

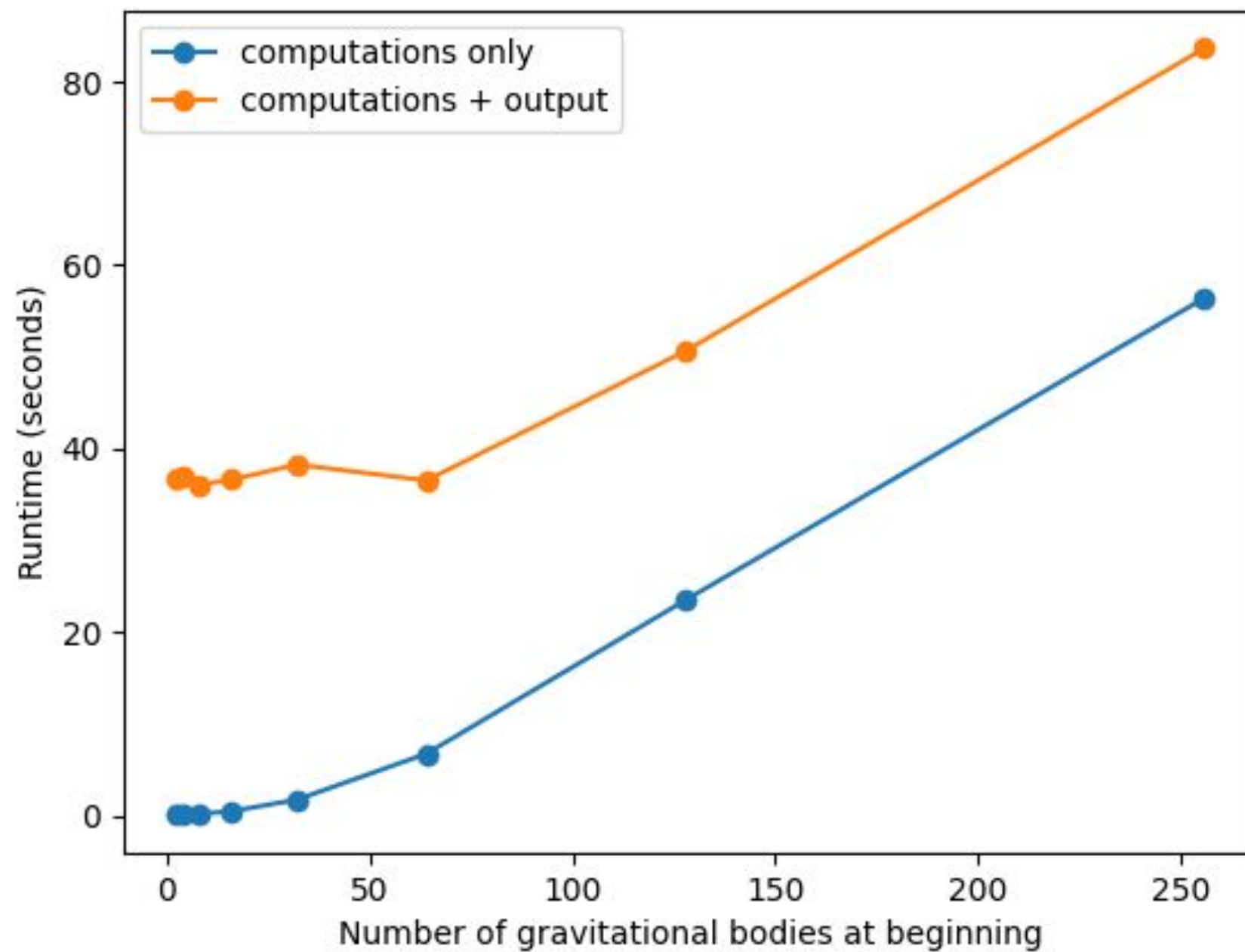
The background is a dark blue space-themed illustration. It features various celestial elements: a ringed planet on the left, a spiral galaxy on the right, and numerous stars of different sizes and shapes scattered throughout. A large, glowing blue oval frame encloses the central text.

Dark Matter Simulator

CS 205

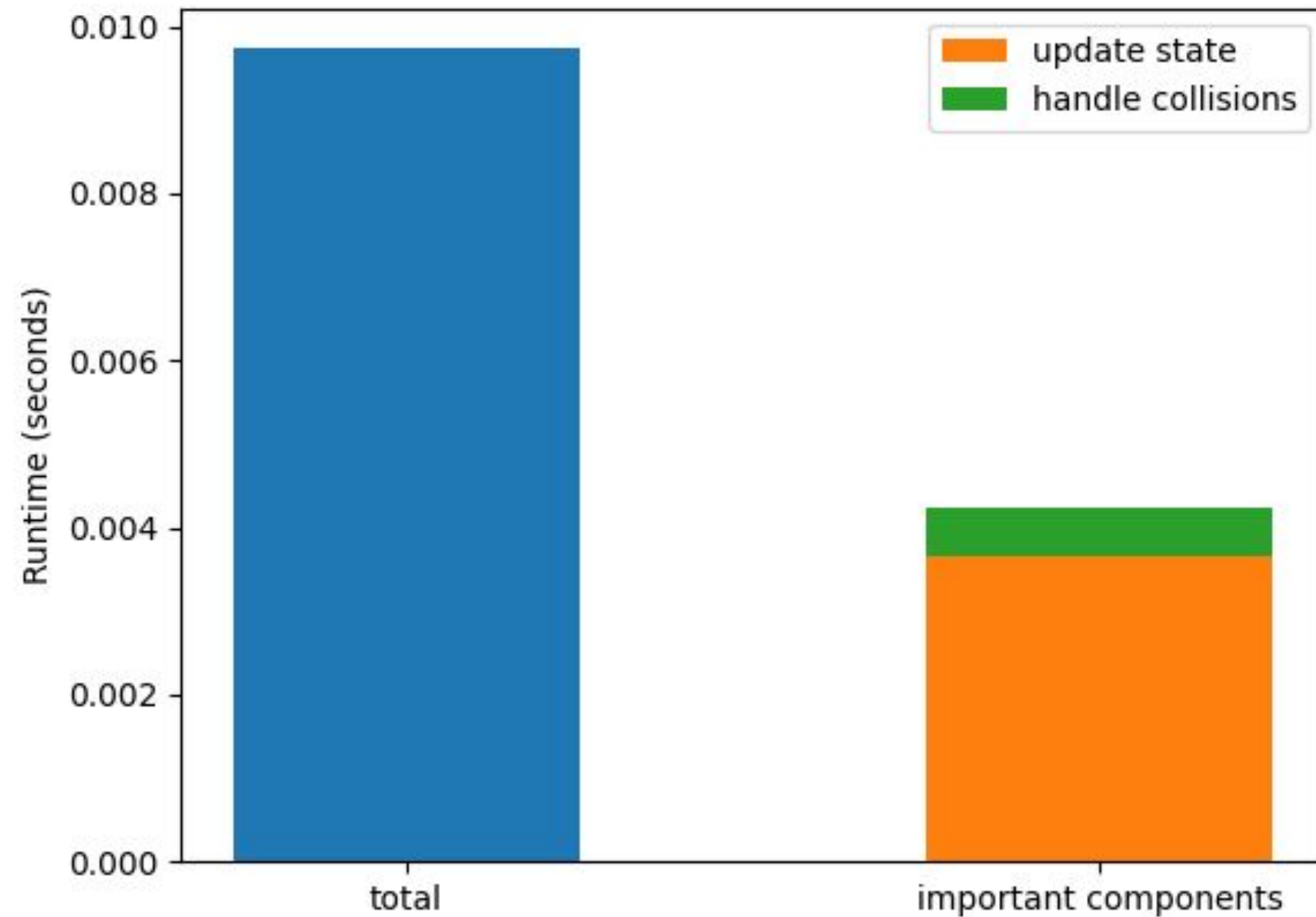
Aditi Raju, Dennis Du, Jared Ni, Rafay
Azhar, Sezim Yertanotov

Sequential Baseline



- Begin with n randomly initialized objects, for powers of two from 2 to 256
- 30,000 position and velocity updates along with collision handling
- Runtime measured with and without writing output

Sequential Baseline



Important computations:

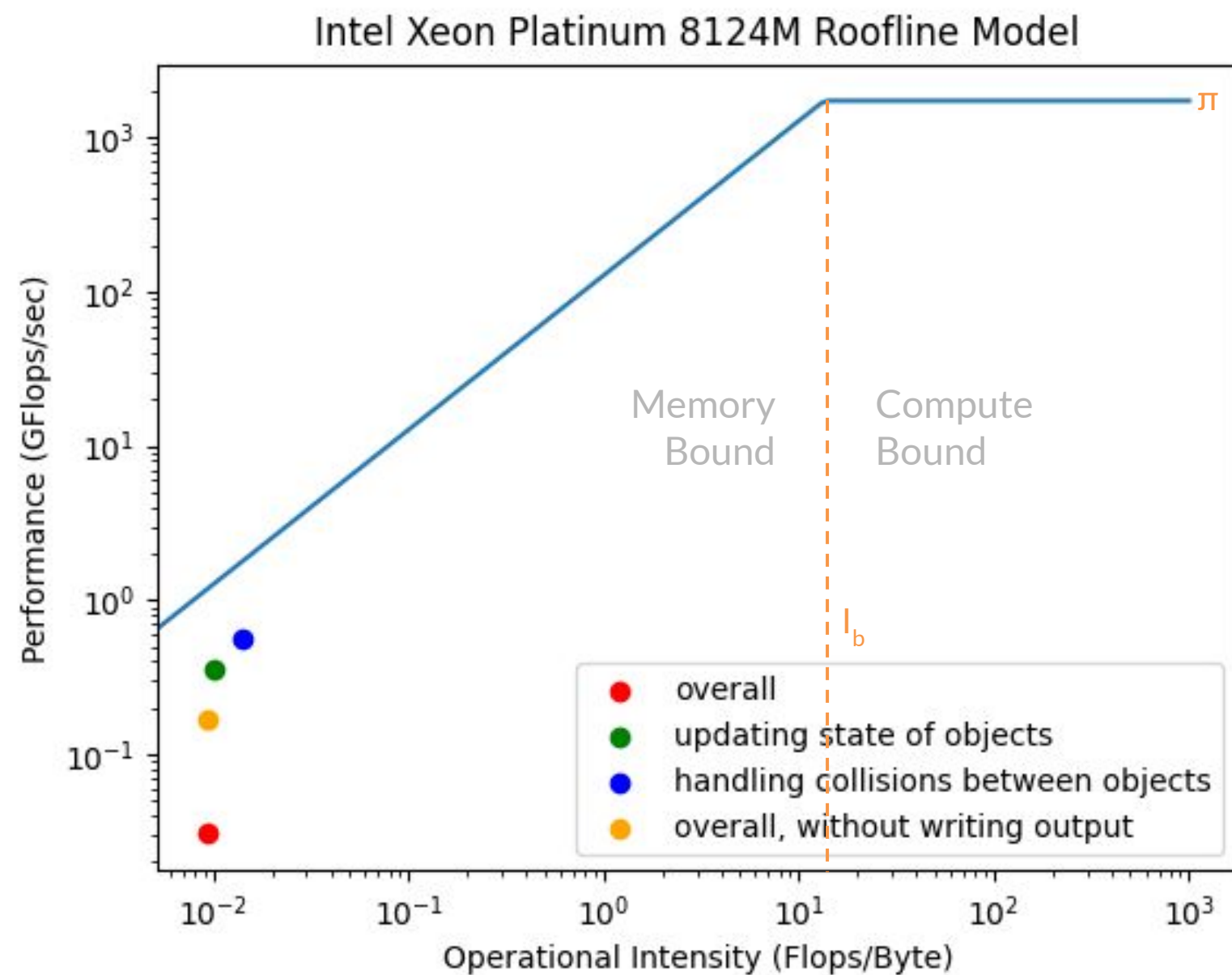
- Updating objects' position and velocity vectors
- Handling collisions between objects

Tested on one iteration with 256 objects

ROOFLINE CALCULATION

Compute Hardware	Intel Xeon Platinum 8124M
Precision	Double precision
Frequency	3.0×10^9 cycle/s
Number of Cores	18
SIMD Lanes	512 bit / 64 bit = 8
Total Flop Per Cycle	4 Flop/cycle
Peak Floating Point Performance π	= frequency * number of cores * SIMD lanes * total flop per cycle = $(3.0 \times 10^9 \text{ cycle/s}) * (18 \text{ cores}) * (8) * (4 \text{ Flop/cycle}) = 1728 \text{ Gflop/s}$
Peak Memory Bandwidth β	128 GB/s
Ridge Point I_b	= $\pi/\beta = 1728 \text{ Gflop/s} / 128 \text{ GB/s} = 13.5 \text{ Flop/byte}$

ROOFLINE MODEL



Peak Floating Point Performance $\pi = 1728$ Gflops/s


Peak Memory Bandwidth $\beta = 128$ GB/s

Ridge Point $I_b = 13.5$ Flops/Byte

All components are memory-bound and below the roofline

Forms of Parallelism

We exploit parallelism in our application by:



Shared memory with
OpenMP

- Each thread computes updates for a small subset of the objects

SIMD approach that
combines OpenMP and MPI

Distributed memory with MPI

- Each process handles a sub-region of the overall gravitational system
 - Each process shares the state of the region assigned to it with neighboring processes
-

Parallel Implementation Plan

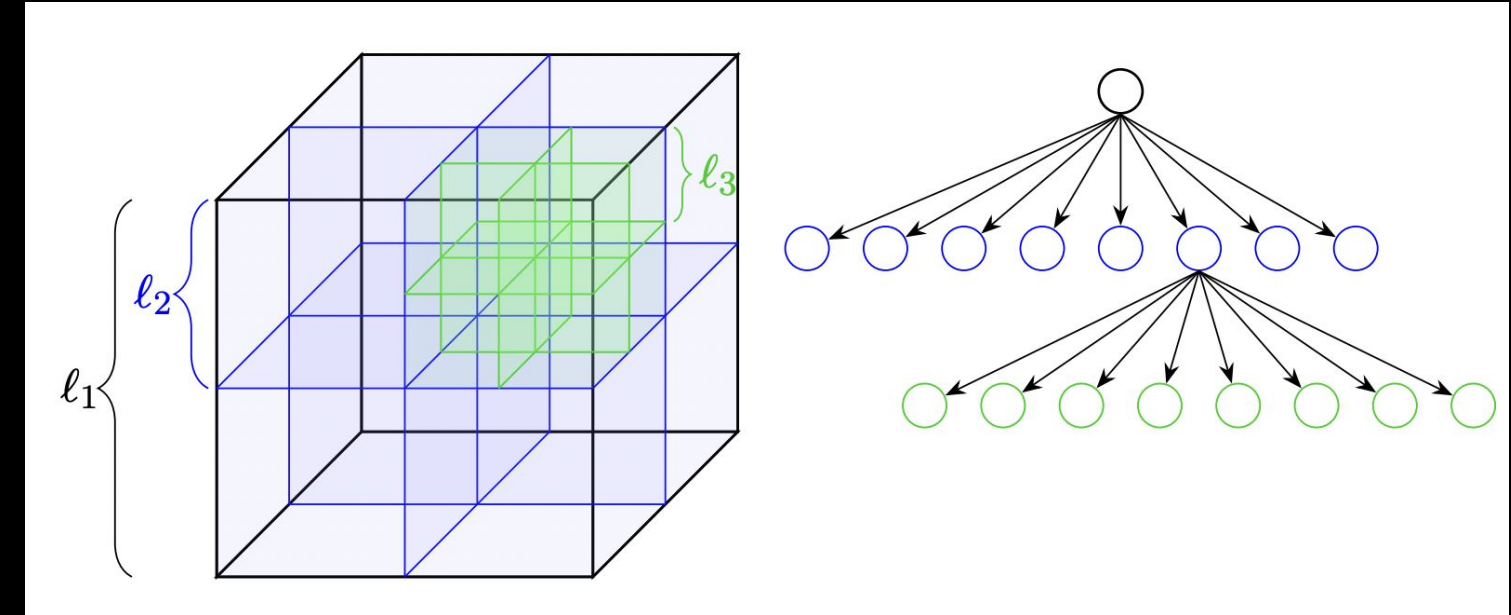
Key idea is to divide the simulation space into octrees via the Barnes-Hut method.

1. Each node carries data:

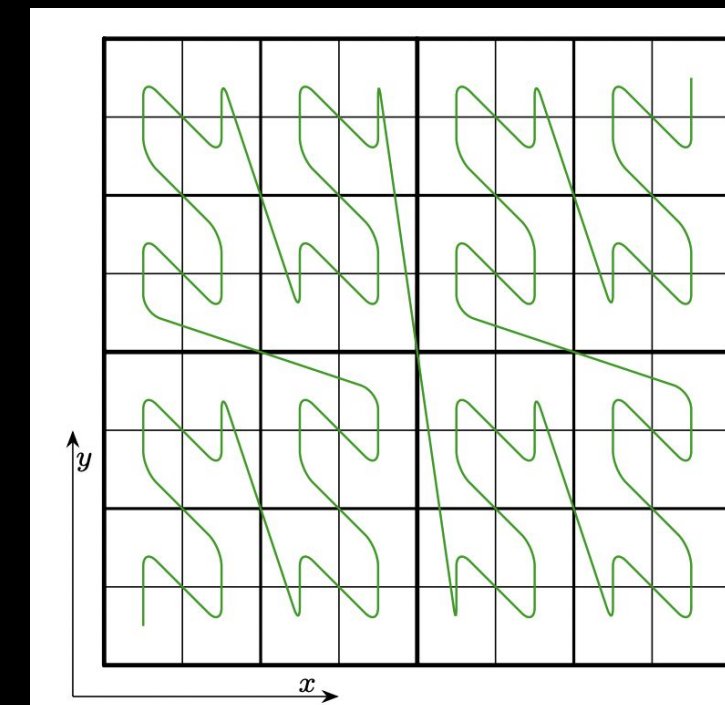
1. The Mass and position of the body (leaf node)
2. Center of mass and other coefficients required to calculate the force (internal node)

2. Dividing objects between different processes results in **load imbalance**. We deal with it using this approach: Assign (N/p) bodies to each process. Each process builds its local octree from the bodies it was given.

We will use a **space-filling curve (Morton's ordering)** to achieve a **balanced partition of bodies between processes**.



A. Brandt. On Distributed Gravitational N-Body Simulations. University of Western Ontario, 2022.



A. Brandt. On Distributed Gravitational N-Body Simulations. University of Western Ontario, 2022.

Parallel Implementation Plan

03

Processes merge their local octrees to create a global octree.*
We will perform a reduction by merging pairs of processes until the root process holds the final tree.

*Results in communication overhead: Deal with this using pairwise (log-wise) reduction to merge octrees

04

The root process broadcasts the global tree to other processes. Then, processes traverse the global tree from top to bottom in parallel to calculate the forces on each of their assigned bodies.

05

Update the velocity and position for each body using the calculated acceleration.





Runtime complexity:
 $O(N/p * \log(N/p))$

Use external libraries for rendering visualizations of universe:

- CImg: Used to output pixels of images
- ffmpeg: Used to take pixels produced by CImg and convert them into a video