Title: 13/14 PSI - Computational Methods in Physics - Lecture 4

Date: Sep 06, 2013 09:00 AM

URL: http://pirsa.org/13090055

Abstract:

Multi-Tiered Strategy

- Tier 1: Personal Devices
  - Laptop, desktop
- Tier 2: Scientific Computing Environment (SCE)
  - Shared workstations with remote access
  - Virtual machines
- Tier 3: High Performance Computing (HPC)
  - HPC system “Titan”, access via batch queue
- Tier 4: Support for using outside resources
  - Sharcnet, Scinet, XSEDE, …

Research Technologies

- Liaison between IT support group and researchers
- Consulting and support e.g. for:
  - Questions on Latex, Mathematica, …
  - Need particular software package
  - Help with programming (C, C++, Fortran, Python, Perl, …)
  - Consulting on numerical algorithms
  - Collaboration on papers/projects

- consult@perimeterinstitute.ca,
  or office 350 (new wing, next to elevator)

Scientific Computing Environment

- 7"regular" Linux workstations, located in server room
- Names: compute, compute1, …, compute6
- Running Ubuntu, with many additional packages installed
  - 8 Intel cores with 2.4 GHz
  - 40 GByte RAM
  - Using Perimeter home directories
- Also 1 GPU workstation named "Nvidia", same software, has Nvidia GPU
- Access via ssh:
  - Login: ssh yourlogin@compute.pi.local
  - Copy files: access via network drive
- Mathematica: can set up a remote kernel

Software Packages on SCE

- Mathematica, Maple, Matlab, Magma, Sage

- Python, Perl
- GNU C, C++, Fortran
- CUDA, OpenCL
- Intel compilers (C, C++, Fortran, MKL, debugger/profiler)
- RNPL

- gnuplot
- SuperMongo
- DataVault
- VisIt (visualisation)

- Many more – ask if you need something

High Performance Computing

- Name: Titan
- Front end: titan.pi.local
- 29 compute nodes
  - 12 Intel cores with 3.47 GHz
  - 96 GByte RAM
- Using Perimeter home directories
- Fast data file system, available at /xfs1/yourlogin (if enabled)
  - also available from SCE
- Need to submit jobs to a queue to run

- Details: see my.pi.local, “Researcher Wiki”, search e.g. for “HPC” there

Accessing Perimeter from the Outside

- Two choices: VPN or ssh
- VPN (most convenient for laptops)
  - Needs software installed on your laptop (ask help desk about this)
  - When started, sets up a "tunnel" over the internet that lets the laptop access Perimeter as if inside Perimeter
- ssh (most convenient for other remote systems)
  - Need to enable remote ssh access at Perimeter for your account (ask help desk about this)
  - From outside, ssh yourlogin@perimeterinstitute.ca, then continue on e.g. to ssh yourlogin@compute.pi.local
  - To copy files, sftp yourlogin@perimeterinstitute.ca

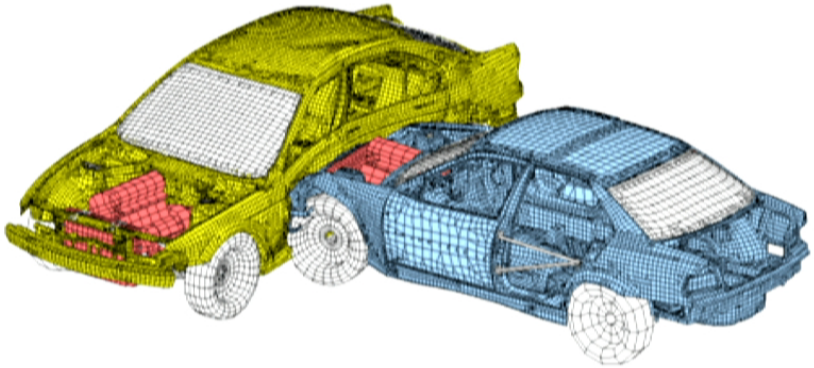
Running Persistent Jobs on SCE

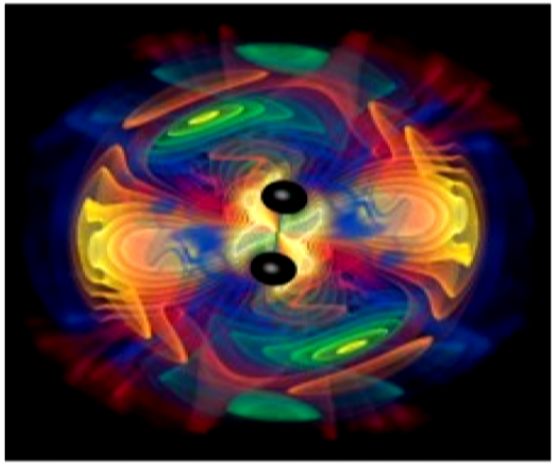
- Persistent job:
  - Add “nohup” when starting a job, e.g. “nohup ./mystuff &”
- Persistent terminal:
  - Use ssh to log in, then use “screen” to start a permanent session:
    - ssh yourlogin@compute.pi.local
    - screen [may need to wait a minute]
    - ./mystuff
  - Will continue to run if connection is lost. To re-connect, log in again, then
    - screen –r
- Mathematica:
  - Write .m script, run it via “math”, then see above

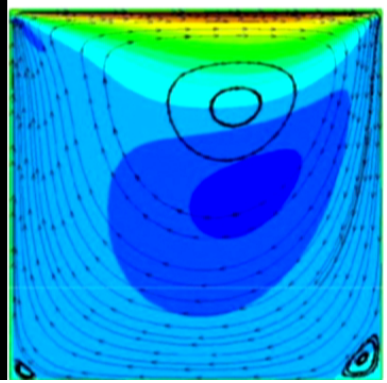
Introduction to Simulation Science

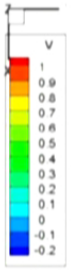
Erik Schnetter

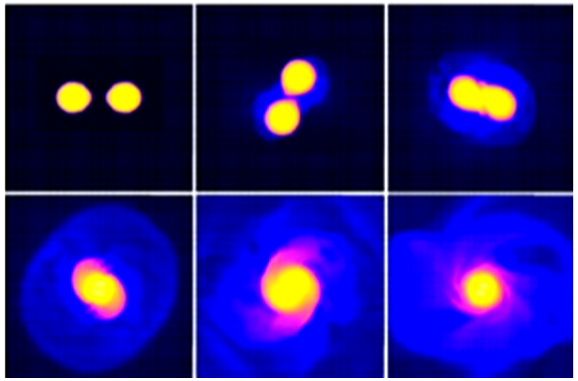
Simulations

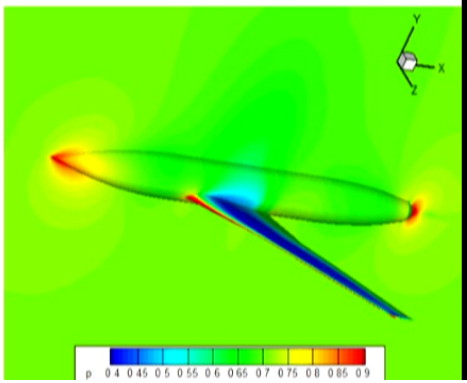








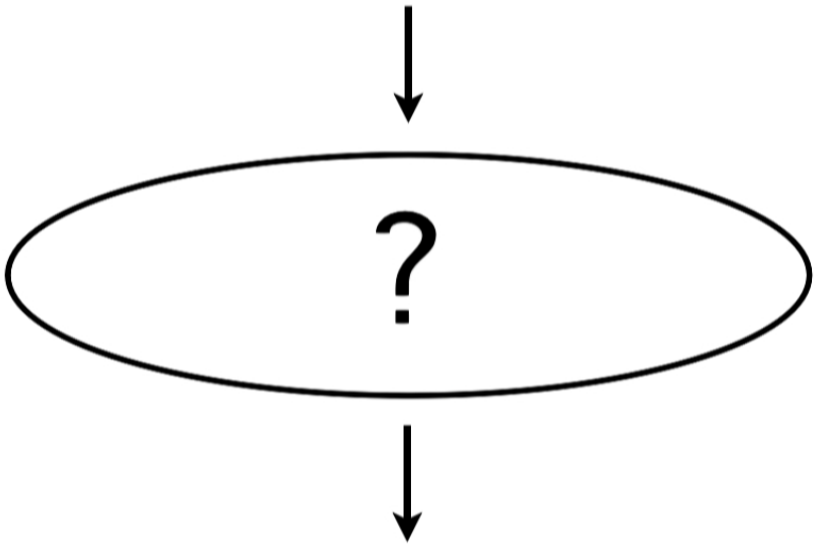




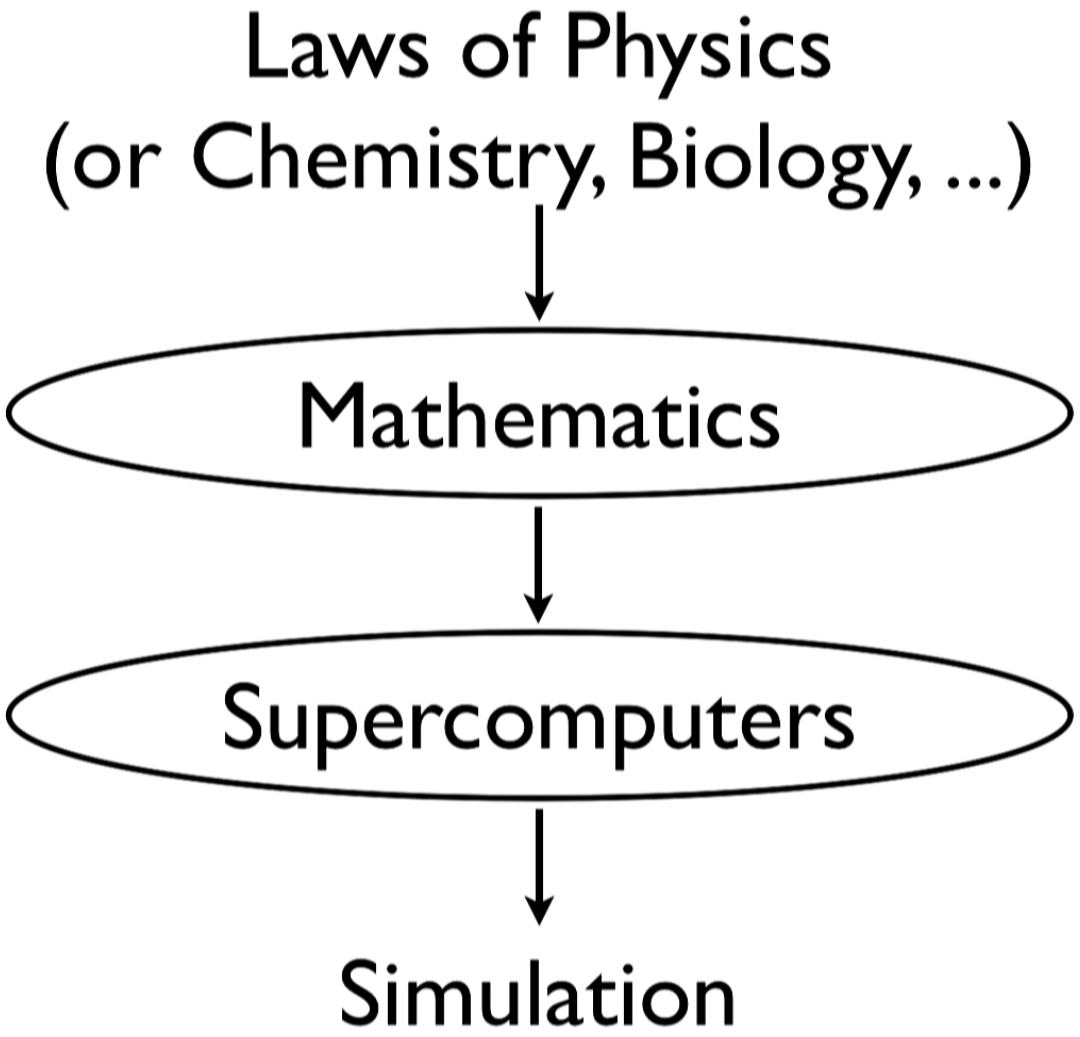
Why Use Simulations?

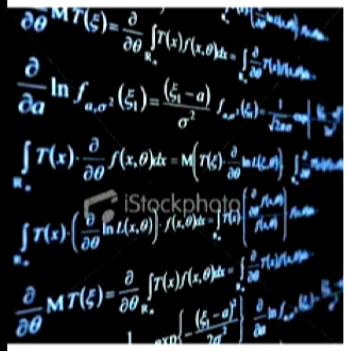
- Flame propagation in combustion engine: *understand* behaviour that is too fast or too small
- Hurricane modelling: *predict* behaviour
- Car crash testing: *engineer* better devices
- Video games: *create* a fantasy world similar to the real one

Laws of Physics
(or Chemistry, Biology, ...)



Simulation


Laws of Physics
(or Chemistry, Biology, ...)
Mathematics
Supercomputers
Simulation



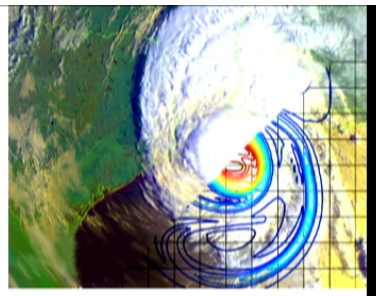
- The physics that is to be simulated is expressed in "the language of Mathematics"
  - Called *Scientific Computing* or *Numerical Analysis*

- The resulting systems of equations are solved on large computers
  - Called *Supercomputers* because they are as large and awkward as a supertanker

Systems and Equations

- The state of a system is described via variables (density, velocity, pressure, etc.)
- Laws of Physics can then often be described via *PDEs* (Partial Differential Equations)
- A PDE describes how a system is *changing* depending on its current *state*

Discretisation



- PDEs describe continuum systems (car body, water, air); these have infinitely many degrees of freedom
- Reduce complexity by approximation via a *discrete system* instead
- Compare e.g. pixels on a TV screen, surface triangulation for visualisation

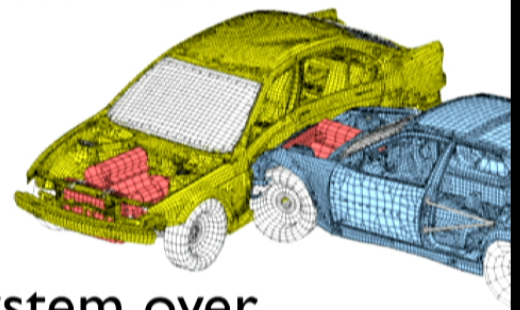
- Many possibilities:
- finite elements (e.g. small rigid triangles)
- finite volumes (e.g. small cubes)
- finite differences (sample solution on regular grid)
- particles (small chunks of matter)
- many more...

Discretisation Error

- Discretising is an approximation and thus leads to an error
- Can use a finer discretisation (higher *resolution*) to reduce this error
- *Order of Accuracy* describes how this error scales with the resolution , e.g.
  fourth order: $E = O(h^4)$
  doubling resolution reduces error by 16

Simulation Procedure

- Choose PDE that describes system well
- Discretise PDE
- Set up initial condition



- Follow each element of the system over many many tiny steps
- A simulation can have billions of elements with millions of steps, taking weeks of computing time

Caveat

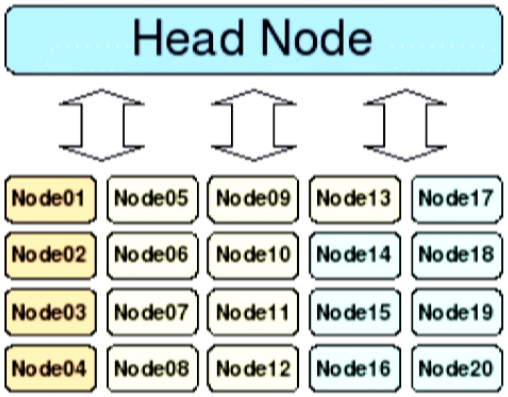
- Some systems are described not by PDEs but otherwise (e.g. coupled ODEs, discrete transitions)
- Sometimes not time evolution is interesting, but e.g. equilibrium configuration
- Usually (in real life), PDEs and initial conditions are only *approximations or guesses*, and simulation results *may not be reliable*

Fast vs. Large

- Supercomputers are not fast, they are large
- They are not interactive (like a notebook or workstation), they operate in batch mode
- Their hardware is complex -- I am going to describe the user's point of view only here

Remote Access

- Supercomputers are located in far away places, need to use ssh/gsissh to access
- Log in is to *front end (head node)* only, usually a large workstation
- Cannot (or should not) use front end to run simulations



File Systems

- Supercomputers need large file systems to store simulation data, often many 100 TByte
- For management and performance reasons, usually split into different parts with different properties
- Different on each supercomputer -- read documentation!

- Home directory: GBytes per user, many small files, backed up
- Data directory: TBytes per user, few large files, backed up, tape backend
- Scratch directory: no quota, few large files, often automatically deleted

Compute Nodes, Interconnect

- Most supercomputers have a *cluster* architecture with many compute nodes
- Each node has (4 to 32?) cores, similar to a large workstation
- Nodes are connected via a low-latency *communication network* (e.g. Infiniband)
- Overall system has (128 to >8,000?) nodes, or up to 100k cores
- My personal scale:
  <1k cores: small,
  <10k cores: medium
  >10k cores: large

Batch System

- Cannot (or should not) use compute nodes directly
- Need to submit *job* to *batch system*, requesting *N* nodes...
- ... wait (a few days?) ...
- ... then the job runs
- (... and then one discovers one's errors)

- There is a run time limit, often 24h or 48h
- ... which is inconvenient if one needs to run for 2 weeks: checkpoint/ restart
- Batch systems ensure that a supercomputer is not idle; there are always jobs waiting to be executed

Allocations

- Need to ensure fair use of supercomputer, prevent individual users from monopolising it
- Typically, an *allocation process* decides who can use how much of a supercomputer's time during a year (similar to writing a grant proposal)
- 1 CPU hour costs about 5 cents (10 cents on Amazon ECC)
- With this metric, Queen Bee produces about $270 worth of CPU time every hour

Software

- Installed/available software is system dependent, not just standard Unix systems
- Therefore cannot just install binaries, need to build software manually (or ask administrators to do that)
- HPC developers often prefer command line tools, don't use GUIs (which may not be available)
- (But: Eclipse and PTP may change this)

Parallel Computing

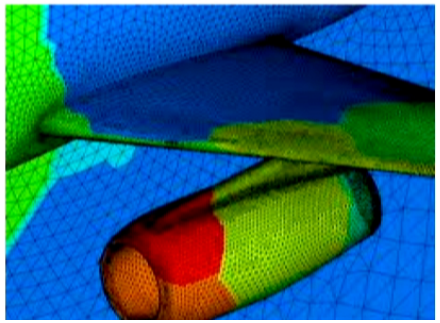
HPC History

- Before MPI: *Vector architectures*, e.g. Cray Y-MP (until ~1992)
- Each instruction acts on 100s or 1000s of data elements "simultaneously" (SIMD)

- Much more efficient than scalar processor (compare conveyor belt vs. hand assembly)
- Disadvantages: too inflexible for dynamic data structures, too expensive due to custom-designed (low-volume) hardware

MPI Programming

- Each process runs an independent copy of the program
- Each copy has a unique number assigned to it(0...*N*-1)
- The program needs to divide the total workload into N pieces, and assign one to each process



- The processes can talk to each other *only* by exchanging messages
- MPI hides low-level, system-dependent communication details from the programmer
- MPI messages are (by default) *ordered* and *reliable*

Examples of Real Life Message Passing

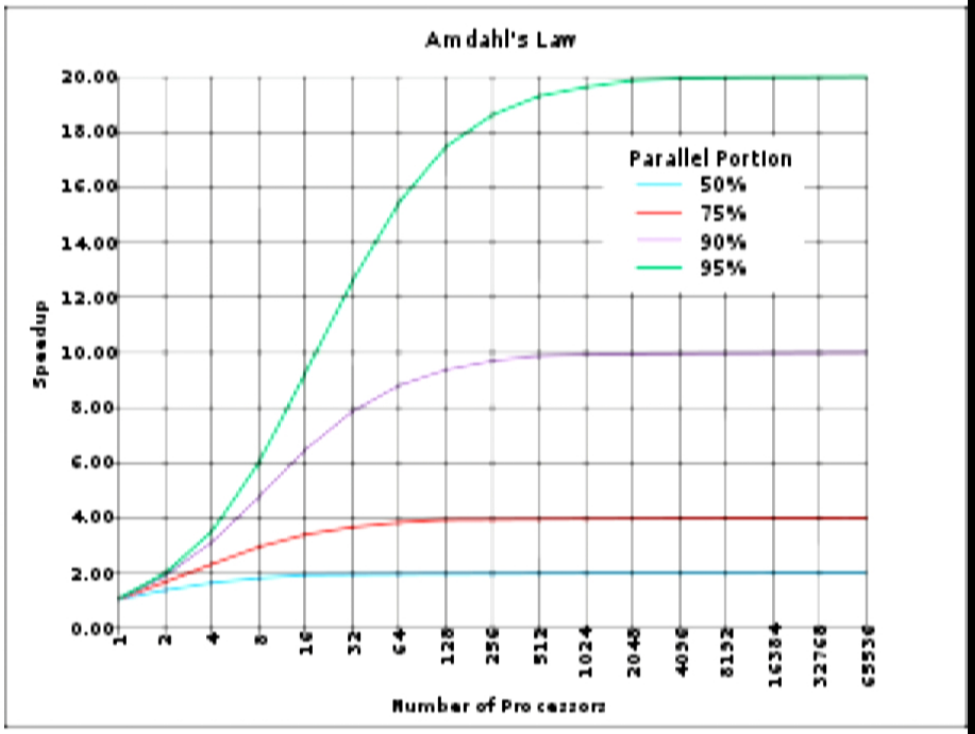
- Message Passing is very common, even in the real world; for example:
  - Letters via the post office
  - Email
  - Phone text messages
  - Newspapers
  - Chinese whispers game
  - Monopoly
- However, these are NOT message passing – they are *streams* or *interactive* instead:
  - Phone conversation
  - Watching TV
  - Google Docs
  - Charades game
  - WoW

Distributing Data Structures

- Example: Distributing a (large) array over multiple MPI processes
- Assuming: 50 elements, 5 processes, thus each process *owns* 10 elements
- Arrays support two operations: set-element and get-element; we need to implement these with MPI, so that each process can access non-local elements

Amdahl's Law

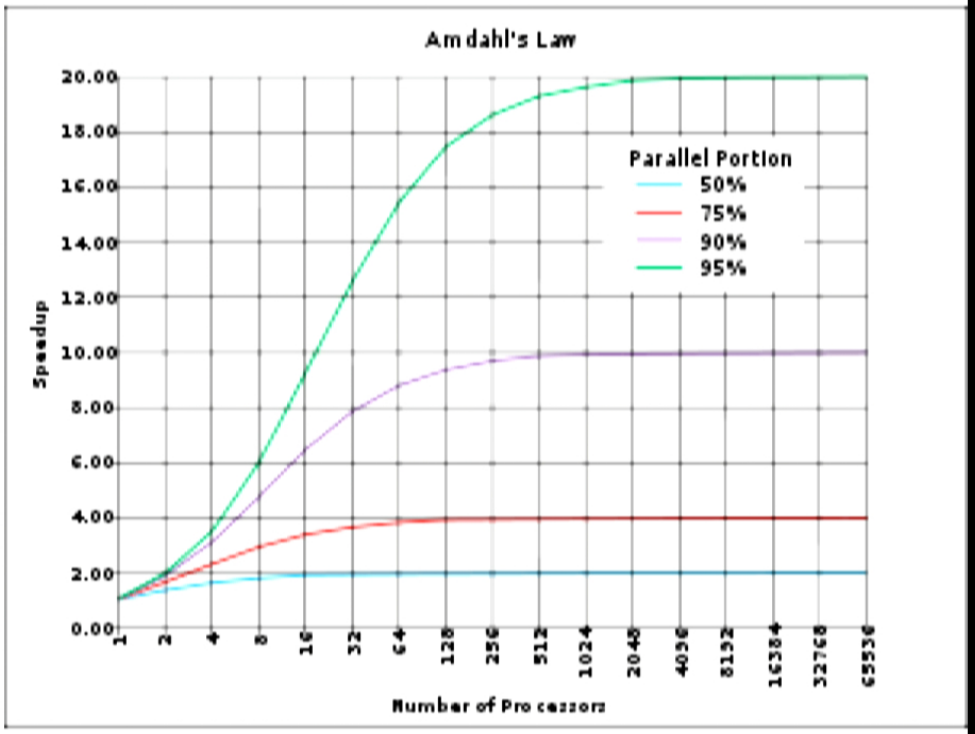
- When running on *N* processes, not necessarily *N* times as fast – overhead
- Overhead determines maximum possible parallel speedup



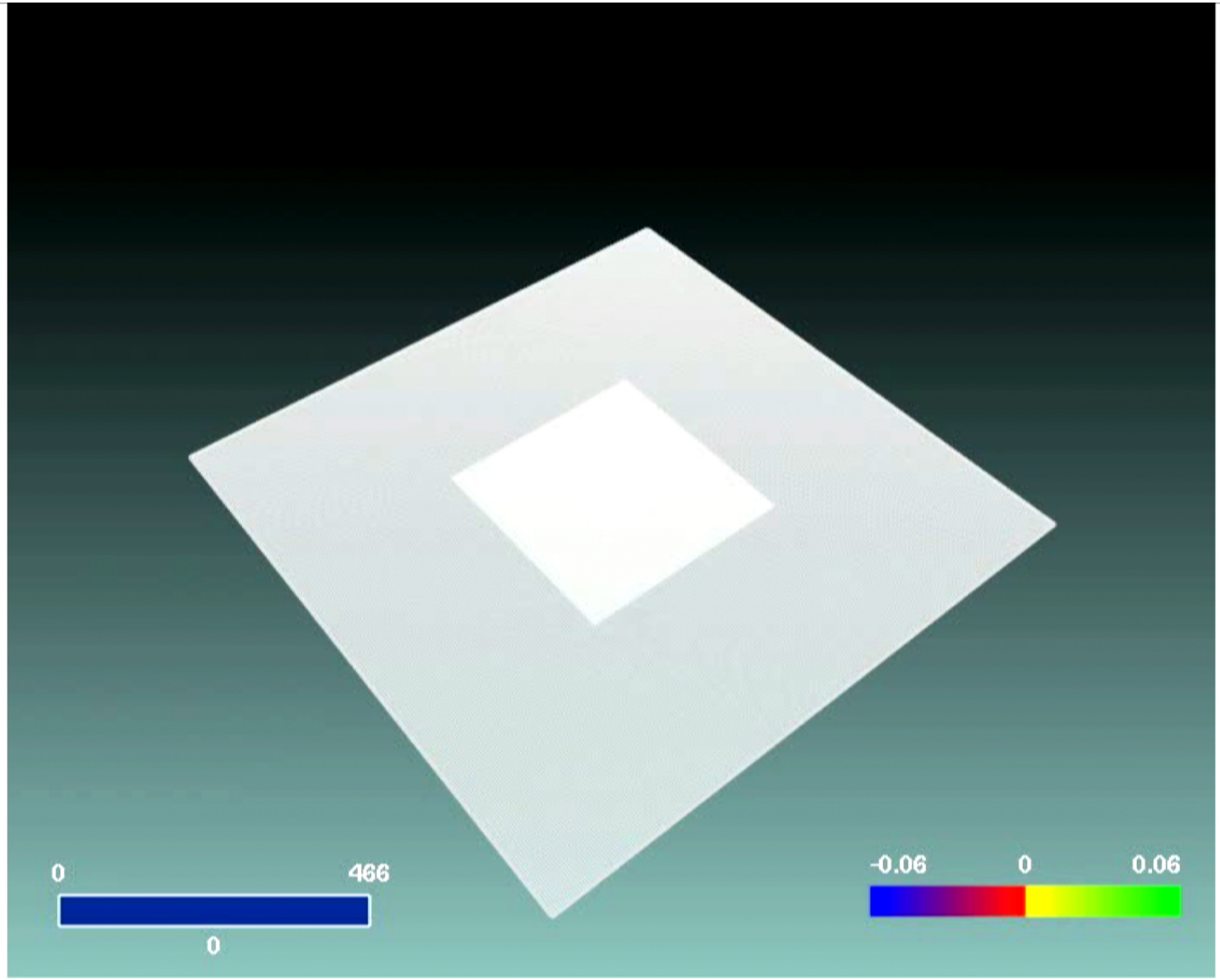
100,000-fold speedup requires >99.999% parallelisation

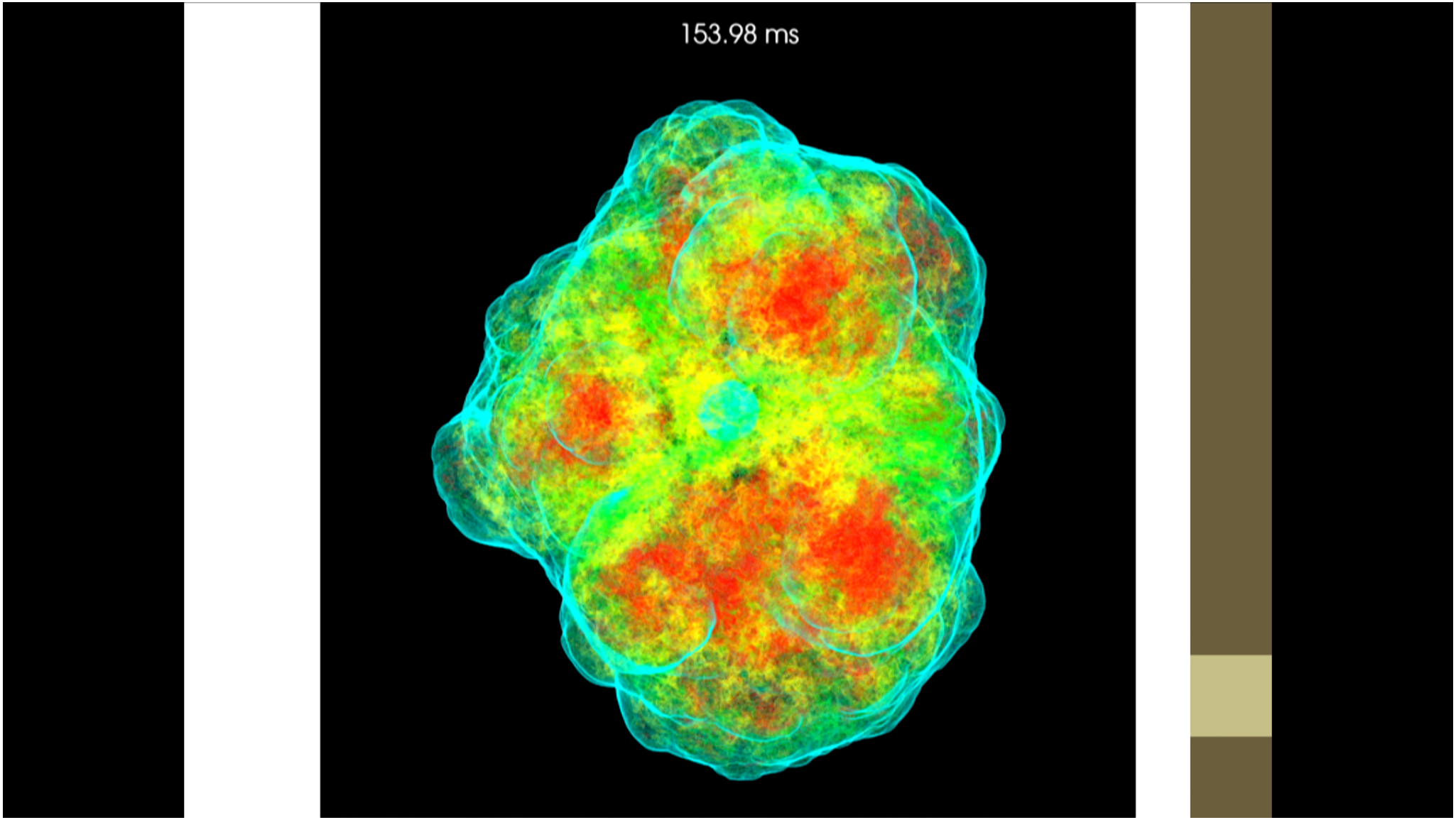
Amdahl's Law

- When running on *N* processes, not necessarily *N* times as fast – overhead
- Overhead determines maximum possible parallel speedup



100,000-fold speedup requires >99.999% parallelisation


0
466
0
-0.06
0
0.06


153.98 ms