Data 604 – Data Management

Parallel and GPU Computing

Learning Objectives

- Understand the interaction between software and hardware
- To learn the major differences between latency devices (CPU cores) and throughput devices (GPU cores)
- To understand that optimal applications often make use of both CPUs and GPUs

Why study parallel computing?

- If you want your software/algorithms to run faster, you must understand the interaction between software and hardware

Parallel Computing – Flynn's Taxonomy*

- SISD- Single Instruction Single Data computing system is a uniprocessor machine that executes single instruction on a single data stream
- SIMD- Single Instruction Multiple Data is a computing system with several identical processors each with local memory and work under the control of a single instruction stream (GPUs).

* Python Parallel Programming Textbook

Parallel Computing – Flynn's Taxonomy*

- MISD- multiple instruction, single data
- MIMD- Multiple instruction, multiple data

* Python Parallel Programming Textbook

“Partition and Summarize”

- A commonly used strategy for processing large input data sets
  - There is no required order of processing elements in a data set (associative and commutative)
  - Partition the data set into smaller chunks
  - Have each thread to process a chunk
  - Use a reduction tree to summarize the results from each chunk into the final answer
- E.G., Google and Hadoop MapReduce frameworks support this strategy
- We will focus on the reduction tree step for now

Reduction enables other techniques

- Reduction is also needed to clean up after some commonly used parallelizing transformations
- Privatization
  - Multiple threads write into an output location
  - Replicate the output location so that each thread has a private output location (privatization)
  - Use a reduction tree to combine the values of private locations into the original output location

What is a reduction computation?

- Summarize a set of input values into one value using a "reduction operation"
  - Max
  - Min
  - Sum
  - Product
- Often used with a user defined reduction operation function as long as the operation
  - Is associative and commutative
  - Has a well-defined identity value (e.g., 0 for sum)
  - For example, the user may supply a custom "max" function for 3D coordinate data sets where the magnitude for the each coordinate data tuple is the distance from the origin.

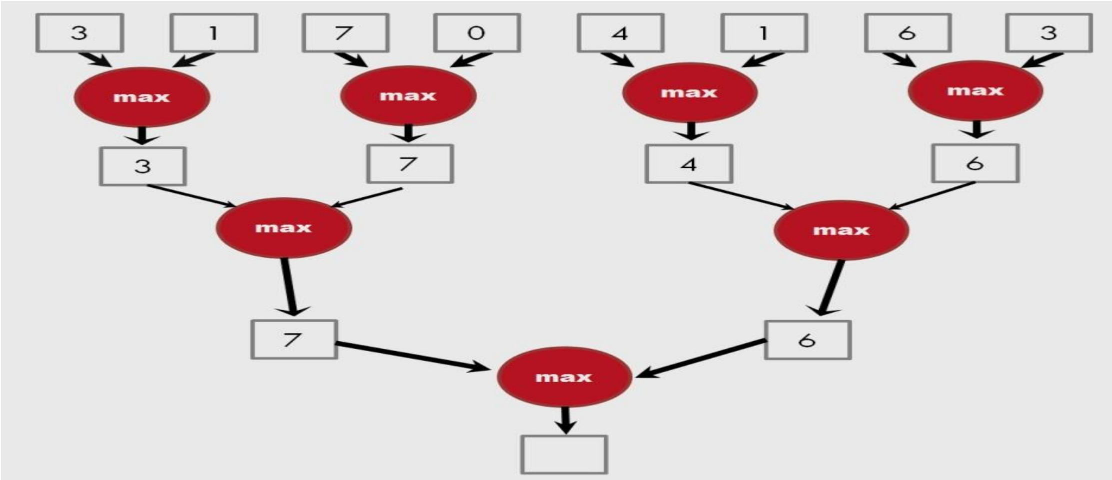
An example of "collective operation"

An Efficient Sequential Reduction O(N)

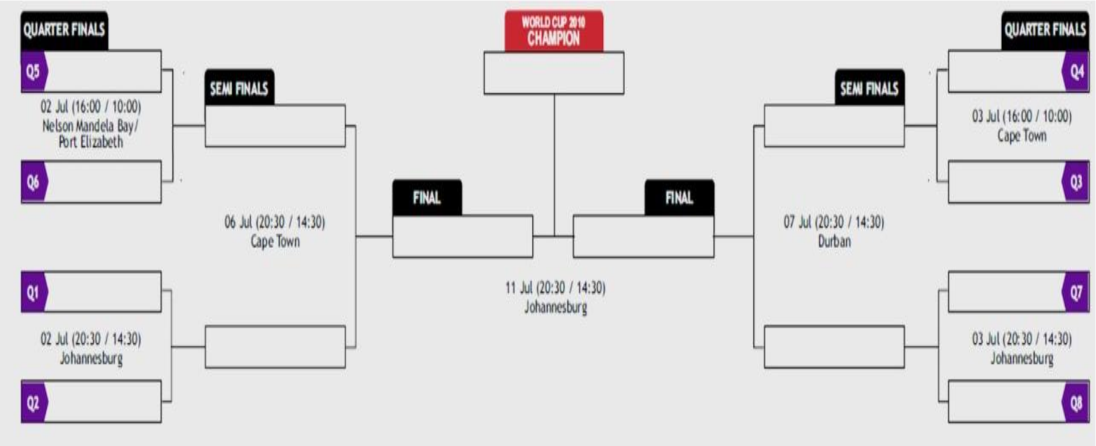
- Initialize the result as an identity value for the reduction operation
  - Smallest possible value for max reduction
  - Largest possible value for min reduction
  - 0 for sum reduction
  - 1 for product reduction
- Iterate through the input and perform the reduction operation between the result value and the current input value
  - N reduction operations performed for N input values
  - Each input value is only visited once – an O(N) algorithm
  - This is a computationally efficient algorithm.

9

A parallel reduction tree algorithm performs N-1 operations in log(N) steps



A tournament is a reduction tree with "max" operation



A Quick Analysis

- For N input values, the reduction tree performs
  - (1/2)N + (1/4)N + (1/8)N + … (1)N = (1- (1/N))N = N-1 operations
  - In Log (N) steps – 1,000,000 input values take 20 steps
    - Assuming that we have enough execution resources
  - Average Parallelism (N-1)/Log(N))
    - For N = 1,000,000, average parallelism is 50,000
    - However, peak resource requirement is 500,000
    - This is not resource efficient
- This is a work-efficient parallel algorithm
  - The amount of work done is comparable to the an efficient sequential algorithm
  - Many parallel algorithms are not work efficient

Speed v. Throughput

NVIDIA

Speed

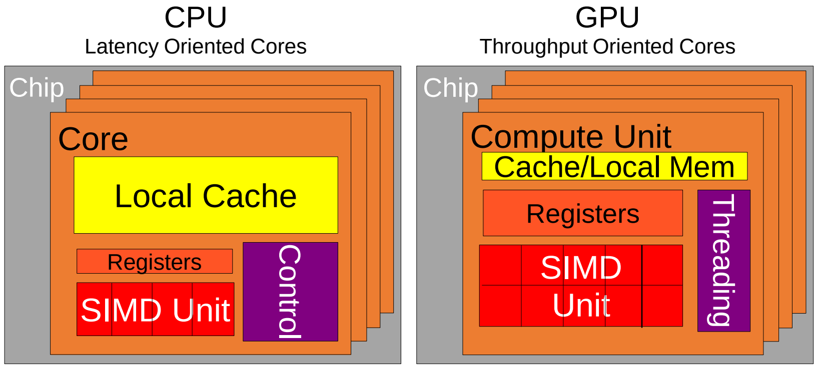
Throughput

Which is better depends on your needs…

*Images from Wikimedia Commons via Creative Commons

© NVIDIA 2013

CPU and GPU are designed very differently



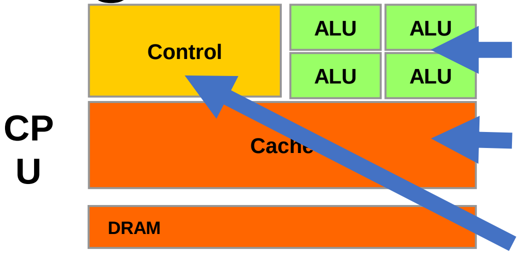
© NVIDIA 2013

CPU

- Latency – beginning to end duration of performing a single computation (*Tuomanen, 2018)
- CPUs are engineered to reduce the latency of a single computation.
- Computations are sequential
- Fewer cores that are restricted with the amount of processes it can compute but it handles those processes very fast

*Hands-On GPU Programming with Python and CUDA

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

© NVIDIA 2013

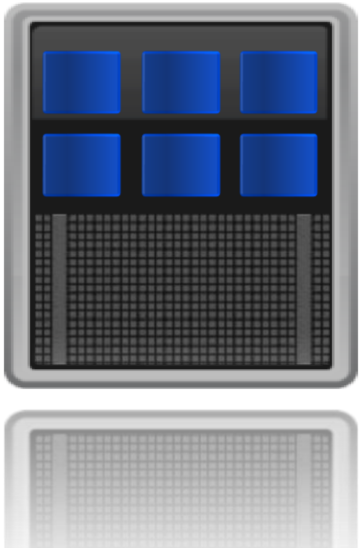
Accelerated Computing

10x Performance & 5x Energy Efficiency for HPC

NVIDIA

CPU

Optimized for Serial Tasks



CPU Strengths

- Very large main memory
- Very fast clock speeds
- Latency optimized via large caches
- Small number of threads can run very quickly

CPU Weaknesses

- Relatively low memory bandwidth
- Cache misses very costly
- Low performance/watt

GPU

- Each core is much simpler than that of the CPU and each core on its own is not as fast as a CPU
- It’s the amount of cores that make a difference. GPU has hundreds to thousands of cores compared to a CPU which has 1 to 6 cores
- Computations are done in parallel, asynchronously.
- Still relies on CPU for managing and passing data
- Programs must be rewritten to enable parallel processing

Accelerated Computing

10x Performance & 5x Energy Efficiency for HPC

GPU Strengths

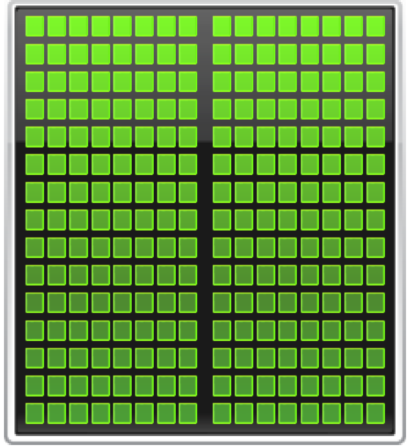
- High bandwidth main memory
- Significantly more compute resources
- Latency tolerant via parallelism
- High throughput
- High performance/watt

GPU Weaknesses

- Relatively low memory capacity
- Low per-thread performance

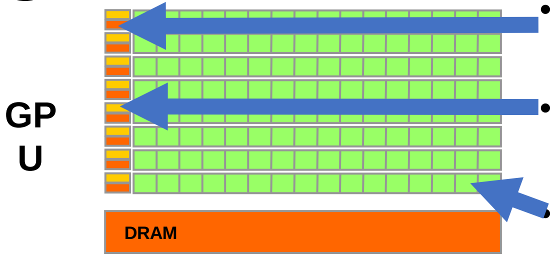
GPU Accelerator

Optimized for Parallel Tasks



© NVIDIA 2013

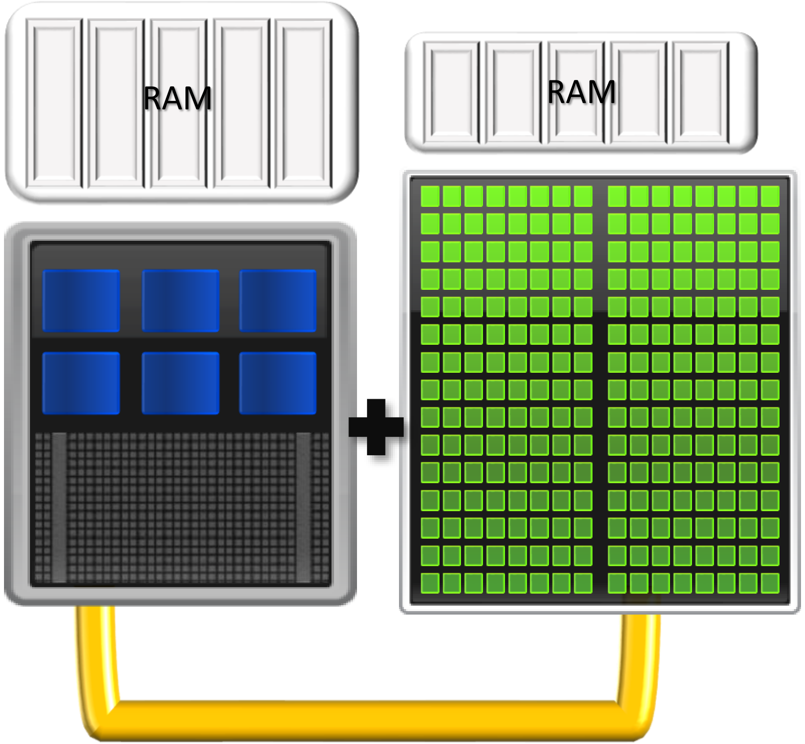
GPUs: Throughput Oriented Design



- 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

© NVIDIA 2013

Accelerator Nodes



CPU and GPU have distinct memories

- CPU generally larger and slower
- GPU generally smaller and faster

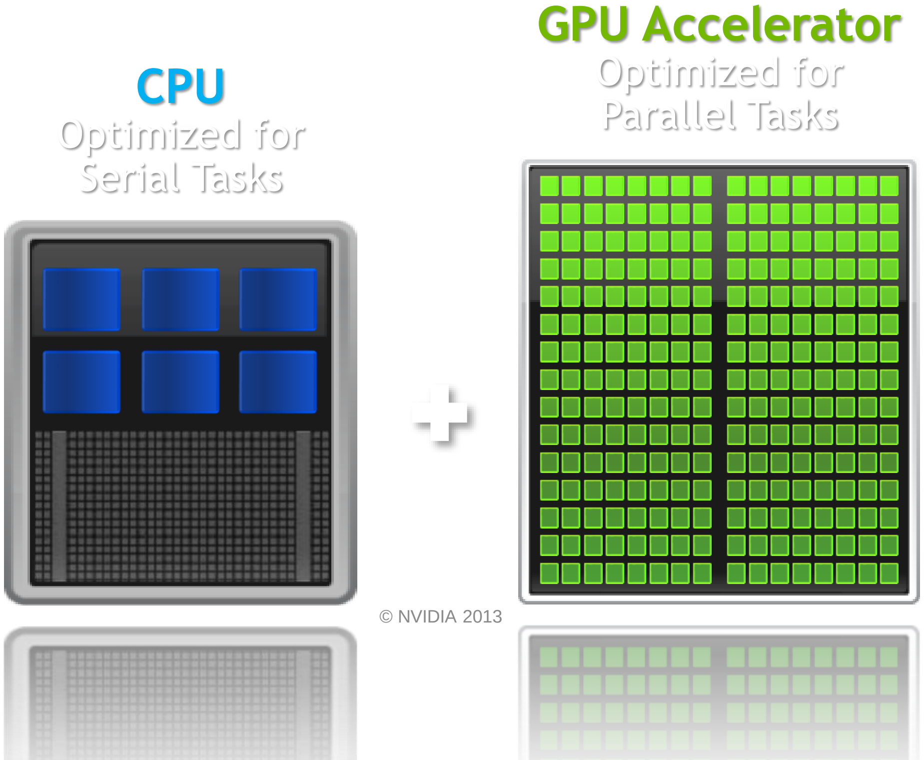
CPU and GPU communicate via PCIe

- Data must be copied between these memories over PCIe
- PCIe Bandwidth is much lower than either memories

Emerging Tech - Nvlink

Accelerated Computing

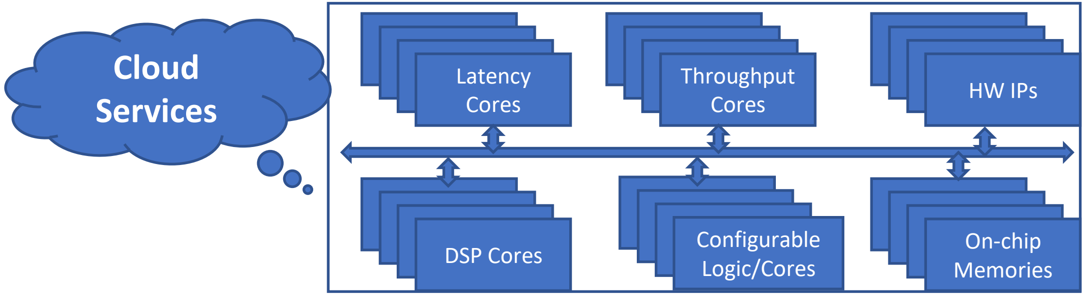
10x Performance & 5x Energy Efficiency for HPC



© NVIDIA 2013

Heterogenous Parallel Computing

- Both GPU and CPU
- Use the best match for the job (heterogeneity in mobile System Of Chip)



© NVIDIA 2013

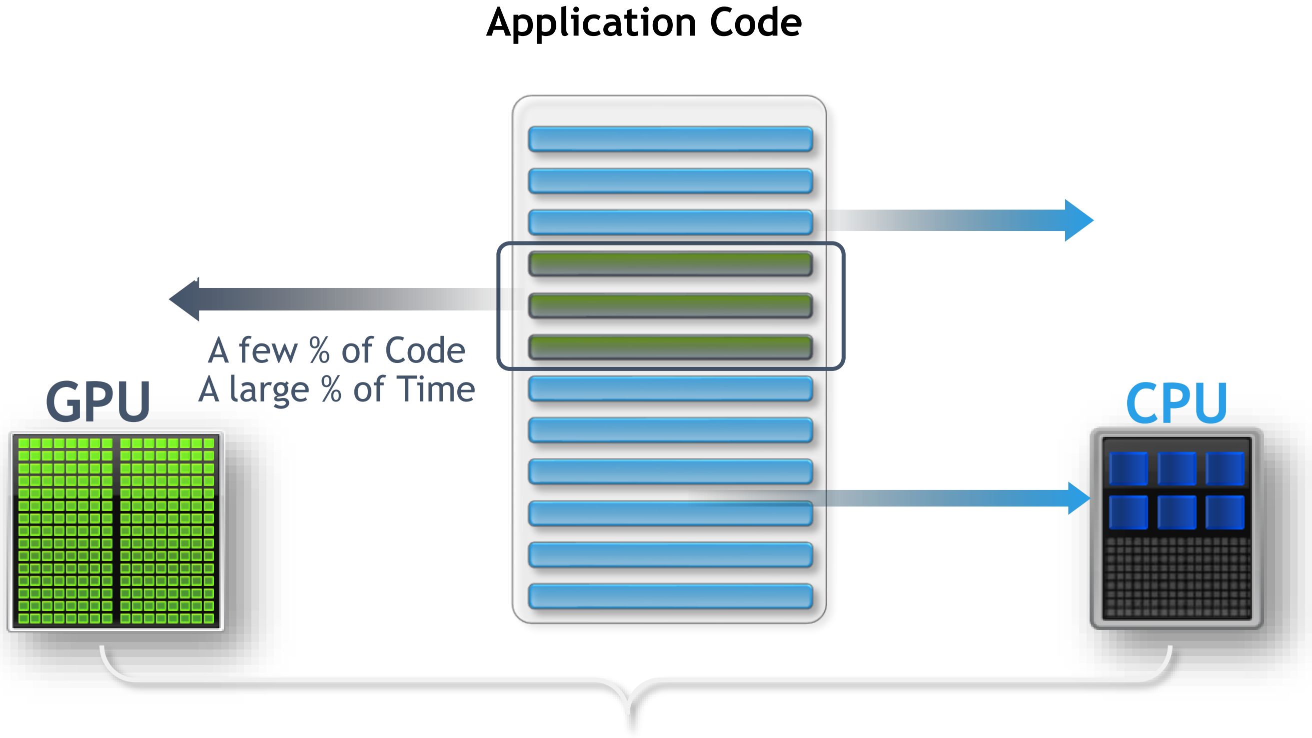
Winning Applications Use Both CPU and GPU

- 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

© NVIDIA 2013

What is Heterogeneous Programming?



© NVIDIA 2013

Amdahl’s Law (*Tuomanen, 2018)

$$\text{Speedup} = \frac{1}{(1 - p) + p / N}$$

p= proportion of code that can be parallelizable

N= number of GPU cuda cores

*Hands-On GPU Programming with Python and CUDA

GPU Programming Languages

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

© NVIDIA 2013

Nvidia CUDA

- Development environment and ecosystem to enable GPU applications
- Integrates with programming languages, GPU libraries and deep learning frameworks
- https://developer.nvidia.com/cuda-toolkit

Deep Learning – Neural Networks

- AlexNet https://en.wikipedia.org/wiki/AlexNet – Deep Convolutional Neural Network
- Resnet https://en.wikipedia.org/wiki/Residual_neural_network – Residual Neural Network
- Numerous types of Neural Networks. Helpful info on neural network architecture can be found at https://www.asimovinstitute.org/author/fjodorvanveen/

Popular GPU Frameworks

- CAFFE https://caffe.berkeleyvision.org/
- MXNet https://mxnet.apache.org/versions/1.7.0/
- Tensorflow https://www.tensorflow.org/
- PYTORCH https://pytorch.org/

Developer Tools - Debuggers

NSIGHT

CUDA-GDB

CUDA MEMCHECK

NVIDIA Provided

allinea DDT

TotalView®

3rd Party

https://developer.nvidia.com/debugging-solutions

© NVIDIA 2013

Python – GPU Enabled Libraries

- Tensorflow GPU https://www.tensorflow.org/install/gpu
- Numpy https://numpy.org/ - highly performance optimized C code that can be run from within Python
- PyCuda https://pypi.org/project/pycuda/
- RAPIDS https://rapids.ai/ - new GPU data science library that is modeled to provide similar look and feel of Panda and Scikit Learn libraries

Sample Python Code - PyCuda

```
import pycuda.gpuarray as gpuarray
import pycuda.driver as cuda
import pycuda.autoinit
import numpy

a_gpu = gpuarray.to_gpu(numpy.random.randn(5,5).astype(numpy.float32))
a_doubled = (2*a_gpu).get()
print ("ORIGINAL MATRIX")
print a_doubled
print ("DOUBLED MATRIX AFTER PyCUDA EXECUTION USING GPUARRAY CALL")
print a_gpu
```

In Class Lab – Google CoLab

- You can access and sign up for this at: https://colab.research.google.com/notebooks/welcome.ipynb#scrollTo=-Rh3-Vt9Nev9 .
- Watch the 'Intro to Google Colab'
- Run the Tensorflow GPU notebook https://colab.research.google.com/notebooks/gpu.ipynb
- Attach GPU to Tensorflow GPU notebook.
- Run code
- RAPIDS Notebook: https://colab.research.google.com/drive/1rY7Ln6rEE1pOlfSHCYOVaqt8OvDO35J0#forceEdit=true&sandboxMode=true&scrollTo=B0C8lV5TQnjN

This work is protected by United States copyright laws and is provided solely for the use of instructors in teaching their courses and assessing student learning. Dissemination or sale of any part of this work (including on the World Wide Web) will destroy the integrity of the work and is not permitted. The work and materials from it should never be made available to students except by instructors using the accompanying text in their classes. All recipients of this work are expected to abide by these restrictions and to honor the intended pedagogical purposes and the needs of other instructors who rely on these materials.