GPU Supercomputing – From Blue Waters to Exascale

Wen-mei Hwu

Professor, University of Illinois at Urbana-Champaign (UIUC)

CTO, MulticoreWare Inc.

New BW Configuration

Cray System & Storage cabinets:	• >300
Compute nodes:	• >25,000
Usable Storage Bandwidth:	• >1 TB/s
System Memory:	• >1.5 Petabytes
Memory per core module:	• 4 GB
Gemin Interconnect Topology:	• 3D Torus
Usable Storage:	• >25 Petabytes
Peak performance:	• >11.5 Petaflops
Number of AMD processors:	• >49,000
Number of AMD x86 core module:	• >380,000
Number of NVIDIA GPUs:	• >3,000

NSF NCSA

ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Today's and Tomorrow's Intellectual Challenges

- Challenges that will be faced by the Science Teams include:
  - **Scaling applications to large processor counts**
    - In the face of limited bandwidth
  - **Effective using of many core and accelerator components**
  - Using both general purpose and accelerated nodes in single applications
  - Application based resiliency

- NCSA establishing a focused effort in *NCSA/UIUC Enhanced Intellectual Services for Petascale Performance* (NEIS-P2) to work directly with the Science Teams and also the general community to enable teams to take full advantage of the extraordinary capabilities of Petascale+ scale systems.
  - Due in February 2012

Science Area	Number of Teams	Codes	Structured Grids	Unstructured Grids	Dense Matrix	Sparse Matrix	N-Body	Monte Carlo	FFT	Significant I/O
Climate and Weather	3	CESM, GCRM, CM1, HOMME	X	X		X		X		
Plasmas/Magnetosphere	2	H3D(M), OSIRIS, Magtail/UPIC	X				X		X	X
Stellar Atmospheres and Supernovae	2	PPM, MAESTRO, CASTRO, SEDONA	X			X		X		X
Cosmology	2	Enzo, pGADGET	X			X	X			
Combustion/Turbulence	1	PSDNS	X						X	
General Relativity	2	Cactus, Harm3D, LazEV	X			X				
Molecular Dynamics	4	AMBER, Gromacs, NAMD, LAMMPS			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, MILD, USQCD	X		X	X	X		X	
Social Networks	1	EPISIMDEMICS								
Evolution	1	Eve								
Computer Science	1			X	X	X			X	X

Current Science Team GPU Plans and Results

- ***Nearly 1/3 of PRAC projects have active GPU efforts, including***
  - AMBER
  - LAMMPS
  - USQCD/MILC
  - GAMESS
  - NAMD
  - QMCPACK
  - PLSQR/SPECFEM3D
- Others are investigating use of GPUs (e.g., Cactus, PPM, AWP-ODC)
- Some examples follow

There is a critical need for scalable kernel libraries

- Both CPUs and GPUs require scalable parallel kernel libraries
  - GPU needs are more urgent
- Only a small percentage of the Intel Math Kernel Library (MKL) functions have scalable forms.
- Software lasts through many hardware generations and needs to be scalable to be economically viable

Solid Scalable Kernels

- Dense SGEMM/DGEMM, LU, Triangular solvers (CUBLAS, CULA, MAGMA)
- Sparse Matrix Vector Multiplication, Tridiagonal solvers (CUSP, QUDA, PARBOIL)
- FFTs, Convolutions (CUFFT, Parboil)
- N-Body (NAMD/VMD, FMM BU, PARBOIL)
- Histograms (PARBOIL)
- Some PDE solvers (CURRENT, PARBOIL)

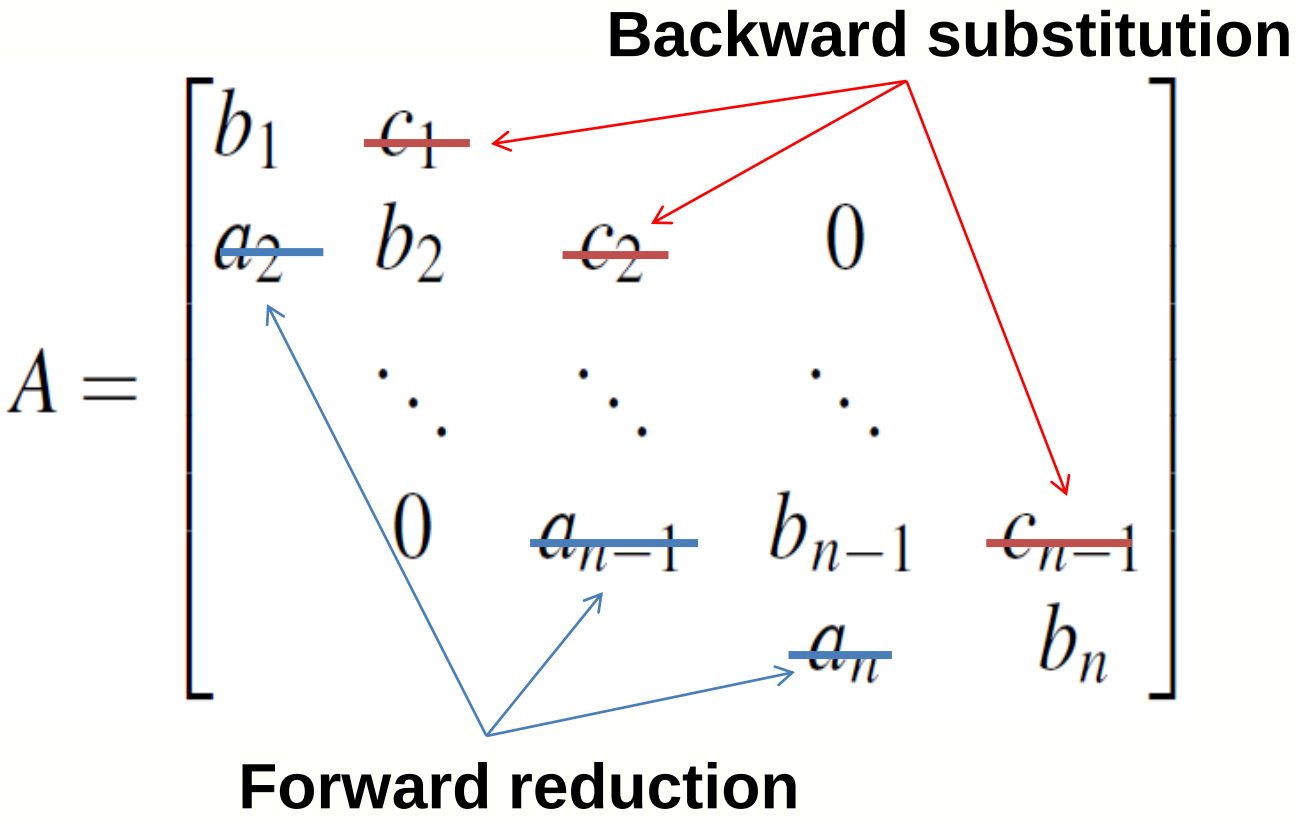
Example: Tridiagonal Solver

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

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

Thomas Algorithm

- Special two-way Gaussian elimination



Parallel Cyclic Reduction(PCR)

- Simultaneous reduction of odd and even rows – a.k.a. forward reduction only CR

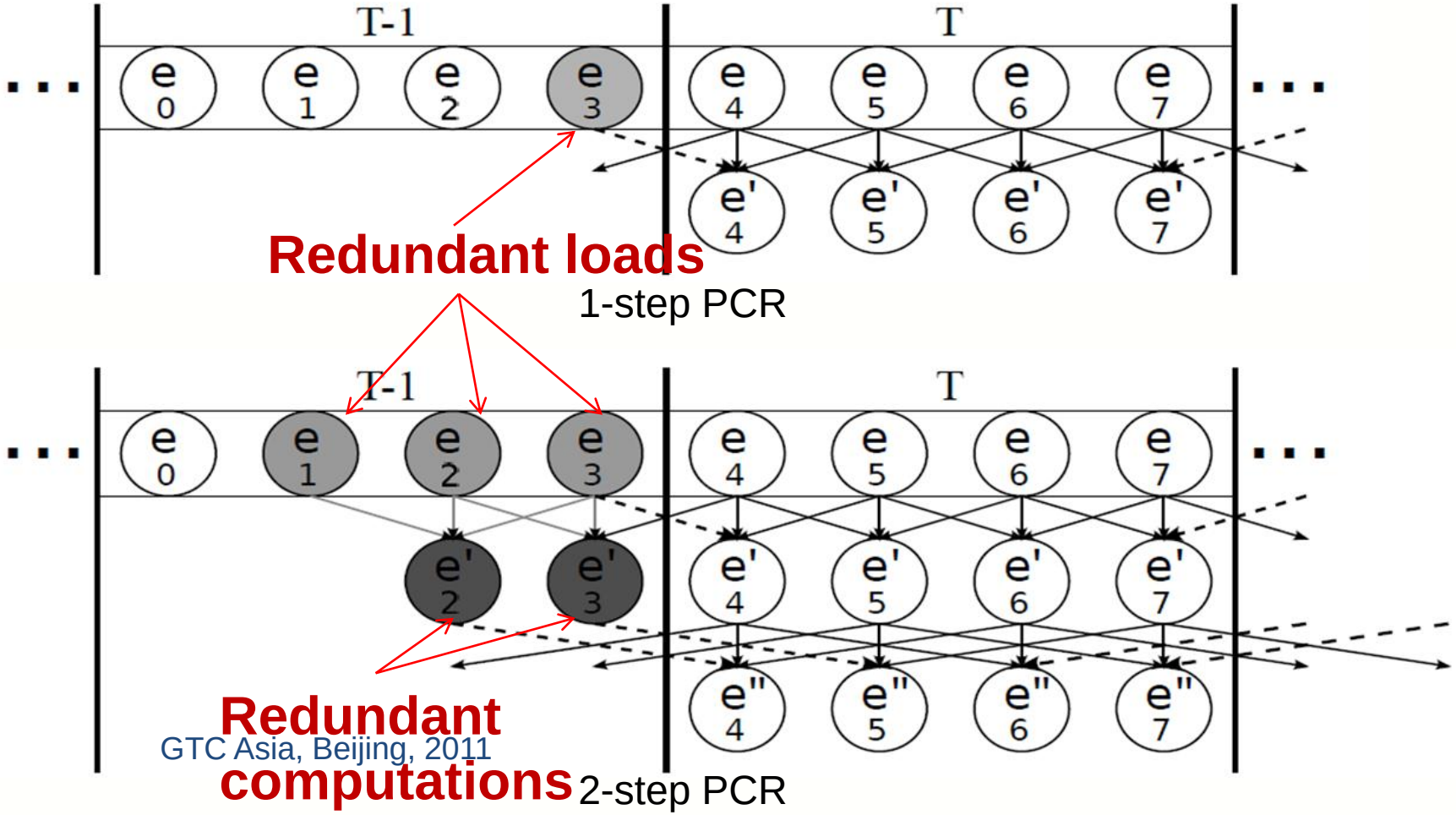
$$\begin{matrix} e_1 \\ e_2 \\ e_3 \\ e_4 \end{matrix}\begin{bmatrix} b_1 & c_1 & & \\ a_2 & b_2 & c_2 & \\ & a_3 & b_3 & c_3 \\ & & a_4 & b_4 \end{bmatrix} \rightarrow \begin{matrix} e'_1 \\ e'_2 \\ e'_3 \\ e'_4 \end{matrix}\begin{bmatrix} b'_1 & 0 & c'_1 & \\ 0 & b'_2 & 0 & c'_2 \\ a'_3 & 0 & b'_3 & 0 \\ & a'_4 & 0 & b'_4 \end{bmatrix} \rightarrow \begin{bmatrix} b'_1 & c'_1 \\ a'_3 & b'_3 \end{bmatrix} \\ \hline \begin{bmatrix} b'_2 & c'_2 \\ a'_4 & b'_4 \end{bmatrix}$$

Summary of Previous Approaches

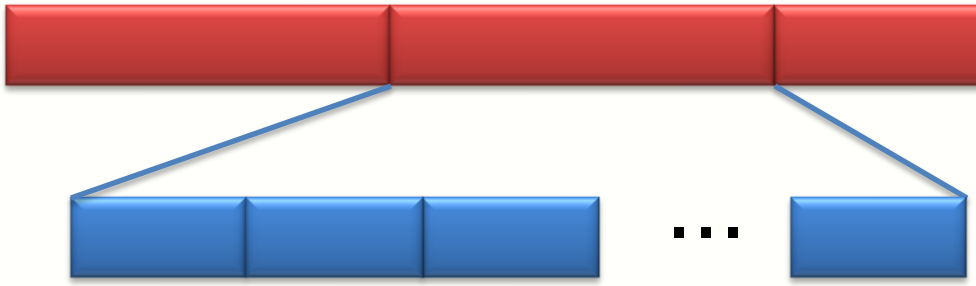
	Complexity	Number of operations	Number of processing steps with n-parallelism machine
Thomas	O(n)	2n	2n
PCR	O(n log n)	12 n log n	log n

Parallelization of Tiled PCR (Owens)

- Exponential growth of halo elements and subsequent reductions



Hierarchical Tiling

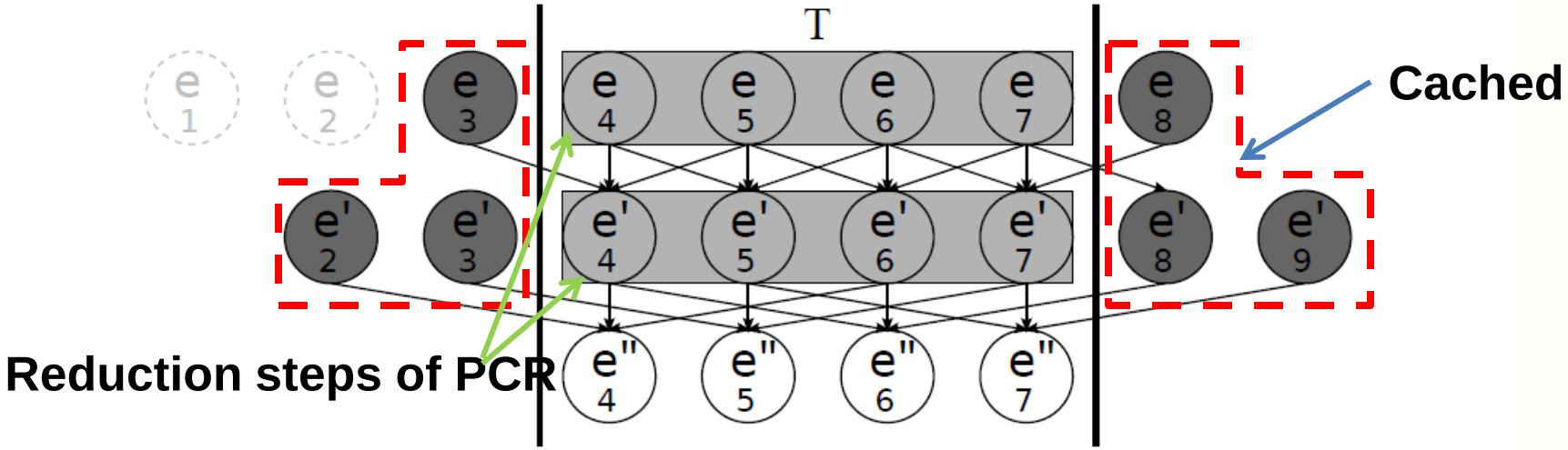


Parallel 1st level tile

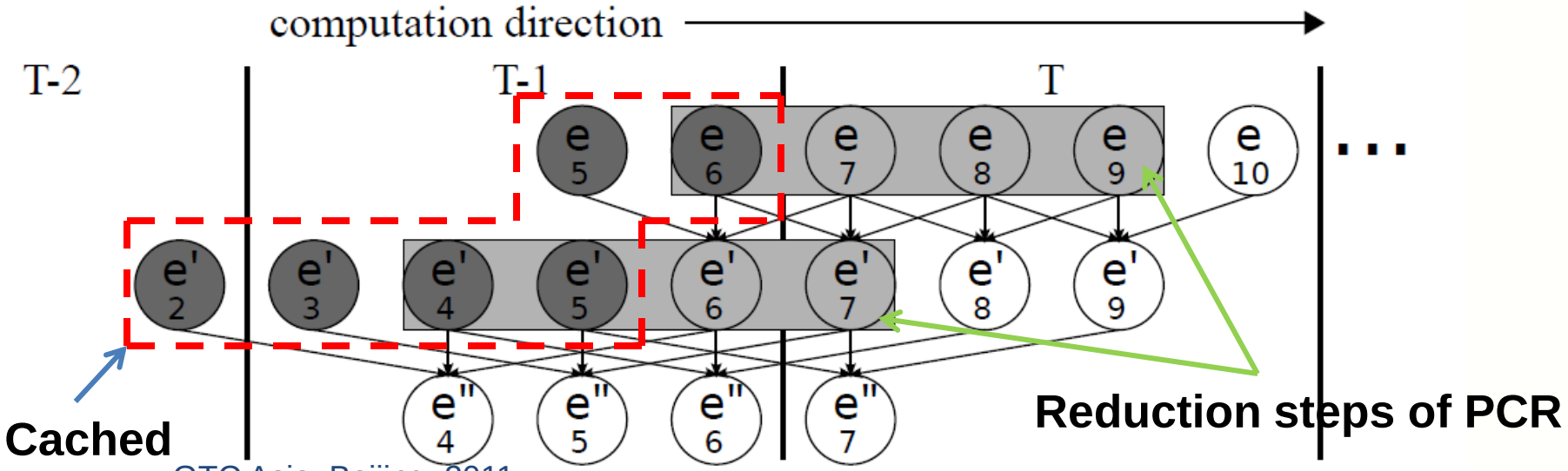
2nd level tile processed Sequentially using the sliding window

Mapping onto GPU	
1st level tile	One per thread block
2nd level tile	Collaborative streaming within thread block
2nd level tile buffer	Shared memory

Tiled PCR using HT Sliding Window

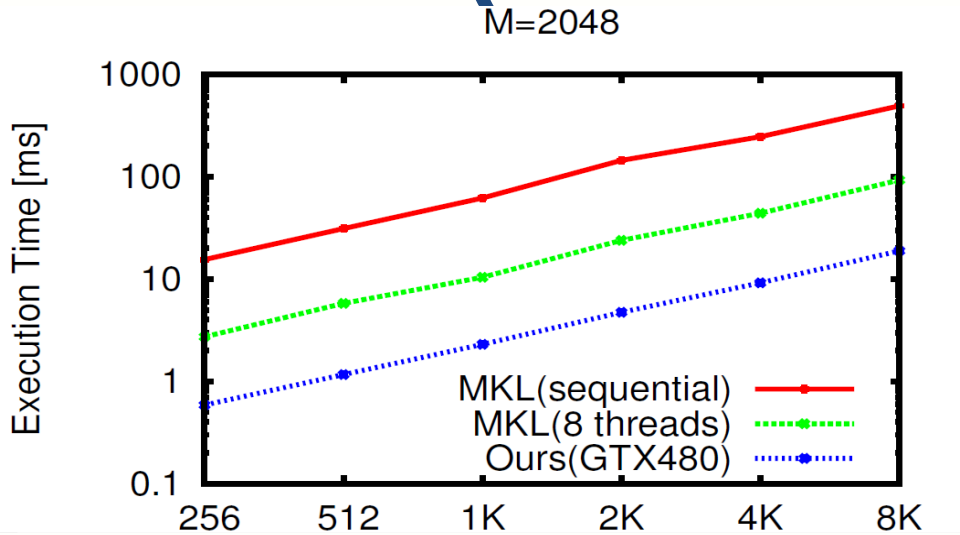


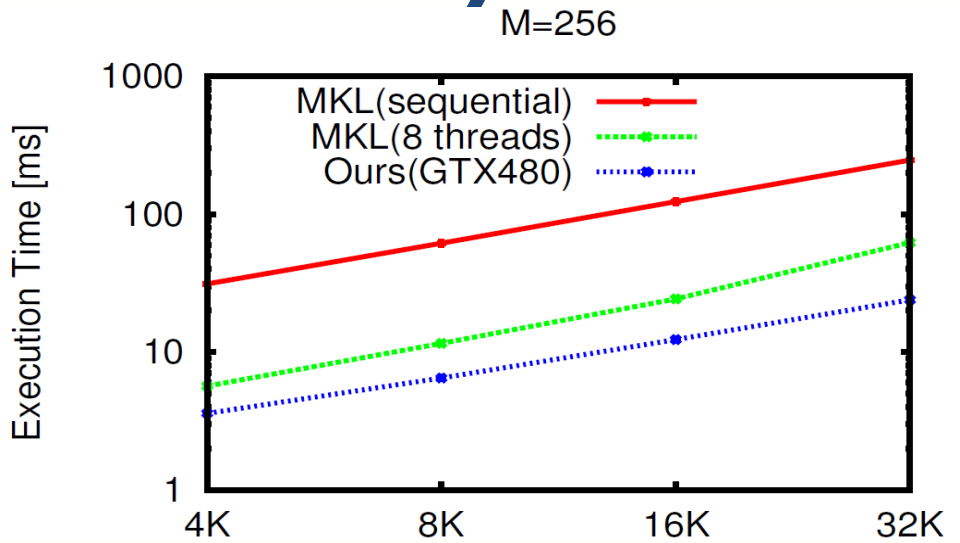
(a) 2-step PCR using cached dependency from both sides

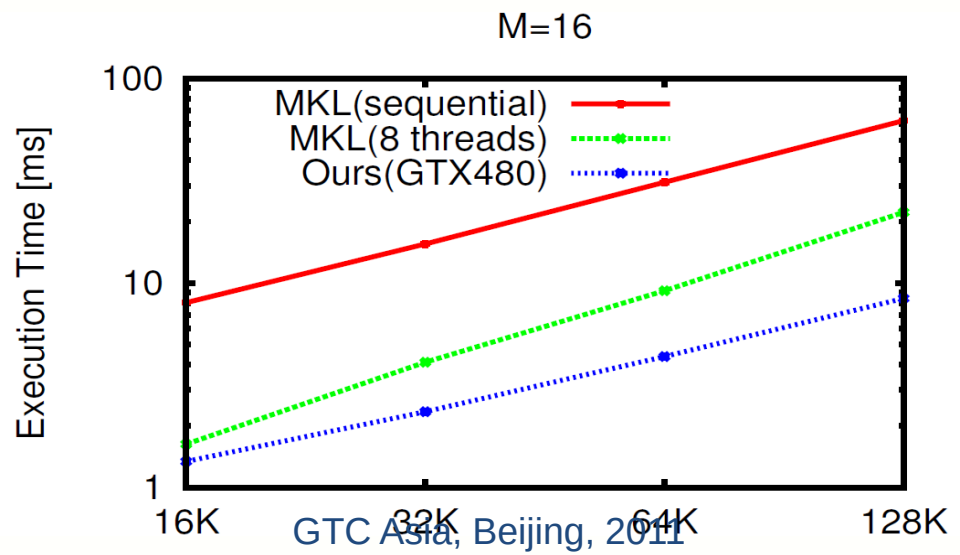


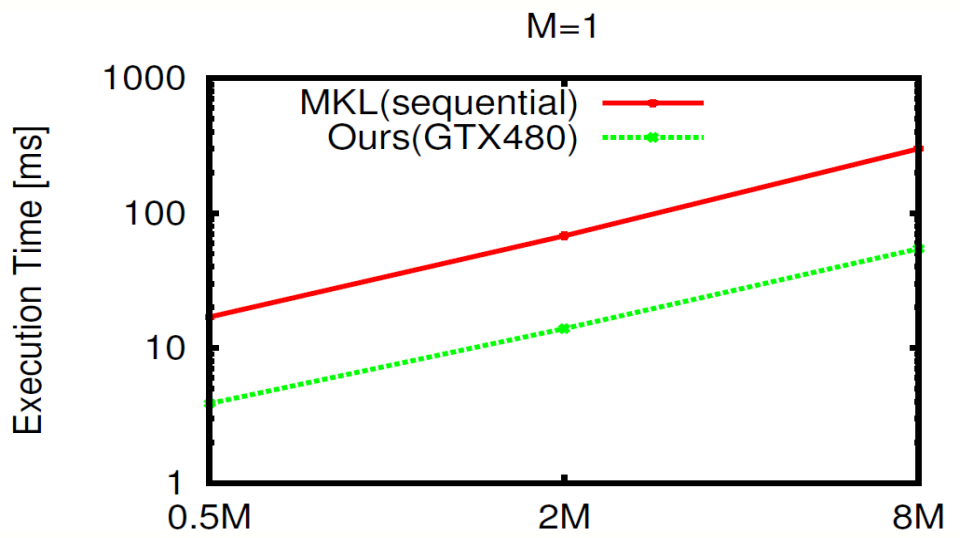
(b) 2-step PCR using cached dependency from lefthand side

Scalable in size and # systems (8 million elements)









M := number of systems / x-axis := number of unknowns

A Scalable Kernel Requires

- Massive parallelism in application algorithms
  - Data parallelism
- Regular computation and data accesses
  - Similar, balanced work for parallel threads
- Avoidance of conflicts in critical resources
  - Off-chip DRAM (Global Memory) bandwidth
  - Conflicting parallel updates to memory locations

Eight Techniques for Scalable Kernels (so far)

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

http://courses.engr.illinois.edu/ece598/hk/

Use of Optimizations in Parboil

Benchmark	Unoptimized Implementation Bottleneck	Optimizations Applied	Optimized Implementation Bottleneck	Primary Limit of Efficiency
cutcp	Contention, Locality	Scatter-to-Gather, Binning, Regularization, Coarsening	Instruction Throughput	Reads/Checks of Irrelevant Bin Data
mri-q	Poor Locality	Data Layout Transformation, Tiling, Coarsening	Instruction Throughput	N/A (true bottleneck)
gridding	Contention, Load Imbalance	Scatter-to-Gather, Binning, Compaction, Regularization, Coarsening	Instruction Throughput	Reads/Checks of Irrelevant Bin Data
sad	Locality	Tiling, Coarsening	Memory Bandwidth/Latency	Register Capacity
stencil	Locality	Coarsening, Tiling	Bandwidth	Local Memory, Register Capacity
tpacf	Locality, Contention	Tiling, Privatization, Coarsening	Instruction Throughput	N/A (true bottleneck)
lbm	Bandwidth	Data Layout Transformation	Bandwidth	N/A (true bottleneck)
dmm	Bandwidth	Coarsening, Tiling	Instruction Throughput	N/A (true bottleneck)
spmv	Bandwidth	Data Layout Transformation	Bandwidth	N/A (true bottleneck)
bfs	Contention, Load Imbalance	Privatization, Compaction, Regularization	Bandwidth	Whole-Device Local Memory Capacity
histogram	Contention, Bandwidth	Privatization, Scatter-to-Gather	Bandwidth	Reads of Irrelevant Input (alleviated by cache)

How a mathematician writes matrix multiplication

$$(MN)_{j,i} = \sum_k M_{j,k} N_{k,i}$$

How a smart CUDA programmer writes matrix multiplication

```
#define TILE_N 16
#define TILE_TB_HEIGHT 8
#define TILE_M (TILE_N*TILE_TB_HEIGHT)
__global__ void mysgemmNT( const float *A, int lda, const float *B,
        int ldb, float* C, int ldc, int k, float alpha, float beta ){
{
  float c[TILE_N];
  for (int i=0; i < TILE_N; i++) c[i] = 0.0f;
  int mid = threadIdx.y * blockDim.x + threadIdx.x;
  int m = blockIdx.x * TILE_M + mid;
  int n = blockIdx.y * TILE_N + threadIdx.x;
  __shared__ float b_s[TILE_TB_HEIGHT][TILE_N];
  for (int i = 0; i < k; i+=TILE_TB_HEIGHT) {
    float a;
    b_s[threadIdx.y][threadIdx.x]=B[n + (i+threadIdx.y)*ldb];
    __syncthreads();
    for (int j = 0; j < TILE_TB_HEIGHT; j++) {
      a = A[m + (i+j)*lda];
      for (int kk = 0; kk < TILE_N; kk++) c[kk] += a * b_s[j][kk];
    }
    __syncthreads();
  }
  int t = ldc*blockIdx.y * TILE_N + m;
  for (int i = 0; i < TILE_N; i++) {
    C[t+i*ldc] = C[t+i*ldc] * beta + alpha * c[i];
  }
}
...
dim3 grid( m/TILE_M, n/TILE_N ), threads( TILE_N, TILE_TB_HEIGHT );
mysgemmNT<<<grid, threads>>>( A, lda, B, ldb, C, ldc, k, alpha, beta);
...
```

Writing efficient code is complicated.

Tools can provide focused help or broad help

Planning how to execute an algorithm

- Choose data structures
- Decompose work into tasks
- Schedule tasks to threads

MCUDA/MOpenCL

Implementing the plan

- Memory allocation
- Data movement

GMAC

- Pointer operations
- Index arithmetic

Data Layout

Pyon

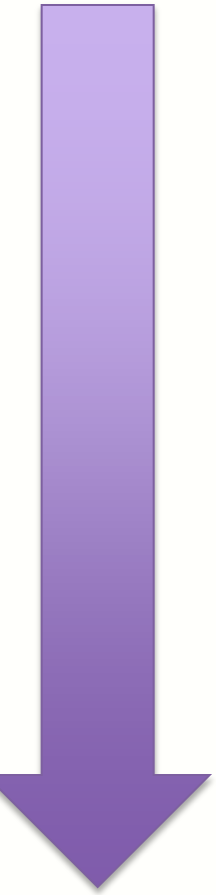
- Kernel dimensions
- Thread ID arithmetic

TM, Thread Coarsening

- Synchronization
- Temporary data structures

UIUC/MCW Tools for Heterogeneous Parallel Programming

Early



Late

Writing optimizable, portable, kernels

Pyon

Task and data movement & multi-GPU copy support

GMAC, TM

Higher-level Interfaces for Programmability, Portability, and Performance

Performance tools

PPA/ADAPT

TC/SM

DL (Data Layout)

Performance portability, Analysis, optimization (with collaboration with DSLs – Hanrahan/Keutzer)

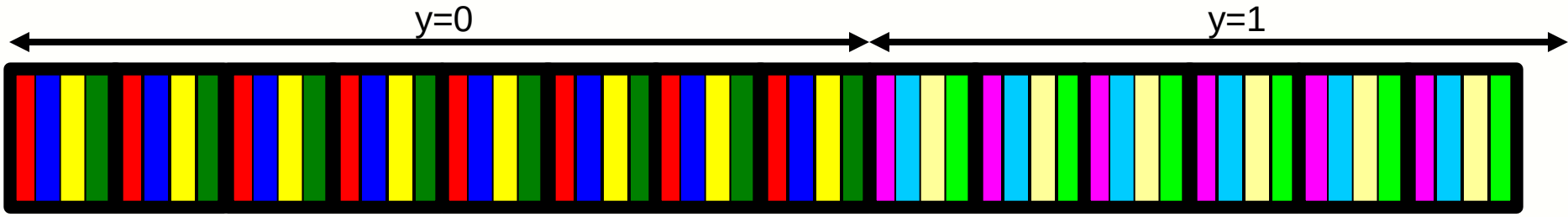
Use of UIUC/MCW Tools in Blue Waters

- Introduce compiler and library capabilities into the science team workflow to significantly reduce the programming effort and impact on code maintainabilitys:
  - Compiler based directives
  - GMAC - a library that provides global shared memory and automates data transfer/coherence between the CPUs and the GPUs in a node
  - DL is a compiler-based memory layout transformation tool that uses a combination of compiler and runtime support to ease the task of adjusting memory layout to satisfy conflicting needs between the CPU and the GPU
  - TC is a compiler based tool for thread coarsening and data tiling.
- Provide expert support to the science teams through hand-on workshops, courses, and individualized collaboration programs.

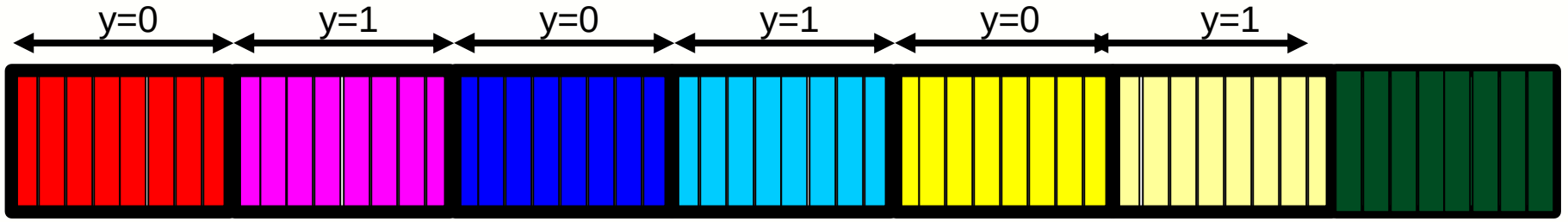
Example - DL (Data Layout)

- DRAM bursts are formed differently in a heterogeneous system
  - From last level cache misses on CPUs
  - From SIMD-ized memory accesses on many-core architectures like GPUs
- Data layout transformation can mitigate the gap
  - E.g.: Array-of-structure / Discrete-arrays
  - Bridging divergent layout requirements between CPU cores and GPU cores
  - Transparent and efficient marshaling

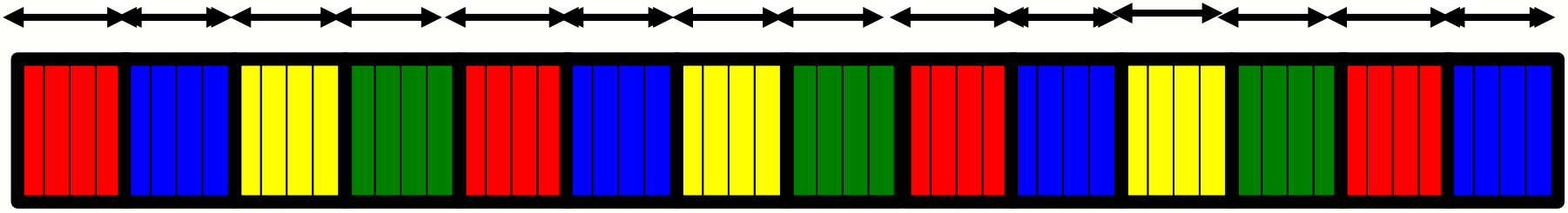
Data Layout Alternatives



Array of Structure: [z][y][x][e]

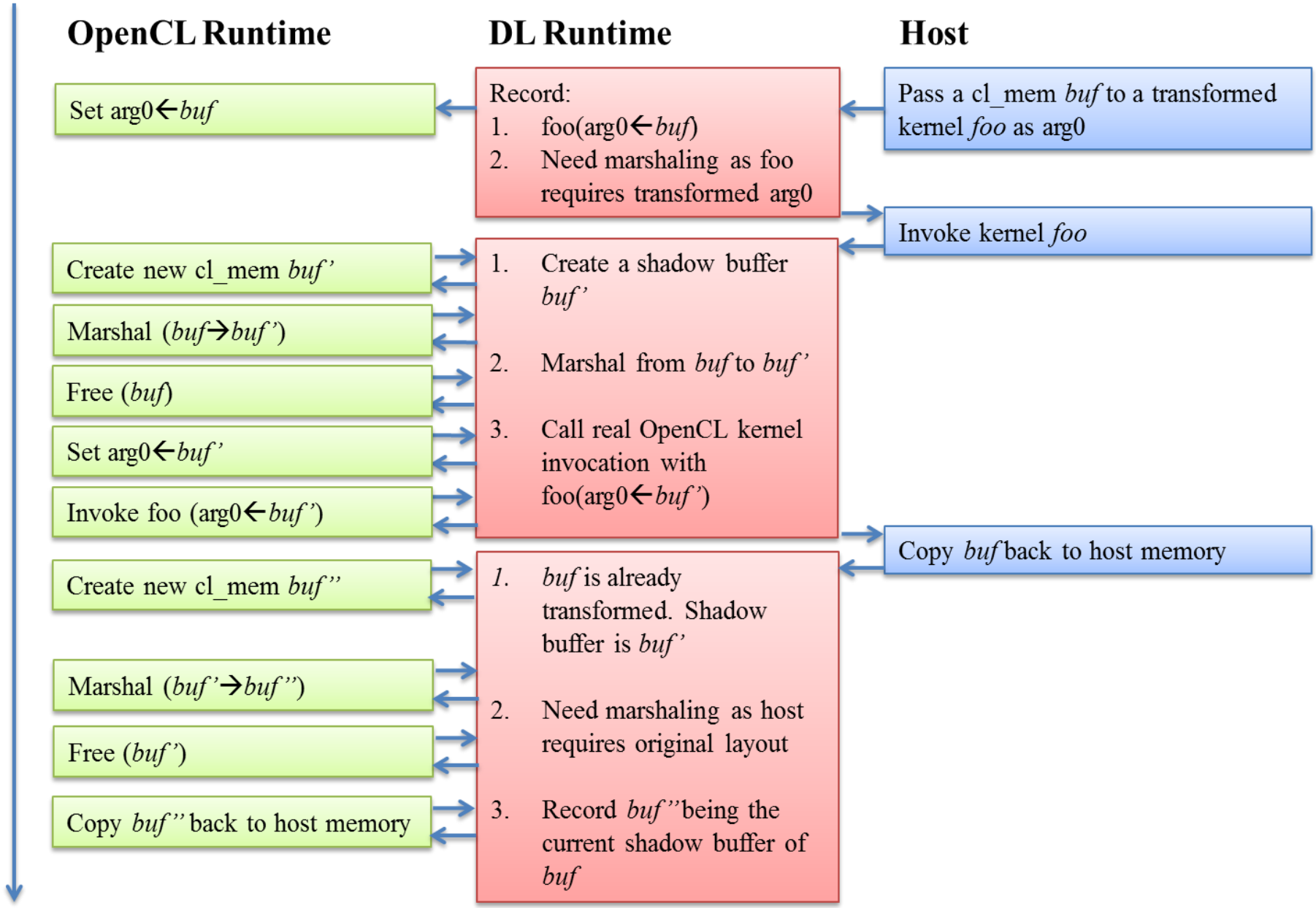


Structure of Array: [e][z][y][x]



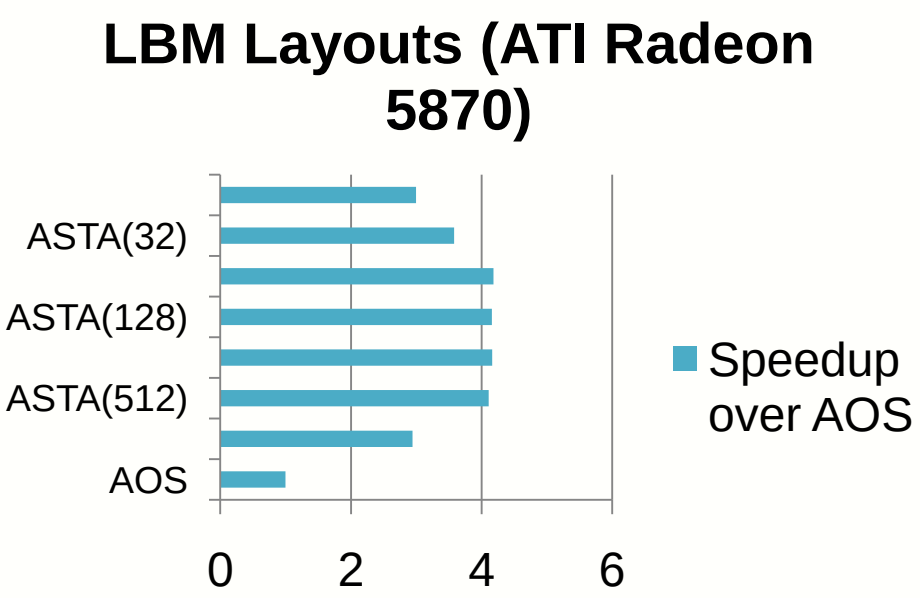
 Array of Structure of Tiled Array (ASTA) $[z][y_{31:4}][x_{31:4}][e][y_{3:0}][x_{3:0}$

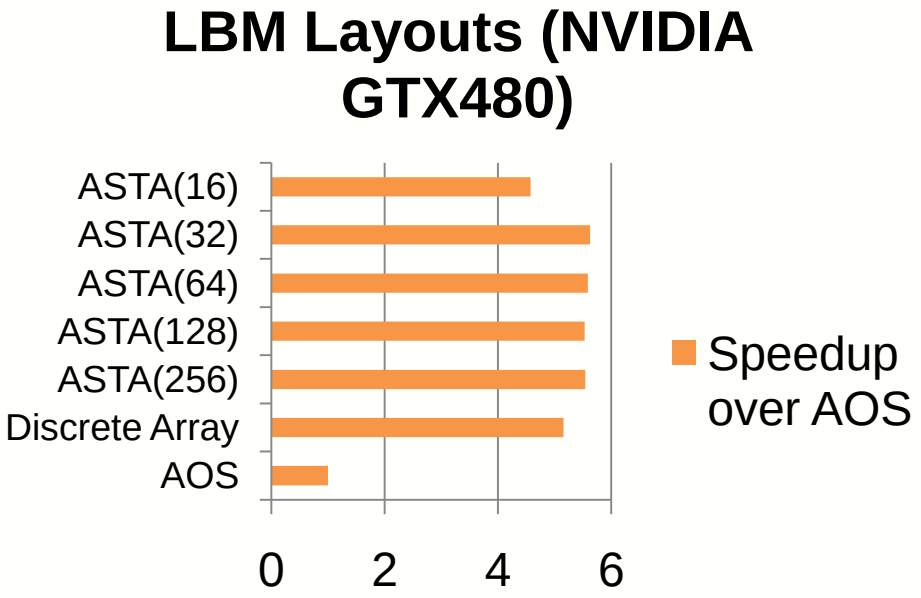
DL for OpenCL



UIUC/MCW solution ASTA

- Array-of-Structure-of-Tiled-Arrays: preserving locality while gaining coalesced memory access
  - A[x].foo → A[x/4].foo[x%4] for ASTA(4)





multicorewareinc.com/index.php?option=com_content&view=article&id=26&Itemid=30

MULTICORE WARE

Company | Technology | Products | News | Contact Us | SEARCH

multicorewareinc.com

Download button at bottom

MulticoreWare Tools

Increasing Programmer Productivity and Performance Portability across Compute Platforms

The MulticoreWare tools framework drastically reduces the number of code versions and the time spent in Performance Optimization efforts, by providing key performance tracking and analysis tools coupled with micro-architecture optimized compile-time libraries. As a result, there is no need to have hand-optimized versions for each platform.

CUDA Code | OpenCL Code | Pyon Code | DSL Code

Login

Email

Password

Remember me

LOGIN

Forgot login?

Register

Services List

Conclusion and Outlook

- Petascale and Exascale intellectual challenges
  - Scaling to large processor count with limited interconnect bandwidth
  - Effective use of massively parallel throughput oriented processors

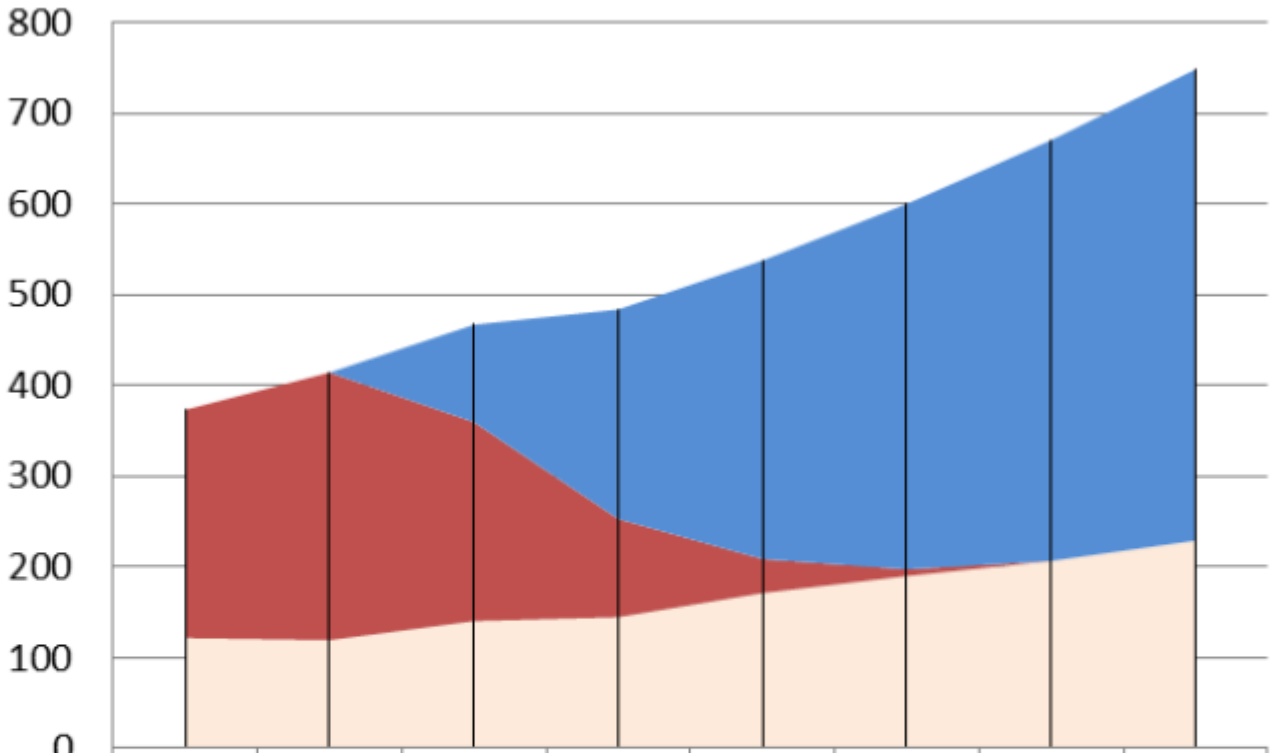
- There is a critical need for scalable kernels
  - Algorithm design for scalable kernel libraries
  - Seamless use of kernels from major languages
  - Productivity tools for kernel development and deployment

THANK YOU!

GTC Asia, Beijing, 2011

Both Fusion and Discrete GPU Markets are Growing

Courtesy: Jon Peddie



	2008	2009	2010	2011	2012	2013	2014	2015
HPU & EPG, CAGR 2010-2015: 37.11%	-	-	107.07	230.65	328.63	402.37	463.29	518.86
IGP, CAGR 2010-2015: -88%	251.30	294.73	219.30	108.14	37.57	7.36	0.19	0.01
Discrete GPU, CAGR 2010-2015: 10.28%	121.88	119.52	140.51	144.91	171.46	190.22	206.76	229.22