

# Building Blocks: Hardware Processors, Cores, Memory and Accelerators

## Partners



## Funding



# Reusing this material



This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License.

[http://creativecommons.org/licenses/by-nc-sa/4.0/deed.en\\_US](http://creativecommons.org/licenses/by-nc-sa/4.0/deed.en_US)

This means you are free to copy and redistribute the material and adapt and build on the material under the following terms: You must give appropriate credit, provide a link to the license and indicate if changes were made. If you adapt or build on the material you must distribute your work under the same license as the original.

Note that this presentation contains images owned by others. Please seek their permission before reusing these images.

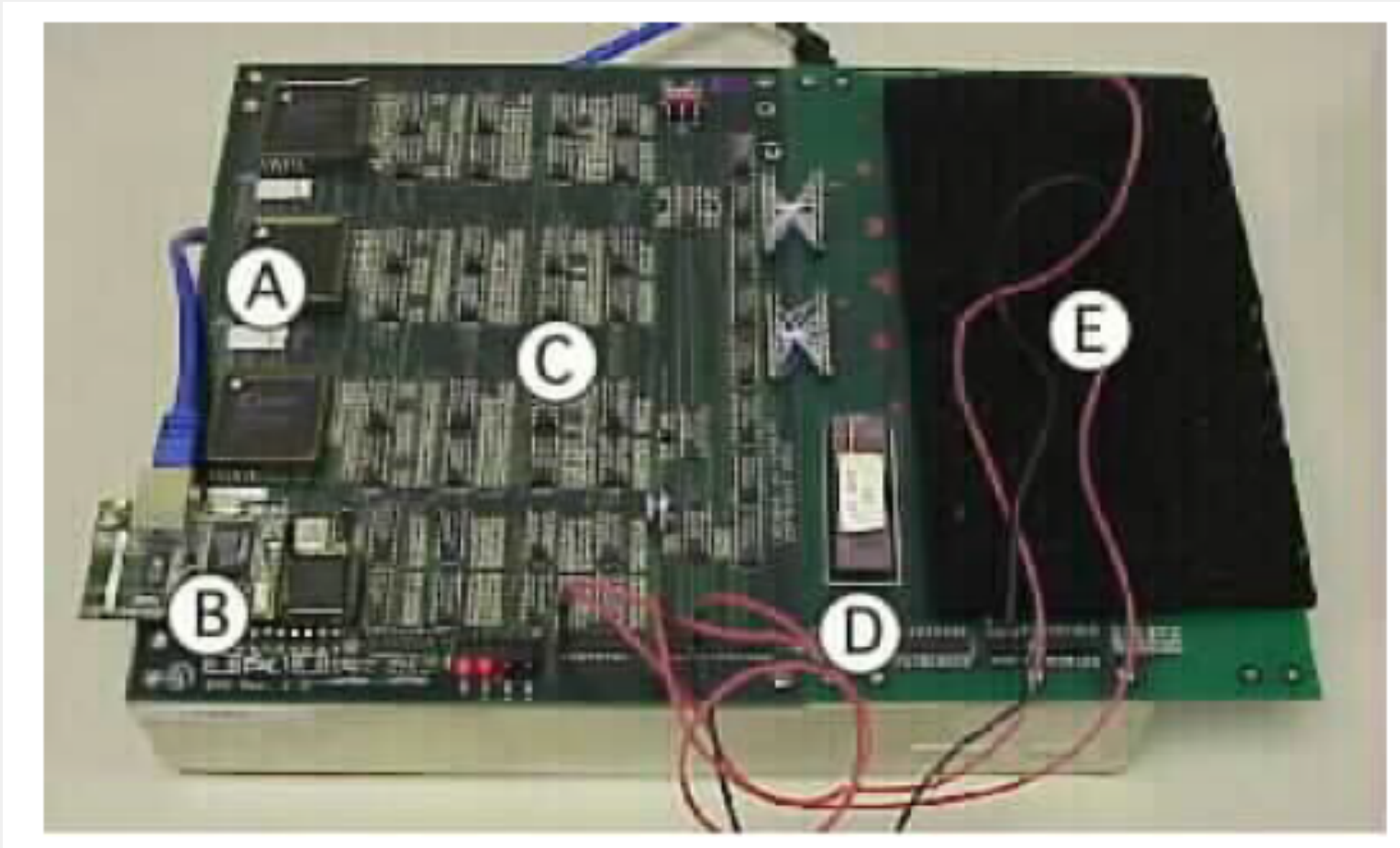
# Outline

- Computer Layout
  - Processors, Memory and Disk
- What does performance depend on?
  - Limits to performance
- Evolution of the Processor
  - Moore's Law
  - Parallelism in Hardware
    - Vector instructions
    - Hardware threads
    - Multicore
- Accelerators (GPGPU and Xeon Phi)
  - What are they good for?

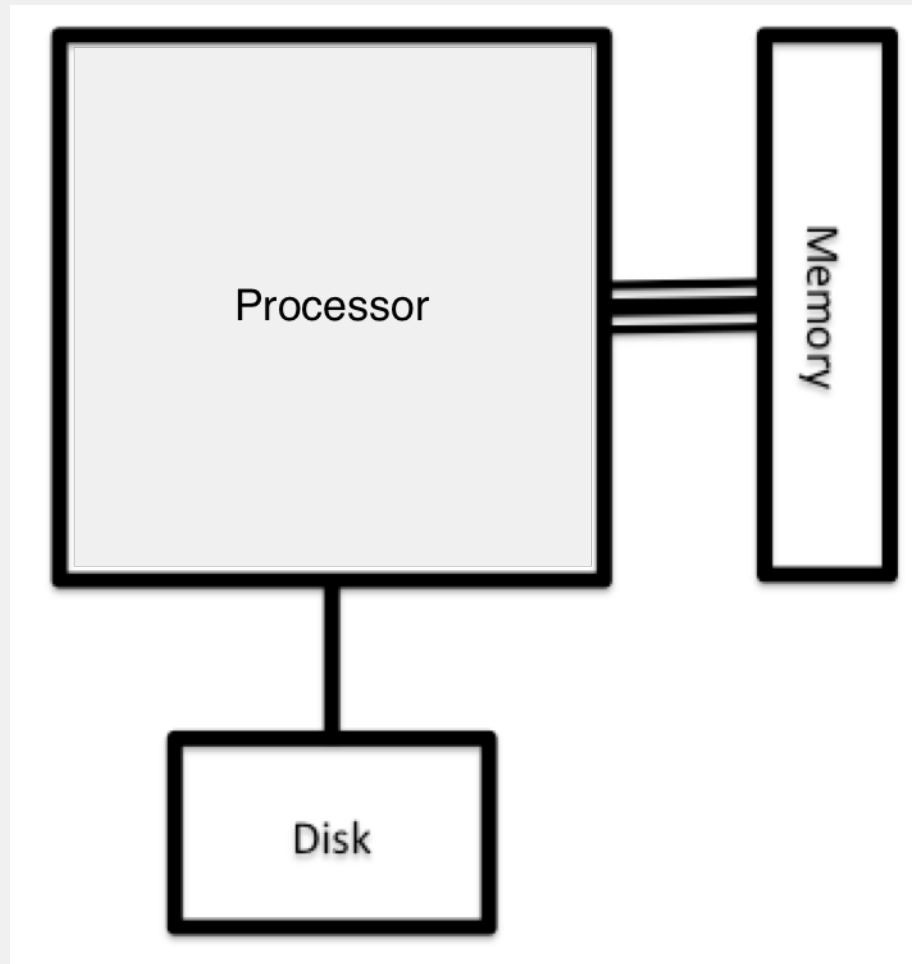
# What is a computer?



# What is a computer?



# Anatomy of a Computer



# Computing Essentials

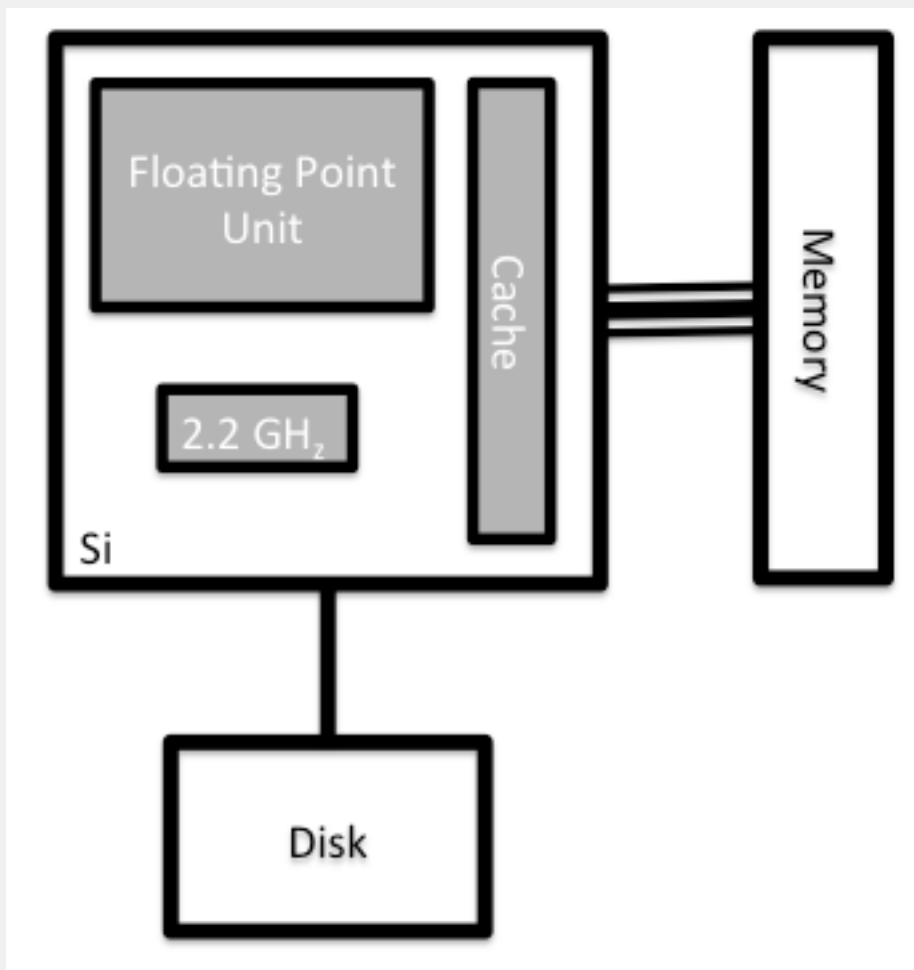
- The computers we care about can:
  - Store data
    - On disk, in memory
  - Perform operations on data
    - Using the processor
  - Store instructions that tell the processor what operations to perform
  - Deliver these instructions along with the required data to the processor at the right time so the operations can be executed without delay

# Data Access Bottlenecks

- Performance depends on getting data to the processor quickly
- Two key concepts regarding speed of data access:
  - Latency: time delay until data starts arriving
  - Bandwidth: amount of data arriving per second
- Disk access is slow
  - few hundred MB/s
  - significantly higher latency than memory
- Memory access is faster than disk
  - Large enough memory may contain all application data
  - Can still be too slow - few tens of GB/s
- Accessing cache (fast memory inside processor) is much faster
  - hundreds of GB/s
  - limited in size: a few MB at most



# Anatomy of a Computer (single-core processor)



# Processor Performance Bottlenecks

- Processors fetch data and execute instructions in cycles with a particular frequency (clock speed)
- Processor performance (time to solution) depends on:
  - Clock speed:
    - how often per second can the processor fetch data and instructions and execute these?
  - Sophistication of floating point and other processing units:
    - width: how much data of different types can be operated on at the same time, i.e. during one clock cycle?
    - what complexity of mathematical and logical operations can be performed efficiently?

# Moore's Law, Processor Evolution, and Parallelism in Hardware



# What to do with all those transistors?

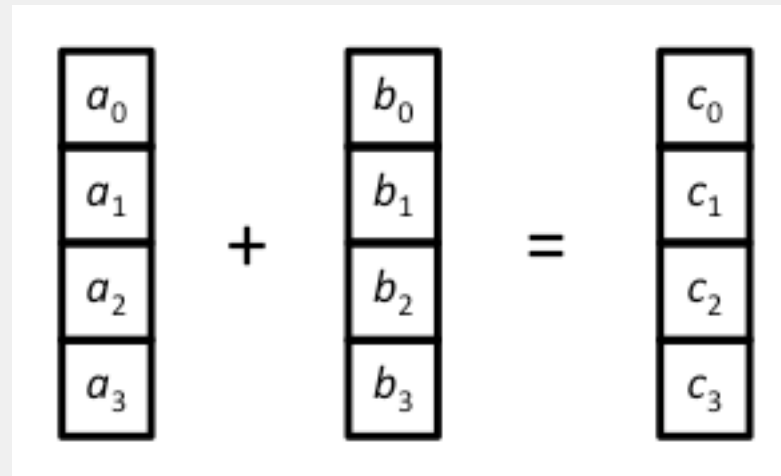
- For over 3 decades (the “good old days”) until early 2000’s
  - processors became more complicated / sophisticated
  - caches became bigger
  - clock speeds increased year on year (100 MHz, 200 MHz, 400MHz, ...)
  - for your program to run faster just wait a year and buy a newer processor
- Clock rate increases as inter-transistor distances decrease
  - so performance doubled every 18-24 months
- Came to a grinding halt about a decade ago
  - reached power and heat limitations
  - who wants a laptop that runs for an hour and scorches your trousers!

# Alternative approaches

- Introduce parallelism into the processor itself
  - vector instructions (“SIMD”)
  - simultaneous multi-threading (“SMT”)
  - multicore

# Single Instruction Multiple Data (SIMD)

- For example, vector addition:



- single instruction adds 4 numbers
- potential for 4 times the performance

# Symmetric Multi-threading (SMT)

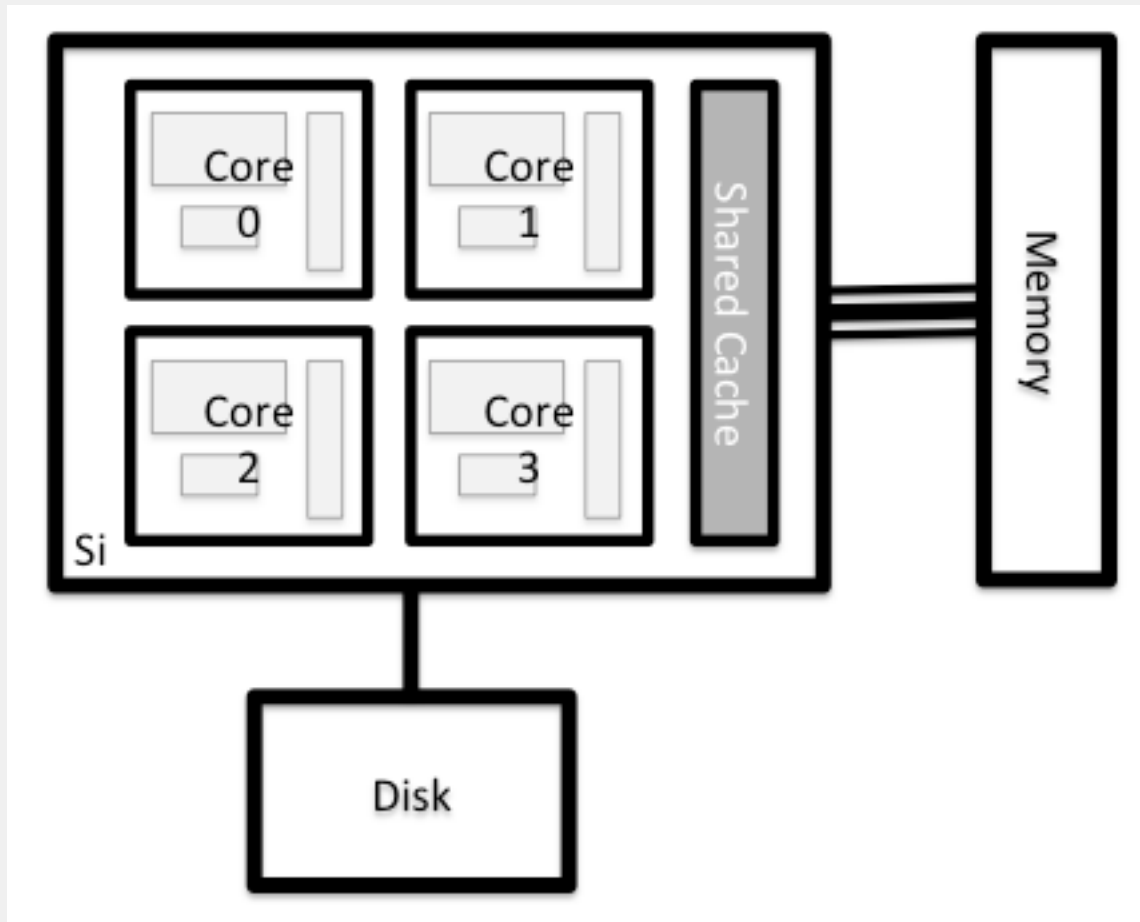
- Some hardware supports running multiple instruction streams simultaneously on the same processor, e.g.
  - stream 1: loading data from memory
  - stream 2: multiplying two floating-point numbers together
- Known as *Symmetric Multi-threading (SMT)* or *hyperthreading*
- Threading in this case can be a misnomer as it can refer to processes as well as threads
  - These are “hardware threads”, not software threads.
  - Intel Xeon supports 2-way SMT
  - IBM BlueGene/Q 4-way SMT



# Multicore

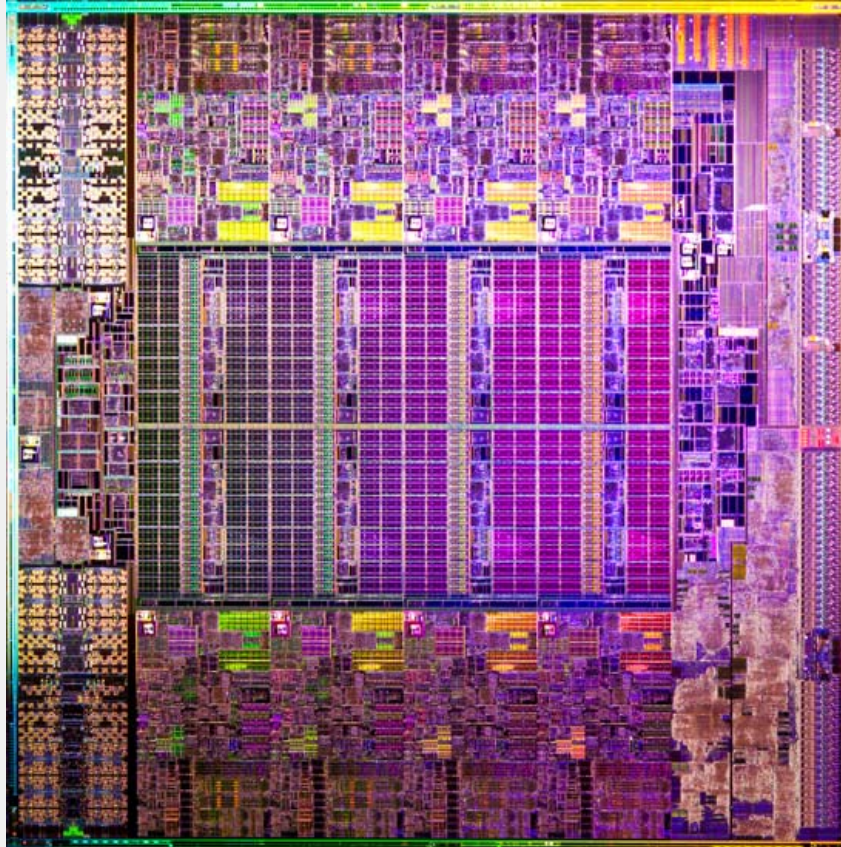
- Twice the number of transistors gives 2 choices
  - a new more complicated processor with twice the clock speed
  - two versions of the old processor with the same clock speed
- The second option is more power efficient
  - and now the only option as we have reached heat/power limits
- Effectively two independent processors
  - ... except they can share cache
  - commonly called “cores”

# Anatomy of a computer (single processor, multicore)



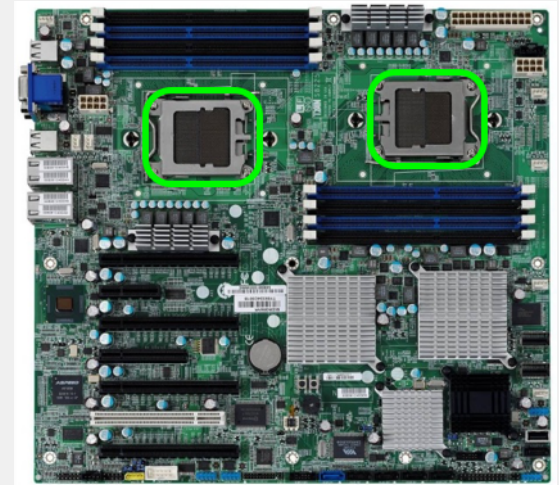
- Cores share path to memory
  - More cores makes this an *increasing* bottleneck!

# Intel Xeon E5-2600 – 8 cores HT



## What do you mean, “processor”?

- Terminology varies over time and in different contexts (hardware, software)
  - can be confusing
- Usually taken to mean “the thing you plug in to a socket on the motherboard”
  - e.g. motherboard top right has two processor sockets
- “CPU” is often ambiguous, and nowadays may refer to:
  - an entire multicore processor
  - a single processor core
  - a subunit of a single physical processor core that looks to the operating system like it’s an independent core (!)



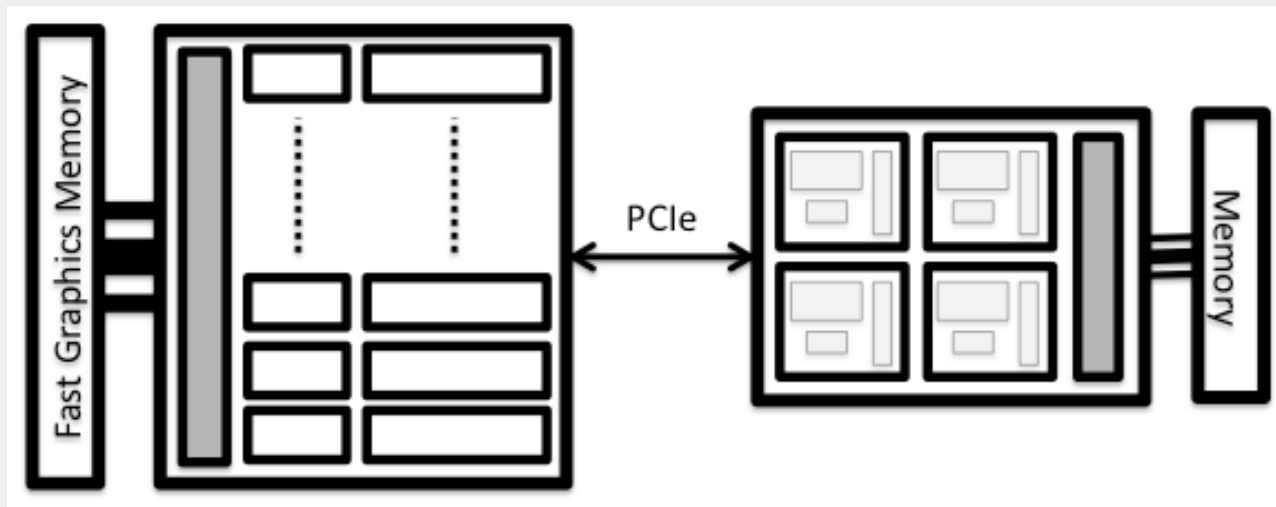
# Chip types and manufacturers

- x86 – Intel and AMD
  - “PC” commodity processors, SIMD (SSE, AVX) FPU, multicore, SMT (Intel); Intel currently dominates the HPC space.
- Power – IBM
  - Used in high-end HPC, high clock speed (direct water cooled), SIMD FPU, multicore, SMT; not widespread anymore.
- PowerPC – IBM BlueGene
  - Low clock speed, SIMD FPU, multicore, high level of SMT.
- SPARC – Fujitsu
- ARM – Lots of manufacturers
  - Starting to become relevant to HPC

# Accelerators

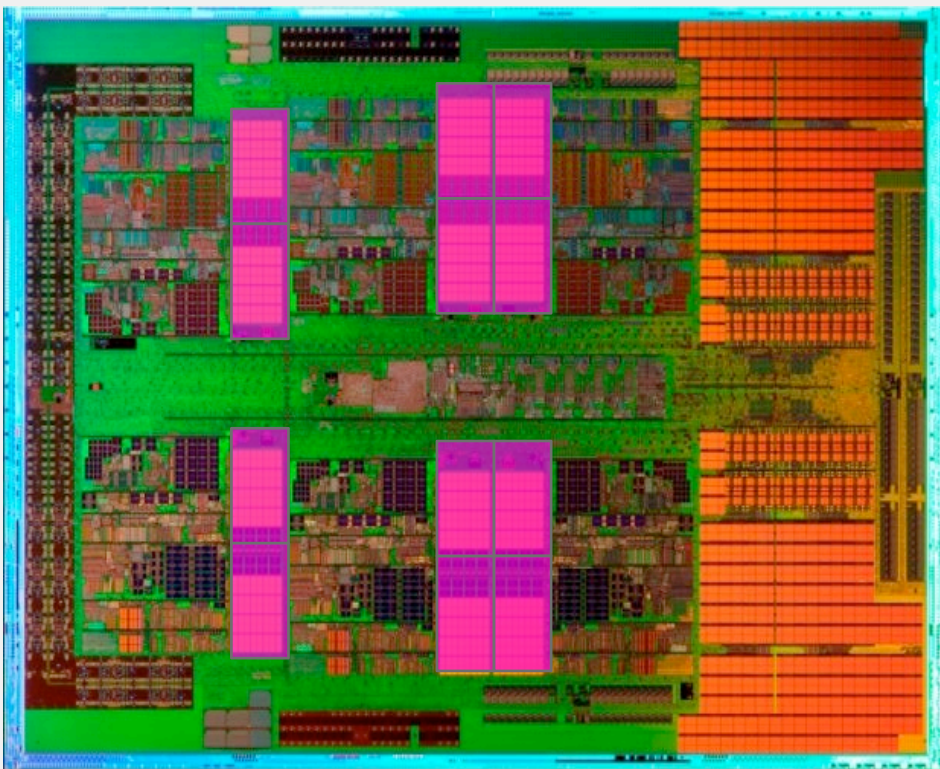
# Anatomy


- An *Accelerator* is a additional resource that can be used to off-load heavy floating-point calculation
  - additional processing engine attached to the standard processor
  - has its own floating point units and memory



# AMD 12-core CPU

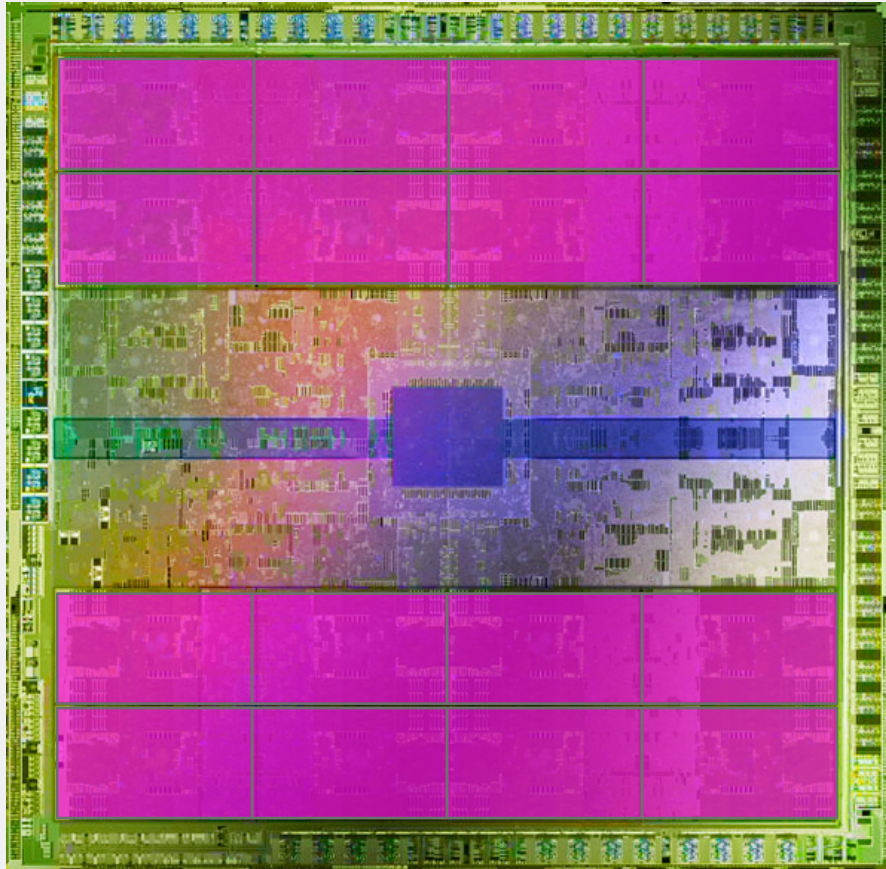
- Not much space on CPU is dedicated to computation




 = compute unit  
(= core)



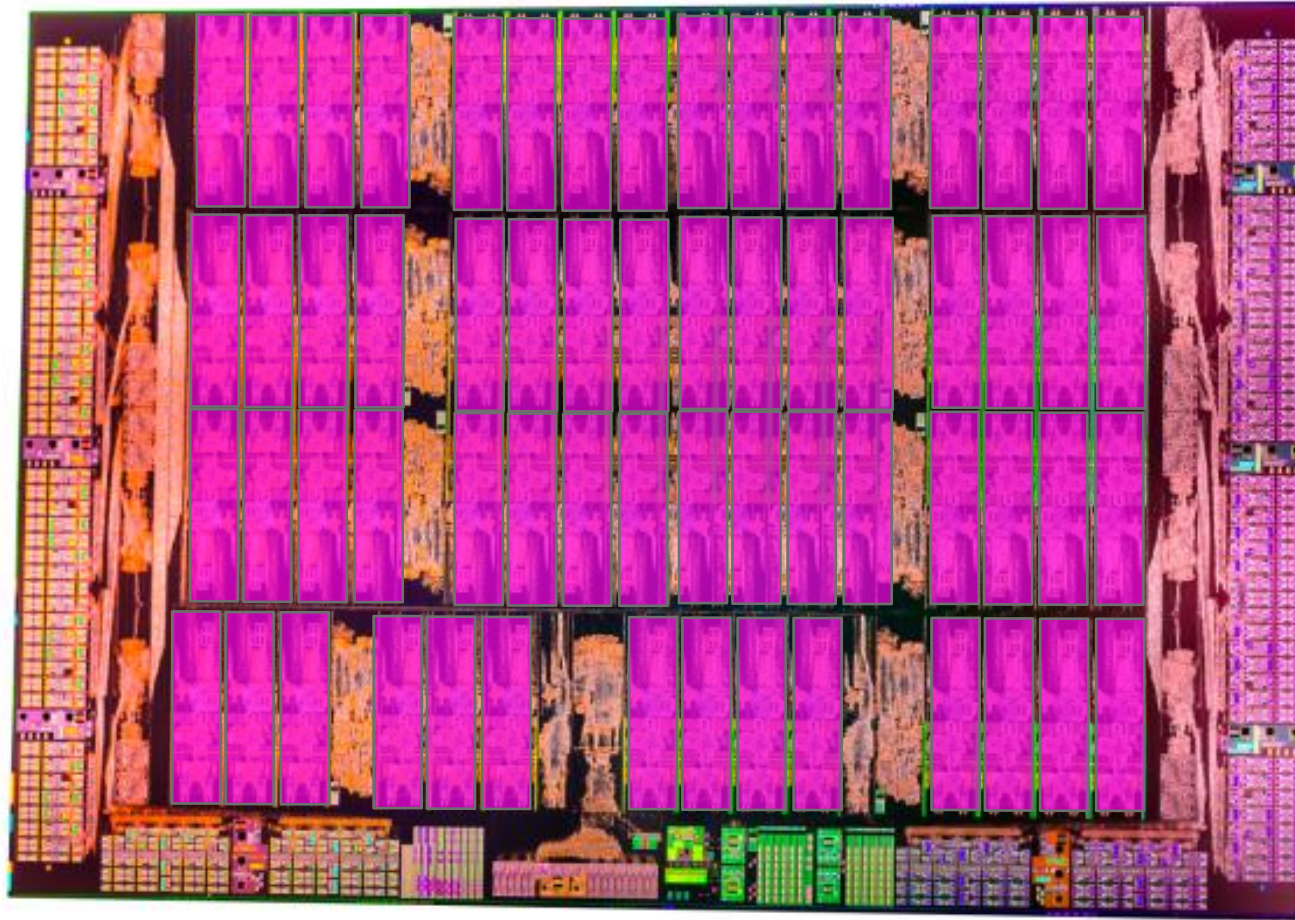
# NVIDIA Fermi GPU




- GPU dedicates much more space to computation
  - At expense of caches, controllers, sophistication etc

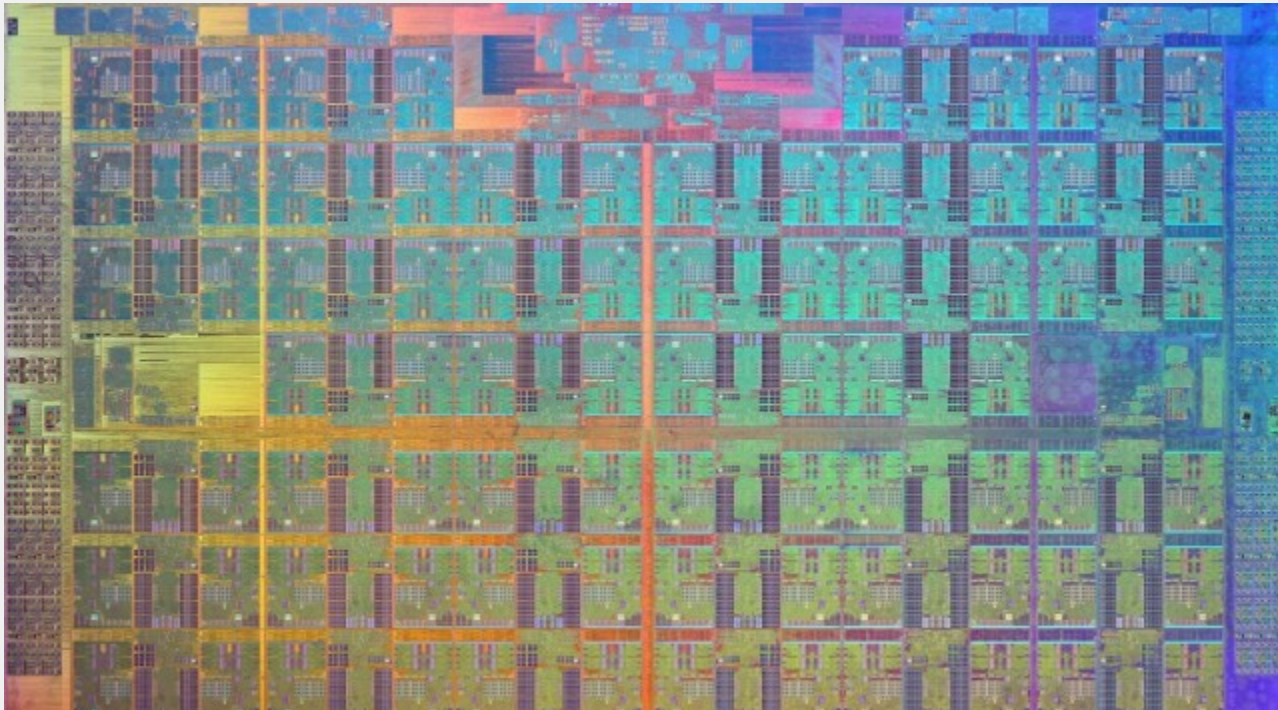
 = compute unit  
 (= SM  
 = 32 CUDA cores)

# Intel Xeon Phi – KNC (Knights Corner)



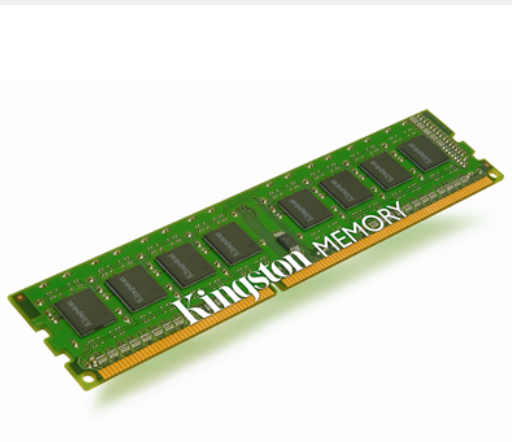
 = compute unit  
 (= core)

# Intel Xeon Phi – KNL (Knights Landing)



# Memory

- For most HPC applications, performance is very sensitive to memory bandwidth
- GPUs and Intel Xeon Phi both use graphics memory: much higher bandwidth than standard CPU memory
- KNL has high bandwidth on-board memory (16GB) in addition to standard (DDR) memory



CPUs use DRAM



GPUs and Xeon Phi use Graphics DRAM

# Performance (overview)

Application performance often described as:

- Compute bound (limited by processor performance)
- Memory bound (limited by memory access)
- IO bound (limited by disk access)
- (Communication bound – more on this later...)

For majority of current HPC codes:

- most calculations are limited by memory bandwidth
- processor can calculate much faster than it can access data

# Summary - What is automatic?

- Which features are managed by hardware/software and which does the user/programmer control?
  - Cache and memory – automatically managed
  - SIMD/Vector parallelism – automatically produced by compiler
  - SMT – automatically managed by operating system
  - Multicore parallelism – manually specified by the user
  - Use of accelerators – manually specified by the user