A High Performance Implementation of the 2D N-Body Gravitational Problem

Benchmark of the Barnes-Hut algorithm compared to the Brute-Force algorithm

Gael Lederrey

June 23, 2016

Parallel and High-Performance Computing - Spring Semester 2016
EPFL-CSE

Table of contents

Project Description

Introduction

Purpose of this project:

- Write a fast program to solve a problem
- Use the knowledge of optimization learned during the course
- Use MPI to implement a parallel version
- Compare the theory with the numerical results

Introduction

Purpose of this project:

- Write a fast program to solve a problem
- Use the knowledge of optimization learned during the course
- Use MPI to implement a parallel version
- Compare the theory with the numerical results

Use-case: 2D N-Body Gravitational problem

N-Body Gravitational problem - Theory

Gravitational force of body i on body j:

$$\vec{F}_{i \to j} = G \cdot \frac{m_i m_j (\vec{x}_j - \vec{x}_i)}{\|\vec{x}_j - \vec{x}_i\|^3}$$

where G is the gravitational constant, m_i, m_j are the masses and $\vec{x}_i$, $\vec{x}_j$ are the positions.

N-Body Gravitational problem - Theory

Gravitational force of body i on body j:

$$\vec{F}_{i \to j} = G \cdot \frac{m_i m_j (\vec{x}_j - \vec{x}_i)}{\|\vec{x}_j - \vec{x}_i\|^3}$$

where G is the gravitational constant, m_i, m_j are the masses and $\vec{x}_i$, $\vec{x}_j$ are the positions.

At each iteration, we will have for body j:

$$\vec{F}_j = \sum_{i=1}^{n} \vec{F}_{i \to j}$$

We can then update it:

$$\vec{v}_j = \vec{v}_j + \frac{dt}{m_j} \cdot \vec{F}_j \qquad \vec{x}_j = \vec{x}_j + dt \cdot \vec{v}_j$$

N-Body Gravitational problem - Theory

Gravitational force of body i on body j:

$$\vec{F}_{i\to j} = G \cdot \frac{m_i m_j(\vec{x}_j - \vec{x}_i)}{\|\vec{x}_j - \vec{x}_i\|^3}$$

where G is the gravitational constant, m_i, m_j are the masses and $\vec{x}_i$, $\vec{x}_j$ are the positions.

At each iteration, we will have for body j:

$$\vec{F}_j = \sum_{i=1}^{n} \vec{F}_{i\to j}$$

We can then update it:

$$\vec{v}_j = \vec{v}_j + \frac{dt}{m_j} \cdot \vec{F}_j \qquad\qquad \vec{x}_j = \vec{x}_j + dt \cdot \vec{v}_j$$

How can we compute all the interactions between all the bodies?

Algorithms

- **Brute-Force**
  - Compute all the interactions between all the bodies
  - Straight-Forward and easy to implement
  - Complexity: $\mathcal{O}(n^2)$

Algorithms

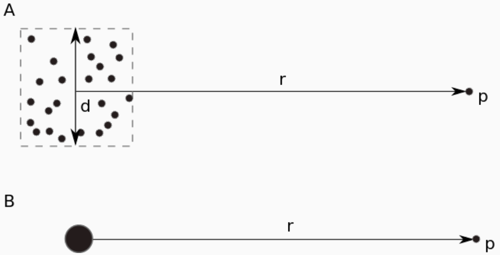
- **Brute-Force**
  - Compute all the interactions between all the bodies
  - Straight-Forward and easy to implement
  - Complexity: $\mathcal{O}(n^2)$
- **Barnes-Hut**
  - Approximation of the far bodies to compute all the forces on a body
  - Not that difficult to implement. (Require more lines of code)
  - Complexity: $\mathcal{O}(n \log n)$

Barnes-Hut algorithm in details

General idea

Idea of this algorithm:

- Use a precision parameter θ to approximate the forces of the far bodies using the center of mass

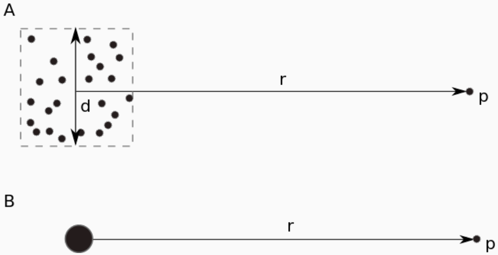


How can we do this?

General idea

Idea of this algorithm:

- Use a precision parameter θ to approximate the forces of the far bodies using the center of mass

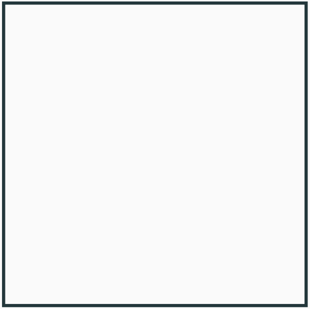


How can we do this? $\Rightarrow$ Use a quadtree and the CM of the nodes.

Quadtree

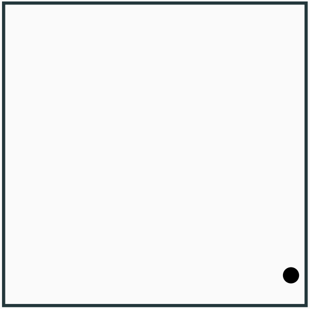
- A quadtree is a tree with four children
  (for the four directions NE, NW, SE and SW)
- In each leaf, there's a maximum of 1 body.
- If a body is added in a leaf with a body, we split the leaf in 4 parts.
  And we add the two bodies to its children.

Quadtree



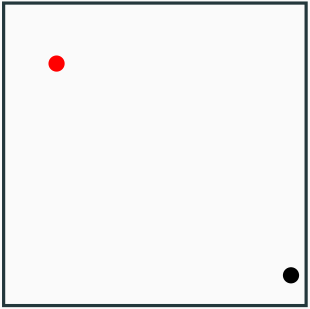
Quadtree

1 body ●

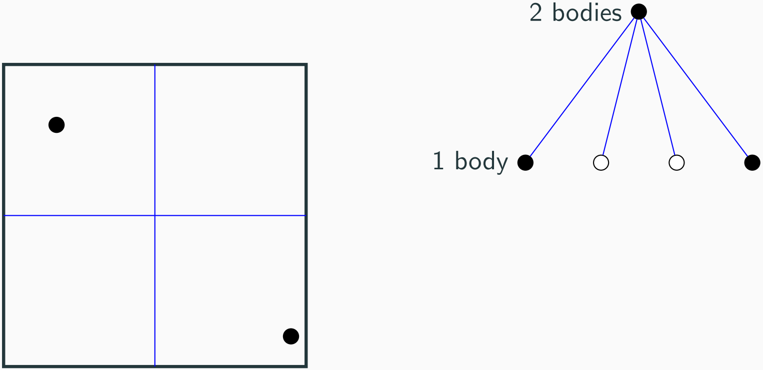


Quadtree

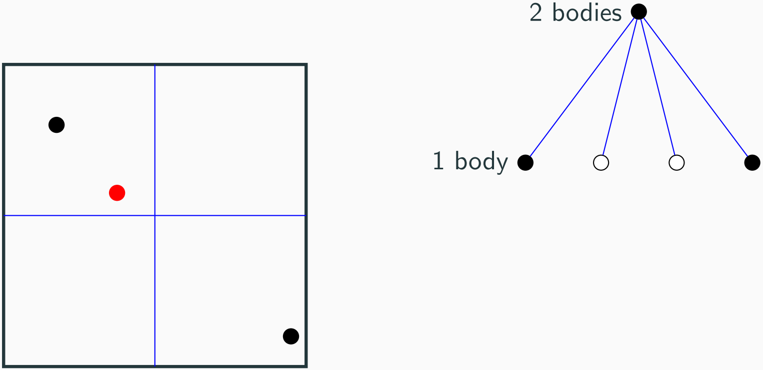
1 body ●



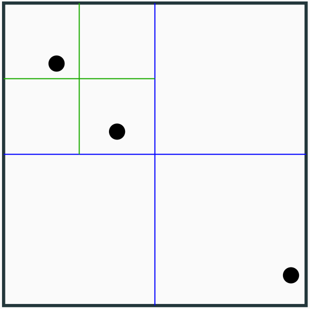
Quadtree

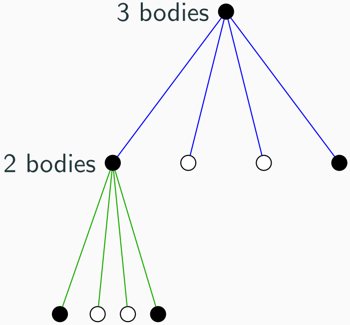


Quadtree

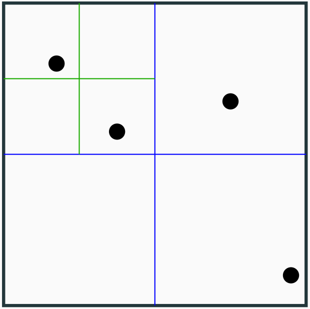


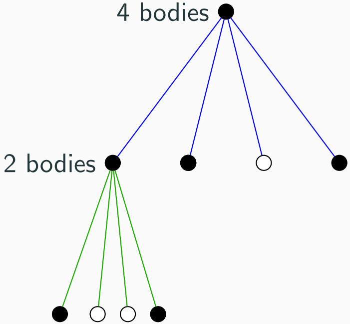
Quadtree



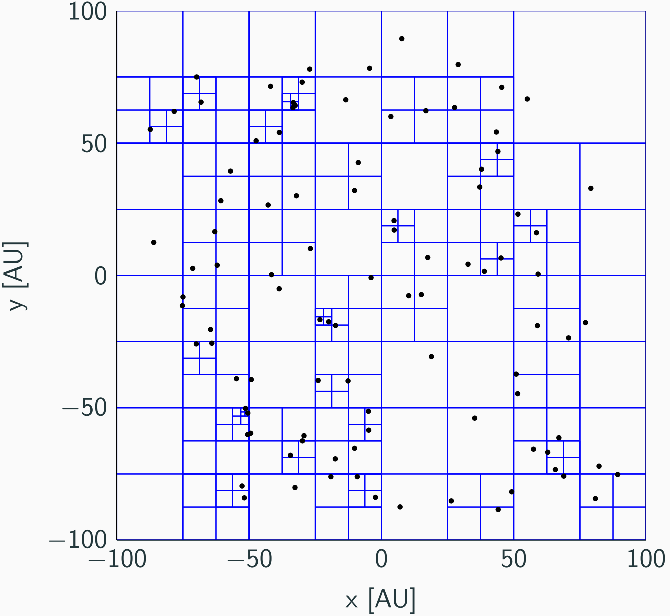


Quadtree





Quadtree



Pseudo-code (with MPI)

Algorithm Barnes-Hut

Read *config* file
Broadcast information to all processes
Build Quadtree
while $t < t_{end}$ **do**
Assign nodes to process
Compute forces for the bodies in the corresponding nodes
Update the bodies
Gather all the bodies
Rebuild Quadtree
Write some information in files

Pseudo-code (with MPI)

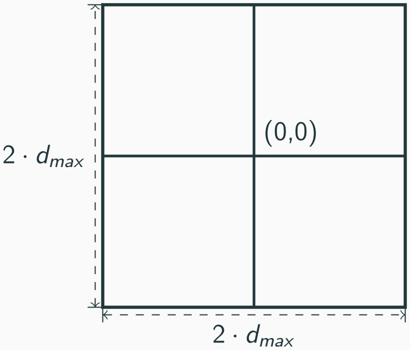
Algorithm Barnes-Hut

```
Read config file
Broadcast information to all processes
Build Quadtree
while t < t_end do
    Assign nodes to process
    Compute forces for the bodies in the corresponding nodes
    Update the bodies
    Gather all the bodies
    Rebuild Quadtree
    Write some information in files
```

1. Lost bodies and collisions?
2. Load-Balancing?
3. Computing the forces?

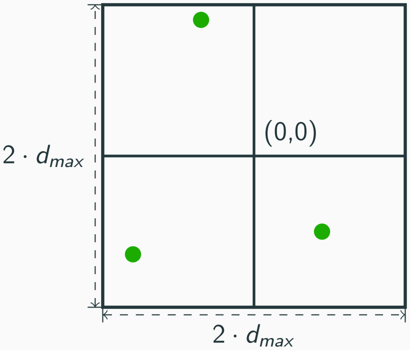
Lost bodies and collisions

Define maximum distance maximum distance d_{max} such that:



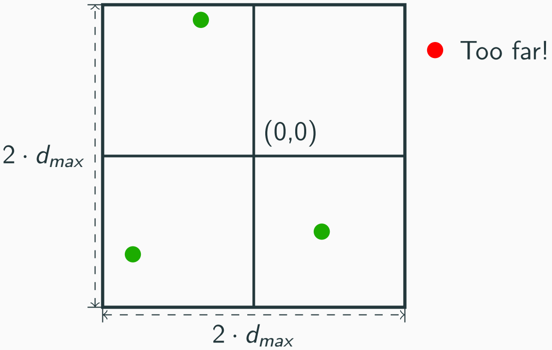
Lost bodies and collisions

Define maximum distance maximum distance d_{max} such that:



Lost bodies and collisions

Define maximum distance maximum distance d_{max} such that:



Lost bodies and collisions

Normally, the mass is defined by the **radius** and the **density**. In this case, we only use the mass.

Lost bodies and collisions

Normally, the mass is defined by the **radius** and the **density**. In this case, we only use the mass.

Therefore, we define a distance d_{min}. The conditions to do a collision are:

1. We insert a body i in a node containing a body j
2. `0.5*(node.width+node.height)` $< d_{min}$

This is not perfect. But the depth of the tree will always stay finite.

Lost bodies and collisions

Normally, the mass is defined by the **radius** and the **density**. In this case, we only use the mass.

Therefore, we define a distance d_{min}. The conditions to do a collision are:

1. We insert a body i in a node containing a body j
2. `0.5*(node.width+node.height)` $< d_{min}$

This is not perfect. But the depth of the tree will always stay finite.

The new body is defined by:

$$\vec{x}_{new} = \frac{m_i \vec{x}_i + m_j \vec{x}_j}{m_i + m_j}$$

$$\vec{v}_{new} = \frac{m_i \vec{v}_i + m_j \vec{v}_j}{m_i + m_j}$$

$$m_{new} = m_i + m_j$$

Load-Balancing

If we put 75% of the bodies in one node and 25% in all the other nodes, what happens?

Load-Balancing

If we put 75% of the bodies in one node and 25% in all the other nodes, what happens? $\Rightarrow$ The algorithm will slow down.

Load-Balancing

If we put 75% of the bodies in one node and 25% in all the other nodes, what happens? $\Rightarrow$ The algorithm will slow down.

Therefore, we need to put approximately the same number of bodies in each process.

Load-Balancing

If we put 75% of the bodies in one node and 25% in all the other nodes, what happens? $\Rightarrow$ The algorithm will slow down.

Therefore, we need to put approximately the same number of bodies in each process.

To achieve that, we can walk in the tree and assign nodes to the processes. Each process can contain n_{max} bodies. If we assign a node to a process, then all the bodies in the children nodes will be assigned to the process.

For maximizing the performance, the nodes assigned to a process should be as close as possible.

Computing the forces

How can we use the precision parameter θ in order to approximate the forces far from a body?

Computing the forces

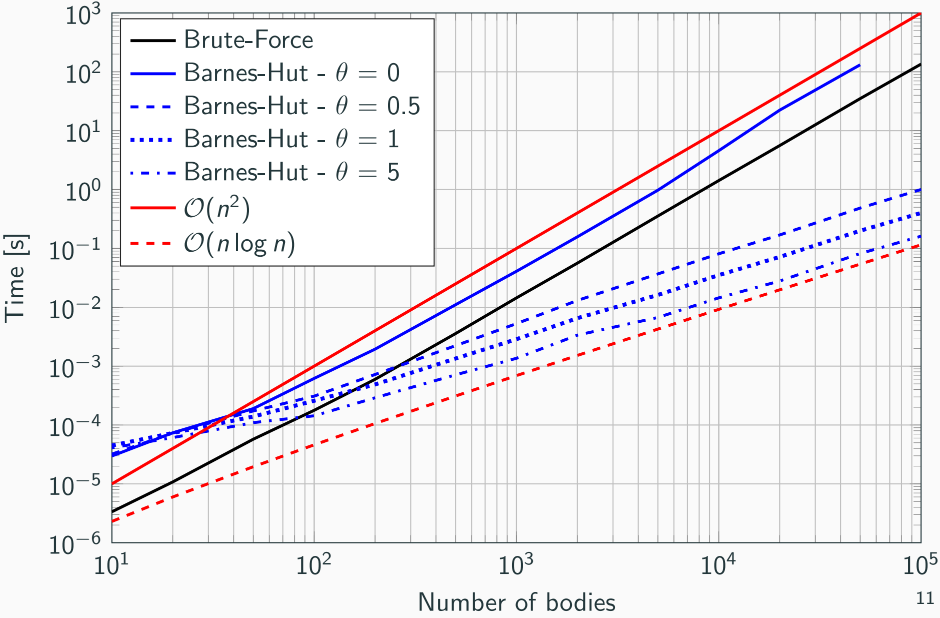
How can we use the precision parameter θ in order to approximate the forces far from a body?

Algorithm Approximation of the force

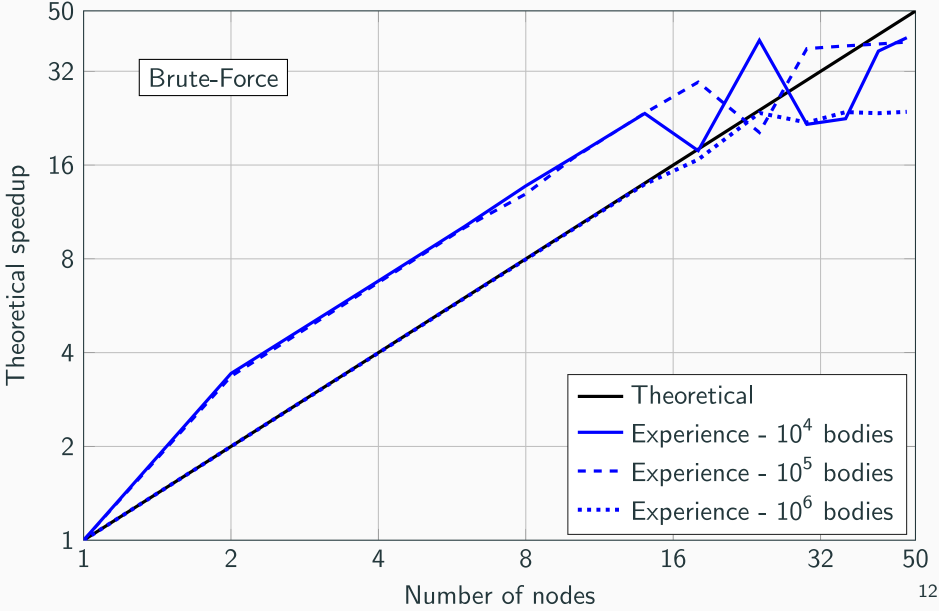
```
dist = node.distance(body);
if !node.isLeaf then
    if 0.5*(node.width+node.height)/dist <= theta then
        node.applyForcesOnBody(body);
    else
        Continue recursion with the children
else
    if node.containsBody then
        node.localBody.applyForcesOnBody(body);
```

Results

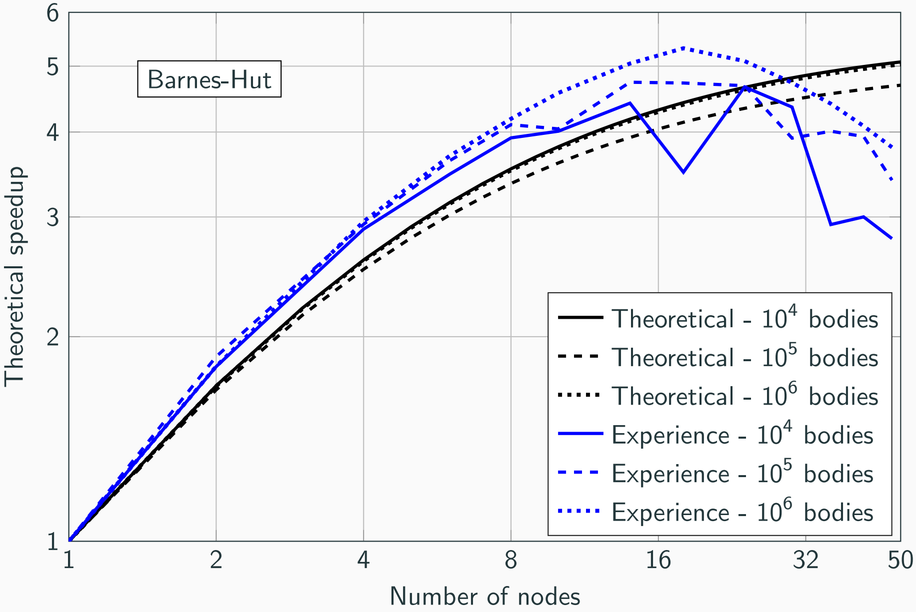
Complexity


Brute-Force
Barnes-Hut - $\theta = 0$
Barnes-Hut - $\theta = 0.5$
Barnes-Hut - $\theta = 1$
Barnes-Hut - $\theta = 5$
$\mathcal{O}(n^2)$
$\mathcal{O}(n \log n)$
Time [s]
Number of bodies
10^3
10^2
10^1
10^0
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^1
10^2
10^3
10^4
10^5

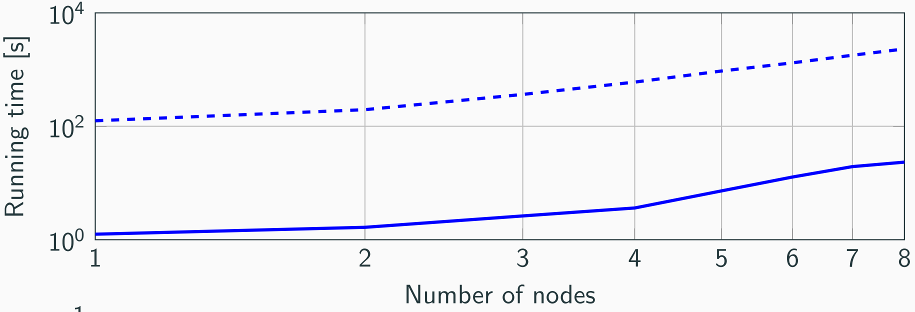
Strong Scaling

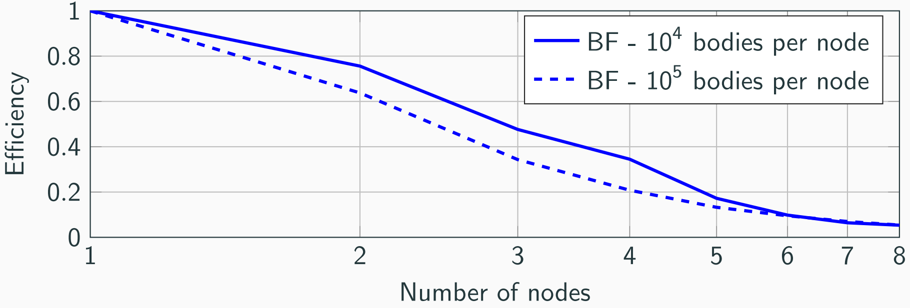


Strong Scaling

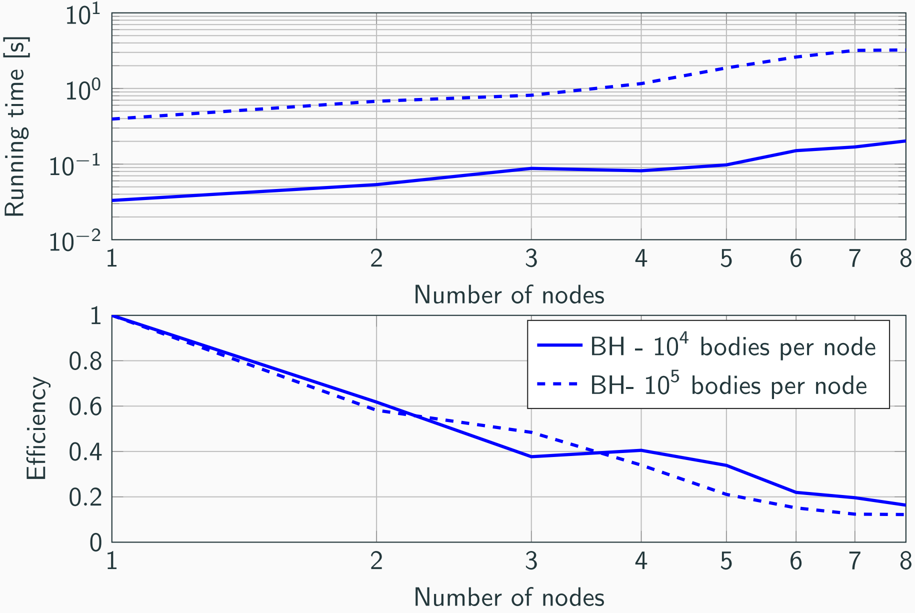


Weak Scaling





Weak Scaling



Conclusion

Conclusion

How can we make it better?

- Use MPI I/O to write the positions of the bodies
- Implement a heuristic for the construction of the tree
  (don't rebuild it at each iteration)
- Use a sorting algorithm for the bodies to build the tree in parallel
- Use RK4 (or another schema) to update the bodies

Conclusion

How can we make it better?

- Use MPI I/O to write the positions of the bodies
- Implement a heuristic for the construction of the tree (don't rebuild it at each iteration)
- Use a sorting algorithm for the bodies to build the tree in parallel
- Use RK4 (or another schema) to update the bodies

What did we learn?

- A clever algorithm in serial can be better (and more ecological) than an easy parallel algorithm.
- Complex serial algorithm can be "easily" updated to a parallel version
- We can always make an algorithm faster. But is it worth it?

Thank you

n-Body simulation of the Solar System

Algorithm: Barnes-Hut

Number of bodies: 1 million

(Sun, 8 planets and 999'991 asteroids)

Length: 10'000 days

(2'000 time steps are shown)