MORE HARDWARE DESIGNS ON PARALLEL PROCESSING

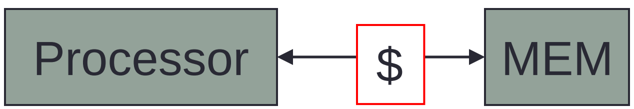
Lecturer: Hui Annie Guo
h.guo@unsw.edu.au
K17-501F

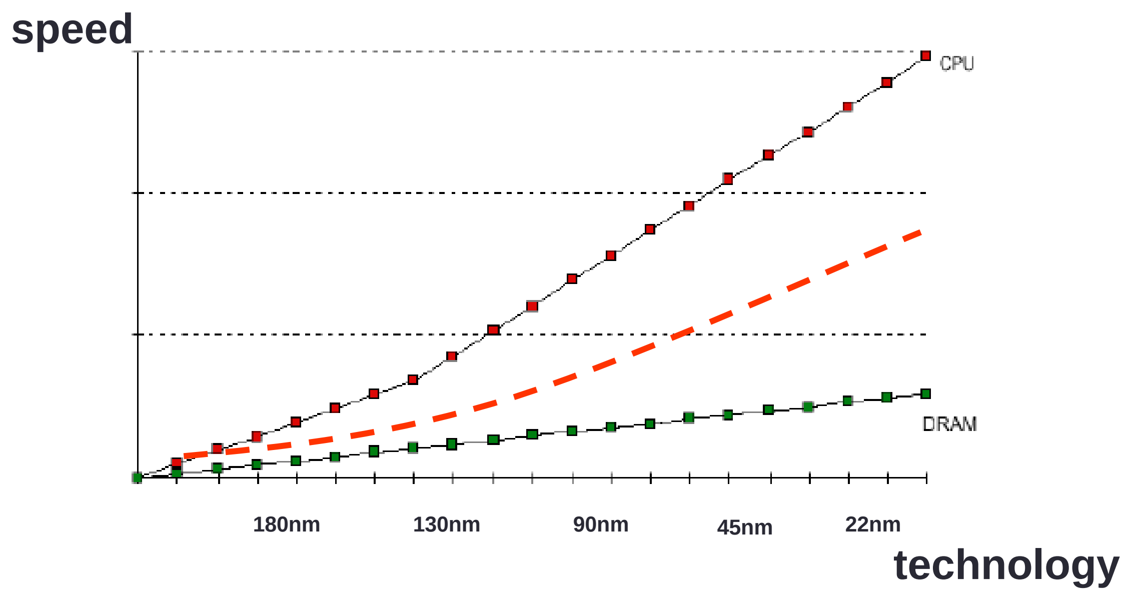
Lecture overview

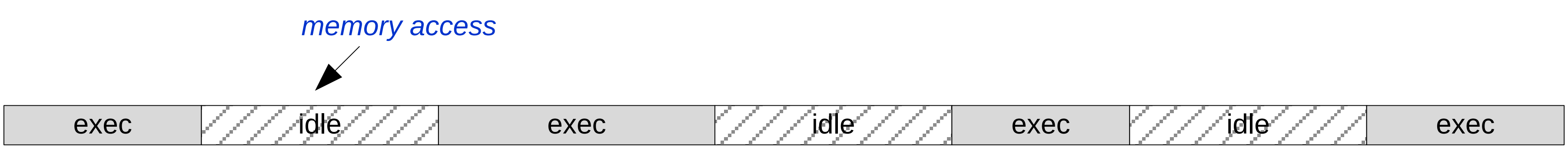
- **Topics**
  - **Multithreaded processor**
  - **GPU**

- **Suggested reading**
  - **H&P Chapter 4.10, 6.4**
  - **https://en.wikipedia.org/wiki/Computer_graphics**
  - **https://en.wikipedia.org/wiki/Graphics_pipeline**

Speed gap affects the processor performance !

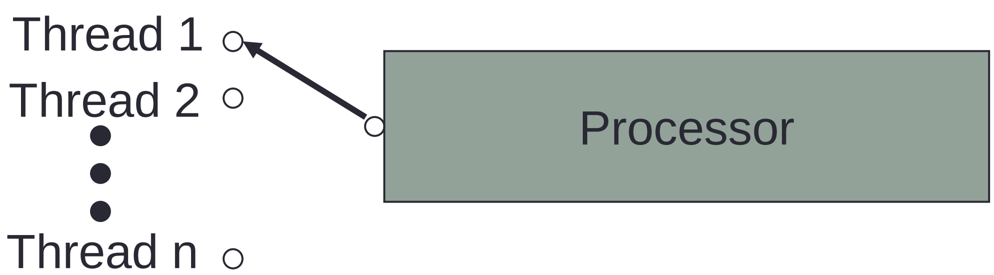


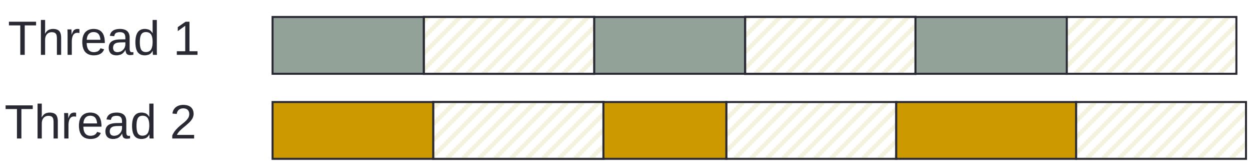




Multi-threaded execution

- **When one thread is not available due to an operation delay (e.g. long memory access), the processor can switch to other thread**





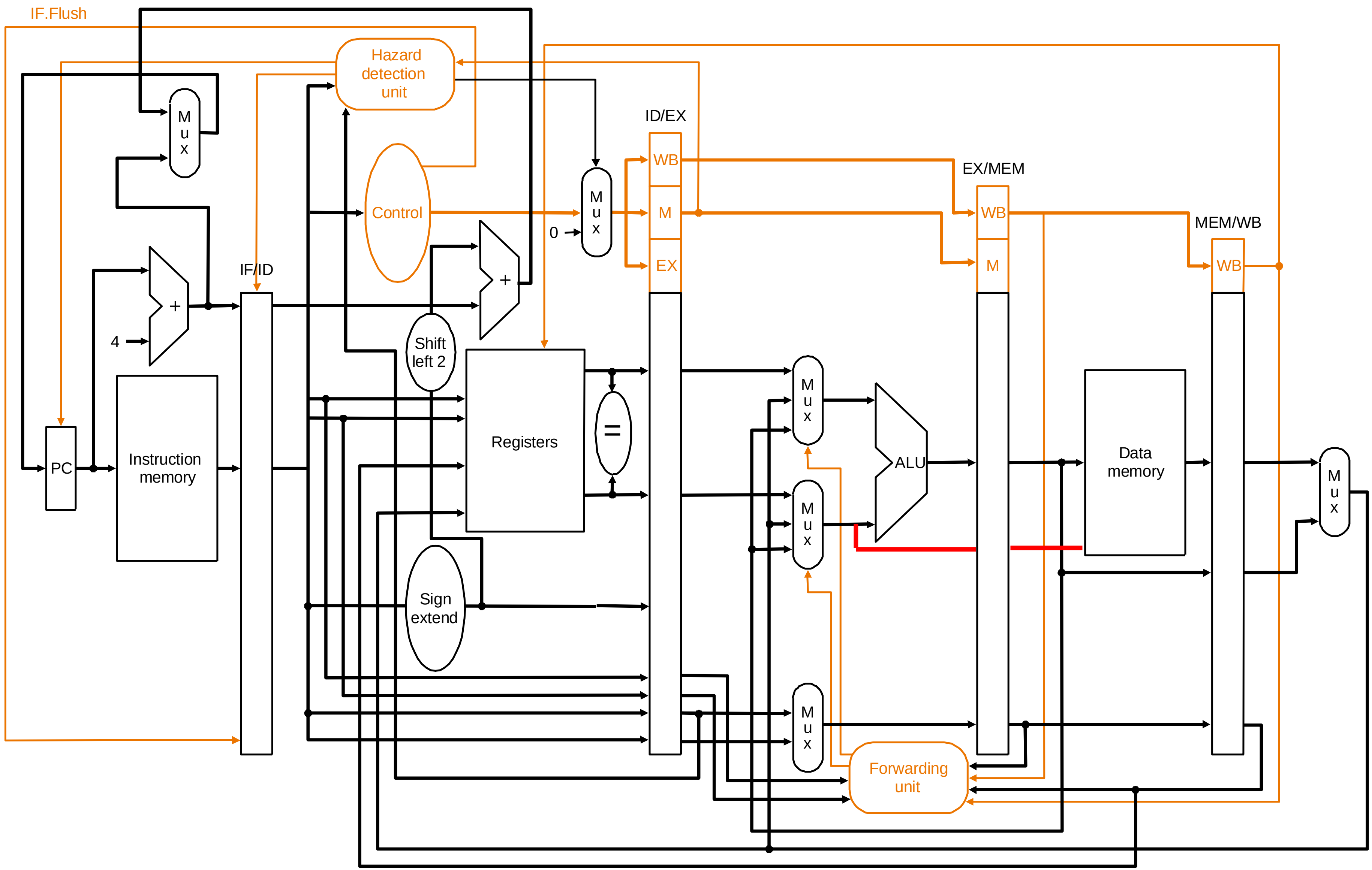
Multithreaded processor

- Hardware level multi-threading
  - Fast switching between threads
  - Replicated registers, PC, etc.
- Two design approaches
  - Fine-grained multithreading
    - Switch threads after each cycle
    - If one thread stalls, another is executed
  - Coarse-grained multithreading
    - Only switch threads on long stall (e.g., L2-cache miss)

Example

- **Fine grained multithreading**
  - **The base pipeline is given in the next slide**

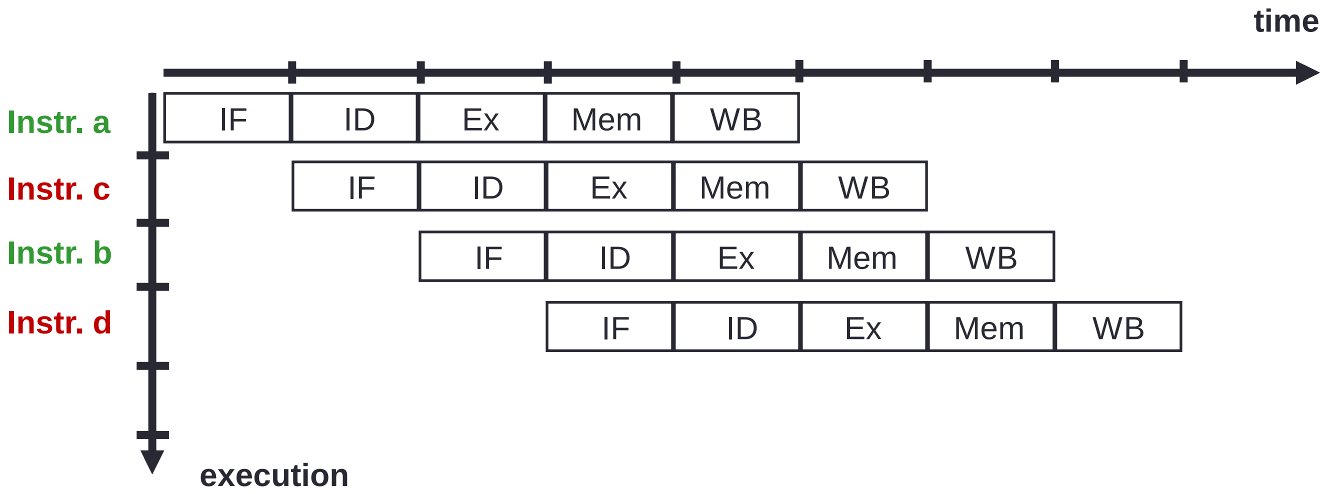
T1:	a: *lw $5, 0($6)*	T2:	c: *sub $10, $11, $12*
	b: add $7, $5, $9		d: *ori $5, $10, $11*


IF.Flush
Hazard detection unit
M u x
Control
ID/EX
WB
M
EX
EX/MEM
WB
M
MEM/WB
WB
0
M u x
+
4
+
IF/ID
Shift left 2
PC
Instruction memory
Registers
=
M u x
ALU
M u x
Data memory
M u x
Sign extend
M u x
Forwarding unit

Example

- **Fine grained multithreading**
  - **The base pipeline is given in the next slide**

T1:	a: *lw $5, 0($6)*	T2:	c. *sub $10, $11, $12*
	b: *add $7, $5, $9*		d. *ori $5, $10, $11*

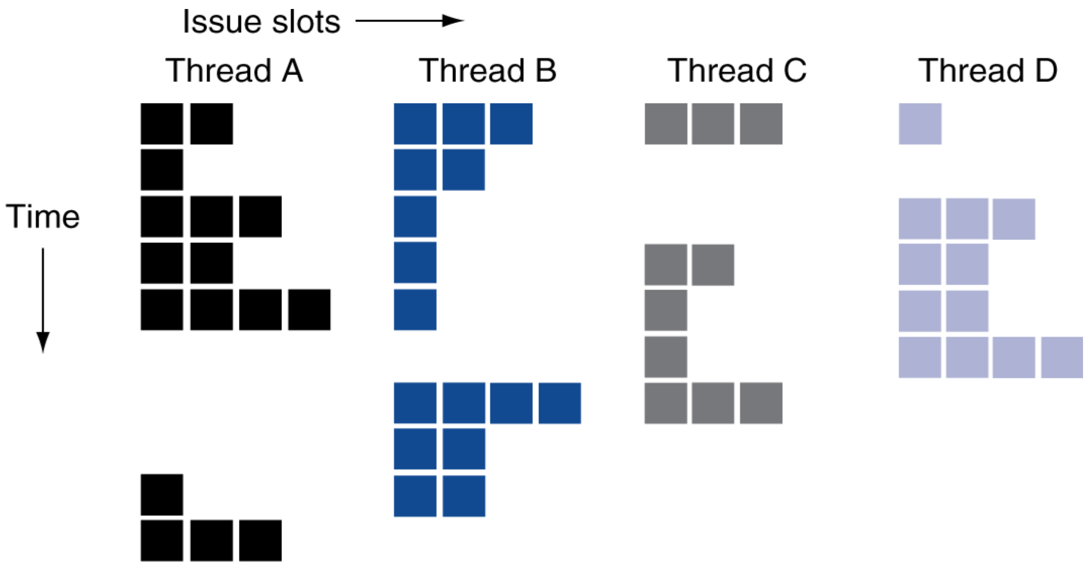


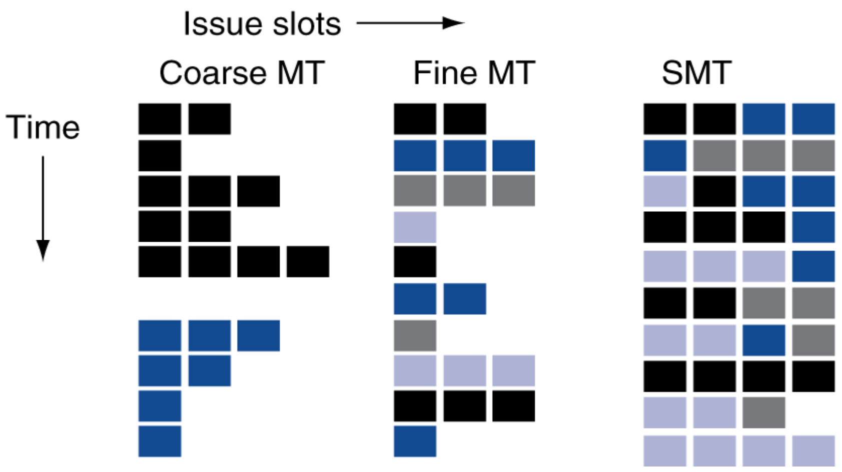
Multithreaded processor (cont.)

- Simultaneous multi-threading (SMT)
  - A variation of HW multi-threading that uses the resources of superscalar architecture
  - Exploiting both instruction level parallelism and thread-level parallelism

Coarse MT vs. Fine MT vs. SMT

four parallel general-purpose processing units





Remarks

- Superscalar and multi-threaded processor are the processor level design for parallel processing
  - Dynamic scheduling is a very efficient approach to exploit instruction parallelism
    - The hardware rearranges instruction execution to reduce the pipeline stalls
    - Can handle cases when dependencies are unknown at compiler time
    - Allows code that was compiled with one processing unit in mind to run efficiently on multiple parallel processing unit.
  - Multi-threaded execution exploits the thread-level parallelism and achieves performance through high resource utilization
- But both designs have a limited scalability

GPU - background

- Initially developed for computer graphics
- Computer graphics
  - A study area for digitally synthesizing and manipulating visual contents.
- Visual contents
  - A scene of objects
  - Motion of objects
  - For example
    - Shape
    - Surface color
    - Surface reflectance
    - Surface texture

How visual contents are processed?

- **Typical processing tasks**
  - **HSR: hidden-surface removal**
  - **Shading: making a "flat" look more like 3D**
  - **Texture mapping: providing high frequency details, surface texture, or color information.**

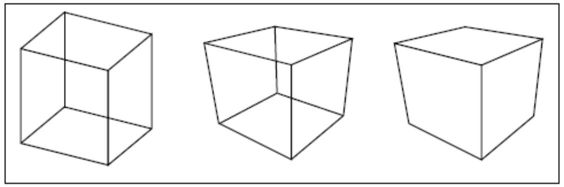
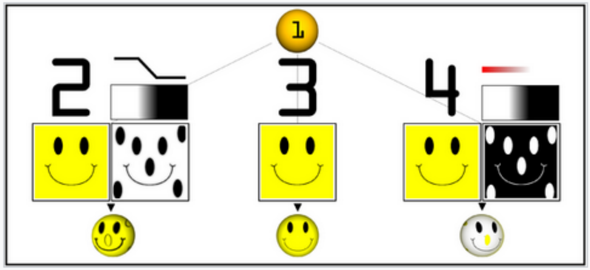


Figure 7.1. Left: wireframe cube in orthographic projection. Middle: wireframe cube in perspective projection. Right: perspective projection with hidden lines removed.

Rendered image of a box. This image has no shading on its faces, but uses edge lines to separate the faces.

This is the same image with the edge lines removed.

This is the same image rendered with shading of the faces to alter the colors of the 3 faces based on their angle to the light sources.



Examples of multitexturing (click for larger image);
1: Untextured sphere, 2: Texture and bump maps,
3: Texture map only, 4: Opacity and texture maps.

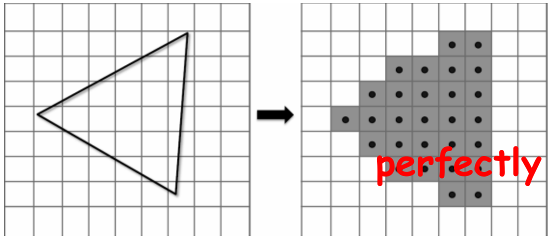
How visual contents are processed? (cont.)

- **All tasks involve huge computations**
  - **Many are of high parallelism**
    - **Embarrassingly parallel**
  - **Demanding computing system with massive parallel processing capability → GPU**

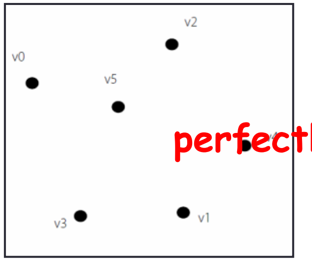
The following slides are based on
"How a GPU Works" by Kayvon Fatahalian

Graphics pipeline

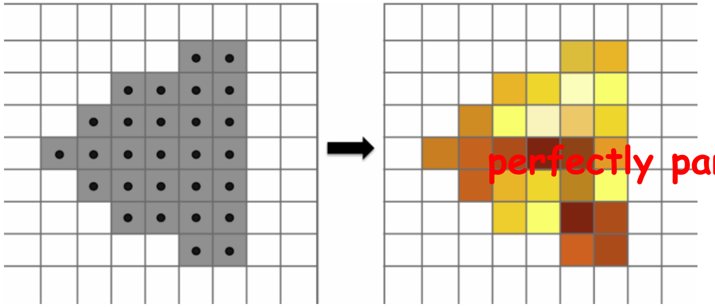
Calculations at each stage are independent and can be performed in parallel



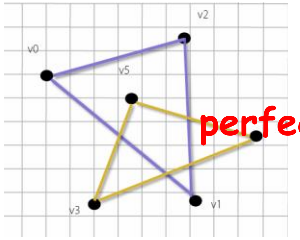
perfectly parallel



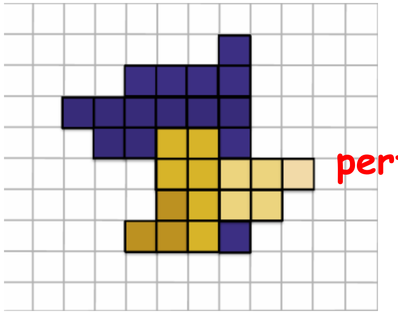
perfectly parallel



perfectly parallel



perfectly parallel

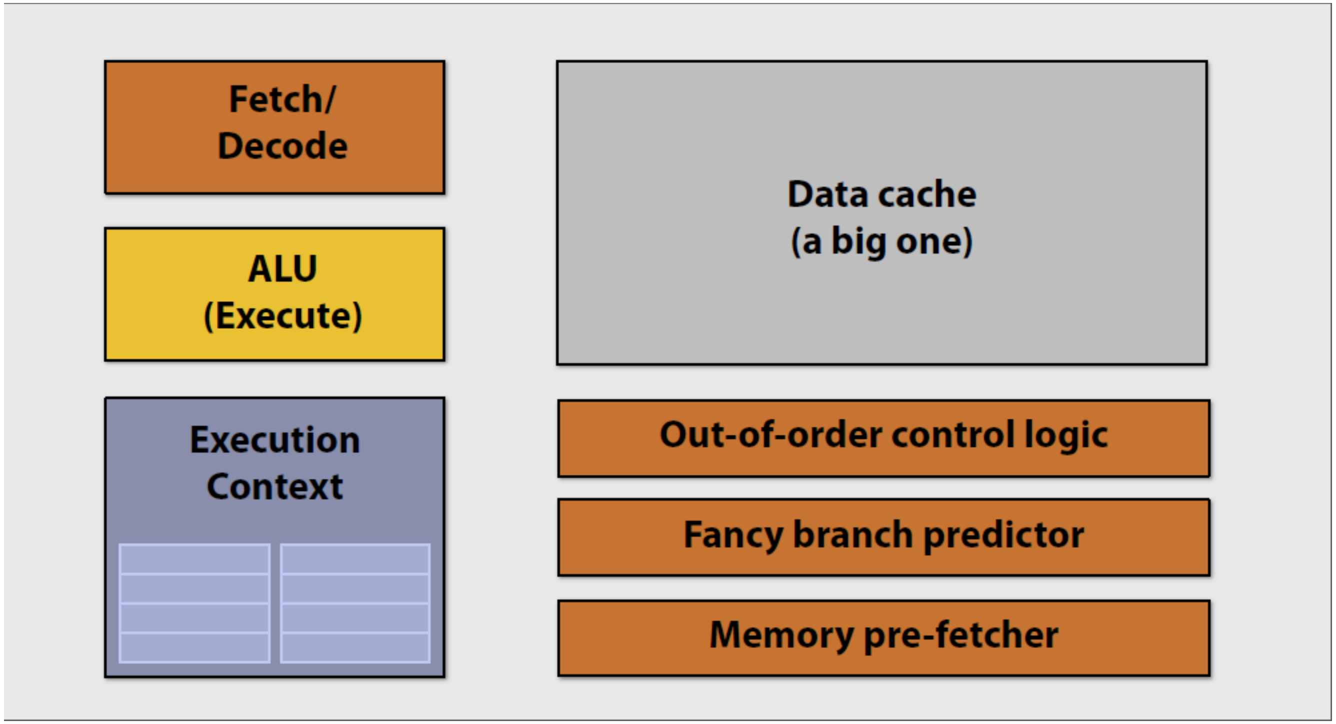


perfectly parallel

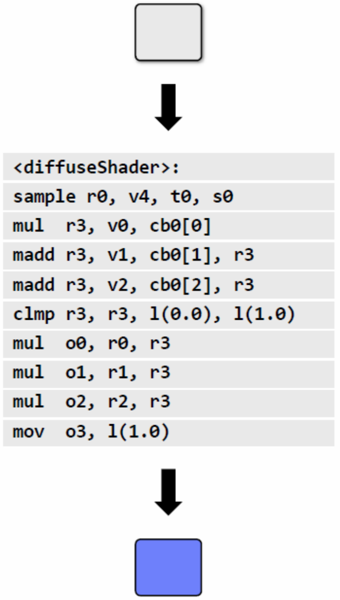
“How a GPU Works” by Kayvon Fatahalian

Hardware design for parallel processing

- **With a powerful but expensive processor**
  - **Not scalable**

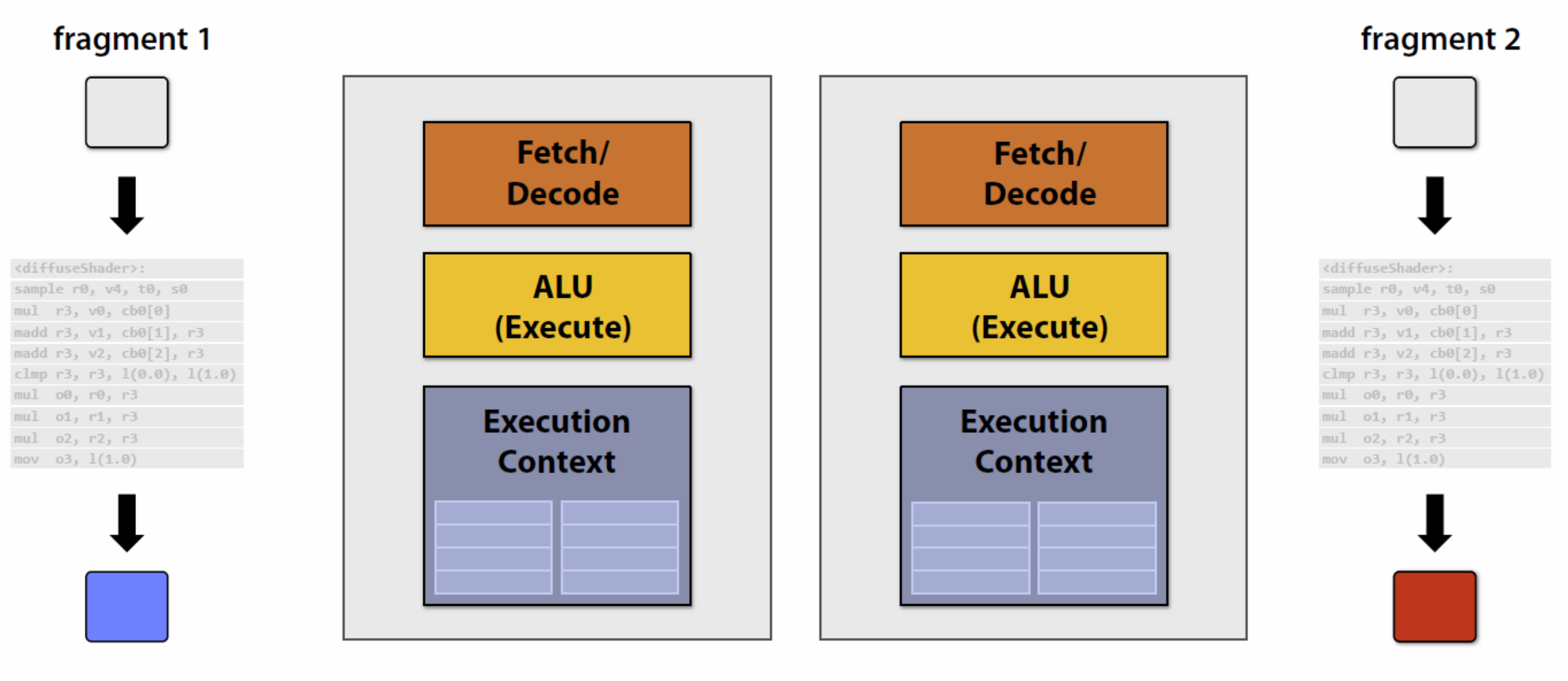


for one fragment



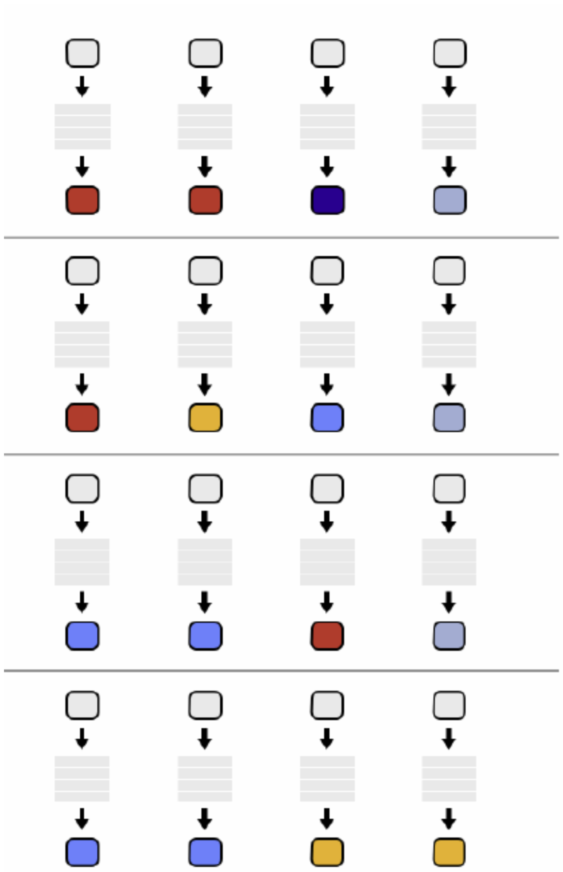
Hardware design for parallel processing

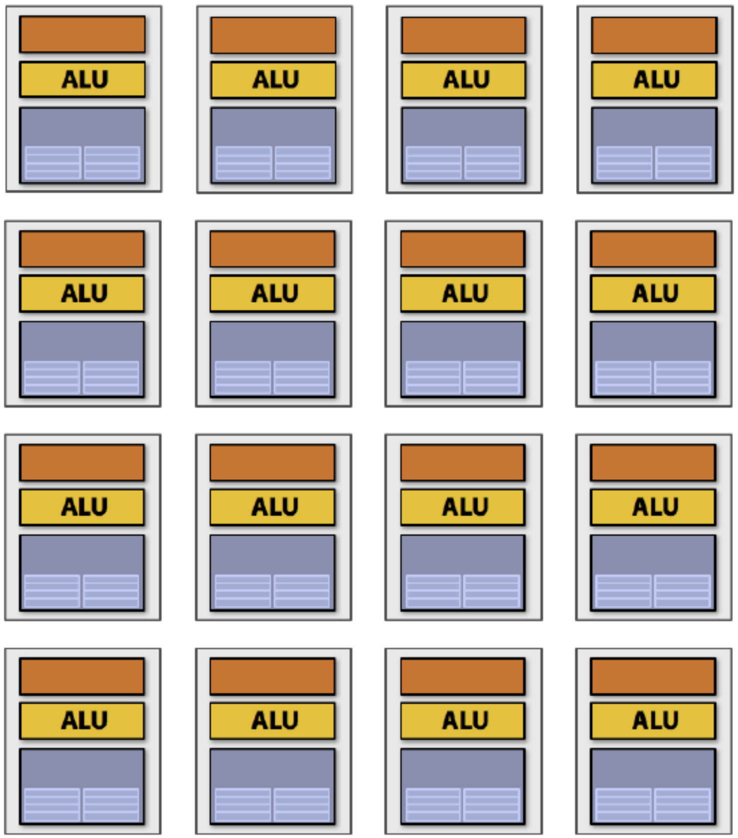
- **With replicated cheap processors**



Hardware design for parallel processing

- **Parallel processing with many cheap processors**



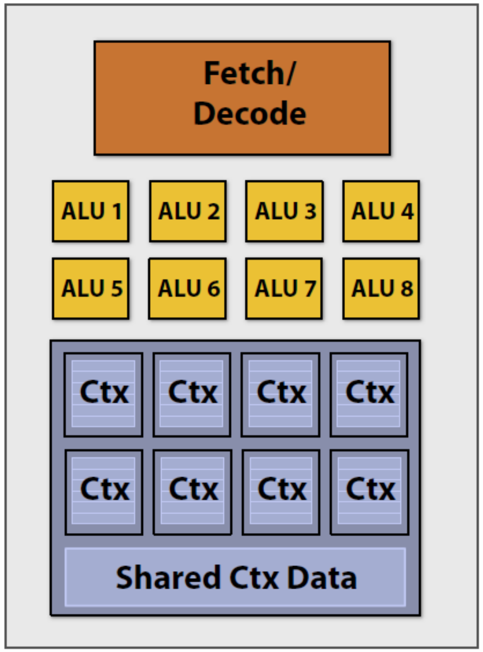


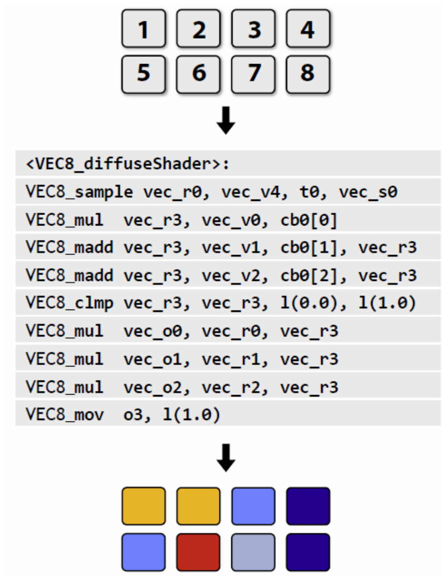
Hardware design for parallel processing

- **Multiple processors perform the same instruction stream on different data/fragments**
  - **SIMD is more efficient**

SIMD

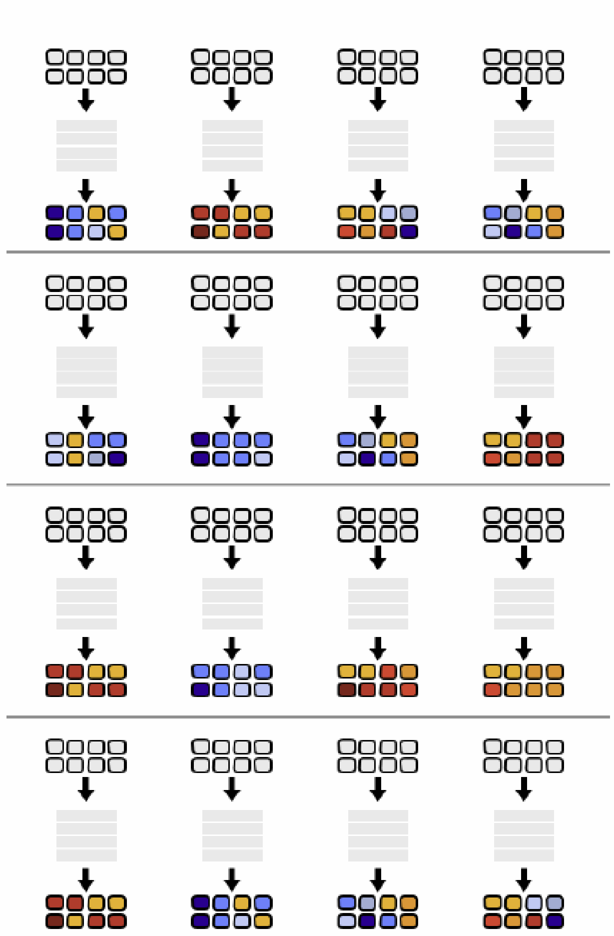
- **The same instruction stream is performed on different inputs by different processing units**
- **Each execution unit has its own local memory**
- **All execution units share a large memory**

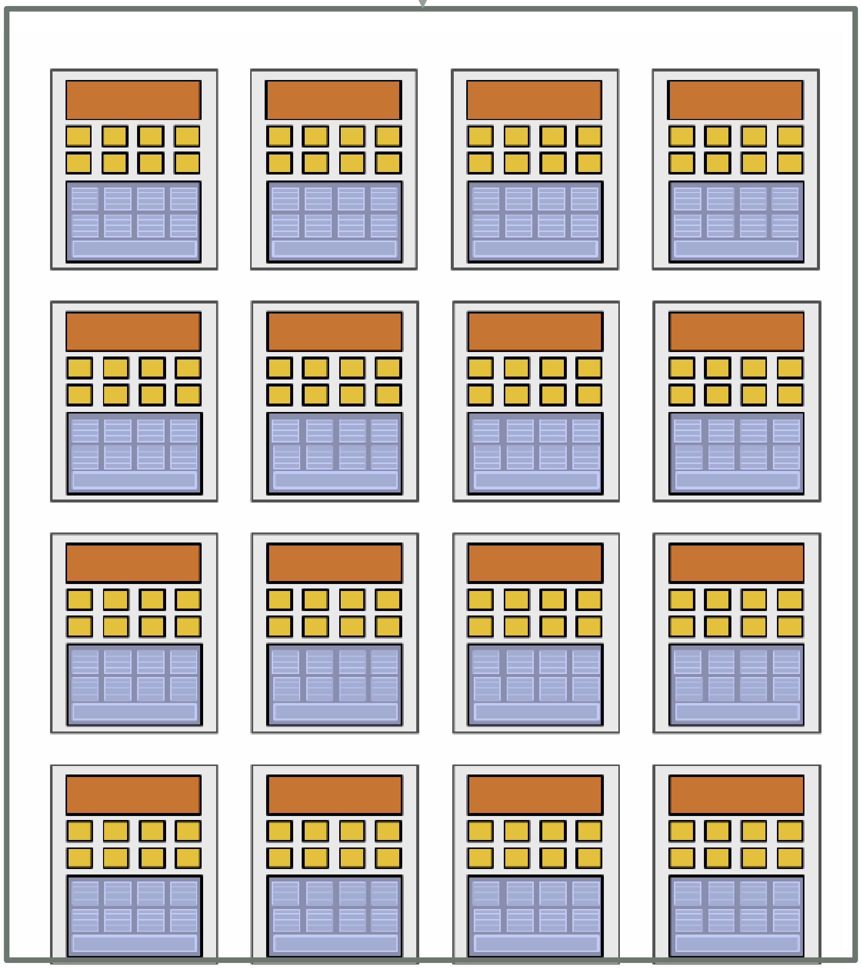




GPU – an example

thread pool
e.g.64 threads

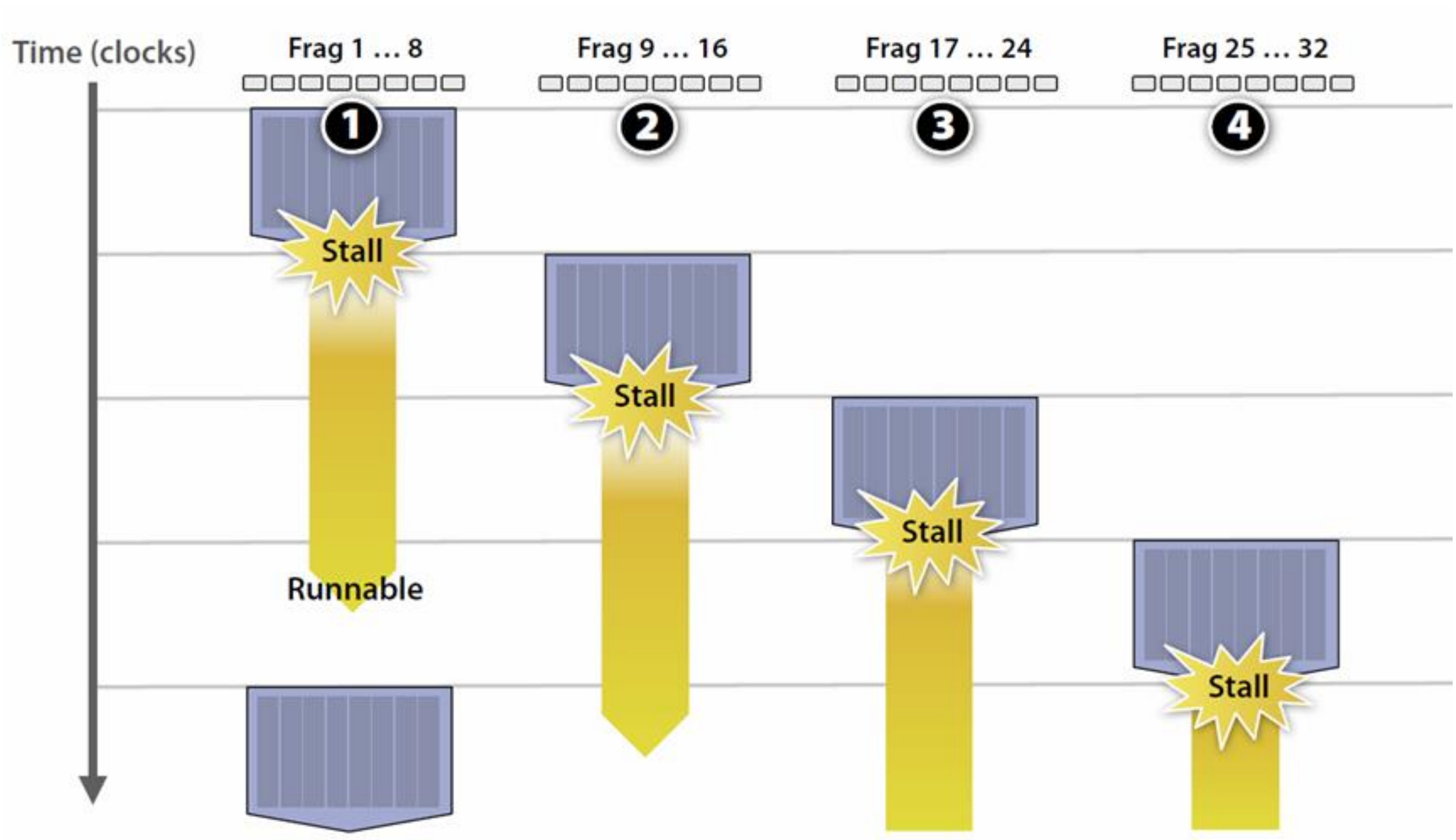




16 cores = 128 ALUs , 16 simultaneous instruction streams

Hiding thread stall

- **Multi-threaded execution is applied**



Remarks

- Three key ideas used in the GPU design
  - Use many “cheap cores” and run them in parallel
  - Pack cores full of ALUs by sharing instruction stream across groups of data sets. For example
    - using SIMD vector instructions
  - Avoid long stalls by interleaving execution of many threads

GPU application

- **Given the hardware invested to do graphics well, how can we supplement it to improve performance of a wider range of applications?**

- **An example solution:**
  - **CUDA**
    - **With a heterogeneous execution model**
      - CPU is the *host*, GPU is the *device*
    - **Use a C-like programming language for GPU**
    - **Unify all forms of GPU parallelism as *CUDA thread***

Example of execution hierarchy

- **Application program calls parallel kernels**
- **Each kernel executes in parallel a set of parallel threads. Each thread has a private local memory**
- **Threads are grouped into blocks. Each thread block has a shared memory**
- **Thread blocks are further packed into grids.**
- **An application can have threads spanned to different grids, but have a shared global memory**
- **GPU hardware handles thread scheduling**