Project 1
N-Body on GPU

High Performance Computing for Science and Engineering

October 1, 2013

CSElab
Computational Science & Engineering Laboratory
http://www.cse-lab.ethz.ch

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Administrative Notes

- Please contact one of the TAs if you need anything
  - Please avoid writing to Anna (our secretary)
  - Please avoid writing to Prof. Koumoutsakos
- Hand-in to your TA
  - If you have a special request, please ask by the end of the day
  - Please check again the list tomorrow evening for changes
- Exam
  - Tuesday, 17.12.2013, 14:00 - 17:00
  - Computer rooms

The Brutus Cluster

High Performance Computing for Science and Engineering

October 1, 2013

CSElab

Computational Science & Engineering Laboratory

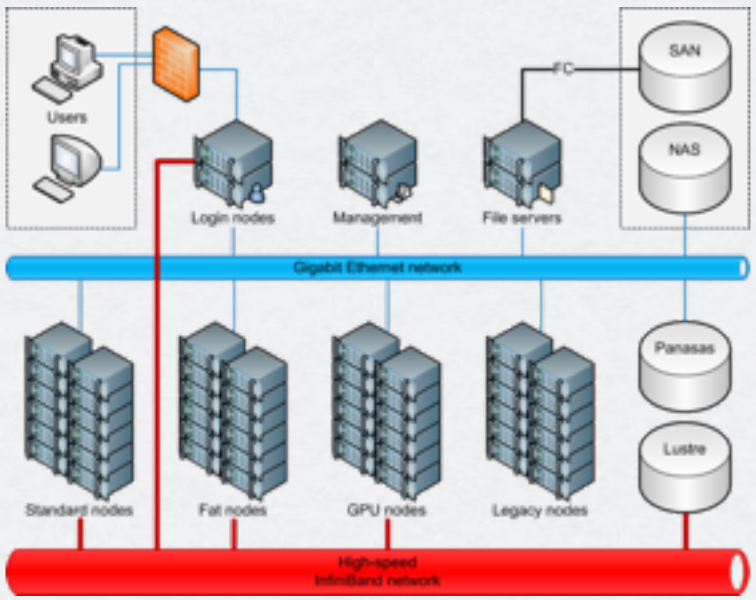
http://www.cse-lab.ethz.ch

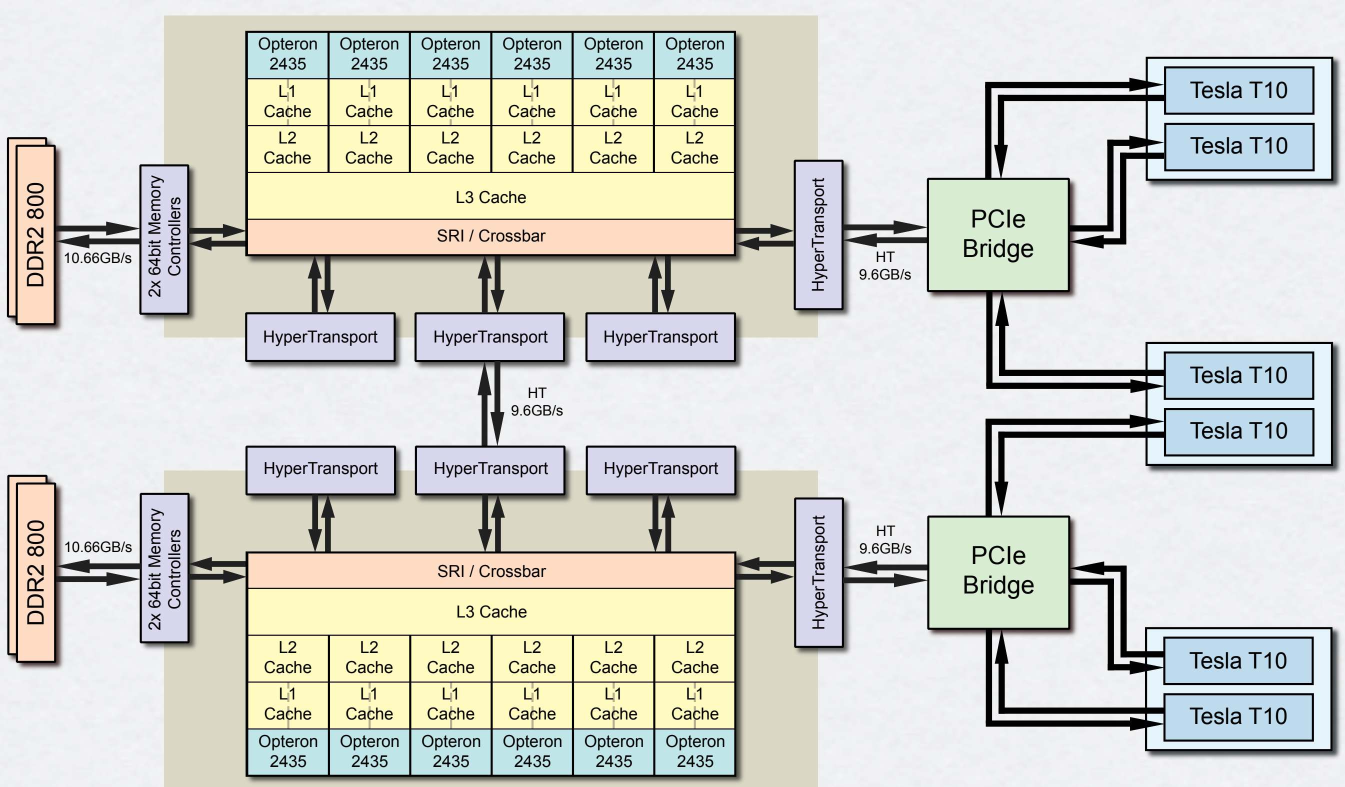
ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Anatomy of a Cluster

Brutus



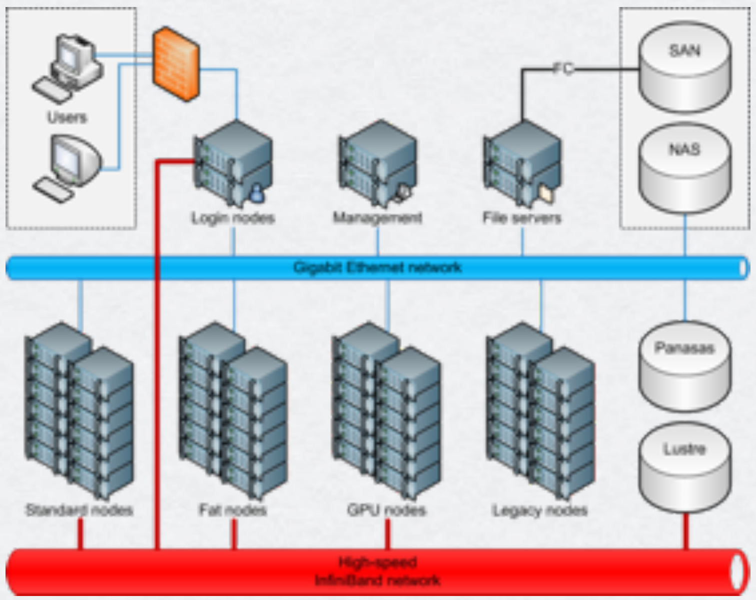


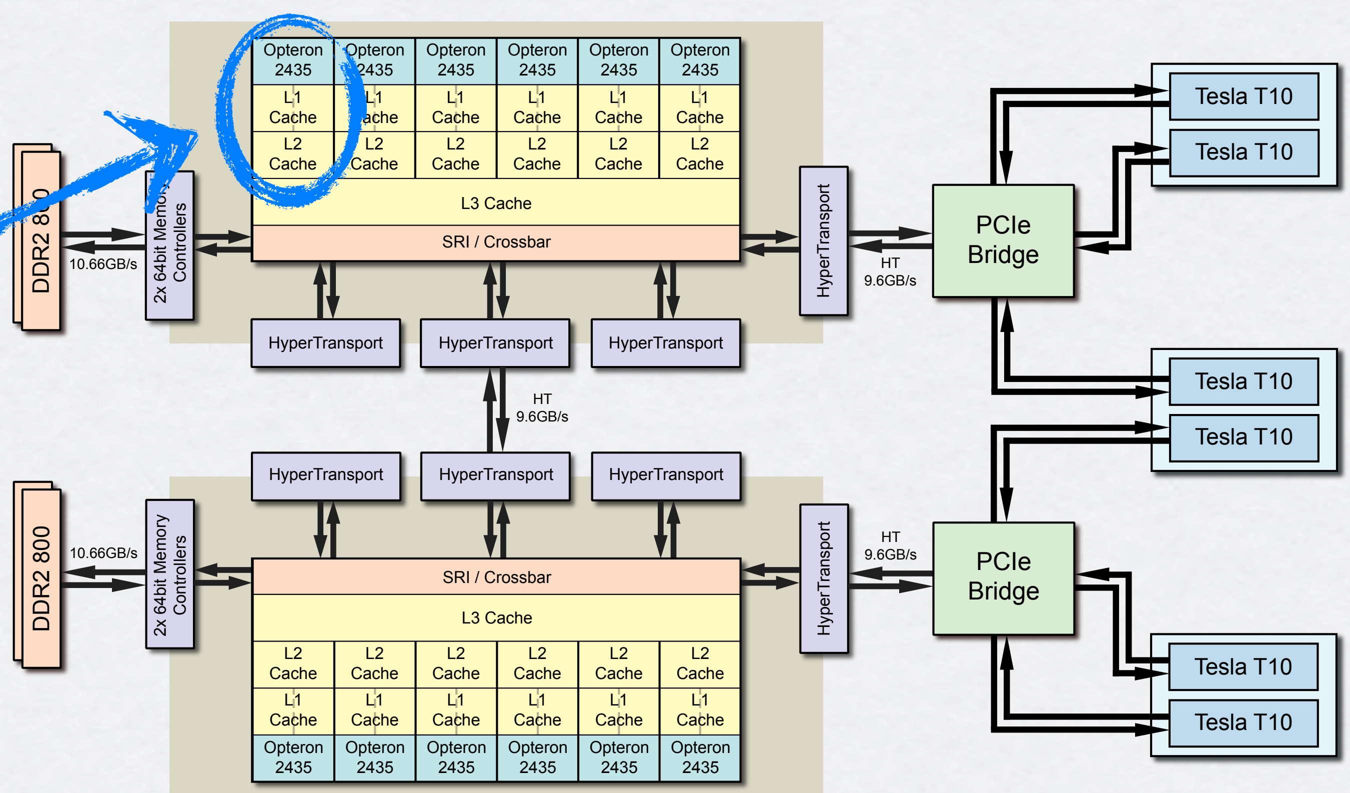
Details

- Effective bandwidth with 12 cores: 20GB/s (STREAM Benchmark)

Anatomy of a Cluster

Brutus





CPU core

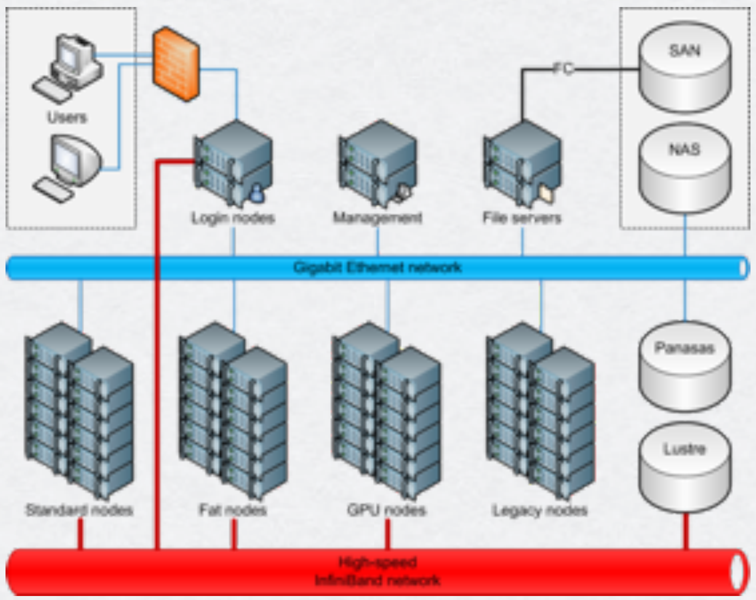
C++

Details

- Effective bandwidth with 12 cores: 20GB/s (STREAM Benchmark)

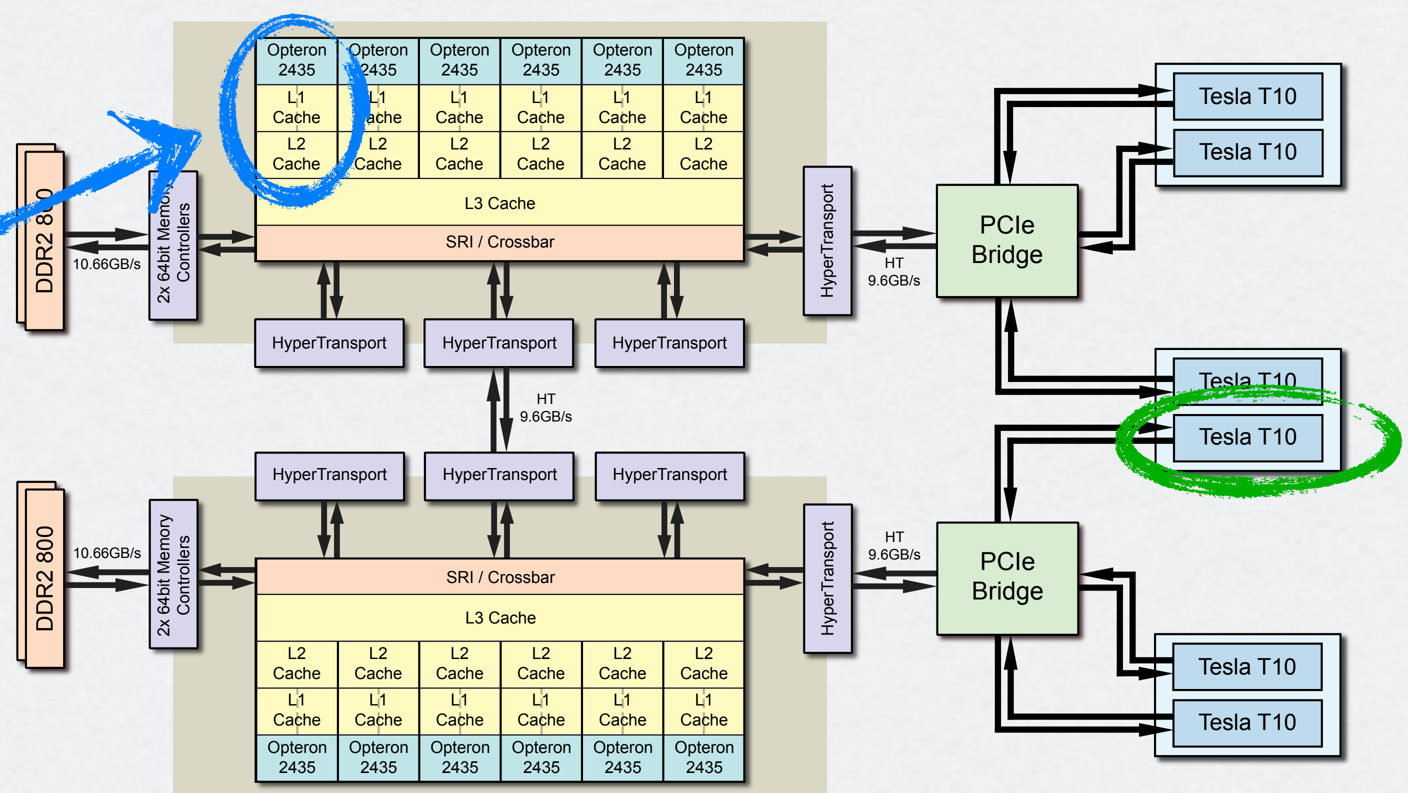
Anatomy of a Cluster

Brutus



CPU core

C++



GPUs

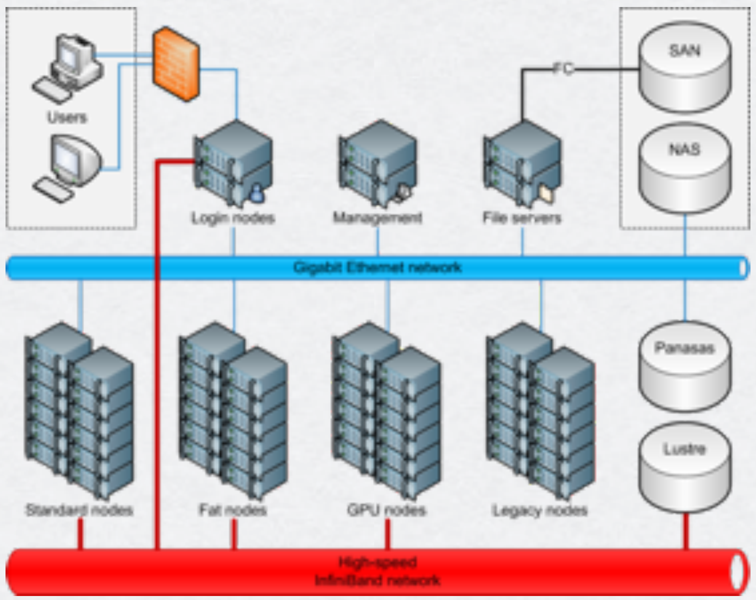
CUDA

Details

- Effective bandwidth with 12 cores: 20GB/s (STREAM Benchmark)

Anatomy of a Cluster

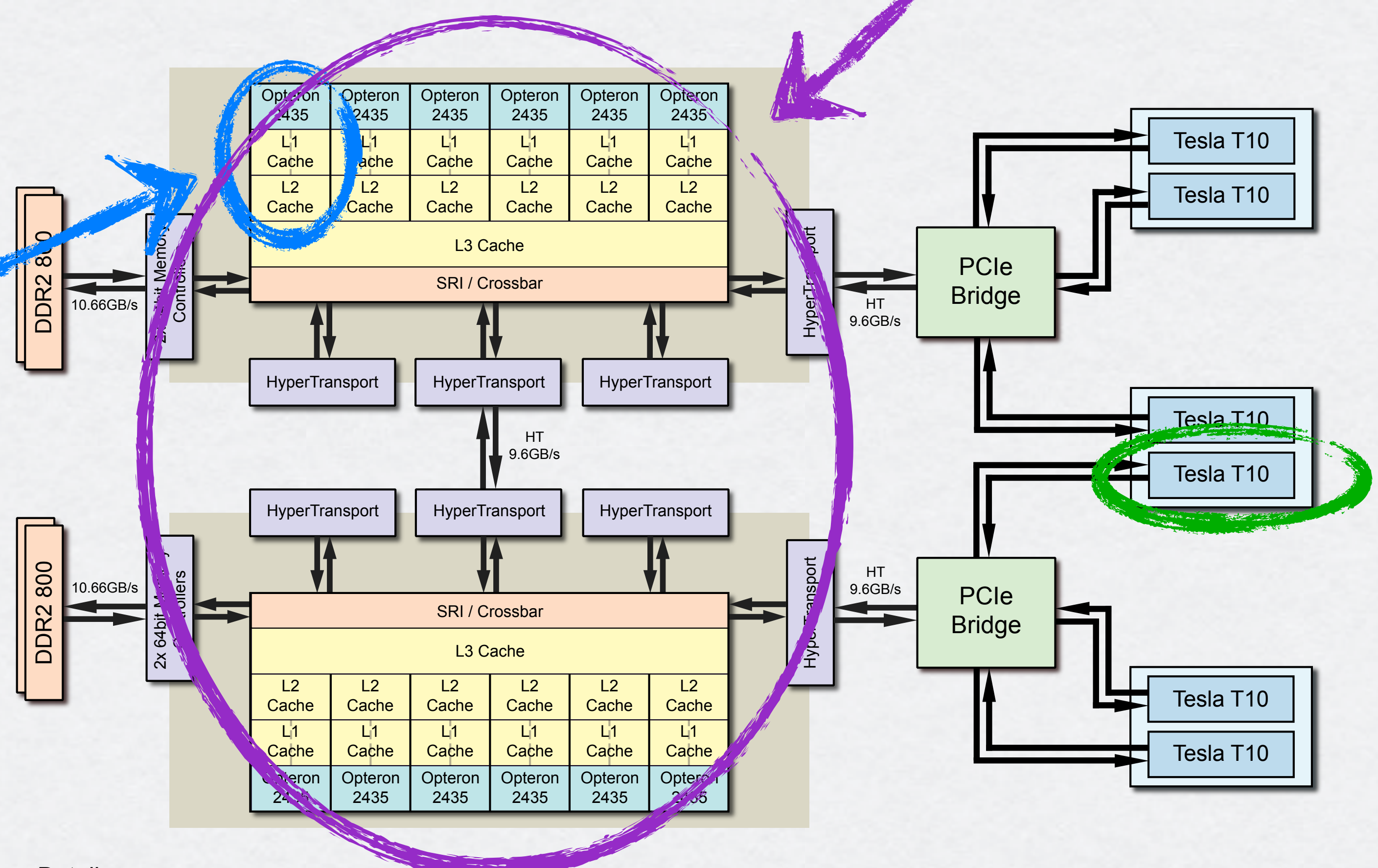
Brutus



Node: multiple processors

Shared Memory

OpenMP



CPU core

C++

GPUs

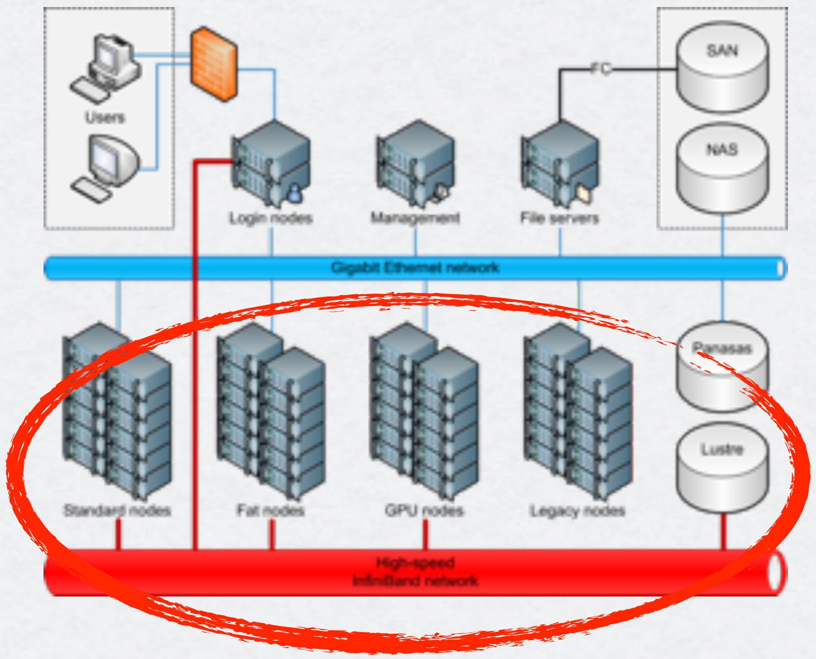
CUDA

Details

- Effective bandwidth with 12 cores: 20GB/s (STREAM Benchmark)

Anatomy of a Cluster

Brutus



Cluster: network of nodes

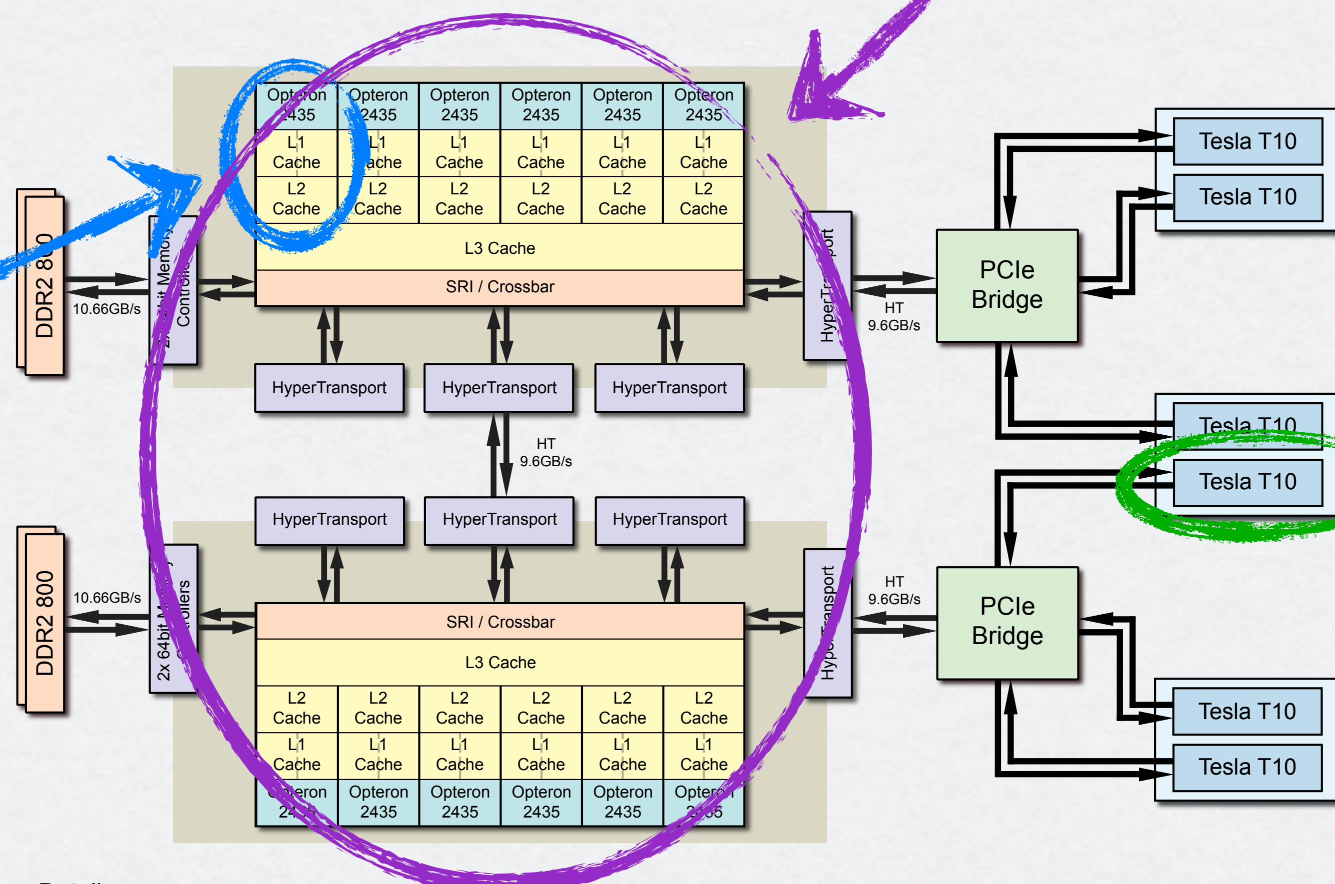
Distributed memory

MPI

Node: multiple processors

Shared Memory

OpenMP



CPU core

C++

GPUs

CUDA

Details

- Effective bandwidth with 12 cores: 20GB/s (STREAM Benchmark)

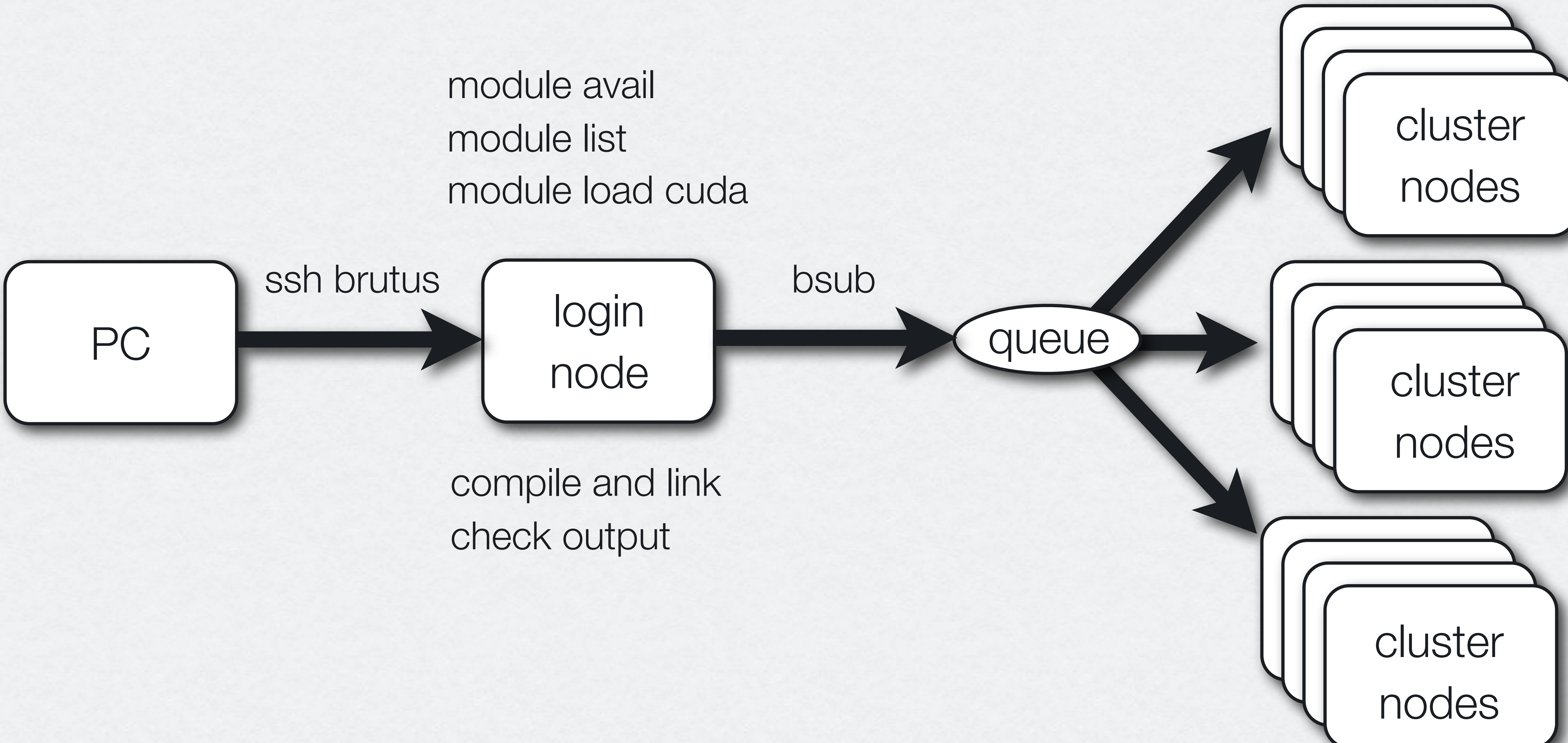
Brutus

- High performance cluster of ETH Zurich
- Composed of different kinds of compute nodes
  - 36x Nvidia Tesla C2050 (Fermi Architecture)
  - 120 nodes with 48 cores each
  - Many others (check brutuswiki.ethz.ch)

Accessing and Using Brutus

Request account (instructions on brutuswiki.ethz.ch)

- put “HPCSE I” as “project”



Account Request

The first thing you must do is to request for an account

- As STUDENTs attending a parallel computing class
- Brutus wiki / Contact us / Brutus account request
- https://www1.ethz.ch/id/services/list/comp_zentral/cluster/brutus_acc_req_pre_EN

Basic steps

1. Connect to a login node of Brutus

2. Copy or edit your program files

3. Compile your program

4. Submit a job / run your program on compute nodes

5. Check your job (status and output)

1. Connect

- ssh -Y <usename>@brutus.ethz.ch
  - -Y: Enables trusted X11 forwarding
  - Access to one of the Brutus login nodes

2. Develop

- Copy your files to Brutus, e.g.
  - scp code.tar.gz <username>@brutus.ethz.ch:*code.tar.gz*
- Use a text editor to write/modify your code
  - vi, emacs, nano, nedit

3. Compile

- You will need the appropriate programming tools and libraries to compile your code
  - By default, only the GNU compiler (gcc-4.4.7) is available
- Just load the environment module you need
- Examples
  - module load gcc/4.6 (newer version of gcc)
  - module load cuda (for GPU codes)
  - module list (shows loaded modules)
  - module avail (what is available)
  - module unload cuda (unloads a module)

3. Compile

- Compile your code and produce the executable
- Examples:
  - g++ cputest.cpp -o CPUexec
  - nvcc cudatest.cu -o CUDAexec

4. Submit your job

- The login nodes are used only for development
- The program must run on a compute node
- To do that, you must use the bsub command
- CPU code: bsub -W 00:10 -n 1 ./CPUexec
- GPU job: bsub -W 00:10 -n 1 -R gpu ./CUDAexec
- You can submit script files too: bsub -n 1 < myscript

5. Check your job

- Some useful commands
  - bqueues: displays information about queues
  - bjobs: displays information about jobs (bjobs -l -u <username>)
  - bkill <jobID>: kills a job
- Output files
  - lsf.o<jobID>: created in your working directory when the job finishes
    - includes information about your job (statistics, etc.) and the messages the program prints (standard output)

The N-Body Problem

High Performance Computing for Science and Engineering

October 1, 2013

CSElab

Computational Science & Engineering Laboratory

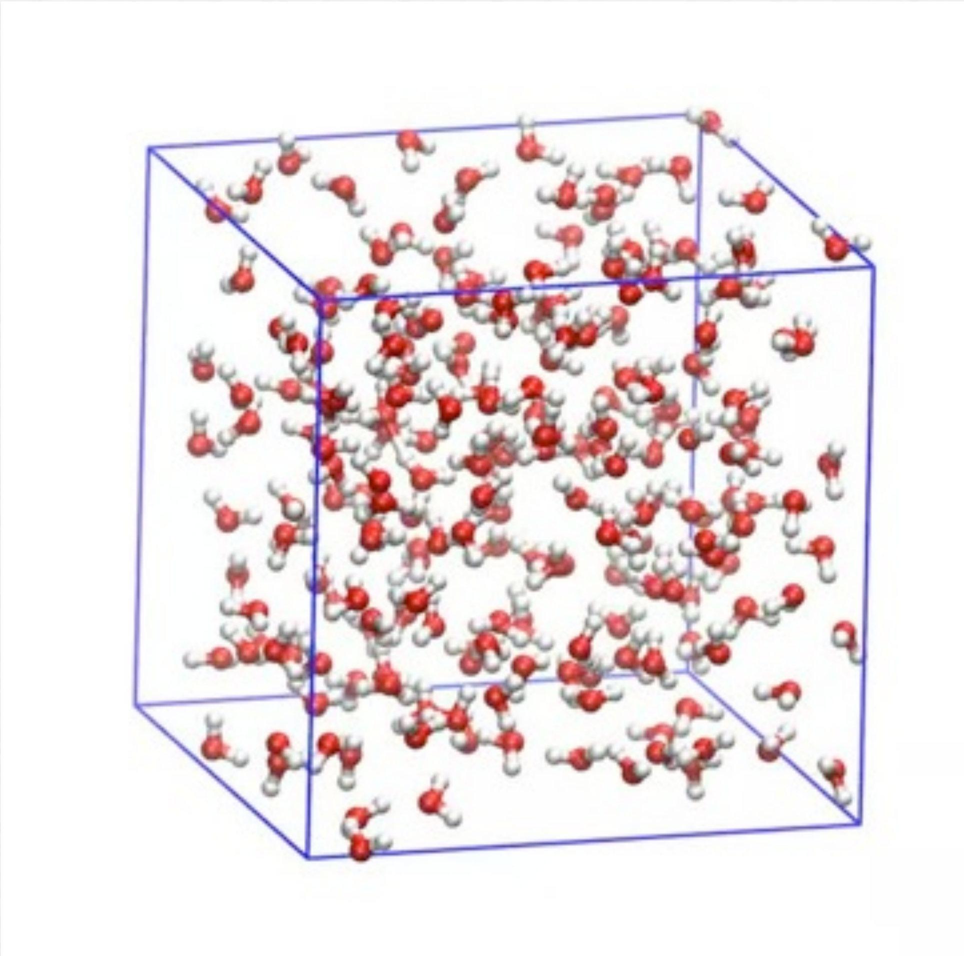
http://www.cse-lab.ethz.ch

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

N-Body - What is it About

- N-body problem is a classical problem of predicting the individual motions of a group of objects interacting with each other under some physical law

- This can be celestial bodies under gravitational forces, molecules under Lennard-Jones potential and so on



General formulation

- n bodies are represented as n points with masses $m_1, m_2, \ldots, m_n$
- Positions: $\mathbf{q}_1(t), \mathbf{q}_2(t), \ldots, \mathbf{q}_n(t)$
- Initial conditions for positions $\mathbf{q}_1(0), \mathbf{q}_2(0), \ldots, \mathbf{q}_n(0)$ and velocities $\dot{\mathbf{q}}_1(0), \dot{\mathbf{q}}_2(0), \ldots, \dot{\mathbf{q}}_n(0)$ are given
- Motion or evolution of $\mathbf{q}_j$ is given by a 2nd-order ODE:

$$\boxed{m_j \ddot{\mathbf{q}}_j = \sum_{\substack{k=1 \\ k \neq j}}^{n} \mathbf{F}\left(\mathbf{q}_k, \mathbf{q}_j\right)}$$

- By introducing the variables $\mathbf{x}_j = \mathbf{q}_j$ and $\mathbf{v}_j = \dot{\mathbf{q}}_j$ we can convert the 2nd-order ODE to a system of first order ODEs:

$$\boxed{\begin{pmatrix} \dot{\mathbf{x}}_j \\ m_j \dot{\mathbf{v}}_j \end{pmatrix} = \begin{pmatrix} \mathbf{v}_j \\ \sum_{k \neq \mathrm{J}} \mathbf{F}\left(\mathbf{x}_k, \mathbf{x}_j\right) \end{pmatrix}}$$

Discretization

- The equations have $6n$ degrees of freedom in 3D:

$$\mathbf{x}_j^i := \mathbf{q}_j(t_i) \qquad t_i = i \cdot \delta t$$

$$\mathbf{v}_j^i := \dot{\mathbf{q}}_j(t_i)$$

- Right-hand side is evaluated as follows:

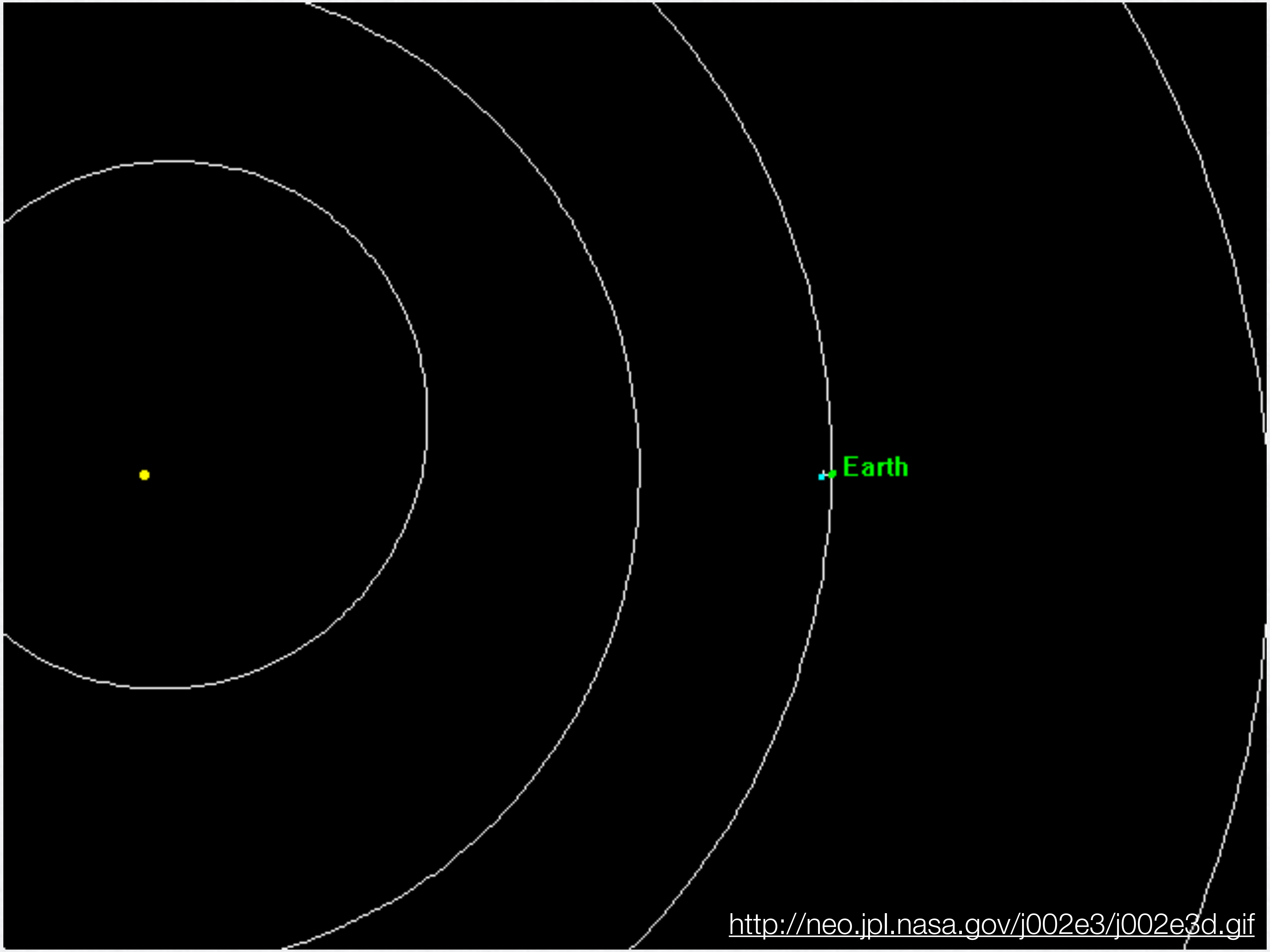
$$\mathbf{a}_j^i := \ddot{\mathbf{q}}_j(t_i) = \sum_{\substack{k=1 \\ k \neq j}}^{n} \mathbf{F}\left(\mathbf{q}_k, \mathbf{q}_j\right)$$

- Time integration with Velocity-Verlet scheme:

$$\mathbf{x}_j^{i+1} := \mathbf{x}_j^i + \delta t \mathbf{v}_j^i + \frac{1}{2}\delta t^2 \mathbf{a}_j^i$$

$$\mathbf{v}_j^{i+1} := \mathbf{v}_j^i + \frac{\delta t}{2}\left(\mathbf{a}_j^{i+1} + \mathbf{a}_j^i\right)$$

Gravitation



http://neo.jpl.nasa.gov/j002e3/j002e3d.gif

Gravitation

- Gravitational force depends on the distance $\mathbf{r}$ between two point masses m and M as

$$\mathbf{F}(\mathbf{r}) = G\frac{mM\mathbf{r}}{\left|\mathbf{r}\right|^3}$$

- So the resulting expression for accelerations in gravitational N-body is as follows:

$$\boxed{m_j\ddot{\mathbf{q}}_j = G\sum_{\substack{k=1\\k\neq j}}^{n}\frac{m_j m_k\left(\mathbf{q}_k - \mathbf{q}_j\right)}{\left|\mathbf{q}_k - \mathbf{q}_j\right|^3}}$$

Molecular Dynamics

- Force depends on the distance $\mathbf{r}$ between two point molecules as

$$\mathbf{F}(r) = 24\,\varepsilon \frac{\mathbf{r}}{r^2}\left(\frac{2\sigma^{12}}{r^{12}} - \frac{\sigma^6}{r^6}\right)$$

- So the resulting expression for accelerations in molecular N-body is as follows:

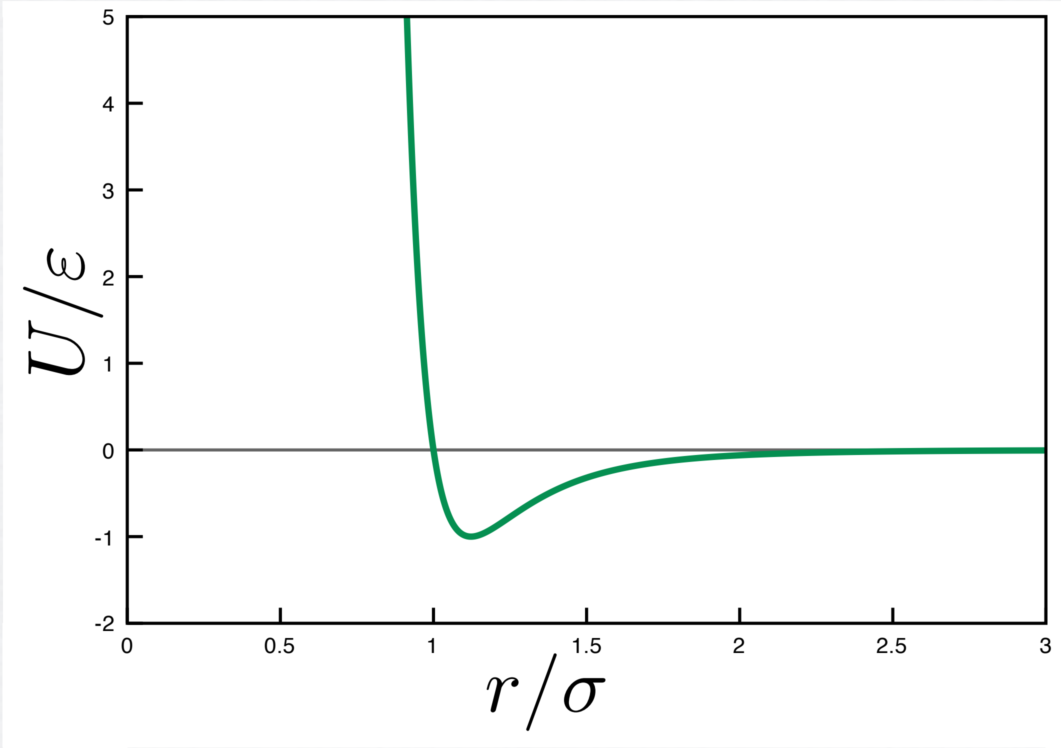
$$\boxed{m_j \ddot{\mathbf{q}}_j = -24\,\varepsilon \sum_{\substack{k=1 \\ k\neq j}}^{n} \frac{\mathbf{q}_k - \mathbf{q}_j}{\left|\mathbf{q}_k - \mathbf{q}_j\right|^2}\left(\frac{2\sigma^{12}}{\left|\mathbf{q}_k - \mathbf{q}_j\right|^{12}} - \frac{\sigma^6}{\left|\mathbf{q}_k - \mathbf{q}_j\right|^6}\right)}$$

Molecular Dynamics

This is just one guy

- The force is given by Lennard-Jones potential:

$$\mathbf{F}(\mathbf{r}) = -\text{grad}\, U(r) = -\text{grad}\, 4\varepsilon\left(\frac{\sigma^{12}}{r^{12}} - \frac{\sigma^{6}}{r^{6}}\right)$$



- The potential vanishes quickly with growth of r, so you can truncate it at a certain cut-off distance r_{cut} (usually $2.5 \cdot \sigma$)

- To avoid a jump discontinuity implying an infinite force the potential must be shifted:

$$U(r) = \begin{cases} U(r) - U(r_{cut}), & \text{for } r < r_{cut} \\ 0, & \text{for } r \geqslant r_{cut} \end{cases}$$

Project

High Performance Computing for Science and Engineering

October 1, 2013

CSElab

Computational Science & Engineering Laboratory

http://www.cse-lab.ethz.ch

ETH

Eidgenössische Technische Hochschule Zürich

Swiss Federal Institute of Technology Zurich

Project 1

1. A-priori performance analysis
2. N-Body problem on CPU
3. N-Body problem on GPU
4. Accelerating the N-body problem
5. CPU and GPU comparisons

A-Priori Performance Analysis

- Why?
    - Estimate achievable performance
    - Understand the algorithm
    - Estimate hardware bottlenecks
- Initial step towards Roofline model

A-Priori Performance Analysis

- Quantities that determine performance
  - Total amount T of memory transfers (read/writes)
  - Total number C of floating point operations (FLOP)
  - Total amount M of memory required to store the data
- Example kernel: $K(\mathbf{x}, \alpha) \rightarrow x_i := \alpha x_i \mid i = 0, ..., n-1$
  - T = 2n
  - C = n
  - M = n+1
- State any assumption you make!

N-Body on CPU

- Starting point for GPU code
  - Simplest base implementation
  - Understand the algorithm
  - Baseline to verify and debug GPU code
- Software
  - C/C++
  - Design to use different forces
  - Design to accomodate GPU code (classes, templates,...)
  - Structure for testing

N-Body on CPU

- Code from scratch
  - Be careful and plan ahead!
  - cppreference.com and cplusplus.com are your friends!
  - Object oriented design will facilitate
    - Use of different forces (gravitation, Lennard-Jones)
    - GPU implementation
- We provide:
  - Timer.h - class for timing
  - ArgumentParser.h - class for reading command line arguments

Box-Muller

Random numbers from a normal distribution

- Required for initial conditions of velocities in MD
- Use drand48() to generate U_1, U_2 (uniformly distributed)
- Box-Muller returns two uncorrelated normally distributed values

$$Z_0 = \sqrt{-2 \ln U_1} \cos(2\pi U_2)$$

$$Z_1 = \sqrt{-2 \ln U_1} \sin(2\pi U_2)$$

N-Body on GPU

- CUDA N-Body Problem
  1. Port C++ code to GPU with CUDA
  2. Optimization
- Keep various optimizations and baseline
  - Easier debugging between optimization steps
  - Easier measurements of performance changes
- CUDA
  - We will introduce it next week
  - CUDA SDK is full of examples

Algorithm Optimization

- Lennard-Jones potential
  - Short range interactions
  - ➡ Cell lists: reduce complexity from $O(N^2)$ to $O(N)$
- Cell lists
  - Will be introduced in two weeks

CPU vs GPU

- Performance comparison
  - “GPUs are 100x faster than CPUs” - **INCOMPLETE INFO**
  - CPU model, #threads, compiler and version, memory
  - GPU model, grid structure, memory
- Accuracy comparison
  - CPUs and GPUs might have different standards for numerics
  - Non-associativity in parallel computing causes different results
  - Useful to detect bugs as well

Report

- Hand-in
  - Code
  - PDF with results and comments
  - Optional: movie of the simulation (VMD, OpenGL,...)
- Rule: feedback is proportional to the effort in reporting
  - Even if something does not work, tell us about it!