Advanced Algorithmic Techniques for GPUs

Lecture 1: Introduction and Computational Thinking

©Wen-mei W. Hwu and David Kirk/NVIDIA,
Berkeley, January 24-25, 2011

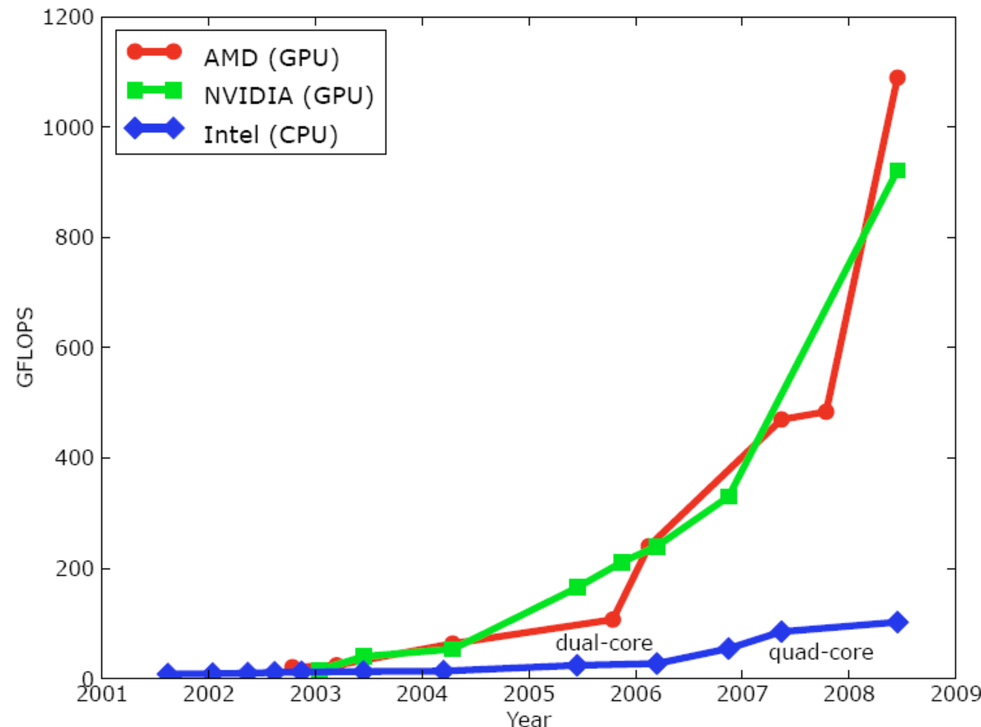
Course Objective

- To master the most commonly used algorithm techniques and computational thinking skills needed for many-core GPU programming
  - Especially the simple ones!
- In particular, to understand
  - Many-core hardware limitations and constraints
  - Desirable and undesirable computation patterns
  - Commonly used algorithm techniques to convert undesirable computation patterns into desirable ones

©Wen-mei W. Hwu and David Kirk/NVIDIA, Berkeley, January 24-25, 2011

Performance Advantage of GPUs

- An enlarging peak performance advantage:
  - Calculation: 1 TFLOPS vs. 100 GFLOPS
  - Memory Bandwidth: 100-150 GB/s vs. 32-64 GB/s

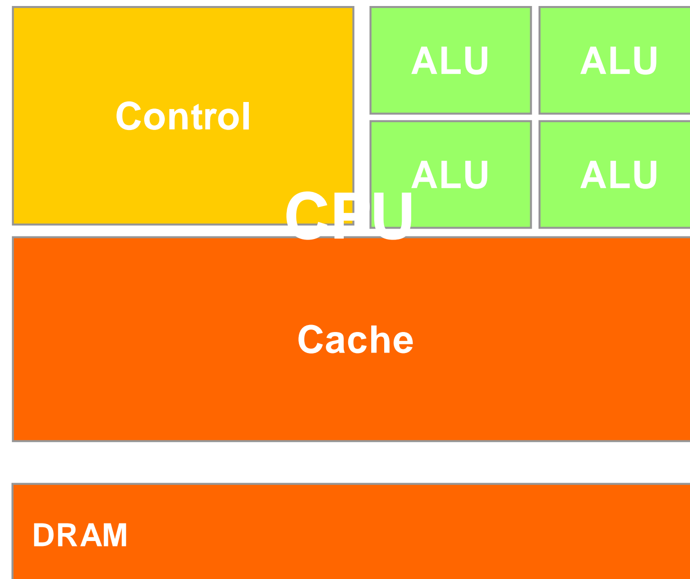


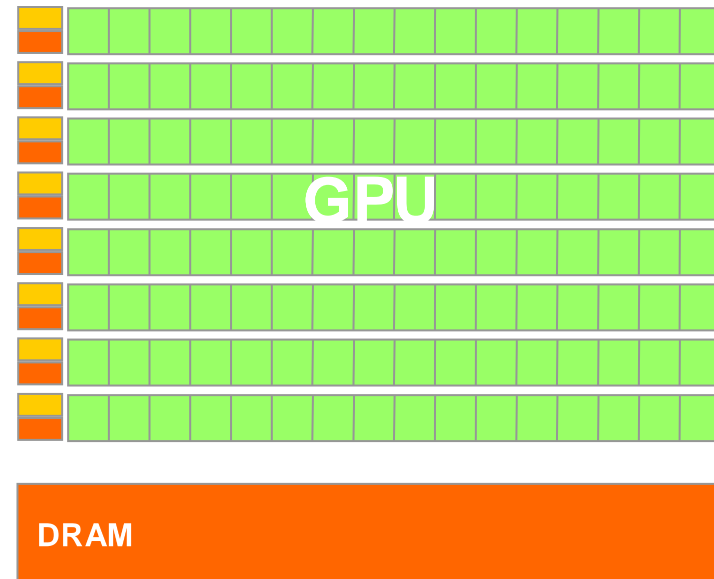
Courtesy: John Owens

  - GPU in every PC and workstation – massive volume and potential impact

©Wen-mei W. Hwu and David Kirk/NVIDIA, Berkeley, January 24-25, 2011

CPUs and GPUs have fundamentally different design philosophies.





UIUC/NCSA AC Cluster

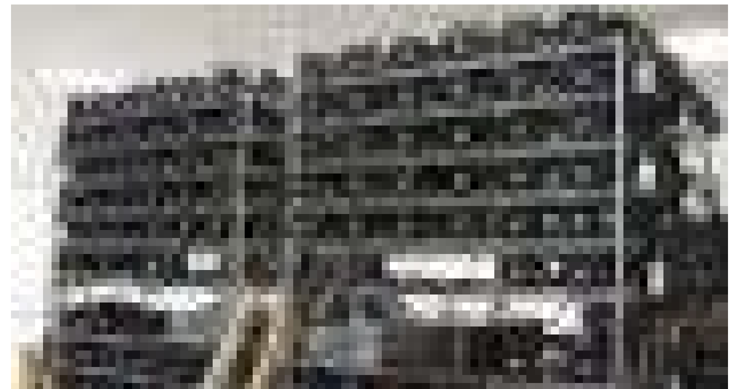
- 32 nodes
  - 4-GPU (GTX280, Tesla) nodes
  - GPUs donated by NVIDIA
  - Host boxes funded by NSF CRI
- Coulomb Summation:
  - 1.78 TFLOPS/node
  - 271x speedup vs. one Intel QX6700 CPU core w/ SSE

©Wen-mei W. Hwu and David Kirk/NVIDIA, Berkeley, January 24-25, 2011

EcoG - One of the Most Energy Efficient Supercomputers in the World

- #3 of the Nov 2010 Green 500 list
- 128 nodes
- One Fermi GPU per node
- 934 MFLOPS/Watt
- 33.6 TFLOPS DP Linpack

- Built by Illinois students and NVIDIA researchers



GPU computing is catching on.

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

- 280 submissions to GPU Computing Gems
  - 110 articles included in two volumes

©Wen-mei W. Hwu and David Kirk/NVIDIA,
Berkeley, January 24-25, 2011

A Common GPU Usage Pattern

- A desirable approach considered impractical
  - Due to excessive computational requirement
  - But demonstrated to achieve domain benefit
  - Convolution filtering (e.g. bilateral Gaussian filters), De Novo gene assembly, etc.

- Use GPUs to accelerate the most time-consuming aspects of the approach
  - Kernels in CUDA or OpenCL
  - Refactor host code to better support kernels

- Rethink the domain problem

CUDA /OpenCL – Execution Model

- Integrated host+device app C program
  - Serial or modestly parallel parts in **host** C code
  - Highly parallel parts in **device** SPMD kernel C code

Serial Code (host)

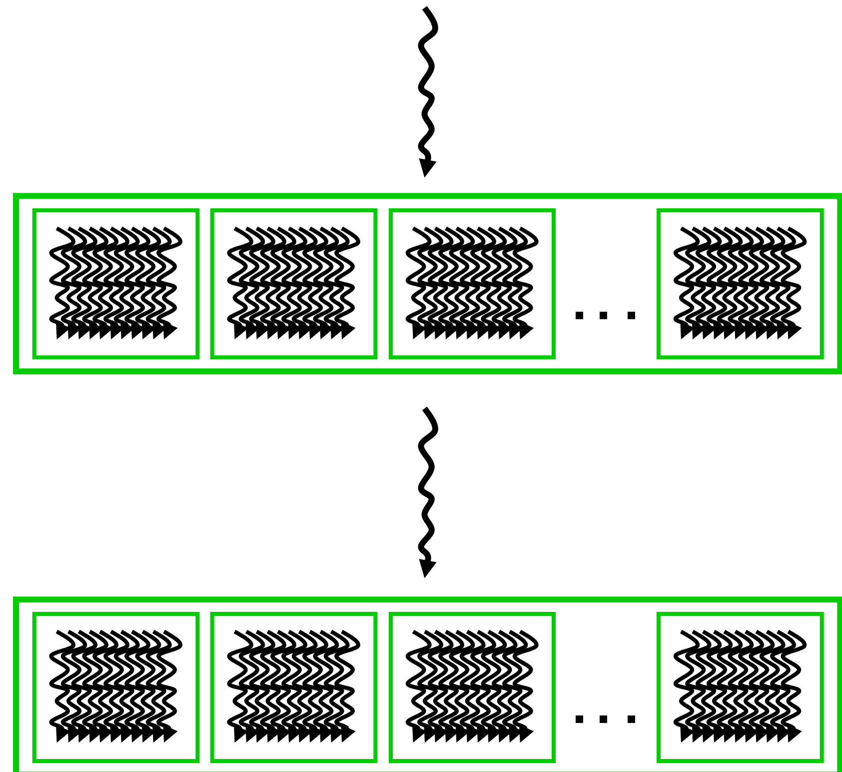
Parallel Kernel (device)

KernelA<<< nBlk, nTid >>>(args);

Serial Code (host)

Parallel Kernel (device)

KernelB<<< nBlk, nTid >>>(args);



©Wen-mei W. Hwu and David Kirk/NVIDIA, Berkeley, January 24-25, 2011

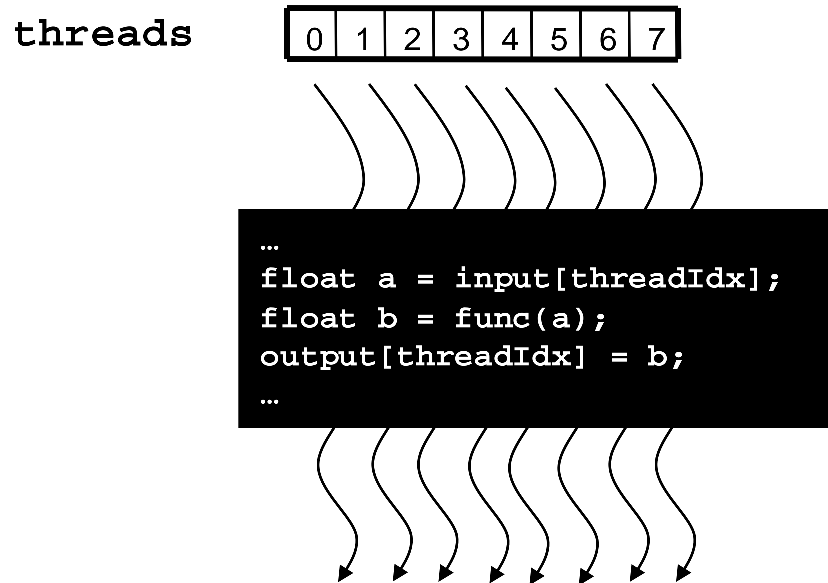
CUDA Devices and Threads

- A compute device
  - Is a coprocessor to the CPU or host
  - Has its own DRAM (device memory)
  - Runs many threads (work elements for OpenCL) in parallel
  - Is typically a GPU but can also be another type of parallel processing device
- Data-parallel portions of an application are expressed as device kernels which run on many threads
- Differences between GPU and CPU threads
  - GPU threads are extremely lightweight
    - Very little creation overhead
  - GPU needs 1000s of threads for full efficiency
    - Multi-core CPU needs only a few

©Wen-mei W. Hwu and David Kirk/NVIDIA, Berkeley, January 24-25, 2011

Arrays of Parallel Threads

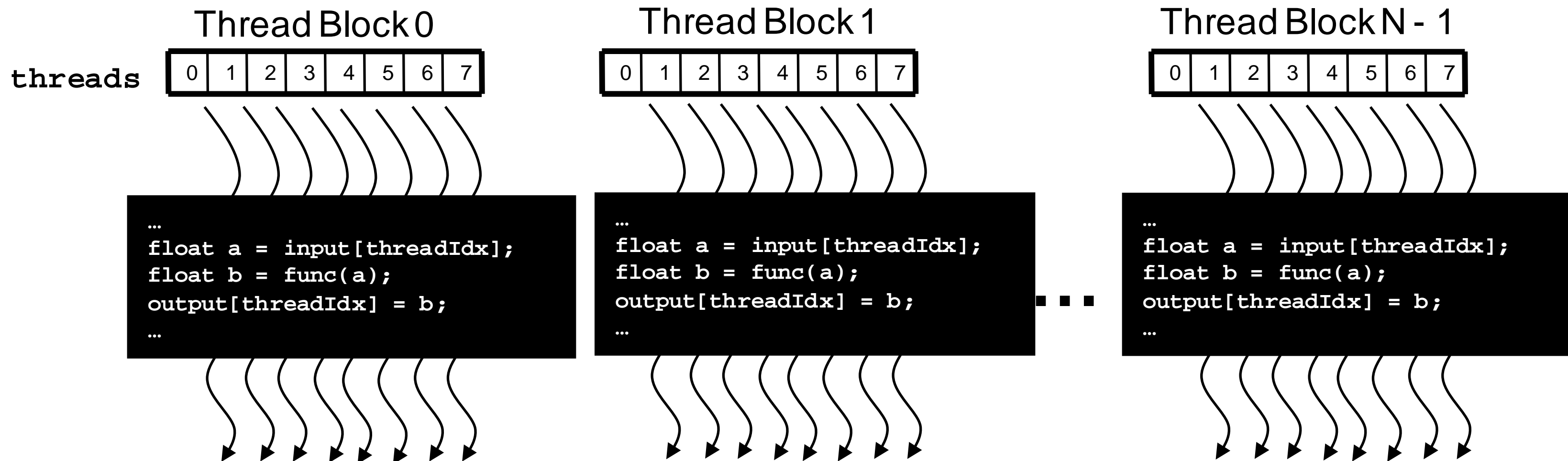
- A CUDA kernel is executed by an array of threads
  - All threads run the same code (SPMD)
  - Each thread has an index that it uses to compute memory addresses and make control decisions



©Wen-mei W. Hwu and David Kirk/NVIDIA, Berkeley, January 24-25, 2011

Thread Blocks: Scalable Cooperation

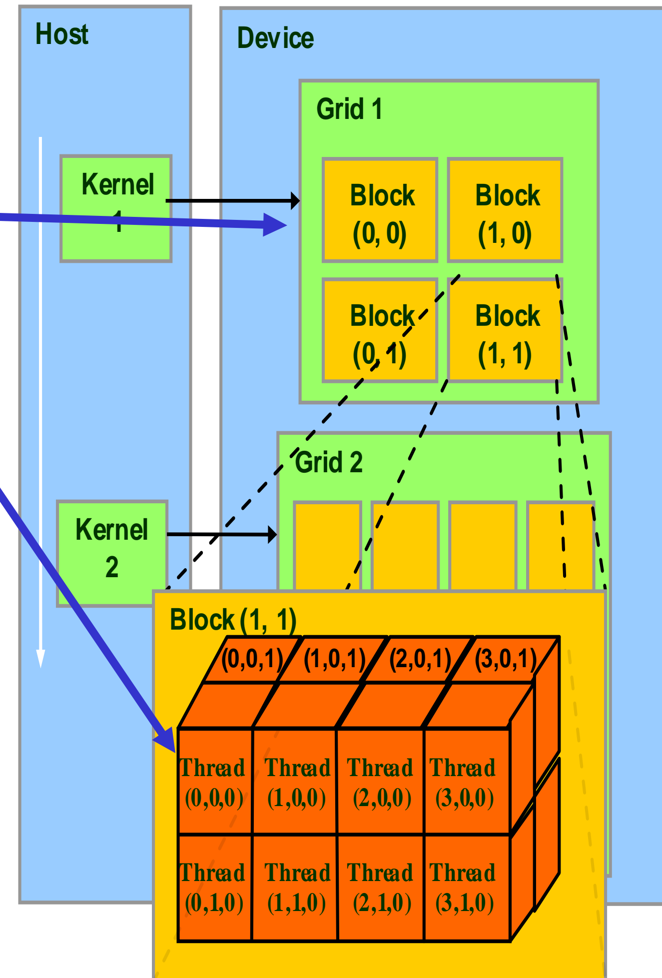
- Divide monolithic thread array into multiple blocks
  - Threads within a block cooperate via **shared memory, atomic operations** and **barrier synchronization**
  - Threads in different blocks cannot cooperate



©Wen-mei W. Hwu and David Kirk/NVIDIA, Berkeley, January 24-25, 2011

blockIdx and threadIdx

- Each thread uses indices to decide what data to work on
  - blockIdx: 1D or 2D
  - threadIdx: 1D, 2D, or 3D
- Simplifies memory addressing when processing multidimensional data
  - Image processing
  - Solving PDEs on volumes
  - …



©Wen-mei W. Hwu and David Kirk/NVIDIA, Berkeley, January 24-25, 2011

Example: Vector Addition Kernel

Device Code

```
// Compute vector sum C = A+B
// Each thread performs one pair-wise addition
__global__
void vecAdd(float* A, float* B, float* C, int n)
{
    int i = threadIdx.x + blockDim.x * blockIdx.x;
    if(i<n) C[i] = A[i] + B[i];
}

int main()
{
    // Run ceil(N/256) blocks of 256 threads each
    vecAdd<<<ceil(N/256), 256>>>(d_A, d_B, d_C, n);
}
```

©Wen-mei W. Hwu and David Kirk/NVIDIA, Berkeley, January 24-25, 2011

Example: Vector Addition Kernel

```
// Compute vector sum C = A+B
// Each thread performs one pair-wise addition
__global__
void vecAdd(float* A, float* B, float* C, int n)
{
    int i = threadIdx.x + blockDim.x * blockIdx.x;
    if(i<n) C[i] = A[i] + B[i];
}

int main()
{
    // Run ceil(N/256) blocks of 256 threads each
    vecAdd<<<ceil(N/256), 256>>>(d_A, d_B, d_C, N);
}
```

Host Code

©Wen-mei W. Hwu and David Kirk/NVIDIA, Berkeley, January 24-25, 2011

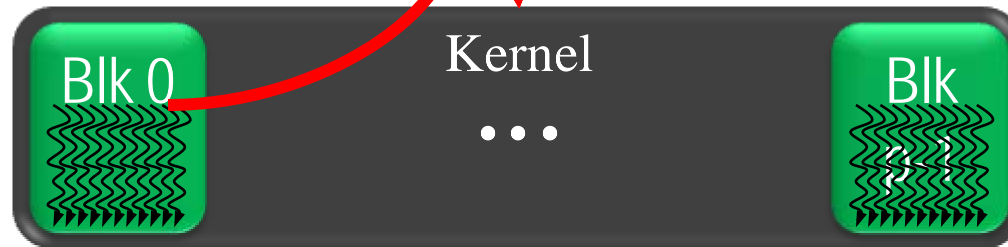
Kernel execution in a nutshell

`__host__`

`__global__`

```
vecAdd<<<P,B>>>(n,a,x,y);
```

```
blockIdx.x    blockDim.x
     threadIdx.x
```



Schedule onto multiprocessors

Harvesting Performance Benefit of Many-core GPU Requires

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

Massive Parallelism - Regularity

©Wen-mei W. Hwu and David Kirk/NVIDIA, Berkeley, January 24-25, 2011

Main Hurdles to Overcome

- Serialization due to conflicting use of critical resources
- Over subscription of Global Memory bandwidth
- Load imbalance among parallel threads

Computational Thinking Skills

- The ability to translate/formulate domain problems into computational models that can be solved efficiently by available computing resources
  - Understanding the relationship between the domain problem and the computational models
  - **Understanding the strength and limitations of the computing devices**
  - **Defining problems and models to enable efficient computational solutions**

©Wen-mei W. Hwu and David Kirk/NVIDIA, Berkeley, January 24-25, 2011

DATA ACCESS CONFLICTS

Conflicting Data Accesses Cause Serialization and Delays

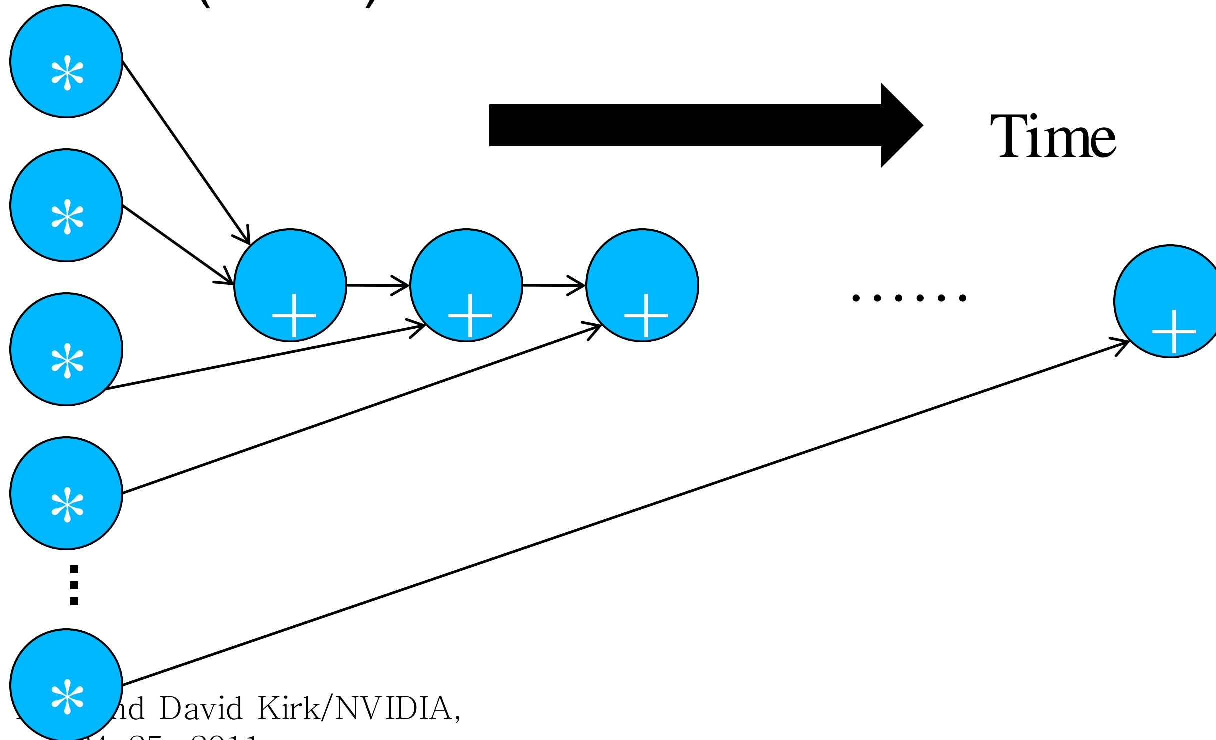
- Massively parallel execution cannot afford serialization

- Contentions in accessing critical data causes serialization

©Wen-mei W. Hwu and David Kirk/NVIDIA, Berkeley, January 24-25, 2011

A Simple Example

- A naïve inner product algorithm of two vectors of one million elements each
  - All multiplications can be done in time unit (parallel)
  - Additions to a single accumulator in one million time units (serial)



How much can conflicts hurt?

- Amdahl's Law
  - If fraction X of a computation is serialized, the speedup can not be more than 1/(1-X)
- In the previous example, X = 50%
  - Half the calculations are serialized
  - No more than 2X speedup, no matter how many computing cores are used

©Wen-mei W. Hwu and David Kirk/NVIDIA,
Berkeley, January 24-25, 2011

Global Memory Bandwidth

Ideal

Reality

©Wen-mei W. Hwu and David Kirk/NVIDIA, Berkeley, January 24-25, 2011

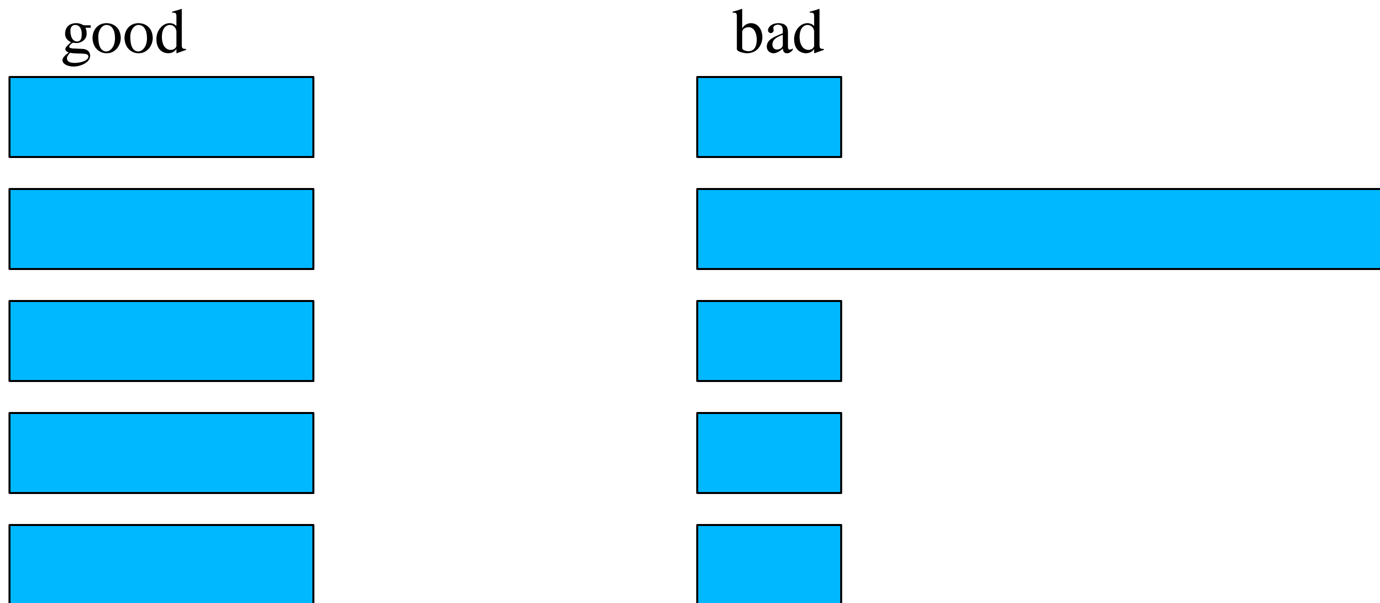
Global Memory Bandwidth

- Many-core processors have limited off-chip memory access bandwidth compared to peak compute throughput
- Fermi
  - 1 TFLOPS SPFP peak throughput
  - 0.5 TFLOPS DPFP peak throughput
  - 144 GB/s peak off-chip memory access bandwidth
    - 36 G SPFP operands per second
    - 18 G DPFP operands per second
  - To achieve peak throughput, a program must perform 1,000/36 = ~28 SPFP (14 DPFP) arithmetic operations for each operand value fetched from off-chip memory

©Wen-mei W. Hwu and David Kirk/NVIDIA, Berkeley, January 24-25, 2011

Load Balance

- The total amount of time to complete a parallel job is limited by the thread that takes the longest to finish

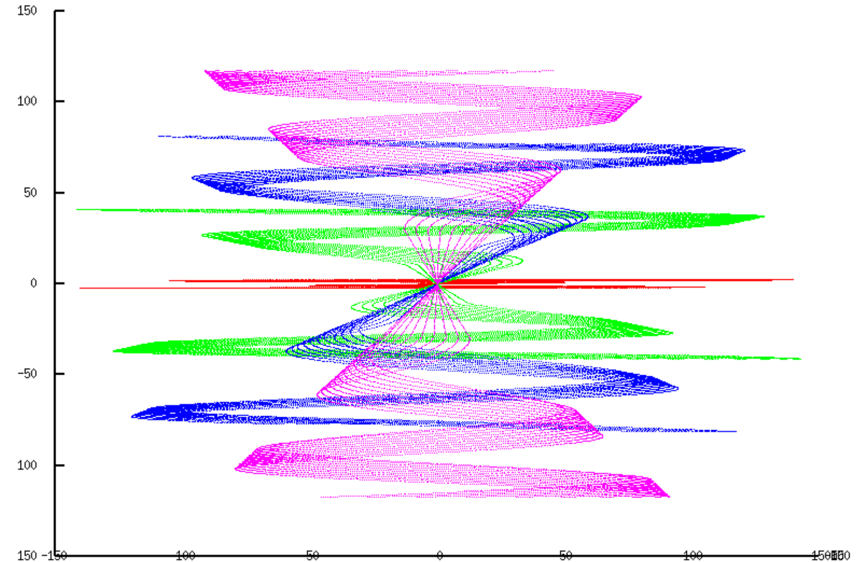


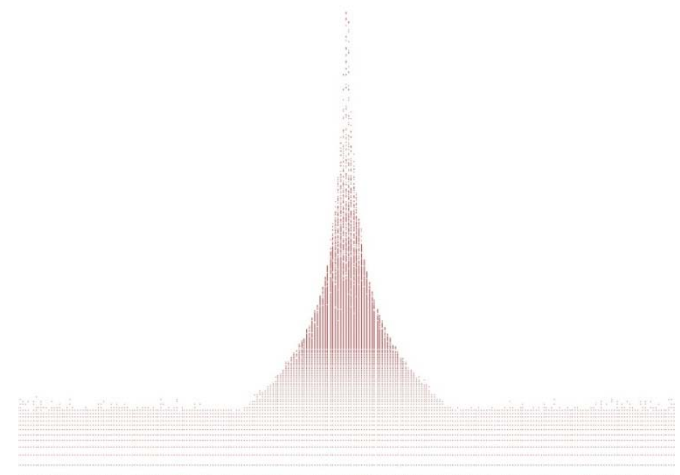
How bad can it be?

- Assume that a job takes 100 units of time for one person to finish
  - If we break up the job into 10 parts of 10 units each and have fo10 people to do it in parallel, we can get a 10X speedup
  - If we break up the job into 50, 10, 5, 5, 5, 5, 5, 5, 5, 5 units, the same 10 people will take 50 units to finish, with 9 of them idling for most of the time. We will get no more than 2X speedup.

How does imbalance come about?

- Non-uniform data distributions
  - Highly concentrated spatial data areas
  - Astronomy, medical imaging, computer vision, rendering, …
- If each thread processes the input data of a given spatial volume unit, some will do a lot more work than others





Eight Algorithmic Techniques (so far)

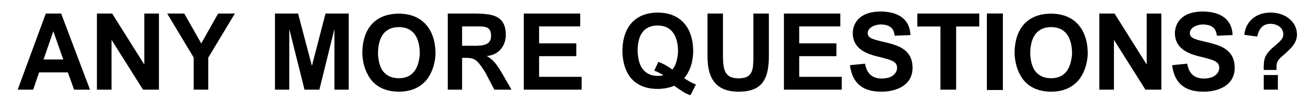
Technique	Contention	Bandwidth	Locality	Efficiency	Load Imbalance	CPU Leveraging
Tiling		X	X			
Privatization	X		X			
Regularization				X	X	X
Compaction		X				
Binning		X	X	X		X
Data Layout Transformation	X		X			
Thread Coarsening	X	X	X	X		
Scatter to Gather Conversion	X					

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

©Wen-mei W. Hwu and David Kirk/NVIDIA,
Berkeley, January 24-25, 2011

You can do it.

- Computational thinking is not as hard as you may think it is.
  - Most techniques have been explained, if at all, at the level of computer experts.
  - The purpose of the course is to make them accessible to domain scientists and engineers.



©Wen-mei W. Hwu and David Kirk/NVIDIA,
Berkeley, January 24-25, 2011