Ubiquitous Parallel Computing:
Architectures and Programming from GPUs to the Cloud

Prof. Stephen Jenks
UCI Electrical Engineering and Computer Science
& Hiperwall, Inc.
stephen@stephenjenks.com

Hiperwall Overview

For more information:

http://hiperwall.calit2.uci.edu

http://hiperwall.com

Outline

- Parallelism Overview
  - Thread and OpenMP Programming
- Parallel Microprocessors (Multi-core)
  - Architecture Bottlenecks and Solutions
- Asymmetric Parallel Computing
  - Cell Processor
  - GPUs
    - CUDA and OpenCL
    - Examples and Performance Benefits
- Cloud Computing with Hadoop

Overview and Motivation

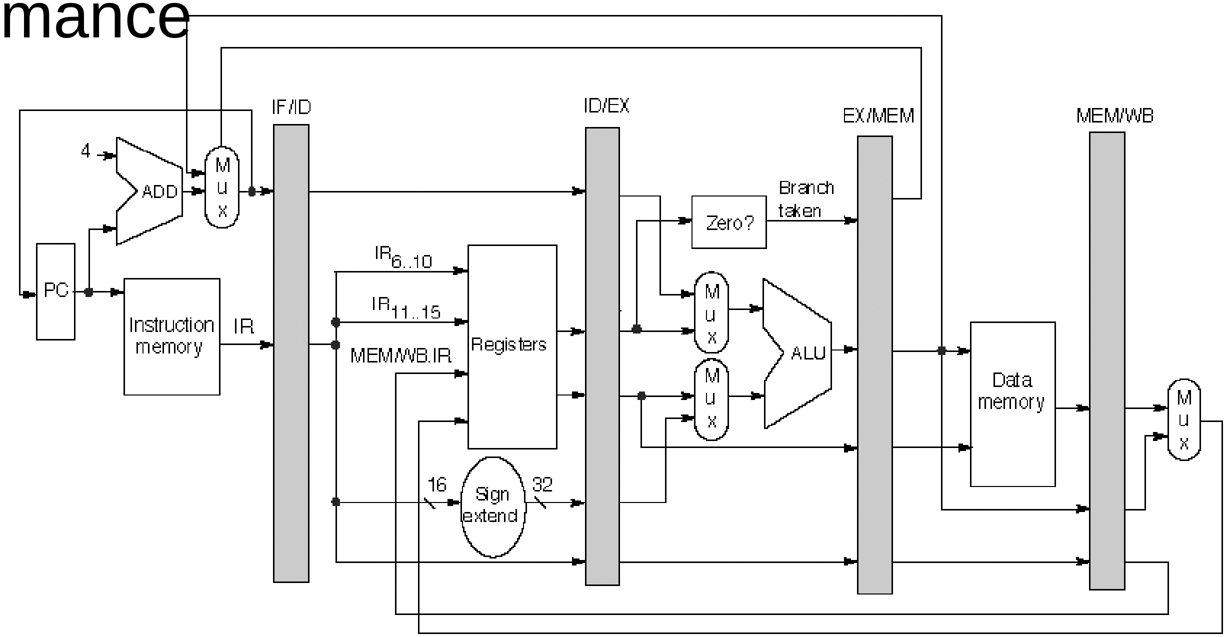
- Conventional Parallel Computing Was:
  - For scientists with large problem sets
  - Complex and difficult
  - Very expensive computers with limited access
- Parallel Computing is Becoming:
  - Ubiquitous
  - Cheap
  - Essential
  - Different
  - Still complex and difficult
- Where Is Parallel Computing Going?

Computing is Changing

- Parallel programming is essential
  - Clock speed not increasing much
  - Performance gains require parallelism
- Parallelism is changing
  - Special purpose parallel engines
  - CPU and parallel engine work together
  - Different code on CPU & parallel engine → Asymmetric Computing

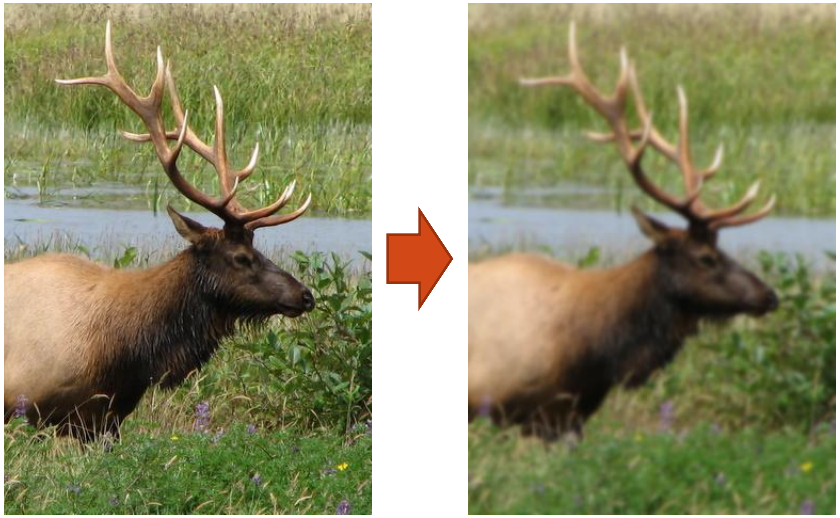
Conventional Processor Architecture

- Hasn't Changed Much For 40 Years
  - Pipelining and Superscalar Since the 1960s
  - But has become integrated → Microprocessors
- High Clock Speed
  - Great performance
  - High Power
  - Cooling Issues
  - Various Solutions



From Hennessy & Patterson, 2nd Ed.

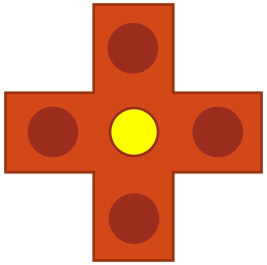
Parallel Computing Problem Overview



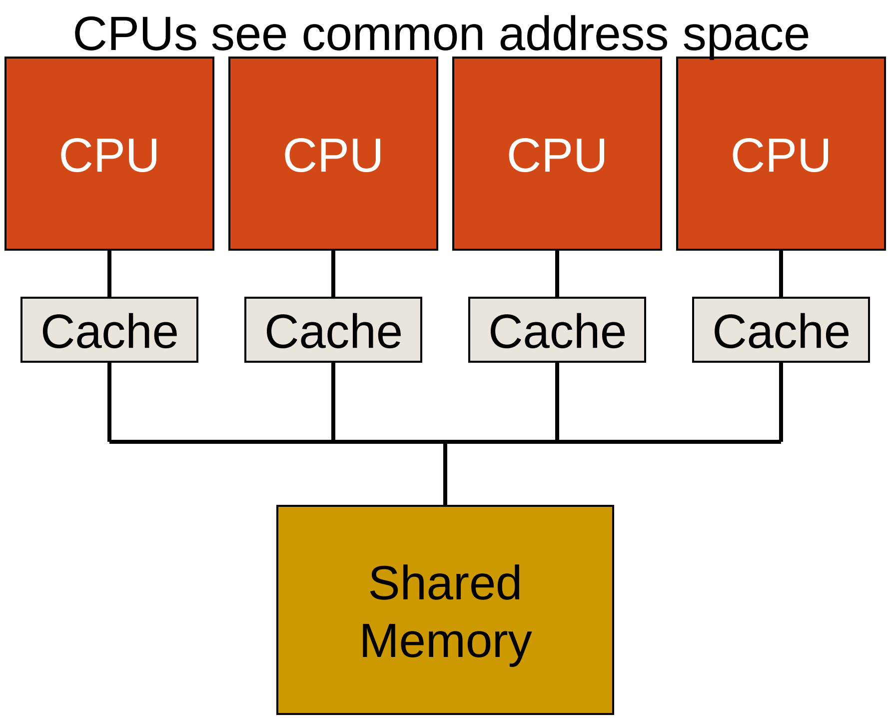
newimage[i][j] = (image[i][j] +
image[i][j-1] + image[i][j+1] +
image[i+1][j] + image[i-1][j]) / 5

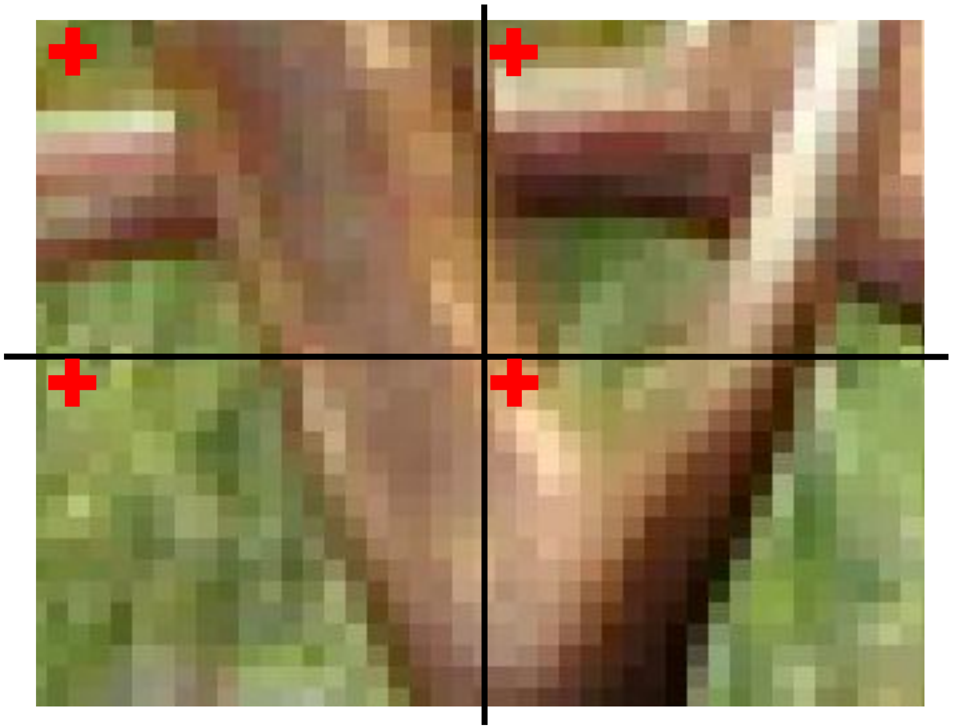
Image Relaxation (Blur)

Stencil



Shared Memory Multiprocessors





- Each CPU computes results for its partition
- Memory is shared so dependences satisfied

Shared Memory Programming

- Threads
  - POSIX Threads
  - Windows Threads

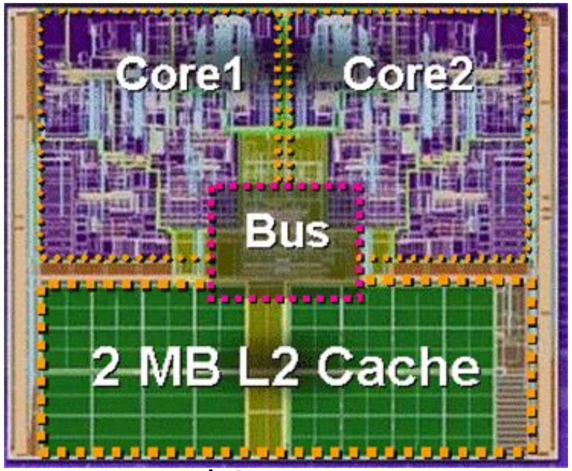
Not Integrated with Compiler or Language
→No idea if code is in thread or not
→Poor optimizations

- OpenMP
  - User-inserted directives to compiler
  - Loop parallelism
  - Parallel regions
  - Visual Studio
  - GCC 4.2

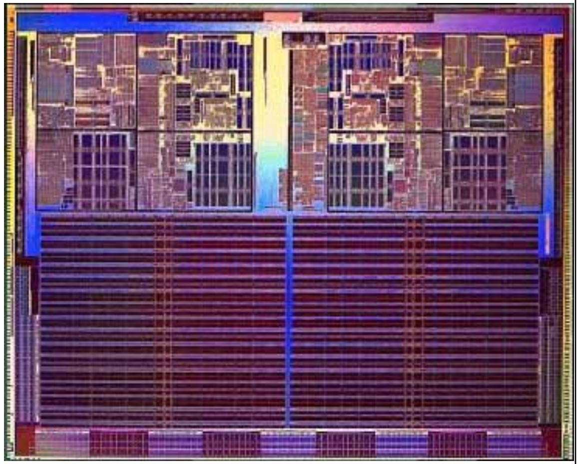
```
#pragma omp parallel for
for (i=0; i<n; i++)
  a[i] = b[i] * c[i];
```

Multicore Processors

- Several CPU Cores Per Chip
- Shared Memory
- Shared Caches (sometimes)
- Lower Clock Speed
  - Lower Power & Heat
  - But Good Performance
- Program with Threads
- Single Threaded Code
  - Not Faster (except on Core i7)
  - Majority of Code Today



Intel Core Duo

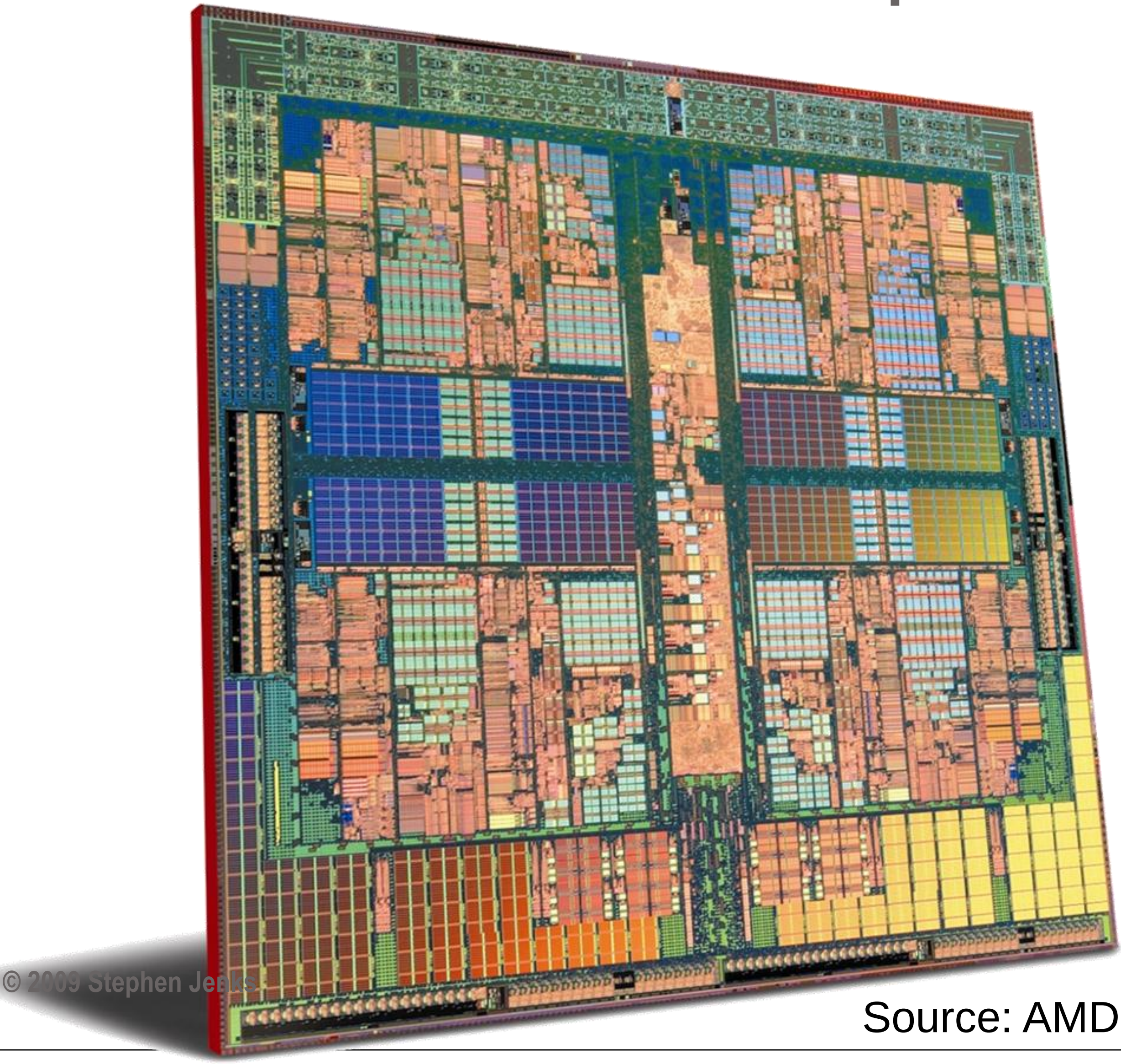

AMD Athlon 64 X2

Conventional Processors are Dinosaurs

- So much circuitry dedicated to keeping ALUs fed:
  - Cache
  - Out-of-order execution/reorder buffer
  - Branch prediction
  - Large Register Sets
  - Simultaneous Multithreading
- ALU (Arithmetic Logic Unit) tiny by comparison
- Huge power for little performance gain

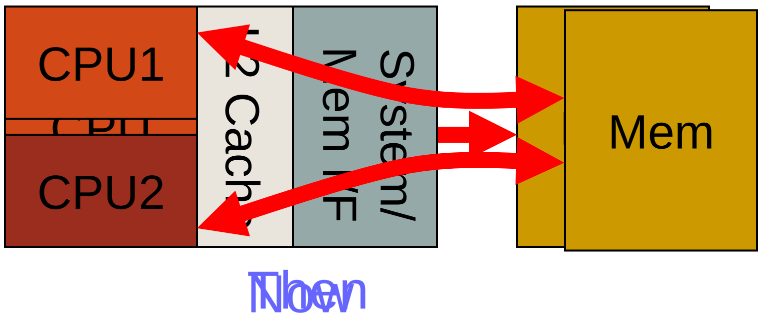
With thanks to Stanford's Pat Hanrahan for the analogy

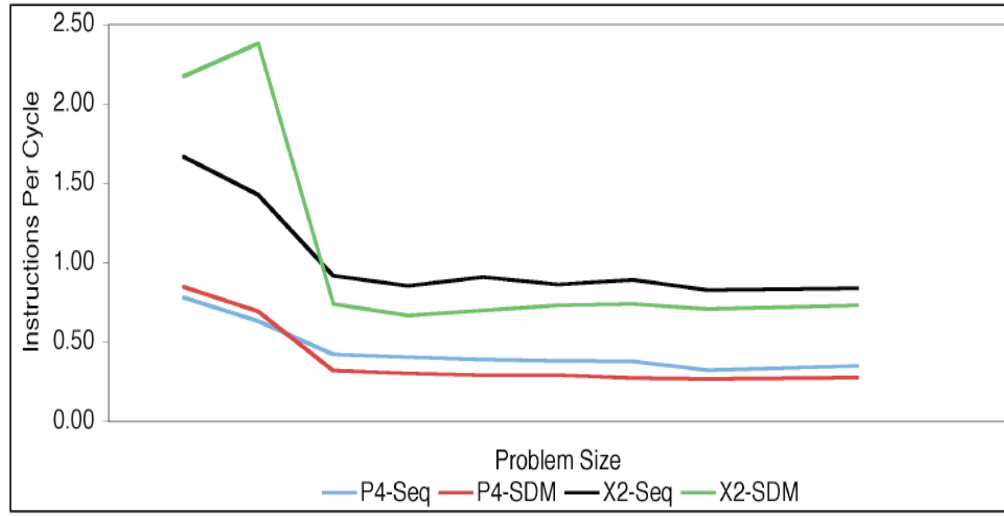
AMD Phenom X4 Floorplan



Source: AMD.com

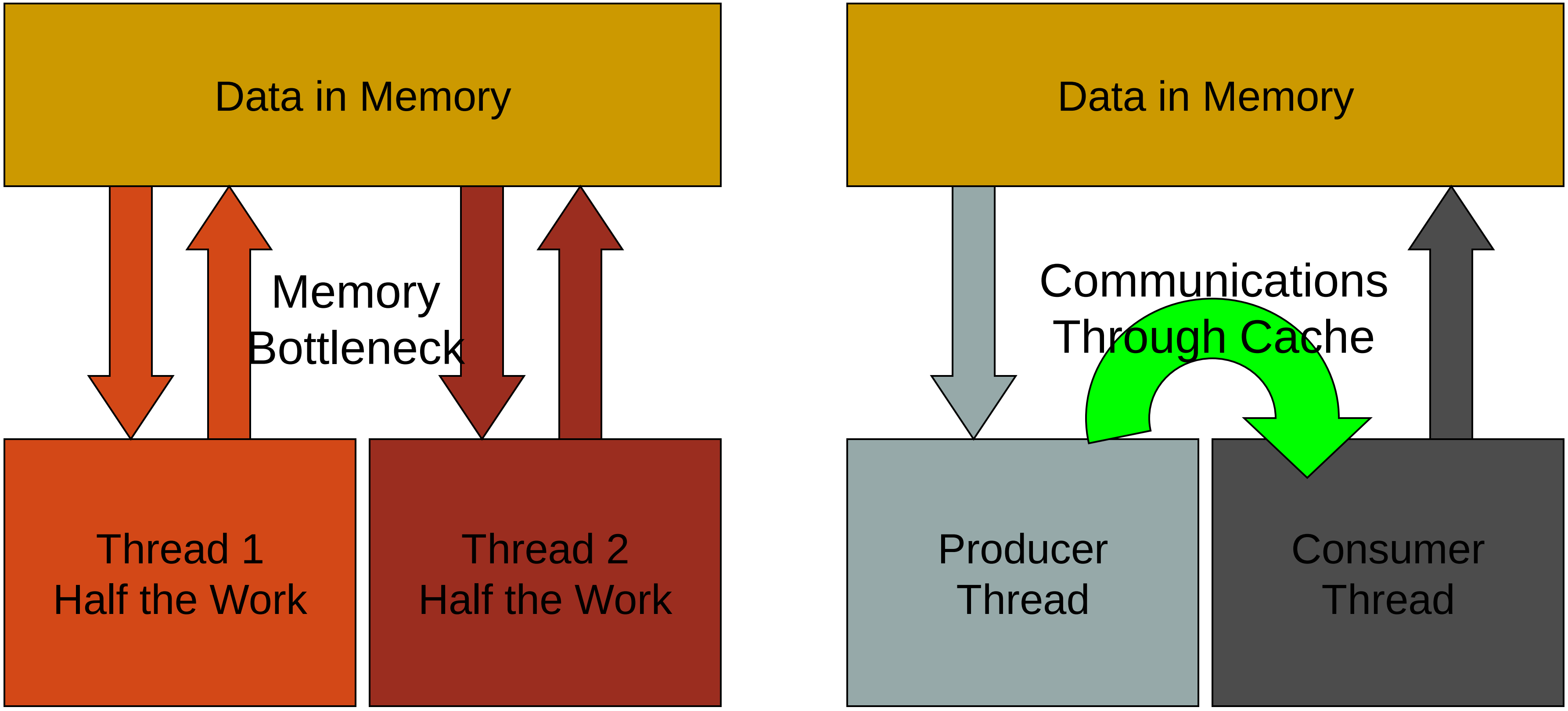
Parallel Microprocessor Problems





- Memory interface too slow for 1 core/thread
- Now multiple threads access memory simultaneously, overwhelming memory interface
- Parallel programs can run as slowly as sequential ones!

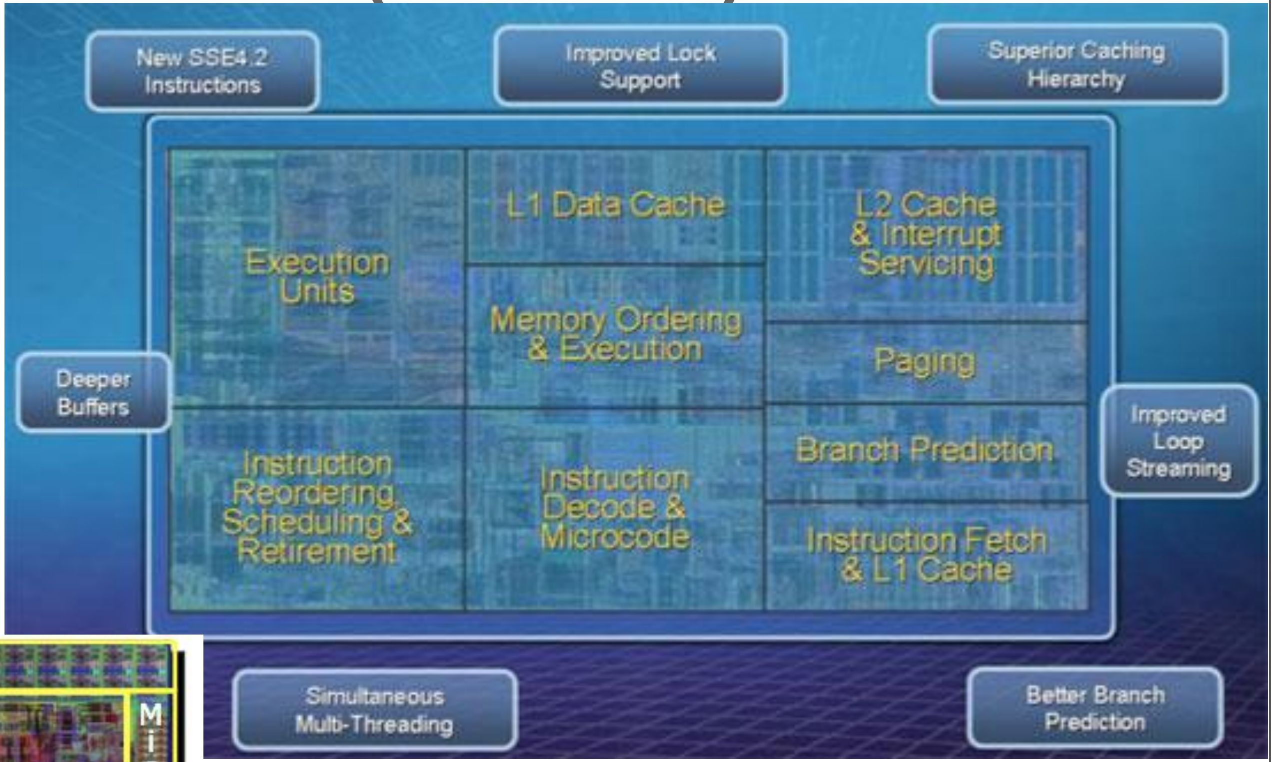
SPPM: Producer/Consumer Parallelism Using The Cache

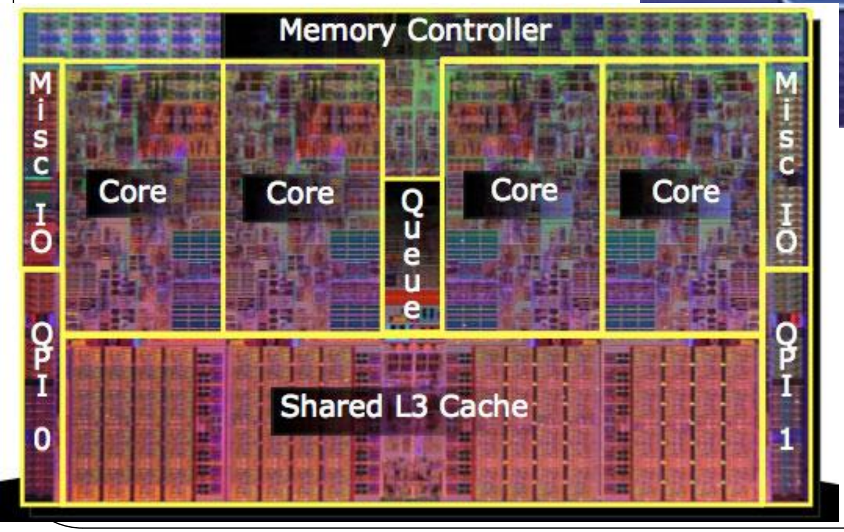


Multicore Architecture Improvements

- Cores in Common Multicore Chips Not Well Connected
  - Communicate Through Cache & Memory
  - Synchronization Slow (OS-based) or Uses Spin Waits
- Register-Based Synchronization
  - Shared Registers Between Cores
  - Halt Processor While Waiting To Save Power
- Preemptive Communications (Prepushing)
  - Reduces Latency over Demand-Based Fetches
  - Cuts Cache Coherence Traffic/Activity
- Software Controlled Eviction
  - Manages Shared Caches Using Explicit Operations
  - Move Data to Shared Cache Before Needed From Private Cache
- Synchronization Engine
  - Hardware-based multicore synchronization operations

Single Nehalem (Intel I7) Core

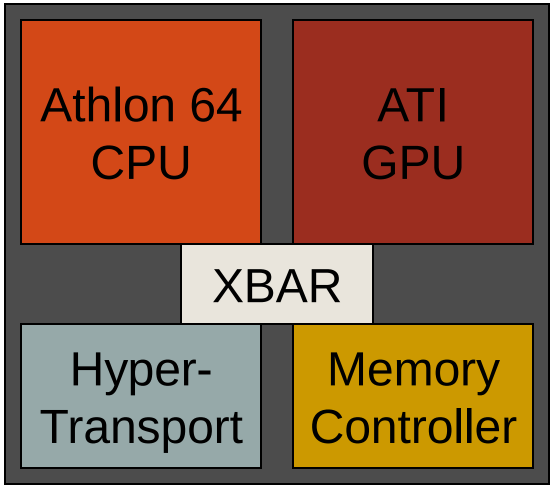




Nehalem - Everything You Need to Know about Intel's New Architecture
http://www.anandtech.com/cpuchipsets/intel/showdoc.aspx?i=3382

Asymmetric Parallel Accelerators

- Current Cores are Powerful and General
  - Some Applications Only Need Certain Operations
  - Perhaps a Simpler Processor Could Be Faster
- Pair General CPUs with Specialized Accelerators
  - Graphics Processing Unit
  - Field Programmable Gate Array (FPGA)
  - Single Instruction, Multiple Data (SIMD) Processor



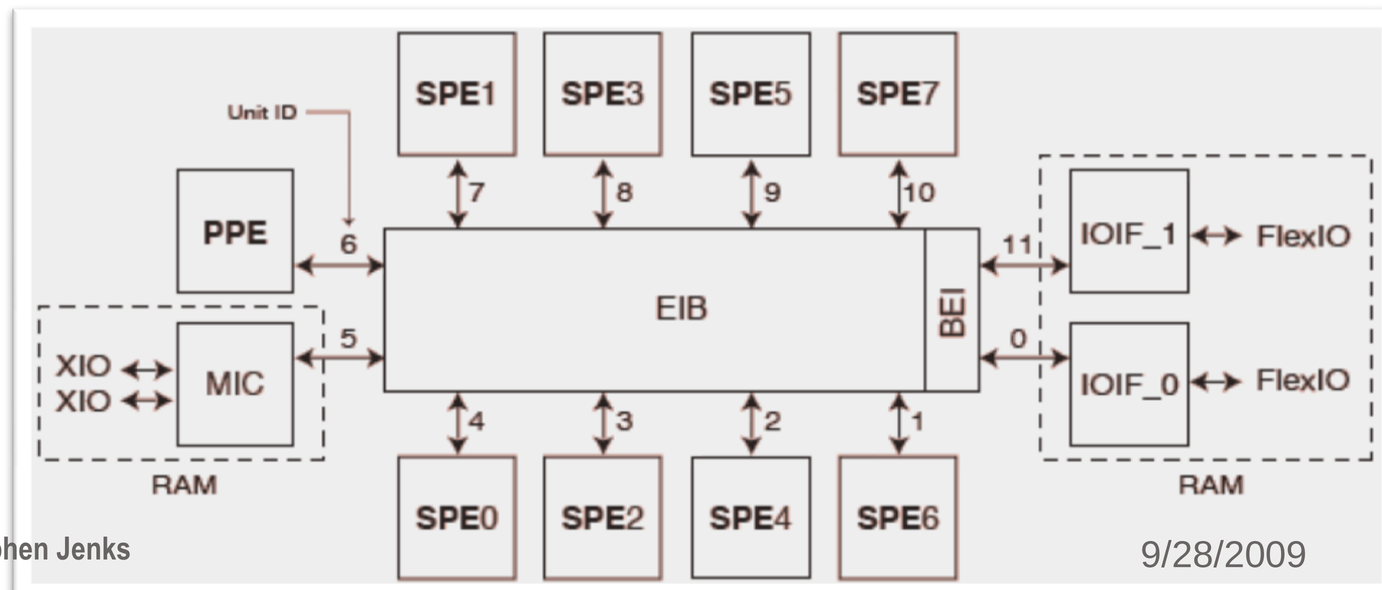
Possible Hybrid AMD Multi-Core Design

Different code runs on CPU cores and accelerators: Asymmetric Computing

Cell Broadband Engine

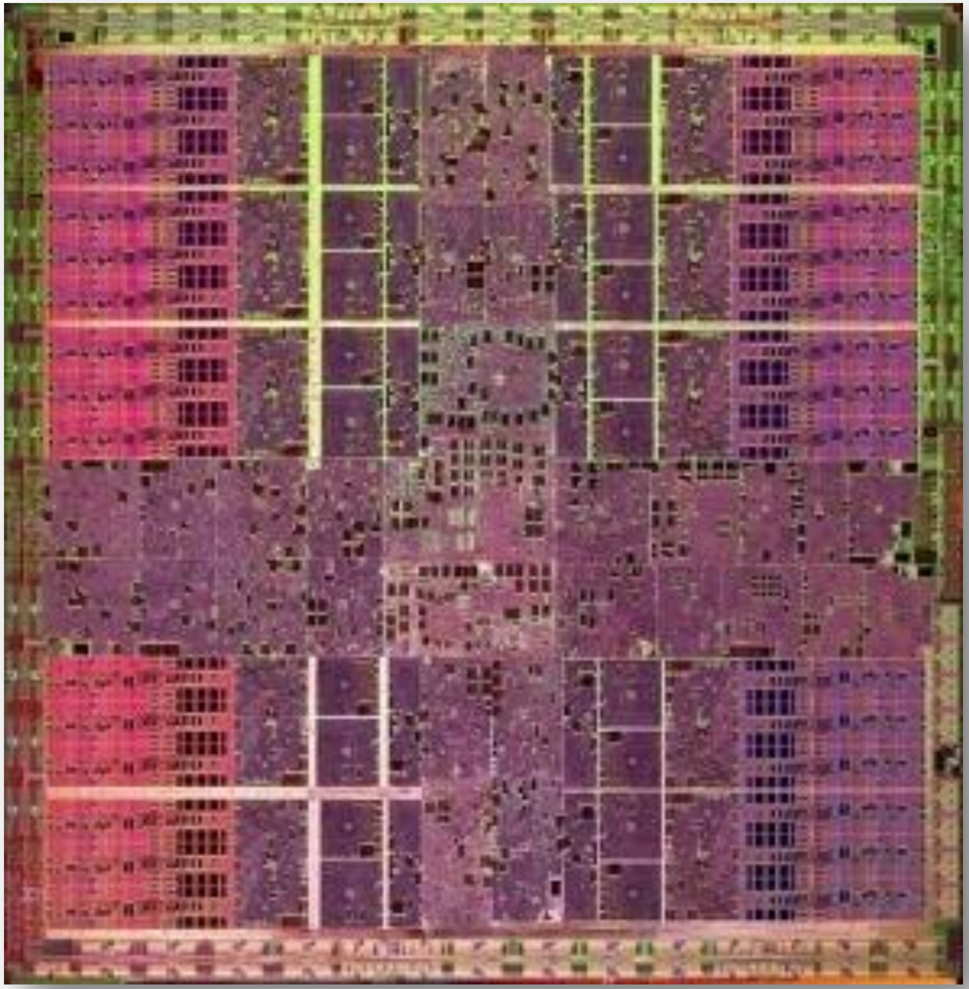
- PowerPC Processing Element with Simultaneous Multithreading at 3.2 GHz
- 8 Synergistic Processing Elements at 3.2 GHz
    - Optimized for SIMD/Vector processing (100 GFLOPS Total)
    - 256KB Local Storage - no cache
- 4x16-byte-wide rings @ 96 bytes per clock cycle

From *IBM Cell Broadband Engine Programmer Handbook*, 10 May 2006



NVIDIA GPU Floorplan

10 multiprocessors
24 threads each
240 simultaneous threads!

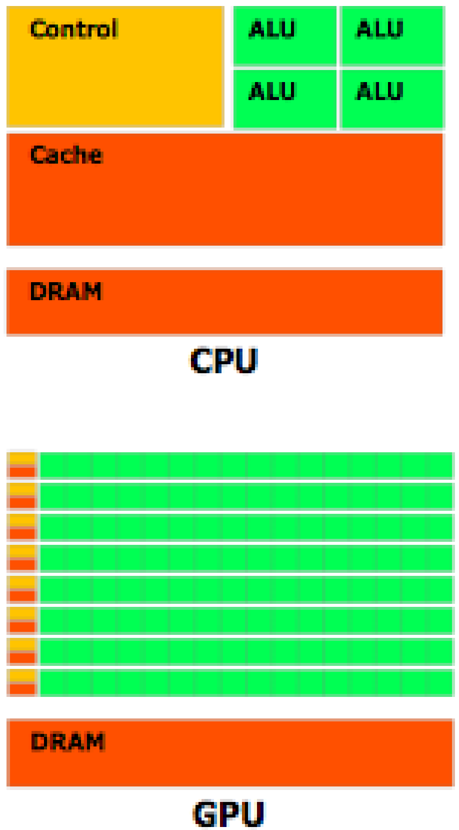


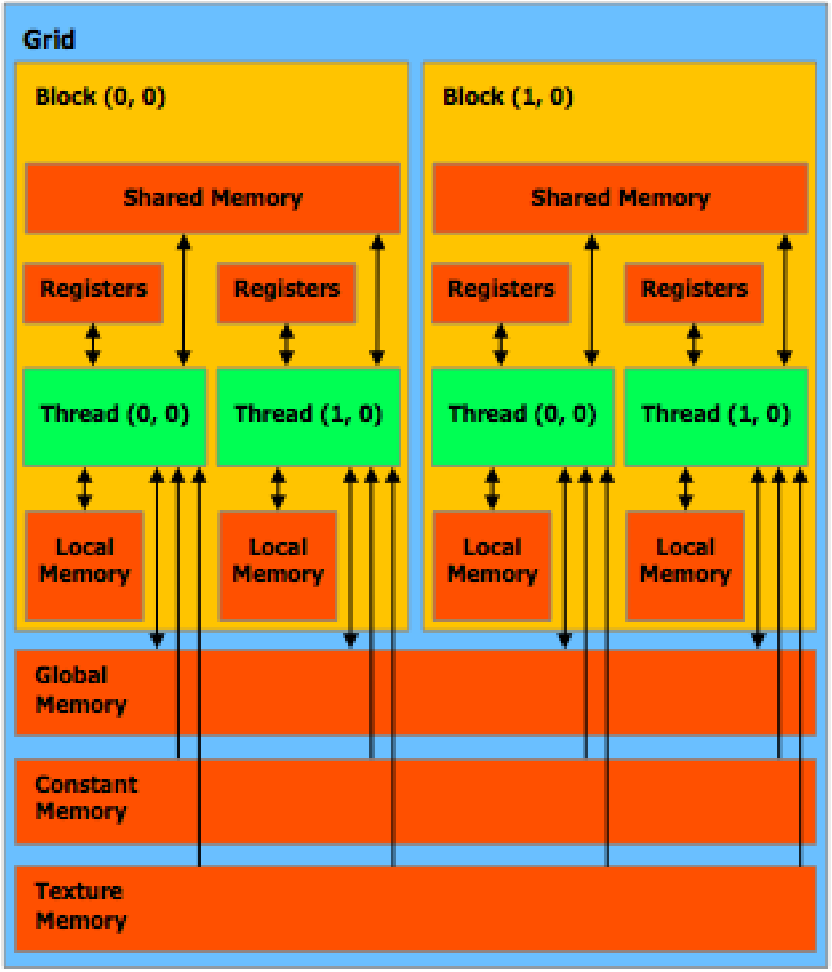
Source: Dr. Sumit Gupta - NVIDIA

Graphics Processing Unit (GPU)

- GPUs Do Pixel Pushing and Matrix Math

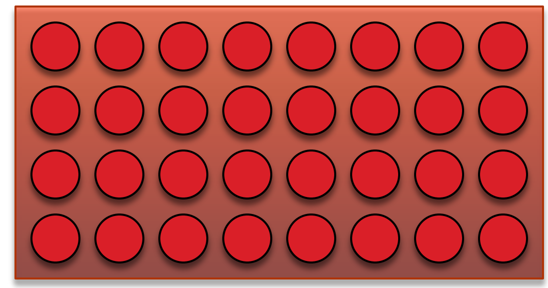
From
NVIDIA CUDA Compute Unified Device Architecture Programming Guide
11/29/2007





CUDA & OpenCL Programming Model

- Data Parallel
  - But not Loop Parallel
  - Very Lightweight Threads
  - Write Code from Thread's Point of View
- No Shared Memory
  - Host Copies Data To and From Device
  - Different Memories
- Hundreds of Parallel Threads (Sort-of SIMD)



Block of Threads

OpenCL Programming Details

- Supports GPUs (NVIDIA and ATI) and CPUs
- Built into Apple's Snow Leopard Mac OS X 10.6
- On-the-fly Compilation
- Supports floats (doubles optional and not supported on all hardware)
- Pattern:

Overhead

1. Acquire "device" and get capabilities
2. Initialize (compile) OpenCL "kernel"
3. Move data to "device" memory from "host" memory
4. Set kernel parameters and execution size, start kernel
5. When done, copy results back from device memory
6. Repeat prior 3 steps, as needed

Simple OpenCL Kernel Code

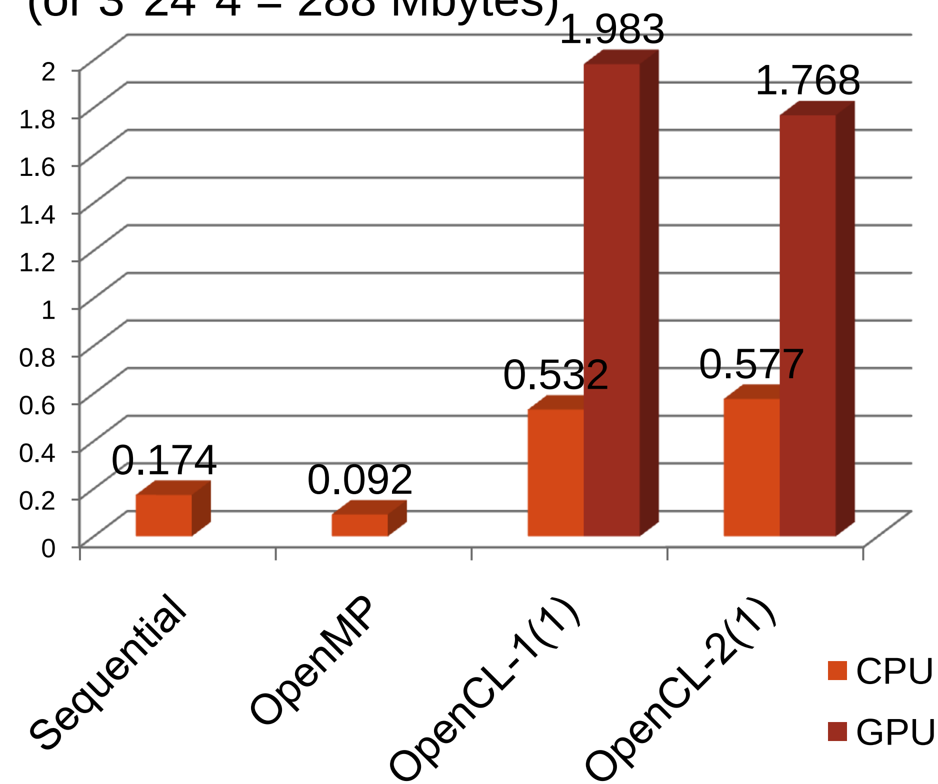
```
__kernel void openclmin(__global float *a,
        __global float *b, __global float *c)
{
  int gid = get_global_id(0);

  c[gid] = fmin(a[gid], b[gid]);
}
```

Get Position In Global Index Space

100 Iterations in OpenCL: 0.031 secs/iteration (or nearly 6 times faster than sequential!)

Problem size: 24 million element arrays (or 3*24*4 = 288 Mbytes)



GPU Performance/Architecture Issues (CUDA and OpenCL)

- Avoid branching

```
__kernel void openclmin(__global float *a,
          __global float *b, __global float *c)
{
   int gid = get_global_id(0);
   float local_a = a[gid];
   float local_b = b[gid];
   if (local_a < local_b)
      c[gid] = local_a;
   else
      c[gid] = local_b;
}
```

Branch instead of fmin()

Runs 32% slower on GPU than before

- Memory is fast, but long latency
  - Cache data in shared space
  - Avoid bank conflicts
- Only new GPUs support Doubles
- Memory limited (256MB to 4GB vs 16 or 32 GB)

GPU Computing is Great, but be aware of the limitations.

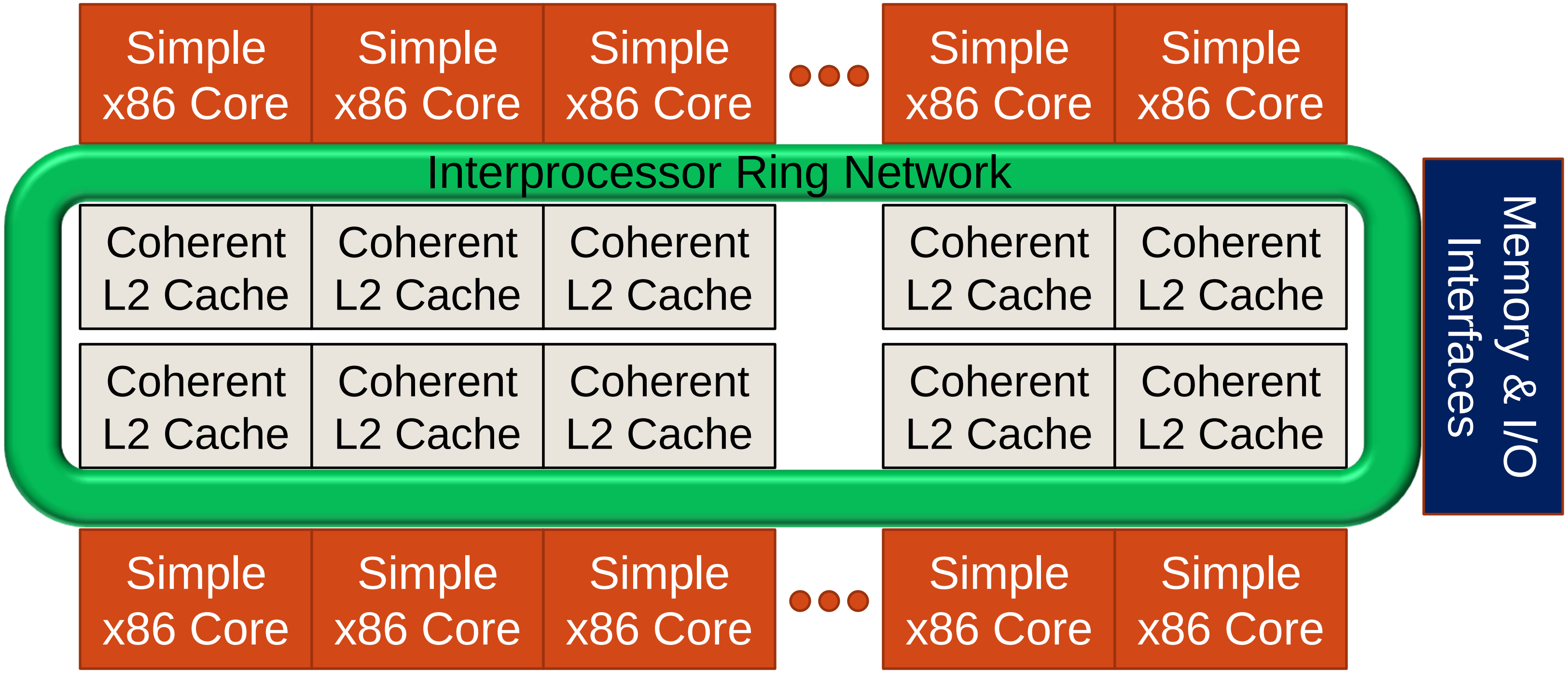
GPU Usage on Hiperwall

- QuickTime renders movies as YUV frames
  - 16 bits per pixel rather than 32 for RGBA
  - Apple OpenGL supports YUV textures natively
- CPU takes 5-8ms to convert 720p frame to RGBA
- Solution GPU computing with Cg
  - Modern NVIDIA card – 100 times faster
  - Older ATI chip (shared memory) – 15 times faster
  - My laptop (Intel GPU w/ shared) – 3 times slower!

GPU Programming Choices

- CUDA (NVIDIA)
  - Mature. Probably best tool support
  - Supported on Windows, Linux, Mac
  - Only on NVIDIA hardware
- OpenCL (Open standard coalition)
  - Stable, but less tool support (for now)
  - Built into Mac OS X 10.6, supported on Linux & Windows
  - ATI and NVIDIA hardware
- DirectCompute (Microsoft)
  - New, but drivers available
  - Built into DirectX 11 (no Mac or Linux support)
  - ATI and NVIDIA hardware

Intel Larrabee



Many simple, fast, low power, in-order x86 cores

Larrabee: A Many-Core x86 Architecture for Visual Computing
ACM Transactions on Graphics, Vol. 27, No. 3, Article 18, Publication date: August 2008.

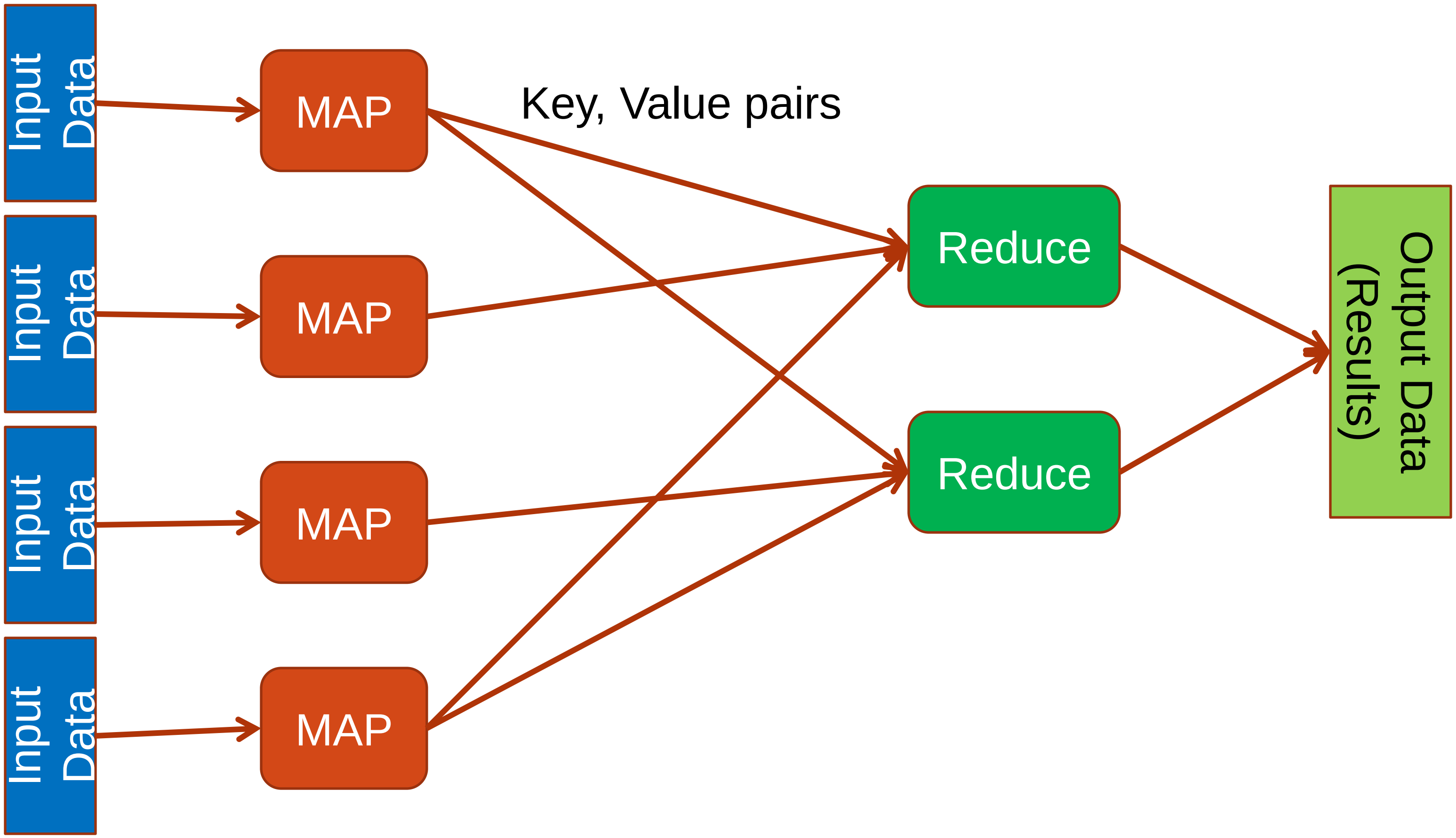
Cloud Computing

- What is cloud computing?
  - Latest buzzword in computing
  - Replaces Grid, Utility Computing, … as latest craze
- Multiple definitions
  - Web-based applications (Google Docs)
  - On-demand Computing Resources
    - Virtual Machines in a server farm (Amazon, Google, IBM)
    - Parallel/Distributed Computing Paradigm to use them

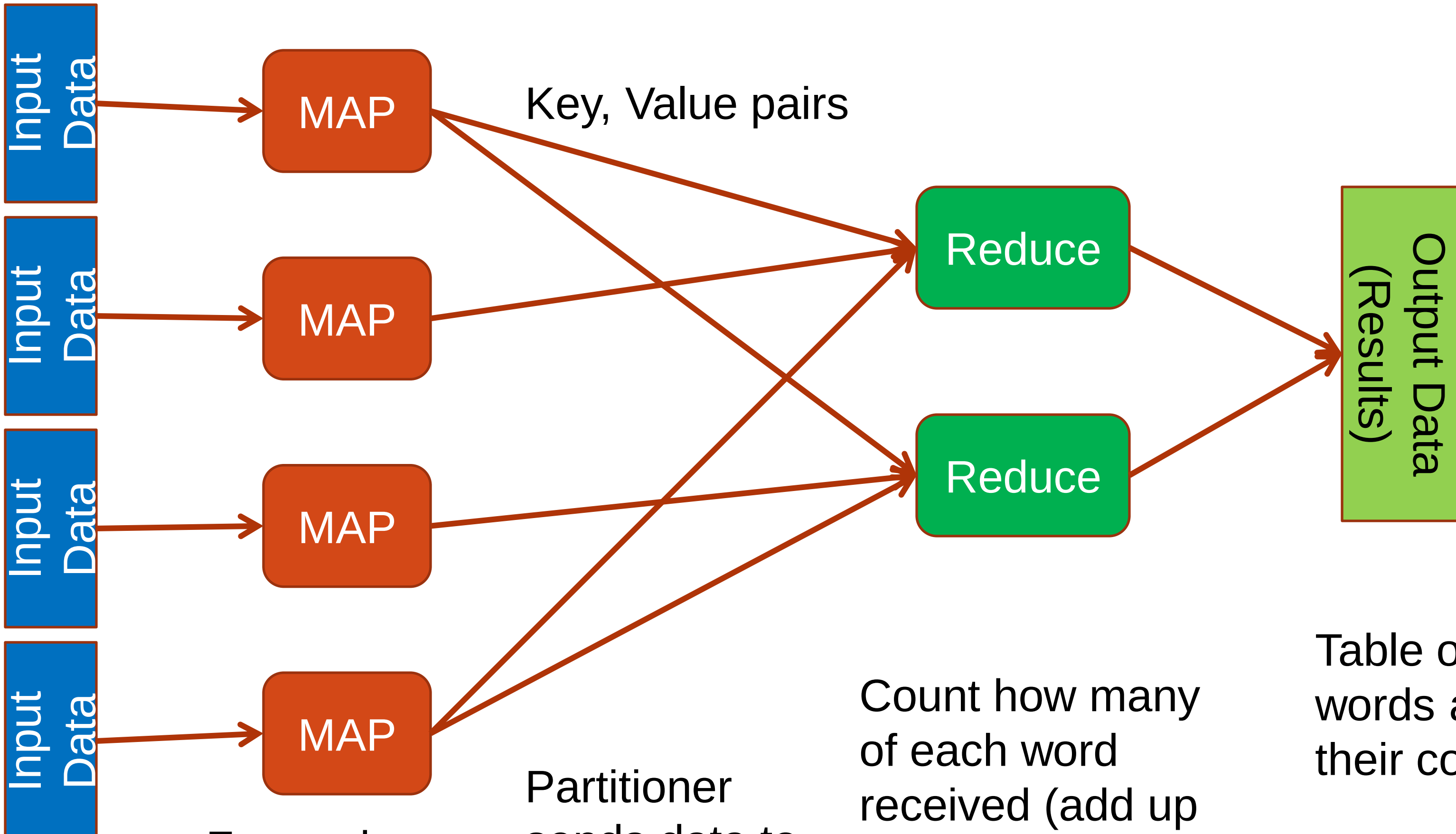
Cloud-Based Map-Reduce

- Special purpose computation with LOTS of data
  - Used by Google and many others
  - Based on Lisp’s Map and Reduce functions
- Examples: Distributed Grep, Count of URL Access Frequency, Reverse Web-Link Graphs, Term-Vector per Host, Inverted Index, Distributed Sort
  - Most produce small results from large input
  - Simple computation per element, but lots of them
- Open source implementation: Hadoop
  - Yahoo and Apache
  - Java-based, includes distributed file system

Cloud (Hadoop) Application Topology



Hadoop Word Count Example



Book text

For each word, emit <word, 1> pair

Partitioner sends data to right Reduce based on key (hash or alpha)

Count how many of each word received (add up the “values” for each key)

Table of words and their counts

Summary

- Parallel Computing will soon be required for good performance
  - Parallel programming is neither free nor easy
  - New tools make it better than before
  - Architecture influences performance
  - Parallelism at various layers: chip to cloud
    - Thread parallelism
    - Data-parallelism (top-down view): OpenMP
    - Data-parallelism (bottom-up view): GPU programming
    - Distributed memory parallelism: Hadoop

Resources

- HIPerWall: http://hiperwall.calit2.uci.edu/
- Hiperwall, Inc.: http://hiperwall.com/
- Vadlamani, S. & Jenks, S. “Architectural Considerations for Efficient Software Execution on Parallel Microprocessors,” *21st IEEE International Parallel & Distributed Processing Symposium,* **2007**
- Fide, S. & Jenks, S. “Architecture Optimizations for Synchronization and Communication on Chip Multiprocessors,” *Workshop on Multithreaded Architectures and Applications (MTAAP08) Held in Conjunction With International Parallel and Distributed Processing Symposium (IPDPS 2008),* **2008**
- Fide, S. & Jenks, S. “Proactive Use of Shared L3 Caches to Enhance Cache Communications in Multi-Core Processors,” *IEEE Computer Architecture Letters,* **2008**
- CUDA: http://www.nvidia.com/object/cuda_home.html#
- OpenCL: http://www.khronos.org/opencl/
- OpenMP: http://openmp.org/wp/
- Hadoop: http://hadoop.apache.org/
- OpenCL Tutorials (David Gohara): http://www.macresearch.org/opencl
- Di Blas, A. & Kaldewey, T. Data Monster: Why graphics processors will transform database processing. *IEEE Spectrum,* **2009**, *46*, 46-51