Parallel Numerical Algorithms
Chapter 16 – Particle Simulations

Prof. Michael T. Heath

Department of Computer Science
University of Illinois at Urbana-Champaign

CS 554 / CSE 512
Many physical systems can be modeled as collection of interacting particles.

“Particles” vary from atoms in molecule to planets in solar system or stars in galaxy.

Particles exert mutual forces on each other, such as gravitational or electrostatic forces.
N-Body Model

- Newton’s Second Law

\[ F = m a \]

- Force between particles at positions \( x_i \) and \( x_j \)

\[ f(x_i, x_j) \]

- Overall force on \( i \)th particle

\[ F(x_i) = \sum_{j=1}^{n} f(x_i, x_j) \]
N-Body Simulation

- System of ODEs
  \[ F(x_i) = m_i \frac{d^2 x_i}{dt^2} \]

- Verlet time-stepping scheme
  \[ x_i^{k+1} = 2x_i^k - x_i^{k-1} + (\Delta t)^2 F(x_i^k)/m_i \]

- For long time integration, *symplectic* integrators are appropriate (preserve geometric properties, such as orbits)

- \( O(n^2) \) cost of evaluating force at each time step dominates overall computational cost
Reducing Cost of Force Evaluation

- Use Newton’s Third Law: $f(x_i, x_j) = -f(x_j, x_i)$ to reduce work by essentially half.

- Use cutoff radius $R$ and update force due to particles more distant than $R$ less often, thereby reducing cost of force evaluation to $O(nR^3 + \epsilon n^2)$.

- Constrain groups of particles to move together using, e.g., SHAKE algorithm.

- Use hierarchical (“tree”) or multipole methods to reduce cost to $O(n \log n)$ or even $O(n)$, but with some sacrifice in accuracy.
Parallel Particle Simulations

- Straightforward force evaluation naturally parallel but total work prohibitive and memory requirements may be excessive
- Methods for reducing total work also complicate parallel implementation
Parallelizing Particle-Particle Method

- Arrange tasks in 2-D grid, with task \((i, j)\) computing force between particles \(i\) and \(j\)
- Let diagonal elements be “home” to respective particles
- Each force pair computation is perfectly parallel:

\[
\begin{align*}
  f_{11} & \quad f_{12} & \quad f_{13} & \quad f_{14} & \ldots & \quad f_{1n} \\
  f_{21} & \quad f_{22} & \quad f_{23} & \quad f_{24} & \ldots & \quad f_{2n} \\
  & \quad f_{31} & \quad f_{32} & \quad f_{33} & \quad f_{34} & \ldots & \quad f_{3n} \\
  & \quad & \quad f_{41} & \quad f_{42} & \quad f_{43} & \quad f_{44} & \ldots & \quad f_{4n} \\
  & \quad & \quad & \quad f_{n1} & \quad f_{n2} & \quad f_{n3} & \quad f_{n4} & \ldots & \quad f_{nn}
\end{align*}
\]
Particle-Particle Method

- Broadcast position of particle \( i \) to all tasks in same row and column
Particle-Particle Method

- Reduce forces to diagonal along column and perform time integration
- Due to symmetry, could reduce along rows instead
If agglomerate by **columns**, then reduction requires no communication, and broadcast needed only across rows.

Due to symmetry, could agglomerate along rows instead.
Implementing Broadcasts

- Simplest approach is to consider this an all-to-all operation: each process sends positions of particles that it owns to all other processes

- Use `MPI_Alltoall` if each process has the same number of particles or `MPI_Alltoallv` otherwise

- This is very communication-intensive and must be completed before any computation

- In addition, each process must store locations of all $n$ particles

- Is there good alternative?
Rather than perform All-to-all communication as single step, arrange communication in pipeline: in first step, process $i$ sends locations of its particles to process $i + 1 \pmod{p}$ and receives from process $i + p - 1 \pmod{p}$

Each process uses information received in computing force, then can discard the position data

Computation and communication can be overlapped

Pipeline requires $p - 1$ steps, however, so algorithm is not scalable
Digital Orrery

- This algorithm is sometimes called a *digital orrery*

- Introduced in 1985 paper in IEEE TOC, 10 SIMD computers, connected in a ring

- GRAPE computers extend approach of using special-purpose hardware, achieving over 2 PetaFLOPS sustained (using $N^2$ direct algorithm). See [nbodylab](http://nbodylab.interconnect.com/nbl_grape_history.html)

GRAPE-6 board
Improving Particle Algorithm

- Simple algorithm has two major drawbacks
  - work is $O(n^2)$
  - communication is $O(p)$

- Reducing work may also allow reduction in communication
Handling Long Range Forces

- Forces have infinite range, but with declining strength

- Three major options
  - Perform full computation at $O(n^2)$ cost
  - Discard forces from particles beyond certain range, introducing error that is bounded away from zero
  - Approximate long-range forces, exploiting behavior of force and/or features of problem

- Various approaches available for approximating long-range forces
Monopole Representation

- Aggregate distant particles into cells and represent effect of all particles in a cell by monopole (first term in multipole expansion) evaluated at center of cell.
- Use larger cells at greater distances.
- Leads to $O(n \log n)$ method.
- But approximation is relatively crude.
- Early versions of this were shown by Barnes and Hut.
- Algorithm often called tree code.
Tree Code for N-Body Problems

- Tree code approach replaces influence of each far-away particle with aggregate approximate force

Shown here is replacement of forces from four boxes in upper left with one force used by box in lower right
Parallelizing Tree Code

- Divide domain into patches, with each patch assigned to a process
- Tree code replaces communication with all processes by communication with fewer processes
To avoid accuracy problem of monopole expansion, use full multipole expansion

Simple approach yields $O(n \log n)$ method with controllable accuracy; can be more accurate than direct method for large $n$ due to reduced rounding error

Additional tricks allow collecting terms, reducing complexity to $O(n)$ (but with substantial constant), giving *Fast Multipole Method* of Greengard and Rokhlin
Particle-in-Cell

- For many $n$-body calculations, force can be represented as gradient of a potential:

$$F = -\nabla \phi$$

- Potential $\phi$ related to forces produced by particles through field equation

- For electostatics or gravity,

$$\nabla^2 \phi = -c \rho$$

where $\rho$ is charge or mass density

- This suggests simple approach: Define mesh and assign particles to nodes of mesh, preserving charge or mass

- Solve Poisson problem

- Compute force as $F = -\nabla \phi$
Limitations of Particle-in-Cell

- Smoothing out particles introduces significant error.
- Error may be reduced (but not eliminated) by splitting force into two parts:
  \[ F = F_{\text{near}} + F_{\text{far}} \]
- Compute \( F_{\text{far}} \) force due to far-away particles using Particle-in-cell; must remove contribution from nearby particles.
- Compute \( F_{\text{near}} \) force due to near-by particles using simple particle-particle method.
- Known as particle-particle-particle-in-mesh, or PPPM, method.
Already know how to solve Poisson problem in parallel using methods such as FFT or multigrid

FFT requires significant communication

Multigrid reduces communication requirements and may scale better

Load balancing requires more adaptive approach to assignment of particles to processes
Final Remarks

- Methods for \( n \)-body problem often trade accuracy for work.
- Success often depends critically on time integration scheme and model of forces.
- Load balancing can be crucial to achieve scalable performance.
- Task or process *virtualization* can help organize code.
- Parallel approach may consider decompositions in space (as in tree or multipole methods) or particles (as in digital orrery).
References - Particle Simulations

References - Particle Simulations

References - Parallel Particle Simulations

- M. Driscoll et al., A communication-optimal n-body algorithm for direct interactions, IPDPS, Boston, May 2013