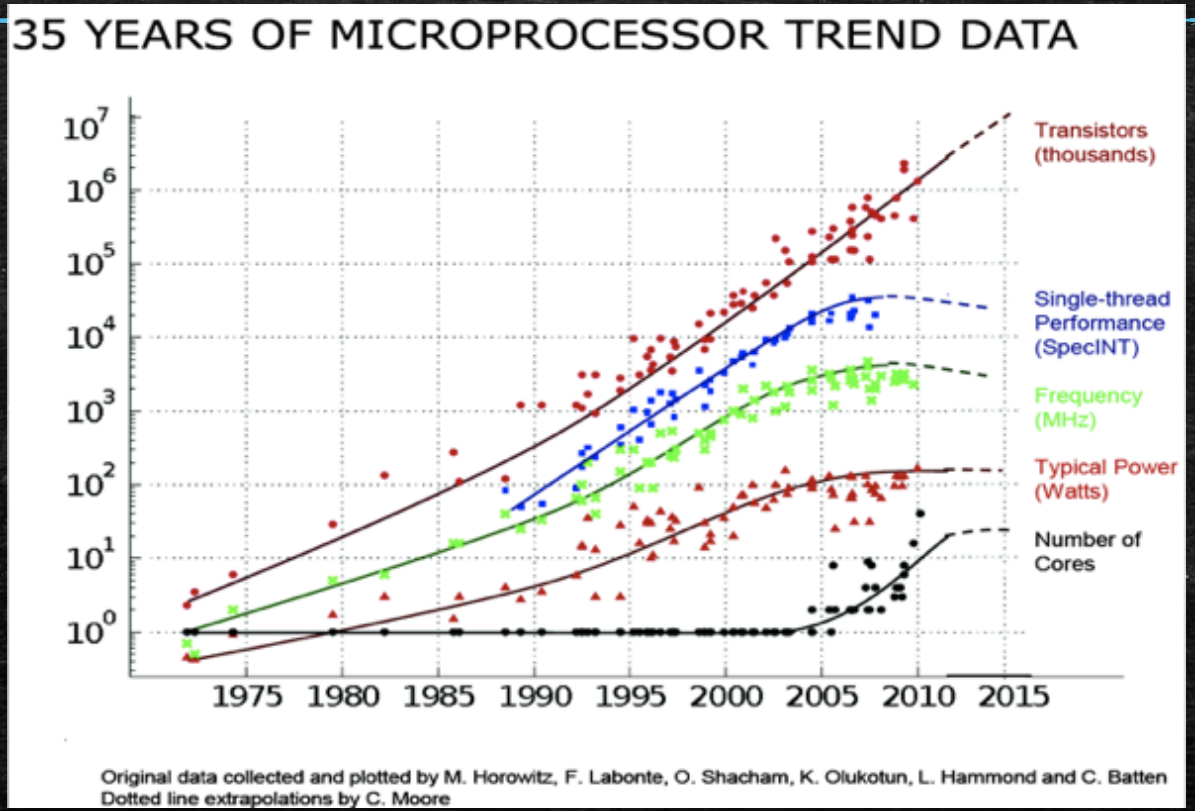
Introduction to Parallel computing architectures

ADACS HPC workshop 2017

Why bother knowing all this?



Source:
www.karlrupp.net

Learning objectives

- Some common HPC terms
- Understanding parallelism
  - Amdahl's Law
  - Scalability
- Types of Parallelism
- Levels of Parallelism and relevant architecture overview
  - Processor Level parallelism
  - Node level parallelism
  - Cluster level parallelism
- Putting it all together -- An abstract supercomputer

Some parallel semantics

Parallel computing: Solving a computational problem by simultaneous use of multiple computing resources.

Parallel computer: A platform with multiple processing elements working as a unified system/compute resource.

FLOPs: FLoating Point Operations per seconds are standard metric to compare performance (speed) of computing resource. With modern day architecture the usual metric is MFLOP or Million FLoating Point Operations per second.

High Performance computing (HPC): Aggregating computing power in a way that either

- Reduces time to solution
- Enables solving large computational problem

Some definitions ….

Supercomputer: A computer at the frontline of the current processing capacity, particularly the speed of calculation in FLOPs.

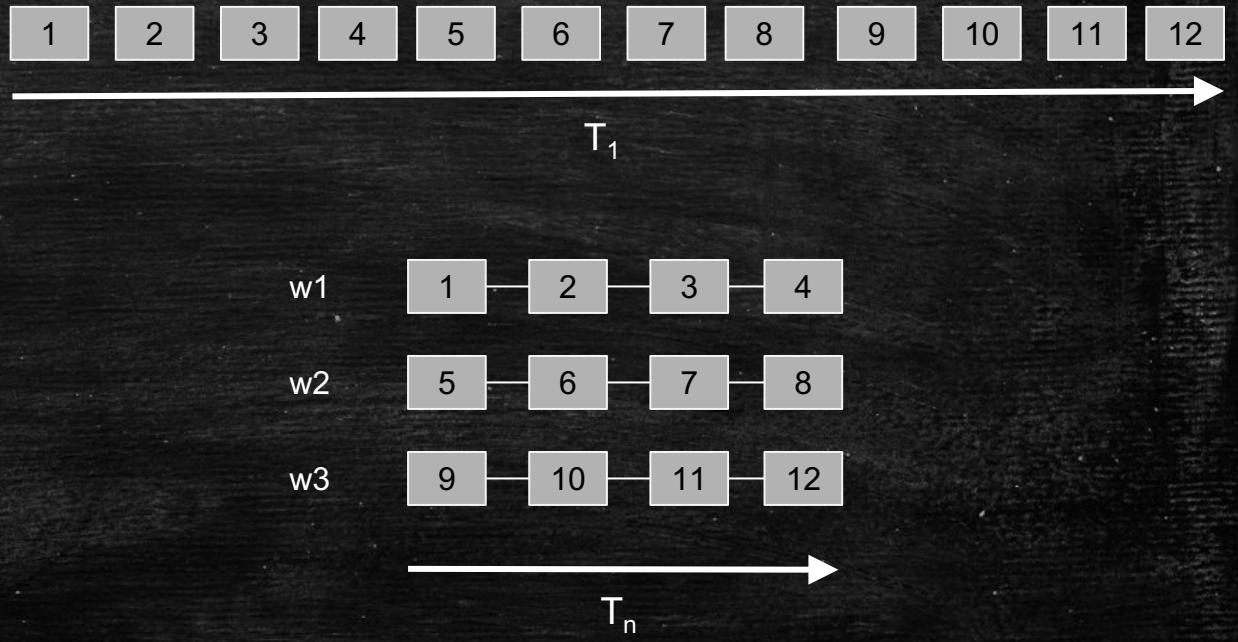
Speedup: of a parallelized code = $\frac{\textit{Wall time of serial run}}{\textit{Wall time of parallel run}}$

Cache: A hardware component or software concept used to store the frequently used data for reuse. A **cache line** is block of data transferred between main memory and CPU cache.

Interconnect: A physical link or network of links between compute devices to transfer data at a time critical manner.

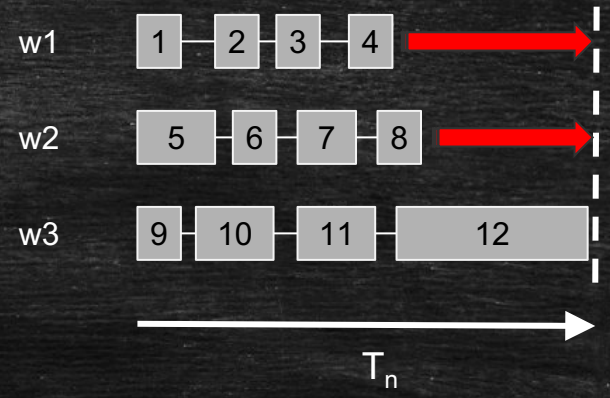
Parallelism

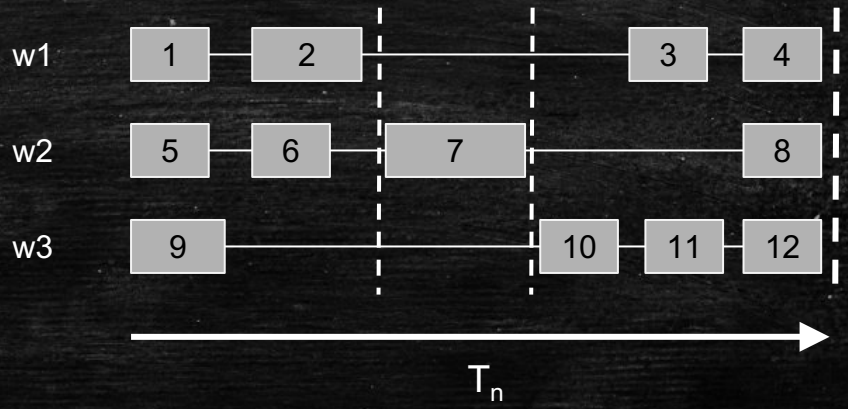
- Can a program composed of a set of tasks be broken into tasks those can executed in parallel on multiple compute units?
- Ideally the time should reduce by the number of available compute units, right?



Parallelism
contd ...

- But what if the tasks have uneven workload? i.e load imbalance
- Or what if there are were dependencies to be resolved?
- Overhead e.g. synchronization between tasks can also take non trivial amount of time will can be a function of number of workers





Amdahl’s Law

It describes the dependence of the achievable speedup of a program execution to the parallel fraction of the program.

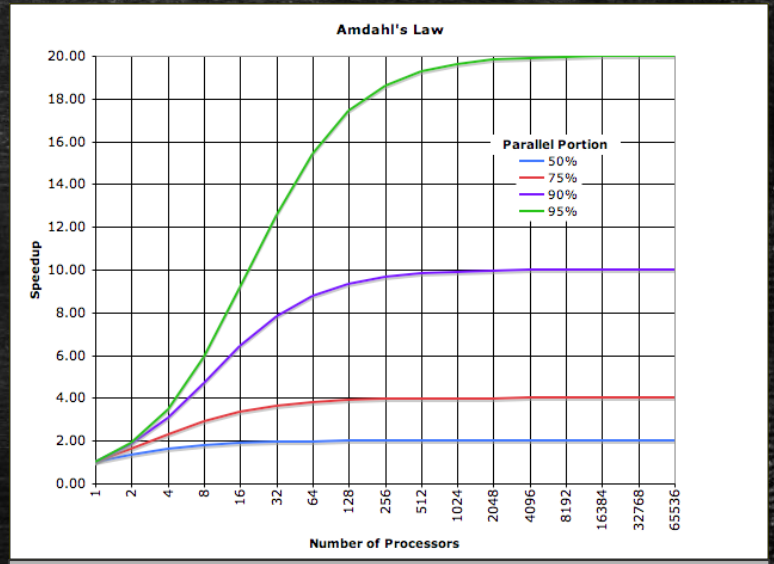
$$S = \frac{1}{(1-P)+\frac{P}{N}}$$

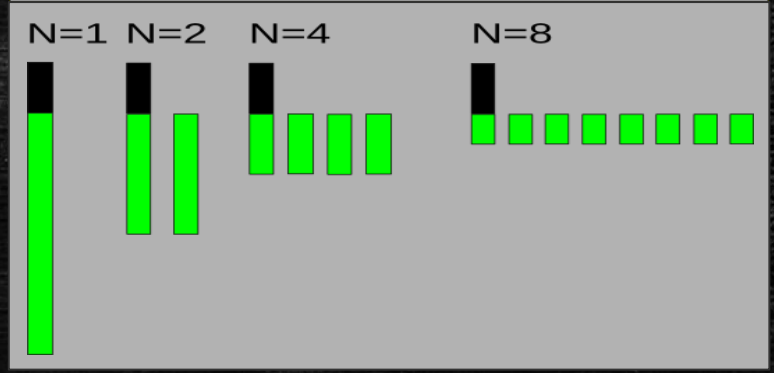
S is speedup

P is the parallel fraction of the program

So **(1-*P*)** is parallel fraction of program

N is number of worker





Parallelism contd…

Parallel Scalability:

It is a characteristic of a parallel program which helps measure how efficiently it runs with the increasing parallel resource.

Scaling tests help decide the optimum parallel resource required to run a parallel program efficiently.

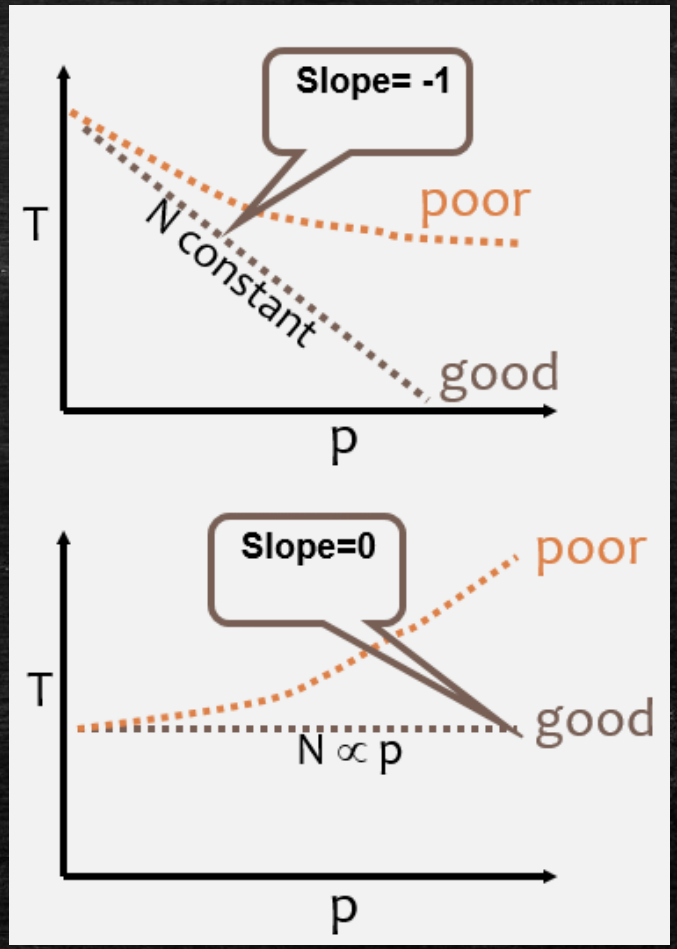
Parallelism contd…

Strong scaling: Keeping the problem size **(N)** same, how many workers can run the application efficiently?

Weak scaling: Keeping the problem size **(N)** per worker the same, how big a problem size can be simulated efficiently if we had unlimited resource?

Parallel Efficiency:

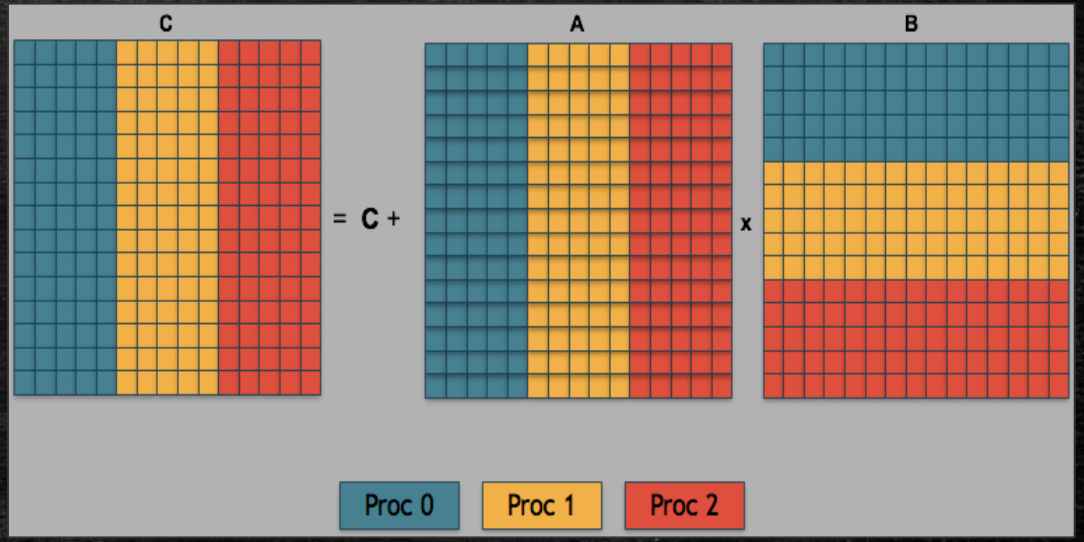
$$Efficiency = \frac{S}{p}$$



Source: https://computing.llnl.gov/tutorials/parallel_comp/

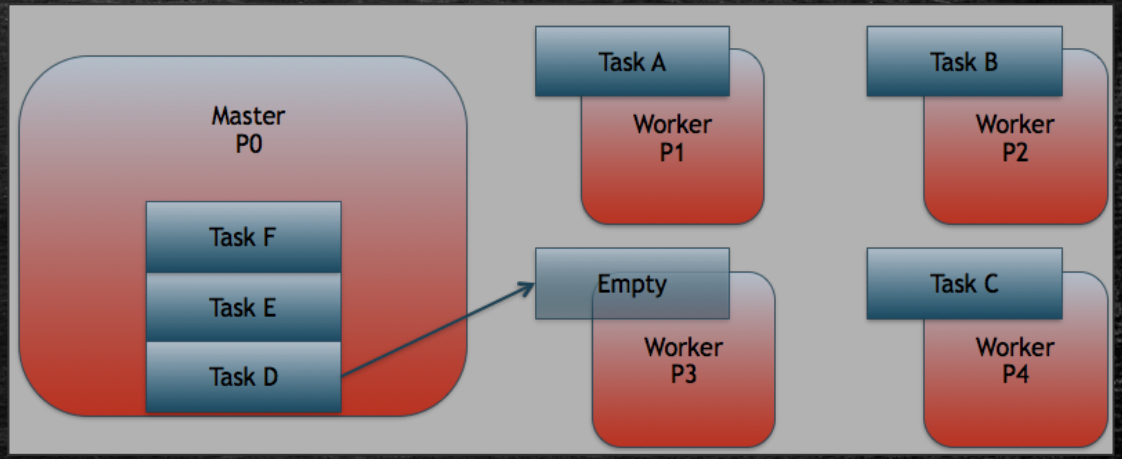
Types of Parallelism

Data Parallelism: (SPMD)
Data can be decomposed so each available worker can work on its domain.



Types of Parallelism contd …

Task Parallelism: (MPMD)
Individual or groups of tasks can execute a set of instructions (functions or subroutines) on identical or different data.



- granularity of decomposition is important
- each worker needs to have enough work to keep them busy
- scaling tests can help identify the sweet spot.

Levels of parallelism

- Processor level parallelism
  - Instruction Level Parallelism
    - On hardware level : OS controls it so is dynamic
    - On software level : compilers implement it so is static
  - Vector units and SIMD (Single Instruction Multiple Data)
  - Simultaneous Multithreading (SMT) or Hyperthreading
- Node level Parallelism
  - Multicore + Multisocket systems
  - NUMAness
  - Shared memory systems
- Cluster level parallelism
  - Distributed memory systems

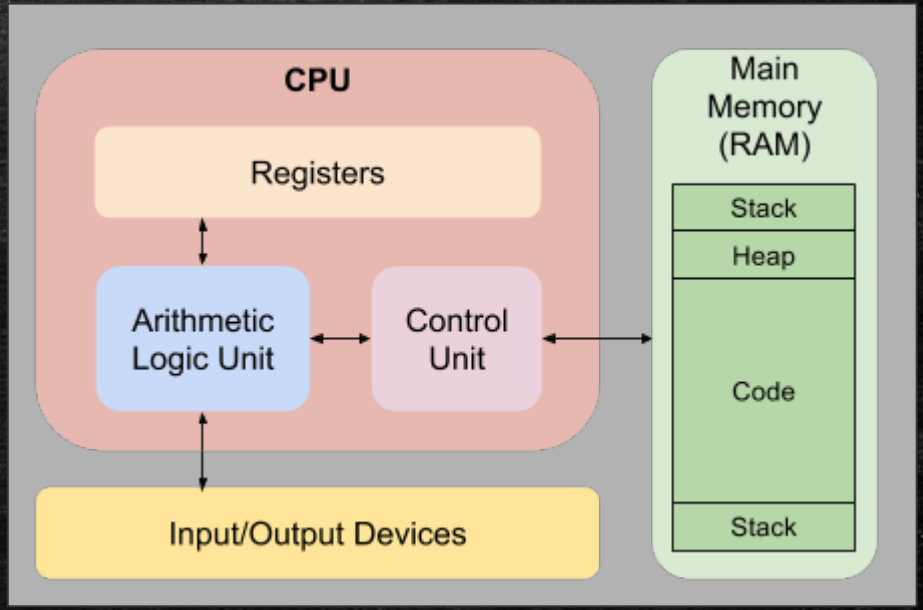
A program can be written to express parallelism at all levels to achieve performance gains at different levels.

Processor Level Parallelism

Basic Von Neumann machine

Characterised as a sequential machine

- **Program Control Unit** interprets and causes execution of instructions (timing, control signals for instruction and data flow)
- **Arithmetic/Logic Unit** performs operations on data
- **Main Memory** stores data and instructions
- **Input Output** allows transfer of data in and out of the machine



Instruction Level parallelism (ILP)

- Multiple instruction from the same instruction stream can be executed simultaneously.

- Implemented in hardware so the OS can make use of it , e.g.
  - Instruction pipelining -- assembly line of Instructions
  - Superscalar execution -- dispatches instructions simultaneously to multiple functional units
  - Out-of-order execution -- reschedules instruction execution based on data dependency to avoid CPU stall time.

Instruction Level parallelism (ILP) contd ...

- Can further be exploited by making code more conducive to hardware
  - Automatic parallelism
    - Compiler optimizations e.g. loop unroll, SIMD
  - Program directives to aide compilers making the most of vector units e.g. AVX
  - Example of loop optimization by compiler

Original Loop:

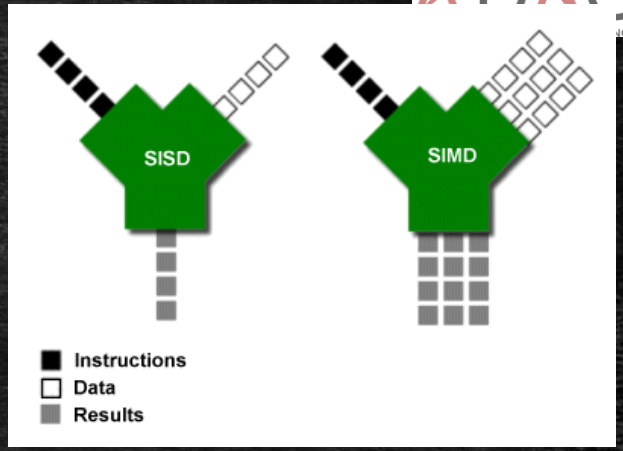
```
do j = 1,100
  do i = 1,4
    a(j,i) = b(j,i) + c(j,i)
  end do
end do
```

Unrolled loop:

```
do j = 1,100
        a(j,1) = b(j,1) +
c(j,1)
    a(j,2) = b(j,2) + c(j,2)
    a(j,3) = b(j,3) + c(j,3)
    a(j,4) = b(j,4) + c(j,4)
end do
```

Instruction Level parallelism (ILP) contd ...

- Single Instruction Multiple Data or SIMD exploits the specialized vector units in modern processors. It can be achieved either by using compiler flags or embedding directives in the code.



Solving A[i] + B[i] = C[i]

$A_0 + B_0 = C_0$

$A_1 + B_1 = C_1$

$A_2 + B_2 = C_2$

$A_3 + B_3 = C_3$

$\begin{Bmatrix} A_0 \\ A_1 \\ A_2 \\ A_3 \end{Bmatrix} + \begin{Bmatrix} B_0 \\ B_1 \\ B_2 \\ B_3 \end{Bmatrix} = \begin{Bmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{Bmatrix}$

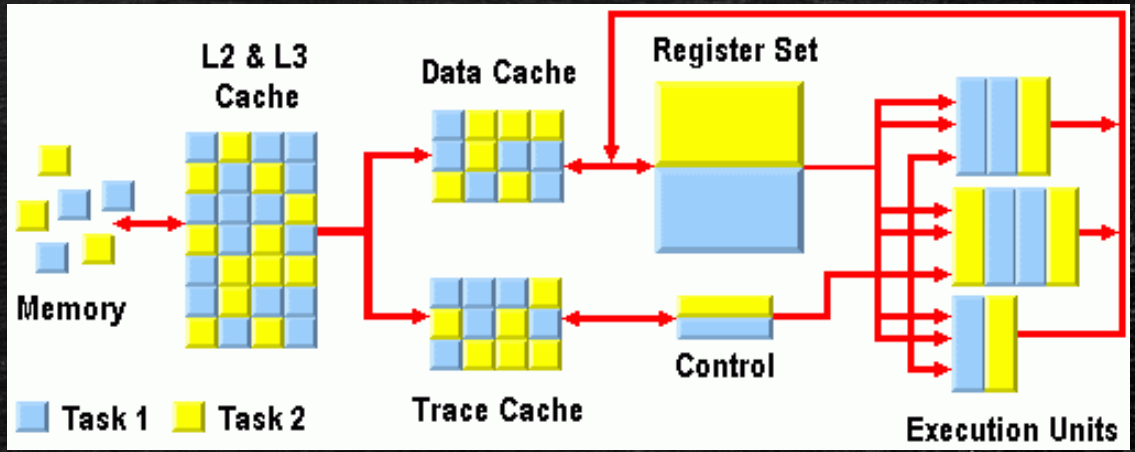
```
for (i=0; i<n; i++)
   C[i] = A[i] + B[i]
```

Supposing vector length = 4

```
for (i=0; i<n; i+=4)
   C[i] = A[i] + B[i]
```

Simultaneous multithreading (Intel's Hyperthreading Technology HT)

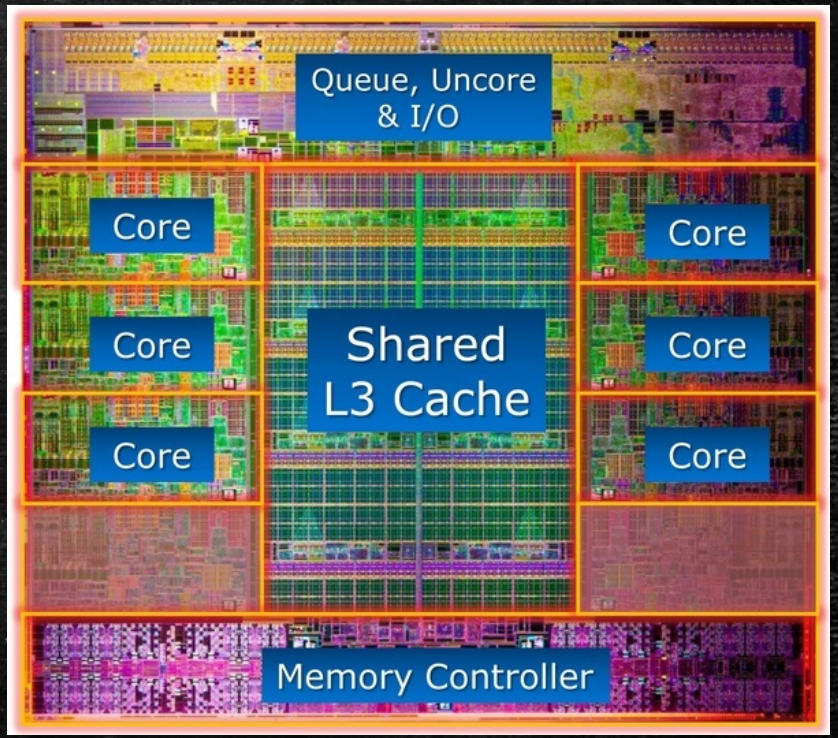
- Simultaneous multithreading (Intel's Hyperthreading Technology) keeps core's resources busy by presenting multiple virtual CPUs per core.



Node Level Parallelism

Node level parallelism -- multicore

- Modern CPU architecture offers multiple cores on a single chip to scale out.
- Power wise it is more economical
  - same clock for all instead of increasing clock frequency for one core
- Each core has dedicated execution resource e.g. registers, ALU, FPU and vector units, L1 & 2 cache (SRAM) etc.
- Last level cache LLC or L3 cache is shared between cores and mitigates latency between main memory (DRAM) and L1,L2.
- Enabling hyperthreading on modern Intel processors is boot time decision.

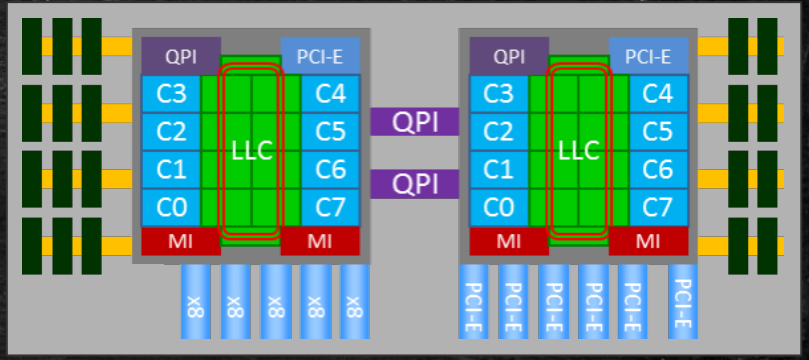


Single socket die of Intel Xeon CORE

Node level parallelism -- multicore + multisocket

Scaling memory with the increasing number of cores on the same silicon die has limitation (e.g. cache coherence overhead increases geometrically with core count)

Solution: multiple sockets or Non Uniform Memory Access nodes on a single board connected via high speed interconnect (vendor specific e.g. Intel's Quick Path Interconnect (QPI) or AMD's HyperTransport).



Node level parallelism -- multicore + multisocket

A shared global and address space presented via a cache hierarchy which is cache coherent.

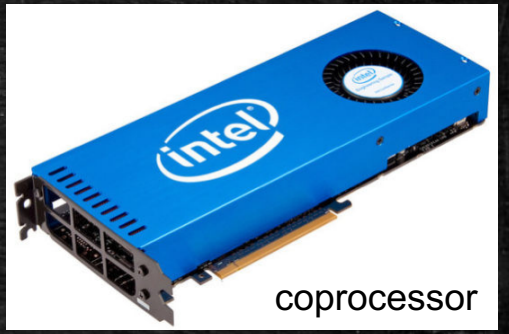
Such node is effectively a **shared memory machine**. Adding more sockets or NUMA nodes can scale up to make a larger shared memory machine. SGI UV2000 is such an example.

Node level parallelism -- Manycore Accelerators and Co-processors

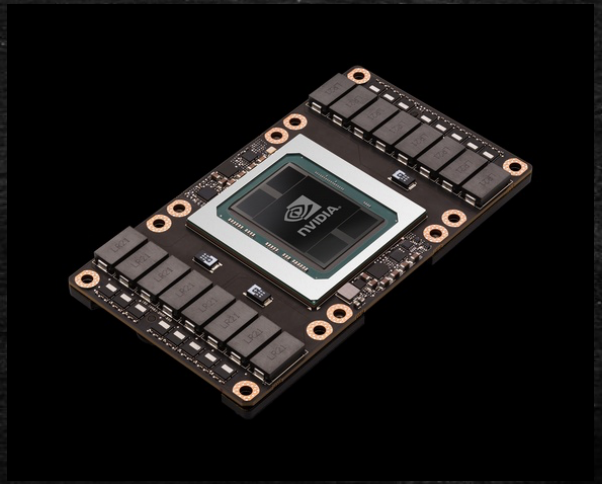
- Rapid advances in semiconductor technology has enabled putting more cores on the same silicon wafer.
- The motivation for these architectures is to
  - Increase node performance
  - Increase power efficiency i.e more FLOPs per Watt
  - Enable parallelism through task specialization (offload model)
- Many Intel Core (MIC) and NVIDIA's and AMD's General Purpose Graphic Processing Units GPGPU have been popular architectures of this kind.

Intel Xeon Phi

stand-alone



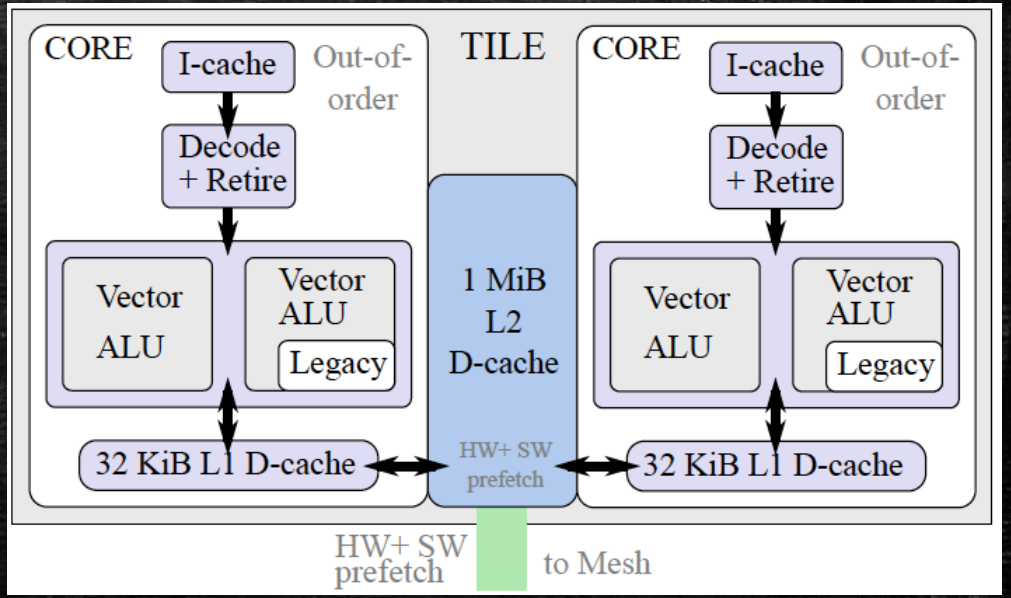
coprocessor



NVIDIA
Tesla P100
with architecture
Pascal GP100

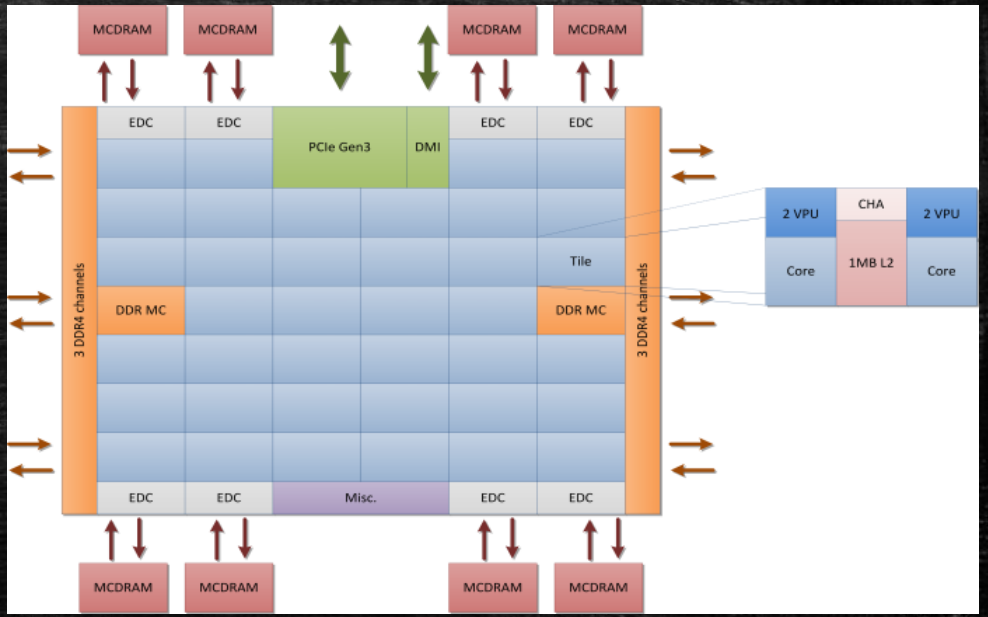
Node level parallelism -- Intel Knight's Landing Gen 2 Xeon Phi

- 4 Hyperthreads per core in single socket
- Dedicated vector units per core
- Out of order execution
- Shared L2 as LLC
- Modest clock frequency=1.30GHz



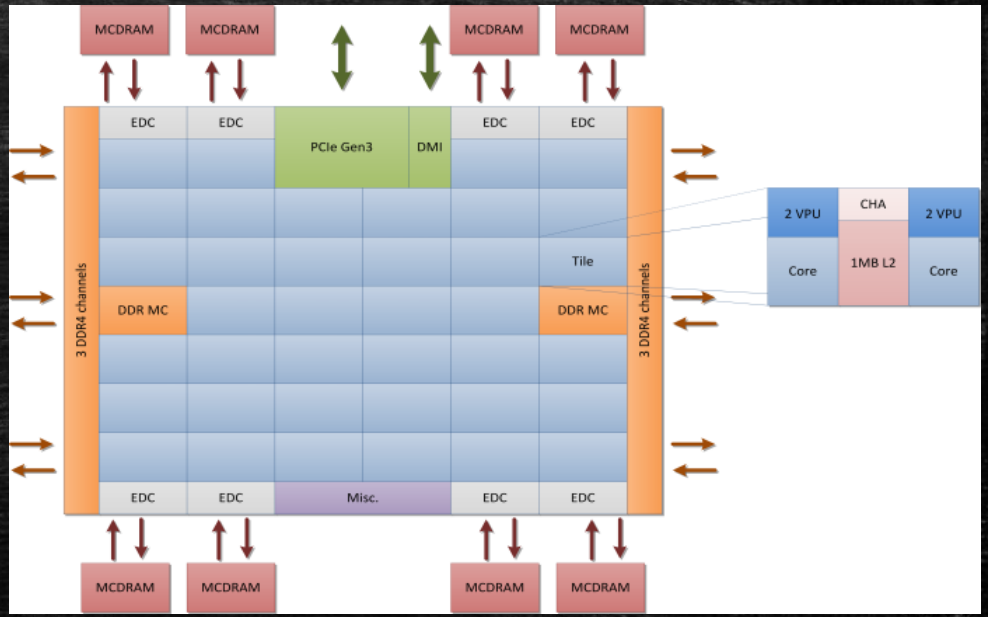
Node level parallelism -- Intel Knight's Landing Gen 2 Xeon Phi

- Higher core density : 64-68 cores
- Two tiers of memory
  - DDR ~ 90GB/s with max size ~ 384GiB
  - MCDRAM or High Bandwidth Memory ~ 400GB/s with max size ~ 16GiB
- MCDRAM is there to keep the cores busy.
- Cores can be clustered into 4 modes (Quadrant mode show in picture).



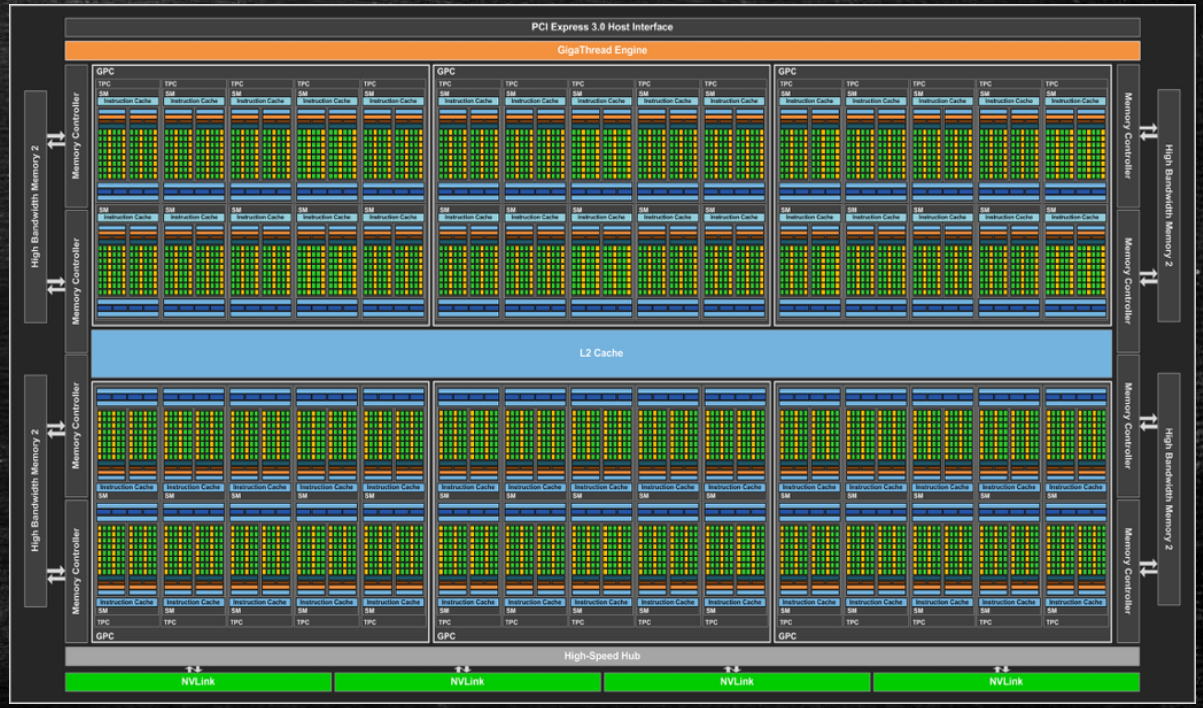
Node level parallelism -- Intel Knight's Landing Gen 2 Xeon Phi

- Codes performant on multicore architecture can run on KNL out of the box
  - May need optimization to tune for performancee
  - **Vectorization is crucial for performance**

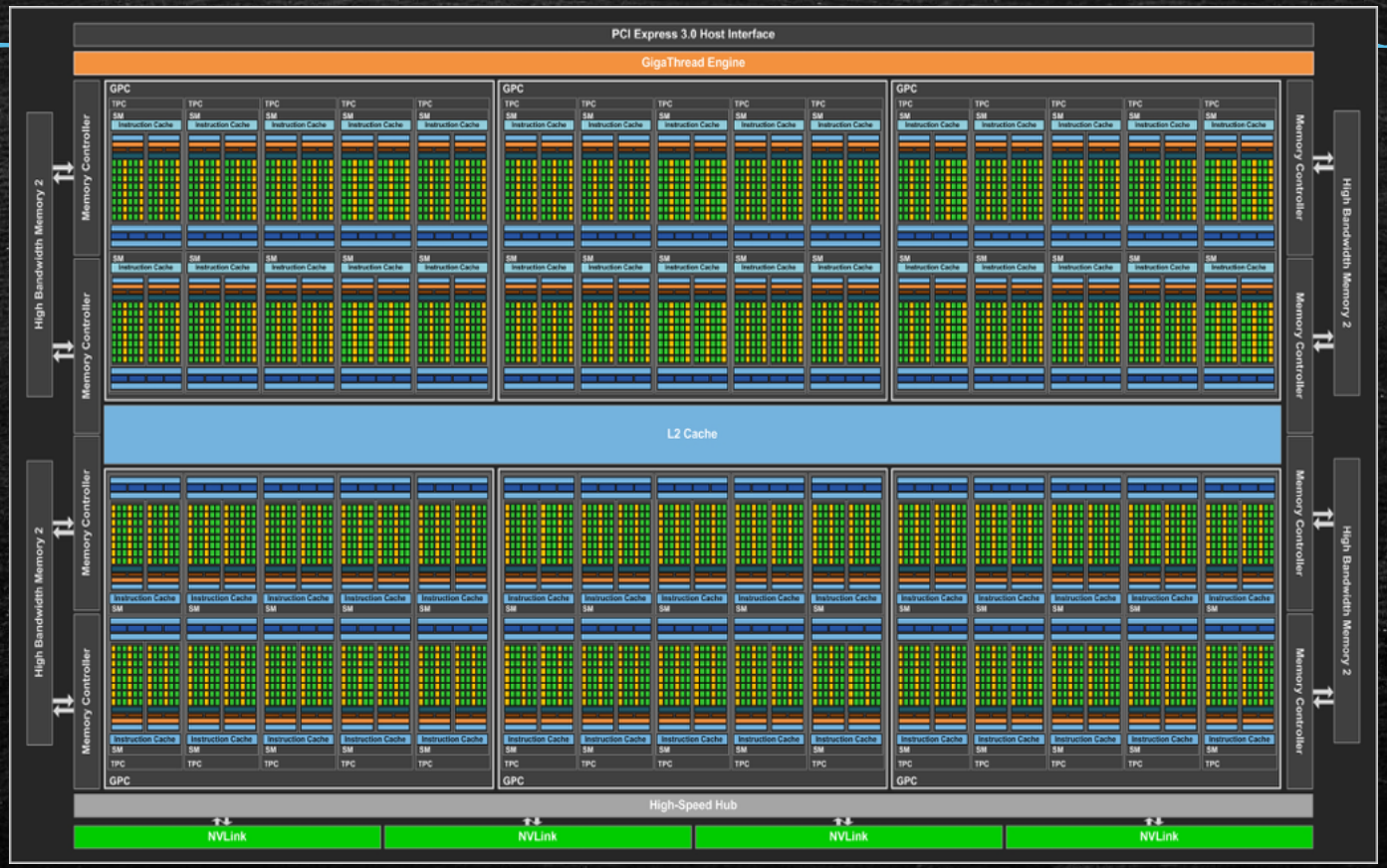


Node level parallelism -- NVIDIA Pascal P100 GPGPU

- Very high density of low performance cores
  - (P100 has 3584 single precision cores on single chip)
  - Very low clock freq= ~1400MHz
  - Meant for fine grained parallelism and high throughput computing
- Host CPU to run OS and offload work or kernels to GPU



Node level parallelism -- NVIDIA Pascal P100 GPGPU

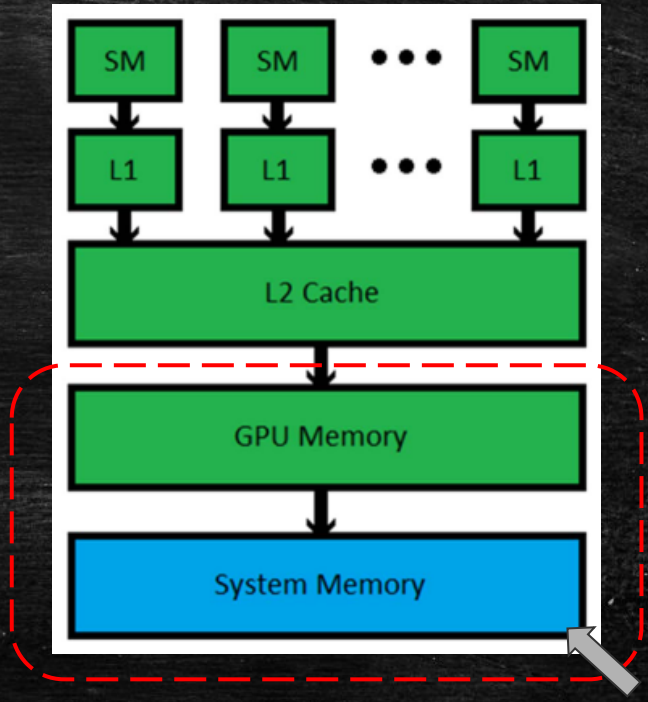


Node level parallelism -- NVIDIA Pascal P100 GPGPU

- Basic execution unit is called Streaming Multiprocessor (SM) -- different models have different SM count
  - 64 cores/SM and 56 SM in total on P100
- Max 64 warps/SM can run on P100
  - in a bunch of 32 lock step threads are called a "warp".
  - A warp gets inactive if waiting for data whereas the other warp becomes active
- Limited registers per thread but aggregate register count per SM is huge compared to CPU architecture.

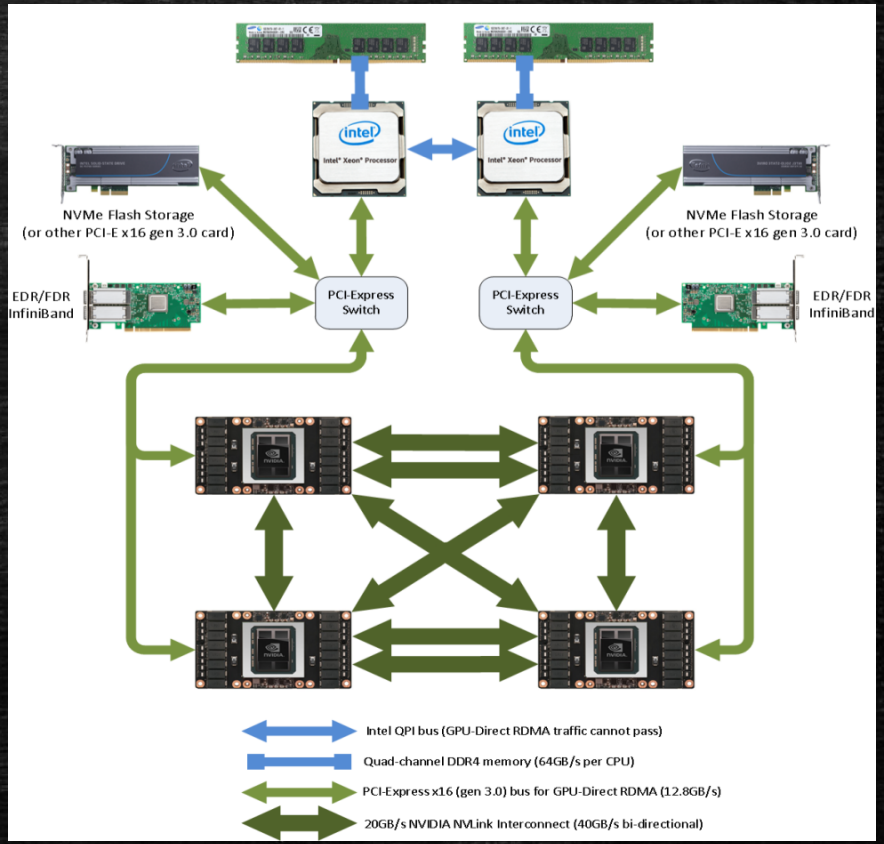
Node level parallelism -- NVIDIA Pascal P100 GPGPU

- L1, warp scheduler and Instruction cache is local to SM
- L2 shared among all SMs
- HBM2 is the local main memory (GDDR5 in previous models)
- Need a special programming interface to program for GPU called CUDA -- Compute Unified Device Architecture
- Unified memory feature of CUDA provides a single virtual address space for accessing all CPU and GPU memory.



Node level parallelism -- GPGPUs e.g. NVIDIA Pascal P100

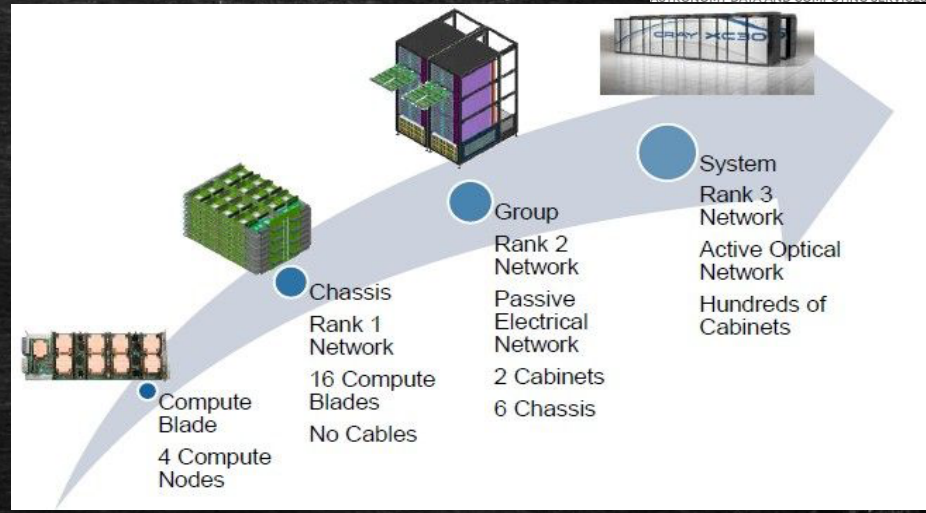
- It is possible to have multiple GPU devices on a single node hosted by a common set of host devices.
- NVLink interconnect is NVIDIA's new advancement.
- GPUs can communicate with each other over this dedicated high-speed link ~20GB/s (40GB/s bidirectional).
- NVIDIA's Collective Communications Library (NCCL) for proficient collective communication.



Cluster Level Parallelism

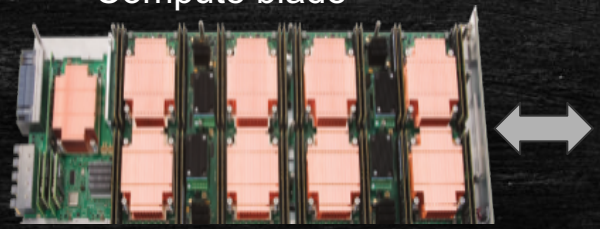
Cluster level parallelism

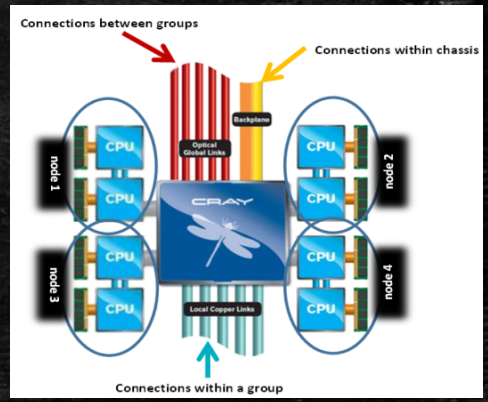
- Multiple multi-socket nodes can be connected via high speed interconnect to scale out to clusters of 1000s of tightly node.
- Memory of a remote node is no longer directly accessible. Data is passed via Message Passing Interface library.
- These are **Distributed memory machines**.



Compute node

Compute blade





Cluster level parallelism contd ...

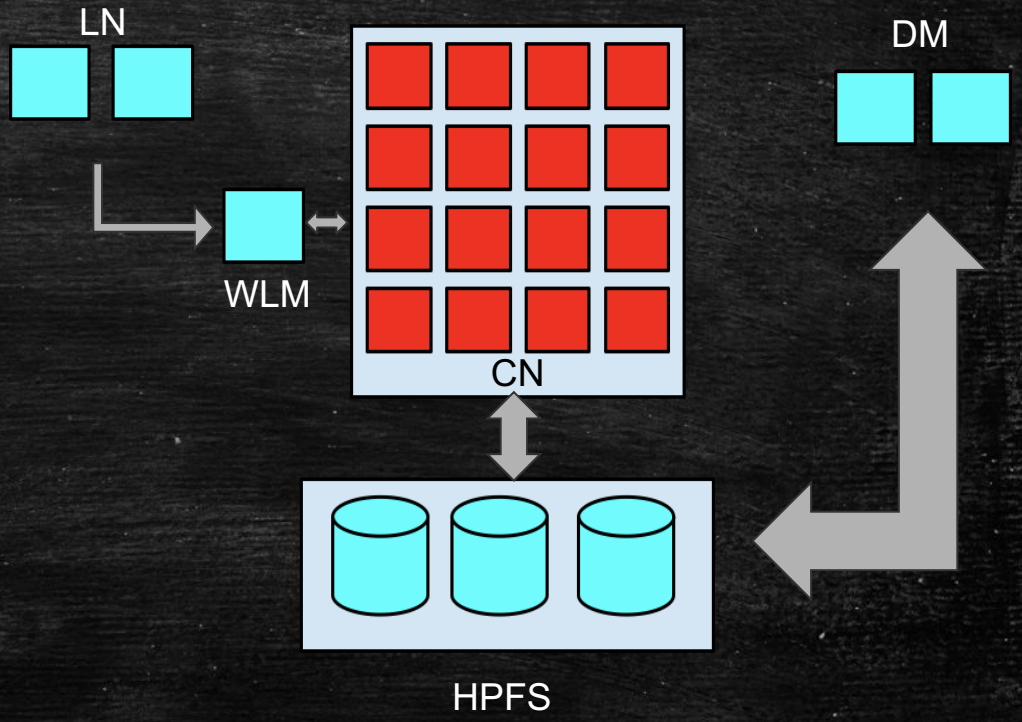
- Distributed memory machines are more common parallel architectures today
  - Scale out to millions of core
  - Less expensive to build and maintain
  - Can support variable types of workloads

- Can be homogeneous or heterogeneous cluster of compute nodes.
- These machine are a bit harder to program than shared memory machine but are inevitable for bigger science
- For best ROI, these machine run a workload manager to cater variable types of workload simultaneously by allocating resources.

Putting it all together!

A supercomputer is not just the compute nodes. Following components complete the picture

- Login nodes (LN)
- Workload Manager (WLM)
- Compute nodes (CN)
- High performance filesystem (HPFS)
  - High speed interconnect b/w CN and HPFS
- Datamover nodes (DM)




Questions?