CODATA – RDA
Data Schools

High Performance Computing

Introduction

Introduction

- Utilize intricate theories in situations lacking a definitive solution: tackle equations or challenges that can only be resolved numerically, involving the insertion of values into expressions and subsequent analysis of outcomes.
- Perform experiments that may seem unattainable: investigate (virtual) experiments in cases where the boundary conditions are either unreachable or beyond control.
- Assess the accuracy of models and theories through benchmarking: the closer a model or theory comes to replicating established experimental findings, the more reliable its predictions become.

Founding visions of computational science

Abel Prize

The "third leg" quotation (1986)

"During its spectacular rise, the computational has joined the theoretical and the experimental branches of science, and is rapidly approaching its two older sisters in importance and intellectual respectability." – Peter D. Lax, in *J. Stat. Phys.*, 43:749-756

The "Grand Challenge" quotation (1989)

"A few key areas with both extreme difficulties and extraordinary rewards for success should be labelled as the "Grand Challenges of Computational Science." [They] should define opportunities to open up vast new domains of scientific research, domains that are inaccessible to traditional experimental or theoretical models of investigation." – Kenneth G. Wilson, in *Future Generation Computer Systems*, 5:171-189

Nobel Prize

The computer as a "scientific instrument."

What is High Performance Computing (HPC)?

CODATA – RDA Data Schools

- It's not a fixed description; it varies depending on the perspective:
  - HPC refers to situations where the speed of obtaining an answer matters.
  - HPC emerges when one anticipates their problem growing significantly.
- As such, HPC can manifest on various platforms:
  - Personal workstations, desktops, laptops, even smartphones!
  - Supercomputers, Linux Clusters, Grids or cloud environments
- HPC also encompasses the idea of High-Productivity Computing.

Why would High-Performance Computing (HPC) be relevant to you?

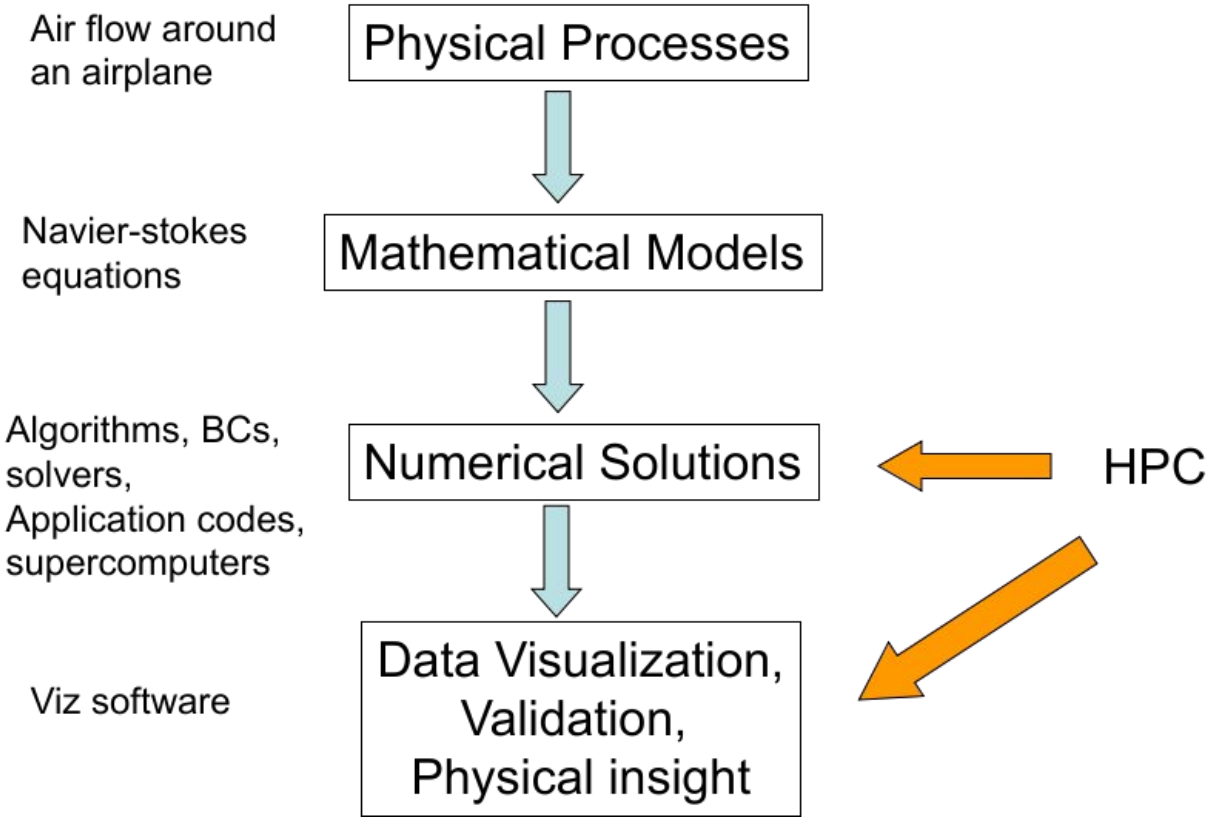
CODATA – RDA Data Schools

- Scientific computation is gaining significance across various research domains.
- Challenges are growing in complexity, demanding intricate software and collaborative efforts from diverse experts.
- HPC hardware is intricate, and application performance relies on multiple factors.
- Technology is also a means to enhance competitiveness.
- Proficiency in HPC offers opportunities for growth.

What and Why?

- What is high performance computing (HPC)?
  - The use of the most efficient algorithms on computers capable of the highest performance to solve the most demanding problems.
- Why HPC?
  - Large problems – spatially/temporally
  - 10,000 x 10,000 x 10,000 grid 10^12 grid points 4x10^12 double variables 32x10^12 bytes = 32 Tera-Bytes.
  - Usually need to simulate tens of millions of time steps.
- On-demand/urgent computing; real-time computing;
- Weather forecasting; protein folding; turbulence simulations/CFD; aerospace structures; Full-body simulation/ Digital human …

What and Why?


Air flow around an airplane
Physical Processes
Navier-stokes equations
Mathematical Models
Algorithms, BCs, solvers, Application codes, supercomputers
Numerical Solutions
HPC
Viz software
Data Visualization, Validation, Physical insight

What and Why?

- Parallel programming is more difficult than it's sequential counterpart
- However we are reaching limitations in uniprocessor design
  - Physical limitations to size and speed of a single chip
  - Developing new processor technology is very expensive
  - Some fundamental limits such as speed of light and size of atoms
- Parallelism is not a silver bullet
  - There are many additional considerations
  - Careful thought is required to take advantage of parallel machines

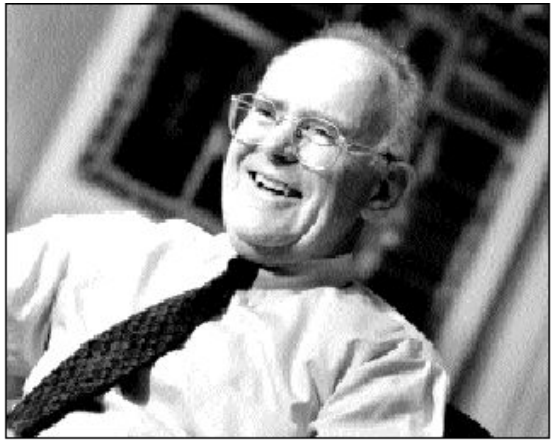
How Fast is my CPU?

- Flops, or Flop/s: FLoating-point Operations Per Second
  - Count of multiply-add (FMA) operations per cycle:
  - Constrained by the architecture itself.
  - Also influenced by the supported instruction set.
- MFlops: MegaFlops, 10^6 flops
- GFlops: GigaFlops, 10^9 flops, home PC
- TFlops: TeraFlops, 10^12 flops,
- PFlops: PetaFlops, 10^15 flops, present-day supercomputers (www.top500.org)
- EFlops: ExaFlops, 10^18 flops, fastest computer today:
  - Oak Ridge National Laboratory: 1.194 EFlop/s

How fast is computational power growing?

2X transistors/Chip Every 1.5 years

Called "Moore's Law"



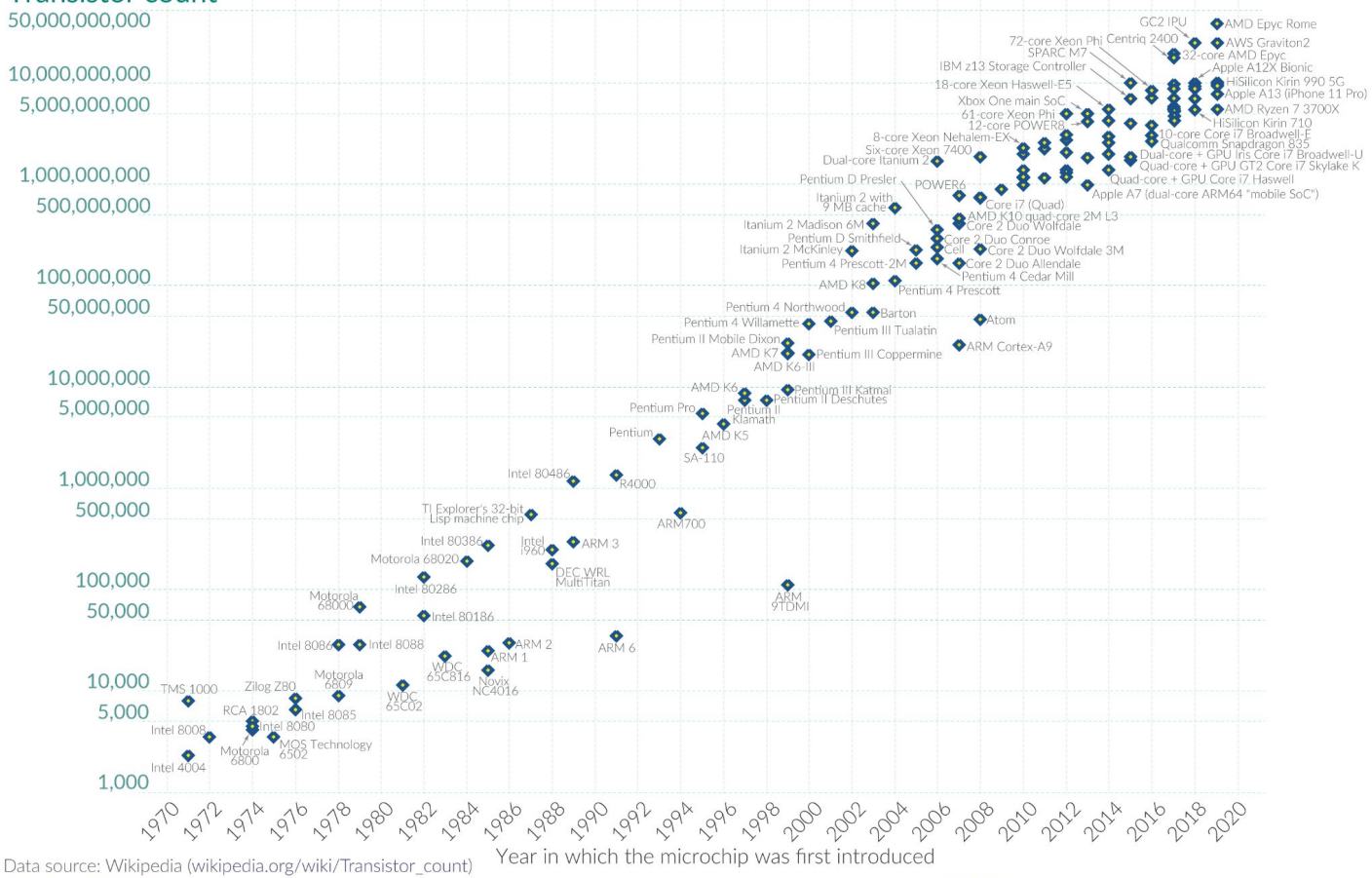
Gordon Moore (co-founder of Intel) predicted in 1965 that the transistor density of semiconductor chips would double roughly every 18 months.

Moore's Law: The number of transistors on microchips doubles every two years

Our World in Data

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years.
This advancement is important for other aspects of technological progress in computing – such as processing speed or the price of computers.

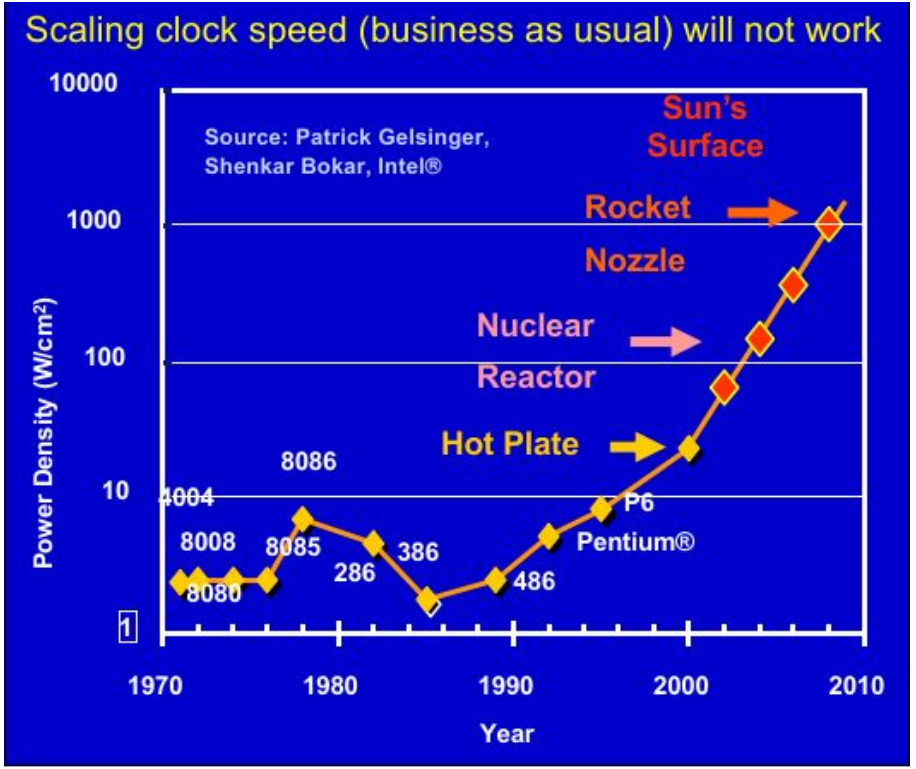
CODATA – RDA Data Schools



Data source: Wikipedia (wikipedia.org/wiki/Transistor_count)
OurWorldinData.org – Research and data to make progress against the world's largest problems.

Power Density Curve

- High performance serial processors waste power
- Speculation, dynamic dependence checking, etc. burn power
- Implicit parallelism discovery
- More transistors, but not faster serial processors



Parallelism 101

- There are two primary motivations for developing a parallel program:
  - Gaining access to greater memory capacity (scaling up, increasing memory)
  - Decreasing the time required for solving (improving speed)

The Third Pillar of Science

- Traditional scientific and engineering method:
  1. Do theory or paper design
  2. Perform experiments or build system
- Limitations:
  1. Too difficult—build large wind tunnels
  2. Too expensive—build a throw-away passenger jet
  3. Too slow—wait for climate or galactic evolution
  4. Too dangerous—weapons, drug design, climate experimentation

The Third Pillar of Science

- Computational science and engineering paradigm:
  1. Use computers to simulate and analyze the phenomenon
- Based on known physical laws and efficient numerical methods
- Analyze simulation results with computational tools and methods beyond what is possible manually

Motivation Example

Global Climate Modeling Problem

- Problem is to compute:

  f(latitude, longitude, elevation, time) "weather" = (temperature, pressure, humidity, wind velocity)

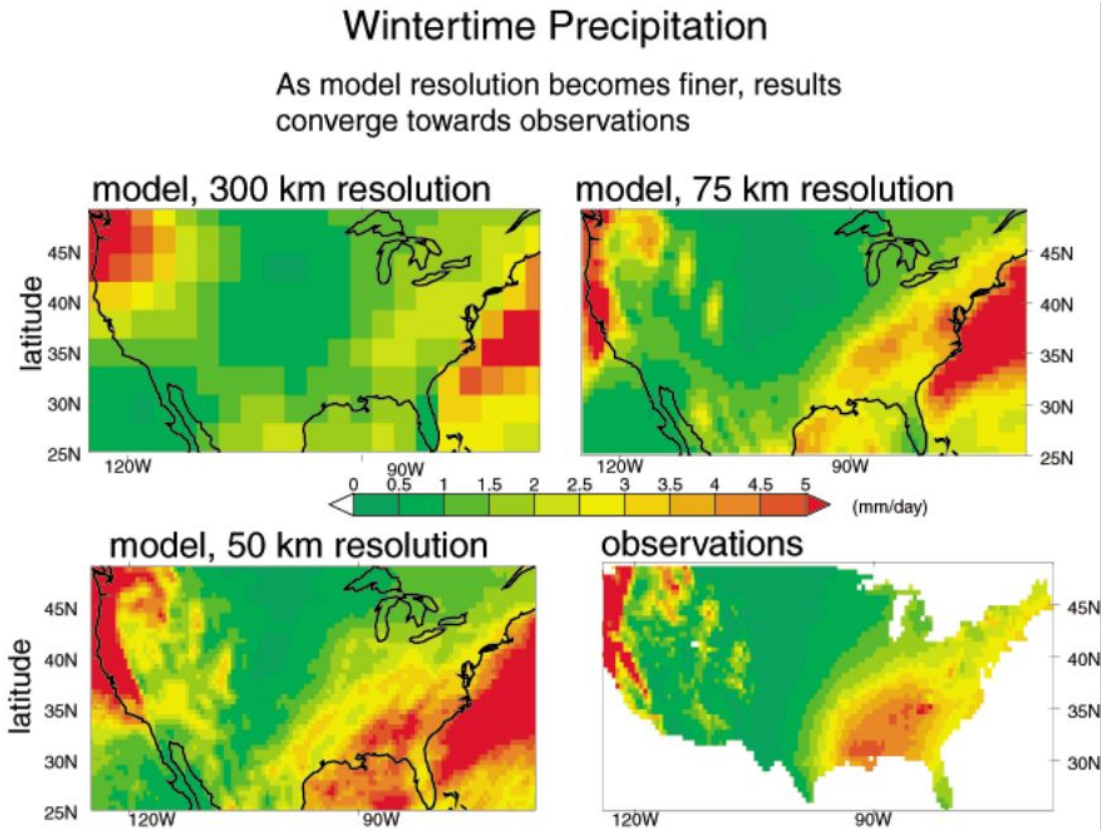
- Approach:
  - Discretize the domain, e.g., a measurement point every 10 km
  - Devise an algorithm to predict weather at time t+dt given t

Motivation Example

- One piece is modeling the fluid flow in the atmosphere
  - Solve Navier-Stokes equations;
  - Roughly 100 Flops per grid point with 1 minute timestep
- Computational requirements:
  - To match real-time, need 5 x 10^11 flops in 60 seconds = 8 Gflop/s
  - Weather prediction (7 days in 24 hours) 56 Gflop/s
  - Climate prediction (50 years in 30 days) 4.8 Tflop/s
  - To use in policy negotiations (50 years in 12 hours) 288 Tflop/s
- To double the grid resolution, computation is 8x to 16x

Motivation Example

High Resolution Climate Modeling on NERSC-3 – P. Duffy, et al., LLNL



What can clusters do?

- Serve to offload code execution from your laptop/workstation
  - Code that runs too long or needs too much memory or disk space
- Clusters are particularly useful for executing parallel code
  - on one compute node
  - on multiple compute nodes at once
- Note on speed of execution:
  - The compute nodes have similar architecture to your desktop - they are not much faster
  - the main advantage of cluster computing lies in parallel code execution

How do we evaluate the improvement?

- *Speedup*:
  - number that measures the relative performance of two systems processing the same problem

$$Speedup = \frac{t(1)}{t(N)}$$

- Where *t(1)* is the time for running the software using 1 processor, and
- *t(N)* is the computational time running the same software with *N* processors;

How do we evaluate the improvement?

CODATA – RDA Data Schools

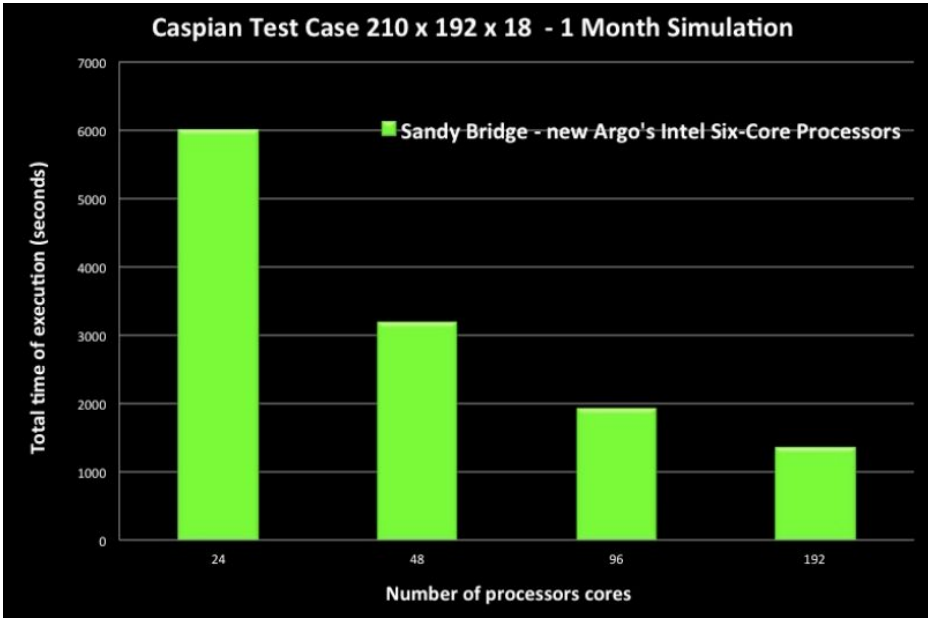
Speedup

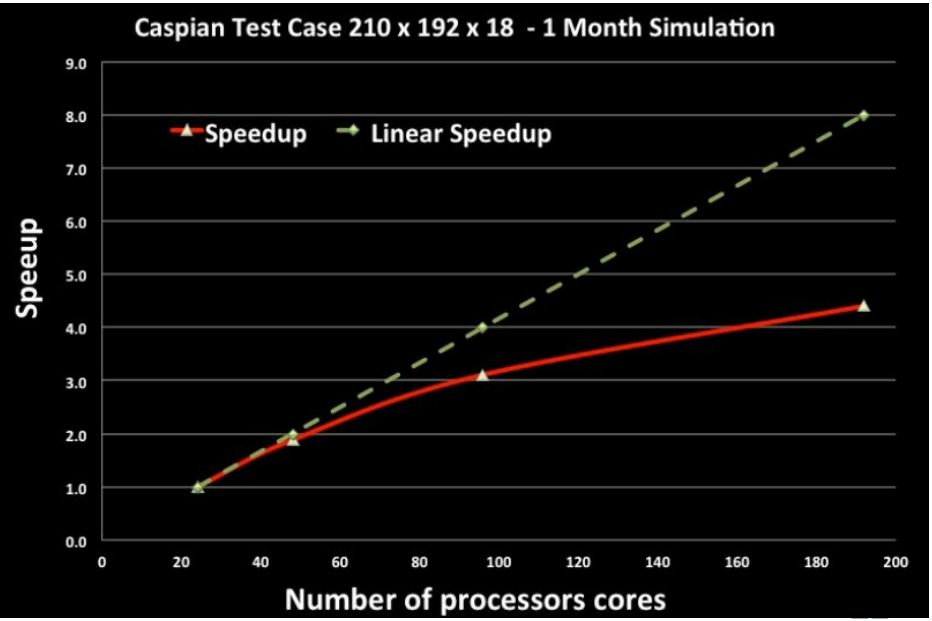
- Ideally, we would like to have a linear speedup that is equal to the number of processors (*speedup = N*),
  - Every processor would be contributing 100% of its computational power.
  - Unfortunately, this is a very challenging goal for real world applications to attain.
- Upper limit when *N* approaches infinity: communication starts to dominate;

How do we evaluate the improvement?

CODATA – RDA Data Schools

Speedup





How do we evaluate the improvement?

Amdahl's Law

- In 1967, Amdahl highlighted that the potential speedup of a parallel program is restricted by the proportion of the software's sequential portion that cannot be effectively parallelized.

$$Speedup = \frac{1}{(s + \frac{p}{N})}$$

- *s* is the proportion of execution time spent on the serial part,
- *p* is the proportion of execution time spent on the part that can be parallelized, and
- *N* is the number of processors

How do we evaluate the improvement?

CODATA – RDA Data Schools

Amdahl's Law

- For a fixed problem, the upper limit of speedup is determined by the serial fraction of the code
- Find a "sweet spot" that allows the computation to complete in a reasonable amount of time

How do we evaluate the improvement?

Parallelization efficiency

- Ratio between the actual speedup and the ideal speedup obtained when using a certain number of processors;

$$efficiency = \frac{T(1)N_1}{T(N_p)N_p}$$

- *T(1)*: time needed to complete a serial task
- *T(Np)*: time to complete the same unit of work with *Np* elements
- *N1*: no of processors needed to complete a serial task
- *Np*: no of processors needed to complete a parallel task.

How do we evaluate the improvement?

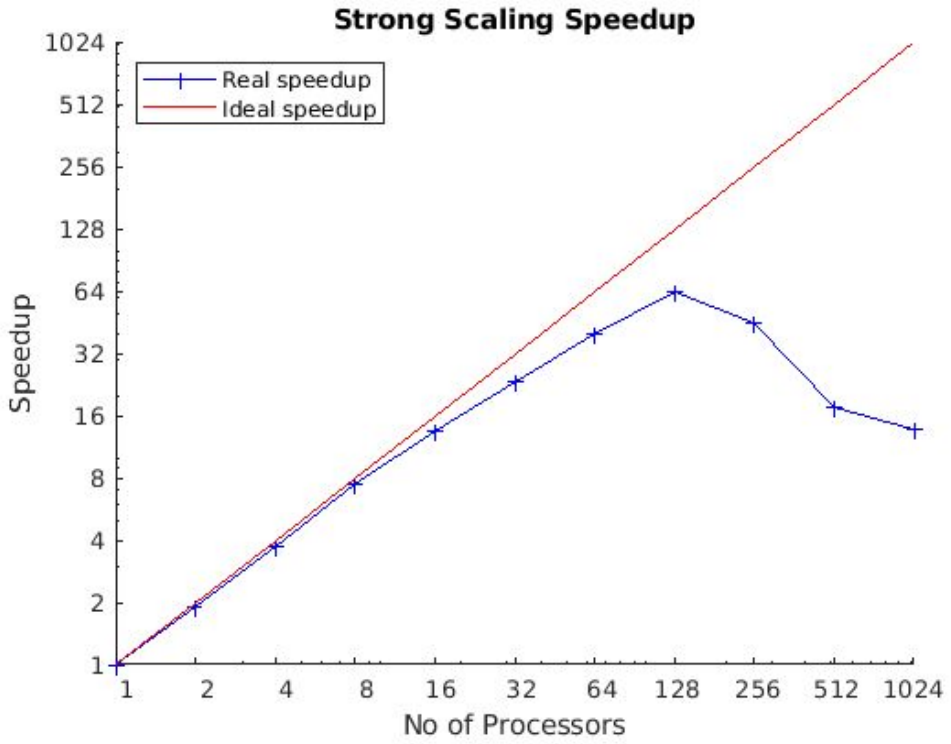
CODATA – RDA Data Schools

Parallelization efficiency - conjugate gradient example

Noctua, PC2, Paderborn	
CPUs per node	2 (20 cores per CPU)
CPU type	Intel Xeon Gold 6148
main memory per node	192 GiB
interconnect	Intel Omni-Path 100 Gb/s
accelerators used (such as GPUs)	no
number of MPI processes per node	40
number of threads per MPI process (e.g. OpenMP threads)	1

#problem size (N x N) = 1600000000, where N is Matrix size and N = 40000 for all different processor numbers			
#Processors Np	#time in seconds T	(Amdahl's law) #speeedup = (T(1)/T(Np))	#efficiency = (T(1)xN1 / T(Np)xNp)
1 (1 node)	64.424242	1	1
2 (1 node)	33.901724	1.90	0.95
4 (1 node)	17.449995	3.69	0.92
8 (1 node)	8.734972	7.38	0.92
16 (1 node)	4.789075	13.45	0.84
32 (1 node)	2.749116	23.43	0.73
64 (2 nodes)	1.627157	39.59	0.62
128 (4 nodes)	1.017307	63.33	0.49
256 (7 nodes)	1.436728	44.84	0.18
512 (13 nodes)	3.689217	17.46	0.03
1024 (26 nodes)	4.709213	13.68	0.01
2048 (52 nodes)	21.462228	3.00	0.001

How do we evaluate the improvement?



How do we evaluate the improvement?

CODATA – RDA Data Schools

Scalability

- When assessing the scalability of our problem, we focus on two key aspects:
  - How much speedup is achieved as we increase the number of processes for a constant problem size (strong scaling).
  - How the application performs when we increase the problem size while maintaining a consistent workload per processor.

HPC terminology

- *Job = your program on the cluster*
- *Submit job = instruct the cluster to run your program*
- *Node = compute node = group of cores that can access the same memory (I may inadvertently also say a computer or a machine)*
- *Memory = main memory or RAM - fast memory directly connected to the processor*
- *Core = the basic computation unit inside a processor that can run a single process*
- *Serial code = runs on one core*
- *Parallel code = program that runs on two or more cores*

Workload Types

- High-throughput tasks – large number of relatively small jobs
- Parallel jobs on a single node
- Parallel jobs on multiple nodes (from 2 to 270 nodes)
- Large single node memory jobs (up to TBs of memory)
- Jobs using GPUs (such as machine learning tasks)

Performance Gain

- A key aim is to solve problems faster
  - To improve the time to solution
  - Enable new a new scientific problems to be solved
- To exploit parallel computers, we need to split the program up between different processors
- Ideally, would like program to run *N* times faster on *N* processors
  - Not all parts of program can be successfully split up
  - Splitting the program up may introduce additional overheads such as communication

Performance Gain

- How we split a problem up in parallel is critical
  1. Limit communication (especially the number of messages)
  2. Balance the load so all processors are equally busy
- Tightly coupled problems require lots of interaction between their parallel tasks
- Embarrassingly parallel problems require very little (or no) interaction between their parallel tasks
  1. E.g. Parameter Sweeps
- In reality most problems sit somewhere between two extremes

Scheduling Basics

- Software that implements a batch system on a HPC (cluster);
- Users do not run their calculations directly and interactively;
- Stores the batch jobs, evaluate their resource requirements and priorities, and distributes the jobs to suitable compute nodes;
- Make up the majority of HPC clusters (about 98%)
  - Most powerful, but also the most power consuming parts;

Scheduling Basics

- Interface for the users on the login nodes to send work to the compute nodes;
- Requires the user to ask the scheduler for time and memory resources and to specify the application inside a jobscript;
- Jobscript is submitted to the batch system via the scheduler:
  - Add the job to a job queue;
  - Decide when the job will leave the queue, and on which (part of the) back-end nodes it will run

Scheduling Basics

Example:

- if you demand less time than your job actually needs to finish, the scheduler will simply kill the job once the time allocated is up.
- specify more memory than there is available on the system, your job might be stuck in the queue forever.

Scheduling Basics

Example: