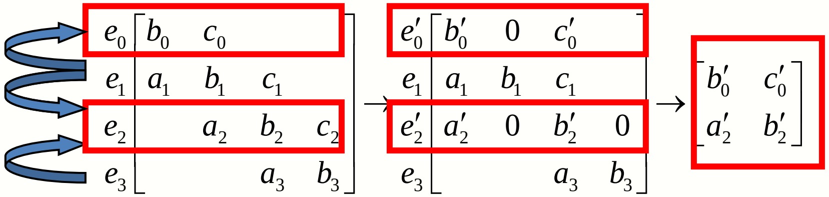
GPU Tridiagonal System Solver Case Study

- Thomas (sequential)



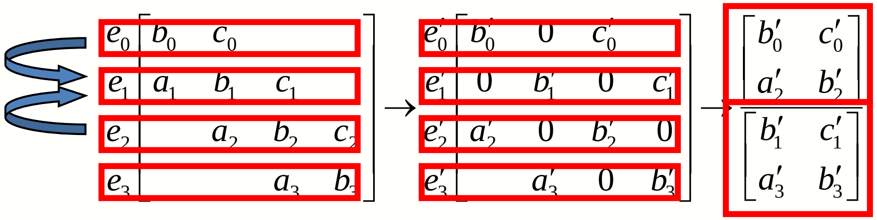
$$\begin{array}{c} e_0 \\ e_1 \\ e_2 \\ e_3 \end{array}\begin{bmatrix} b_0 & \cancel{c_0} & & \\ \cancel{a_1} & b_1 & \cancel{c_1} & \\ & \cancel{a_2} & b_2 & \cancel{c_2} \\ & & \cancel{a_3} & b_3 \end{bmatrix}$$

- Cyclic Reduction (1 step)



$$\begin{array}{c} e_0 \\ e_1 \\ e_2 \\ e_3 \end{array}\begin{bmatrix} b_0 & c_0 & & \\ a_1 & b_1 & c_1 & \\ & a_2 & b_2 & c_2 \\ & & a_3 & b_3 \end{bmatrix} \rightarrow \begin{array}{c} e_0' \\ e_1 \\ e_2' \\ e_3 \end{array}\begin{bmatrix} b_0' & 0 & c_0' & \\ a_1 & b_1 & c_1 & \\ a_2' & 0 & b_2' & 0 \\ & & a_3 & b_3 \end{bmatrix} \rightarrow \begin{bmatrix} b_0' & c_0' \\ a_2' & b_2' \end{bmatrix}$$

- PCR (1 step)



$$\begin{array}{c} e_0 \\ e_1 \\ e_2 \\ e_3 \end{array}\begin{bmatrix} b_0 & c_0 & & \\ a_1 & b_1 & c_1 & \\ & a_2 & b_2 & c_2 \\ & & a_3 & b_3 \end{bmatrix} \rightarrow \begin{array}{c} e_0' \\ e_1' \\ e_2' \\ e_3' \end{array}\begin{bmatrix} b_0' & 0 & c_0' & \\ 0 & b_1' & 0 & c_1' \\ a_2' & 0 & b_2' & 0 \\ & a_3' & 0 & b_3' \end{bmatrix} \rightarrow \begin{matrix} \begin{bmatrix} b_0' & c_0' \\ a_2' & b_2' \end{bmatrix} \\ \begin{bmatrix} b_1' & c_1' \\ a_3' & b_3' \end{bmatrix} \end{matrix}$$

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

- Numerically unstable

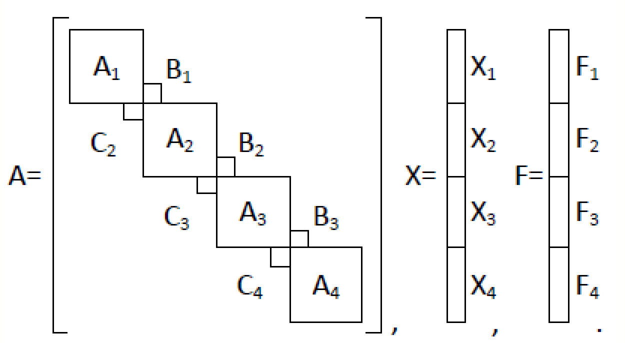
Pivoting

- Judiciously swap rows to avoid bad cases

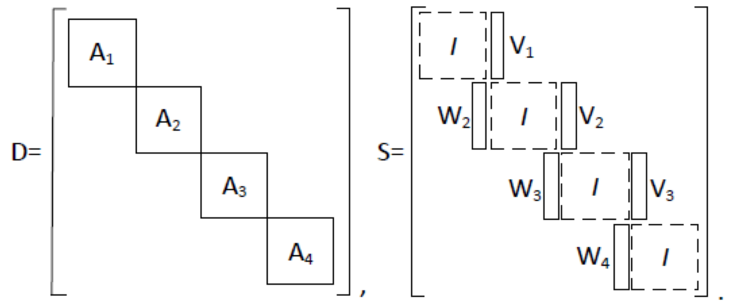
$$\begin{vmatrix} 10^{-10} & 10^{10} \\ 10^{10} & 0 \end{vmatrix} \Rightarrow \begin{vmatrix} 10^{10} & 0 \\ 10^{-10} & 10^{10} \end{vmatrix} \Rightarrow \begin{vmatrix} 10^{10} & 0 \\ 0 & 10^{10} \end{vmatrix}$$

Problem Decomposition

- SPIKE (Polizzi et al)



A X = F



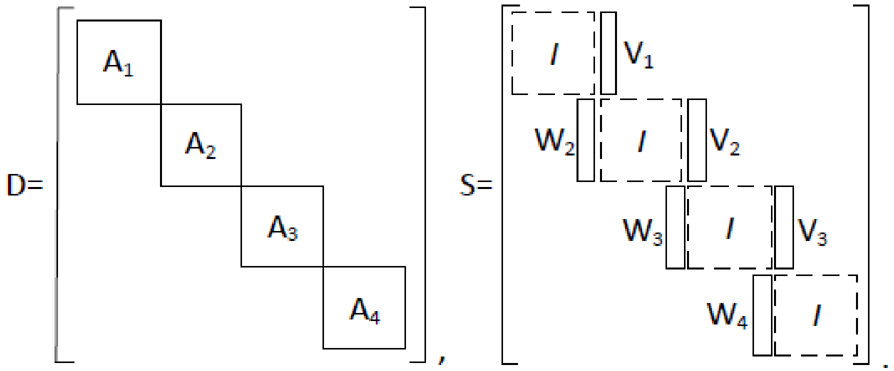
A = DS

D (SX) = F

D Y = F (step 1)

SX = Y (step 2)

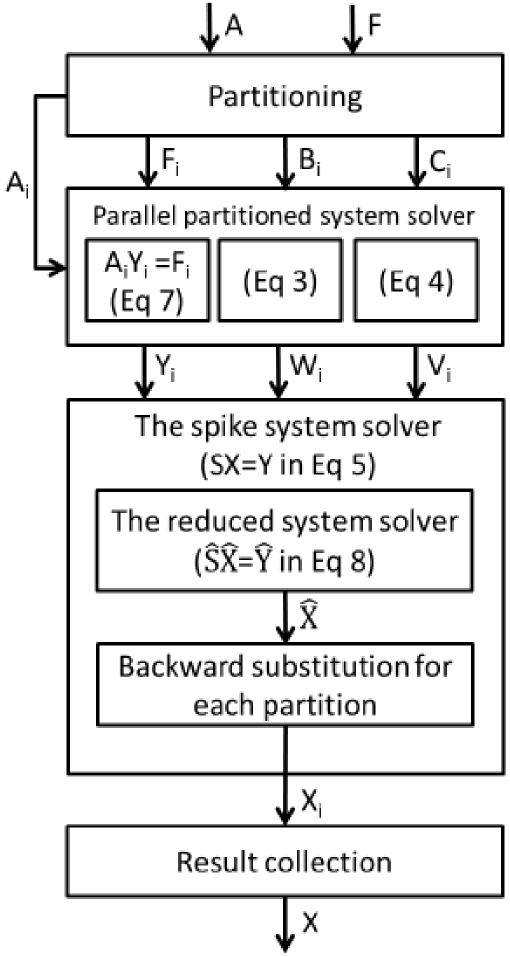
Forming S



$$A_iV_i = \begin{bmatrix}0\\ \vdots\\ 0\\ B_i\end{bmatrix} \qquad (3)$$

$$A_iW_i = \begin{bmatrix}C_i\\ 0\\ \vdots\\ 0\end{bmatrix} \qquad (4)$$

All i tiles call be solved in parallel


A
F
Partitioning
Aᵢ
Fᵢ
Bᵢ
Cᵢ
Parallel partitioned system solver
AᵢYᵢ =Fᵢ (Eq 7)
(Eq 3)
(Eq 4)
Yᵢ
Wᵢ
Vᵢ
The spike system solver (SX=Y in Eq 5)
The reduced system solver (ŜX̂=Ŷ in Eq 8)
X̂
Backward substitution for each partition
Xᵢ
Result collection
X

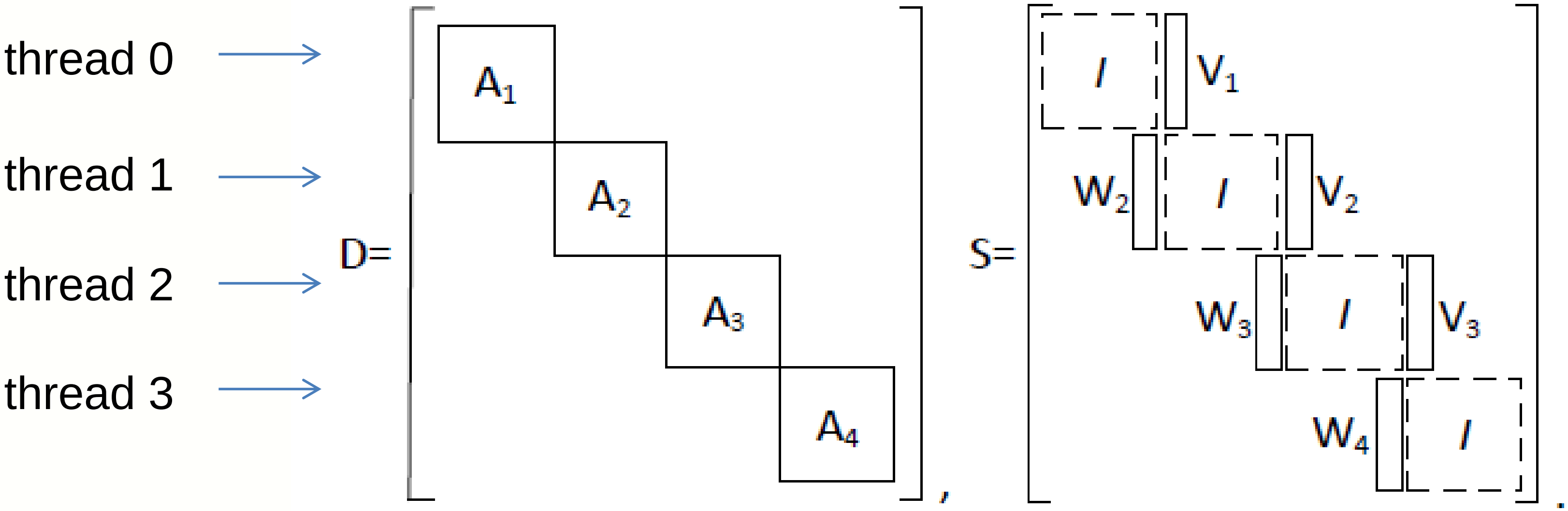
Put the stable sequential algorithm inside each GPU thread

- Each thread will process one tile by itself with a sequential, numerically stable pivoting algorithm

- Note that each thread accessing the first element of its own tile will result in large, strided accesses

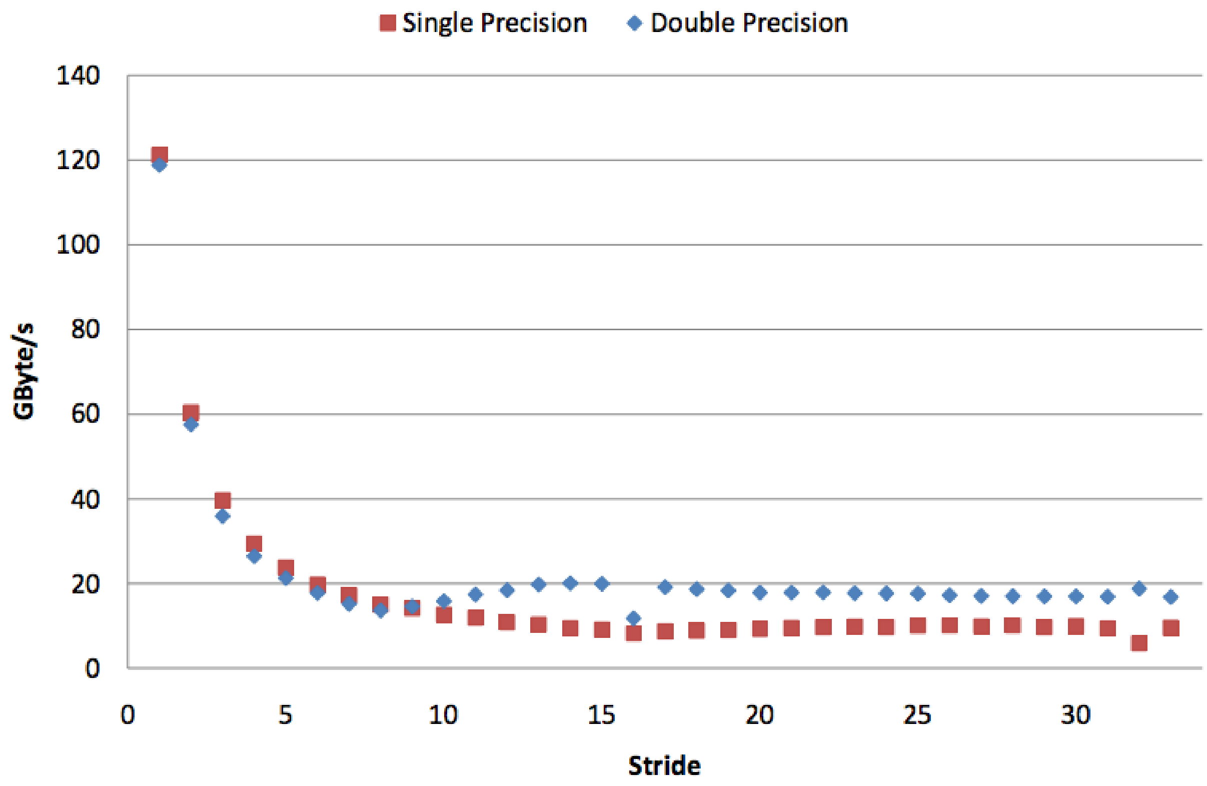
Memory Layout Issue

$$\begin{array}{ll}
\text{thread } 0 \longrightarrow & e_0 \\
\text{thread } 1 \longrightarrow & e_1 \\
\text{thread } 2 \longrightarrow & e_2 \\
\text{thread } 3 \longrightarrow & e_3
\end{array}
\begin{bmatrix} b_0 & c_0 & & \\ a_1 & b_1 & c_1 & \\ & a_2 & b_2 & c_2 \\ & & a_3 & b_3 \end{bmatrix}
\rightarrow
\begin{array}{l} e'_0 \\ e'_1 \\ e'_2 \\ e'_3 \end{array}
\begin{bmatrix} b'_0 & 0 & c'_0 & \\ 0 & b'_1 & 0 & c'_1 \\ a'_2 & 0 & b'_2 & 0 \\ & a'_3 & 0 & b'_3 \end{bmatrix}
\rightarrow
\frac{\begin{bmatrix} b'_0 & c'_0 \\ a'_2 & b'_2 \end{bmatrix}}{\begin{bmatrix} b'_1 & c'_1 \\ a'_3 & b'_3 \end{bmatrix}}$$



GPU Memory Bandwidth vs. Stride

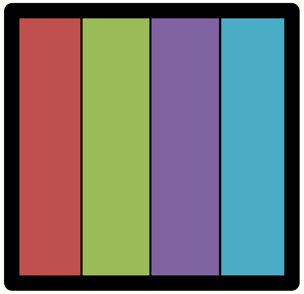
- SAXPY with stride:
  - – y[i * stride] = a * x[i * stride] + y[i * stride];



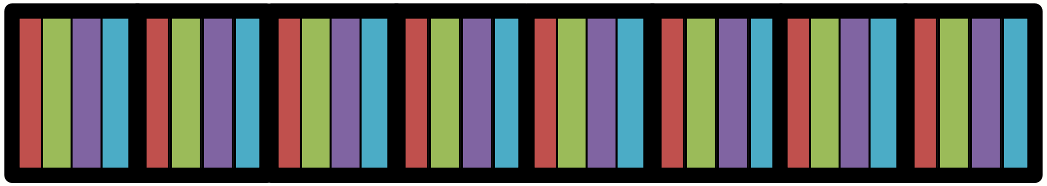
"Efficient Sparse Matrix-Vector Multiplication on CUDA"
Nathan Bell and Michael Garland, in, *"NVIDIA Technical Report NVR-2008-004"*,,

Tiles Processed by Each Thread

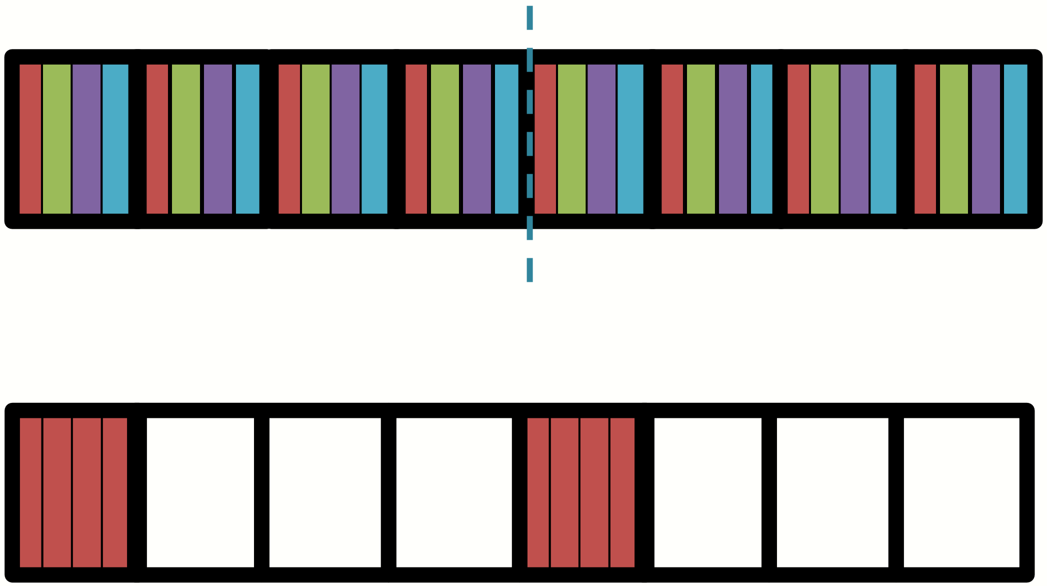
- Each tile:



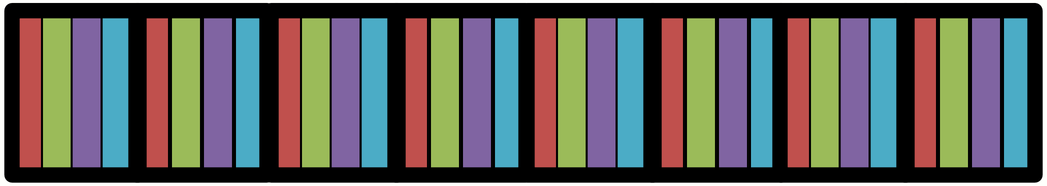
- Layout of all tiles: (similar to ELL before transposition)

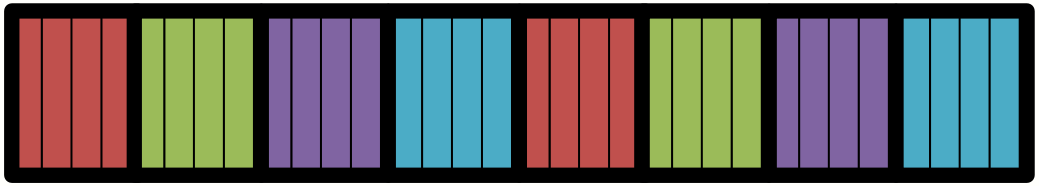


Another Data Layout Alternative ASTA



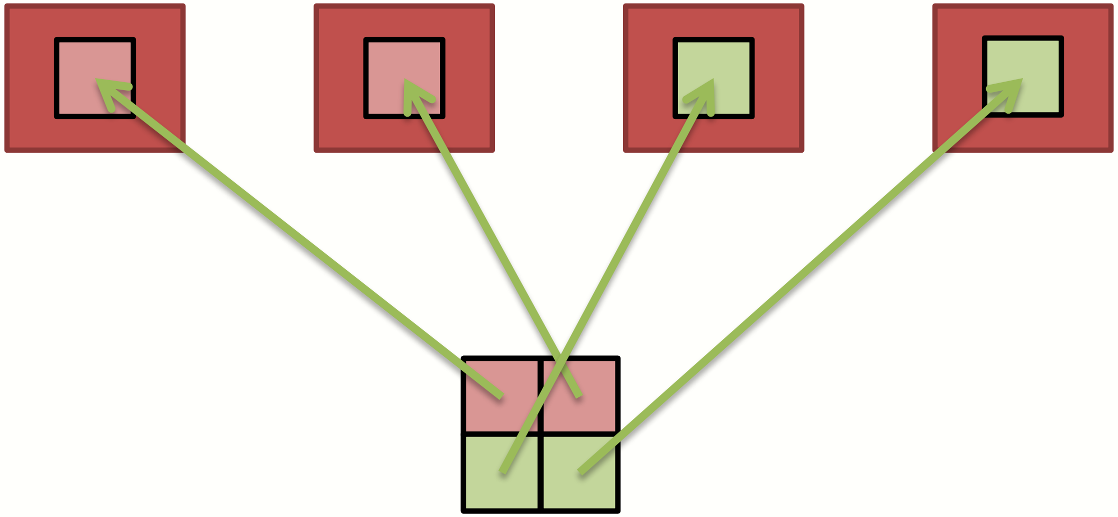
ASTA Data Layout





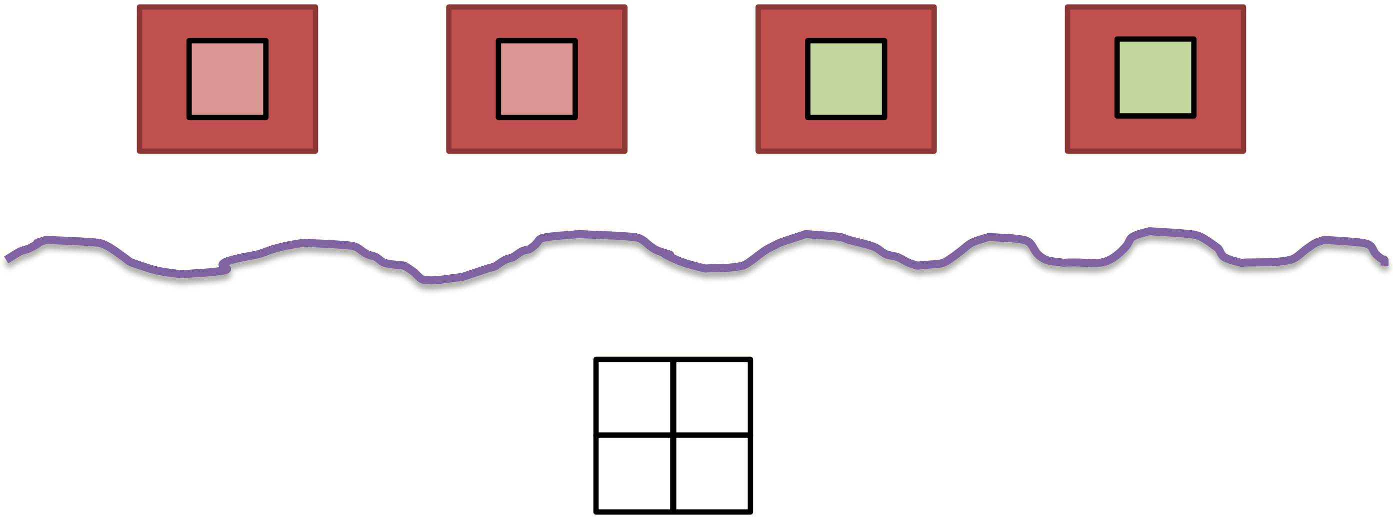
In-place Transpostion – Step 1

```
// data[W][H]-->data[H][W]
parallel for (j<W)
  parallel for (i<H)
    float temp = data[j][i]; //offset = j*H + i
```



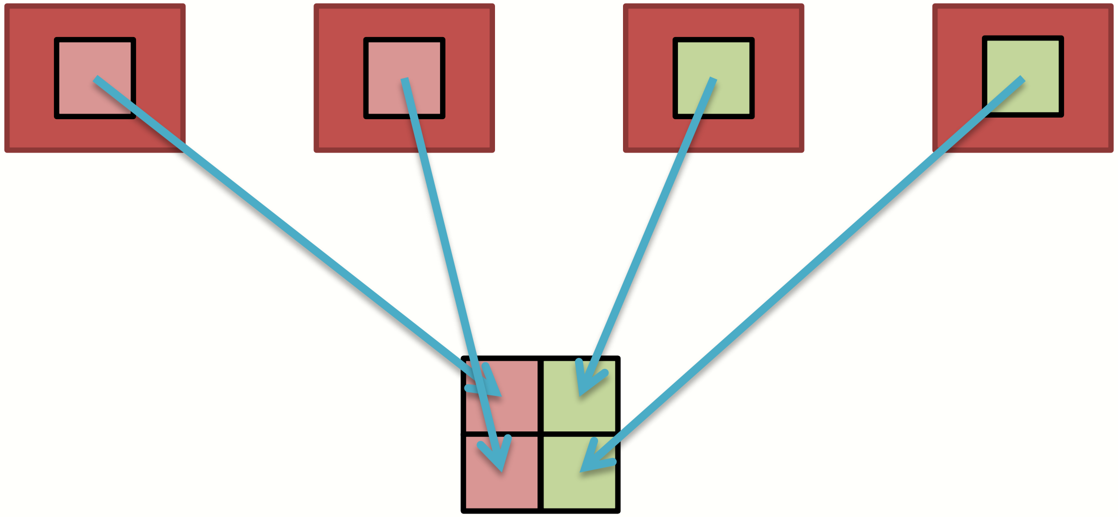
In-place Transpostion: Barrier

```
// data[W][H]-->data[H][W]
parallel for (j<W)
  parallel for (i<H)
    float temp = data[j][i]; //offset = j*H + i
    barrier();
```



In-place Transpostion: Step 2

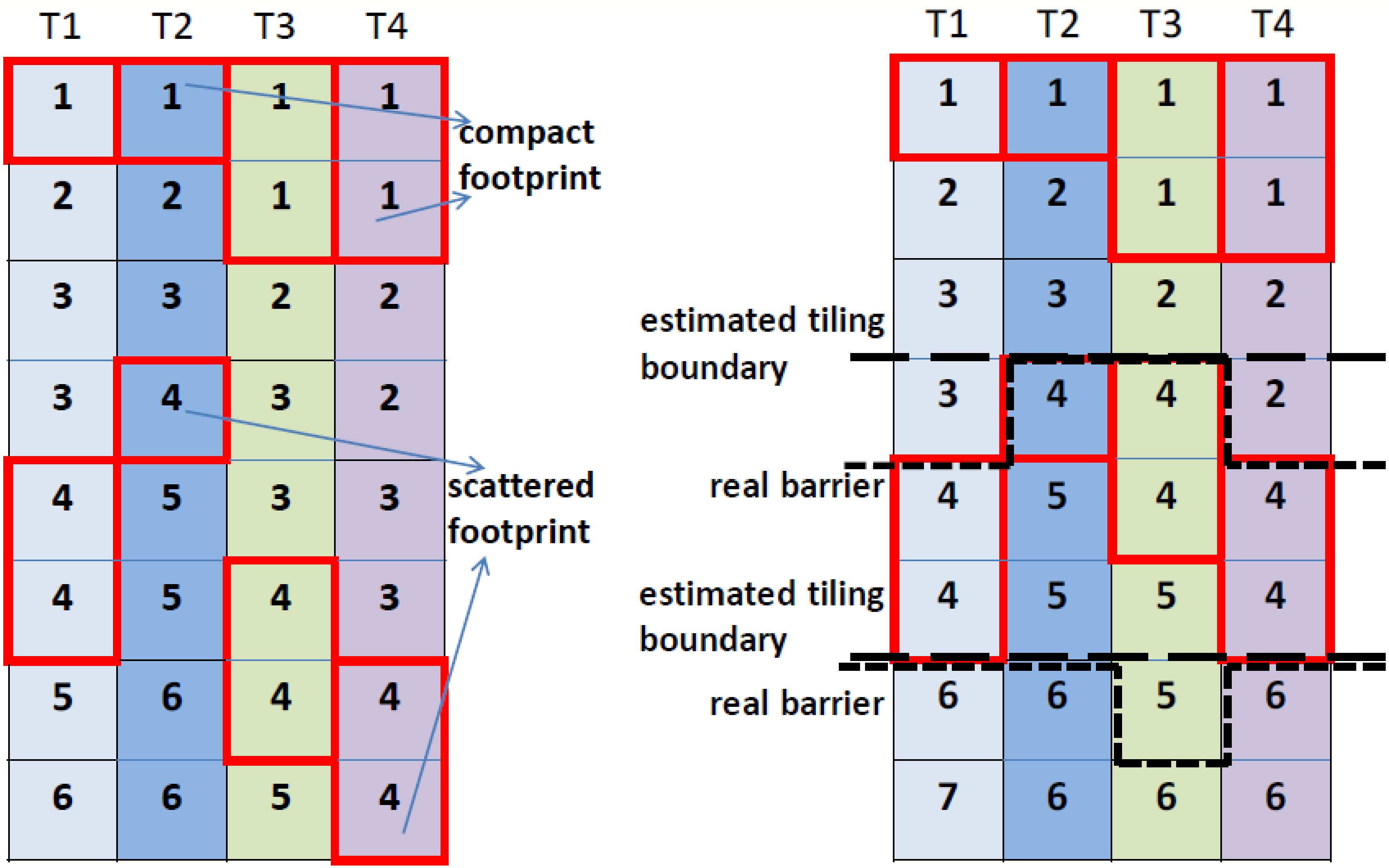
```
// data[W][H]-->data[H][W]
parallel for (j<W)
  parallel for (i<H)
    float temp = data[j][i]; //offset = j*H + i
    barrier();
    data[i][j] = temp; //offset = i*W + j
```



AoS to ASTA Transformation

AoS to ASTA Marshaling Kernel	Global Memory Throughput (GB/s)	Fine Print
Out-of-Place	80	2x Space
In-Place Barrier Sync	95	Tile Size (tunable) < On-chip Memory

Dynamic Tiling



T1	T2	T3	T4
1	1	1	1
2	2	1	1
3	3	2	2
3	4	3	2
4	5	3	3
4	5	4	3
5	6	4	4
6	6	5	4

compact footprint

scattered footprint

(a) un-tiled

T1	T2	T3	T4
1	1	1	1
2	2	1	1
3	3	2	2
3	4	4	2
4	5	4	4
4	5	5	4
6	6	5	6
7	6	6	6

estimated tiling boundary

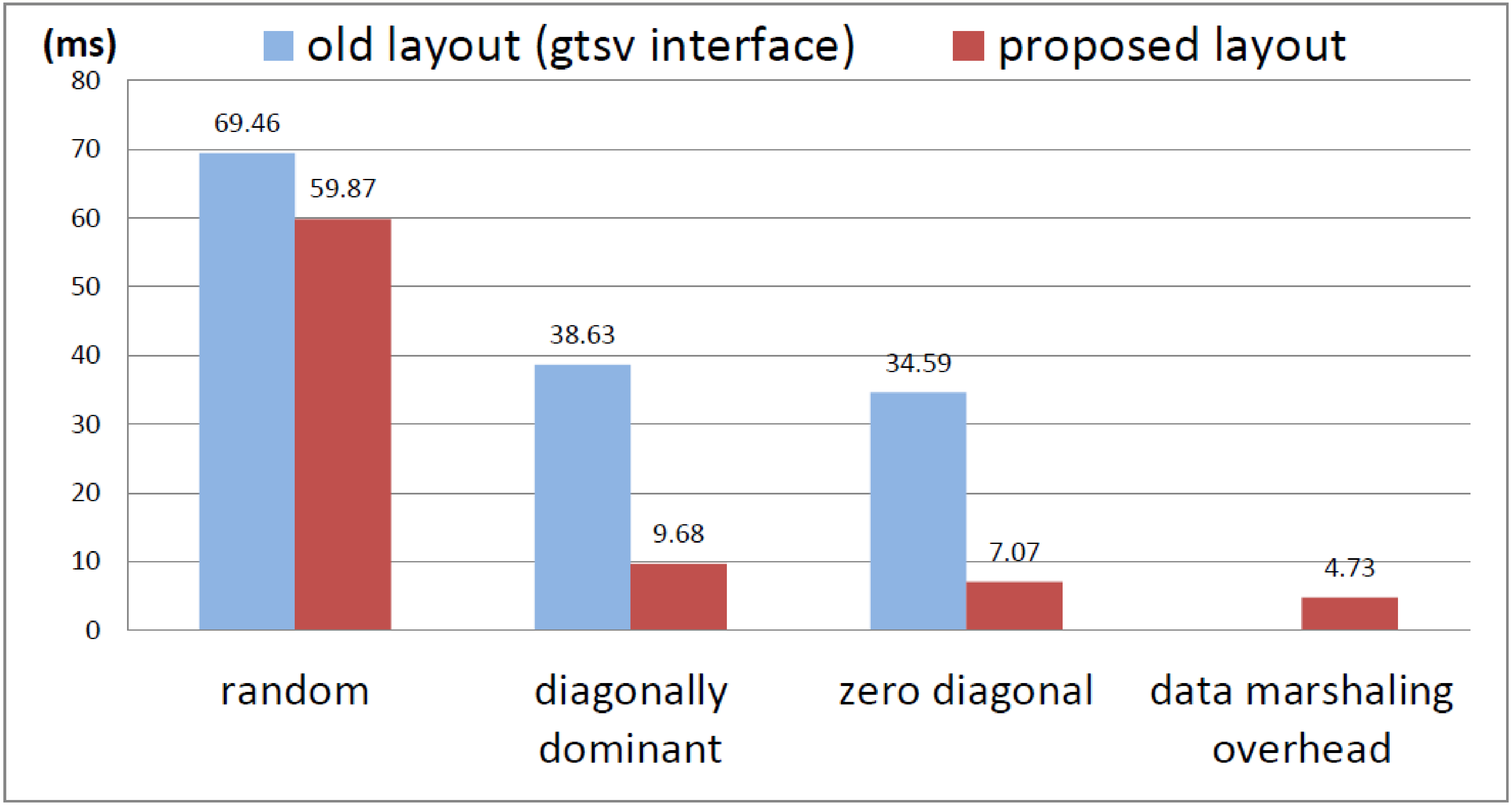
real barrier

estimated tiling boundary

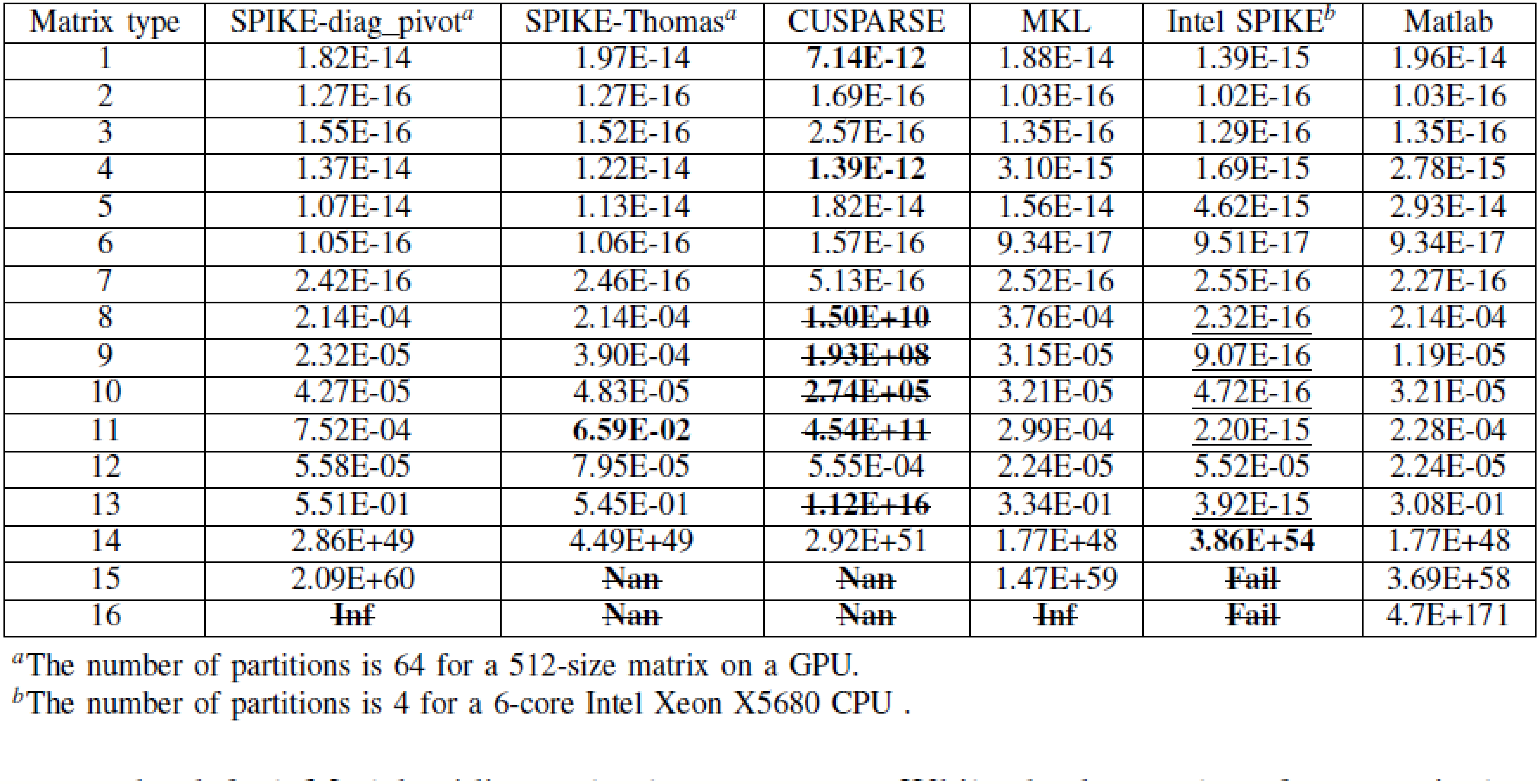
real barrier

(b) tiled

Cost and Benefit of ASTA Layout Marshaling



Error and Stability

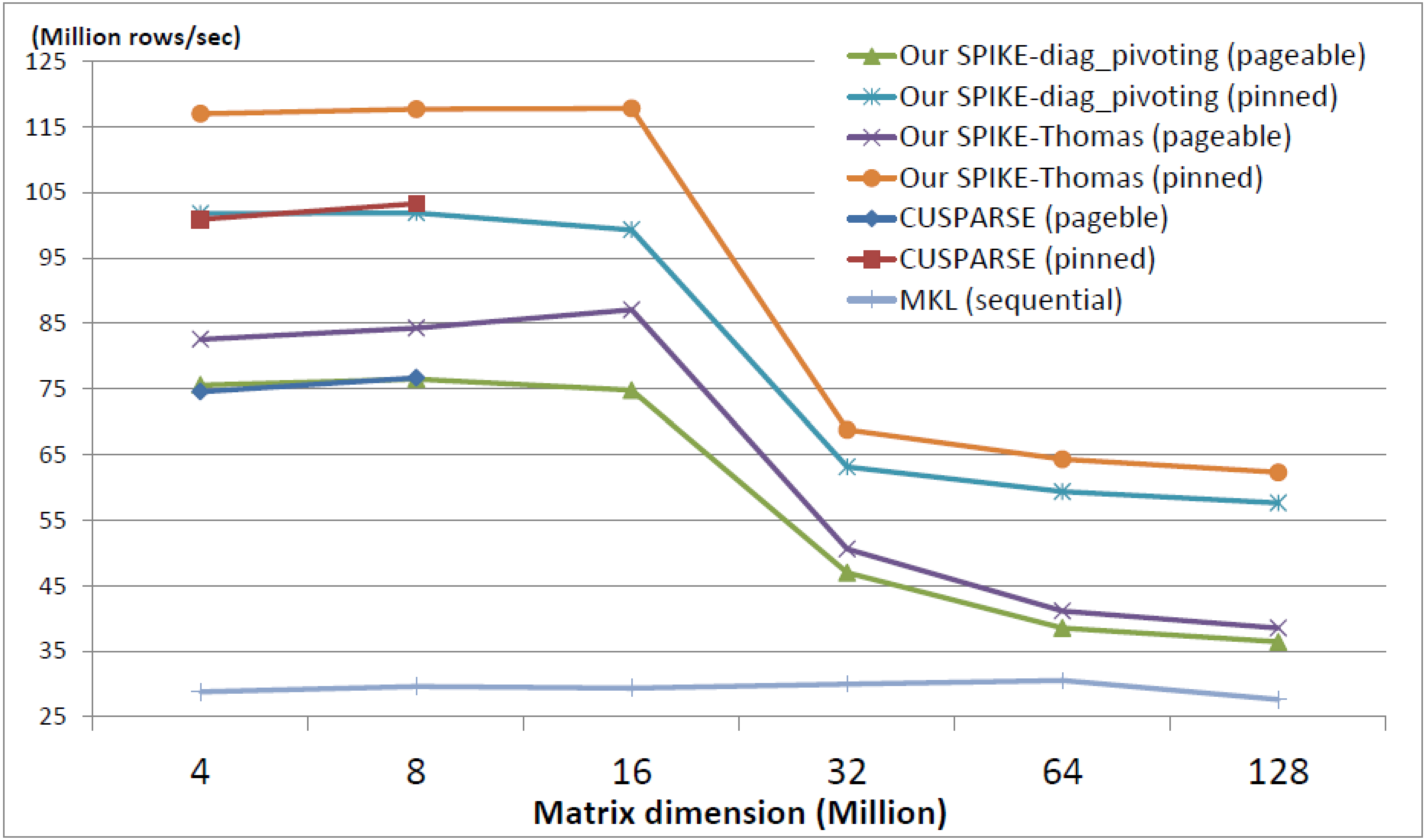


Matrix type	SPIKE-diag_pivot[a]	SPIKE-Thomas[a]	CUSPARSE	MKL	Intel SPIKE[b]	Matlab
1	1.82E-14	1.97E-14	**7.14E-12**	1.88E-14	1.39E-15	1.96E-14
2	1.27E-16	1.27E-16	1.69E-16	1.03E-16	1.02E-16	1.03E-16
3	1.55E-16	1.52E-16	2.57E-16	1.35E-16	1.29E-16	1.35E-16
4	1.37E-14	1.22E-14	**1.39E-12**	3.10E-15	1.69E-15	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**~~	4.7E+171

[a] The number of partitions is 64 for a 512-size matrix on a GPU.

[b] The number of partitions is 4 for a 6-core Intel Xeon X5680 CPU .

Speed


(Million rows/sec)
125
115
105
95
85
75
65
55
45
35
25
Our SPIKE-diag_pivoting (pageable)
Our SPIKE-diag_pivoting (pinned)
Our SPIKE-Thomas (pageable)
Our SPIKE-Thomas (pinned)
CUSPARSE (pageble)
CUSPARSE (pinned)
MKL (sequential)
4
8
16
32
64
128
Matrix dimension (Million)

Summary

- Designing high-performance, scalable, and numerically stable algorithms is challenging
- Fast transposition and dynamic tiling provides strong building blocks
- We have built the first high-performance, scalable, and numerical stable tri-diagonal solver many-cores
  - Matches the speed of CUSPARSE
  - Surpasses the data scalability of CUSPARSE
  - Matches numerical stability of Intel MKL

THANK YOU!
ANY QUESTIONS?

New Kernel Development Tools

- OpenACC Accelerator Pragmas
  - Wider use of GPU in large applications but less performance in each kernel
  - Cray and others
- Portland Group CUDA FORTAN compiler
- NVIDIA Thrust
- Microsoft C++AMP

VecAdd in OpenACC

```
 void computeAcc(float *C, const float *A, const float *B, int n)
{

#pragma acc parallel loop copyin(A[0:n]) copyin(B[0:n]) copyout(C[0:n])
  for (int i=0; i<n; i++) {
        C[i] = A[i] + B[i];
  }
}
```

VecAdd in C++AMP

```
#include <amp.h>
using namespace concurrency;

void vecAdd(float* A, float* B, float* C, int n)
{
    array_view<const float,1> AV(n,A), BV(n,B);
    array_view<float,1> CV(n,C);
    CV.discard_data();
    parallel_for_each(CV.extent, [=](index<1> i) restrict(amp)
    {
        CV[i] = AV[i] + BV[i];
    });
    CV.synchronize();
}
```

Thank You!

Numerical Stability

- Algorithms that can always find an appropriate operation order and thus finding a solution to the problem as long as it exists for any given input values are *numerically stable*.

- Algorithms that fall short are *numerically unstable*.