Course Outline

• Introduction in algorithms and applications
• Parallel machines and architectures
• Programming methods, languages, and environments
  Message passing (SR, MPI, Java)
  Higher-level language: HPF
• Search algorithms
• Many-core programming, 3 lectures (Ana Varbanescu)
• Real applications:
  – N-body problems, climate modeling
  – LOFAR software telescope (Rob van Nieuwpoort)
N-Body Methods

Source:
Load Balancing and Data Locality in Adaptive Hierarchical N-Body Methods: Barnes-Hut, Fast Multipole, and Radiosity
by Singh, Holt, Totsuka, Gupta, and Hennessy

(except Sections 4.1.2., 4.2, 9, and 10)
N-body problems

- Given are N bodies (molecules, stars, ...)
- The bodies exert forces on each other (Coulomb, gravity, ...)
- Problem: simulate behavior of the system over time
- Many applications:
  - Astrophysics (stars in a galaxy)
  - Plasma physics (ion/electrons)
  - Molecular dynamics (atoms/molecules)
  - Computer graphics (radiosity)
Example from current practice

• Leiden astrophysics group (Simon Portegies-Zwart)
• AMUSE: Astrophysical Multipurpose Software Environment
• Couple different independent simulations into one multi-model simulation
  – Gravity, Radiative transport, Stellar Evolution ...
  – Fortran code written in 1960
  – CUDA code written yesterday
• See:
  – http://www.youtube.com/watch?v=E8nw2x6YV0A
Basic N-body algorithm

for each timestep do
  Compute forces between all bodies
  Compute new positions and velocities
od

• Easy to parallelize:
  – Distribute the N bodies equally among all machines
  – $O(N)$ communication and $O(N^2)$ compute time per timestep
  – No load balancing problems

• Question: how big is N?
Leiden Observatory nominated for 2014 Gordon Bell Prize

Each year, the Gordon Bell Prize, administered by the Association for Computing Machinery's (ACM), is awarded at the Supercomputing Conference (SC). This prestigious prize is intended to track the progress of parallel computing, with particular emphasis on rewarding innovation in applying high performance computing (HPC) to applications in science.

The ACM-appointed awards committee has selected five technical papers that demonstrate exceptional achievements in parallel computing. The final Gordon Bell Prize winner will be announced at the 26th annual Supercomputing Conference that will take place from 16-21 November 2014 in New Orleans.

A technical paper titled "24.77 Pflops on a Gravitational Tree-Code to Simulate the Milky Way Galaxy with 18600 GPUs" with research led by Simon Portegies Zwart and Jeroen Bédorf of the Leiden Observatory (Leiden University), is among this year’s finalists. In close cooperation with SURF-sara, researchers of Leiden Observatory, together with their Japanese colleagues have simulated the long term evolution of the Milky Galaxy using 51 billion particles on the Swiss Piz Daint supercomputer with their N-body gravitational tree-code Bonsai. The highest performance was achieved with a 242 billion particle Milky Way model using 18600 GPUs on Titan, thereby reaching a sustained GPU and application performance of 33.49 Pflops and 24.77 Pflops respectively.

SURF-sara contributed with the code optimization and parallelization work, both on the CPU and the GPU side, and is currently working on adding parallel I/O and an in-situ visualization engine to the Bonsai code.
Hierarchical N-body problems

- Exploit physics of many applications:
  Forces fall very rapidly with distance between bodies
  Long-range interactions can be approximated

- Key idea: group of distant bodies is approximated by a single body with same mass and center-of-mass
Data structure

- Octree (3D) or quadtree (2D):
  Hierarchical representation of physical space

- Building the tree:
  - Start with one cell with all bodies (bounding box)
  - Recursively split cells with multiple bodies into sub-cells

Example (Fig. 5 from paper)

Each node contains center of mass data for its cell
Barnes-Hut algorithm

for each timestep do
  Build tree
  Compute center-of-mass for each cell
  Compute forces between all bodies
  Compute new positions and velocities
od

• Building the tree: recursive algorithm (can be parallelized)
• Center-of-mass: upward pass through the tree
• Compute forces: costs almost all of the time (for big N)
• Update positions and velocities: simple (given the forces)
• $O(N \log N)$ for many practical problems
for each body $B$ do
    $B$.force := ComputeForce(tree.root, $B$)
od

function ComputeForce(cell, $B$): float;
    if distance($B$, cell.CenterOfMass) > threshold
    then
    else
        sum := 0.0
        for each subcell $C$ in cell do
            sum +:= ComputeForce($C$, $B$)
        return sum
Parallelizing Barnes-Hut

- Distribute bodies over all processors
  - In each timestep, processors work on different bodies
- Communication/synchronization needed during
  - Tree building
  - Center-of-mass computation
  - Force computation
- Key problem is efficient parallelization of force-computation
- Issues:
  - Load balancing
  - Communication overhead
    - Only $O(N \log N)$ computation time
  - Key is *data locality*
Load balancing

- **Goal:**
  Each processor must get the same amount of work.

- **Problem:**
  Amount of work per body differs widely.
Communication: Data locality

- Each machine needs only part of the bodies

- Goal:
  - Each CPU must access a few bodies many times
  - Reduces communication overhead

- Problems
  - Access patterns to bodies not known in advance
  - Distribution of bodies in space changes (slowly)
Simple distribution strategies

• Distribute the space
  – Each CPU gets part of the physical space; optimal locality
  – Huge load imbalance, each CPU gets different #bodies

• Static distribution of bodies
  – Each processor gets equal number of bodies
  – Still load imbalances, amount of work per body differs
  – Does not take locality into account

• Dynamic load balancing of bodies
  – Distribute bodies dynamically (like Replicated Workers)
  – Does not take locality into account
  – High communication overhead for handing out jobs
Barnes-Hut algorithm

for each timestep do
    Build tree
    Compute center-of-mass for each cell
    Compute forces between all bodies
    Compute new positions and velocities
od
More advanced distribution strategies

- **Load balancing: cost model**
  
  Associate a computational cost with each body
  
  \[
  \text{Cost} = \text{amount of work (\# interactions) in previous timestep}
  \]
  
  Each processor gets same total cost
  
  Works well, because system changes slowly

- **Data locality: costzones**
  
  Observation: octree more or less represents spatial (physical) distribution of bodies
  
  Thus: partition the tree, not the bodies
  
  Costzone: contiguous zone of costs
Example costzones

Optimization:

improve locality using clever child numbering scheme
Experimental system-DASH

• DASH multiprocessor
  Designed at Stanford university
  One of the first NUMAs (Non-Uniform Memory Access)

• DASH architecture
  Memory is physically distributed
  Programmer sees shared address space
  Hardware moves data between processors and caches it
  Uses directory-based cache coherence protocol
DASH prototype

• 48-node DASH system
  12 clusters of 4 processors (MIPS R3000) each
  Shared bus within each cluster
  Mesh network between clusters
  Remote reads 4x more expensive than local reads

• Also built a simulator
  More flexible than real hardware
  Much slower
Performance results on DASH

• Costzones reduce load imbalance and communication overhead

• Moderate improvement in speedups on DASH
  - Low communication/computation ratio
Speedups measured on DASH (fig. 17)

(a) 32K Particles on DASH

(b) 8K Particles on Simulator
Different versions of Barnes-Hut

- Static
  - Each CPU gets equal number of bodies, partitioned arbitrarily
- Load balance
  - Use replicated workers (job queue, as with TSP) to do dynamic load balancing, sending 1 body at a time
- Space locality
  - Each CPU gets part of the physical space
- Costzones
  - As discussed above
- (Ignore ORB)
Simulator statistics (figure 18)
Conclusions

• Parallelizing efficient $O(N \log N)$ algorithm is much harder than parallelizing $O(N^2)$ algorithm

• Barnes-Hut has nonuniform, dynamically changing behavior

• Key issues to obtain good speedups for Barnes-Hut
  Load balancing $\rightarrow$ cost model
  Data locality $\rightarrow$ costzones

• Optimizations exploit physical properties of the application