Performance of Modern Computer Systems

CS2053 Computer Architecture

Computer Science & Engineering

University of Moratuwa

Sulochana Sooriyaarachchi

Chathuranga Hettiarachchi

Outline

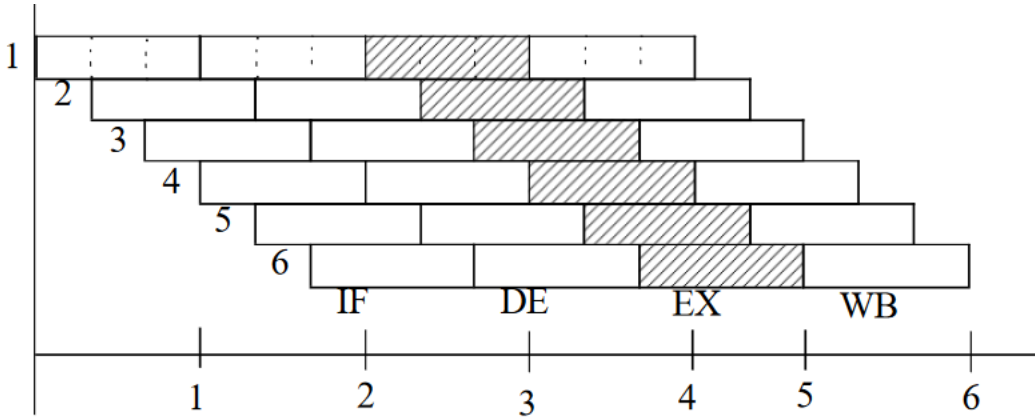
- Instruction Level Parallelism : Beyond Pipelining
- Other Performance enhancements

Beyond pipelining

- What are the limitations of performance so far?
  - Pipelining
    - Clocks per Instruction (CPI)? Instructions per clock (IPC)?

Beyond pipelining

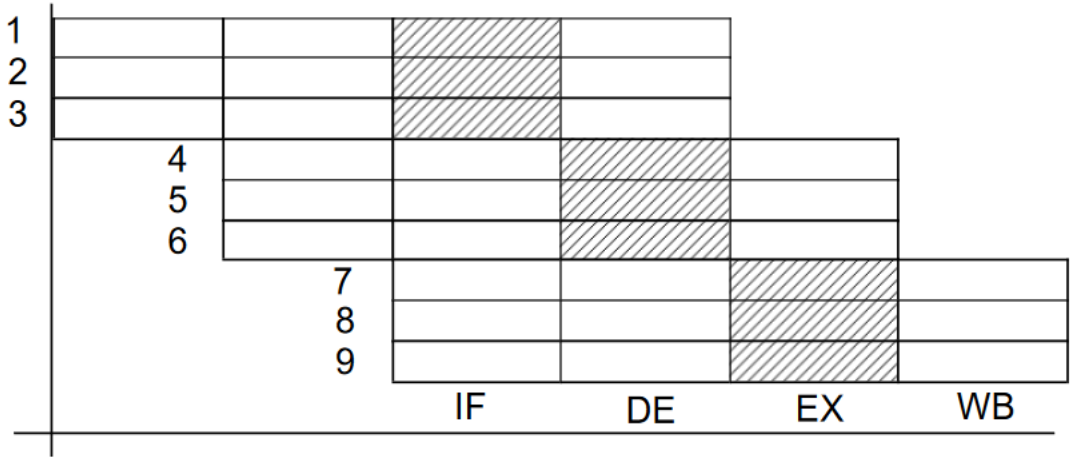
- What are the limitations of performance so far?
  - Pipelining
    - Clocks per Instruction (CPI)? Instructions per clock (IPC)?
  - Upper-bound on throughput
    - IPC <=1 or CPI >= 1
    - A.k.a. “Flynn’s Bottleneck”
- Super-pipelining?
  - Minor cycles



Beyond pipelining

- What are the limitations of performance so far?
  - Pipelining
    - Clocks per Instruction (CPI)? Instructions per clock (IPC)?

- Superscalar
  - Can we execute multiple instructions parallelly?
  - Two (or more) parallel pipelines?



Superscalar design

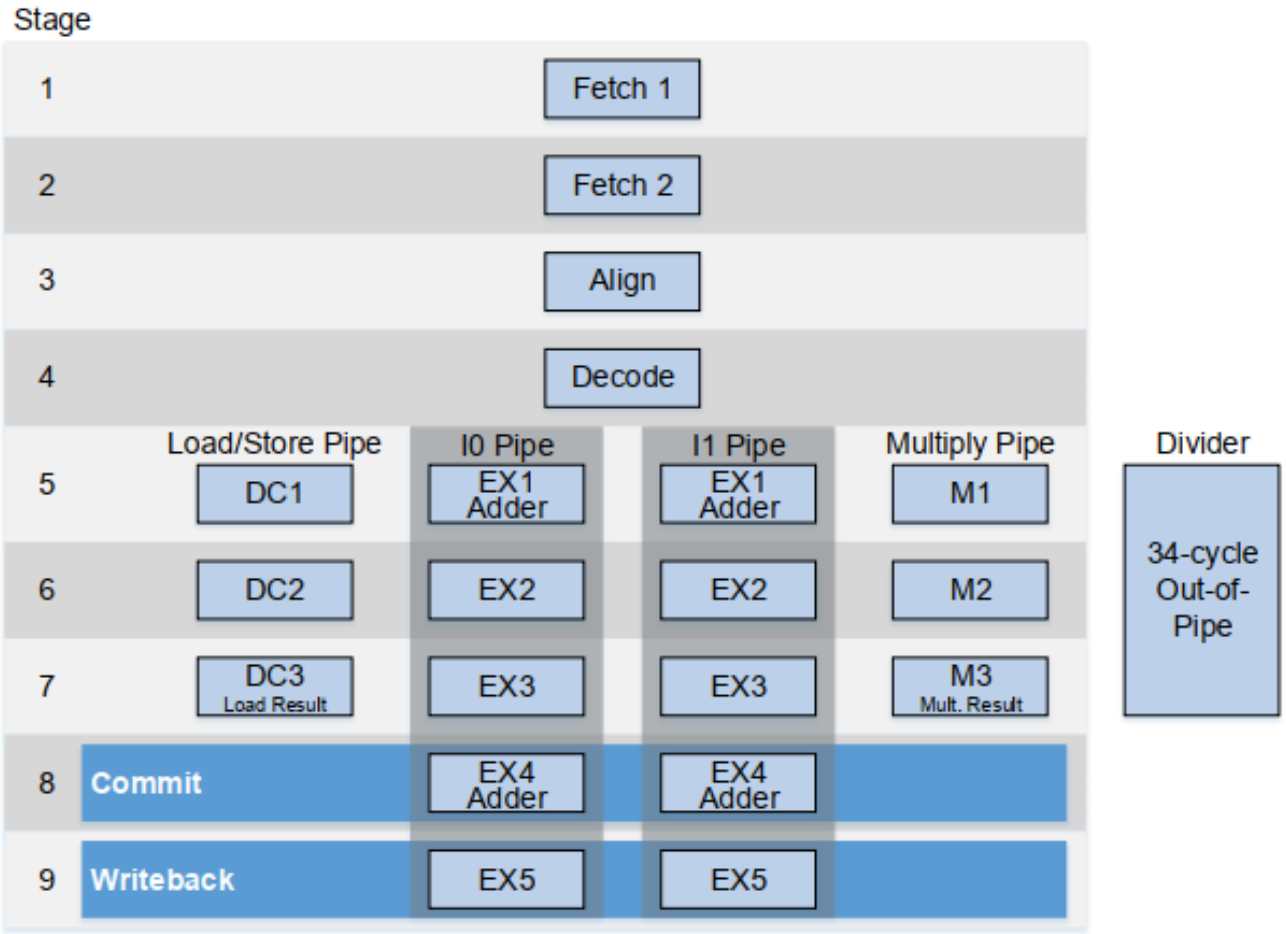


Figure 1-2 SweRV EH1 Core Pipeline

Superscalar design

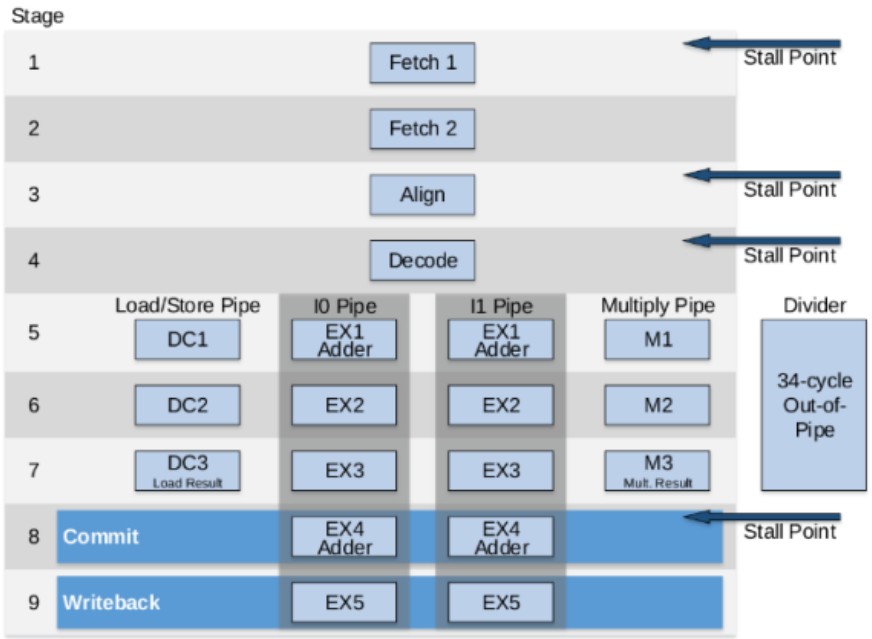


Figure 18. SweRV EH1 core microarchitecture

- Pipeline: Single issue; in-order
- Superscalar: Multiple issue; in-order
  - Multiple pipelines
- Stall Points?
- Out of order execution?

https://courses.cs.washington.edu/courses/cse378/10au/lectures/Pentium4Arch.pdf

Single-core systems

- Pipelining
- Superscalar execution units
  - Dispatch multiple instructions during a single clock cycle
- Out of order execution
  - Execute instructions as soon as their operands are ready
- Large cache

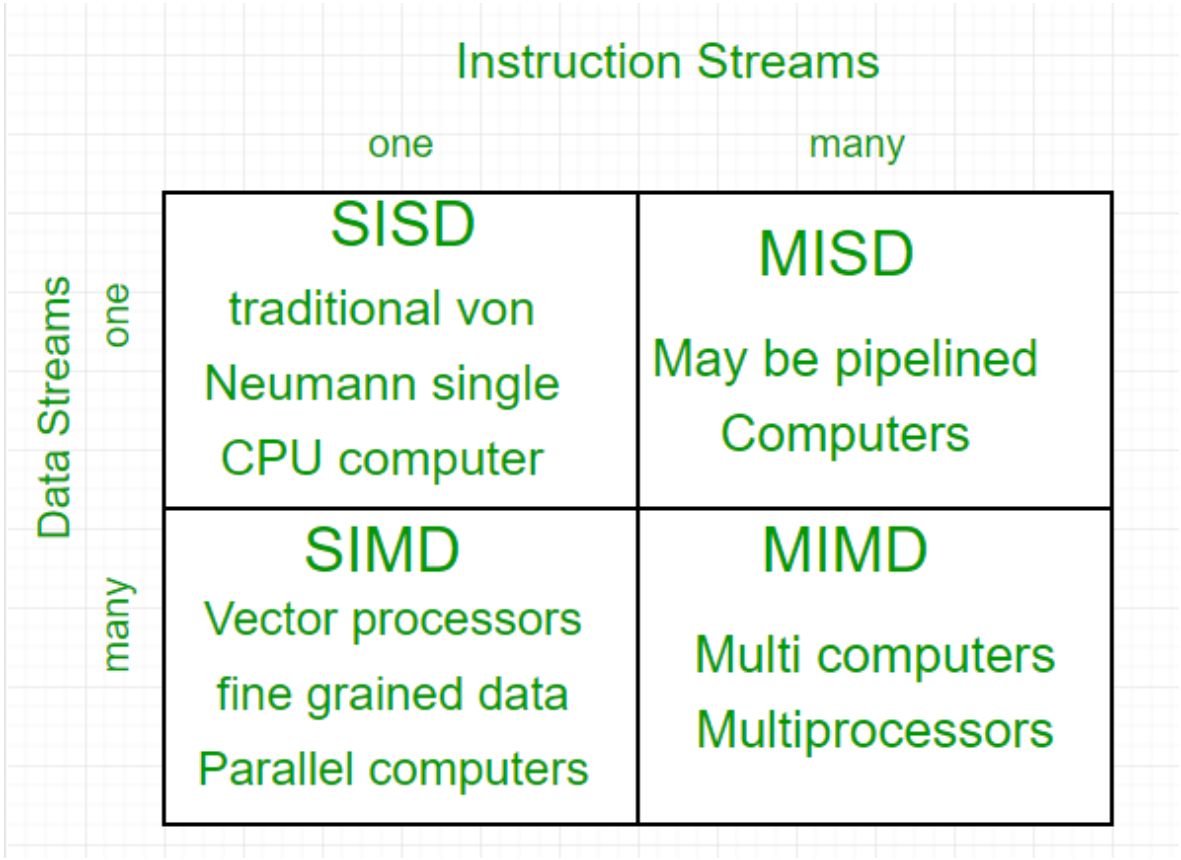
Von Neumann bottleneck

- Data accessing is slow
- Overcoming the bottleneck
  - Caching
  - Prefetching
  - Speculative execution (Branch prediction)
  - New types of RAM
  - Multithreading
  - Near data processing
  - Hardware acceleration
  - System on a chip

Parallelism

- Classes of parallelism in applications
  - Data-Level Parallelism (DLP)
  - Task-Level Parallelism (TLP)
- Classes of architectural parallelism
  - Instruction-Level Parallelism (ILP)
    - Exploits DLP in pipelining & speculative execution
  - Vector architectures/Graphic Processor Units (GPUs)
    - Exploit DLP by applying same instruction on many data items
  - Thread-Level Parallelism
    - Exploit DLP & TLP in cooperative processing by threads
  - Request-Level Parallelism
    - Parallel execution of tasks that are independent

Flynn's Taxonomy



Multithreading

Multithreading

- So far, in the processor we run a single program sequence
  - Single thread

- Can we run two programs at once?
  - Ex:
    - Operating system / GUI / User applications
    - Listen to music while editing a document
    - Download files via network while browsing another web page

Multi-threading

How to implement multithreading?

- Time sharing schemes in a single processor/core/ heart
  - Usually, processor core can run much faster than the human response times
  - OS supported context switching
- Run two threads in two different processors at once
  - Two computers?

Multithreading

- Simultaneous multi-threading/ Hyperthreading
  - CPU divides up its physical cores into virtual cores that are treated as if they are actually physical cores by the operating system
  - Intel proprietary technology

Complications in Multi-threading

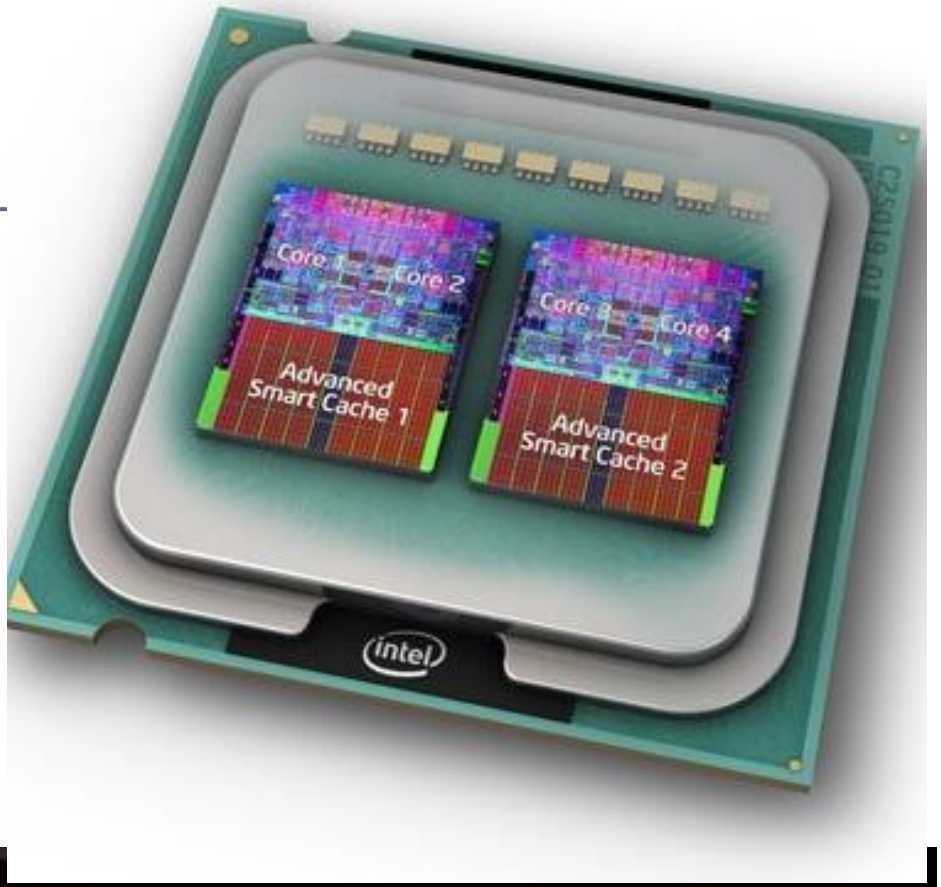
- Communicate between different applications
  - Communicate between different threads
- Shared memory
  - Each program should have its own data and instruction memory
  - Different programs need to communicate using a shared memory
  - Need to ensure programs do not corrupt other programs (Should maintain access rights)

Multi-core processors

Multi-core processors

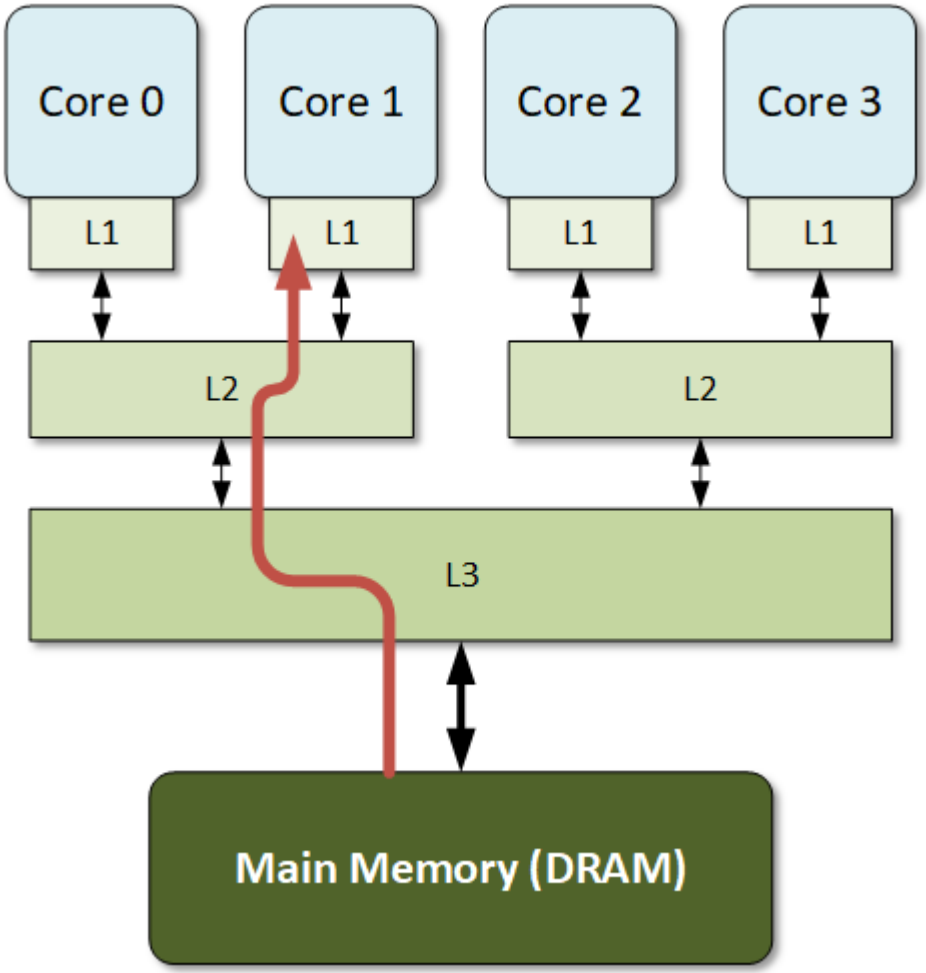
- More than one processor chip (Dual CPU)
  - Motherboard can mount two different processors
- Multiple cores / hearts in a single chip

- If there is more than one core, each thread can be assigned to run in a different core.
  - Pros:
    - Alternative to time sharing scheme
    - Own register banks. So, no need to switch contexts
  - Cons:
    - Cache coherence?



https://cpuninja.com/dual-cpu-motherboard/
https://www.datacenterknowledge.com/amd/amd-launches-next-generation-epyc-server-processors

Cache placement options



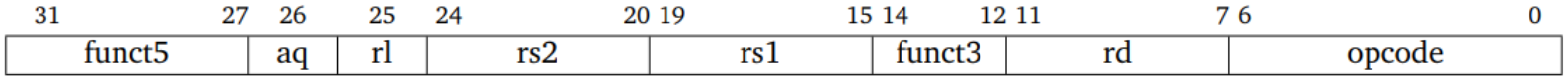
https://www.insidetheiot.com/cache-in-multicore-systems/

Programming Model

- Programs which is written for serial processing can run without a problem.
  - OS can assist the management of the thread
- Programs which are parallelly executable can be written using multi-thread support
  - “Concurrent programming”
  - Shared memory concerns must be handled
    - Atomicity
    - Mutual exclusion
    - Synchronization

Instructions for multithreading

RV32A Atomic Extension



Inst	Name	FMT	Opcode	funct3	funct5	Description (C)
`lr.w`	Load Reserved	R	`0101111`	`0x2`	`0x02`	`rd = M[rs1], reserve M[rs1]`
`sc.w`	Store Conditional	R	`0101111`	`0x2`	`0x03`	`if (reserved) { M[rs1] = rs2; rd = 0 } else { rd = 1 }`
`amoswap.w`	Atomic Swap	R	`0101111`	`0x2`	`0x01`	`rd = M[rs1]; swap(rd, rs2); M[rs1] = rd`
`amoadd.w`	Atomic ADD	R	`0101111`	`0x2`	`0x00`	`rd = M[rs1] + rs2; M[rs1] = rd`
`amoand.w`	Atomic AND	R	`0101111`	`0x2`	`0x0C`	`rd = M[rs1] & rs2; M[rs1] = rd`
`amoor.w`	Atomic OR	R	`0101111`	`0x2`	`0x0A`	`rd = M[rs1] | rs2; M[rs1] = rd`
`amoxor.w`	Atomix XOR	R	`0101111`	`0x2`	`0x04`	`rd = M[rs1] ^ rs2; M[rs1] = rd`
`amomax.w`	Atomic MAX	R	`0101111`	`0x2`	`0x14`	`rd = max(M[rs1], rs2); M[rs1] = rd`
`amomin.w`	Atomic MIN	R	`0101111`	`0x2`	`0x10`	`rd = min(M[rs1], rs2); M[rs1] = rd`

`ecall`	Environment Call	I	`1110011`	`0x0`	`imm=0x0`	`Transfer control to OS`	
`ebreak`	Environment Break	I	`1110011`	`0x0`	`imm=0x1`	`Transfer control to debugger`	

Other instructions like csrrs
Privileged instructions

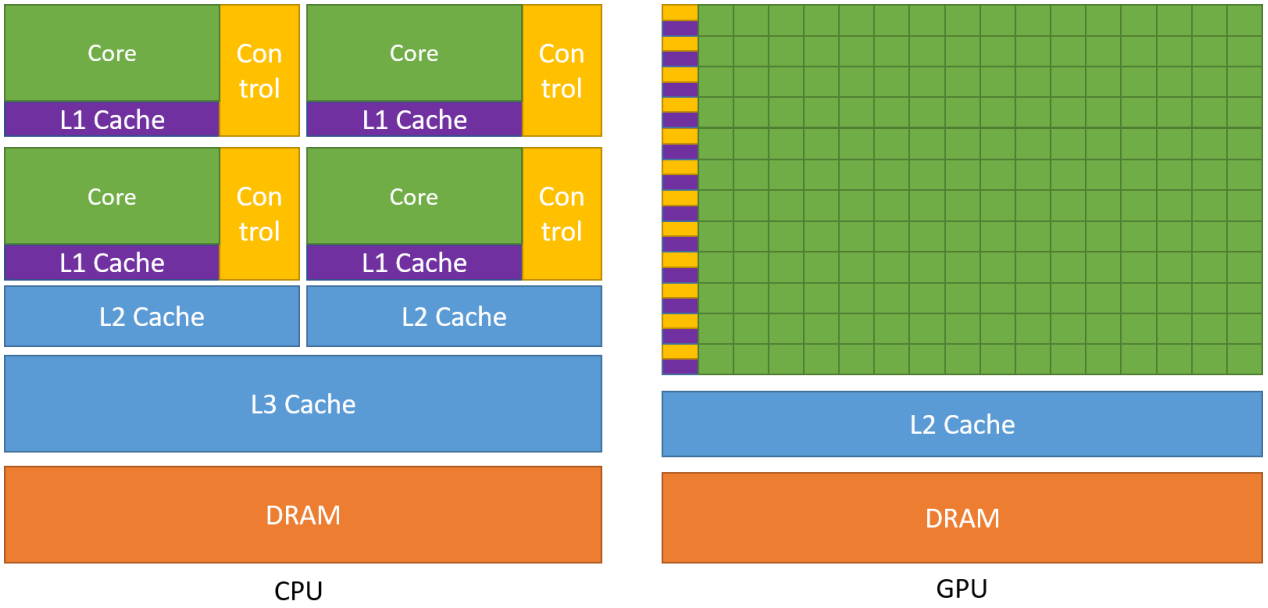
Many-core Architectures

Multi-core vs Many-core machines

Multi-Core Processors	Many-Core Processors
• As an extension of single core processors • Designed to run both parallel and serial code • Performance emphasis on single thread performance (out-of-order execution, pipelining, superscalar execution units, larger cache, shared memory.) • Few cores • Complemented by manycore accelerators (ex. GPU)	• Intended for higher degree of explicit parallelism • Higher throughput • Low power consumption • High latency and lower single thread performance

Graphical Processing Units (GPU)

- Graphics are inherently parallel
  - Drawing each pixel can be done independently.
    - Scanning row by row is not efficient
    - Divide the image to number of sections and assign to number of threads/processors
    - If we have
      - Two cores: Twice the performance as single core
      - Four cores: Four times the single core
      - N cores: N times the single core
- Offload the graphics processing entirely to a specialized co-processor(s)



- For applications with high degree of parallelism
- Large number of cores allow massively parallel programs to execute in parallel

Programming for GPUs

- For applications with high degree of parallelism
- Large number of cores allow massively parallel programs to execute in parallel

Traditional loops

```
void
mul(const int n,
    const float *a,
    const float *b,
          float *c)
{
  int i;
  for (i = 0; i < n; i++)
    c[i] = a[i] * b[i];
}
```

OpenCL Kernel

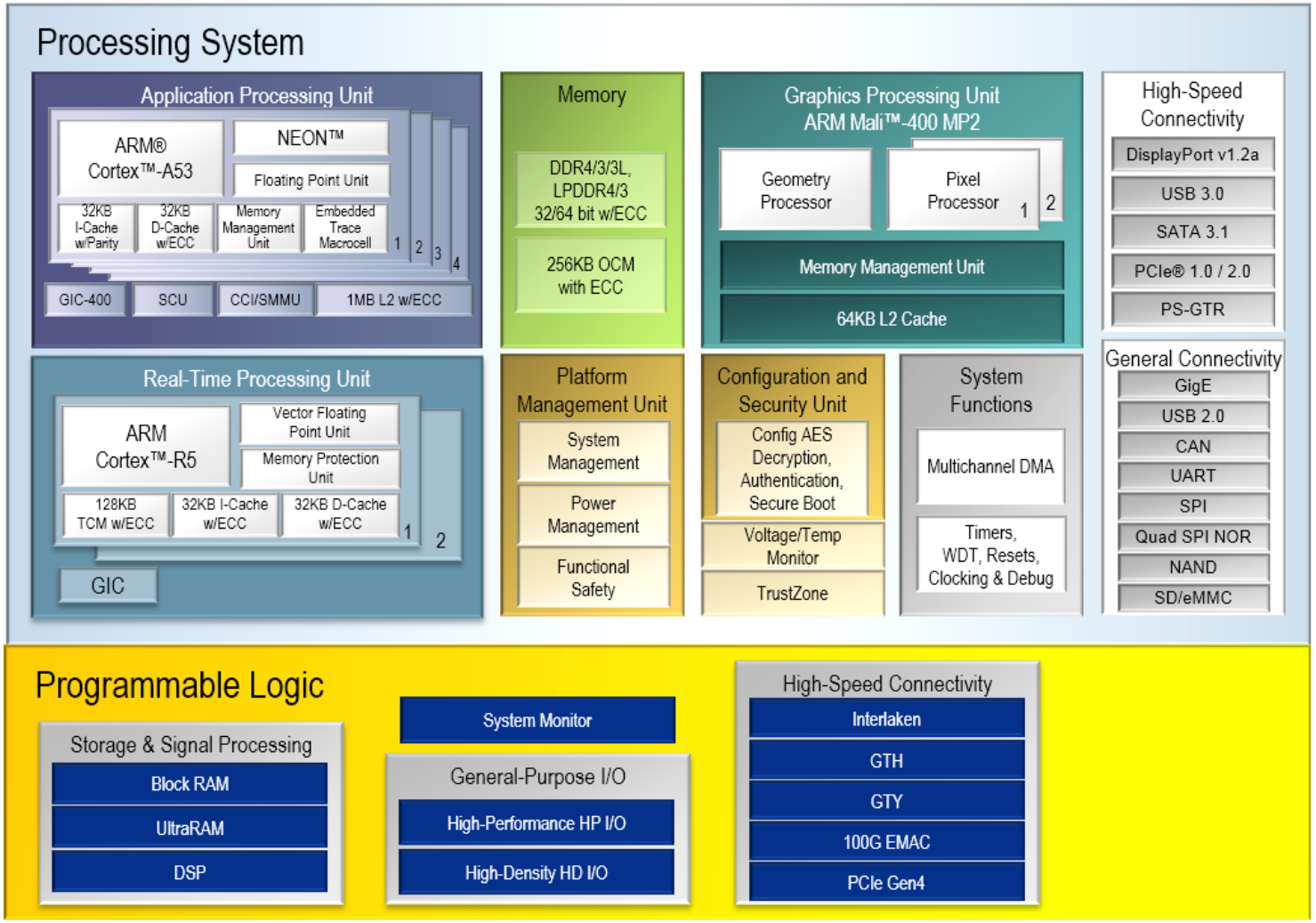
```
__kernel void
mul(__global const float *a,
    __global const float *b,
    __global       float *c)
{
  int id = get_global_id(0);
  c[id] = a[id] * b[id];
}
```

Heterogeneity

Heterogeneity

- Symmetric multiprocessing (SMP)
  - Two (or more) copies of similar processors
  - Typical in Many-core/multi-core systems
- Asymmetric multiprocessing(AMP)
  - Processors for specific tasks
    - Audio processor
    - Graphic co-processor
    - Direct Memory Access (DMA)
    - Cryptographic co-processors
- MPSoCs
    - Heterogenous Multi Processor Systems on Chip.

System-on-Chip (SoC)s



https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html

Other Co-Processing Units

- Neural Processing Units / Tensor Processing Units / Systolic arrays / Data Processing Units
  - Neural Network on hardware
  - Network on a chip
  - Multiply and accumulate applications
  - Massively parallel integration
  - Convolution, correlation, matrix multiplication or data sorting tasks.
  - Dynamic programming algorithms, used in DNA and protein sequence analysis

Warehouse scale parallelism

- Cloud computing/Distributed computing approaches
- Different Programming models
  - Example:
    - Map - Reduce
    - Lambda architecture

New Paradigms

- Processing In Memory
  - Can we bring processing and memory together.
    - Brain does not have separate memory and processing units

Thank you!