Cloud Computing

Hui Zeng *

Summer Semester 2021

*All notes are summarized from the lecture and tutorial materials provided by Prof. Michael Gerndt and his team. Images are retrieved from the lecture as well as tutorial slides.

Contents

1 Introduction

Different IT-trends boosts the need for cloud computing:

- Outsourcing, either infrastructure or management
- IT as a service: pay per use
- Re-centralization of data: similar to data centers, cloud be provided as a central place for data storage.
- Resource sharing instead of over-provisioning: same resource can be used for multiple purposes
- Server consolidation: instead of having multiple physical servers, with each dedicated to a certain service, servers are virtualized and put on one/reduced number of physical machines.
- Scalable computing
- Application dynamism: amount of request on web changes over time.
- Green computing, big data, stream processing, IoT, machine learning, etc.

Cloud Computing the definition is mainly divided by

- ubiquitous, convenient, on-demand network access to a **shared pool of configurable computing resources** (eg: networks, servers, storage, applications, services)
- resources can be **rapidly provisioned** and released with **minimal management effort** or service provider interaction
- cloud model is composed of
  - 3 service models
  - 4 deployment models
  - 5 essential characteristics

1.1 3 Service Models: IaaS, PaaS and SaaS

Three service models, ranking from outsourcing the least to the most: IaaS → PaaS → SaaS.

1.1.1 IaaS: Infrastructure as a Service

- Offering: provision processing, storage, networks, other fundamental computing resources
- Rights as consumer:
  - deploy and run **arbitrary** software, including **operating systems and applications**

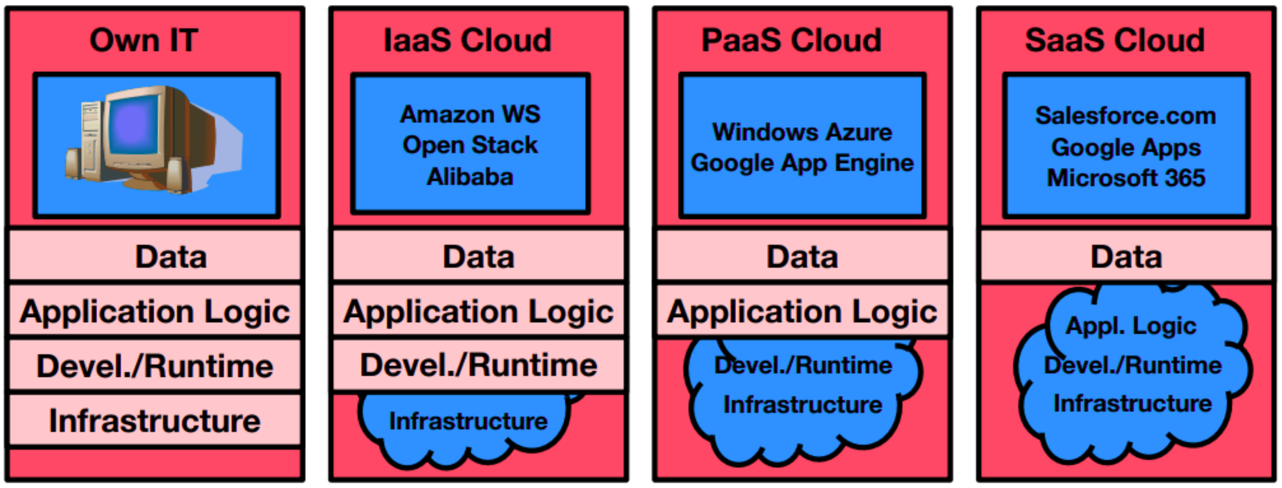
  - control over OS, storage, deployed applications
  - limited control of select networking components
- No control as consumer:
  - underlying cloud infrastructure

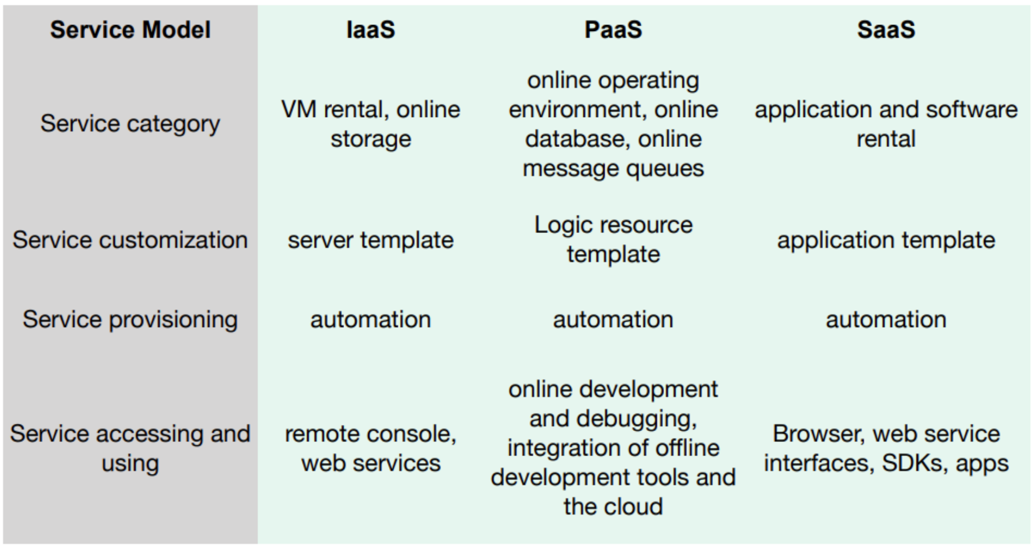
1.1.2 PaaS: Platform as a Service

- Offering: application infrastructure services(eg: development platforms, libraries, tools, databases) through client interface
- Rights as consumer:
  - limited user-specific application configuration settings
- No control as consumer:
  - underlying cloud infrastructure
  - network, servers, storage, OS
  - individual application capabilities
- Example: MS Azure, Amazon FaaS, Google application engine

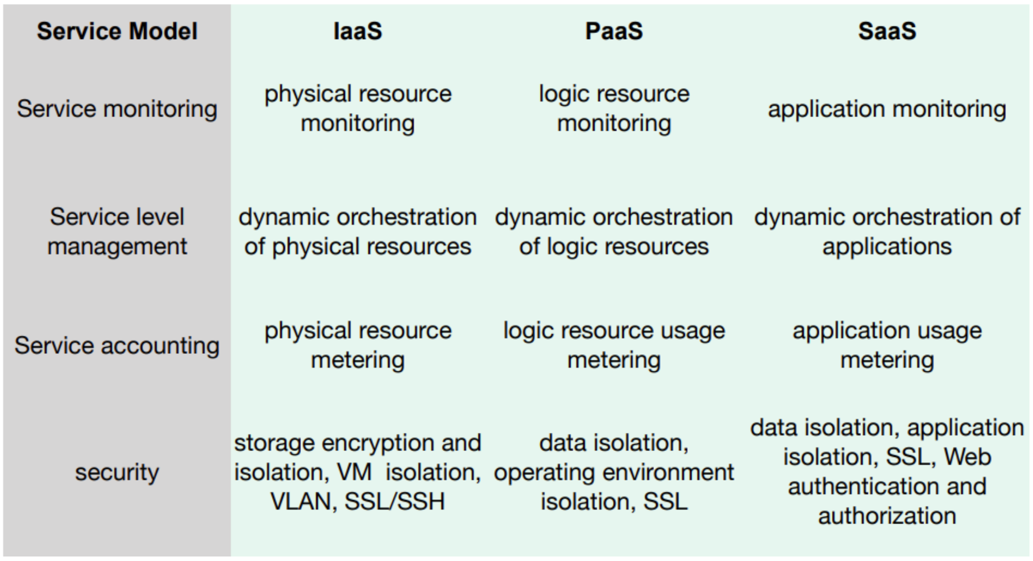
1.1.3 SaaS: Software as a Service

- Offering: provider's applications on cloud through client interface
- Rights as consumer:
  - limited user-specific application configuration settings
- No control as consumer:
  - underlying cloud infrastructure
  - network, servers, OS, storage
  - individual application capabilities





Service Model	**IaaS**	**PaaS**	**SaaS**
Service category	VM rental, online storage	online operating environment, online database, online message queues	application and software rental
Service customization	server template	Logic resource template	application template
Service provisioning	automation	automation	automation
Service accessing and using	remote console, web services	online development and debugging, integration of offline development tools and the cloud	Browser, web service interfaces, SDKs, apps



Service Model	**IaaS**	**PaaS**	**SaaS**
Service monitoring	physical resource monitoring	logic resource monitoring	application monitoring
Service level management	dynamic orchestration of physical resources	dynamic orchestration of logic resources	dynamic orchestration of applications
Service accounting	physical resource metering	logic resource usage metering	application usage metering
security	storage encryption and isolation, VM isolation, VLAN, SSL/SSH	data isolation, operating environment isolation, SSL	data isolation, application isolation, SSL, Web authentication and authorization

1.2 4 Deployment Models: Private, Community, Public, Hybrid

- Private Cloud:
  - service offered **via private network** for **single client**.
- Community Cloud:
  - service offered to **a specific group of clients**.
- Public Cloud:
  - service offered **over Internet via Web-application** or third-party provider for **everyone**.
- Hybrid Cloud: combination of public and private cloud.

1.3 5 Essential Characteristics

- **on-demand self-service**:
  - able to **provision computing capabilities** unilaterally(no interaction required with provider).
- **broad network access**:
  - capabilities can be available and accessed through by **diversely thin or thick client platforms** (mobile, tablets, cable, etc.)
- **resource pooling**:
  - **multi-tenant model** is used, multiple customers shares the computing capabilities at the same time, according to their self-customized demand. Specification of resource location can be possible at higher abstraction level.
- **rapid elasticity**:
  - computing capabilities can be **elastically provisioned and released** in any quantity at any time. The process can be automated or scaled according to dynamic demand.
- **measured service**:
  - automatically control and optimize resource use by **leveraging a metering capability**. Resource usage can be monitored, controlled and reported.

1.4 Pros & Cons of Clouds

- Advantages:
  - scalability, elasticity
  - rapid deployment
  - no capital investment for physical resources
  - outsourcing of infrastructure management
  - limited access to on-premise servers
  - fault tolerance: multiple servers have data replicas, if one node fails, other nodes will replace.
  - collaboration
- Disadvantages:
  - no control over security, based on ”trust”.
  - no control over hardware/infrastructure
  - vendor lockin: service is not standardized, not compatible to other vendors.
  - cost on monthly fees: if demand for same computational power is constant, fee may be higher than building own hardware. Only recommendable for dynamic demand.
  - breaking SLAs: your performance may be influenced by other tenants(multi-tenant model).

2 Base Technologies of Clouds

2.1 Process Technology

Production the processors are produced from semi-conductor materials. It's primarily produced on a **waver** consisting of a lot of chips. Later the waver is cut after the production process. Individual chips will be packaged into the system.

Transistor

- traditional 2D planar transistor: a 2D-planar structure, the gate controls how much current flows from source through the drain.
- 3D tri-gate transistor **FinFET**: conducting channels on 3 sides with a **vertical fin structure**. The width of a Fin is **10nm**, and it keeps shrinking.

The smaller the structure gets, the more transistors fit on the same space, the faster the transistor gets.

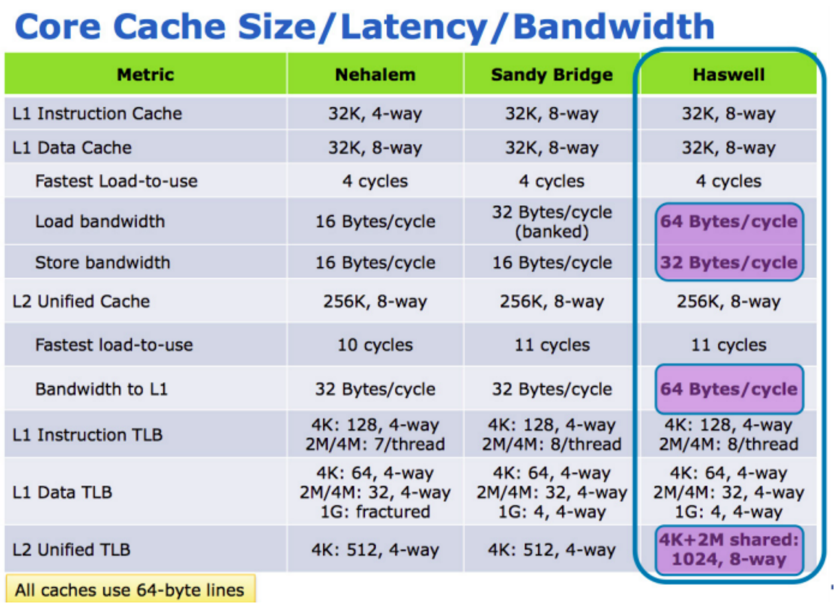
2.2 Processor Architecture

2.2.1 CPU

- current state of CPU development:
  - increase in transistors: up to 10nm or even 5nm.
  - stop in increase of clock speed: up to 4GHz. Limitation: cooling. (the faster the clock speed, the more energy consumed, the hotter)
  - continuous need of performance improvement: parallelism on the chip, since halt in clock speed.
- trends in CPU development:
  - multi-core processors: parallelism
  - SIMD support (Single Instruction, Multiple Data): parallelism inside of a single instruction, computation of vectors of values in parallel.
  - combination of core private and shared caches:
    - data saved in cache for repeated operations
    - with multiple caches, cores can communicate. However, this may disturb the usage of cache.
  - hardware support for energy control: **dynamic voltage and frequency scaling**
    - chips work in a dynamic frequency controlled by hardware according to the need of the running software.
    - It checks whether operations are memory-bound or compute-bound.
  - 64-bit architectures
- challenge: the **memory hierarchy**

- access to the main memory is slow, involving several hundreds of cycles for CPU, while each of these cycles can be responsible for multiple operations.

  → save such cycles for accessing data but keep the **data near to execution**.
- **Level-1 Instruction & Data Cache**: fastest, but too slow to feed all data in large size. (64 Bytes: 8 double precision values, 64 bit long values)
- **Level-2 Unified Cache**: not separated, access better.
- **Translation Lookaside Buffer (TLB)**: translates virtual addresses into physical addresses and loads into main memory, it only stores the **most recent translation**, no need to lookup constantly.

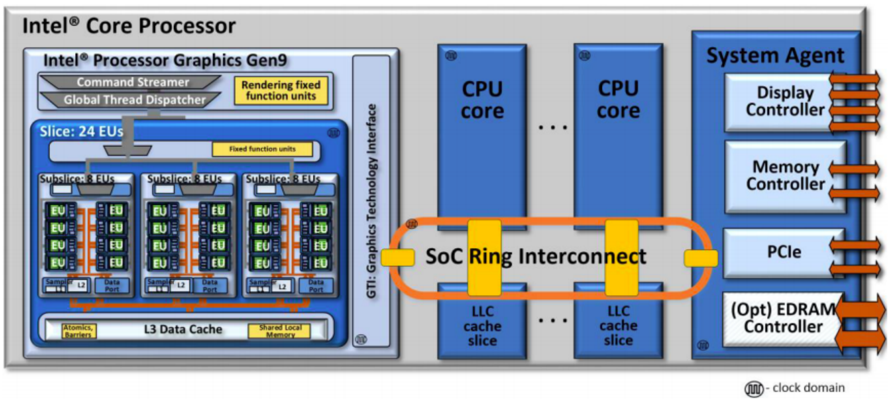


Core Cache Size/Latency/Bandwidth

Metric	Nehalem	Sandy Bridge	Haswell
L1 Instruction Cache	32K, 4-way	32K, 8-way	32K, 8-way
L1 Data Cache	32K, 8-way	32K, 8-way	32K, 8-way
Fastest Load-to-use	4 cycles	4 cycles	4 cycles
Load bandwidth	16 Bytes/cycle	32 Bytes/cycle (banked)	64 Bytes/cycle
Store bandwidth	16 Bytes/cycle	16 Bytes/cycle	32 Bytes/cycle
L2 Unified Cache	256K, 8-way	256K, 8-way	256K, 8-way
Fastest load-to-use	10 cycles	11 cycles	11 cycles
Bandwidth to L1	32 Bytes/cycle	32 Bytes/cycle	64 Bytes/cycle
L1 Instruction TLB	4K: 128, 4-way 2M/4M: 7/thread	4K: 128, 4-way 2M/4M: 8/thread	4K: 128, 4-way 2M/4M: 8/thread
L1 Data TLB	4K: 64, 4-way 2M/4M: 32, 4-way 1G: fractured	4K: 64, 4-way 2M/4M: 32, 4-way 1G: 4, 4-way	4K: 64, 4-way 2M/4M: 32, 4-way 1G: 4, 4-way
L2 Unified TLB	4K: 512, 4-way	4K: 512, 4-way	4K+2M shared: 1024, 8-way

All caches use 64-byte lines

2.2.2 Skylake Architecture



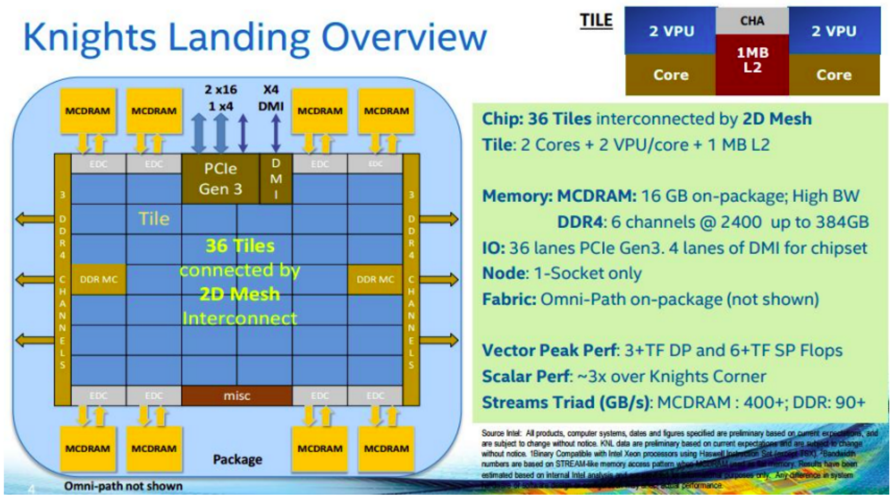
The archiecture of a processor (one chip)

- graphic processor: accelerator for specified computation

- system agent: support structures (eg: display controller, memory controller, PCIe for I/O, EDRAM controller)
- CPU cores: homogeneous cores with a **private cache each**.
- LLC cache slice: each slice cache is associated to each CPU core.
  - If core misses information in its private cache: through interconnect, it checks which slice cache contains the info, then it propagates to the cache associated the CPU core and returns the info back to the CPU.
- SoC Ring interconnect: all parts are connected by a ring bust.
  - if CPU writes to I/O device – PCIe, it first puts information onto the bust, then it propagates into the PCIe and is written to the disk.

→ the **interconnect ring** is the **bottleneck** for increasing the cores.
→ alternatives: Xeon Phi

2.2.3 Xeon Phi Architecture



Chip: 36 Tiles interconnected by 2D Mesh
Tile: 2 Cores + 2 VPU/core + 1 MB L2

Memory: MCDRAM: 16 GB on-package; High BW
DDR4: 6 channels @ 2400 up to 384GB
IO: 36 lanes PCIe Gen3. 4 lanes of DMI for chipset
Node: 1-Socket only
Fabric: Omni-Path on-package (not shown)

Vector Peak Perf: 3+TF DP and 6+TF SP Flops
Scalar Perf: ~3x over Knights Corner
Streams Triad (GB/s): MCDRAM : 400+; DDR: 90+

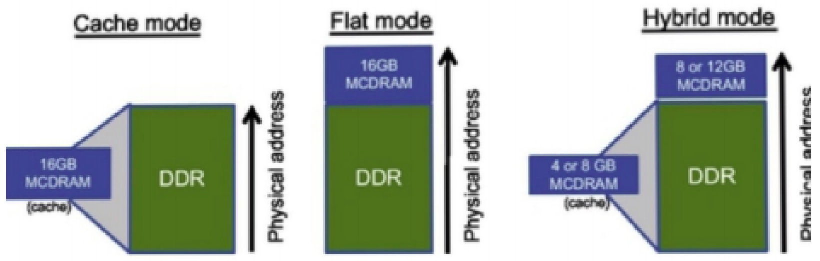
- Goal: allows significantly **more cores** in a single processor, in a single CPU die.
- Idea:
  - the die is organized into a **tile-architecture**.
  - 36 compute tiles are connected through a **2D mesh network** → connection between tiles in both x- and y-direction.
  - each tile has 2 cores → **72 cores** in total
- Tile structure:
  - 2 cores
  - 2 VPUs (Vector Processing Unit) for each core → 4 in total per tile.

- L2 cache: shared between the cores, but as a private cache for each tile.

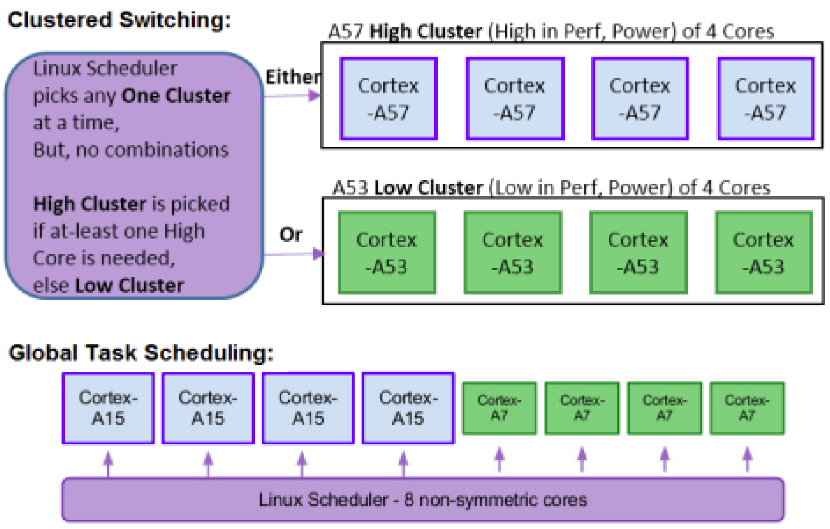
  → multiple copies of an address can be in different private L2 caches of different tiles, which **must be coherent**.
- CHA (Caching Hold Agent): responsible for the **coherence**. It's connected to each tile, keeping track of the status of the copies by implementing a **coherence protocol**.

- Memory:
  - DDR4 memory
  - **MCDRAM**– Multi-channel DRAM, **high bandwidth**(450 GB/s)

    Memory modes:
    * flat mode: all MCDRAM is used as **physical address**. Data structure can explicitly choose between MCDRAM or DDR.
    * cache mode: all MCDARM is used as **L3 cache**. physical addresses only on DDR. If data is being processed from DDR, it's mapped to MCDRAM.
    * hybrid mode: combination of flat and cache mode. Part of MCDRAM is used as **L3 cache**, the other as **physical addresses**.



2.2.4 Processors for mobile devices: ARM


Clustered Switching:
Linux Scheduler picks any One Cluster at a time, But, no combinations
High Cluster is picked if at-least one High Core is needed, else Low Cluster
Either
Or
A57 High Cluster (High in Perf, Power) of 4 Cores
Cortex -A57
Cortex -A57
Cortex -A57
Cortex -A57
A53 Low Cluster (Low in Perf, Power) of 4 Cores
Cortex -A53
Cortex -A53
Cortex -A53
Cortex -A53
Global Task Scheduling:
Cortex-A15
Cortex-A15
Cortex-A15
Cortex-A15
Cortex-A7
Cortex-A7
Cortex-A7
Cortex-A7
Linux Scheduler - 8 non-symmetric cores

- **Big Little Principle**
  - Combination of **high clusters** and **low clusters**, controlled by clustered switching.
    - high cluster: high in performance
    - low cluster: low in performance, but energy efficient
    - only **one cluster** at a time, **no combinations**.
  - Global task scheduling: tasks are scheduled according to the requirement between 2 clusters.
- Use-cases: Apple Processor A14 (2 high performance Firestorm, 2 energy-efficient Icestorm)

2.3 Accelerator Programming

- Motivation:
  - increase computational speed and reduce energy consumption

    → achieved by **specialization** in operations/on-chip communication/memory accesses

    → **accelerator**
- Types:
  - GPGPU (General Purpose Graphic Processors)
  - FPGA
  - standard cores
- Designs:
  - CPU with accelerators attached: computation can be offloaded onto the accelerator.
  - accelerators-only design
  - accelerator booster: a collection of accelerators as a separate part from the whole system. Jobs can be computed by these accelerators when necessary. Accelerator booster can be shared among parallel jobs.

2.3.1 Graphic Processing Units (GPU)

- Usage:
  - visualization
  - **general processing** (NVIDIA)
- Parallelism: multi-threading, MIMD, SIMD
- Challenges:

  - a **specialized programming interface** for the GPGPU needed (eg: CUDA from NVIDIA)
  - **scheduling coordination** on system processor and GPU
  - **transfer of data** between system memory and GPU memory
- example: NVIDIA Tesla P100

NVIDIA Tesla P100

- GP100 (GPU):
  - L2 cache: shared among all compute units – streaming multi-processor
  - NVLink: able to connect multiple GPGPUs together
  - memory controller: access to high bandwidth memory
  - 6 Graphic Processing Clusters(GPC)
    - 10 Streaming Multi-Processor each GPC, 60 in total
    - 5 Textural Processing Clusters (TPC), 1 for 2 SM, 30 in total
- High Bandwidth Memory (HBM)
  - **vertical stacks** of memory dies connected by microscopic wires
    → near and tight connection between memories
  - 180 GB/s per stack bandwidth

→ good for data parallel processing like vector processing.

2.3.2 Field Programmable Gate Arrays (FPGA)

- hardware which is programmable
- Consist of:
  - **array of logic gates** to implement **hardware-programmed special functions**
  - **specialized functional units** (eg: signal processors, multipliers)
  - static **memory**
- programmed in VHDL: the program describe functions to be executed, it will then be translated into look-up tables, which are put into the logic gates
- Use-case:
  - as **accelerator** for specialized computations
  - **filtering** for databases
  - as **switches, routers** for communication
  - **preprocessing**: I/O hardware of FPGA accesses the DDR, local computation can be organized in the pipeline of FPGA. Local preprocessing will be done on FPGA and the output will be sent ot external processor via PCIe.
- examples: Altera, Xilinx

2.4 Architecture for Parallelism: Shared Memory Systems

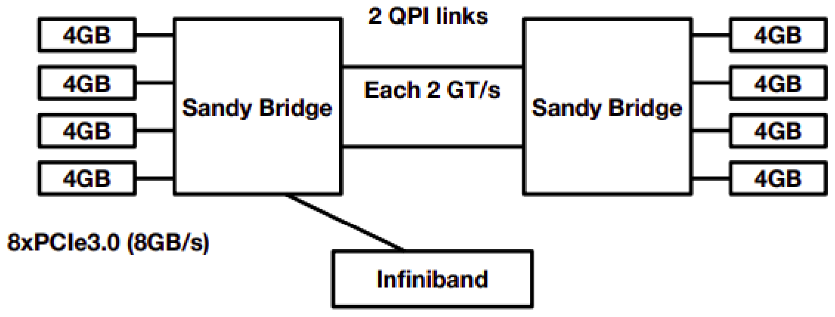
- Idea: 2 architectures to **achieve parallelism**, which is combining multiple processors together for computation.
  - shared memory system
  - distributed memory system
- **Non-Uniform Memory Access (NUMA)**:
  - **multiple CPU** with multiple cores are connected through **single physical address** space.

    → access time depends on the **distance to the physical address** (memory)

    → CPU accesses either local(near) or remote memory → locate the memory near for fast access.

Example: SuperMUC

- 2 multi-core processors(Sandy Bridge) with 32GB memory in total
- each processor can access to all 32GB memory, **however access time differs**
- **Latency**
  - local: ∼50ns, ∼135 cycles
  - remote: ∼90ns, ∼240 cycles



- Programming interfaces for Shared Memory Systems
  - explicit threading
  - automatic parallelization: sequential code is given to compiler, which **automatically** parallelize the work among available CPUs.
  - OpenMP: directive-based parallel programming, parallel computations are **explicitly expressed**.
- Challenges in parallel computing:
  - explicit synchronization needed.
  - **cocurrency bugs**: the outcome of the computation depends on the speed of access to the memory → non-deterministic results possible.
  - control of **data locality**

2.5 Architecture for Parallelism: Distributed Memory Systems

- Characteristics
  - **Coupling** of individual nodes via network: processor only have access to the memory in node.
  - **no shared** physical address space
  - communication between nodes: transfer of **messages**
- Programming in Distributed Memory Systems: **more difficult** than shared memory systems.
  - + : **rare race conditions** → cocurrency bugs low.
  - – : Message Passing Interface(MPI), have to explicitly decompose or insert message passing → more difficult to program.
  - Process to Process communication and collective operations
- Challenges:
  - **more difficult** to programm than shared memory systems
  - **expensive communication**, much slower than access to memory.

$$t(message) = \text{startup time} + \frac{\text{message size}}{\text{bandwidth}}$$

    - communication with one large message is more efficient than multiple small messages (startup time)
    - mapping onto processors has performance impact. Communication may not need to go through the entire but locally. (eg: 2D mesh network)