Course Overview

- Day 1: Fundamentals
  - accelerator architectures, review of shared-memory programming
- Day 2: Programming for GPUs
  - thread management, memory management, streaming
- Day 3: Advanced GPU Programming
  - performance profiling, reductions, synchronization
- Day 4: OpenCL Programming
  - C and C++ APIs, kernel programming, memory hierarchy
- Day 5: Advanced OpenCL and Futures
  - synchronization, metaprogramming, FPGA, next-generation architectures
- https://cs.anu.edu.au/courses/acceleratorsHPC/fundamentals/
- https://github.com/ANU-HPC/accelerator-programming-course

Setup

```
git clone https://github.com/ANU-HPC/accelerator-programming-course.git
```

or fork the repository and clone your fork, then

```
git remote add upstream https://github.com/ANU-HPC/accelerator-programming-course.git
```

```
cd accelerator-programming-course
./run_docker.sh
```

or

```
./run_docker_with_gui.sh
```

Accelerator Architectures

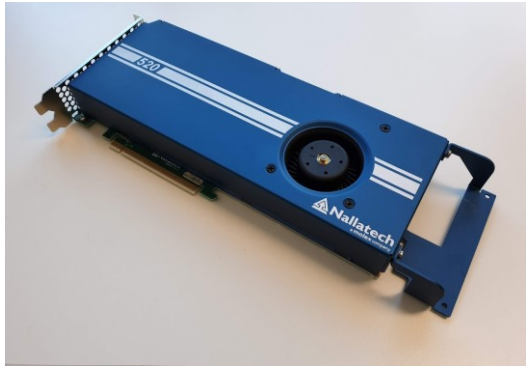
Accelerators for Parallel Computing

Goal: solve big problems (quickly)

-> Divide into sub-problems that can be solved concurrently

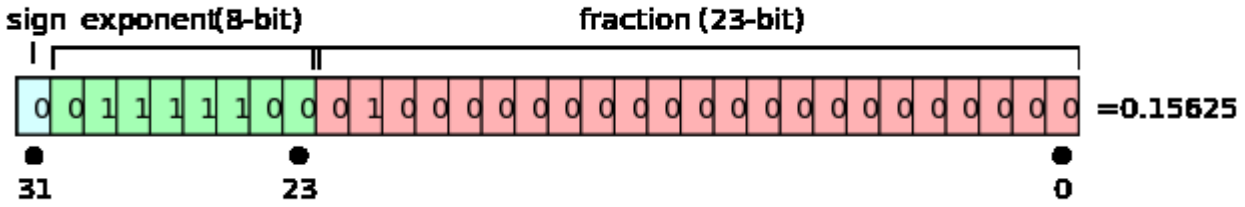
Why not use traditional CPUs?

-> Performance and/or energy



Pipelining

- Example: adding floating-point numbers

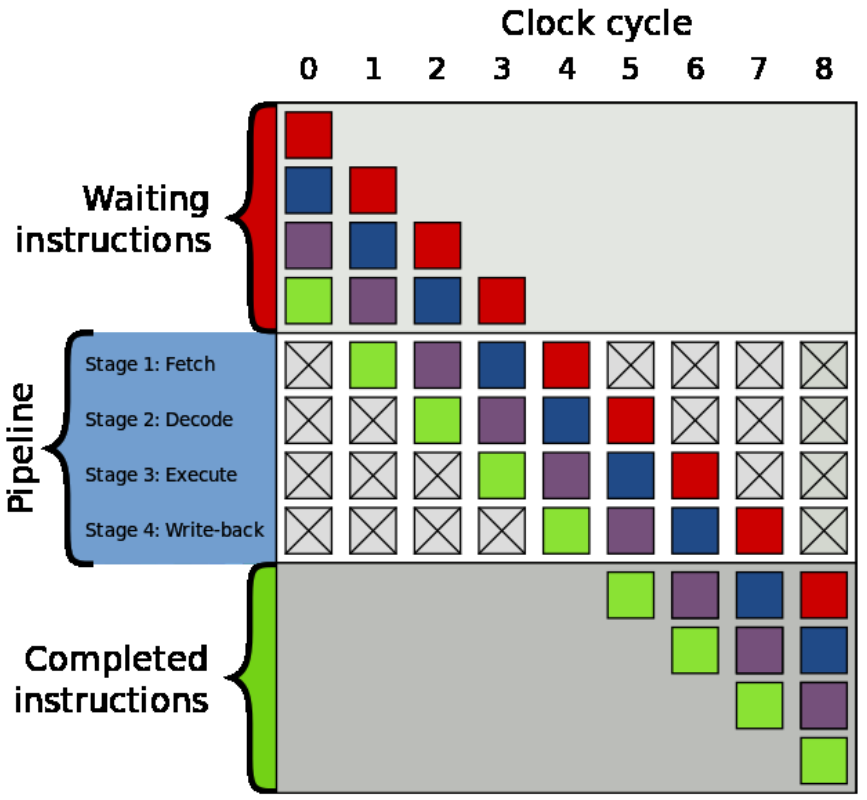


Codekaizen, IEEE 754 Single Floating Point Format, CC BY 3.0

- Possible steps:
  - determine largest exponent
  - normalize significand of the smaller exponent to the larger
  - add significand
  - re-normalize the significand and exponent of the result
- Multiple steps each taking 1 tick implies 4 ticks per addition (FLOP)

Operation Pipelining

- First instruction takes four cycles to appear (startup latency)
- Asymptotically achieves one result per cycle
- Steps in the pipeline are running in parallel
- Requires same operation consecutively on independent data items
- Not all operations are pipelined



operation	latency	repeat
+ - ×	3-5	1
/	16	5
sqrt	21	7

Agner Fog (2018). Instruction Tables (Intel Skylake)

Instruction Pipelining

- Break instruction into *k* stages
  ⇒ can get ⩽ k-way parallelism
- E.g. (*k* = 5) stages:
  - IF = Instruction Fetch
  - ID = Instruction Decode
  - EX = Execute
  - MEM = Memory Access
  - WB = Write Back

Instr. No.	Pipeline Stage						
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX
Clock Cycle	1	2	3	4	5	6	7

Inductiveload, 5 Stage Pipeline,

- Note: MEM and WB memory access may stall the pipeline
- Branch instructions are problematic: a wrong guess may flush succeeding instructions from the pipeline

Pipelining: Dependent Instructions

- Principle: CPU must ensure result is the same as if no pipelining / parallelism
- Instructions requiring only 1 cycle in EX stage:

```
add %1, -1, %1      ! r1 = r1 - 1
cmp %1, 0           ! is r1 = 0?
```

Can be solved by pipeline feedback from EX stage to next cycle

(Important) instructions requiring *c* cycles for execution are normally implemented by having *c* EX stages. The delays any dependent instruction by *c* cycles e.g. (*c* = 3):

```
fmuld %f0 , %f2 , %f4       ! I0: fr4 = fr0  fr2 (f.p.)
...                         ! I1:
...                         ! I2:
faddd %f4 , %f6 , %f6       ! I3: fr6 = fr4 + fr6 (f.p.)
```

Superscalar (Multiple Instruction Issue)

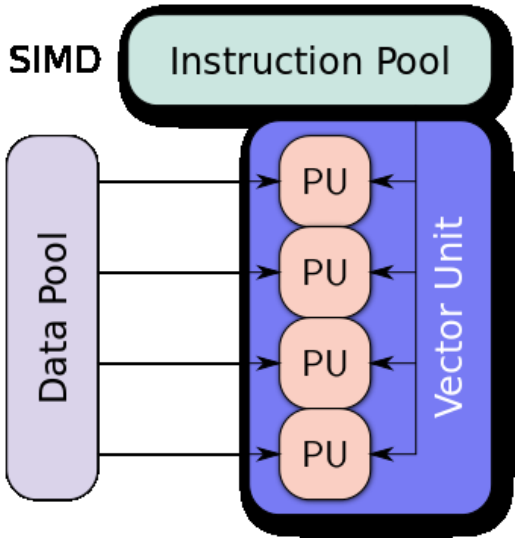
- Up to *w* instructions are scheduled by the H/W to execute together
- groups must have an appropriate 'instruction mix' e.g. UltraSPARC (*w* = 4):
  - $\leqslant$ 2 different floating point
  - $\leqslant$ 1 load / store ; $\leqslant$ 1 branch
  - $\leqslant$ 2 integer / logical
- have $\leqslant$ *w*-way ||ism over different types of instruction types
- generally requires:
  - multiple ($\geqslant$ w) instruction fetches
  - an extra grouping (G) stage in the pipeline
- amplifies dependencies and other problems of pipelining by w
- the instruction mix must be balanced for maximum performance
  - i.e. floating point ×, + must be balanced

Instruction Level Parallelism

- pipelining and superscalar, offer $\leqslant$ *kw*-way ||ism
- branch prediction alleviates issue of conditional branches
  - record the result of recently-taken branches in a table
- out-of-order execution: alleviates the issue of dependencies
  - pulls fetched instructions into a buffer of size W, $W \geqslant w$
  - execute them in any order provided dependencies are not violated
  - must keep track of all 'in-flight' instructions and associated registers ($O(W^2)$ area and power!)
- in most situations, the compiler can do as good a job as a human at exposing this parallelism (ILP was part of the 'Free Lunch')

SIMD (Vector Instructions)

- Data parallelism: apply the same operation to multiple data items at the same time
- More efficient: single instruction fetch and decode for all data items
- Vectorization is key to making full use of integer / FP capabilities:
  - Intel Core i7-8850H AVX-2 (256-bit)
    e.g. 8x32-bit operands
  - Intel KNL: AVX-512 (512-bit)
    e.g. 16x32-bit operands



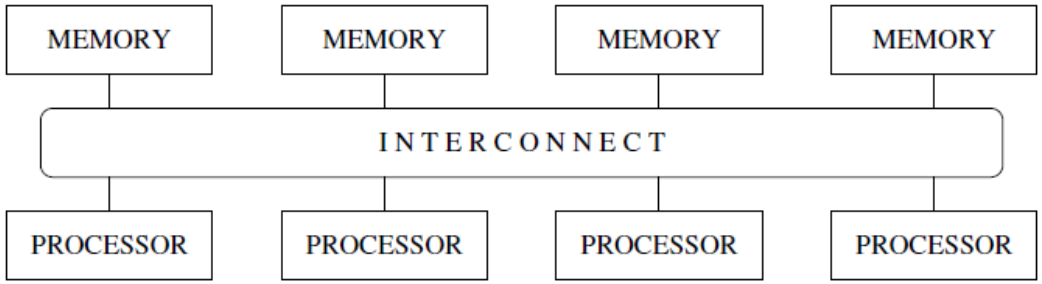
Vadikus, SIMD2, CC BY-SA 4.0

Barriers to Sequential Speedup

- Clock frequency:
  - Dennard scaling 0.7× dimension / 0.5× area
    ⇒ 0.7× delay / 1.4× frequency
    ⇒ 0.7× voltage / 0.5× power
  - … until 2006: cannot reduce voltage further due to leakage current
- Power wall: energy dissipation limited by physical constraints
- Memory wall: transfer speed and number of channels also limited by power
- ILP wall: diminishing returns on parallelism due to risks of speculative execution

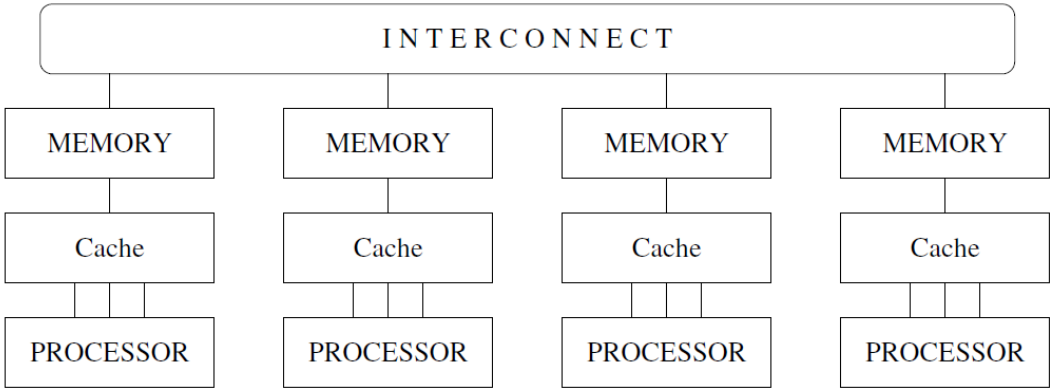
Multicore

- processors interact by modifying data objects stored in a shared address space
- simplest solution is a flat or uniform memory access (UMA)
- scalability of memory bandwidth and processor-processor communications (arising from cache line transfers) are problems
- so is synchronizing access to shared data objects
- Cache coherency & energy

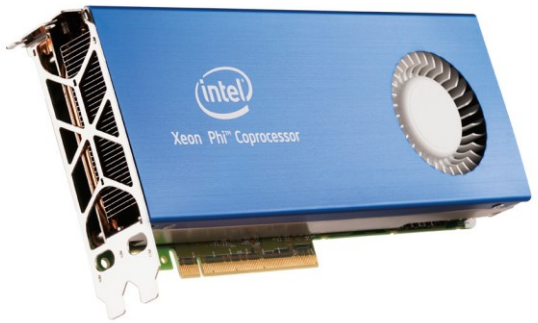


Non-Uniform Memory Access (NUMA)

- Machine includes some hierarchy in its memory structure
- all memory is visible to the programmer (single address space), but some memory takes longer to access than others
- in a sense, cache introduces one level of NUMA
- between sockets in a multi-socket Xeon system



Many-Core: Intel Xeon Phi

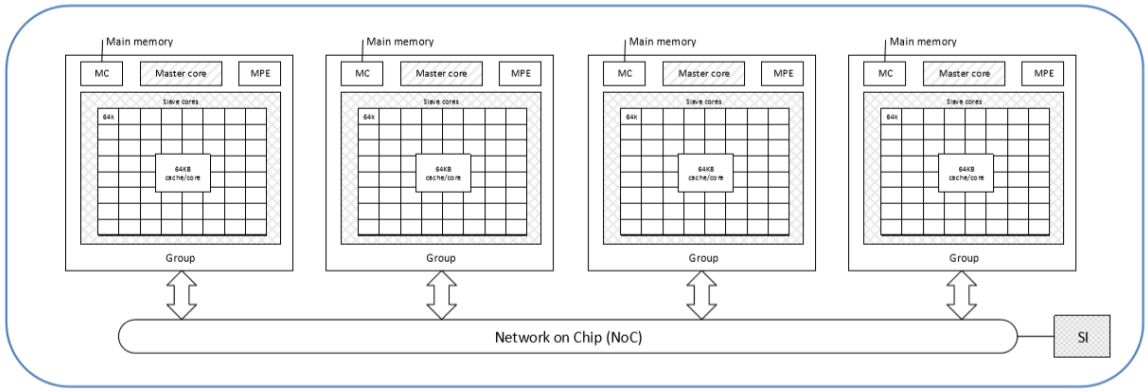


intel.com

- Knights Landing (14nm):
  - 64–72 simplified x86 cores
  - 4 hardware threads per core
  - 1.3–1.5 GHz
  - 512-bit SIMD registers
  - 2.6–3.4 TFLOP/s
  - 16GB 3D-stacked MCDRAM @ 400GB/s
  - Self-boot card (PCIe or Omni-Path), or as co-processor (PCIe)
- Knights Hill (10nm) – cancelled
- Knights Mill (14nm) = Knights Landing for deep learning

Many-Core: Sunway SW26010

- Non-cache-coherent chip:
  - Sunway 64-bit RISC instruction set, 1.45 GHz
  - 260 cores: 4 core groups (Management Processing Element + Compute Processing Element with 64 cores)
  - 256-bit SIMD registers
  - 8GB DDR3 RAM @ 136GB/s
  - 3 TFLOP/s
  - Sunway TaihuLight: 6 GFLOPS/W



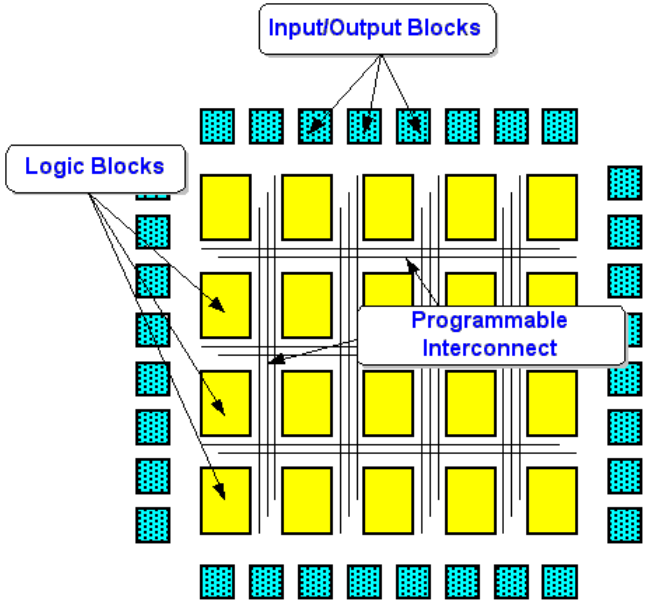
Jack Dongarra (2016). *Report on the Sunway TaihuLight System*

GPU

- Single Instruction, Multiple Thread (SIMT)
  - thread groups, divergence
- High-bandwidth, high-latency memory
  ⇒ many threads & register sets
- Multiple memory types:
  - Register, Local, Shared, Global, Constant
- Often limited by host-device transfer
- Nvidia Tesla P100
  - 56 cores, 3584 hardware threads
  - 1.3 GHz
  - 4.7 TFLOP/s
  - 16 GB stacked HBM2 @ 732 GB/s
  - TSUBAME 3.0: 13.7 GFLOPs/W

Field-Programmable Gate Array (FPGA)

- Reconfigurable hardware e.g. Stratix 10
- Types of functional units
  - LUTs, flip-flops
  - Logic elements
  - Memory blocks
  - Hard blocks: FP, transceivers, IO
- Long work pipeline is key
- Compile path:
  - OpenCL
  - Verilog/VHDL
  - Gate-level description
  - Layout
- Specialized microprocessors (ASIC, DSP)



http://www.fpga-site.com/faq.html