

# Many-Core GPU Computing

## Current Victories and Coming Battles

**Wen-mei Hwu**

University of Illinois, Urbana-Champaign



# Agenda

- Context and Many-core GPU Usage Patterns
- Current Victories
- Coming Battles
- Conclusion and Outlook

# Setting the Context

The battles are between computing community and challenges in application strong scaling

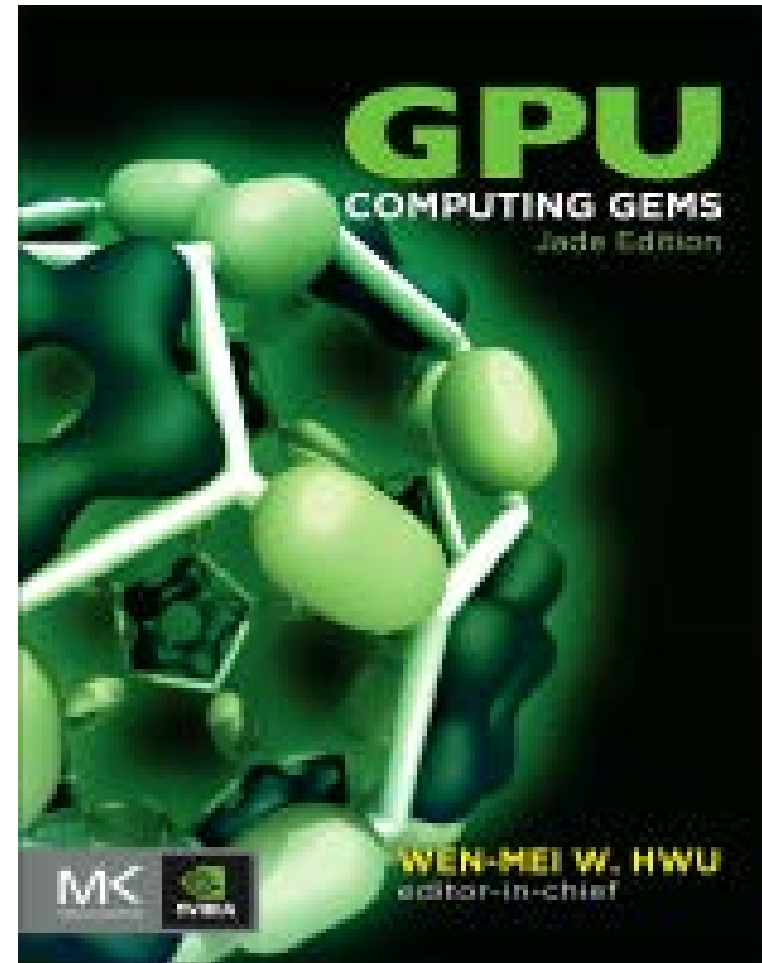
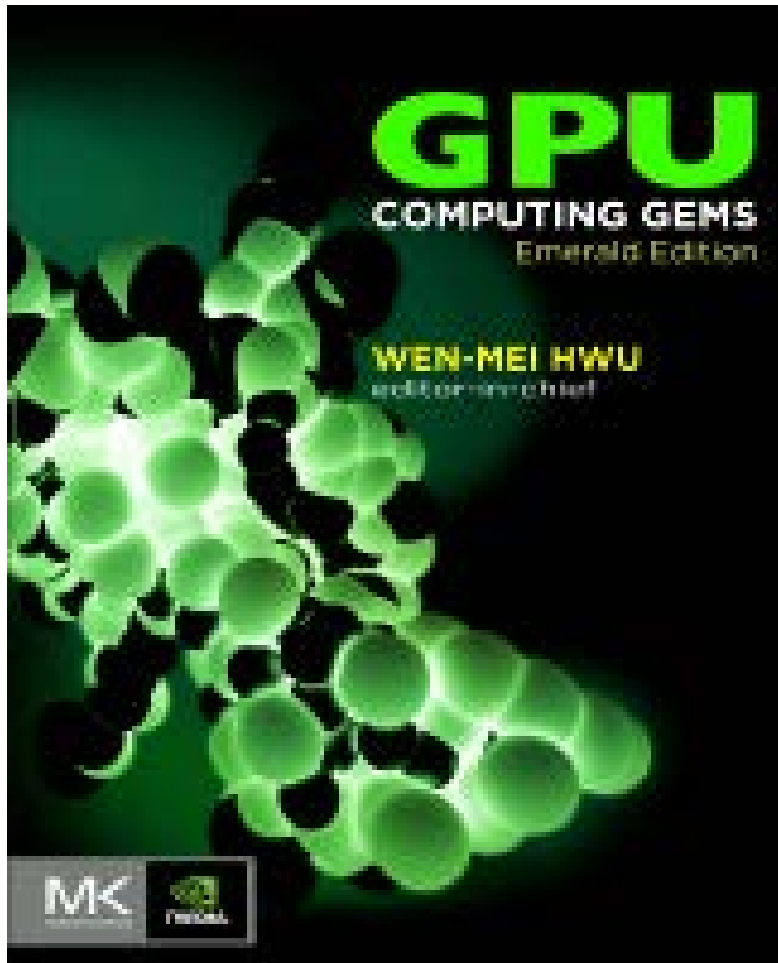
Not between CPUs and GPUs

Not between vendors

The GPU computing community are on the frontline today.



# GPU computing is catching on.



# GPU Computing Gems

## 280 Submissions, 90 chapters

Financial  
Analysis

Scientific  
Simulation

Engineering  
Simulation

Data  
Intensive  
Analytics

Medical  
Imaging

Digital  
Audio  
Processing

Digital  
Video  
Processing

Computer  
Vision

Biomedical  
Informatics

Electronic  
Design  
Automation

Statistical  
Modeling

Ray  
Tracing  
Rendering

Interactive  
Physics

Numerical  
Methods



ILLINOIS

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

TM

# A Common GPU Usage Pattern

- Use GPUs to accelerate the most time-consuming aspects of a computational problem
  - Kernels in CUDA or OpenCL
  - Refactor host code to better support kernels and data transfer
  - Convolution filtering (e.g. bilateral Gaussian filters), De Novo gene assembly, etc.
  - ...
- Rethink the domain problem



# CURRENT VICTORIES

HPC applications



# NAMD Released GPU Features and Future Plans (100,000 users)

## NAMD 2.8

- CUDA features supported - full electrostatics with PME and most simulation features (not alchemical methods), NBFIX parameters
- 100M-atom capability functional on CUDA

## NAMD 2.9

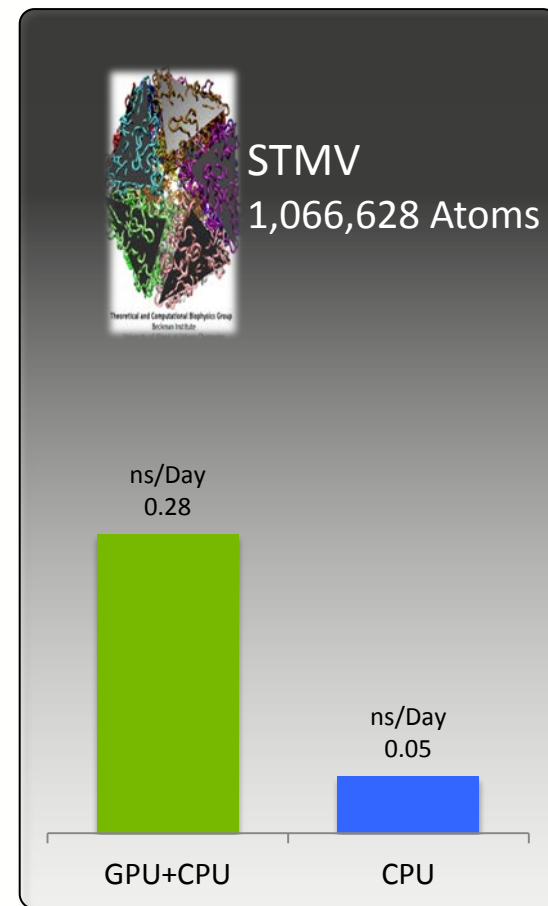
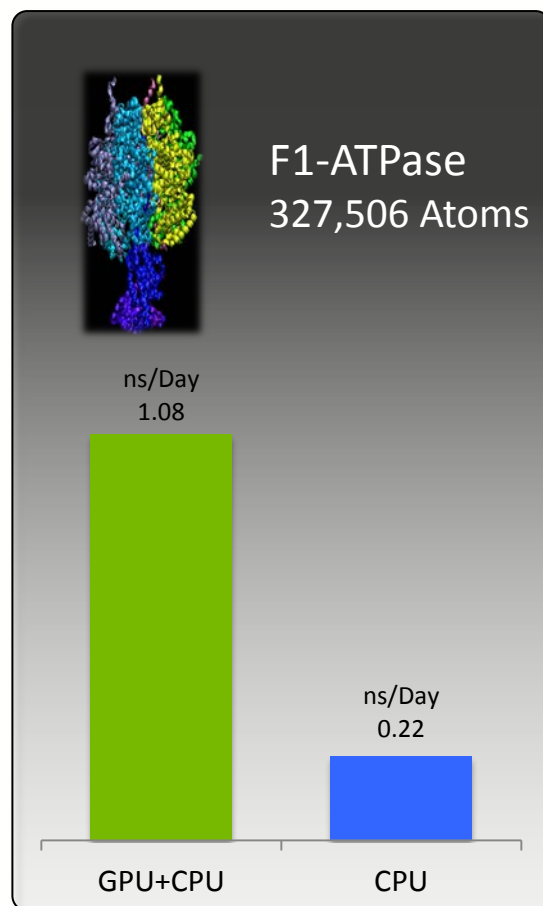
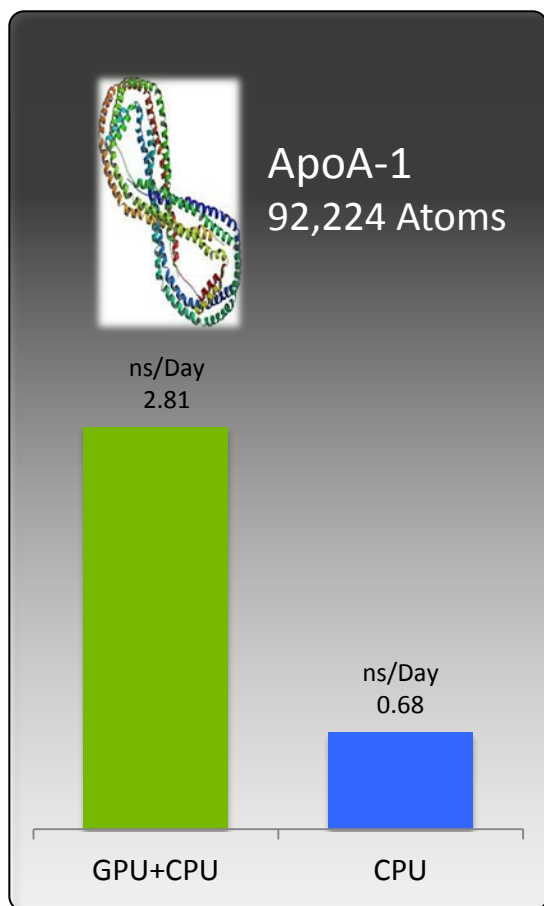
- Alchemical free energy perturbation
- Locally enhanced sampling
- Methods that modify nonbonded interactions for small sets of atoms
- New multi-level summary method (MSM)

## Longer Term

- Specialized methods such as Lowe-Anderson thermostat, Go potentials, and tabulated nonbonded interactions
- Various performance improvements, including reduce CPU-side performance bottlenecks such as shifting various calculations to GPUs



# Pushing Limits of Innovation with NAMD

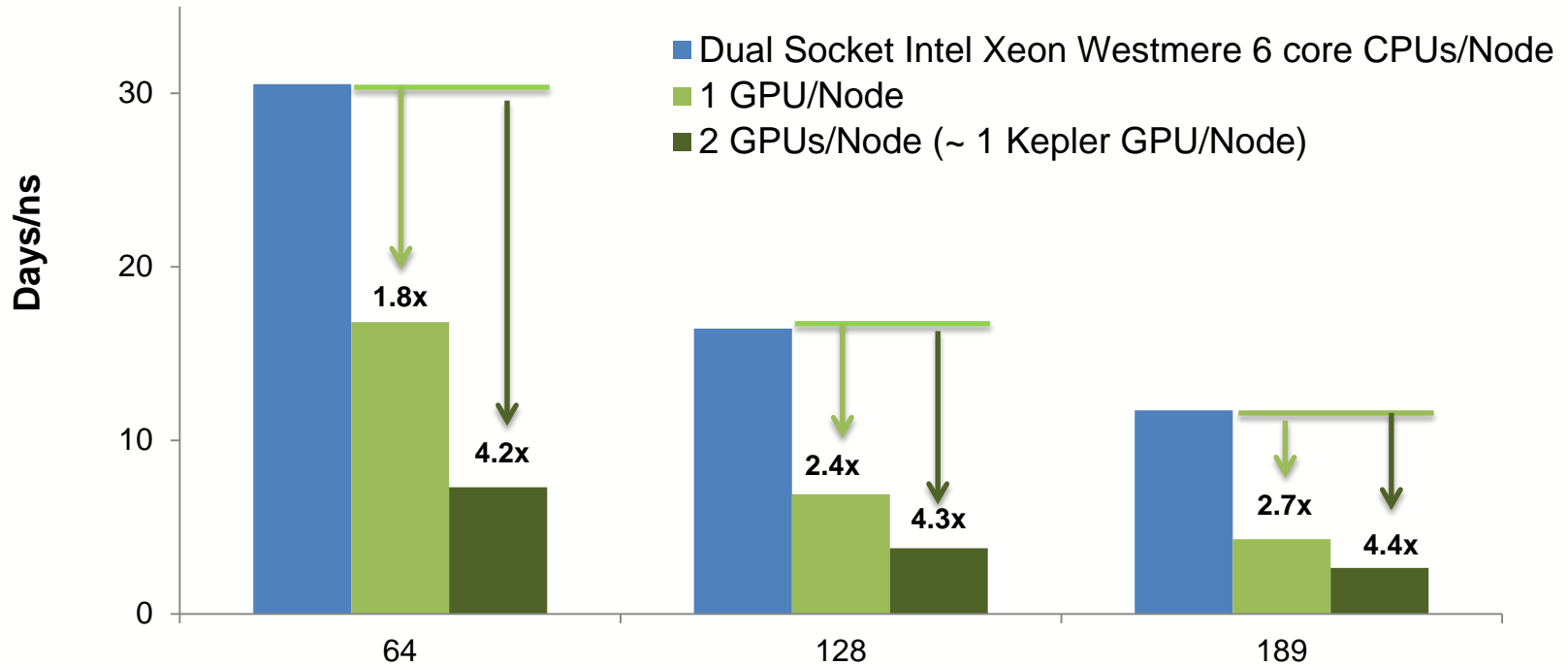


Test Platform: 1 Node, Dual Tesla M2070 GPU (6GB), Dual Intel 4-core Xeon (2.4 GHz), NAMD 2.8, CUDA 4.0, ECC On.

Visit [www.nvidia.com/simcluster](http://www.nvidia.com/simcluster) for more information on speed up results, configuration and test models.

# GPU Scaling on NAMD

## STMV Benchmark on Tsubame 2.0



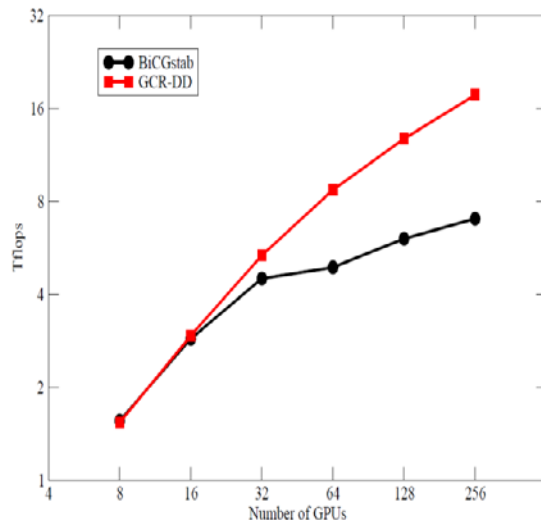
# of Nodes NAMD Benchmark on Tsubame 2.0, 9/7/2011  
100STMV, ibverbs-smp

- NAMD run on Tsubame 2.0 from 64 to 189 nodes.
- Using 1 or 2 Fermi GPUs per node

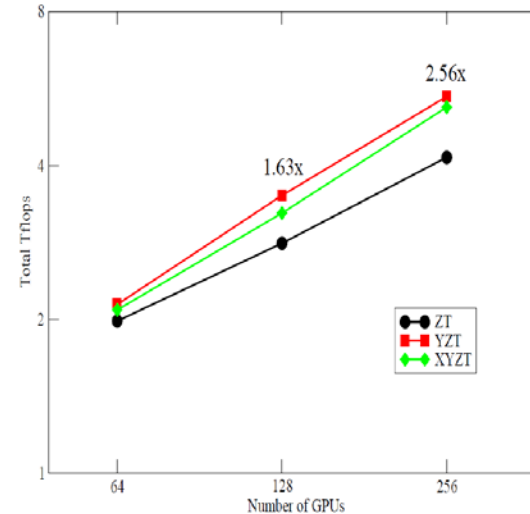
# QCD Strong Scaling using GPUs

- **General Problem** : as core counts increase, the ratio of communication to local computation tends to grow. For a sufficient number of cores, the problem becomes communications bound (vs. computation)
- **Solution** : solvers that minimize communication, such as “domain-decomposition” solvers. An additive Schwarz domain-decomposed preconditioner with a Generalized Conjugate Residual solver (GCR-DD) successfully demonstrates strong scaling
- **Results** : Strong scaling to 256 GPUs on  $32^3 \times 256$  lattice in Chroma (Wilson-clover fermions)
- **Results** : Strong scaling to 256 GPUs on  $64^3 \times 256$  lattice in MILC (improved staggered fermions)

Sustained strong-scaling performance in Chroma 3.41.0 using Schwarz generalized conjugate residual solver (GCR-DD). BiCGstab is the reference Krylov solver.



Sustained strong-scaling performance in MILC 7.6.3 using mixed-precision conjugate gradient solver (parallelized along multiple dimensions)



\* Guochun Shi (NCSA), Bálint Joó (Jefferson Labs), Ron Babich (BU), Mike Clark (Harvard), Rich Brower (BU), Steve Gottlieb (Indiana), “Scaling Lattice QCD beyond 100 GPUs,” SC11, ACM (Nov 2011)



# USQCD Software GPU Roadmap

- **ETA September 2011** : exploiting GPU Direct (QUDA 0.4.0 doesn't currently support peer-to-peer transfers to minimize inter-GPU communication).
- **ETA Q3/Q4 2011** : Multi-GPU DWF fermions.
- **ETA fall 2011** : Adaptive multigrid (MG), expected to deliver  $O(10)$ -fold speedup over current solvers.
- **ETA Q4 2011 / Q1 2012** : Refinement of domain-decomposition algorithms. Currently simple block Jacobi. Expect significant speedup from overlapping blocks, multiplicative Schwarz (e.g. block Gauss-Seidel).
- **Active R&D 2012** : Exploitation of cache locality (e.g. more efficient use of shared memory to reduce memory traffic). Better scaling for GPU cores vs. GPU memory bandwidth.
- **ETA 2012** : Full Hybrid Monte Carlo (e.g. gauge generation) on GPUs. Includes support for high-order symplectic symmetric integrators which improves the volume scaling from HMC, which will result in substantial computational cost reduction at large volumes.
- **Beta 1H 2012** : Complete deployment of QCD applications (e.g. Chroma) on GPUs by implementing the domain specific language (QDP++) in CUDA. Currently pre-alpha, Jlab R&D (Jie Chen), Frank Winter (Edinburgh).
- **R&D 2012, Deployment 2013** : Combine HMC and MG on GPUs.



# CURRENT VICTORIES

Available Kernels



# 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  $\mathbf{Ax} = \mathbf{d}$ , where  $A$  is an  $n$ -by- $n$  tridiagonal matrix and  $d$  is an  $n$ -element vector

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

# Thomas Algorithm

- Special two-way Gaussian elimination

**Backward substitution**

$$A = \begin{bmatrix} b_1 & \cancel{c_1} & & & \\ \cancel{a_2} & b_2 & \cancel{c_2} & 0 & \\ & \ddots & \ddots & \ddots & \\ & 0 & \cancel{a_{n-1}} & b_{n-1} & \cancel{c_{n-1}} \\ & & & \cancel{a_n} & b_n \end{bmatrix}$$

**Forward reduction**

# Parallel Cyclic Reduction(PCR)

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

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



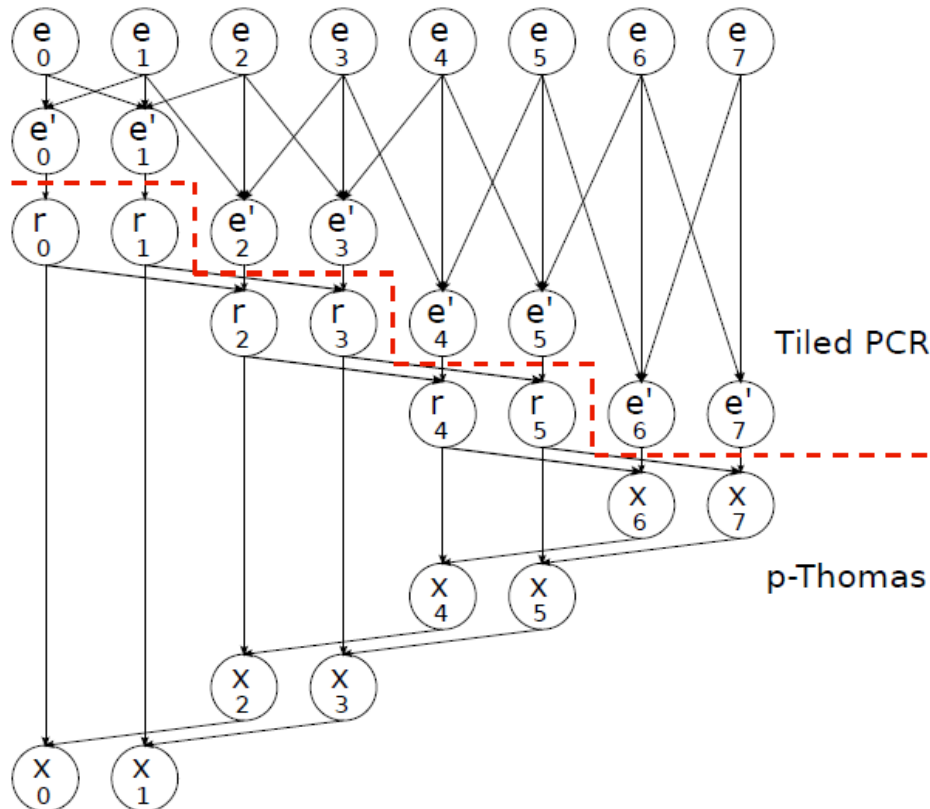
# Summary of Previous Approaches

	Complexity	Number of operations	Number of processing steps with n-parallelism machine
Thomas	$O(n)$	$2n$	$2n$
CR	$O(n)$	$2.7 * 2n$	$2\log n - 1$
PCR	$O(n \log n)$	$12 n \log n$	$\log n$



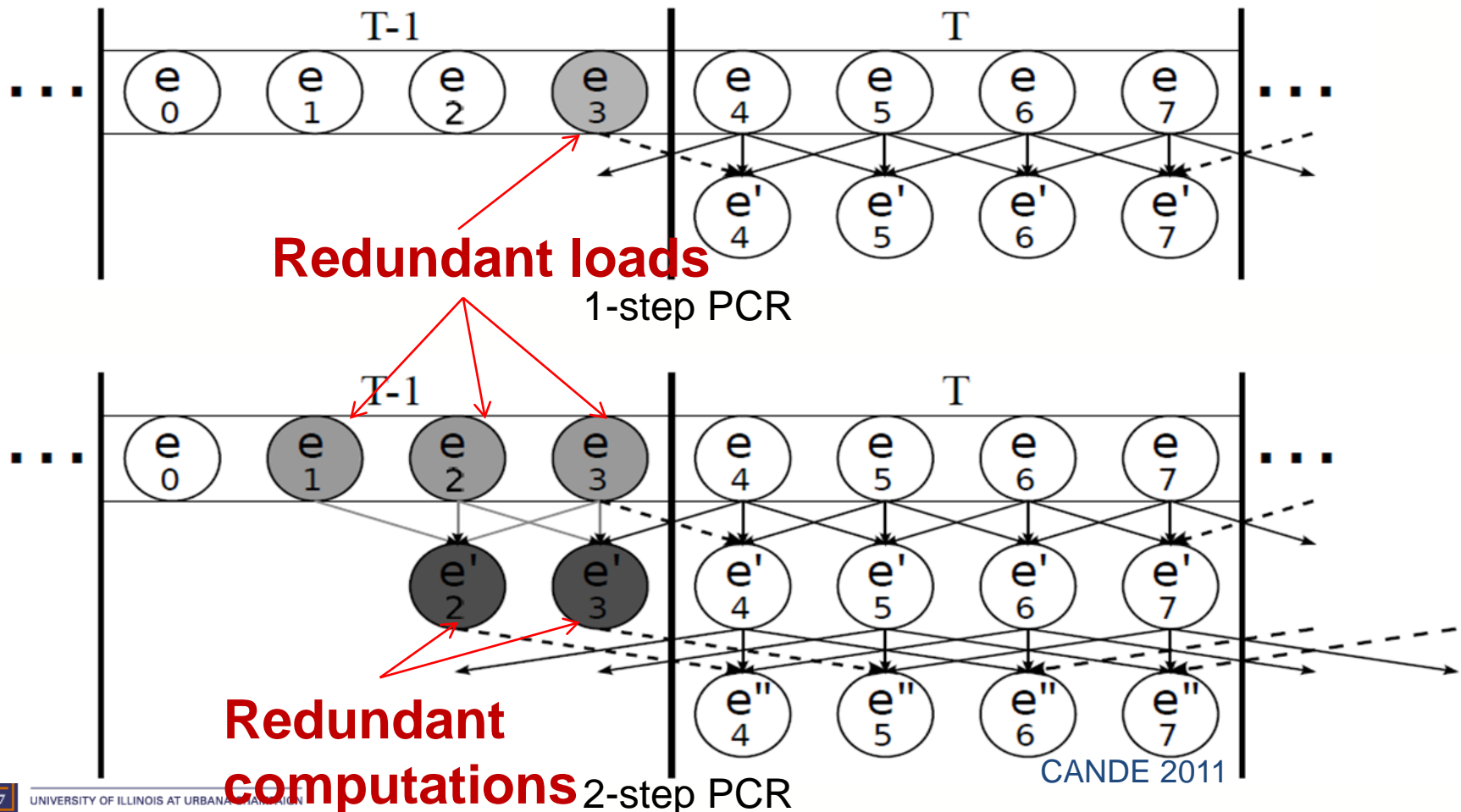
# Scalable Method

- A hybrid of PCR and parallel Thomas
  - PCR as parallelism excavating frontend
  - P-Thomas as an efficient and parallel solver

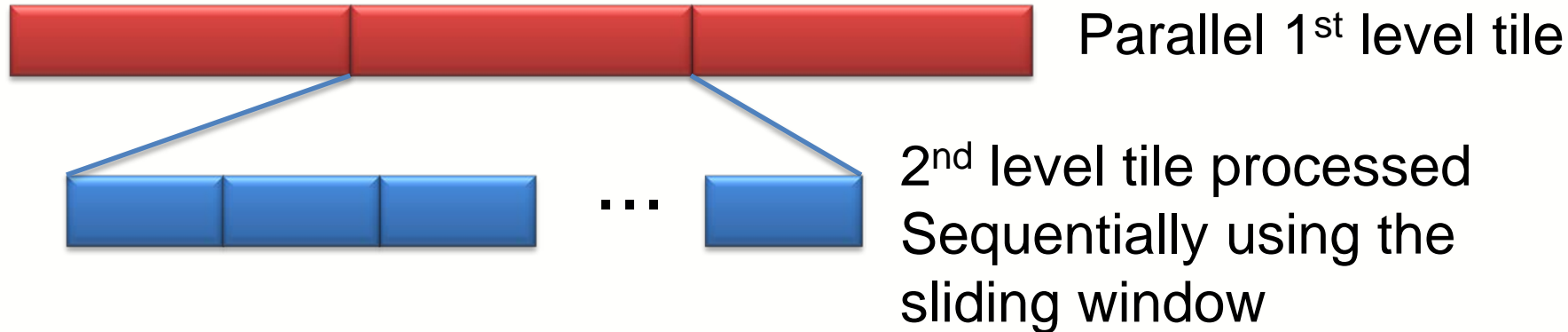


# Parallelization of Tiled PCR (Owens)

- Exponential growth of halo elements and subsequent reductions

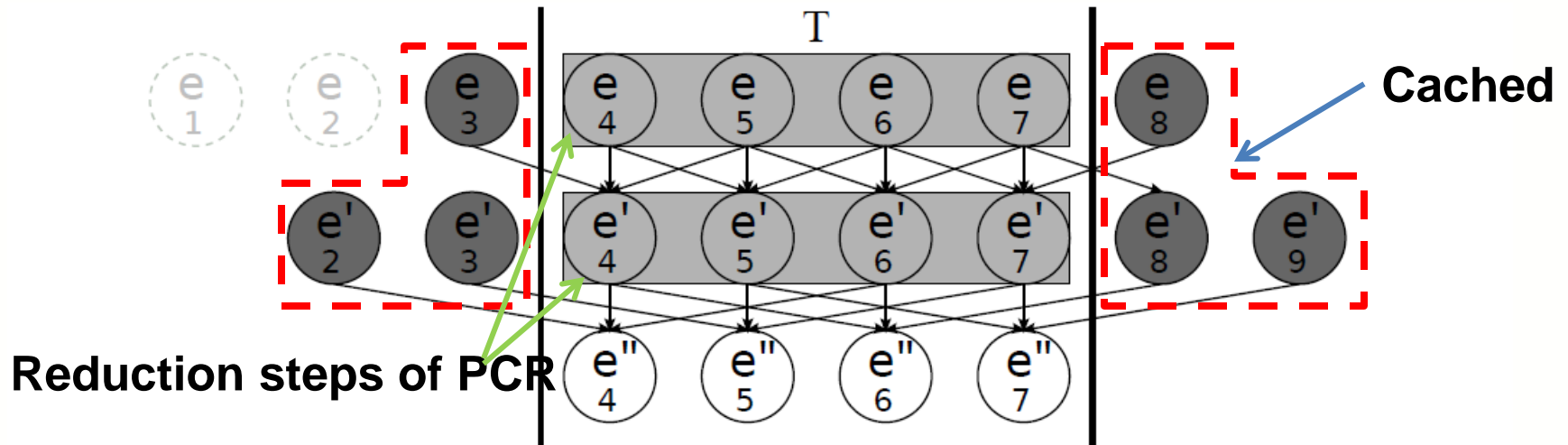


# Hierarchical Tiling

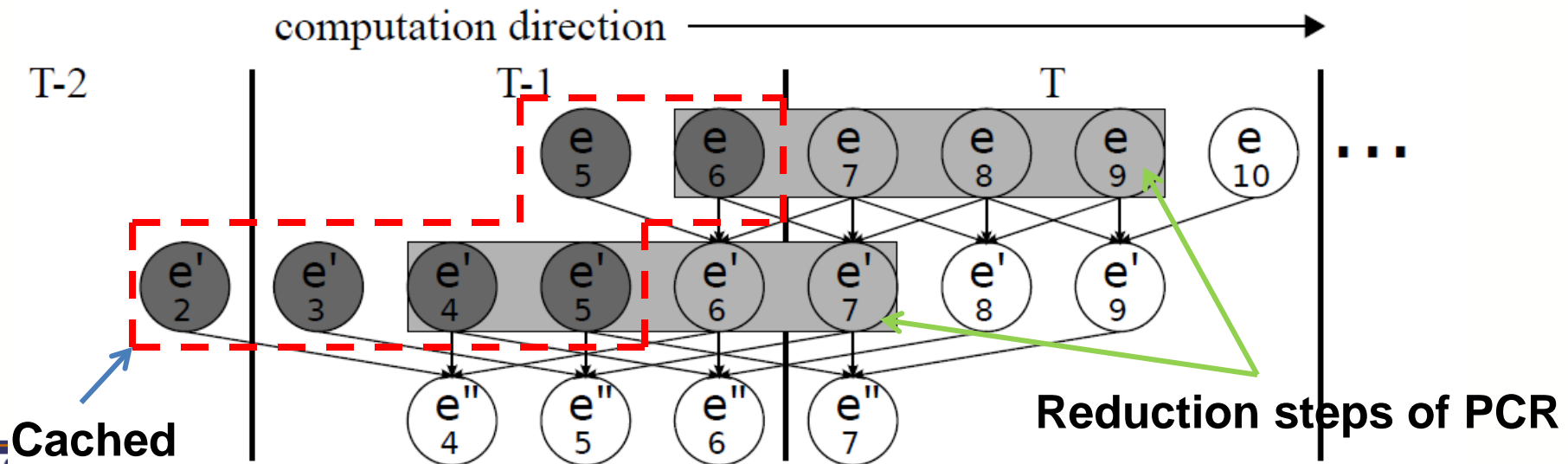


Mapping onto GPU	
1 <sup>st</sup> level tile	One per thread block
2 <sup>nd</sup> level tile	Collaborative streaming within thread block
2 <sup>nd</sup> level tile buffer	Shared memory

# Tiled PCR using Sliding Window



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

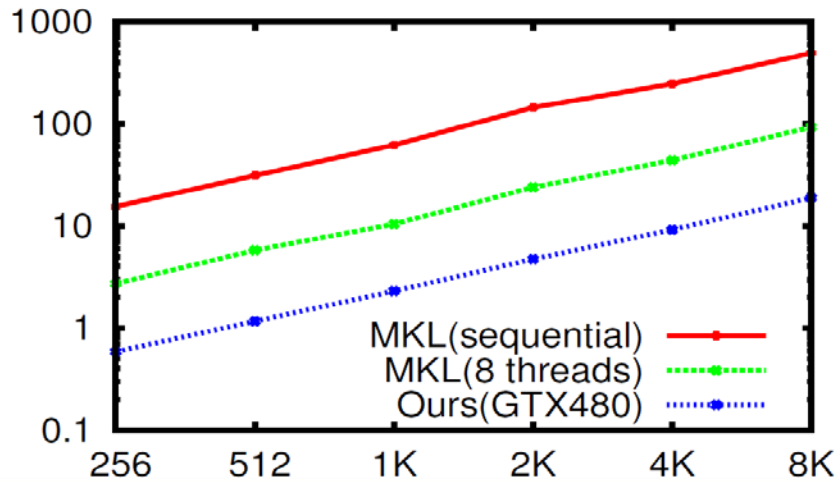


(b) 2-step PCR using cached dependency from left-hand side

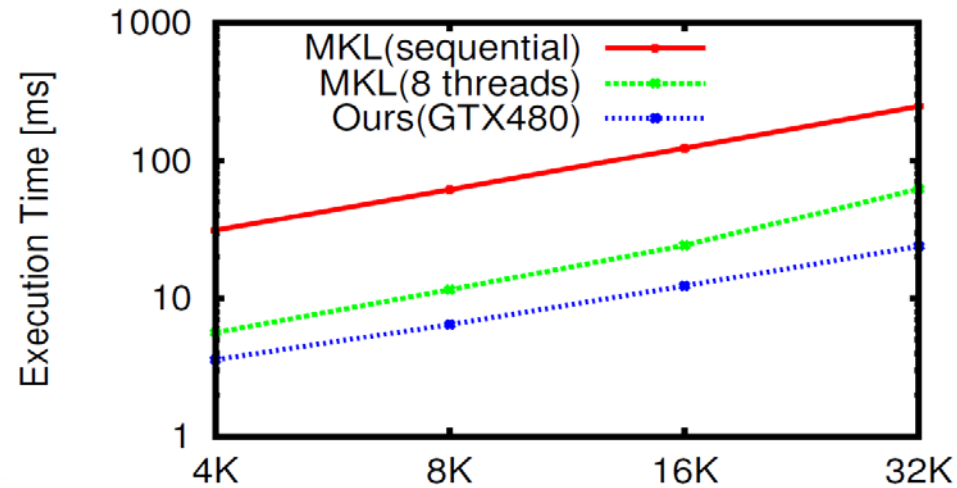
# Scalable in size and # systems

Execution Time [ms]

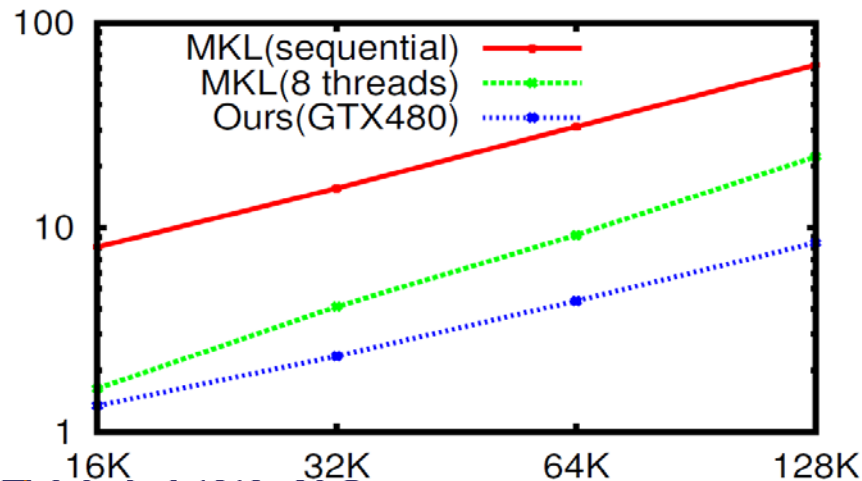
M=2048



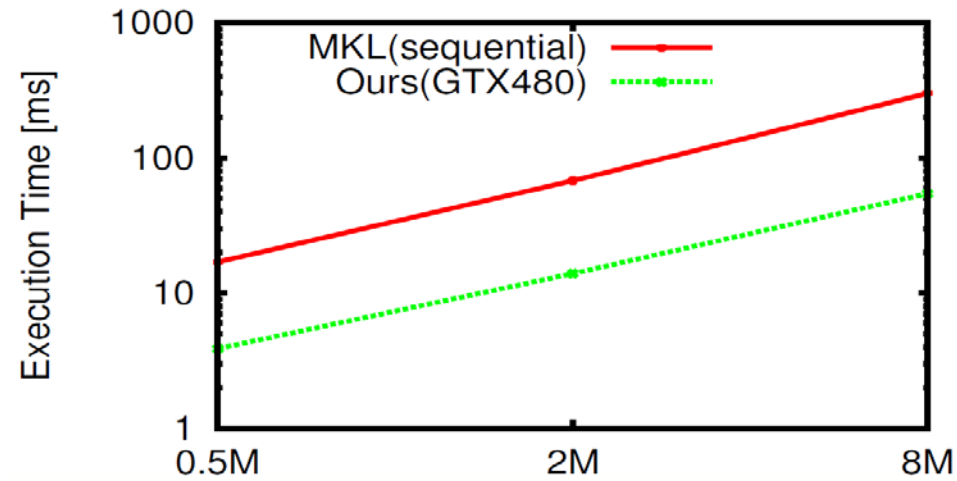
M=256



M=16



M=1



# Using GPU kernels is also becoming easier.

- Python PyCUDA interface
- MatLab Jacket and Mathematica interfaces
- Thrust C++ interface for CUDA
- Microsoft C++ Accelerated Massive Parallelism
- Java does not currently have an easy way of using CUDA or OpenCL kernels
  - Especially for Android platforms
  - JavaCUDA project in Illinois
  - Renderscript from Google
- GMAC for data sharing between host and device



# COMING BATTLES

Hardware trends

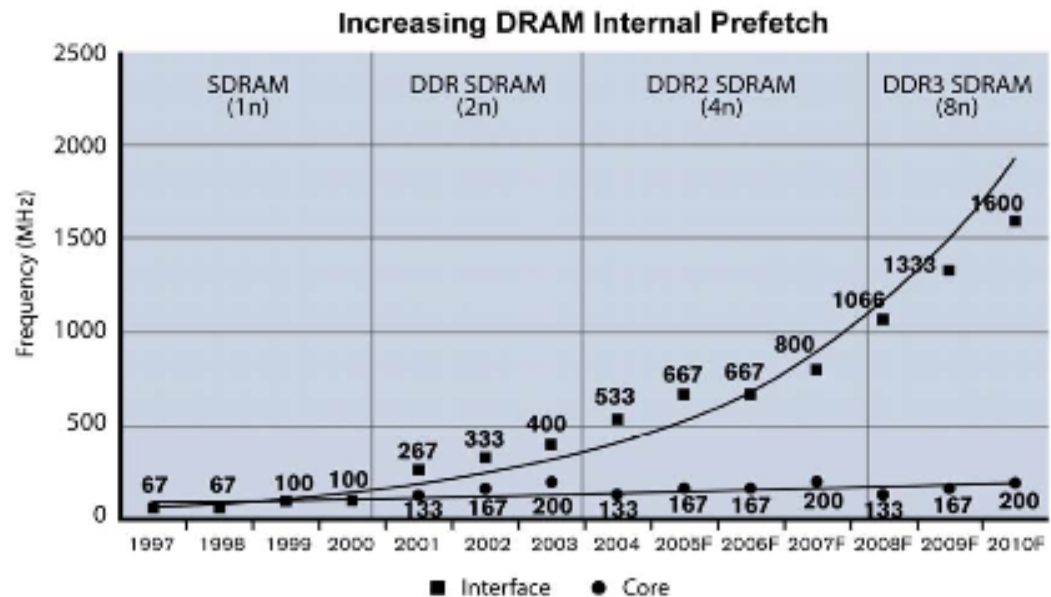
# Important GPU Architecture Trends

- CPU/CPU fusion architectures
  - For reduced part count and data movement
  - Reduced hand-off granularity
  - Larger GPU accessible memory
- Emphasis on energy efficiency
  - By reducing data movement and control flow overhead
- More general forms of parallelism
  - To accommodate algorithm and locality needs
- Even higher degree of SIMD
  - Due to increasing width of memory interface



# DRAM Trends

- Increased minimum DRAM burst size to meet the increasing interface/core clock gap
  - 8 in GDDR5
- # of banks per chip also increases to increase parallelism
  - Usually 4 in GDDR2
  - 4 or 8 in GDDR3
  - Usually 8 in GDDR4/5
  - More
    - Intel/Micron Hybrid Memory Cube (HMC)



Courtesy: Micron, Challenges and Solutions for Future Main Memory, 2009

CANDE 2011



# DRAM Trend Implications

- More DRAM banks:
  - Need well-distributed DRAM requests (high bank-level parallelism)
- Wider bursts
  - Size of DRAM request  $\geq$  min burst size (high SIMD/SIMT parallelism)
- Holistic approaches to meet the challenge
  - From algorithm-level transformation to micro-architecture
  - E.g. data layout transformation in applications + thread scheduling in runtime + memory coalescing hardware in GPUs



# COMING BATTLES

Scalable Kernels

# 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



# Example of kernel Needs

- Sparse LU factorization, Cholesky factorization, Triangular, and related inverse solvers
- Sparse eigen solvers and related eigen analysis
- Graph partitioning (Metis)
- ...

# Four Challenges

- Computations with no known scalable parallel algorithms
  - Shortest path, Delaunay triangulation, ...
- Data distributions that cause catastrophic load imbalance in parallel algorithms
  - Scale-free graphs, MRI compressed sensing
- Computations that have little data reuse
  - Matrix vector multiplication, ...
- Algorithm optimizations that are hard and labor intensive
  - Locality and regularization transformations



# Example - Dynamic Data Extraction

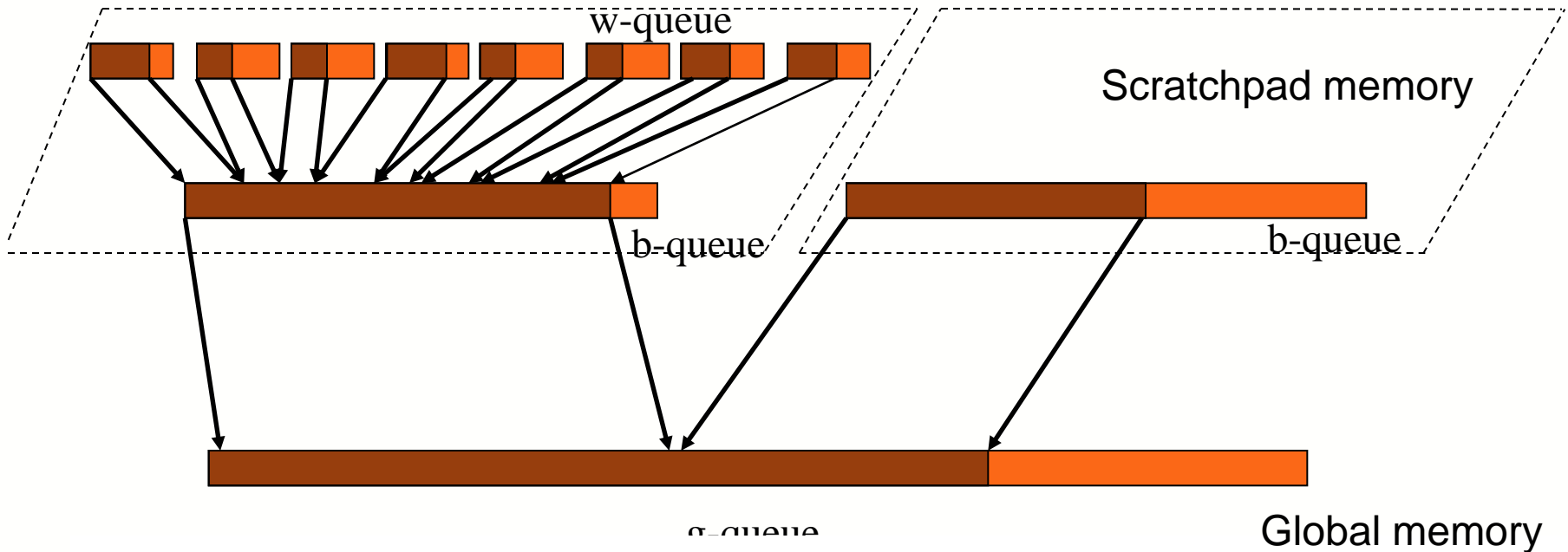
- The data to be processed in each phase of computation need to be dynamically determined and extracted from a bulk data structure
  - Harder to organize for massively parallel access
- Graph algorithms are popular examples that deal with dynamic data extraction
  - Widely used in EDA and large scale optimizations
  - Breadth-First Search (BFS) as an example

# Dynamic Data Extraction using Queues

- Input data extraction is done by many-threads in parallel
  - Must have a systematic way to avoid contention in assembling extracted data
- Obvious approach is queues with privatization
  - Replicate queues to reduce contention
  - Combine queues with concatenation
  - Works only when global order does not matter (queue insertion is commutative)



# Three-level Queue Hierarchy



- At the end of each the kernel
  - Threads cooperate to assemble b-queue
  - Multiple threads collaborate to merge b-queue contents into g-queue
  - Fast atomic operation helps

# Hierarchical Kernels

- Customize kernels based on the size of frontiers.
- Use fast barrier synchronization when the frontier is small.



One-level parallel propagation



Kernel 1: Intra-block Sync.

Kernel 2: Inter-block Sync.

Kernel 3: Kernel re-launch

# Experimental Evaluation

- CPU implementation
  - The classical BFS algorithm ( $O(V+E)$ )
  - dual socket dual core 2.4 Ghz Opteron processor with 8GB memory.
- GPU: NVIDIA GeForce GTX280
- Benchmarks
  - Near-regular graphs (degree = 6) up to 10X
  - Real world graphs (avg. degree = 2, max degree = 9) up to 5X
  - Scale free graphs, slow down
    - 0.1% of the vertices: degree = 1000
    - The remaining vertices: degree = 6



# COMING BATTLES

tools



## 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);
...

```

CANDE 2011



# IMPACT Tools for Heterogeneous Parallel Programming

Early

Writing optimizable, portable, kernels

▶ **Pyon**

Shared memory model & multi-GPU copy support

▶ **GMAC**

Performance tools

▶ **ADAPT**

▶ **Thread Coarsening**

▶ **DL (Data Layout)**

▶ **MCUDA**

Higher-level Interfaces for Programmability, Portability, and Performance

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

Late



# Writing efficient code is complicated.

Tools can provide focused help  
or broad help

Planning how to execute an algorithm    Implementing the plan

- Choose data structures

- Memory allocation
- Data movement

GMAC

- Pointer operations
- Index arithmetic

Data  
Layout

Pyon

- Decompose work into tasks
- Schedule tasks to threads

MCUDA

- Kernel dimensions
- Thread ID arithmetic
- Synchronization
- Temporary data structures

Thread  
coarsening

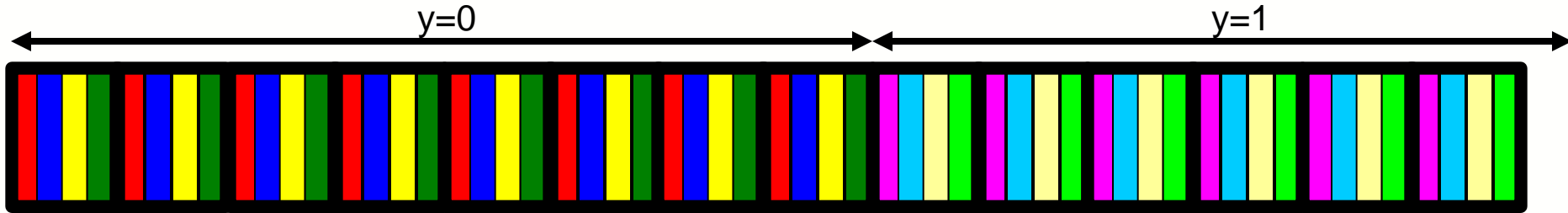


# Example - DL (Data Layout)

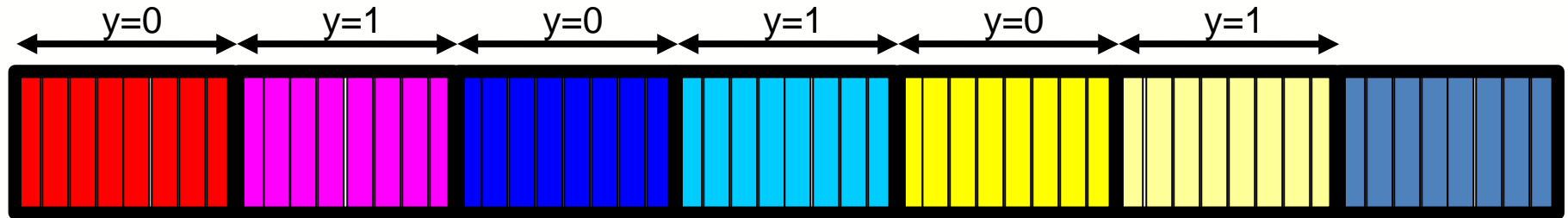
- DRAM bursts are formed differently in a heterogeneous system
  - From last level cache misses on CPUs
  - From SIMD-ized memory accesses on many-core architectures like GPUs
- Data layout transformation can mitigate the gap
  - E.g.: Array-of-structure / Discrete-arrays
  - Bridging divergent layout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



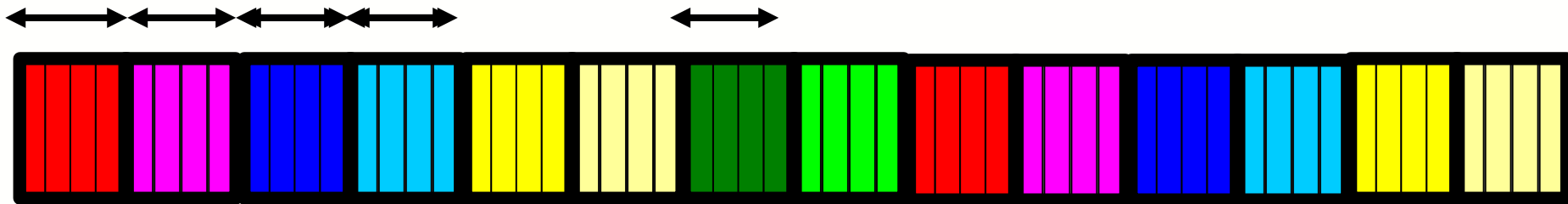
# Data Layout Alternatives



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



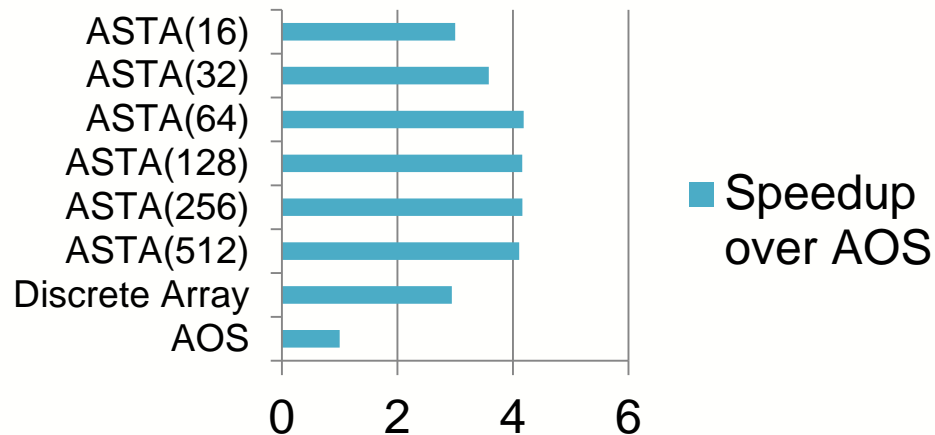
Structure of Array:  $[e][z][y][x]$



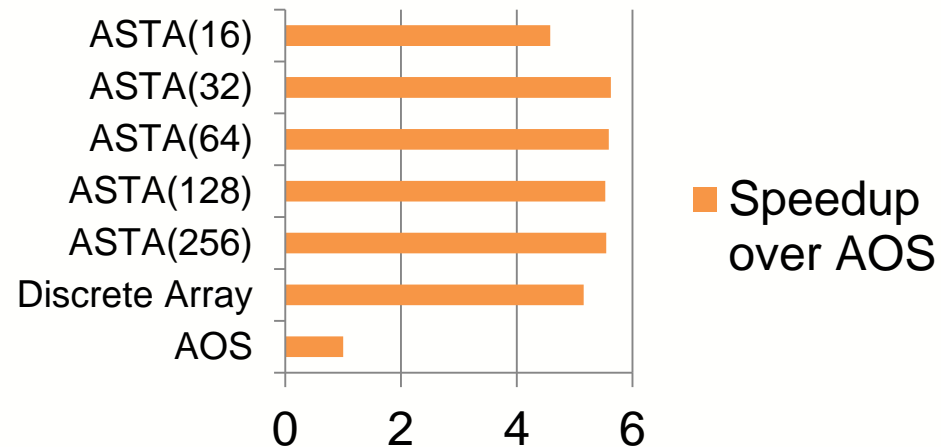
# ASTA (Sung et al)

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

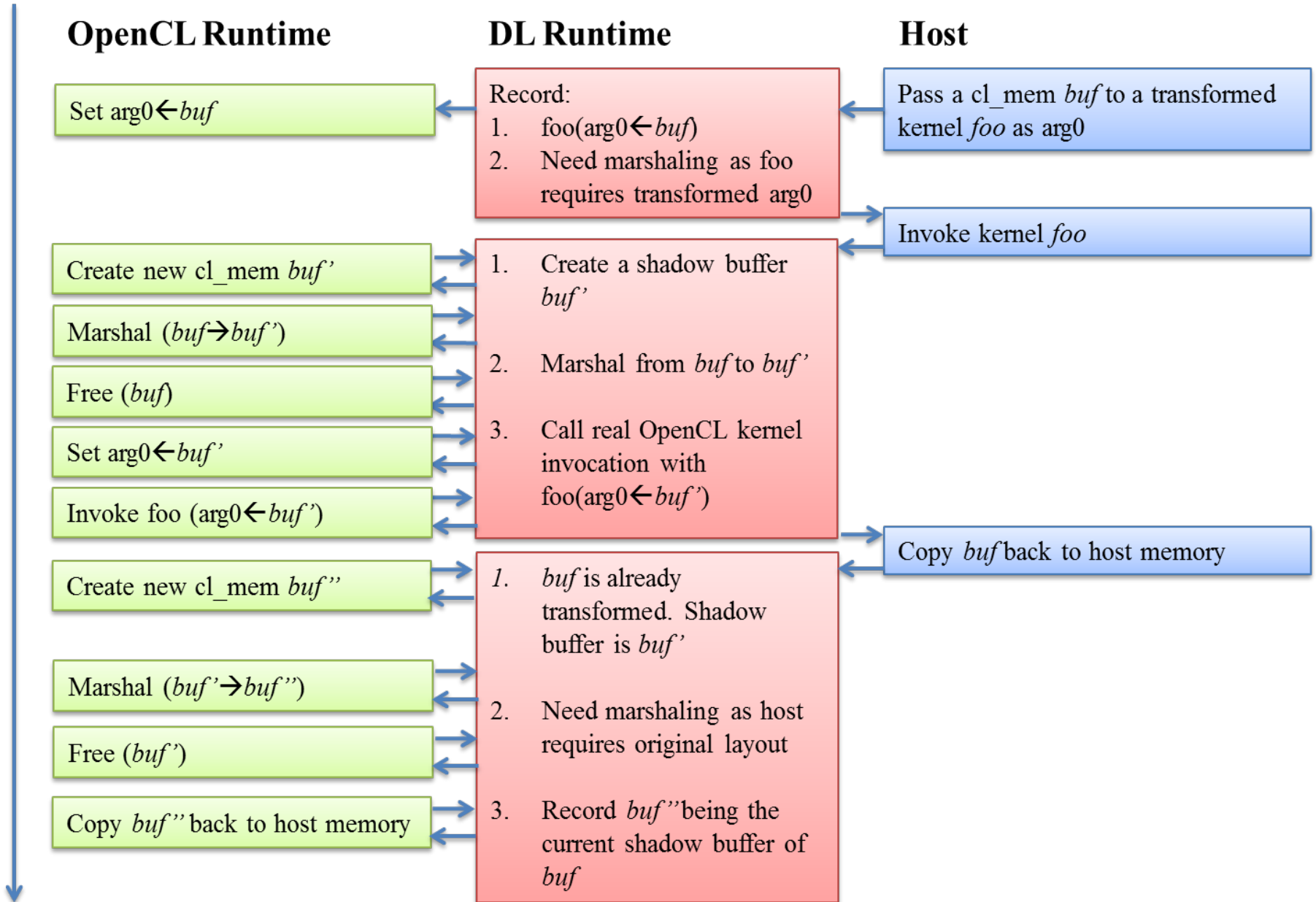
## LBM Layouts (ATI Radeon 5870)



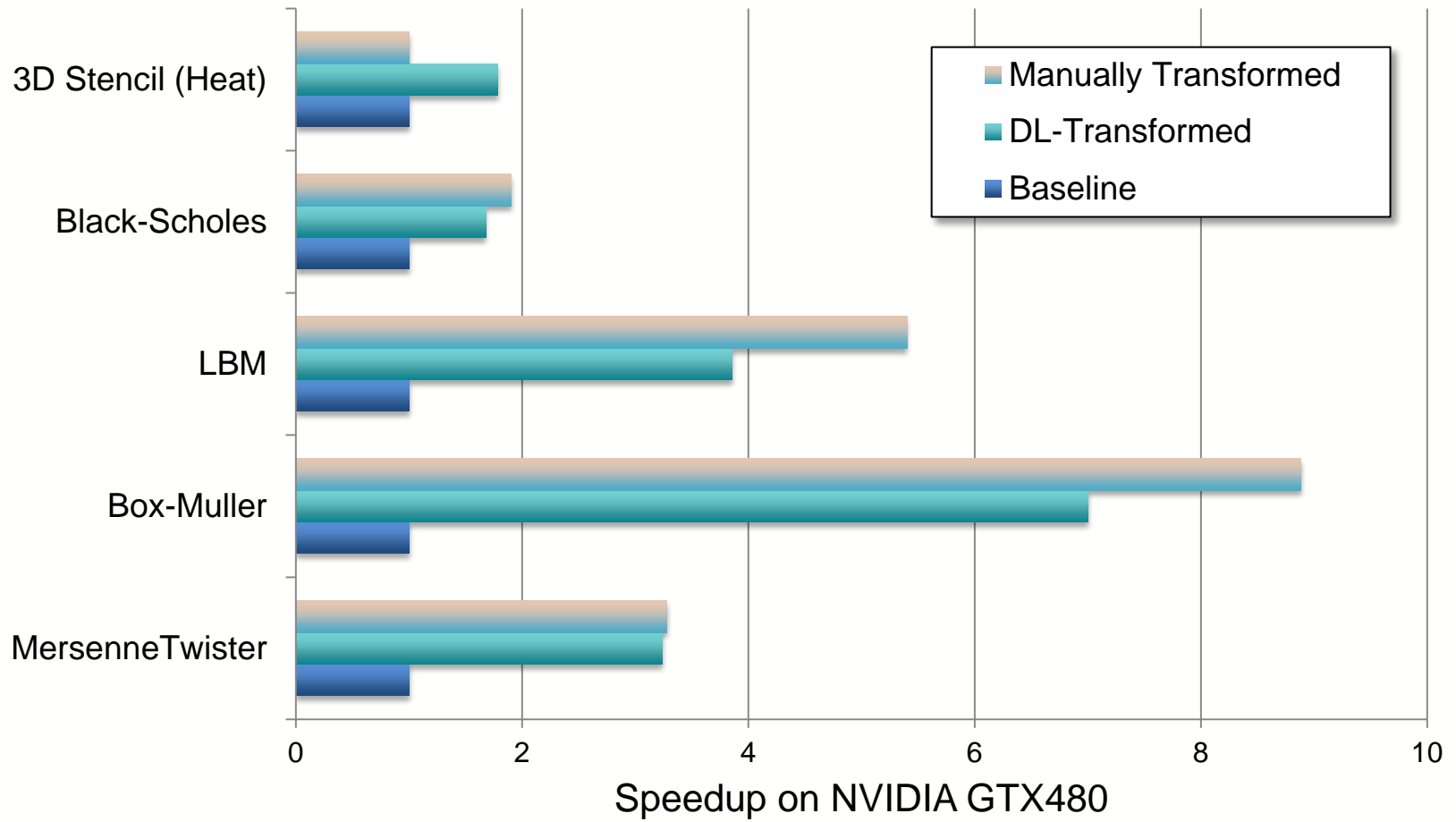
## LBM Layouts (NVIDIA GTX480)



# DL for OpenCL



# Data layout improves performance even with marshaling.



# Related Kernel Development Tools

- OpenMP Accelerator Pragmas
  - Wider use of GPU in large applications but less performance in each kernel
  - Cray and others
- Portland Group FORTAN compiler
- Intel Array Building Block (Cilk)

# Conclusion and Outlook

- We have enjoyed some victories
  - Good initial set of applications and kernels
  - Good deployment interface in major languages
  - Good initial results, educated developers
- We will face more battles
  - Energy efficiency vs. easy of programming
  - Potential fragmentation of programming interface
  - Widen the set of applications, algorithms and kernels
  - Better tools

There is always hope.

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

CANDE 2011



**THANK YOU!**



**ILLINOIS**  
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

CANDE 2011