CMPE 200
Computer Architecture & Design

Lecture 5.
Parallel Processing (3)

Haonan Wang

SJSU SAN JOSÉ STATE UNIVERSITY

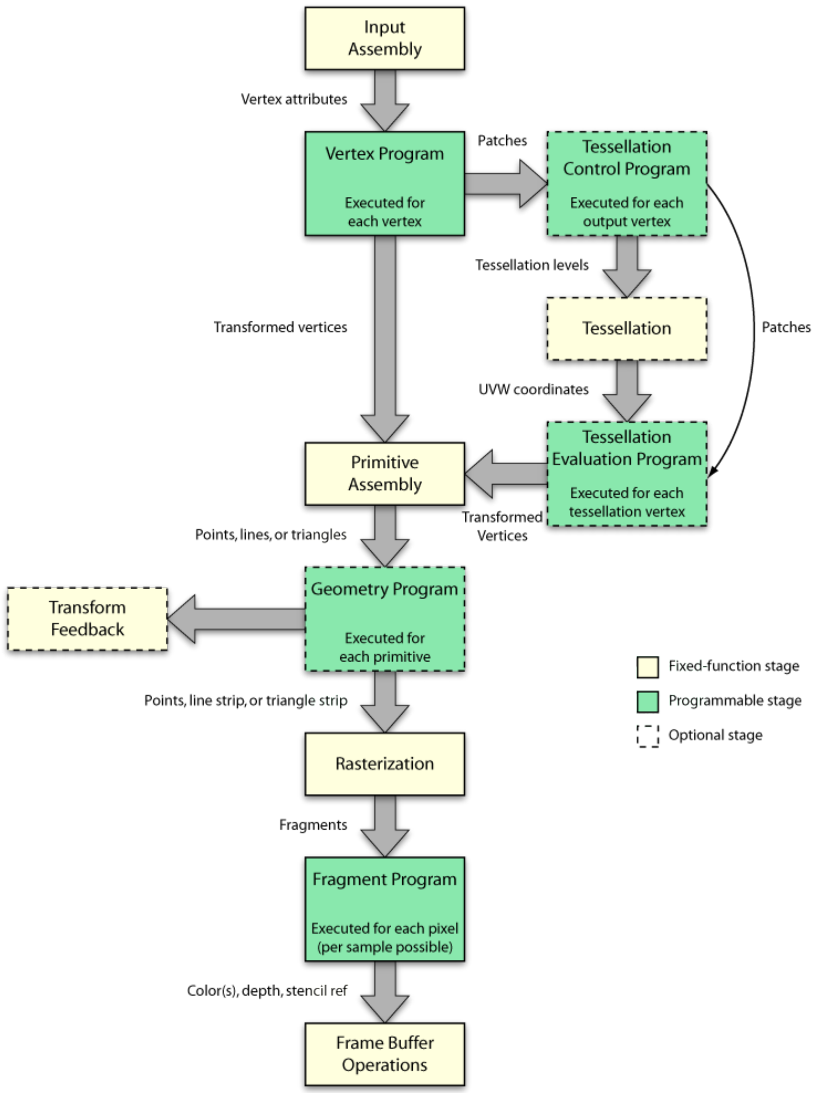
Multi-core CPUs, Multiprocessors, and GPUs

- **Multi-core CPUs, Multiprocessors**
  - Individual cores are fine-tuned for high ILP
  - Good for Task-level parallelism (TLP)
  - However, it is hard to employ hundreds of cores in one system
    - Cache coherence, control, power, cost …

- **For specific tasks, Graphics Processing Unit (GPU) can be an alternative solution**
  - For tasks with high Data-Level parallelism
  - E.g., particle simulation, image/video processing, games, machine learning …
  - The usage of GPU is common now

- **Instruction & Data level parallelism combinations**
  - SISD: Classical Von Neumann machine
  - MISD: NA
  - SIMD: GPU
  - MIMD: multi-core & multiprocessor

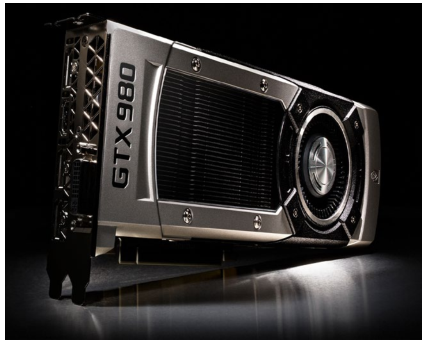
Isn’t GPU used for graphics processing?

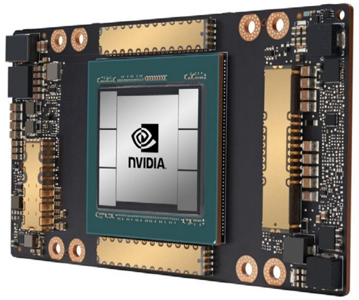


- **Recent GPUs employ regular ALUs for programmable stages, which can be used for general purpose computing not only for the graphics applications**
  - GPGPU
  - Programming language support: CUDA, OpenCL, …
- **GPU is efficient**
  - High performance
    - hundreds of TFLOPS (floating point operations per second)
    - CPU: hundreds of GFLOPS
  - Energy/cost efficient by design
    - Used in Top 500 Supercomputers in the world

https://commons.wikimedia.org/wiki/File:The_OpenGL_-_DirectX_graphics_pipeline.png

GPU Scaling Trends





GTX 480
(Fermi)
448
CUDA
Cores
(139 GB/sec)
2010

GTX 680
(Kepler)
1536
CUDA
Cores
(192 GB/sec)
2012

GTX 980
(Maxwell)
2048
CUDA
Cores
(224 GB/sec)
2014

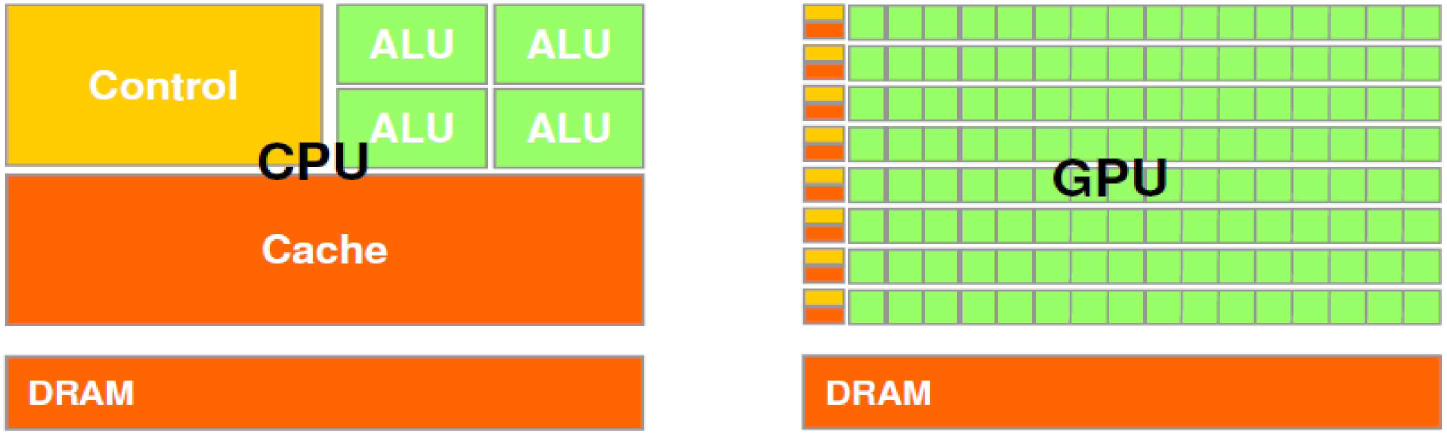
GP 100
(Pascal)
3584
CUDA
Cores
(720 GB/sec)
2016

GV 100
(Volta)
5120
CUDA
Cores
(900 GB/sec)
2017

GA 100
(Ampere)
8192
CUDA
Cores
(1555 GB/sec)
2020

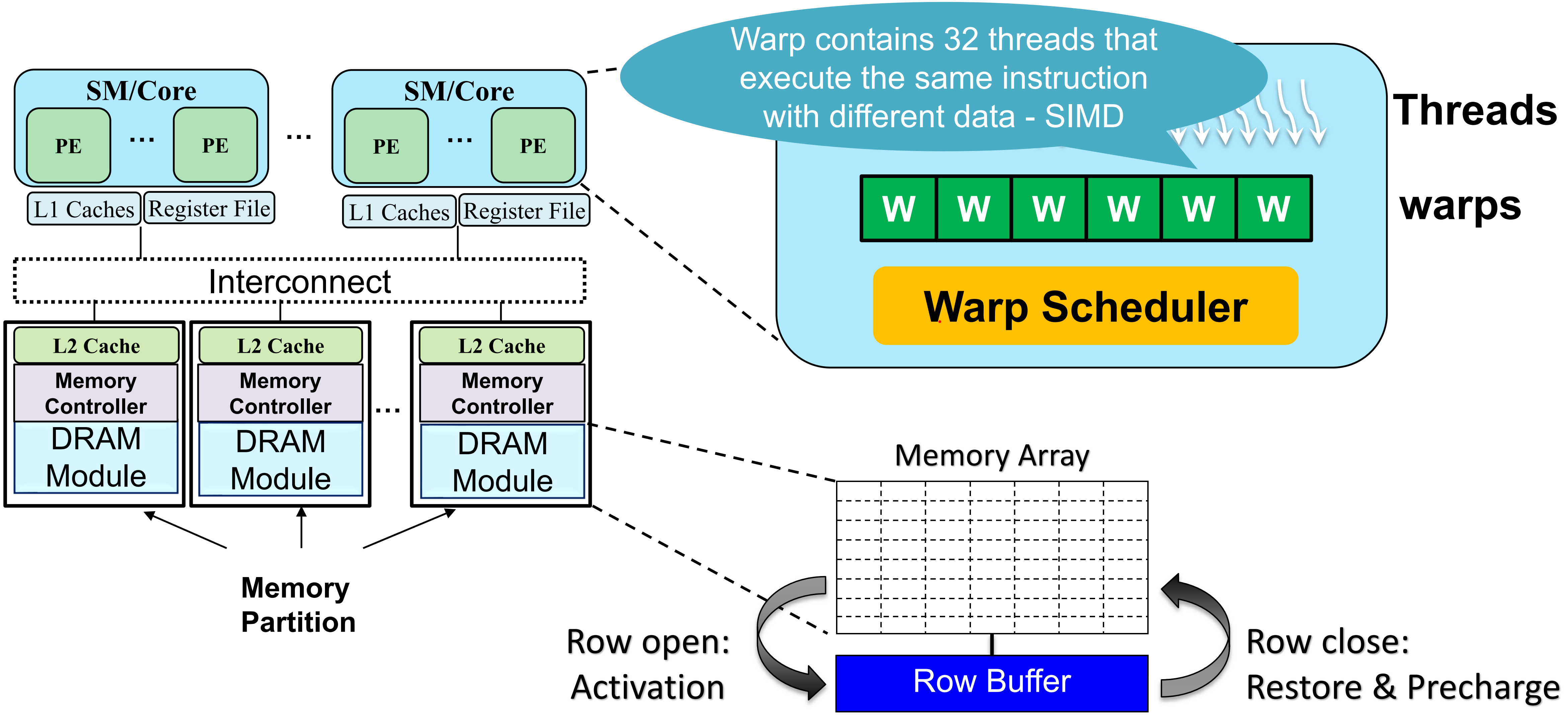
Why is GPU so Efficient?

- **GPU is specialized for compute-intensive, highly data parallel computation (owing to its graphics rendering origin)**
  - More transistors can be devoted to data processing rather than caching and control
  - At peak performance GPU uses order of magnitude less energy per operation than CPU
  - Working with suitable applications: high arithmetic intensity (the ratio between arithmetic operations and memory operations), high DLP, not too sensitive to latency



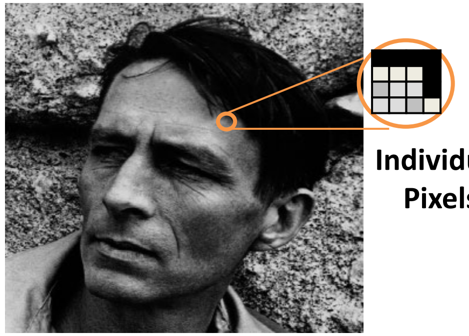
Credit: Dan Negrut, ME964 UW-Madison

Classical GPGPU Architecture (Nvidia)



State-Of-The-Art GPU (Ampere)

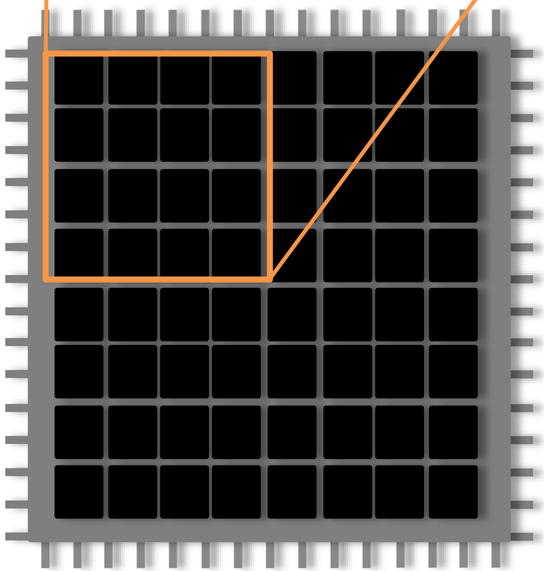
Working Philosophy (1): SIMD



Individual Pixels

(0 = black, 255 = white)

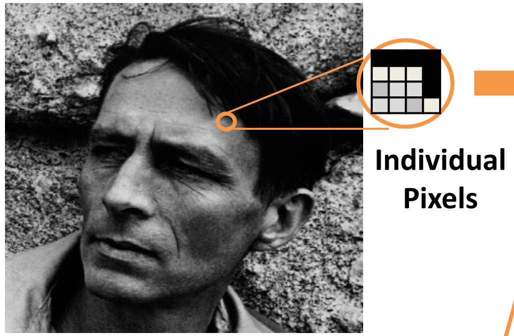
0	0	0	0
64	64	64	0
128	192	192	0
192	192	128	64



GPU Shader

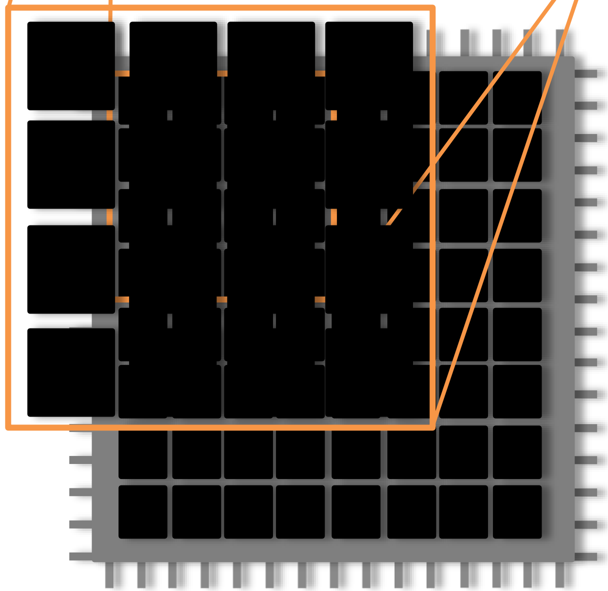
Image taken from the photo "Robin Jeffers at Ton House" (1927) by Edward Weston

Working Philosophy (1): SIMD



(0 = black, 255 = white)

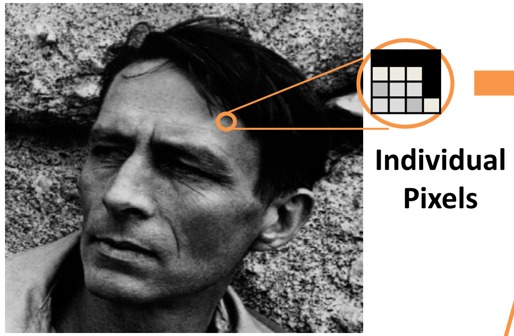
0	0	0	0
64	64	64	0
128	192	192	0
192	192	128	64



GPU Shader

Image taken from the photo "Robin Jeffers at Ton House" (1927) by Edward Weston

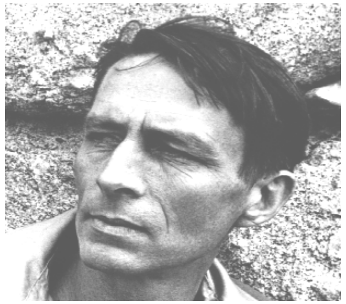
Working Philosophy (1): SIMD



(0 = black, 255 = white)

20	20	20	20
84	84	84	20
148	212	212	20
212	212	148	84

+ Brightness



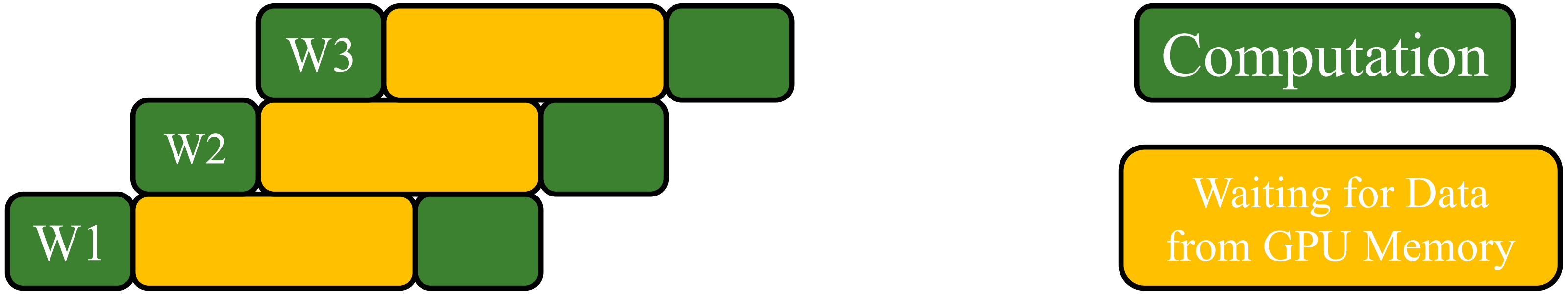
20	20	20	20
84	84	84	20
148	212	212	20
212	212	148	84

+ 20 (brightness factor)

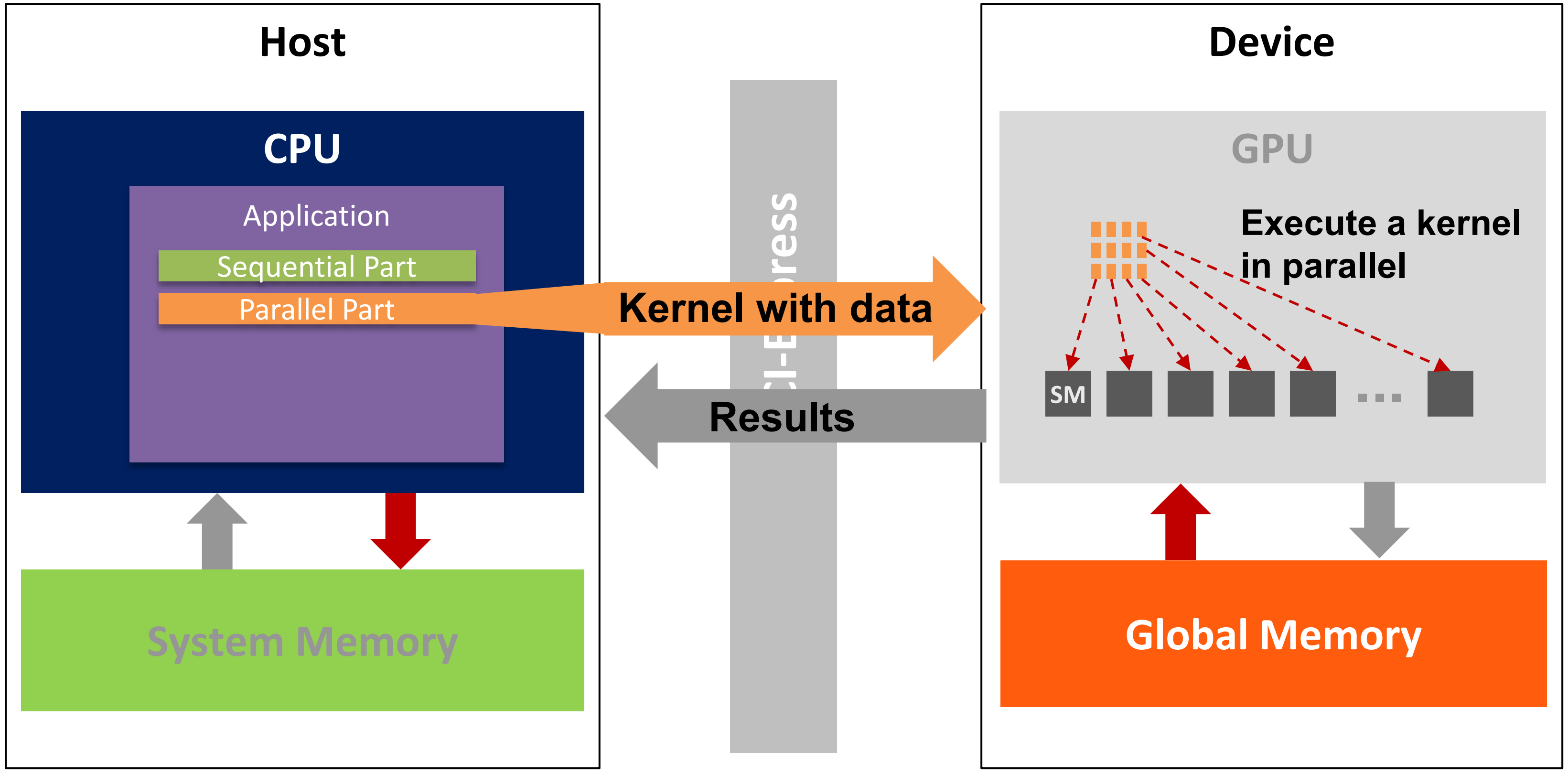
Image taken from the photo "Robin Jeffers at Ton House" (1927) by Edward Weston

Working Philosophy (2): Hide Latency

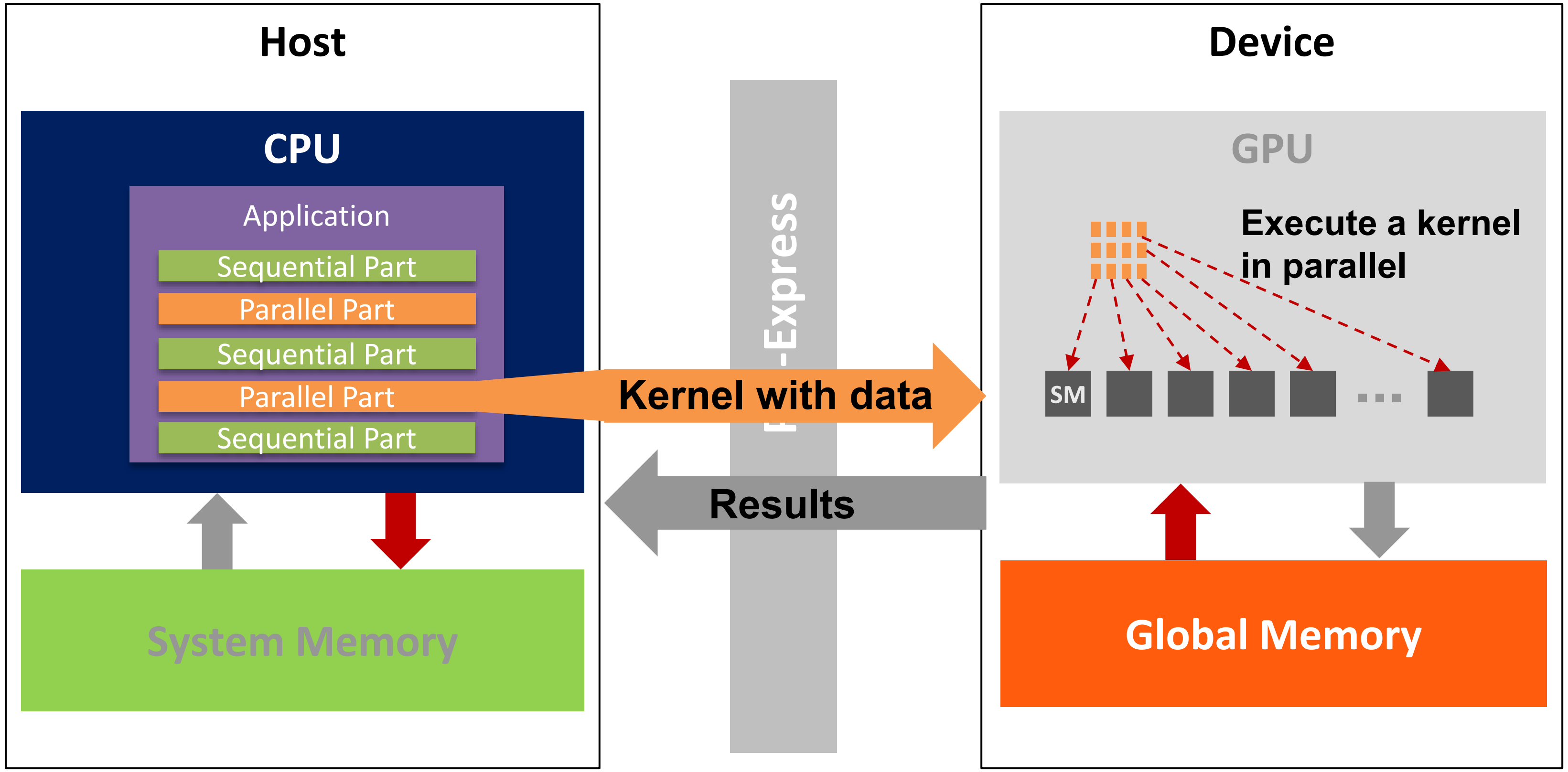
GPU memory hierarchy provide high BW, but usually high latency
→ Hide long memory latency with computation from other thread warps



GPU Program Execution Flow



GPU Program Execution Flow



SJSU SAN JOSÉ STATE UNIVERSITY

CUDA Code Example: Vector Addition

- C code

```
int main(){    // executed on CPU
    ...
    for (int i=0; i<32; i++)
        C[i] = A[i] + B[i];
}
```

- CUDA C code

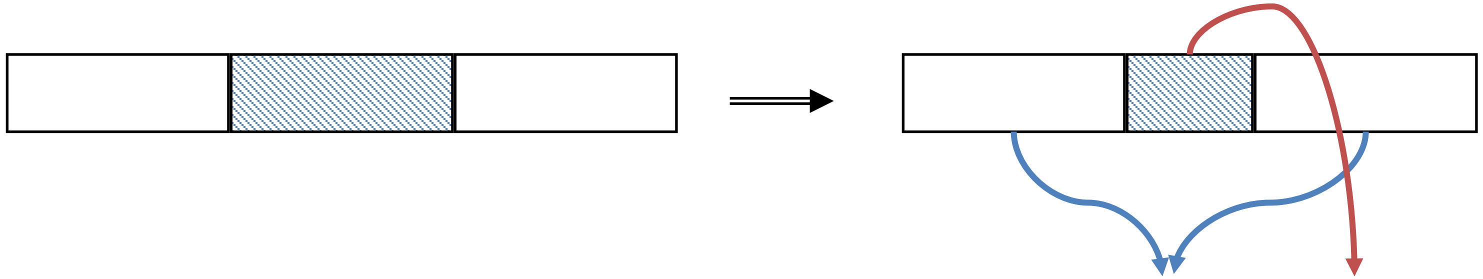
```
__global__ void vecAddCUDA (float* A, float* B, float* C){   // executed on the GPU
    ...
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}

int main(){    // executed on CPU
    ...
   int numBlocks = 1;
   int numThreads = 32;
   vecAddCUDA <<<numBlock,numThreads>>>(A,B,C);
}
```

Multicore Performance: Amdahl's Law

- **Speedup due to enhancement E:**

$$\text{Speedup w/ E} = \frac{\text{Exec time w/o E}}{\text{Exec time w/ E}}$$

- **Suppose that enhancement E accelerates a fraction F (F <1) of the task by a factor S (S>1) and the remainder of the task is unaffected:**



$$\text{ExTime w/ E} = \text{ExTime w/o E} \times ((1\text{-}F) + F/S)$$

$$\text{Speedup w/ E} = 1 / ((1\text{-}F) + F/S)$$

Example 1: Amdahl's Law

Speedup w/ E = 1 / ((1-F) + F/S)

- **Consider an enhancement that runs 20 times faster but is only usable 25% of the time**

  Speedup w/ E = 1/(.75 + .25/20) = 1.31

- **What if its usable only 15% of the time?**

  Speedup w/ E = 1/(.85 + .15/20) = 1.17

- **Amdahl's Law tells us that to achieve linear speedup with 100 cores, none of the original computation can be scalar!**

- **To get a speedup of 90 from 100 cores, the percentage of the original program that could be scalar would have to be 0.1% or less**

  Speedup w/ E = 1/(.001 + .999/100) = 90.99

Example 2: Amdahl's Law

Speedup w/ E = 1 / ((1-F) + F/S)

- **Consider summing 10 scalar variables and two 10 by 10 matrices on 10 cores**

  Speedup w/ E = 1/(.091 + .909/10) = 1/0.1819 = 5.5

- **What if there are 100 cores?**

  Speedup w/ E = 1/(.091 + .909/100) = 1/0.10009 = 10.0

- **What if the matrices are 100 by 100 (or 10,010 adds in total) on 10 cores?**

  Speedup w/ E = 1/(.001 + .999/10) = 1/0.1009 = 9.9

- **What if there are 100 cores?**

  Speedup w/ E = 1/(.001 + .999/100) = 1/0.01099 = 91

Example 2: Amdahl's Law

Speedup w/ E = 1 / ((1-F) + F/S)

- **Consider summing 10 scalar variables and two 10 by 10 matrices on 10 cores**

  Speedup w/ E = 1/(.091 + .909/10) = 1/0.1819 = 5.5

- **What if there are 100 cores?**

  Speedup w/ E = 1/(.091 + .909/100) = 1/0.10009 = 10.0

- **What if the matrices are 100 by 100 (or 10,010 adds in total) on 10 cores?**

  Speedup w/ E = 1/(.001 + .999/10) = 1/0.1009 = 9.9

- **What if there are 100 cores?**

  Speedup w/ E = 1/(.001 + .999/100) = 1/0.01099 = 91

Conclusion Time

What are the ways to do synchronization for multicore?

Barrier sync vs. MPP primitives

What are the key ideas of GPU?

SIMD & hiding latency

SAN JOSÉ STATE UNIVERSITY *powering* SILICON VALLEY