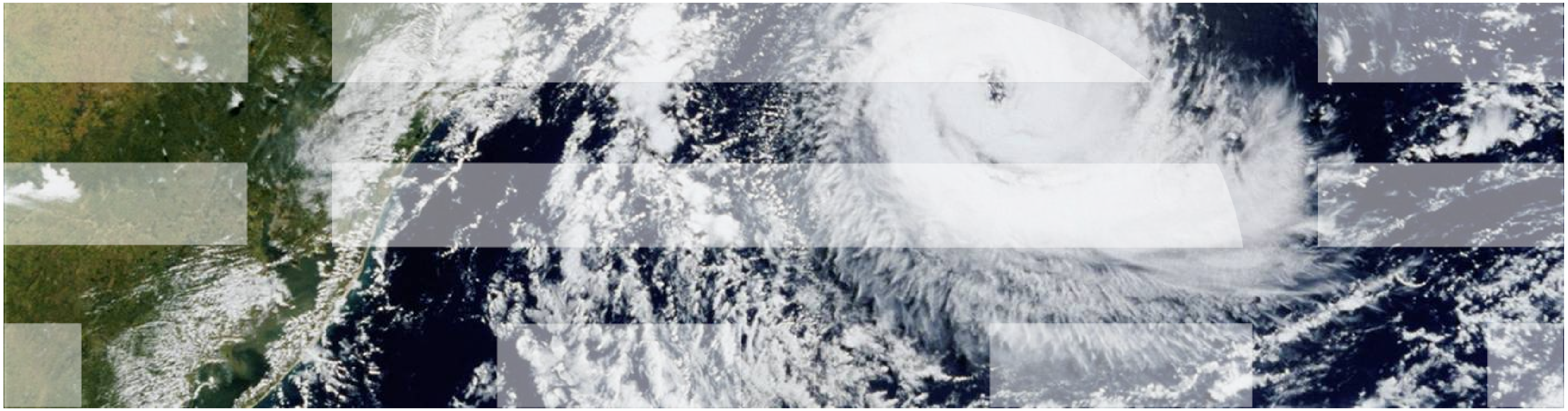
Programming Techniques for Supercomputers

(Heterogeneous and Distributed systems)

– Aditya Nitsure (IBM India)



Github Repo

- For presentations and reference material :

  https://github.com/anitsure/SREC_PTFS_Course

Content

Why do we need parallel computing?

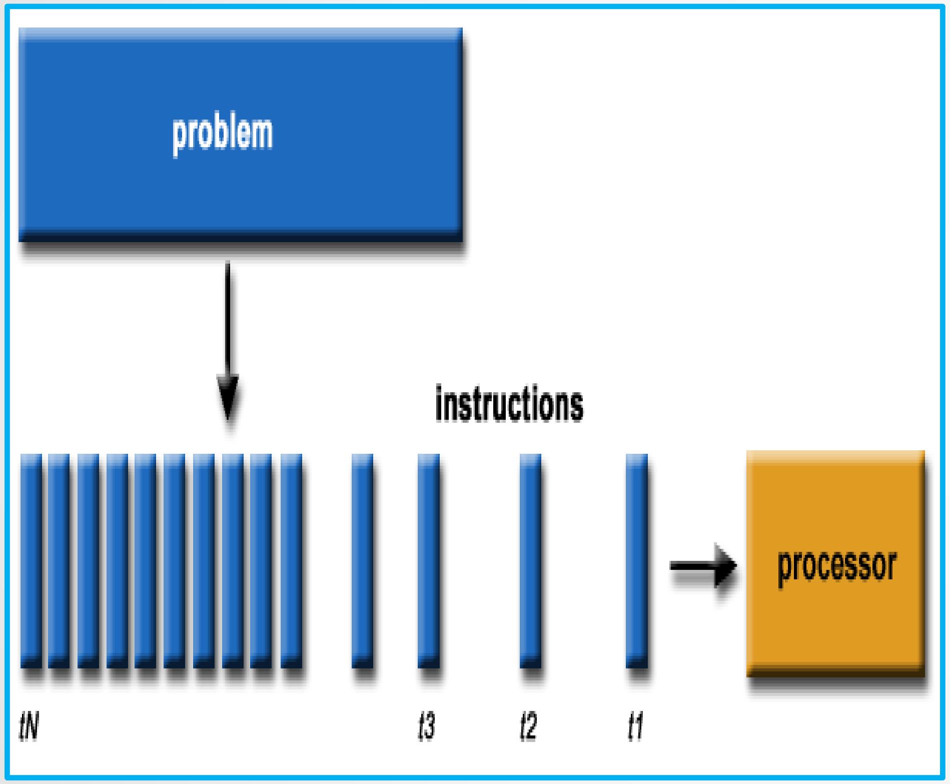
Why Parallel Computing ?

- Real world is parallel
- Data in nature is parallel
- Better utilization of resources
- Saves the time/money
- Huge problem sizes

“ parallel computing is the simultaneous use of multiple compute resources to solve a computational problem

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

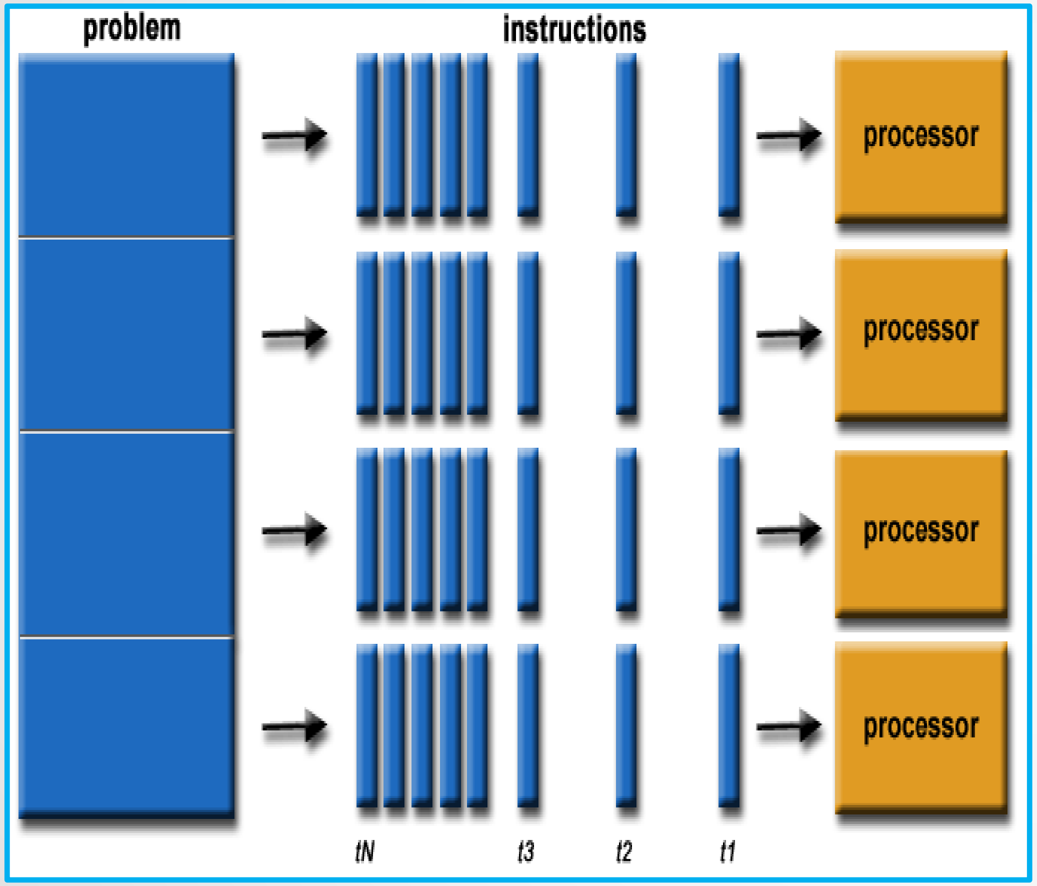
Sequential Computing



- A problem is broken into a discrete series of instructions
- Instructions are executed sequentially one after another
- Executed on a single processor
- Only one instruction may execute at any moment in time

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

Parallel Computing

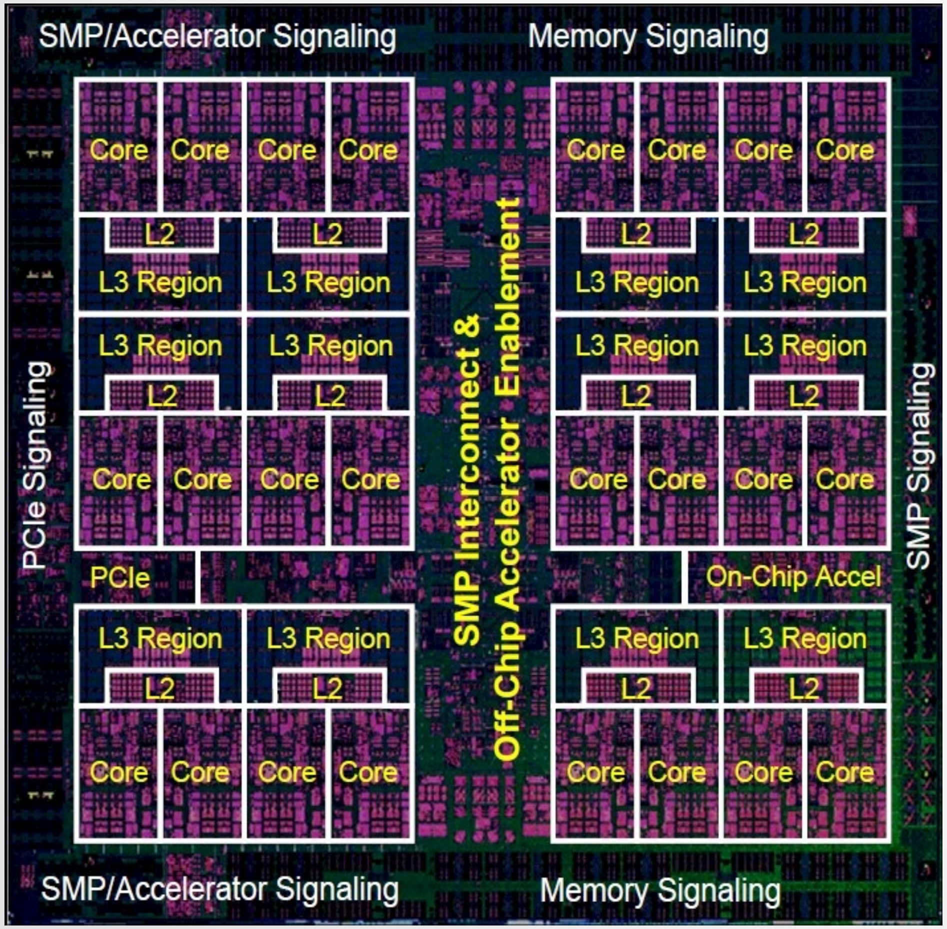


- A problem is broken into discrete parts that can be solved concurrently
- Each part is further broken down to a series of instructions
- Instructions from each part execute simultaneously on different processors
- **Note – more than 1 resources required !**

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

Parallel Computing

Every stand alone computer has parallelism

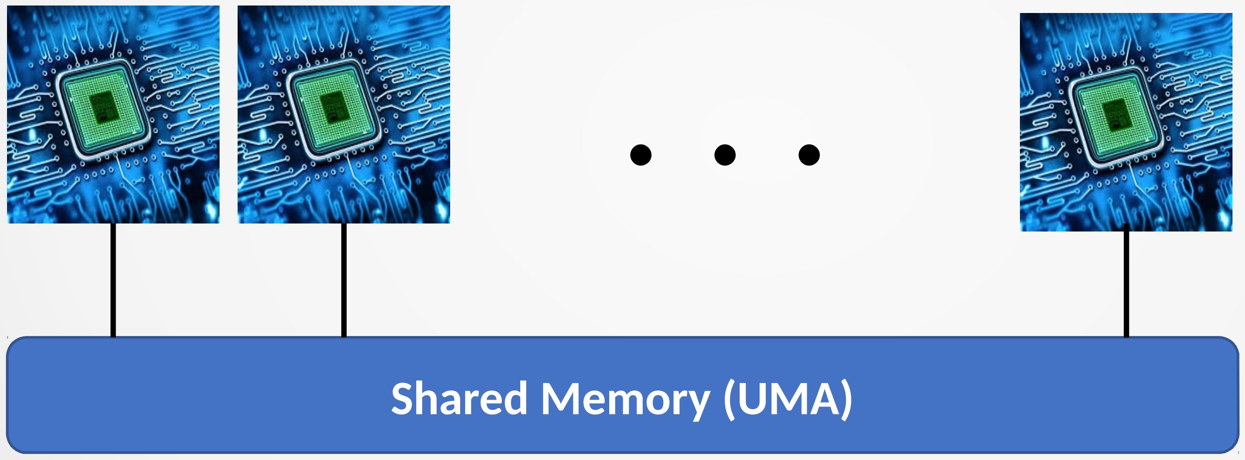

SMP/Accelerator Signaling
Memory Signaling
Core
L2
L3 Region
PCIe
On-Chip Accel
SMP Interconnect & Off-Chip Accelerator Enablement
PCIe Signaling
SMP Signaling
SMP/Accelerator Signaling
Memory Signaling

- Multiple execution units (cores)
- Multiple functional units (cache, pre-fetch, decode)
- Multiple hardware threads

Parallel Architectures

Shared Memory Architecture

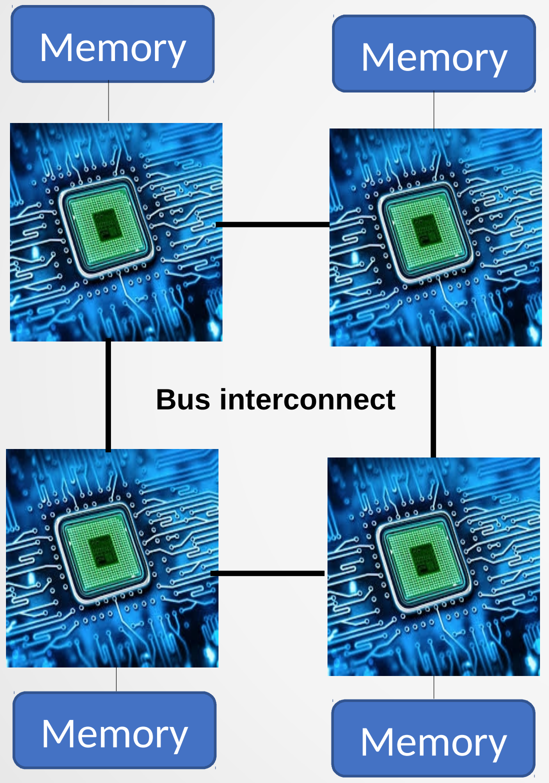
- All processors can access all memory as global address space
- Changes in a memory location effected by one processor are visible to all other processors.
- Most of the shared memory systems are cache coherent.



Uniform memory access (UMA)

- Most commonly represented today by Symmetric Multiprocessor (SMP) machines
- All processors have equal access time to all memory

Shared Memory Architecture



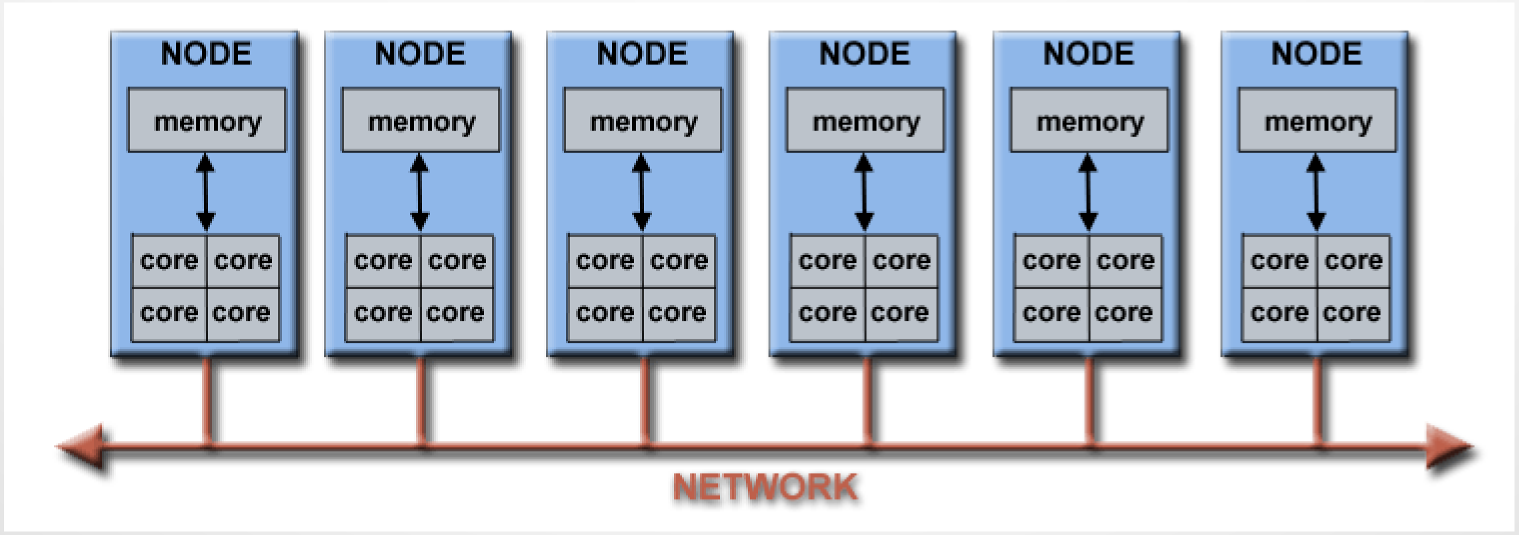
Shared memory (NUMA)

Non-uniform memory access (NUMA)

- Often made by physically linking two or more SMPs
- One SMP can directly access memory of another SMP
- Not all processors have equal access time to all memories
- Memory access across link is slower

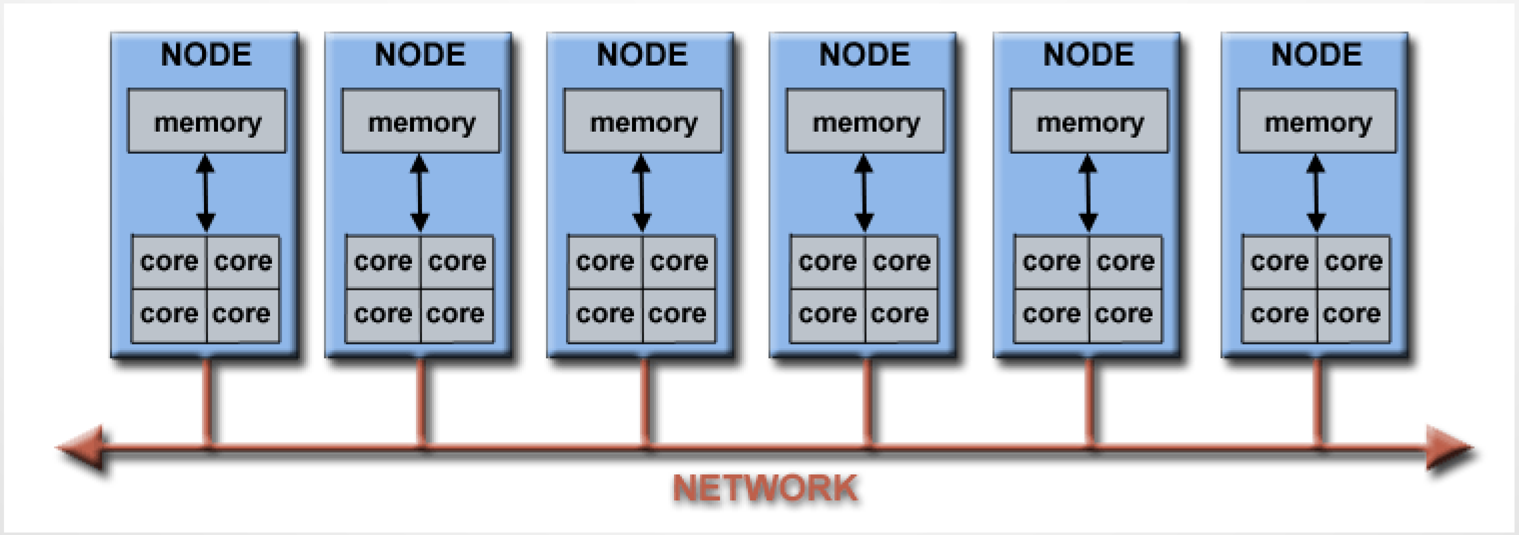
Distributed Memory Architecture

- Distributed memory systems require a communication network to connect inter-processor memory.
- Processors have their own local memory. Memory addresses in one processor do not map to another processor, so there is no concept of global address space across all processors.
- Changes made by processor in its local memory have no effect on the memory of other processors.



Distributed Memory Architecture

- Concept of cache coherency does not apply.
- When a processor needs access to data in another processor, it is usually the task of the programmer to explicitly define how and when data is communicated.
- Synchronization between tasks is likewise the programmer's responsibility.

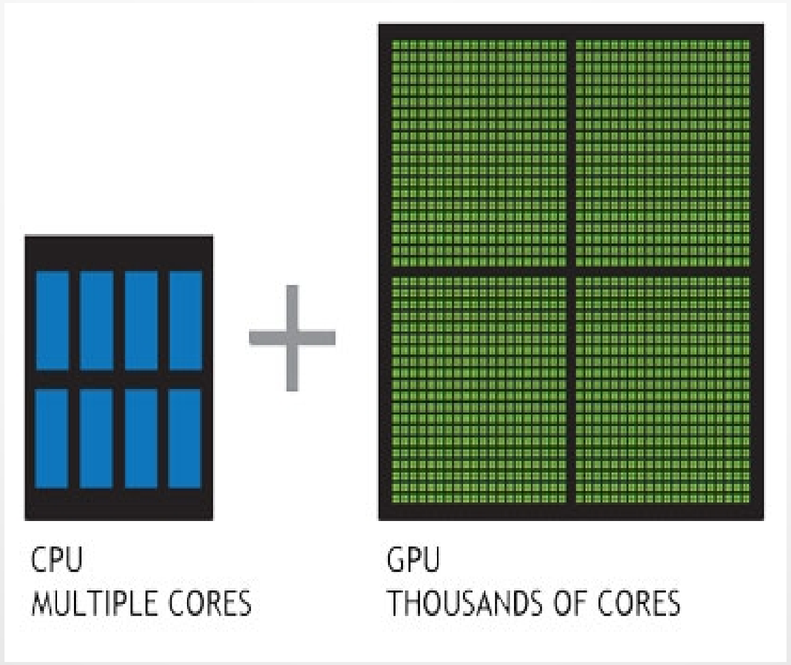


Heterogeneous Architecture

Heterogeneous System

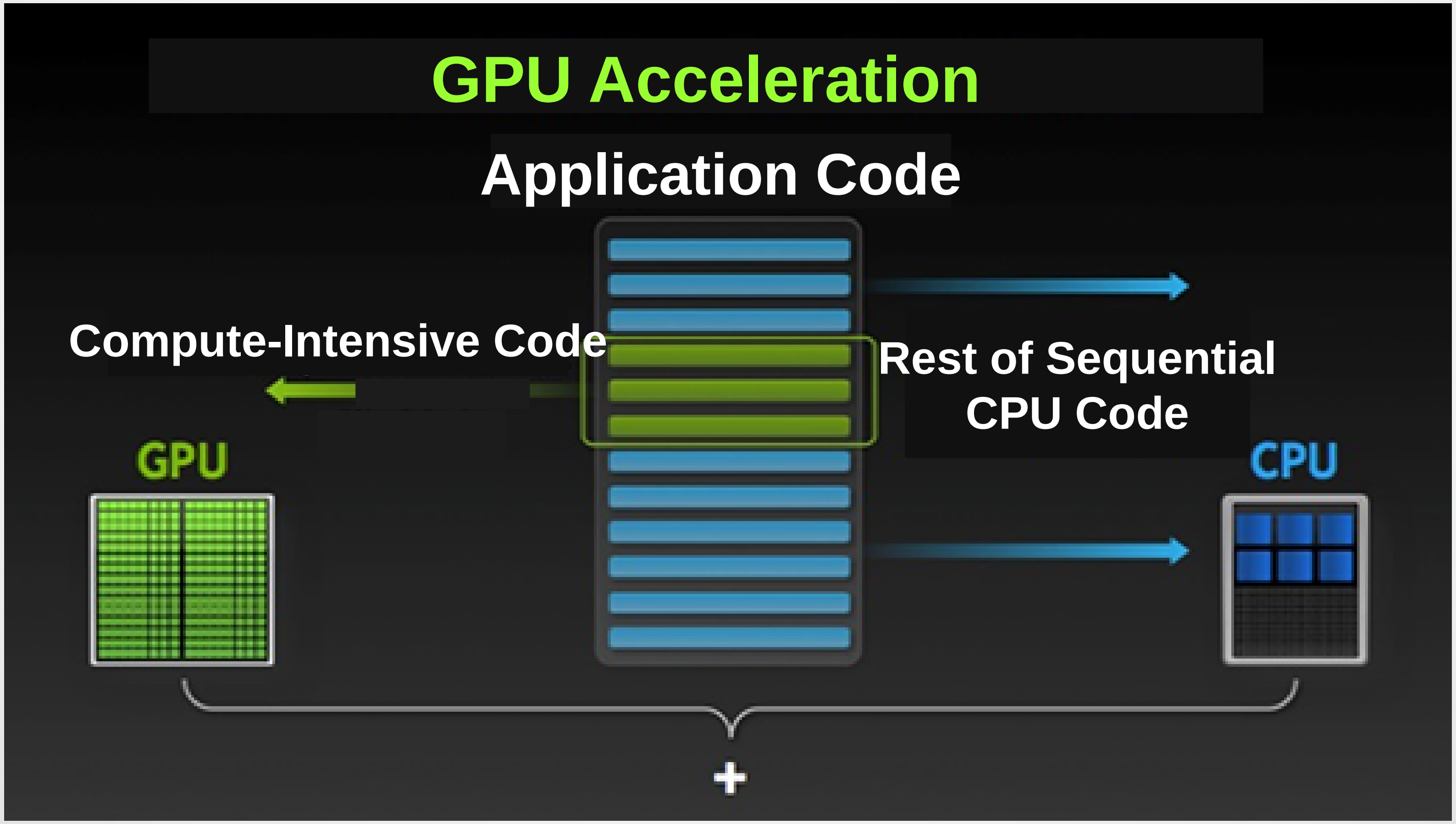
Combines more than one type of processors

- CPU
  - Large and broad instruction set to perform complex operations
- GPU
  - A High throughput – Massive parallelization through large number of cores
  - Specialized for SIMD/SIMT

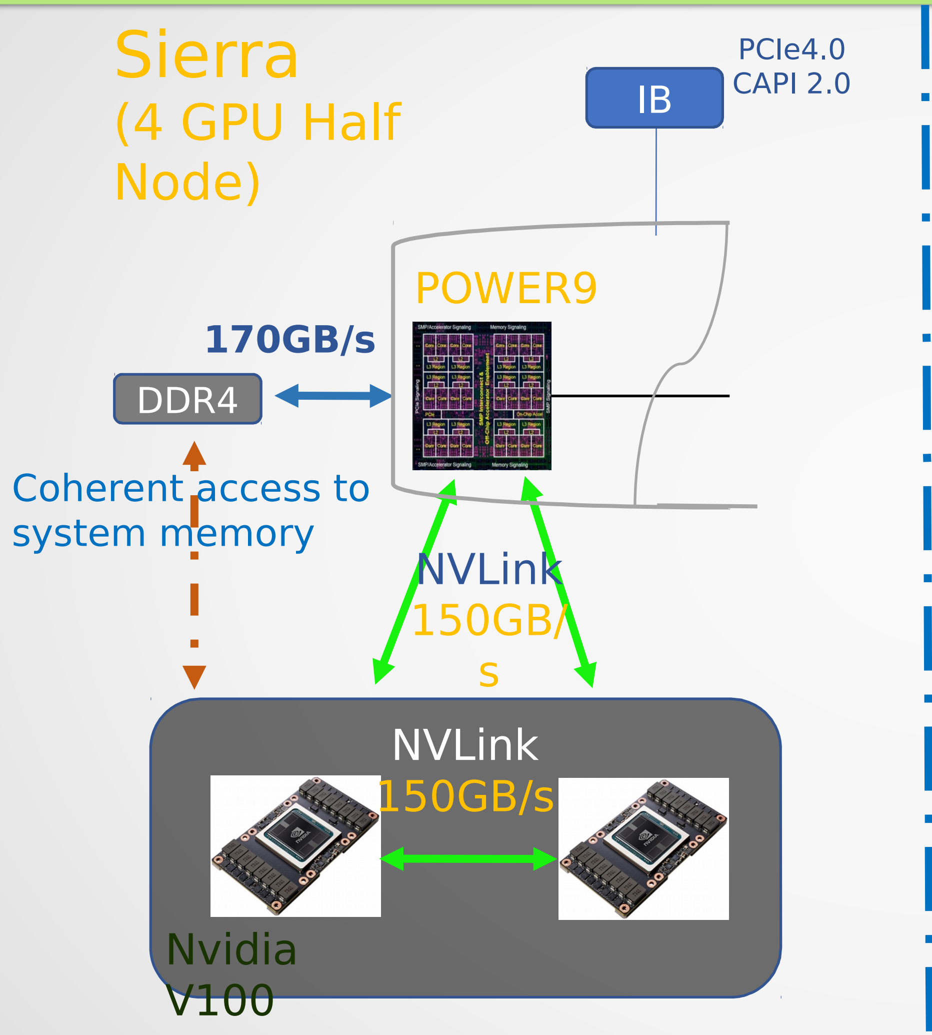


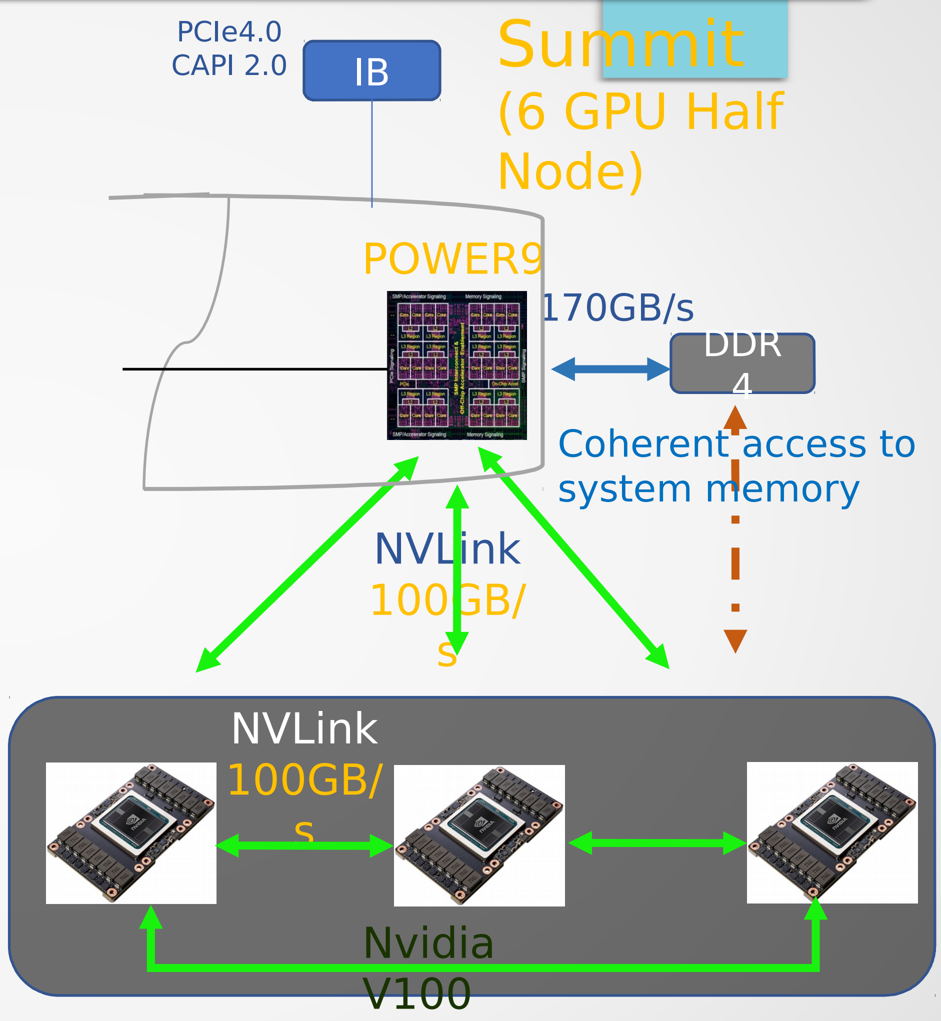
Heterogeneous System

CPU and GPU co-operate in execution of work



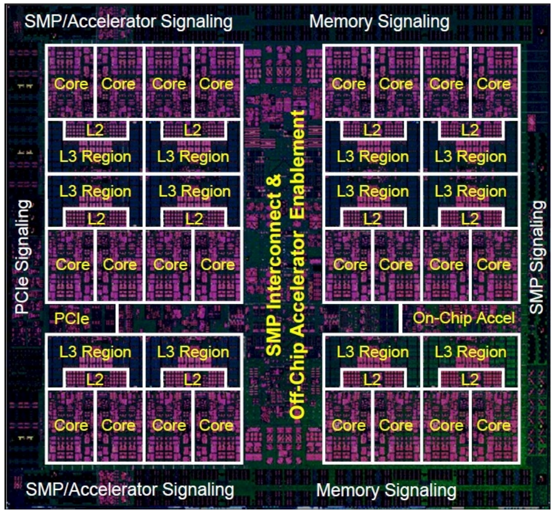
Summit and Sierra supercomputer configuration





- Nvlink 2.0 – high-bandwidth interconnect
  - 150 bi-directional bandwidth (or 100 GB/s for 6 GPU config) between CPU-GPU and GPU-GPU
- CPU Coherently access to CPU memory

IBM Power9 Processor


SMP/Accelerator Signaling
Memory Signaling
Core
L2
L3 Region
PCIe Signaling
SMP Signaling
SMP Interconnect & Off-Chip Accelerator Enablement
PCIe
On-Chip Accel

- 14nm finFET semiconductor
- Stronger thread performance
- POWER ISA 3.0
- Enhanced Cache Hierarchy
- NVIDIA NVLink 2.0 with 25GB/s per link b/w
- I/O System – PCIe Gen4
- Improved device performance & reduced energy
- Nominal & Peak freq: 2.8GHz & 3.8GHz

Nvidia Tesla V100 GPU

- Designed for AI Computing and HPC
- Second-Generation NVLink™
- HBM2 Memory: Faster, Higher Efficiency
- Number of SM/cores : 80/5120
- Double Precision Performance : 7.5 TFLOPS
- Single Precision Performance : 15 TFLOPS
- 125 Tensor TFLOPS
- GPU Memory : 16 or 32 GB
- Memory bandwidth : 900 GB/s

Source :https://devblogs.nvidia.com/inside-volta

Parallel Programming Paradigms

Flynn’s Taxonomy

Classification of parallel computers

SISD
(Single Instruction Single Data)

- A serial (non-parallel) computer
- Deterministic execution
- This is the oldest type of computer
- Examples: older generation systems

SIMD
(Single Instruction Multiple Data)

- A type of parallel computer
- Synchronous and deterministic execution
- Best suited for problems with a high degree of regularity e.g graphics/image processing.
- Examples: Most modern computers

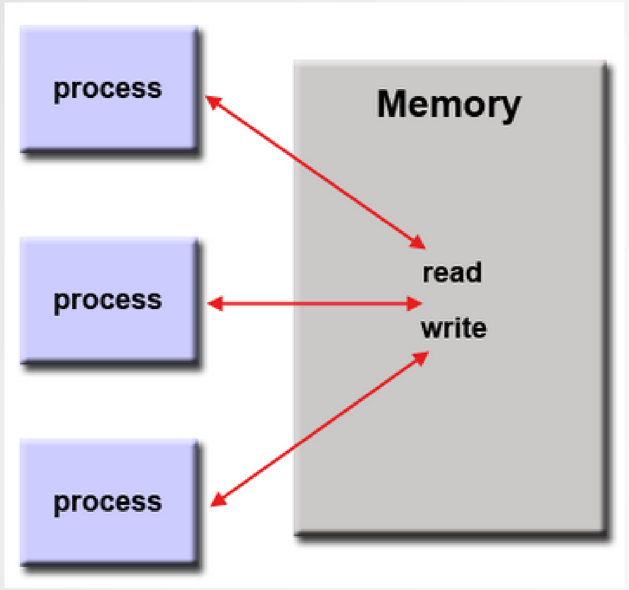
MISD
(Multiple Instructions Single Data)

- A type of parallel computer
- Few (if any) actual examples of this class of parallel computer have ever existed.
- Example: multiple cryptography algorithms attempting to crack a single coded message.

MIMD
(Multiple Instructions Multiple Data)

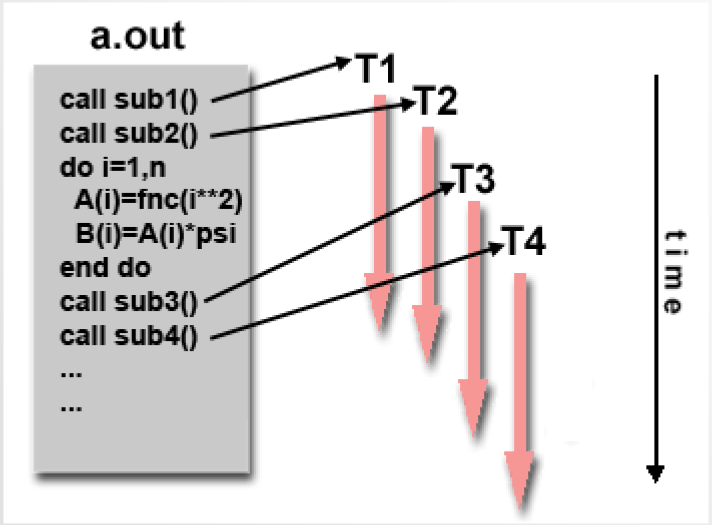
- A type of parallel computer
- Execution can be synchronous or asynchronous, deterministic or non-deterministic
- Examples: most current supercomputers, clusters, multi-processor SMP computers, multi-core PCs.

Shared Memory



Process

- All Processes share a common address space.
- All processes see and have equal access to shared memory.
- No explicitly the communication of data between tasks



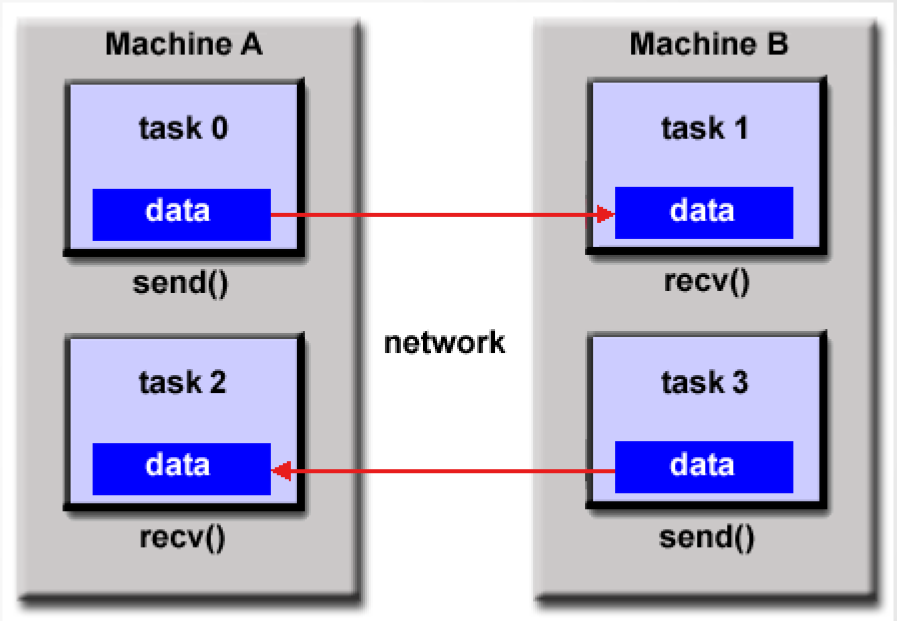
```
call sub1()
call sub2()
do i=1,n
  A(i)=fnc(i**2)
  B(i)=A(i)*psi
end do
call sub3()
call sub4()
...
...
```

Thread

- A single process can have multiple threads
- Example : OpenMP, POSIX Threads

Distributed Memory

- A set of tasks that use their own local memory during computation.
- Multiple tasks can reside on the same physical machine and/or across an arbitrary number of machines.
- Tasks exchange data through communications by sending and receiving messages.



- Data transfer usually requires cooperative operations to be performed by each process. For example, a send operation must have a matching receive operation.
- Example : MPI (Message Passing Interface)

Pros-cons of shared and distributed memory

Shared Memory

Pros

- Easier to program (global memory address space)
- Fast memory access

Cons

- Hard to scale
- Adding CPUs increases traffic
- Programmer initiates synchronization of memory access

Distributed Memory

Pros

- Easily Scalable
- Local memory access is fast

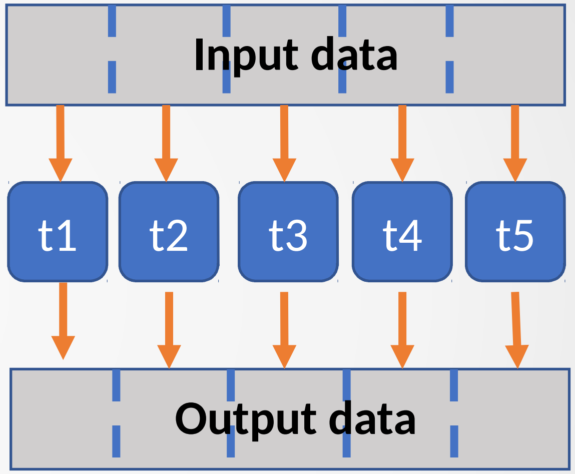
Cons

- Programming is complex
- Communication is complex

Task Parallelism

Concurrent execution of different tasks on same or different data

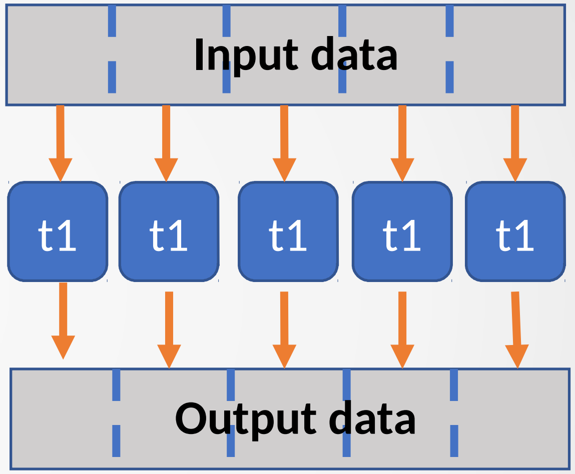
- Functionality is divided among threads/processes
- Each compute unit performs different task
- Follows MIMD model



Data Parallelism

Concurrent execution of the same task (instructions) on different datasets.

- Data is partitioned
- Data subsets are distributed on threads/processes
- Each thread/compute unit performs same task
- Follows SIMD/SPMD model



Amdal’s Law

Speedup and Efficiency

$$\textit{Speedup with p processors, } S_p = \frac{\textit{Execution time of single processor } (t_1)}{\textit{Execution time of p processor } (t_p)}$$

$$\textit{Parallel efficiency} = \frac{\textit{Speedup } (S_p)}{\textit{Number of processors } (P)}$$

Amdal’s law

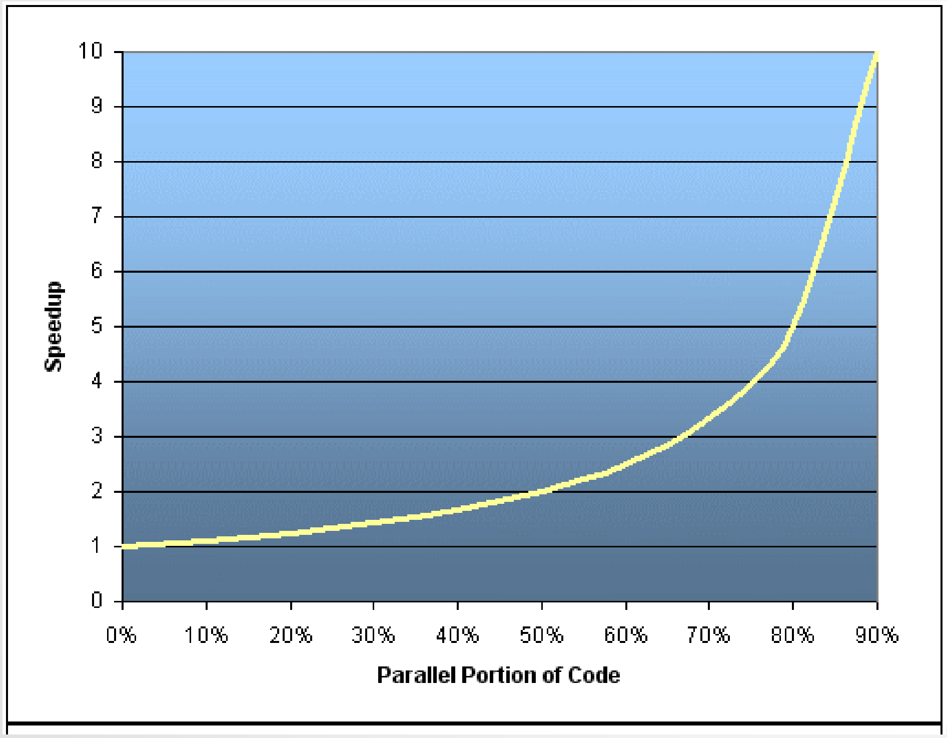
- S = sequential portion in the program
- 1 – S = parallel portion in the program
- P = number of processors
- t_1 = execution time of single processor

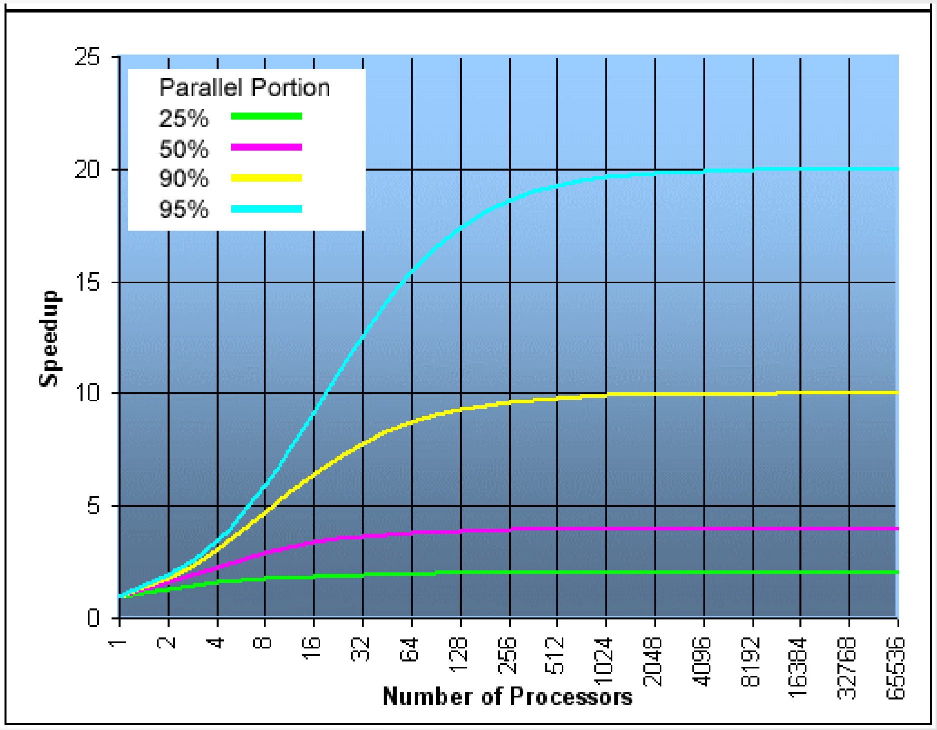
Parallel time $(t_p) = t_1 \times (S + (1 - S) / p)$

Speedup with p processors, $S_p = \dfrac{1}{(S + (1 - S) / p)}$

Amdal’s law – Illustration

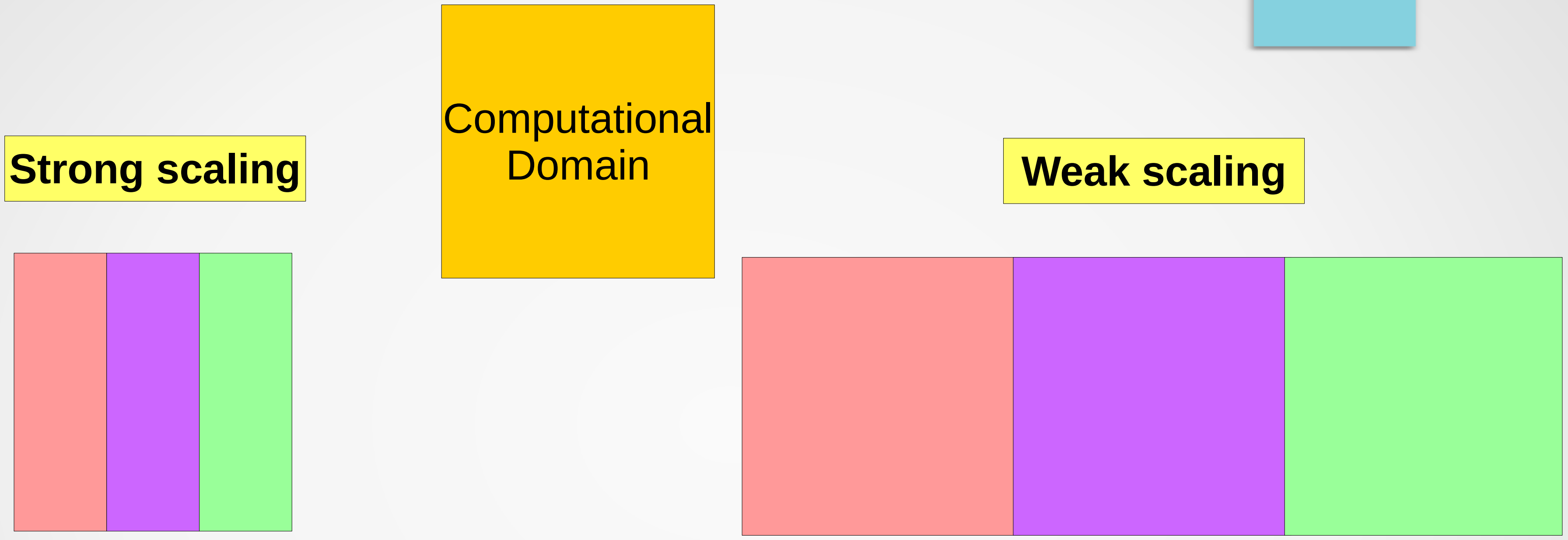
"Famous" quote: *You can spend a lifetime getting 95% of your code to be parallel, and never achieve better than 20x speedup no matter how many processors you throw at it!*





Overcoming Amdal’s law

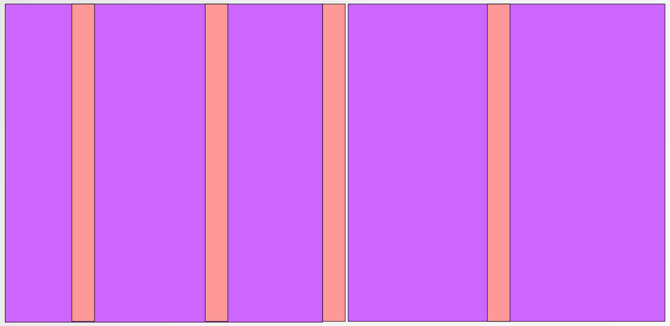
Scalability



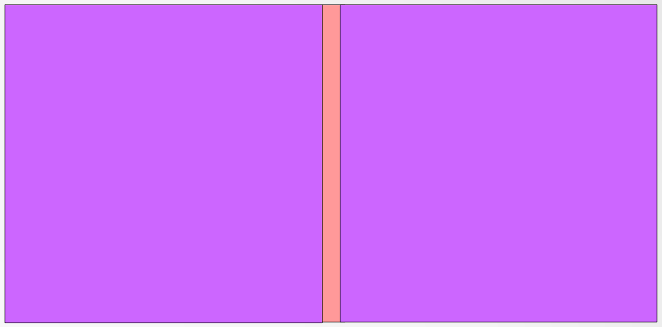
- The total problem size stays fixed as more processors are added.
- Goal is to run the same problem size faster
- Perfect scaling means problem is solved in 1/P time (compared to serial)

- The problem size per processor stays fixed as more processors are added. The total problem size is proportional to the number of processors used.
- Goal is to run larger problem in same amount of time
- Perfect scaling means problem Px runs in same time as single processor run

Granularity



Fine grain



Course grain

- Relatively small amounts of computational work are done between communication events
- Low computation to communication ratio
- Facilitates load balancing
- Implies high communication overhead

- Relatively large amounts of computational work are done between communication/synchronization events
- High computation to communication ratio
- Load balancing could be challenging

Best Practices

Best Practices

1. Effective partitioning
   - Computational domain decomposition
   - Functional decomposition
2. Overlap communication with computation
   - Choose wisely between synchronous and asynchronous communication
   - Monitor compute–to–communication ratio.
3. Equal distribution of Load
4. Remove unnecessary synchronizations
   - Blocking and run the code sequentially
5. Utilize local memory effectively

Think in parallel !!! **Think concurrent !!!**

Disclaimer

- Some of the images and information is copied from publicly available web site from Lawrence Livermore National Laboratory. (https://computing.llnl.gov/tutorials/parallel_comp/#Whatis)
- The material is used only for teaching purpose (non commercial).
- I would like to thank author of the web page Blaise Barney.

Thank you !