MAX PLANCK
COMPUTING & DATA FACILITY

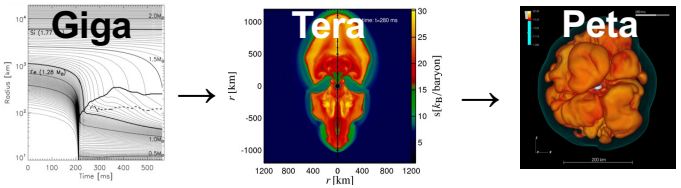
INTRODUCTION TO PARALLEL COMPUTING

Erwin Laure, MPCDF

Nomad School, 2023-10-03

APPLICATIONS: WHY BIGGER AND FASTER COMPUTERS ?

The quest for more (physical, chemical, biological, …) realism:



supernovae simulations in 1d/2d/3d
courtesy Markus Rampp

Improving numerical accuracy

- higher resolution, larger system sizes (→ more grid points, particles, …)

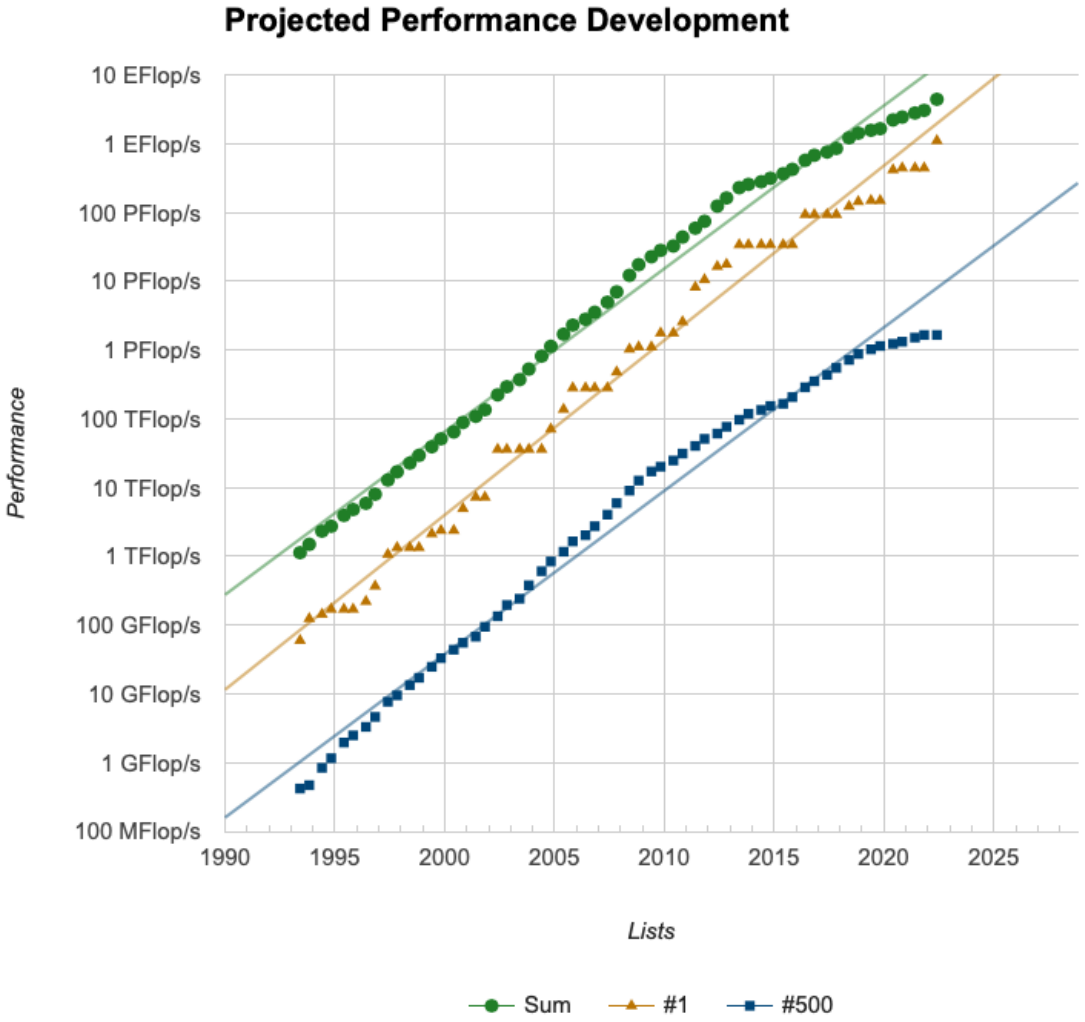
Analyzing bigger datasets

- e.g. SKA > 600 PB/year

AI

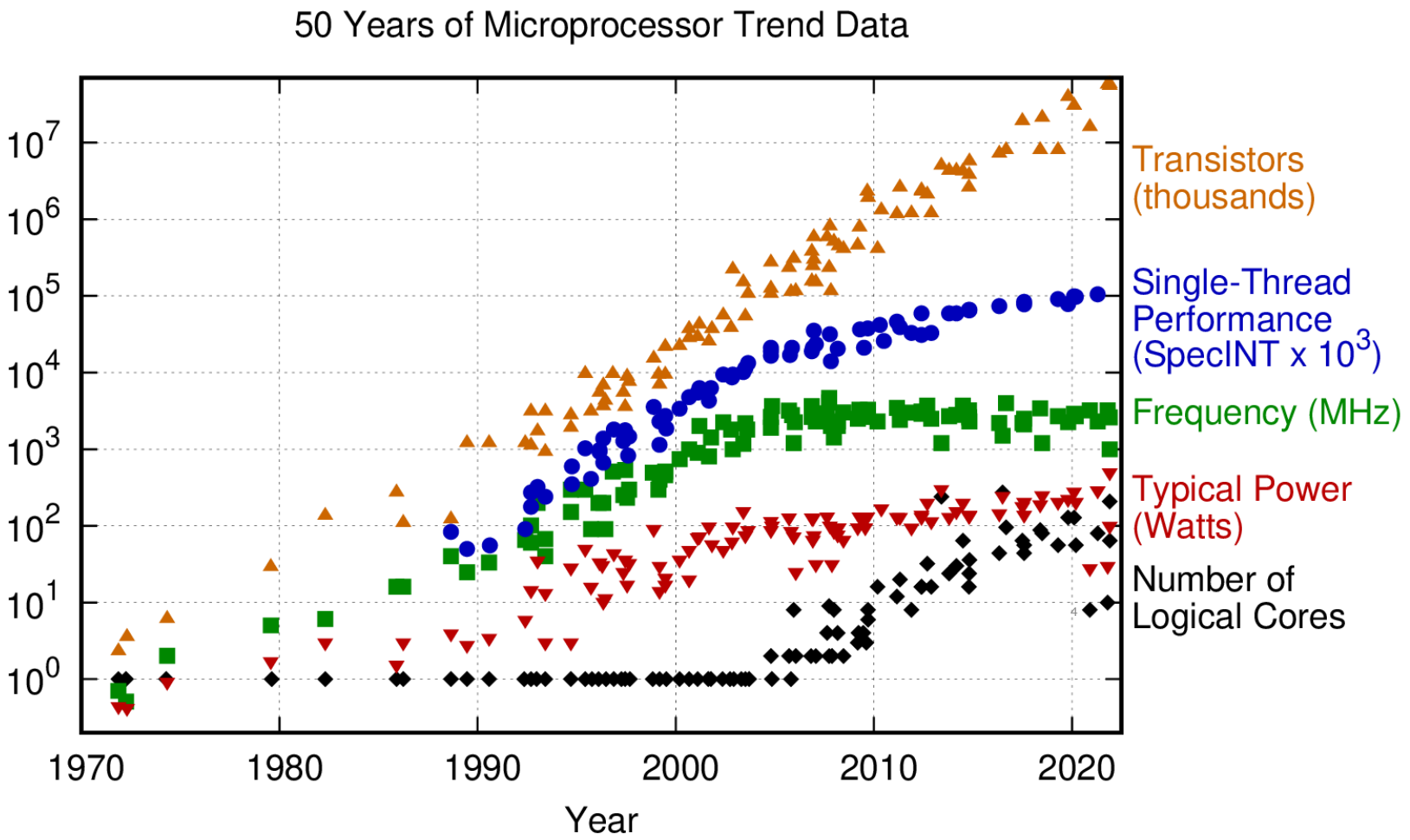
- GPT4 was likely trained on > 10.000 GPUs for ~11 months

WE ARE STILL BUILDING LARGER AND LARGER COMPUTERS



Top500 list (www.top500.org)

FURTHER PERFORMANCE IMPROVEMENTS ONLY POSSIBLE THROUGH PARALLELISM



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2021 by K. Rupp

PARALLELISM ON DIFFERENT LEVELS

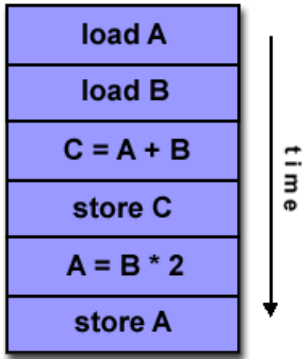
- **CPU**
  - Instruction level parallelism, pipelining
  - Vector unit
  - Multiple cores
  - Multiple threads or processes
- **Node**
  - Multiple CPUs
  - Co-processors (GPUs, FPGAs, …)
- **Network**
  - Tightly integrated network of computers (supercomputer)
  - Loosely integrated network of computers (distributed computing)

FLYNN'S TAXONOMY (1966)

- **{Single, Multiple} {Instructions, Data}**

SISD Single Instruction, Single Data	SIMD Single Instruction, Multiple Data
MISD Multiple Instruction, Single Data	MIMD Multiple Instruction, Multiple Data

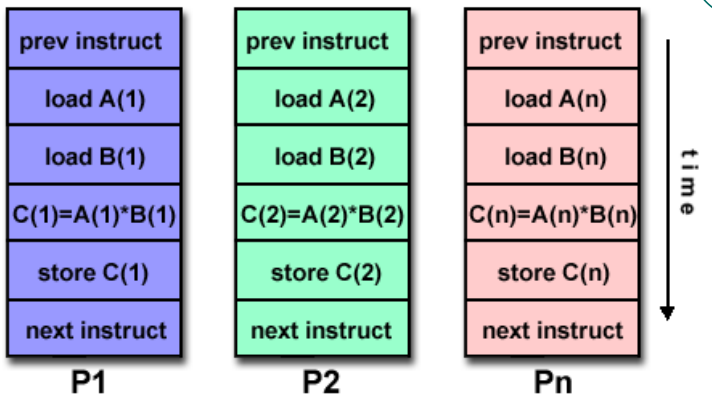
SINGLE INSTRUCTION SINGLE DATA



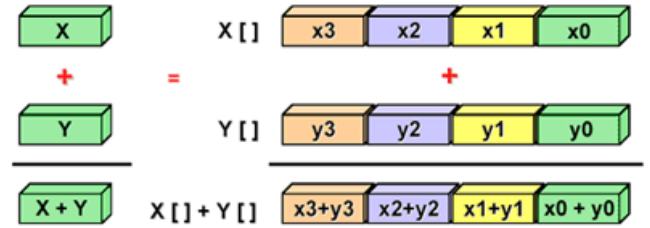
- A serial (non-parallel) computer
- Single instruction: only one instruction stream is being acted on by the CPU during any one clock cycle
- Single data: only one data stream is being used as input during any one clock cycle
- Deterministic execution
- This is the oldest and used to be the most common type of computer (up to arrival of multicore CPUs)
- Examples: older generation mainframes, minicomputers and workstations; older generation PCs.

- Attention: single core CPUs exploit instruction level parallelism (pipelining, multiple issue, speculative execution) but are still classified SISD

SINGLE INSTRUCTION MULTIPLE DATA

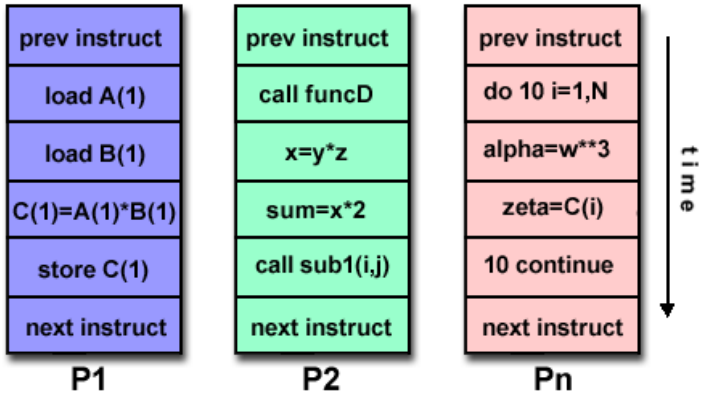


- **"Vector" Computer**
- **Single instruction:**
  All processing units execute the same instruction at any given clock cycle
- **Multiple data:**
  Each processing unit can operate on a different data element
- **Best suited for specialized problems characterized by a high degree of regularity, such as graphics/image processing.**
- **Synchronous (lockstep) and deterministic execution**
- **Two varieties: Processor Arrays and Vector Pipelines**
- **Most modern computers, particularly those with graphics processor units (GPUs) employ SIMD instructions and execution units.**



MULTIPLE INSTRUCTION, MULTIPLE DATA

- **Currently, the most common type of parallel computer. Most modern computers fall into this category.**
- **Multiple Instruction:**
  every processor
  may be executing a different
  instruction stream



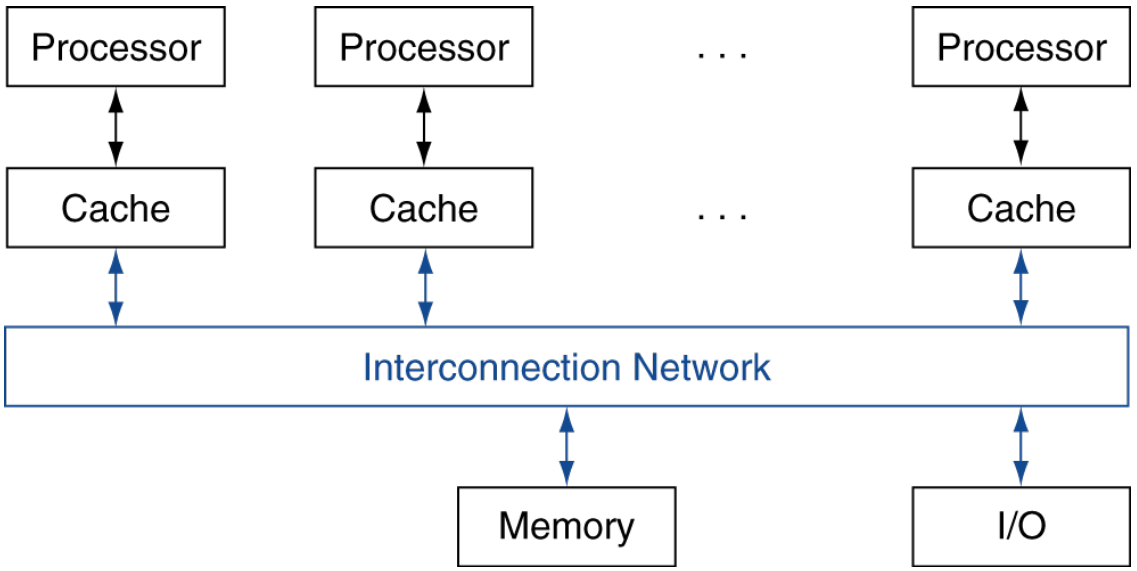
- **Multiple Data:**
  every processor may be working with a different data stream
- **Execution can be synchronous or asynchronous, deterministic or non-deterministic**
- **Examples: most current supercomputers, networked parallel computer clusters and "grids", multi-processor SMP computers, multi-core PCs.**
- **Note: many MIMD architectures also include SIMD execution sub-components**

MULTIPLE INSTRUCTION, SINGLE DATA

- **No examples exist today**
- **Potential uses might be:**
  - Multiple cryptography algorithms attempting to crack a single coded message
  - Multiple frequency filters operating on a single signal

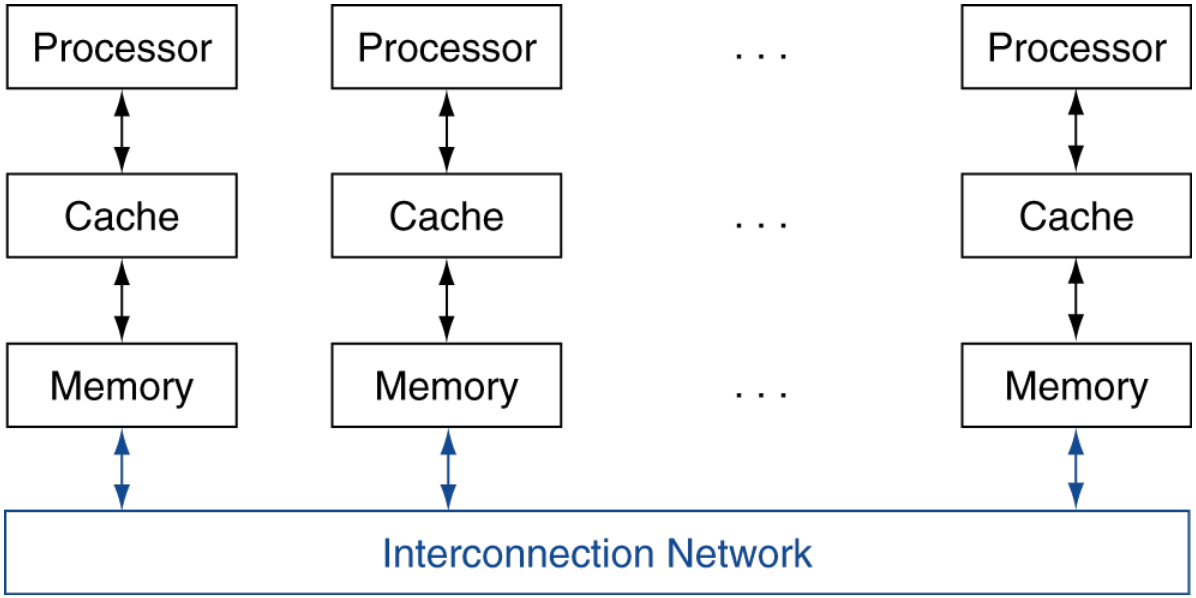
SHARED MEMORY MULTIPROCESSOR

- **Hardware provides single physical address space for all processors**
- **Global physical address space and symmetric access to all of main memory (*symmetric multiprocessor - SMP*)**
- **All processors and memory modules are attached to the same interconnect (bus or switched network)**



DISTRIBUTED MEMORY MULTIPROCESSORS (DMMPS)

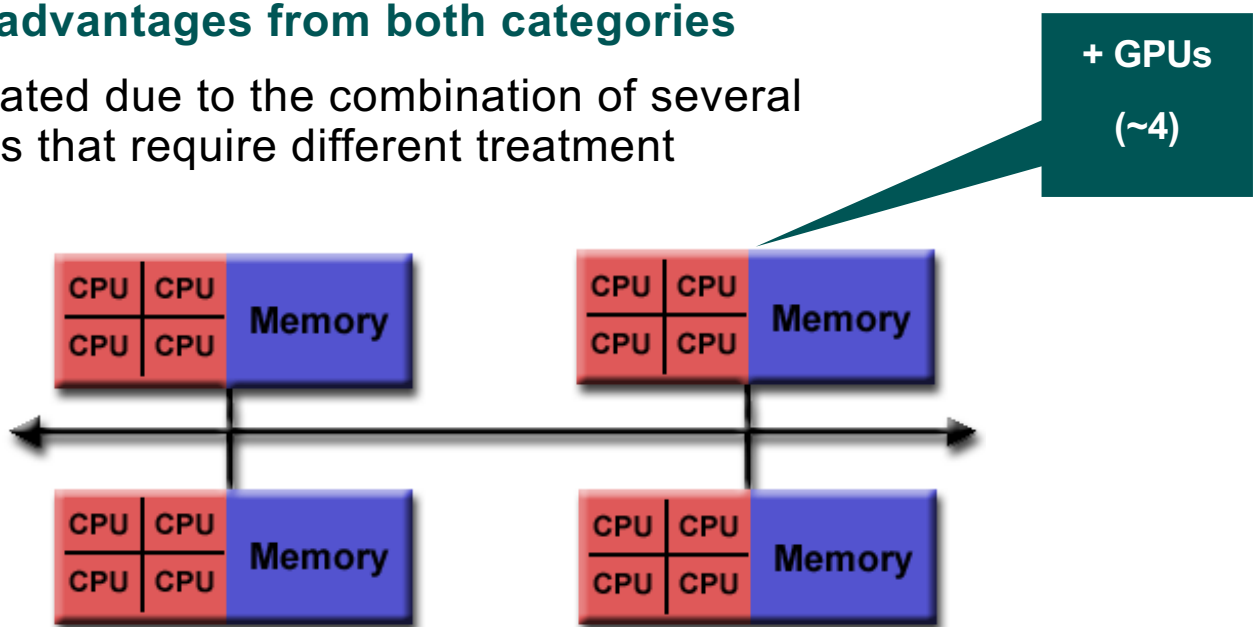
- **Each processor has private physical address space**
  - No cache coherence problem
- **Hardware sends/receives messages between processors**
  - Message passing



COMBINING SMPS AND DMMPS

- **Today, DMMPs are typically built with SMPs as building blocks**
  - E.g. LUMI-C has two AMD CPUs with 64 cores each per DMMP node
  - Soon systems with more CPUs and many more cores will appear (upcoming AMD CPUS ~200 cores)

- **Combine advantages and disadvantages from both categories**
  - Programming is more complicated due to the combination of several different memory organizations that require different treatment



MAX PLANCK
COMPUTING & DATA FACILITY

MPCDF

PROGRAMMING PARALLEL SYSTEMS

MILLION SCALE PARALLELISM REQUIRED FOR LARGEST MACHINES

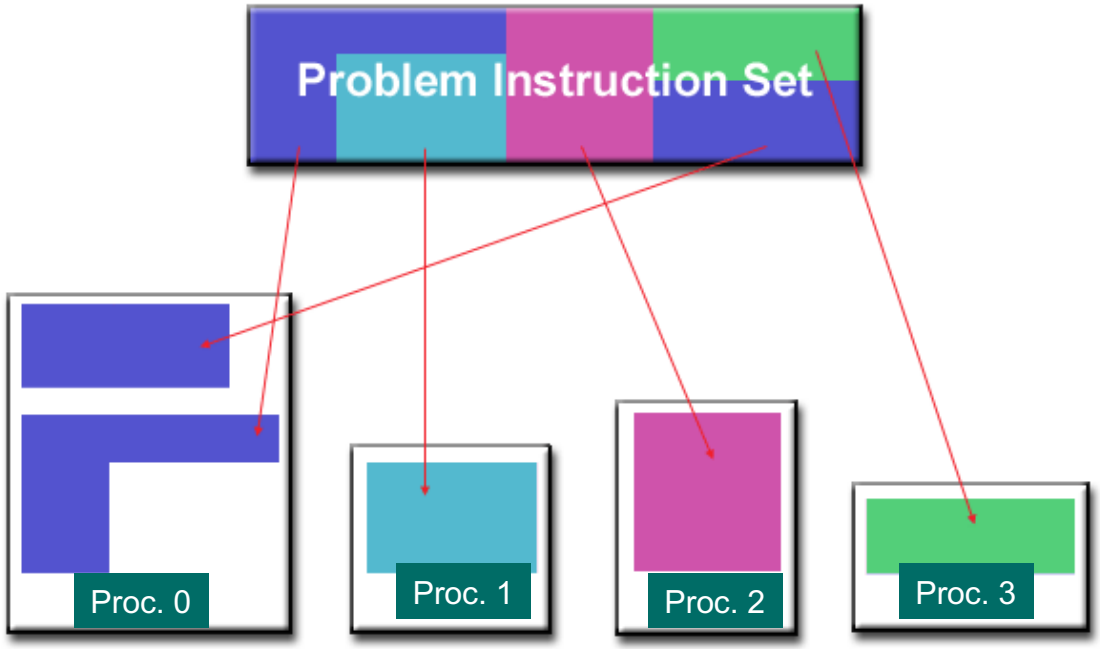
FRONTIER @ OLCF (US): HPE/CRAY

- **AMD EPYC CPUs, AMD MI250 GPUs**
- **8.7 M CPU cores & GPU compute units**
- **52 GFlop/s/W**
- **1102 PFlop/s HPL, rank 1 in Top500 11/2022**

FUNCTIONAL DECOMPOSITION

The problem is decomposed according to the work that must be done. Each task then performs a portion of the overall work.

Also called “Task Parallelism”



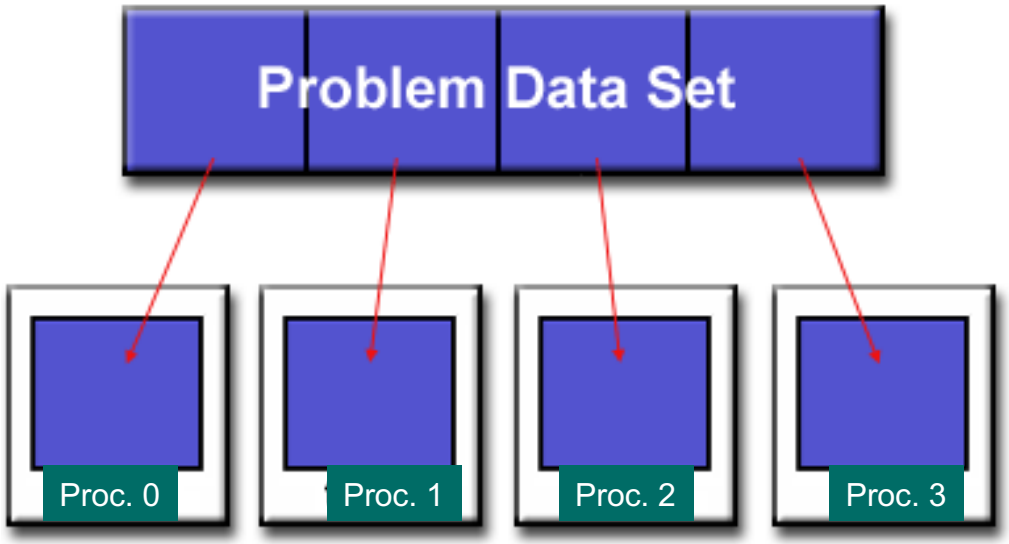
TASK PARALLELISM DISCUSSION

- **Often pipelined approaches or Master/Worker**
  - Master assigns work items to its workers
- **“Natural” approach to parallelism**
- **Typically good efficiency**
  - Tasks proceed without interactions
  - Synchronization/communication needed at the end
- **In practice scalability is limited**
  - Problem can by split only into a finite set of different tasks

DOMAIN DECOMPOSITION

The data associated with a problem is decomposed. Each parallel task then works on a portion of of the data.

Also called “Data Parallelism”



PROGRAMMING PARALLEL SYSTEMS

- **Libraries**
  - Message Passing Interface (MPI)
  - Abstraction Libraries (Kokkos, Raja, etc.)
- **Explicit parallel constructs in higher-level languages**
  - Parallel loops, array operations, …
  - Fortran >90, DPC/Sycl
- **Compiler directives**
  - “Hints” to the compiler on how to parallelize a program
  - OpenMP
- **Parallel Languages**
  - Chapel, X10, Partitioned Global Address Space (PGAS)
  - Mostly research projects, almost no production use
  - Some concepts have been transferred to “normal” programming languages (Fortran, Sycl)

MAX PLANCK
COMPUTING & DATA FACILITY

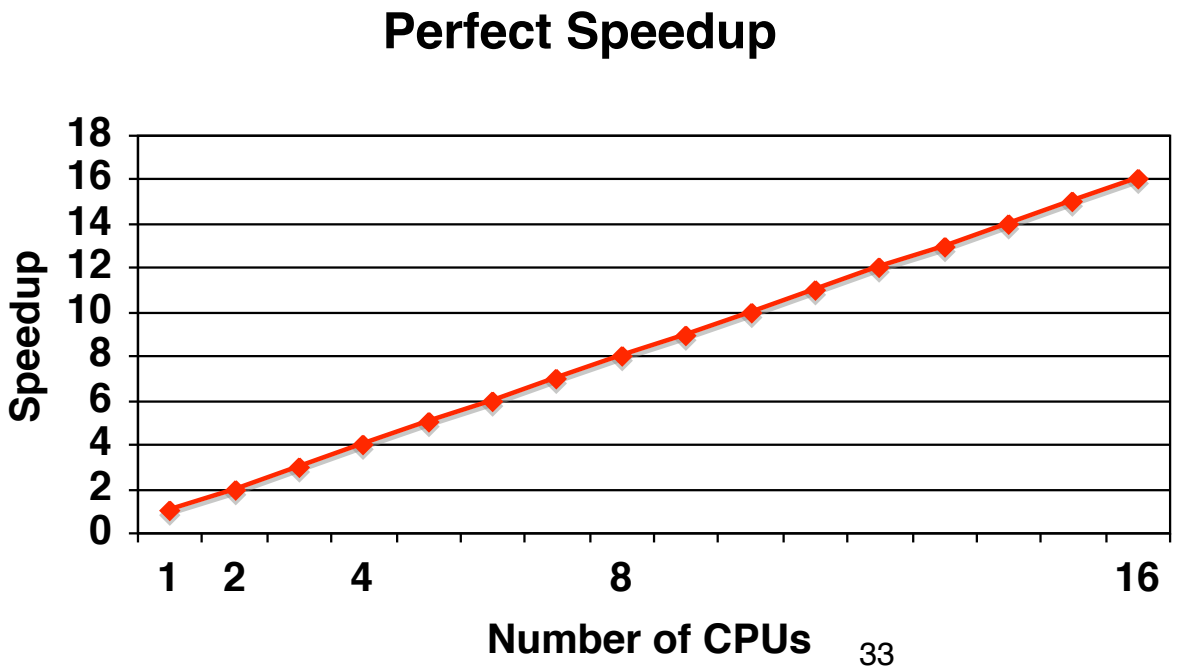
MPCDF

PERFORMANCE

SPEEDUP

- Speedup (S) is defined as the improvement in execution time when increasing the amount of parallelism or sequential execution time (T_S) over parallel execution time (T_P)

$$S = \frac{T_S}{T_P}$$



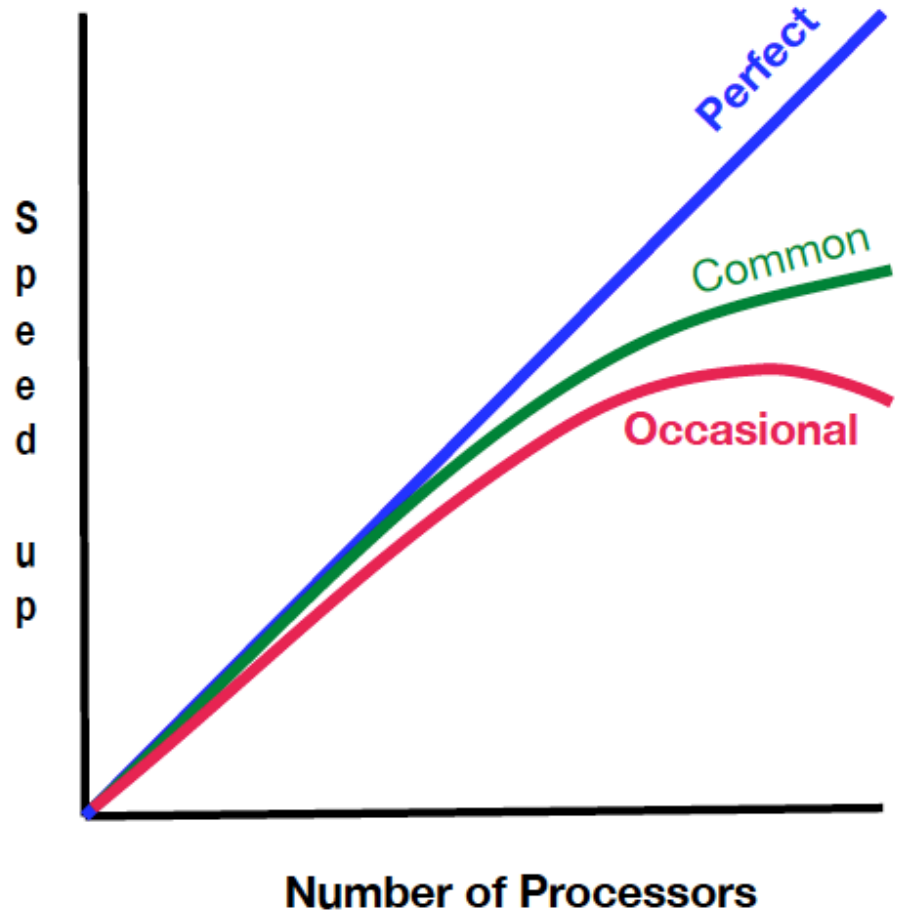
EFFICIENCY

- **Speedup as percentage of number of processors**

$$E = \frac{1}{P} \times \frac{T_S}{T_P} = \frac{1}{P} S$$

- **A speedup of 90 with 100 processors yields 90% efficiency.**

TYPICAL SPEEDUP CURVES



STRONG AND WEAK SCALING

- **Strong scaling is the speedup achieved without increasing the size of the problem**

- **Weak scaling is the speedup achieved while increasing the size of the problem (the work to be done) proportionally to the increase in number of processors**

WEAK SCALING EXAMPLE

- Workload: sum of 10 scalars, and 10 × 10 matrix sum
- Speed up from 10 to 100 processors
- Single processor: Time = (10 + 100) × tadd

10 processors, 10 × 10 matrix

Time = $10 \times t_{add} + 100/10 \times t_{add} = 20 \times t_{add}$

100 processors, 32 × 32 matrix

Time = $10 \times t_{add} + 1000/100 \times t_{add} = 20 \times t_{add}$

Constant performance in this example

Algorithmic complexity might require other problem sizes (e.g. non-linear) to double the amount of work

AMDAHL'S LAW

Pitfall: Expecting the improvement of one aspect of a computer to increase overall performance by an amount proportional to the size of the improvement

Gene Amdahl (1967):

$$\text{Improved time} = \frac{\text{time effected by improvement}}{\text{Amount of improvement}} + \text{time unaffected}$$

Example: Suppose a program runs for 100 seconds, with 80 seconds spent in multiply operations. Doubling the efficiency of multiply operations will result in new runtime of 60 seconds and thus a performance improvement of 1.67.

AMDAHL'S LAW AND PARALLEL PROCESSING

- According to Amdahl's law speedup is limited by the non-parallelizable fraction of a program
- Assume r_p is the parallelizable fraction of a program and r_s the sequential one. $r_p+r_s=1$
  We can compute the maximum theoretical speedup achievable on n processors with

$$S_{\max} = \frac{1}{r_s + \frac{r_p}{n}}$$

- If 20% of a program is sequential, the maximal achievable speedup is 5 for $n \Rightarrow \infty$

HOW TO LIVE WITH AMDAHL'S LAW

- Many real-world problems have significant parallel portions
- Yet, to use 100,000 cores with 90% efficiency, the sequential part needs to be limited to 0,00001%!

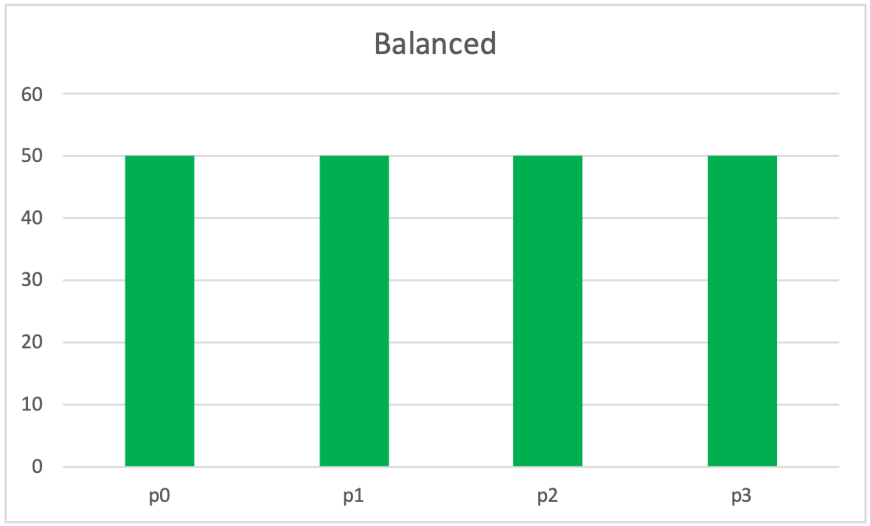
- Conclusion: minimize r_s and maximize r_p

- Increase amount of work done in the parallel (typically compute intensive) parts

LOAD BALANCING

- **Good speedup can only be achieved if the parallel workload is relatively equally spread over the available processors**

- **If workload is unevenly spread, overall performance is bound to the slowest processor (i.e. processor with most workload)**



HOW TO START PARALLEL PROGRAMMING

- **Incremental development**
  - Parallel programming is complex
  - Move from one working version to another
- **Focus on Data Structure first**
  - They will dominate performance
- **Identify bottlenecks and hot spots**
  - Where is most time spent
  - Use performance tools to identify bottlenecks
  - Compute trivial things sequentially or replicate computation
- **Be willing to write extra code**
  - To eliminate race conditions, to facilitate testing, instrumentation
- **Use existing algorithms and implementations**
  - A lot of work went into development of parallel algorithms and highly efficient implementations of common problems - reuse them