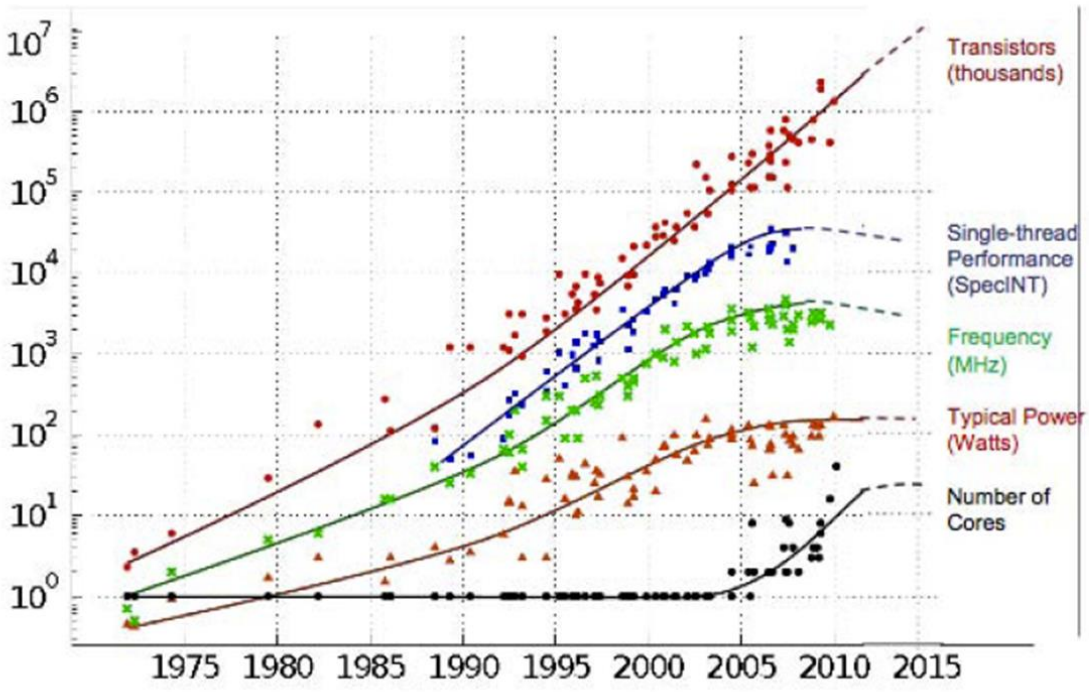
GPU Teaching Kit

Accelerated Computing

Introduction to GPU Computing

Naga Vydyanathan

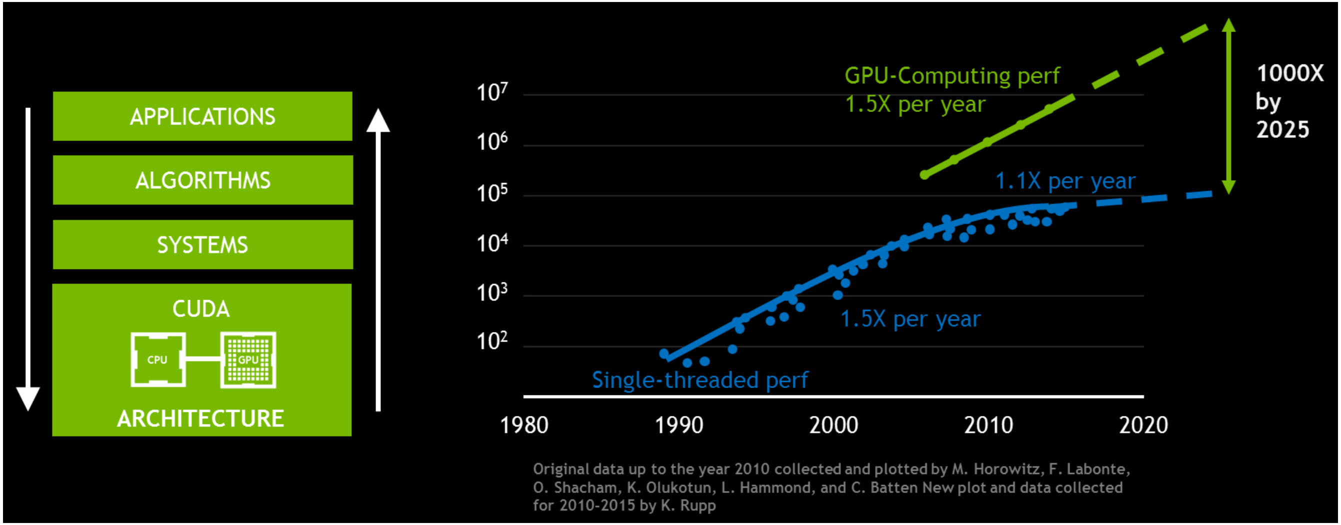
Processor Evolution Trends



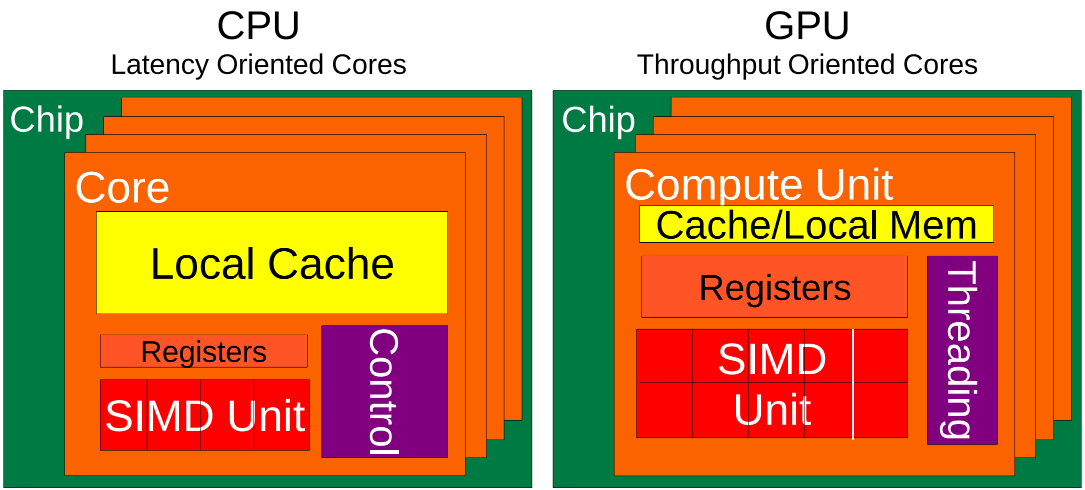
Original data collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond and C. Batten
Dotted line extrapolations by C. Moore

C Moore, *Data Processing in ExaScale-ClassComputer Systems*, Salishan, April 2011

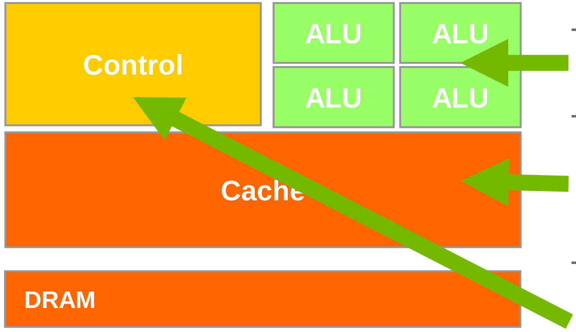
Rise of GPU Computing



CPU and GPU are designed very differently

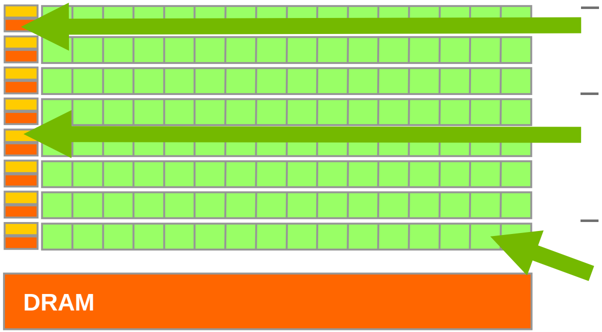


CPUs: Latency Oriented Design



- Powerful ALU
  - Reduced operation latency
- Large caches
  - Convert long latency memory accesses to short latency cache accesses
- Sophisticated control
  - Branch prediction for reduced branch latency
  - Data forwarding for reduced data latency

GPUs: Throughput Oriented Design



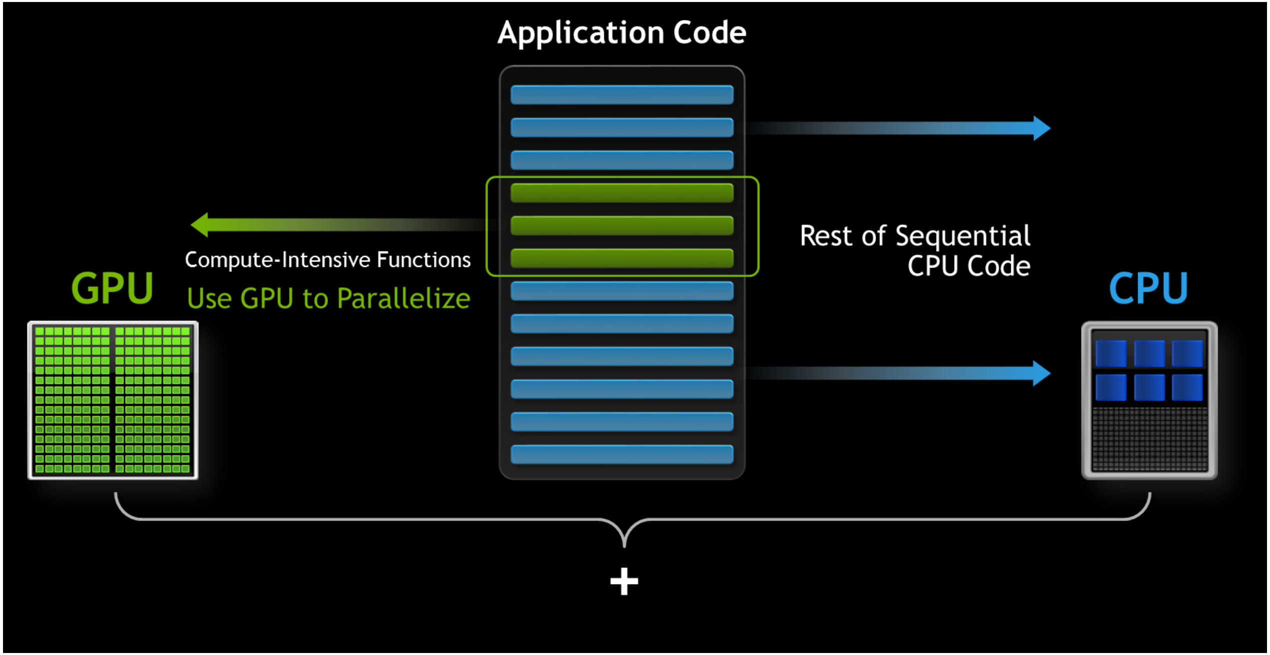
- Small caches
  - To boost memory throughput
- Simple control
  - No branch prediction
  - No data forwarding
- Energy efficient ALUs
  - Many, long latency but heavily pipelined for high throughput
- Require massive number of threads to tolerate latencies
  - Threading logic
  - Thread state

Winning Applications Use Both CPU and GPU

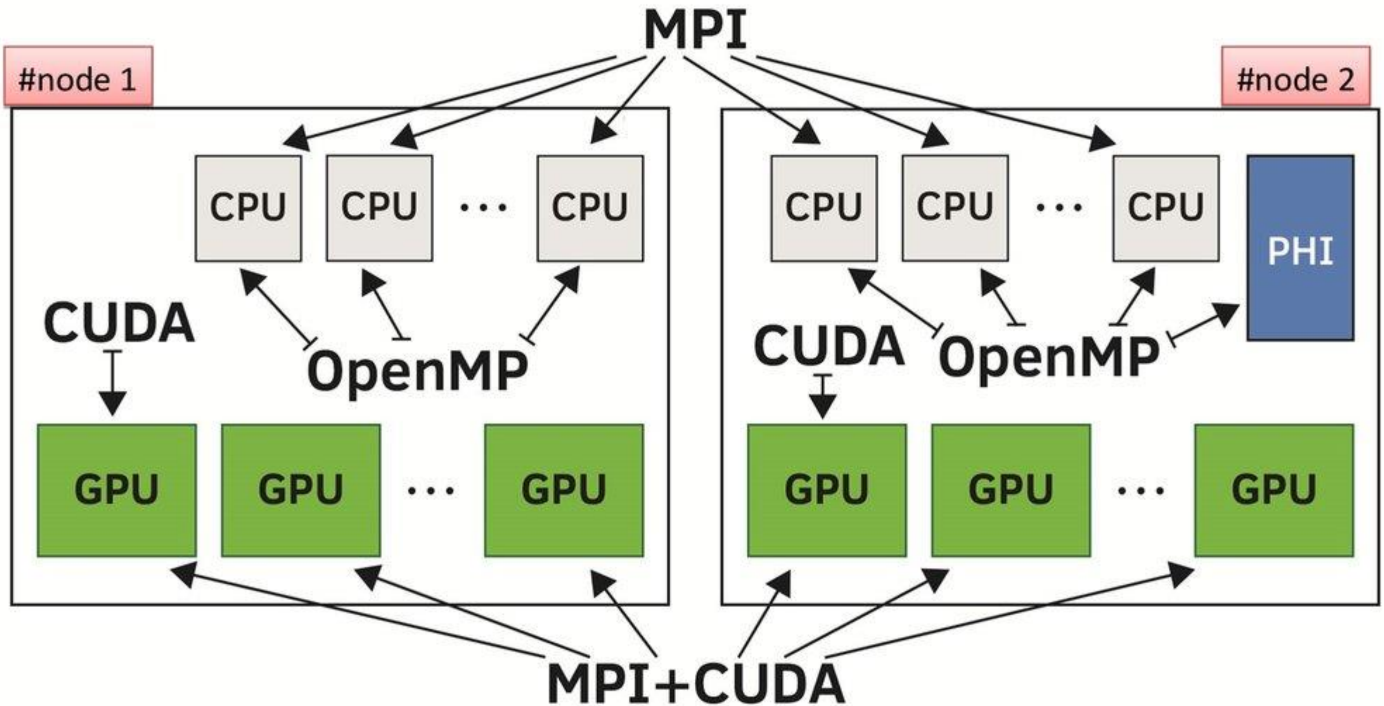
- CPUs for sequential parts where latency matters
  - CPUs can be 10X+ faster than GPUs for sequential code

- GPUs for parallel parts where throughput wins
  - GPUs can be 10X+ faster than CPUs for parallel code

Small changes, big speedup



Parallel technologies: levels of parallelism



How to control hybrid hardware:
MPI – OpenMP – CUDA - OpenCL ...

3 Ways to Accelerate Applications

Applications

Libraries	Compiler Directives	Programming Languages
Easy to use Most Performance	Easy to use Portable code	Most Performance Most Flexibility

Libraries: Easy, High-Quality Acceleration

- **Ease of use:** Using libraries enables GPU acceleration without in-depth knowledge of GPU programming
- **“Drop-in”:** Many GPU-accelerated libraries follow standard APIs, thus enabling acceleration with minimal code changes
- **Quality:** Libraries offer high-quality implementations of functions encountered in a broad range of applications

GPU Accelerated Libraries

Linear Algebra
FFT, BLAS,
SPARSE, Matrix

NVIDIA cuFFT, cuBLAS, cuSPARSE | CULA tools | MAGMA | CUSP

Numerical & Math
RAND, Statistics

IMSL Fortran Numerical Library | NVIDIA Math Lib | ArrayFire | NVIDIA cuRAND

Data Struct. & AI
Sort, Scan, Zero Sum

Thrust | GPU AI - Board Games | GPU AI - Path Finding | RAPIDS

Visual Processing
Image & Video

NVIDIA NPP | NVIDIA Video Encode | Sundog Software

Compiler Directives: Easy, Portable Acceleration

- **Ease of use:** Compiler takes care of details of parallelism management and data movement
- **Portable:** The code is generic, not specific to any type of hardware and can be deployed into multiple languages
- **Uncertain:** Performance of code can vary across compiler versions

OpenACC

– Compiler directives for C, C++, and FORTRAN

```
#pragma acc parallel loop
copyin(input1[0:inputLength],input2[0:inputLength]),
  copyout(output[0:inputLength])
  for(i = 0; i < inputLength; ++i) {
     output[i] = input1[i] + input2[i];
  }
```

Programming Languages: Most Performance and Flexible Acceleration

- **Performance:** Programmer has best control of parallelism and data movement
- **Flexible:** The computation does not need to fit into a limited set of library patterns or directive types
- **Verbose:** The programmer often needs to express more details

GPU Programming Languages

Numerical analytics	MATLAB Mathematica, LabVIEW
Fortran	CUDA Fortran
C/C++	CUDA C/C++
Python	PyCUDA, Copperhead, Numba
F#	Alea.cuBase

Session Outline (HPC)

Introduction	• Introduction to Heterogeneous Parallel Computing • How to program GPUs
Understanding OpenACC	• Benefits of using OpenACC • Understanding OpenACC compute directives • Applying OpenACC to a simple program • Explicit data management in OpenACC • Data movement and loop optimizations
OpenACC Hands-on	• Guided hands-on on applying OpenACC to conjugate gradient
GPU Computing with CUDA	• Introduction to CUDA C • CUDA memory model • CUDA thread model
CUDA Hands-on	• Guided hands-on on CUDA acceleration of XXX

Session Goals

– Learn how to program GPUs using OpenACC and CUDA
– Learn how to profile, analyze and optimize for GPU performance

Acknowledgement

- Wen-mei Hwu (University of Illinois)
- David Kirk (NVIDIA)
- Joe Bungo (NVIDIA)
- Mark Ebersole (NVIDIA)
- Abdul Dakkak (University of Illinois)
- Izzat El Hajj (University of Illinois)
- Andy Schuh (University of Illinois)
- John Stratton (Colgate College)
- Isaac Gelado (NVIDIA)
- John Stone (University of Illinois)
- Javier Cabezas (NVIDIA)
- Michael Garland (NVIDIA)

GPU Teaching Kit

Accelerated Computing

Naga Vydyanathan: nvydyanathan@nvidia.com

Prathu Bharti Tiwari: prtiwari@nvidia.com

Sayak Bhowmick: sayakb@nvidia.com