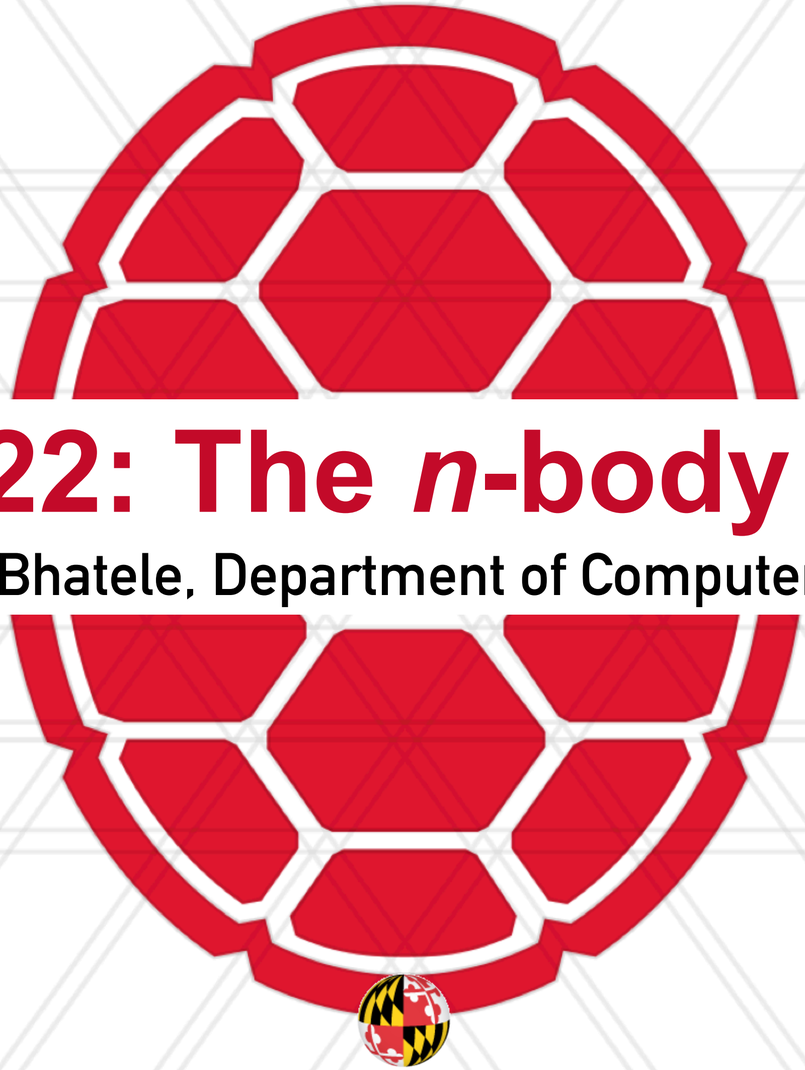
High Performance Computing Systems (CMSC714)



# Lecture 22: The *n*-body Problem

Abhinav Bhatele, Department of Computer Science

UNIVERSITY OF MARYLAND

# Announcements

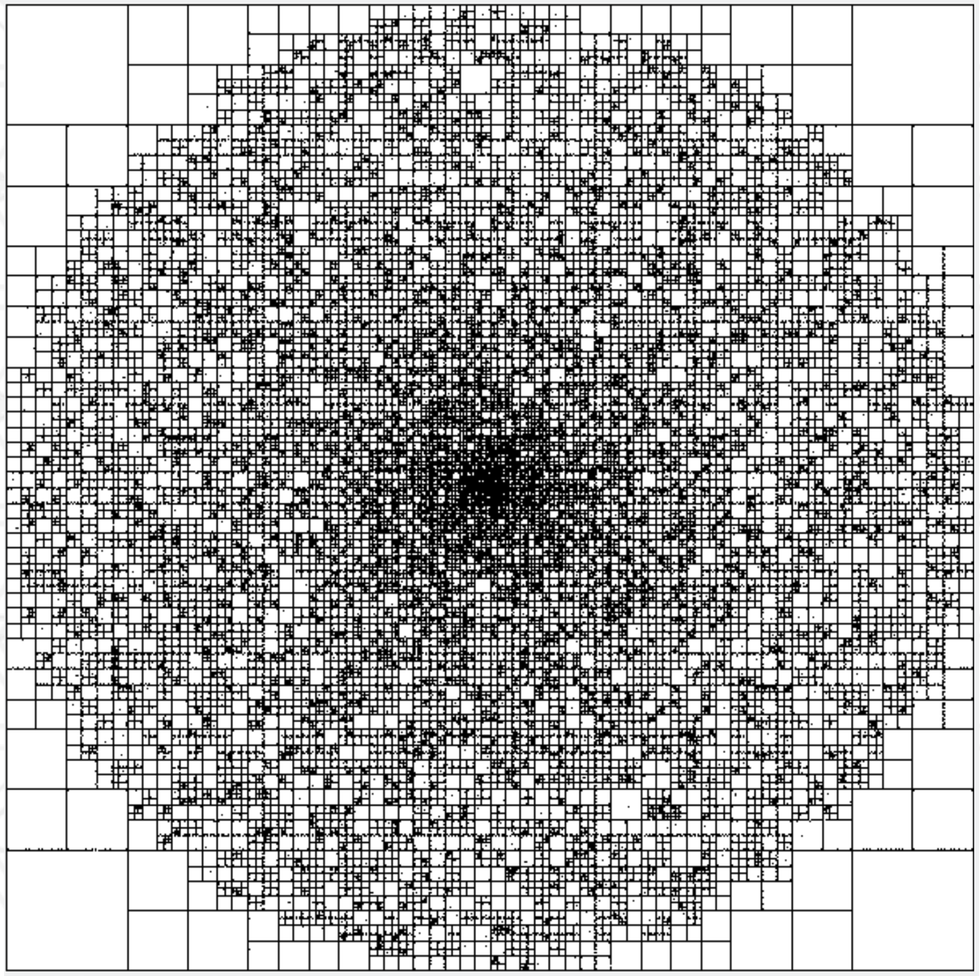
- No class on Nov 14
- Interim report due on Nov 18, 5:00 pm
- Mid-term on Nov 19 in class, 9:30 am

# Summary of last lecture

- Molecular dynamics: calculate trajectories of atoms
- Parallelization strategies
  - Atom decomposition
  - Force decomposition
  - Spatial decomposition
  - Hybrid spatial-force decomposition
- Simulation codes: NAMD, Gromacs, Amber, Blue Matter, Desmond

# The *n*-body problem

- Calculate the motion of celestial objects interacting with one another due to gravitational forces
- $O(n^2)$ force calculations

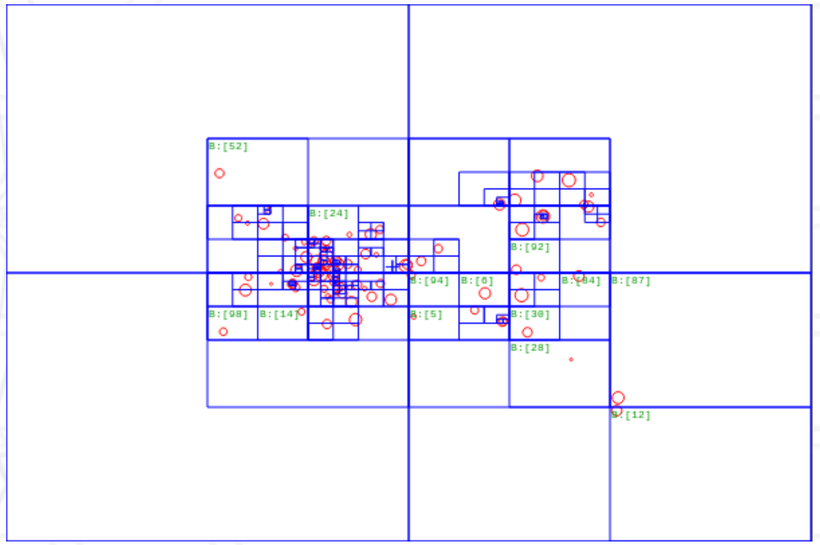


# Different parallelization methods

- Tree codes: Barnes-Hut simulations
- Fast multiple methods (FMM): Greengard and Rokhlin
- Particle mesh methods
- Particle-particle particle-mesh ($P^3M$) methods

# Barnes-Hut simulation

- Represent the space containing the particles as an oct-tree
- Pairwise force calculations for nearby particles
- For tree nodes that are sufficiently far away, approximate the particles in the node by a single large particle at the center of mass



# Fast multipole methods

- Use multipole expansion for distant particles
- Takes advantage of the fact that for nearby particles, multipole-expanded forces from distant particles are similar
- Reduces the time complexity further to O(n)

# Particle-particle particle-mesh methods

- Explicit calculation of forces on nearby particles
- Fourier-based Ewald summation for calculating potentials on a grid
- Smoothed particle hydrodynamics

# Questions

## A Parallel Hashed Oct-tree N-Body Algorithm

- Why did the authors choose the O(NlogN) Treecode algorithm when the O(N) FMM was already known?
- The authors talk about hashing functions of the keys and about a hash table, but it was not clear to me why we need a hash table. Is it to avoid creating a list, which would be inefficient if there are a lot of octree nodes with zero particles? (which will happen for a non-uniform particle distribution).
- The author mentions the Peano curve, Hilbert curve, but there is also the Moore curve (maybe also other ones). Will there be a performance difference in terms of load balance if these are used for domain decomposition? Is there any study comparing the performance of these curves?
- We have access to the interacting cells through the key hashing scheme, so why do we need to do tree traversal? (probably this is some basics of the algorithm, but it was not clear to me.)
- Why would it be so bad to use pointers instead of hashing for navigating the tree? Was the change more about code complexity or performance? Also, is there increased memory footprint as a result of this change in approach?
- The authors identify that the Morton ordering has the disadvantage of spatial discontinuities when split between processes. They propose the Peano-Hilbert ordering may solve this problem. Why then, didn't they go with this ordering instead? Is there a disadvantage to using this ordering?
- The paper identifies how evaluating performance solely based on flops is a poor way to evaluate for their application. They express the hope that performance evaluation will become more balanced and also consider integer operations and memory bandwidth. Has this happened at all? Is there a standard performance measurement that's more balanced?

# Questions

## 42 TFlops Hierarchical N-body Simulations on GPUs with Applications in both Astrophysics and Turbulence

- If my understanding is correct, in the proposed implementation, two domain decompositions are simultaneously used in an overlayed fashion (the octree, and the decomposition based on Orthogonal Recursive Bisection). Another way might be to just use the octree, which is required anyway for the force computation, for the domain decomposition (as in paper1). The overlayed approach may seem, at first glance, to increase communication because of the mismatch between the octree cells and the ORB cells, but why is this not the case?
- The system used in this work uses Ethernet as the interconnect. Does this mean that algorithms like Treecode or FMM are compute-bound and a fast network is not required? Or, will the performance become much better if faster interconnects are used?
- The system is based on GPUs, but what are the CPUs doing during GPU computation? Why not using both CPU and GPU, and exploit heterogeneous parallelism? Is the CPU so much slower that it is not worth doing heterogeneous computation?
- In the implementation details, the authors talk about "global memory". Is this referring to the memory of the GPU, or the main memory of the node?
- No breakdown of computation times of each step of the Treecode and FMM (p2p, p2M, M2M, M2L, L2L, L2p or M2p) is shown. I am curious how that would look like.
- This is kind of a funny question. In Figure 16 they show a picture of the GPU cluster, and it's a total mess. Is this sort of setup typical when people are putting together a DIY supercomputer? There must be a better way, right?
- I was confused by this line "It is expensive to recompute the ORB at each time step, but the cost of incremental load-balancing is negligible". Does "incremental" here imply that rebalancing is done much less frequently than every time step, and that's sufficient to balance well enough without incurring too much cost?
- I thought it was interesting to report on the flops per dollar for the simulation. While this definitely makes sense to do, it seems like the numbers could become out of date very quickly as the price of hardware changes. Is there a way to control for this when comparing to work from the past?

# Questions?

UNIVERSITY OF
MARYLAND

Abhinav Bhatele

5218 Brendan Iribe Center (IRB) / College Park, MD 20742

phone: 301.405.4507 / e-mail: bhatele@cs.umd.edu