GPUs

A bit of history

A bit of history

- Originally designed to render graphics for gaming

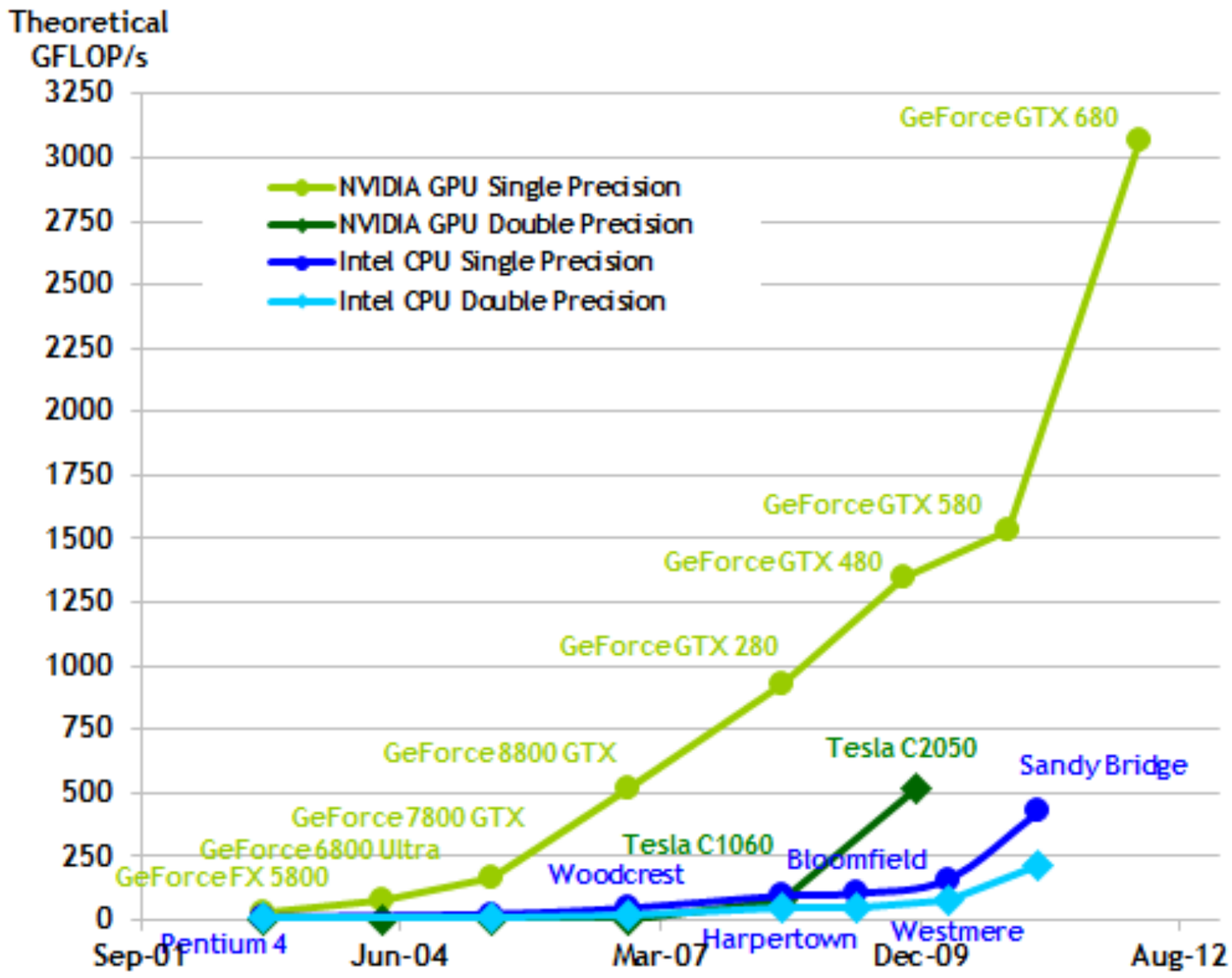
A bit of history

- Originally designed to render graphics for gaming
- First GPGPU applications were developed via Shader techniques

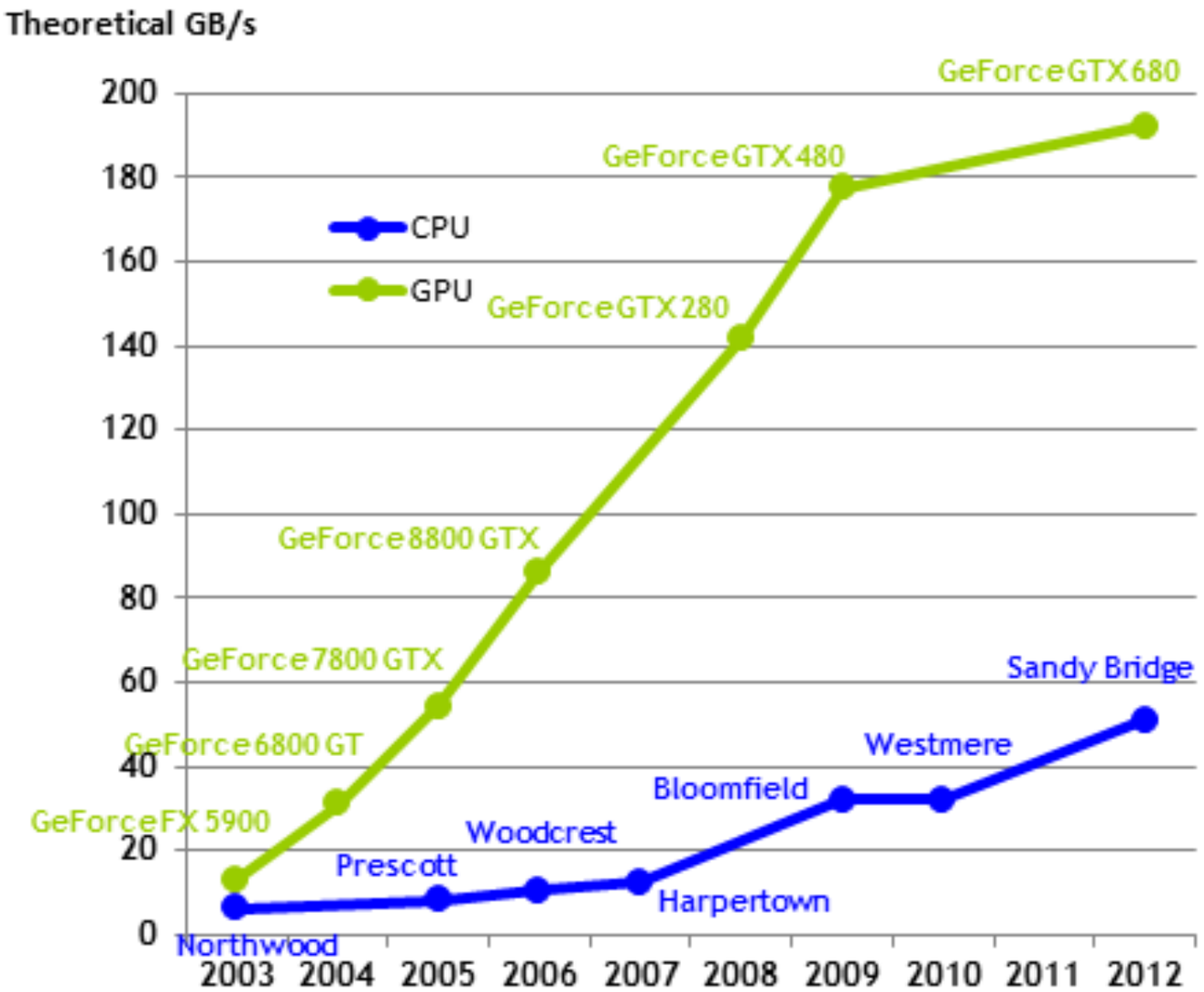
A bit of history

- Originally designed to render graphics for gaming
- First GPGPU applications were developed via Shader techniques
- CUDA / OpenCL became the first set of languages written specifically compute purposes

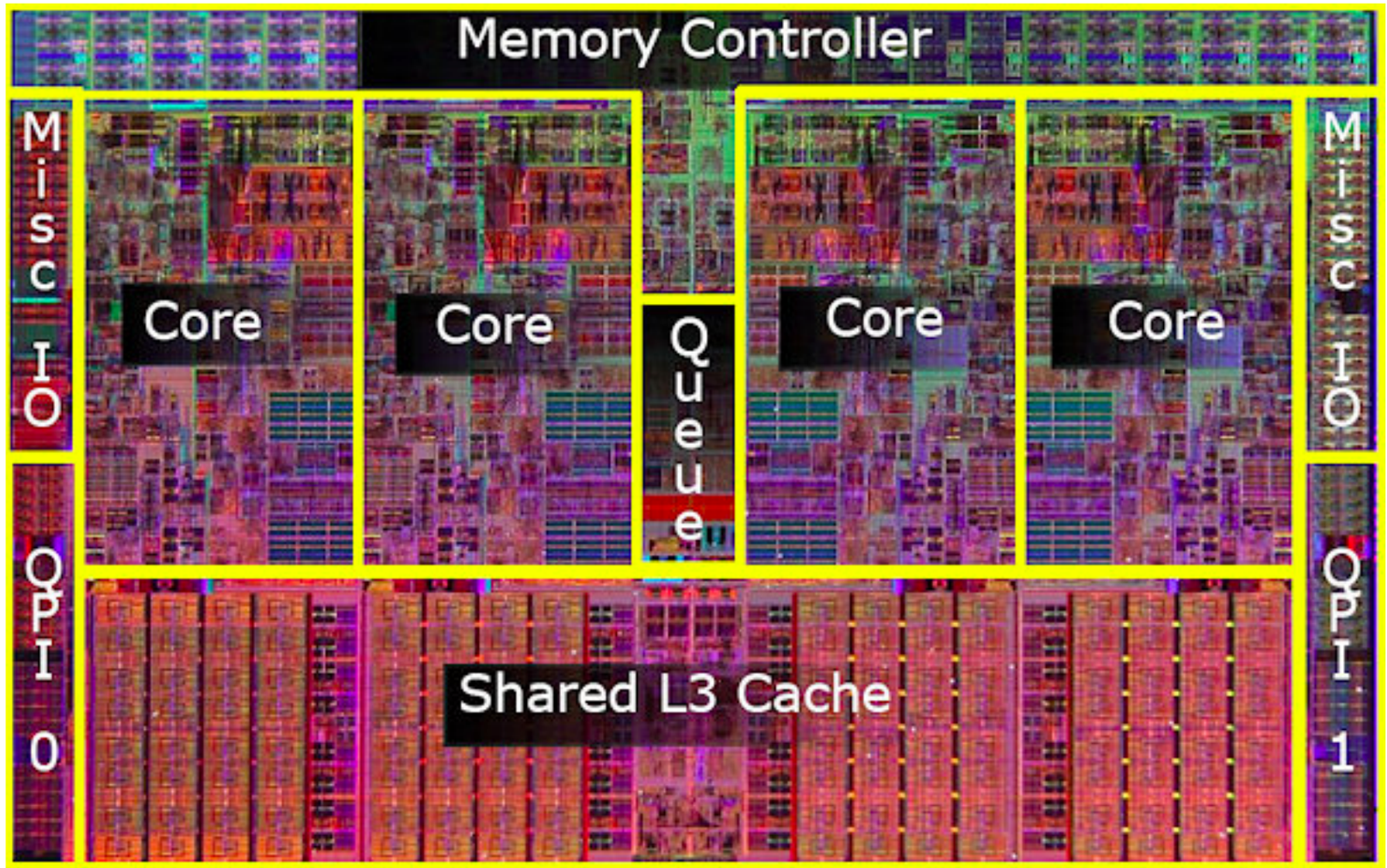
Performance over the years



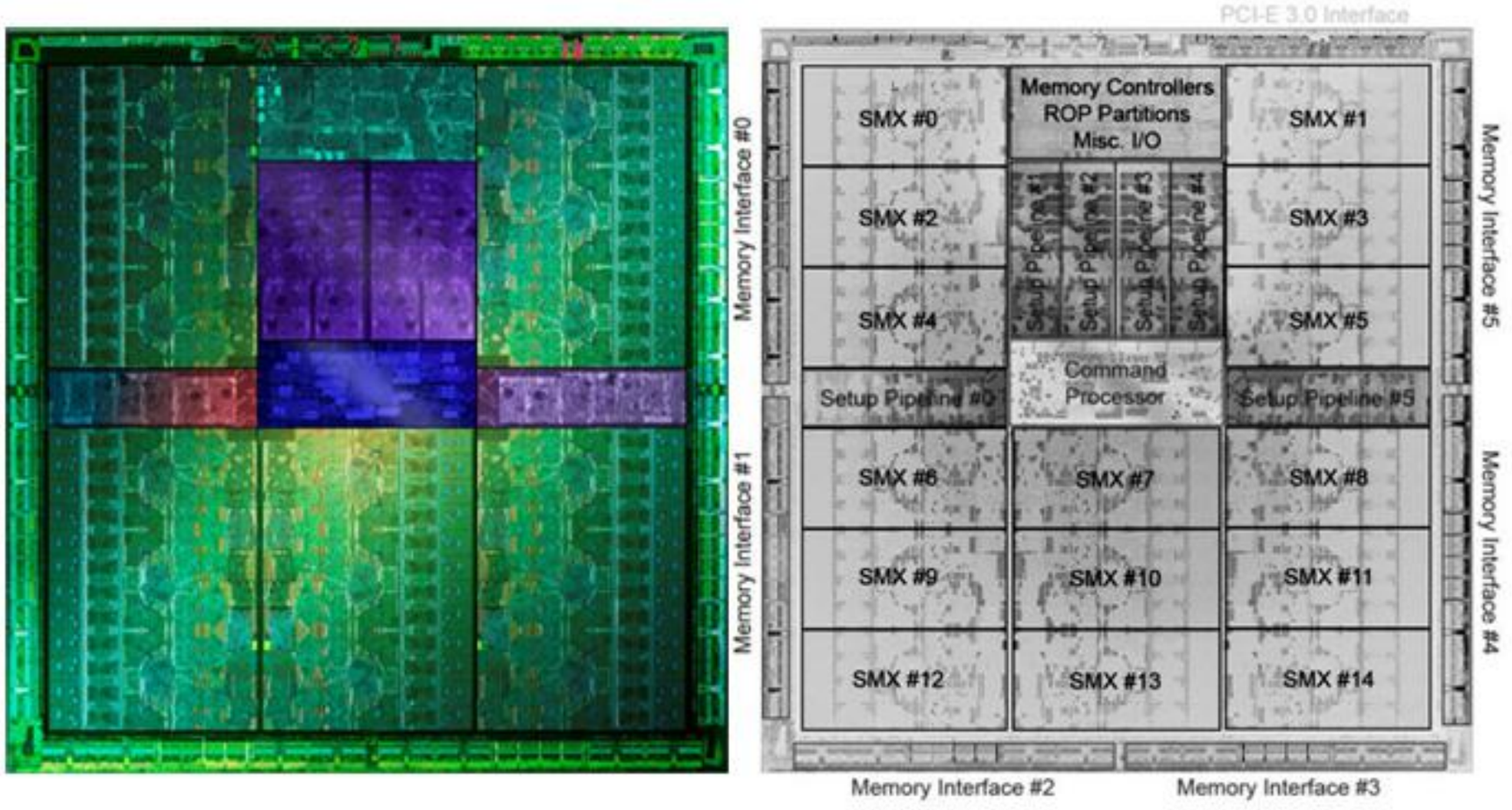
Performance over the years



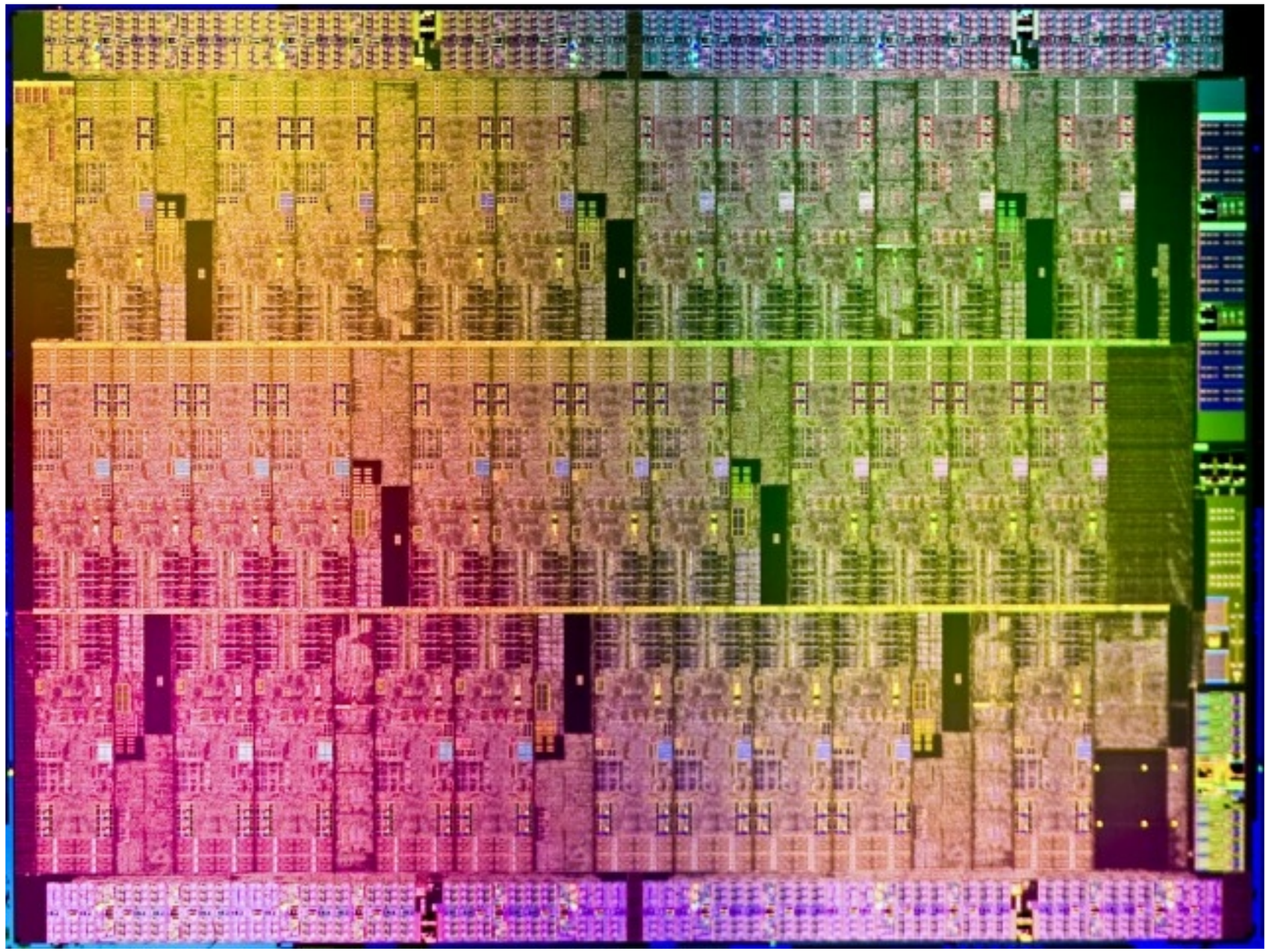
CPU Architecture



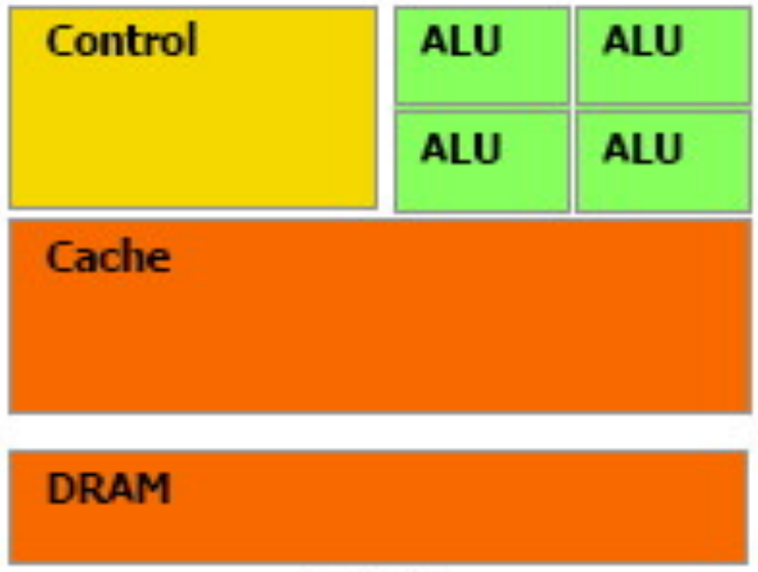
GPU Architecture



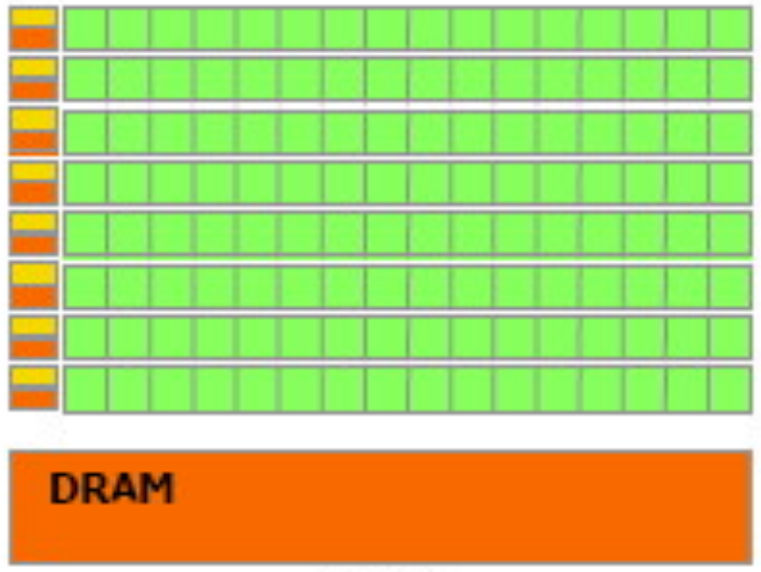
Any Guesses?



Allocation of Transistors



CPU



GPU

Memory Space

Memory Space

- Separate Memory Space:

Memory Space

- Separate Memory Space:
  - on board “host” memory (DDR3, etc.)

Memory Space

- Separate Memory Space:
  - on board "host" memory (DDR3, etc.)
  - on GPU "device" memory

Memory Space

- Separate Memory Space:
  - on board "host" memory (DDR3, etc.)
  - on GPU "device" memory
    - Global memory (Slowest)

Memory Space

- Separate Memory Space:
  - on board "host" memory (DDR3, etc.)
  - on GPU "device" memory
    - Global memory (Slowest)
    - L1, L2 Cache (Fast)

Memory Space

- Separate Memory Space:
  - on board "host" memory (DDR3, etc.)
  - on GPU "device" memory
    - Global memory (Slowest)
    - L1, L2 Cache (Fast)
    - Registers (Super-Fast)

Memory Space

- Separate Memory Space:
  - on board "host" memory (DDR3, etc.)
  - on GPU "device" memory
    - Global memory (Slowest)
    - L1, L2 Cache (Fast)
    - Registers (Super-Fast)
- Memory Management is critical!

Programming Paradigm

Programming Paradigm

- Host code (CPU) launches GPU threads to execute GPU code

Programming Paradigm

- Host code (CPU) launches GPU threads to execute GPU code
- A single line is code executed by N threads

Programming Paradigm

- Host code (CPU) launches GPU threads to execute GPU code
- A single line is code executed by N threads
- if() statements create divergence

Pitfalls

Pitfalls

- Trying to syncthreads() in a divergent block of code

Pitfalls

- Trying to syncthreads() in a divergent block of code
- Race conditions

Pitfalls

- Trying to syncthreads() in a divergent block of code
- Race conditions
- Hiding latency

Pitfalls

- Trying to syncthreads() in a divergent block of code
- Race conditions
- Hiding latency
- Pre-mature optimization

Algorithms

Algorithms

- What does it mean to parallelize an algorithm?

Algorithms

- What does it mean to parallelize an algorithm?
- Ideal: do same amount of total work as serial counter part

Algorithms

- What does it mean to parallelize an algorithm?
  - Ideal: do same amount of total work as serial counter part
  - Less Ideal: do a bit of extra work

Example: Prefix Sum

Example: Prefix Sum

- Problem:

Example: Prefix Sum

- Problem:
  - Given an array of N elements, find its running sum, ex. Exclusive Prefix Sum:

Example: Prefix Sum

- Problem:
  - Given an array of N elements, find its running sum, ex. Exclusive Prefix Sum:
    - 1 2 4 5 6 8 INPUT

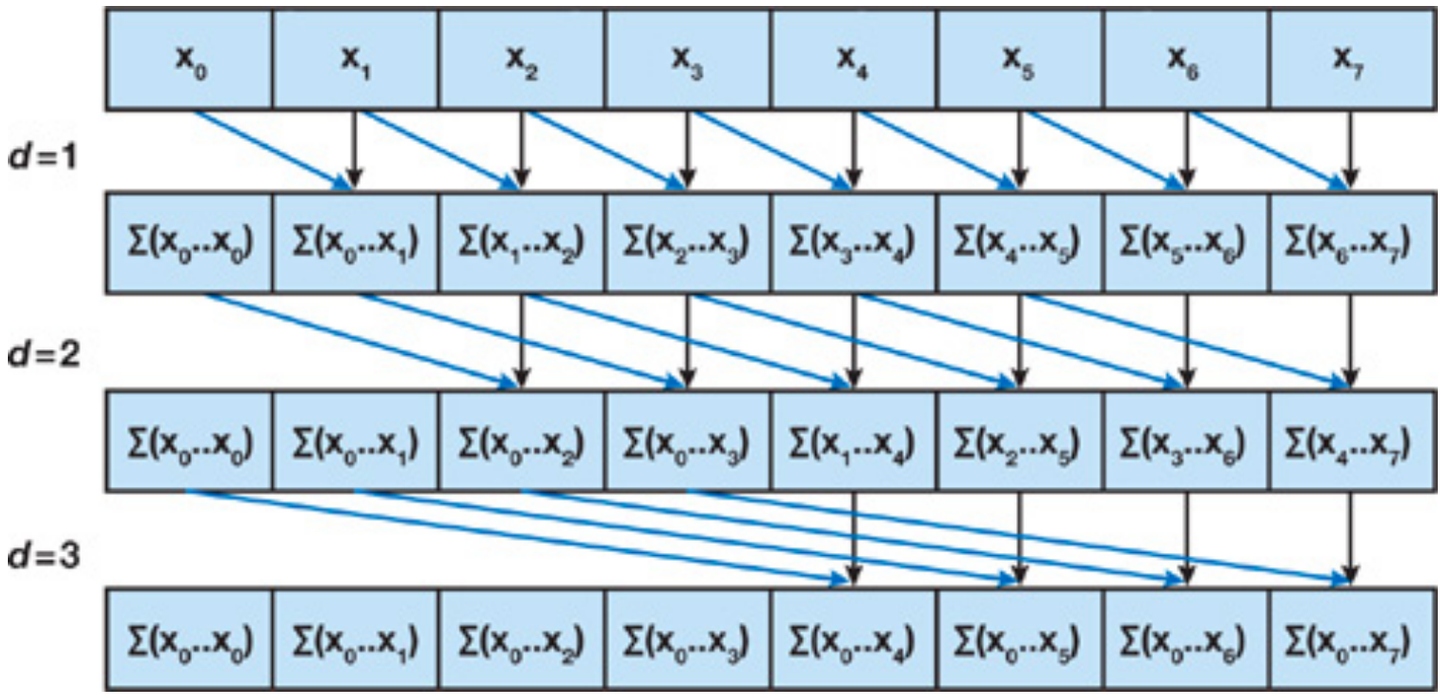
Example: Prefix Sum

- Problem:
  - Given an array of N elements, find its running sum, ex. Exclusive Prefix Sum:
    - 1 2 4 5 6 8 INPUT
    - 0 1 3 7 12 18 OUTPUT

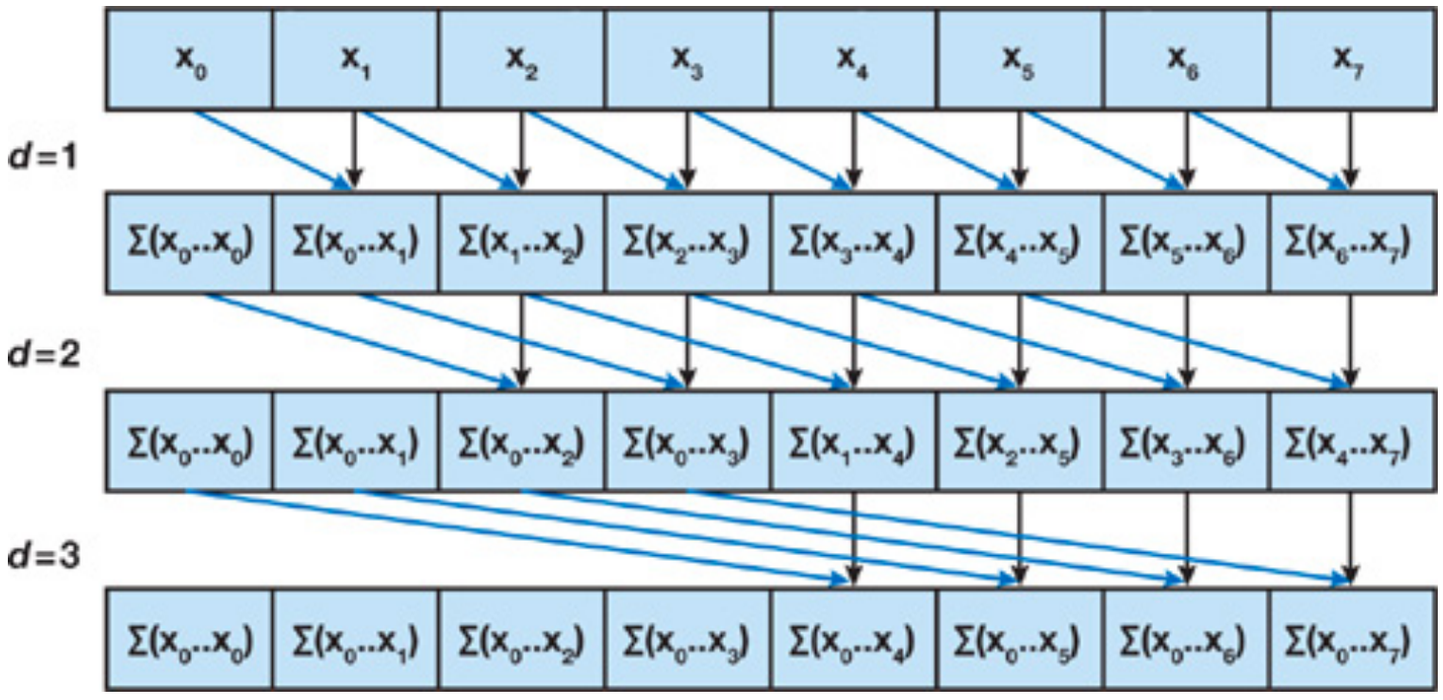
Example: Prefix Sum

- Problem:
  - Given an array of N elements, find its running sum, ex. Exclusive Prefix Sum:
    - 1 2 4 5 6 8 INPUT
    - 0 1 3 7 12 18 OUTPUT
- Trivial for CPU, with total work O(N)

Work Inefficient Parallel Reduction

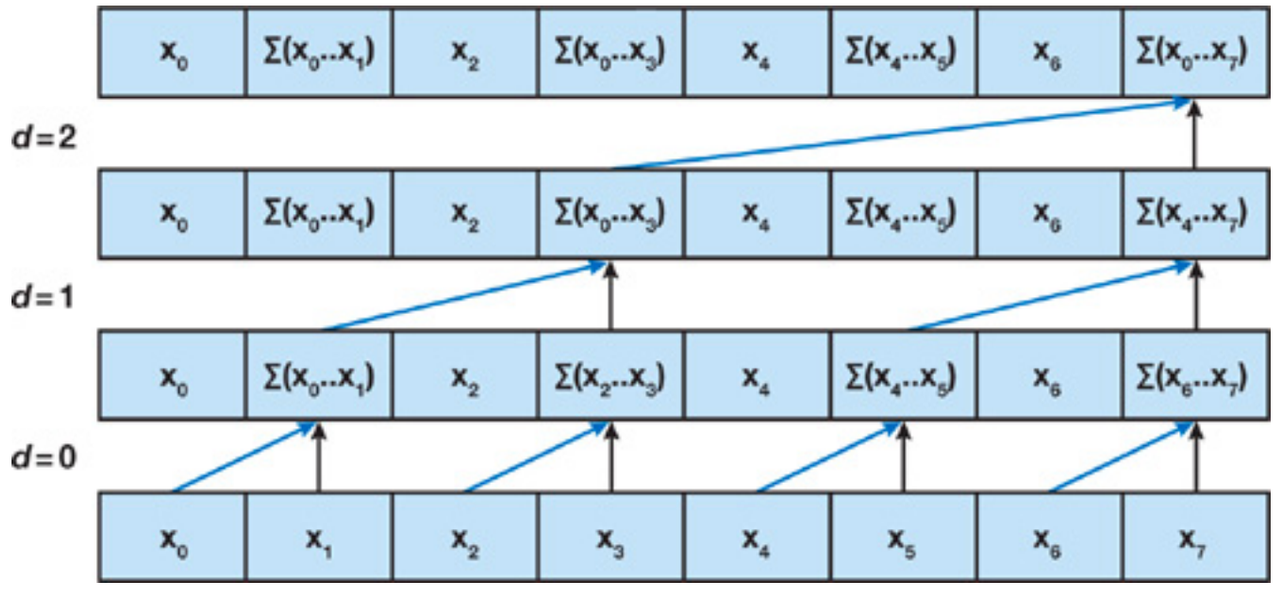


Work Inefficient Parallel Reduction

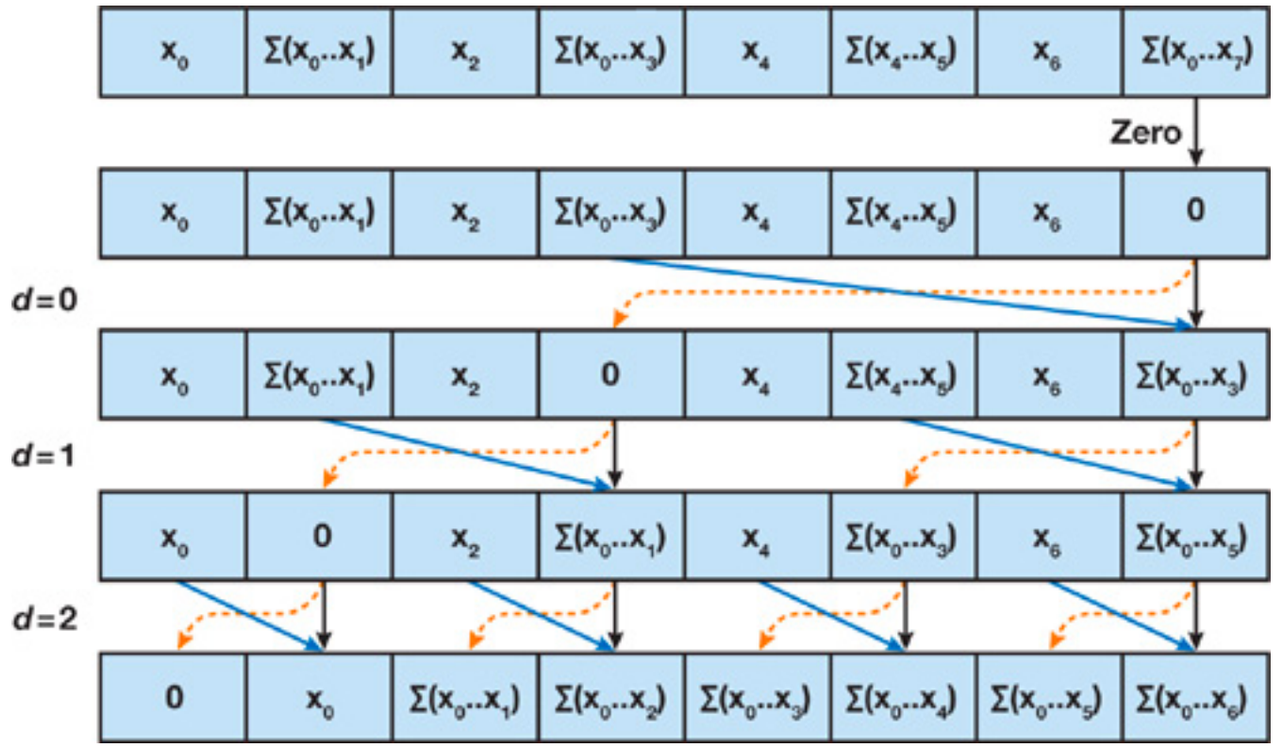


O(N log N)

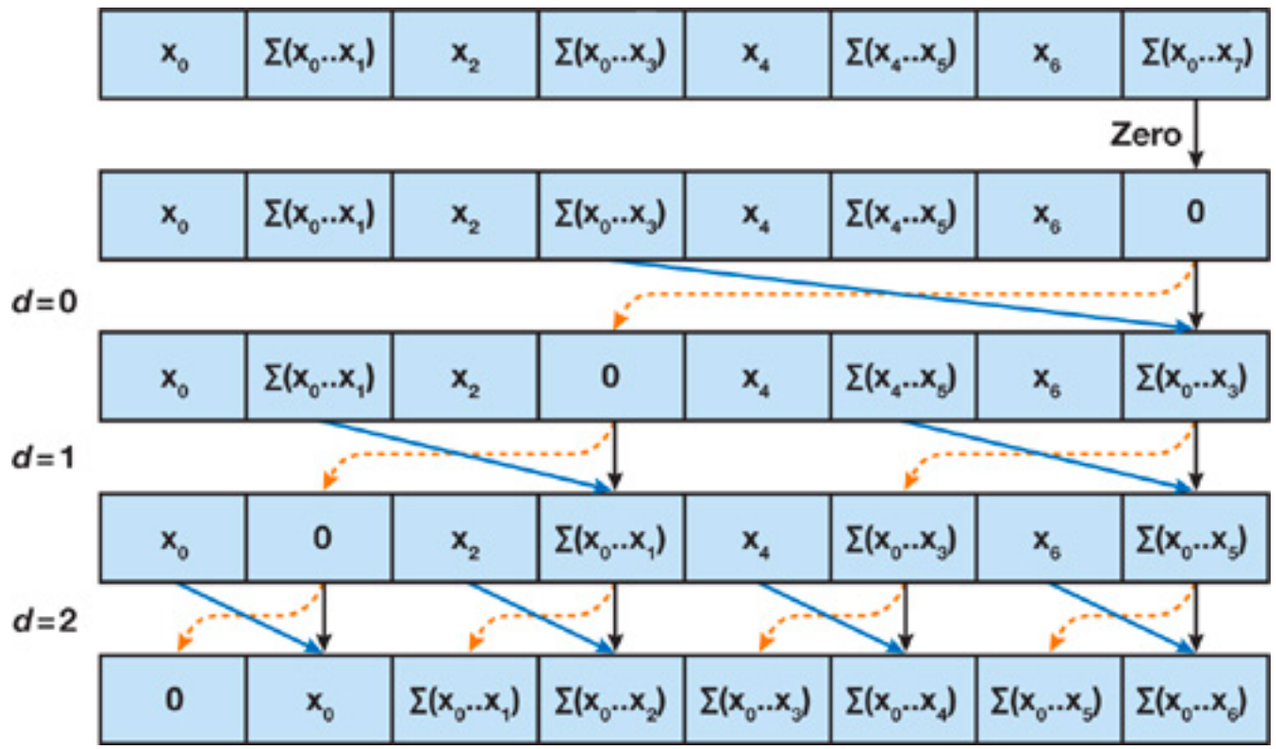
Work Efficient Parallel Reduction



Work Efficient Parallel Reduction



Work Efficient Parallel Reduction



O(N)

In reality

In reality

- In reality, we often end up with:

In reality

- In reality, we often end up with:
  - Work inefficient algorithms that adapt well to memory (naive prefix sum)

In reality

- In reality, we often end up with:
  - Work inefficient algorithms that adapt well to memory (naive prefix sum)
  - Work efficient algorithms that don't adapt well to memory (eg. neighbourlists)

In reality

- In reality, we often end up with:
  - Work inefficient algorithms that adapt well to memory (naive prefix sum)
  - Work efficient algorithms that don't adapt well to memory (eg. neighbourlists)
  - Really lucky if we can find something works well in both cases

Upper Bound: Amdahl's Law

Upper Bound: Amdahl's Law

- P - fraction of program parallelizable

Upper Bound: Amdahl's Law

- P - fraction of program parallelizable
- Maximum Speedup Attainable:

Upper Bound: Amdahl's Law

- P - fraction of program parallelizable
- Maximum Speedup Attainable:
- S(N) = 1/((1-P)+P/N)

Upper Bound: Amdahl's Law

- P - fraction of program parallelizable
- Maximum Speedup Attainable:
  - S(N) = 1/((1-P)+P/N)
- Very unlikely for a large program to be 100% parallelizable

Algorithms that can be effectively parallelized

Algorithms that can be effectively parallelized

- Reduction, Prefix Sums

Algorithms that can be effectively parallelized

- Reduction, Prefix Sums
- Radix-based sorting

Algorithms that can be effectively parallelized

- Reduction, Prefix Sums
- Radix-based sorting
- Breadth First Search

Algorithms that can be effectively parallelized

- Reduction, Prefix Sums
- Radix-based sorting
- Breadth First Search
- Matrix multiplication, Linear Algebra

Algorithms that can be effectively parallelized

- Reduction, Prefix Sums
- Radix-based sorting
- Breadth First Search
- Matrix multiplication, Linear Algebra
- FFTs

Algorithms that can be effectively parallelized

- Reduction, Prefix Sums
- Radix-based sorting
- Breadth First Search
- Matrix multiplication, Linear Algebra
- FFTs
- Simple Dynamic Programming

Algorithms with no good parallel solution

Algorithms with no good parallel solution

- Many graph search algorithms

Algorithms with no good parallel solution

- Many graph search algorithms
- Quick Sort

Algorithms with no good parallel solution

- Many graph search algorithms
- Quick Sort
- Binary Search

Asymmetric Algorithms

Asymmetric Algorithms

- Compression algorithms (eg. given a Huffman code, encoding is parallelizable, decoding not easily parallelizable.)

Asymmetric Algorithms

- Compression algorithms (eg. given a Huffman code, encoding is parallelizable, decoding not easily parallelizable.)
- Converting the representation of a large molecule into internal coordinates (bijection from R^3N into Phi/Psi space)

Language Differences

Language Differences

- CUDA is a language: supports only NVIDIA cards, lots of good documentation
- OpenCL is a standard: supports NVIDIA, ATI, and Intel CPUs and Xeon Phi's