CPSC 4240/5240: Parallel Programming Techniques

Lecture 18: Vectorization and GPUs

Lecturer: Quanquan C. Liu

Spring 2026

Final Project

- If you'd like for me to comment on your final project ideas:
  - Post a private message on Ed Discussion
- Now till end of semester: Parallel GPU programming

Vectorization in NLP

Example Application of Vectorization: NLP

- What Is Vectorization in NLP?
  - **Definition**: Converting words, sentences, or documents into numerical vectors
- Why do we need it?
  - Machine learning algorithms require numerical input
  - Enables measurement of similarity, semantic relatedness, etc.

Classic Approaches to Word Vectorization

- **One-Hot Encoding**
  - Large sparse vectors
  - High dimensionality (vocabulary size)
  - Lacks semantic relationships (no notion of distance except orthogonal vs. identical)

Vocabulary={"cat", "dog", "cow", "lion", "bird"}

Classic Approaches to Word Vectorization

- **One-Hot Encoding**

Vocabulary={"cat", "dog", "cow", "lion", "bird"}

- If we index them as:
  - **cat** → index 0
  - **dog** → index 1
  - **cow** → index 2
  - **lion** → index 3
  - **bird** → index 4

cat → [1, 0, 0, 0, 0]
dog → [0, 1, 0, 0, 0]
cow → [0, 0, 1, 0, 0]
lion → [0, 0, 0, 1, 0]
bird → [0, 0, 0, 0, 1]

Vectorization in NLP

- **Vectorization techniques**:
  - Bag of words: Extracts features from the text; calculate frequency of words in document
  - **TF-IDF: Information retrieval, keyword extraction**
  - Word2Vec: Semantic analysis task
  - GloVe: Word analogy, named entity recognition tasks
  - BERT: language translation, question answering system

TF-IDF Vectorization

Bag-of-Words (BoW)

- **Definition**:
  - A representation where each document (sentence, paragraph, or article) is described by the frequencies of each word (or token) it contains, irrespective of word order.
- Procedure:
  - Define a vocabulary of all unique words
  - For each document, build a numerical vector of word counts or frequencies
  - A **sparse vector** because most words do not appear in each document

Bag-of-Words (BoW)

- Procedure:
  - Define a vocabulary of all unique words
  - For each document, build a numerical vector of word counts or frequencies
  - A **sparse vector** because most words do not appear in each document
- **Limitations**:
  - Treats all words equally; doesn't differentiate between datasets
  - Doesn't capture word order or context

TF-IDF

- **Definition**: Term Frequency–Inverse Document Frequency
  - Weighting scheme that adjusts how much weight a word contributes based on **how frequently it appears in a specific document**
  - And **how rarely it appears** across all documents
- Why use this?
  - Words like *"the"*, *"a"*, *"and"* might be very frequent but carry little distinguishing power for a particular document
  - Uncommon or domain-specific words (e.g., *"hematology"*, *"tensor"*) are more indicative
  - Emphasizes **unique words**

TF-IDF Formula

- Term Frequency (TF)
  - The number of times a term t appears in a document d

$$tf(t,d) = \frac{count\ of\ t\ in\ d}{total\ number\ of\ terms\ in\ d}$$

- Inverse Document Frequency (IDF)
  - Downweight terms that appear in many documents, upweight rare terms

TF-IDF Formula

- Term Frequency (TF)

$$tf(t,d) = \frac{count\ of\ t\ in\ d}{total\ number\ of\ terms\ in\ d}$$

- Inverse Document Frequency (IDF)
  - Downweight terms that appear in many documents, upweight rare terms

$$idf(t,N) = \log\left(\frac{N}{1+df(t)}\right)$$

  - N := total number of documents
  - $df(t)$:= number of documents containing t

TF-IDF Formula

- TF-IDF Formula:

$$tf - idf(t, d, N) = tf(t, d) \cdot idf(t, N)$$

- A term will have a **high tf-idf** weight if:
  - It occurs many times in a single document; $tf(t, d)$ is high
  - It appears in relatively few documents; $idf(t, N)$ is high
- Conversely, words common to many documents (e.g., “the”, “and”) get a low tf-idf

TF-IDF Example

- Suppose you have 3 documents:
  - **D1**: “the cat sat on the mat”
  - **D2**: “the dog chased the cat”
  - **D3**: “the lion roared”

- Step 1: Vocabulary

{the, cat, sat, on, mat, dog, chased, lion, roared}

TF-IDF Example

- Suppose you have 3 documents:
  - **D1**: “the cat sat on the mat”
  - **D2**: “the dog chased the cat”
  - **D3**: “the lion roared”

Term frequency for **D1**:

$tf(\text{“}the\text{”})$?

- Step 1: Vocabulary

{the, cat, sat, on, mat, dog, chased, lion, roared}

TF-IDF Example

- Suppose you have 3 documents:
  - **D1**: “the cat sat on the mat”
  - **D2**: “the dog chased the cat”
  - **D3**: “the lion roared”

Term frequency for **D1**:

$$tf(\text{“}the\text{”}) \text{ ?}$$

$$\boldsymbol{tf(\text{“}the\text{”}) = \frac{2}{6}}$$

- Step 1: Vocabulary

{the, cat, sat, on, mat, dog, chased, lion, roared}

TF-IDF Example

- Suppose you have 3 documents:
  - **D1**: “the cat sat on the mat”
  - **D2**: “the dog chased the cat”
  - **D3**: “the lion roared”

IDF:

$$idf(\text{“}the\text{”}) \text{ ?}$$

- Step 1: Vocabulary

{the, cat, sat, on, mat, dog, chased, lion, roared}

TF-IDF Example

- Suppose you have 3 documents:
  - **D1**: “the cat sat on the mat”
  - **D2**: “the dog chased the cat”
  - **D3**: “the lion roared”

IDF:

$$idf(\text{“}the\text{”}) \; ?$$

$$\log\left(\frac{3}{4}\right) < 1$$

- Step 1: Vocabulary

{the, cat, sat, on, mat, dog, chased, lion, roared}

TF-IDF Example

- Suppose you have 3 documents:
  - **D1**: “the cat sat on the mat”
  - **D2**: “the dog chased the cat”
  - **D3**: “the lion roared”

Term frequency for **D3**:

$$tf(\text{“}lion\text{”}) \text{ ?}$$

- Step 1: Vocabulary

{the, cat, sat, on, mat, dog, chased, lion, roared}

TF-IDF Example

- Suppose you have 3 documents:
  - **D1**: “the cat sat on the mat”
  - **D2**: “the dog chased the cat”
  - **D3**: “the lion roared”

Term frequency for **D3**:

$$tf(\text{“}lion\text{”}) \text{ ?}$$

$$\boldsymbol{tf(\text{“}lion\text{”}) = \frac{1}{3}}$$

- Step 1: Vocabulary

{the, cat, sat, on, mat, dog, chased, lion, roared}

TF-IDF Example

- Suppose you have 3 documents:
  - **D1**: “the cat sat on the mat”
  - **D2**: “the dog chased the cat”
  - **D3**: “the lion roared”

IDF:

$$idf(\text{“}lion\text{”}) \; ?$$

$$\log\left(\frac{3}{2}\right) > 1$$

- Step 1: Vocabulary

{the, cat, sat, on, mat, dog, chased, lion, roared}

TF-IDF Example

- Suppose you have 3 documents:
  - **D1**: “the cat sat on the mat”
  - **D2**: “the dog chased the cat”
  - **D3**: “the lion roared”

TF-IDF:

$$tf - idf(\text{“}lion\text{”}) \text{ ?}$$

- Step 1: Vocabulary

{the, cat, sat, on, mat, dog, chased, lion, roared}

TF-IDF Example

- Suppose you have 3 documents:
  - **D1**: “the cat sat on the mat”
  - **D2**: “the dog chased the cat”
  - **D3**: “the lion roared”

TF-IDF:

$$tf - idf(\text{“}lion\text{”}) \text{ ?}$$

$$\frac{1}{3} \cdot \log\left(\frac{3}{2}\right)$$

- Step 1: Vocabulary

{the, cat, sat, on, mat, dog, chased, lion, roared}

TF-IDF Example

- Suppose you have 3 documents:
  - **D1**: “the cat sat on the mat”
  - **D2**: “the dog chased the cat”
  - **D3**: “the lion roared”

TF-IDF:

$$tf - idf(\text{“}lion\text{”}) \text{ ?}$$

$$\frac{1}{3} \cdot \log\left(\frac{3}{2}\right) >> \frac{1}{3} \cdot \log\left(\frac{3}{4}\right)$$

- Step 1: Vocabulary

{the, cat, sat, on, mat, dog, chased, lion, roared}

TF-IDF Example

- Suppose you have 3 documents:
  - **D1**: “the cat sat on the mat”
  - **D2**: “the dog chased the cat”
  - **D3**: “the lion roared”

- Step 1: Vocabulary

  {the, cat, sat, on, mat, dog, chased, lion, roared}

- Final step:
  - Matrix of numbers: each row corresponds to a document
  - Each column is a vocabulary term
  - Each value is a TF-IDF number

TF-IDF Afterwards

- TF-IDF matrix: sparse but large
  - **Dimensionality reduction**: reduce the dimension of the matrix via methods like Singular Value Decomposition (SVD), PCA, or Truncated SVD
- Passed into ML Models
  - Classification (Logistic Regression, Naïve Bayes, SVM)
  - Clustering (kNN, k-Means, Hierarchical Clustering)
  - Similarity-Based Retrieval Systems (Recommendation systems)

Parallelization in GPUs

Parallel Hardware – SIMD

- In the vector addition example from last class, a SIMD unit with a width of four could execute four iterations of the loop at once
- All current GPUs are **based on SIMD hardware**
  - The GPU hardware implicitly maps each thread to a SIMD "lane/core"
    - The programmer does not need to consider the SIMD hardware for correctness, just for performance
  - This model of running threads on SIMD hardware is referred to as **Single Instruction Multiple Threads (SIMT)**

Modern GPUs

- Why SIMD for GPUs?
  - GPUs are designed to handle **large numbers of parallel operations** (e.g., thousands or millions of pixels, particles, or matrix elements)
  - SIMD helps process many data elements efficiently at once
  - Rather than separate hardware for each “thread” as a fully independent CPU core, **GPUs share control logic among multiple lanes** (the SIMD unit)
  - GPU hardware might have wide vector units (e.g., 32 elements on NVIDIA, 64 on AMD)

Modern GPUs: Thread Mapping to SIMD Lanes

- **Implicit Mapping**
  - When writing a GPU kernel (e.g., in CUDA or OpenCL), typically spawn many “threads.”
  - Under the hood, GPU groups these threads in **warps** (NVIDIA) or **wavefronts** (AMD).
  - Each warp/wavefront is executed by a SIMD unit
- Each thread within a warp is **mapped to a distinct SIMD “lane”**
- **Hardware** executes each group of threads in lockstep (synchronously)
  - Don’t manually program lanes

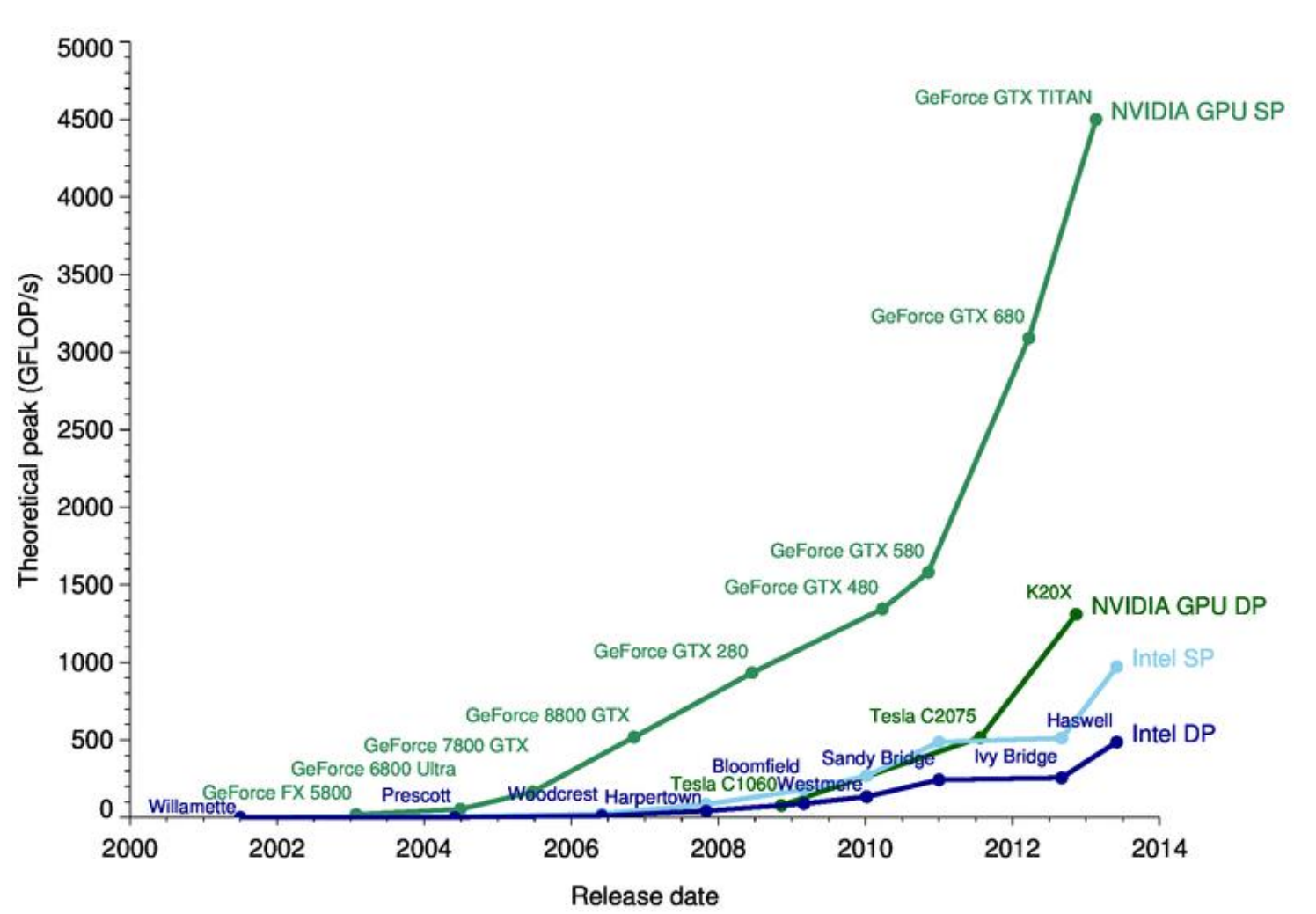
The SIMT Model: Single Instruction, Multiple Threads

- **Definition**
  - **SIMT** stands for **Single Instruction, Multiple Threads**
  - Describes how GPUs implement a “thread-centric” programming model on top of underlying SIMD hardware

The SIMT Model: Single Instruction, Multiple Threads

- **How does it work?**
  - **Programmer's View**: See thousands (even millions) of parallel threads, each with its own thread ID
  - **Hardware Reality**:
    - Threads are grouped into fixed-size batches (e.g., 32 threads per warp on NVIDIA)
    - A single instruction fetch/decode is then broadcast to all threads in the warp
    - Each thread corresponds to one SIMD lane
  - **Synchronization**: Within a warp, threads implicitly run in lockstep when they are executing the same instructions

Why Study Parallelism on GPUs?



https://michaelgalloy.com/2013/06/11/cpu-vs-gpu-performance.html

But there are challenges

- GPUs deliver massive speedups—but only under the right conditions
- CPUs remain critical for workloads with **complex control flow or irregular access patterns**
- GPU architecture: high throughput, low flexibility
  - Uniform data access patterns (**data parallelism** as opposed to task parallelism)
  - Poor at handling divergent code paths or dynamic workloads

But there are challenges

- GPU architecture: high throughput, low flexibility
  - Uniform data access patterns (**data parallelism** as opposed to task parallelism)
  - Poor at handling divergent code paths or dynamic workloads

Bad example:

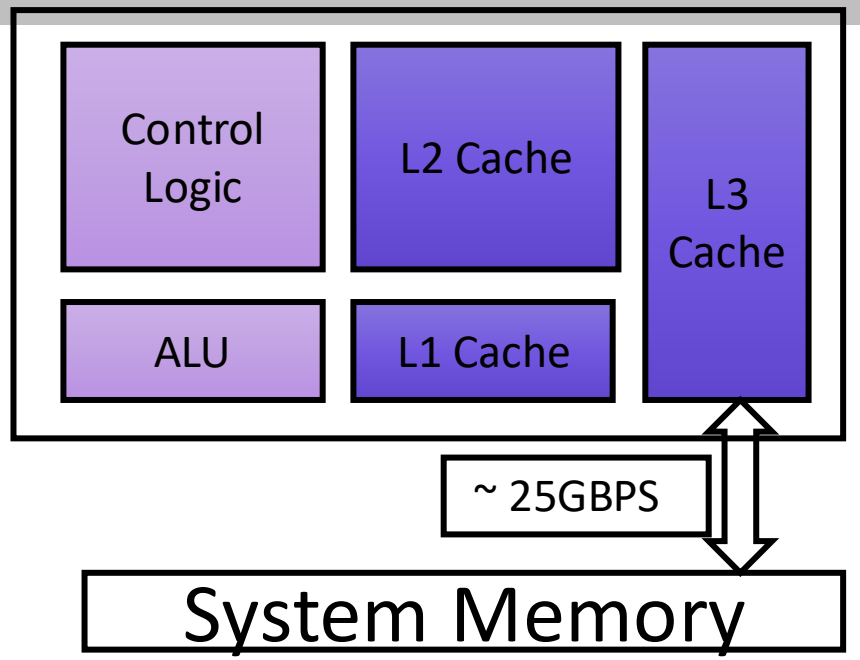
```
if (x[i] > 0)
   A[i] = sqrt(x[i]);
else
   A[i] = log(-x[i]);
```

Real-World Examples: Irregular Workloads

Task Type	GPU Performance	CPU Performance
Matrix Multiply (dense)	✅ High	⚠️ Moderate
Binary Tree Traversal	⚠️ Poor	✅ High
Sorting (radix)	✅ High	✅ High
Regex Parsing	⚠️ Poor	✅ High
Neural Net Inference	✅ High	⚠️ Lower

Conventional CPU Architecture

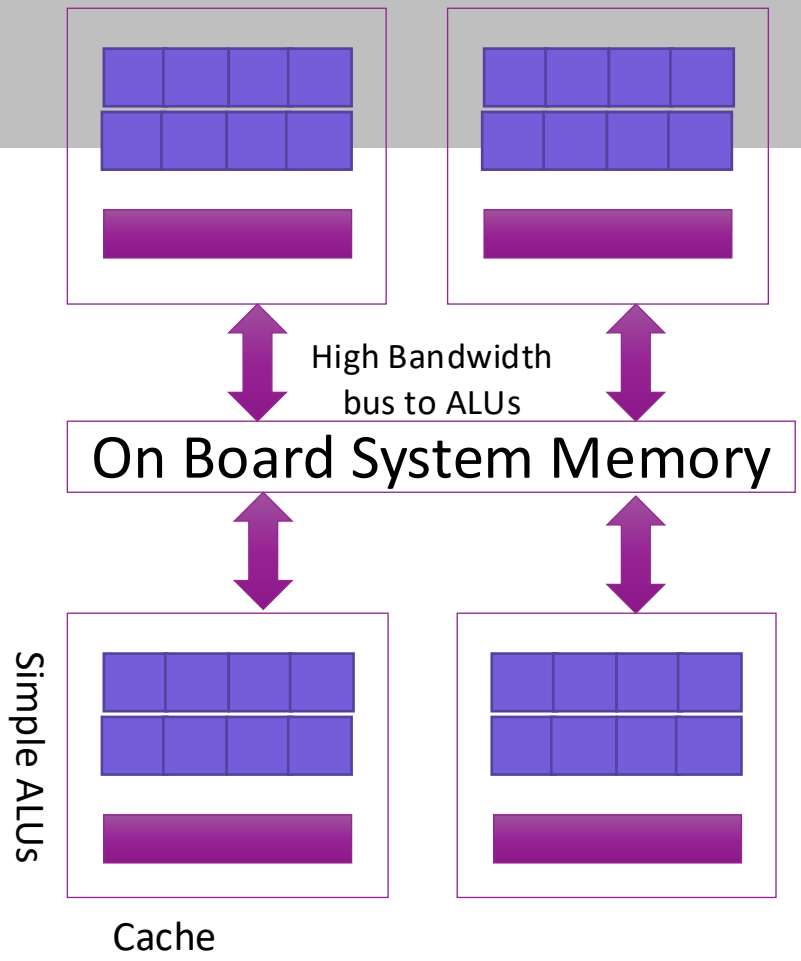
- Space devoted to control logic instead of ALU
- CPUs are optimized to minimize the latency of a single thread
  - Can efficiently handle control flow intensive workloads
- Multi-level caches used to hide latency
- Limited number of registers due to smaller number of active threads



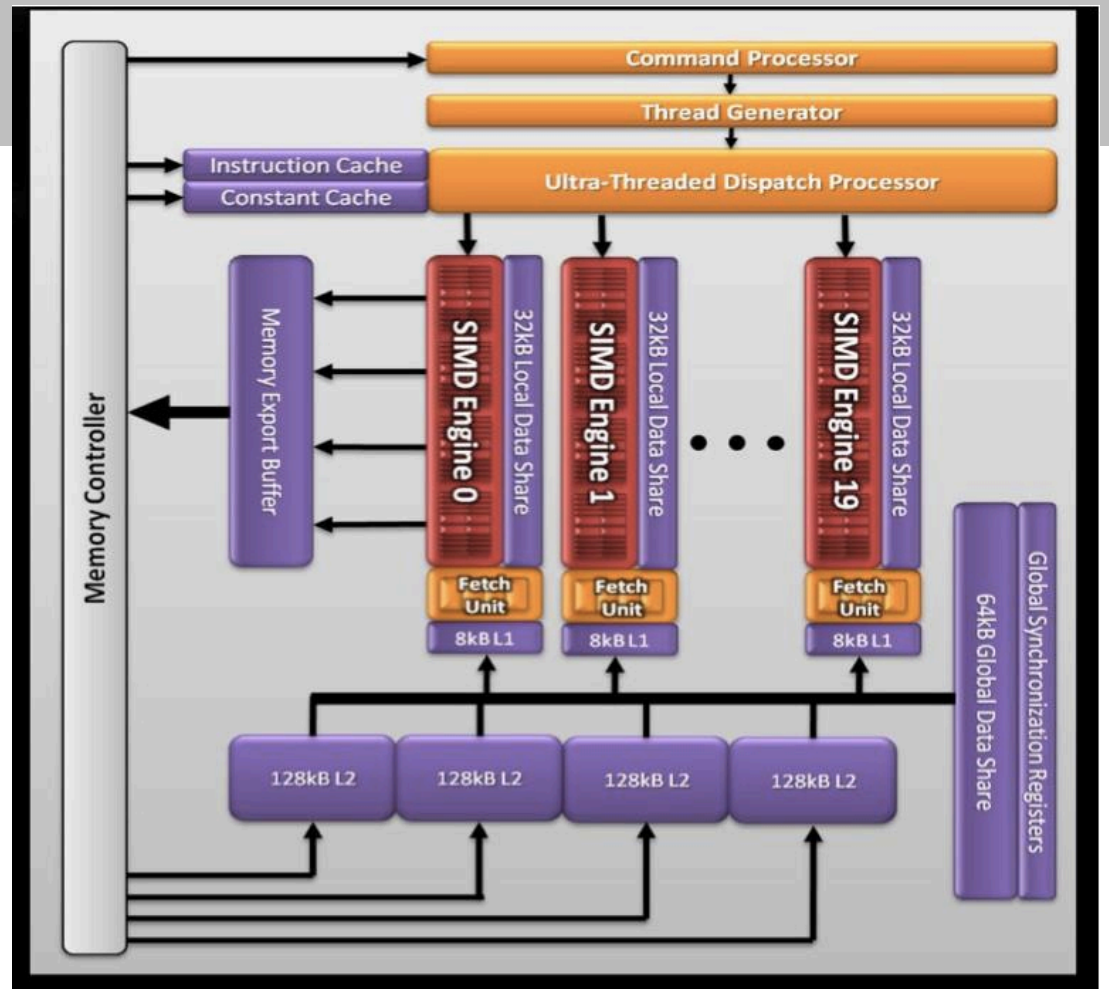
A present day multicore CPU could have more than one ALU (typically < 32) and some of the cache hierarchy is usually shared across cores

Modern GPU Architecture

- Generic many core GPU
  - Less space devoted to control logic and caches
  - Large register files to support multiple thread contexts
- Low latency hardware managed thread switching
- Large number of ALU per “core” with small user managed cache per core
- Memory bus optimized for bandwidth
  - ~150 GBPS bandwidth allows us to service a large number of ALUs simultaneously



AMD GPU Hardware Architecture

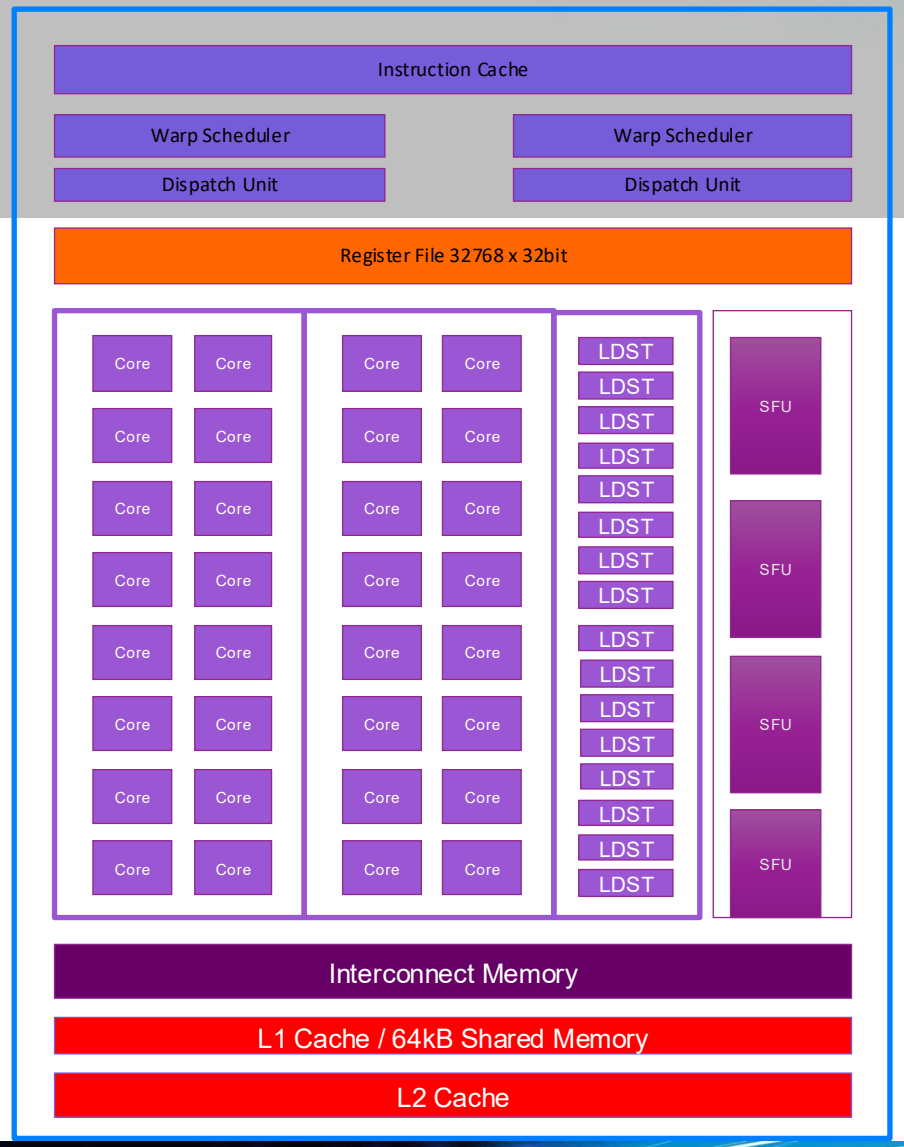


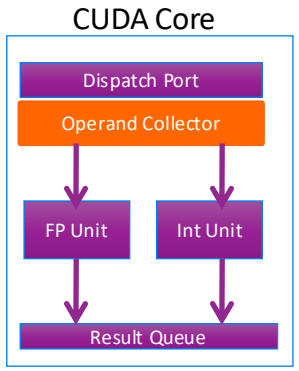
- AMD 5870 – Cypress
- 20 SIMD engines
- 16 SIMD units per core
- 5 multiply-adds per functional unit (VLIW processing)
- 2.72 Teraflops Single Precision
- 544 Gigaflops Double Precision

Source: Introductory OpenCL *SAAHPC2010,* Benedict R. Gaster

Nvidia GPUs - Fermi Architecture

- GTX 480 - Compute 2.0 capability
  - 15 cores or Streaming Multiprocessors (SMs)
  - Each SM features 32 CUDA processors
  - 480 CUDA processors
- Global memory with ECC





Source: NVIDIA's Next Generation CUDA Architecture Whitepaper

Parallelization on GPUs

Challenges of Parallelization

- On CPUs, **hardware-supported atomic operations** are used to enable concurrency
  - Atomic operations allow data to be **read and written without intervention from another thread**
- Some GPUs support system-wide atomic operations, but with a large performance trade-off
  - Usually code that requires **global synchronization** is not well suited for GPUs
  - Any problem that is decomposed using **input data partitioning** (i.e., requires results to be combined at the end) will likely need to be restructured to execute well on a GPU

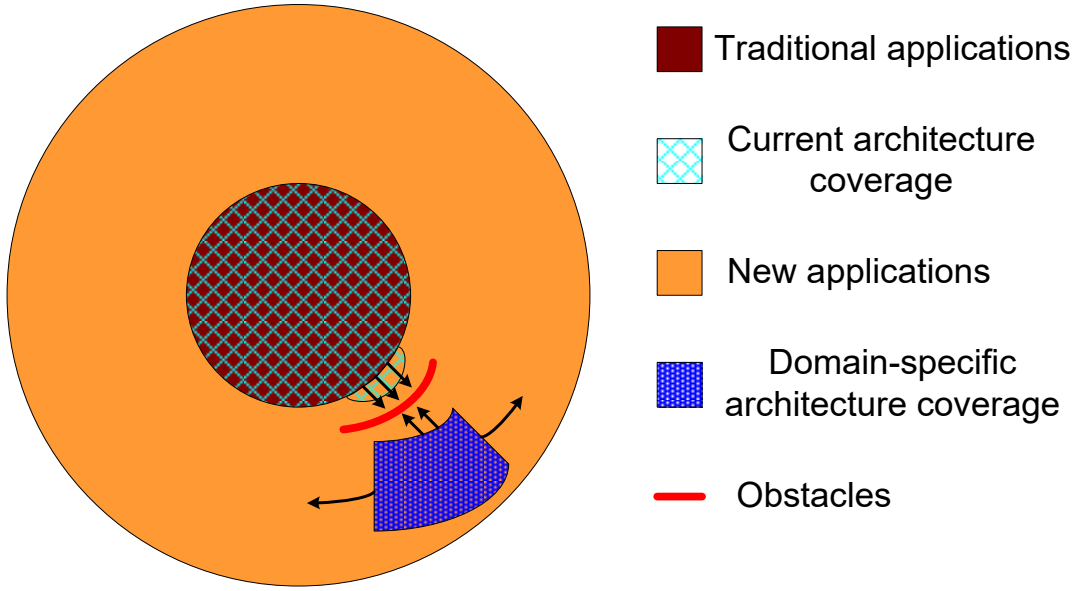
General-Purpose Computing on GPUs (GPGPU)

Concept of GPGPU (General-Purpose Computing on GPUs)

- Idea:
  - Potential for very high performance at low cost
  - Architecture well suited for certain kinds of parallel applications ***(data parallel)***
  - Demonstrations of ***30-100x*** speedup over CPU
- Early challenges:
  - Architectures very customized to graphics problems (e.g., vertex and fragment processors)
  - Programmed using graphics-specific programming models or libraries
- Recent trends:
  - Some convergence between commodity and GPUs and their associated parallel programming models

Stretching Traditional Architectures

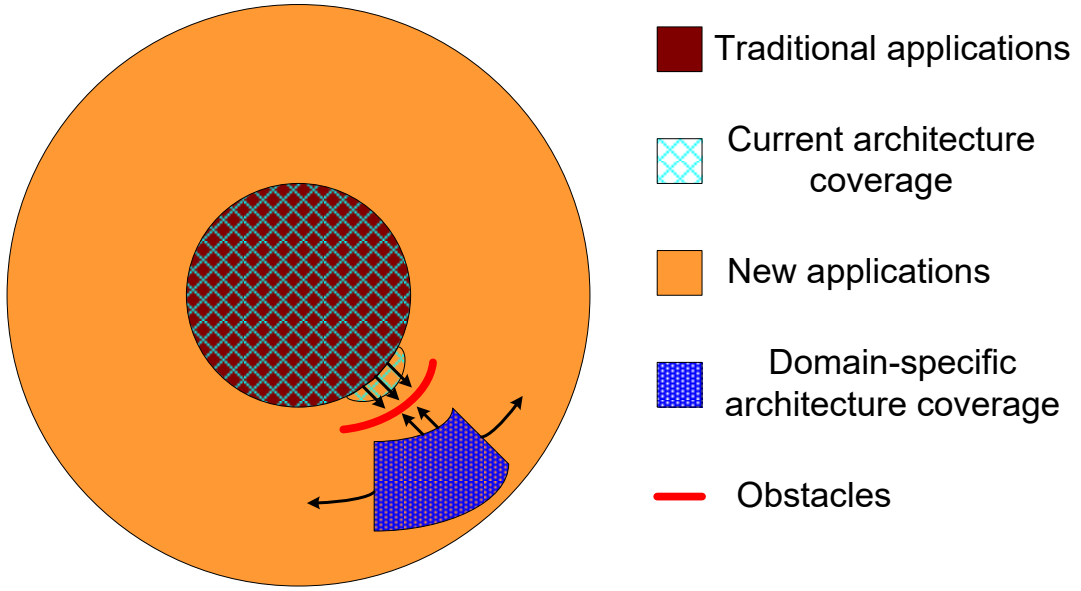
- Traditional parallel architectures cover some super-applications
  - DSP, GPU, network apps, Scientific, Transactions
- The game is to grow mainstream architectures “out” or domain-specific architectures “in”
  - CUDA is latter



Stretching Traditional Architectures

Hit a "memory wall"!

- Traditional parallel architectures cover some super-applications
  - DSP, GPU, network apps, Scientific, Transactions
- The game is to grow mainstream architectures "out" or domain-specific architectures "in"
  - CUDA is latter



Memory Wall

- **Memory wall** refers to disparity between how quickly processors (CPUs or GPUs) can perform computations versus how quickly data can be moved to and from main memory
- As processors get faster, ability to process data **outpaces ability of memory systems** to supply it
- Leaves the CPU/GPU idle or underutilized while it waits for data to arrive from memory
- Insufficient memory bandwidth or high memory latency can negate the benefits of massive parallelism in GPUs
- GPU-based computing (e.g., CUDA) often needs **careful attention to memory optimizations**
  - **Efficient data layouts, caching, shared memory usage**

Overcoming the Memory Wall

- **Cache Hierarchies**: Multiple levels of caches to keep data closer to the core and reduce round trips to main memory
- **On-Chip Memory/Shared Memory**: GPUs often provide small, fast on-chip memories that threads can share
- **Prefetching and Data Locality**: Software and hardware techniques try to fetch data into caches before it's needed, taking advantage of predictable access patterns
- **High-Bandwidth Memory (HBM):** New memory technologies aim to increase memory bandwidth to better keep pace with processor speeds

The fastest computer in the world today

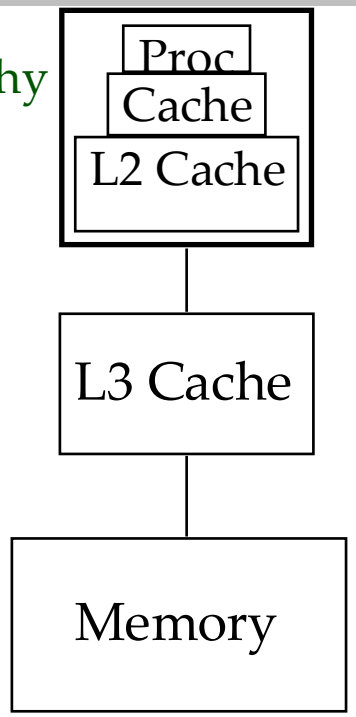
• What is its name?	El Capitan
• Where is it located?	Los Alamos National Laboratory
• How many processors does it have?	11,039,616 combined CPU and GPU cores based on AMD
• What kind of processors?	AMD 4th generation EPYC processors; AMD Instinct MI300A accelerators
• How fast is it?	1.742 exaFLOPs/second; One quintillion operations/s 1 x 10^{18}

See http://www.top500.org

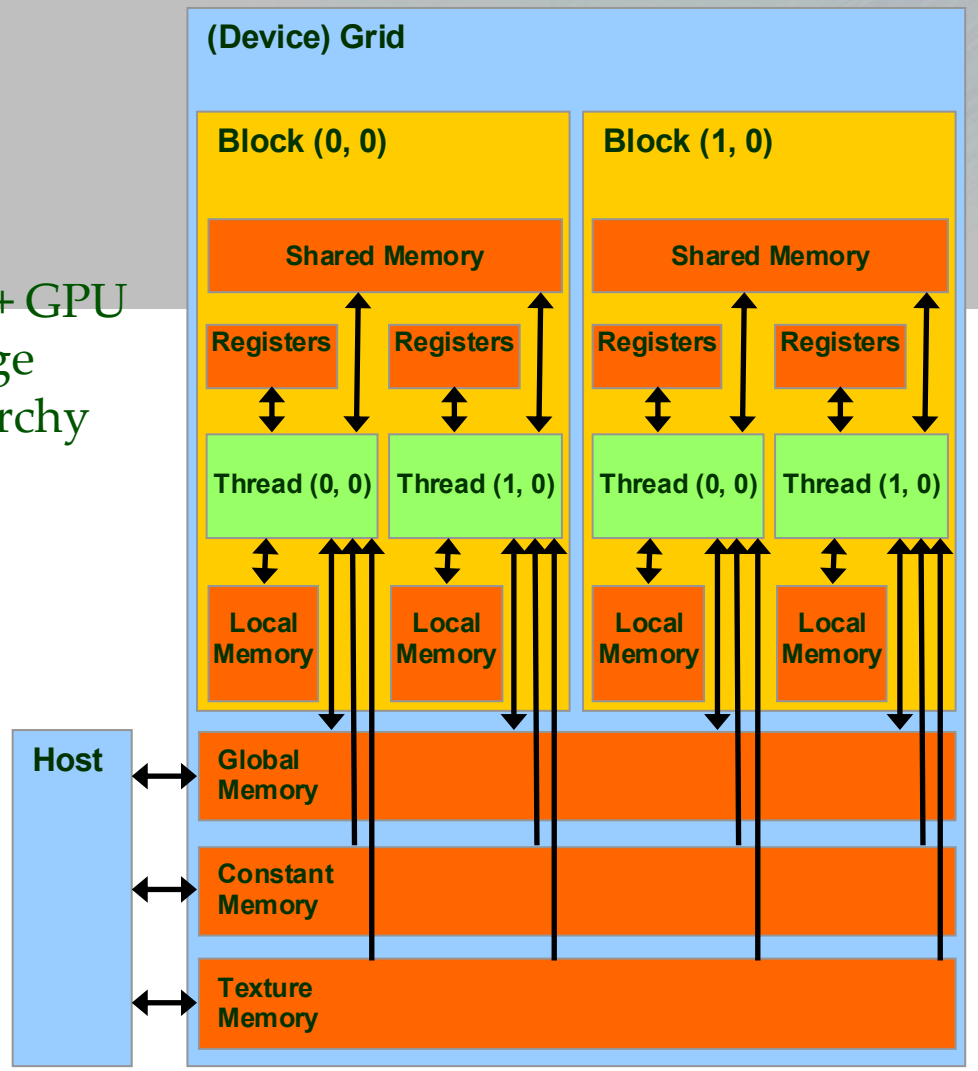
Locality and Parallelism

- Large memories are slow, fast memories are small
- Storage hierarchies are large and fast on average
- Algorithm should do most work on nearby data

Conventional Storage Hierarchy



Host + GPU Storage Hierarchy



Courtesy NVIDIA