Parallelization of the Barnes-Hut N–Body Simulation Algorithm

Dren Gashi[1], Joe Mayer[1], Nicolas Mayer[1],
S. Varrette[2], V. Plugaru[2] and Pascal Bouvry[2]
[1] University of Luxembourg, MICS
[2] Computer Science and Communications (CSC) Research Unit
2, avenue de l'Université, L-4365 Esch-sur-Alzette, Luxembourg
Firstname.Name@uni.lu

[1] : These authors contributed equally.

***Abstract*—Sir Isaac Newton defined the principles describing the motion of two particles under the influence of their mutual gravitational attraction. However, he was unable to solve the problem for more than two particles. Systems of three or more particles can only be solved numerically. The Barnes-Hut N–body simulation algorithm approximates nearby bodies by considering them as a single body and is widely used due to its time complexity of $O(n \log n)$. In this paper, we show our results of a parallel version of an alternative implementation of the Barnes-Hut algorithm that differs from current state-of-the-art solutions in terms of tree construction, resulting in faster computations while allocating more memory in the beginning. We also compute the forces of all bodies, including those outside of the visible space which also differs from other implementations. We provide a sequential and a parallel version using MPI and discuss the results such as speedup gained by parallelism vs. sequential and other parallel executions.**

***Index Terms*—N–Body Problem, N–Body Simulation, Barnes-Hut Parallelization**

I. Introduction

The N-body problem in physics concerns the prediction of motions of individual particles or bodies that are interacting with each other in a gravitational field. A N–body simulation reproduces the behaviour and evolution of a system that is composed of N bodies under physical influences, such as gravity or physical forces, in order to solve the N-body problem. This simulation is widely used in different areas, i.e. astrophysics or molecular dynamics, to study the interaction of celestial objects or the formation of the Universe as being done by the Millennium Simulation [1] as a popular example.

The *two-body* problem with $N = 2$ has already been solved and described by Newton's Laws of Motion and by the inverse square Law of Gravitation. However, for $N > 2$, the law cannot be applied because no closed form solutions to Newton's law exist since the motions of the particles become unpredictable, hence making the N–body problem one of the most challenging problems in the history of science [2]. Nowadays there exist other proposals for solving the N–body problem with small N, i.e. $3 \leq N \leq 5$. However, these formulas are *restricted*, in the sense that they make additional assumptions that must hold. For instance, for $N = 3$, i.e. the well-known *three-body* problem [2], the mass of the third particle is set to zero so that the problem can be reformulated into the two-body issue [3]. Systems with $N < 3$ have unique and analytical solutions, while larger N require numerical integration. For large N, however, the computations become intensive, thus making a scalable simulation impossible for the past decades. In order to speed-up the simulations, several algorithms have been proposed that try to achieve best time complexities. Also, with high performance computing (HPC) nowadays, we can run larger simulations and achieve more accurate and reliable results within reasonable time. For our simulation, we make use of the popular Barnes-Hut algorithm [4] that reduces time complexity from previous algorithms of $O(n^2)$ to $O(n \log n)$. Our contribution is an approach that is different in two perspectives from the current state-of-the-art methods. We achieve a fast simulation by allocating more memory for the tree construction and reducing computation time by parallelizing the construction of the tree. Also we compute the physical forces for *all* bodies, i.e. also of those outside the "visible"/initial space dimensions which differentiates our approach from current state-of-the-art solutions. We provide a sequential as well as a parallel version using MPI of the simulation and compare the speedup obtained using parallelism with different parameters. A comparison against sequential executions is also presented. The remainder of this paper is organized as follows. In Section II, we define the problem, list notations, describe properties of our model and assumptions that must hold. Section III reports on related work. Section IV explains our approach of solving the N–body problem using efficient parallelization. We conduct a simulation study in Section V. Finally we conclude in Section VIII.

II. Problem Definition and Model

A. Problem definition

The problem we are tackling is about predicting iteration-wise the next positions of N bodies in a 2D–space according to their mutual gravitational force attractions. This will be done in the C programming language in a sequential and a parallel version using MPI and the HPC cluster of the University of Luxembourg.

B. Definitions and notations

Acronym / Notation	Definition
c_m	Mass multiplicator
G	Gravitational constant
I	Number of iterations
v_F	Force vector
m_i	Mass of particle i
m_c	Center mass
r	Root node

C. Properties

- Every particle is initially positioned at a random 2D position with a random limited velocity.
- Every particle has a random mass m_i.

D. Assumptions

- The rotations of bodies are neglected.
- We consider a simulation in a 2D space.
- An endless space without boundaries, therefore no body is neglected.
- The collision of bodies is not taken into account.

III. Related Work and Background

This section describes related work conducted by other researches. In [4], the original Barnes-Hut algorithm is described.

With the trend of high performance computing in the last decade, it is nowadays much easier to perform simulations that would take ages to finish some decades ago. This is why many researches are starting to reconsider old problems from the very bottom. The N–Body problem has been addressed by several scientists in the past years where a major goal is to try to get a simulation work with as much bodies as possible. One of the most outstanding works with this goal was done by Ishiyama *et al.* [5] where the researches successfully simulated one *trillion* bodies on a HPC-cluster of 4.45 pflops of capacity.

Other works done in this realm is the attempt to further improve the algorithm of Barnes-Hut, i.e. achieving $O(n)$ complexity [6] as opposite to $O(n \log n)$ which is already an improvement compared to $O(n^2)$.

A. The Barnes-Hut algorithm

The Barnes-Hut algorithm allows us to reduce the time complexity from $O(n^2)$ to $O(n \log n)$. There even exist further improvement techniques as discussed in [6] and [7] as examples. Considering parallelization, researchers are also working on high scalable methods based on Barnes-Hut using GPU accelerated parallelization [8] [9] [10] [11]. However Barnes-Hut is only an approximation and therefore all techniques that improve the $O(n^2)$ complexity are not 100% correct and precise. However, the difference between simulation and reality is extremely small and therefore insignificant when looking at the bigger picture.

In the following subsections we will shortly describe how this algorithm works.

1) Tree Construction: First of all we need to create a quad-tree which sort the bodies into the right position. A quad-tree is a tree where every node has exactly 4 children. Every body in the tree is represented by an unique leaf, however, not every leaf represents a body because it may be empty. In order to build a quad-tree with all the bodies, we first need to create an empty root leaf r and put the first body into it. This root node represents the smallest possible box, which contains all the bodies, also called bounding box. Next we pick the second body and place it again into r. However now 2 bodies are in r, which means that the root can no longer be considered as a leaf. Therefore r has to be considered from now on as a node and we need to add 4 empty leafs corresponding to the north-west (NW), north-east (NE), south-west (SW), south-east (SE). The 2 bodies from r need to be placed at the direct sub-leaf, by looking at the bodies coordinates and the middle of the upper node's bounding box. This may even mean that 2 bodies may be placed into the same sub-nodes. Therefore this procedure has be repeated until both bodies are represented by 2 distinct leafs. This whole procedure will be repeated until the last body, including all the other ones, have reached a leaf.

2) Force Calculations: After having constructed the tree, we may apply the force impact each body receives. For each body, we first look at c_m of the root node r of our quad-tree. If the body is enough far away from c_m, it will calculate the all force vectors v_F with c_m from r and the impulse calculation for that particular body is finished. However, if the body is close enough, it will recursively redo the operations for all the sub-nodes from r. This recursive procedure will be stopped until we reach a leaf. θ, which is often static, defines if a body is far enough and determines if the body can immediately calculate the force impact with a specific node. In case θ is shrinking to 0, the closer we get to the real world expectation. If θ becomes 0, the force calculation acts exactly the same way as the N-squared algorithm, as every body has to calculate it's v_F with every other body.

IV. Our approach

In this section we will elaborate on a different way on how we compute the necessary steps to calculate the final impact forces.

A. Tree Construction

The current state of the art takes each body after the other and places it into the tree until the body reaches a leaf. As soon as a body reaches a node which already contains a body has more than 1 body, it will become a node which has 4 sub-nodes and sort the 2 bodies as shown in the figures 2-5 from the given universe which is depicted in figure 1.

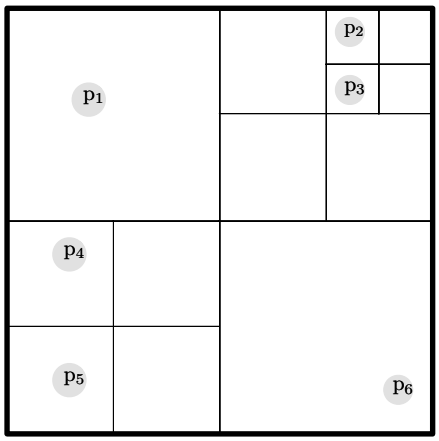

Fig. 1. Universe

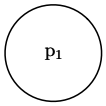

Fig. 2. State-of-the-art approach - 1

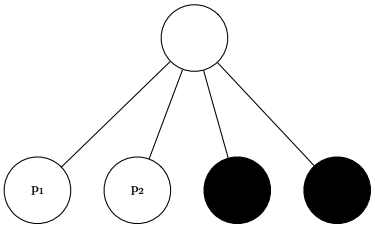

Fig. 3. State-of-the-art approach - 2

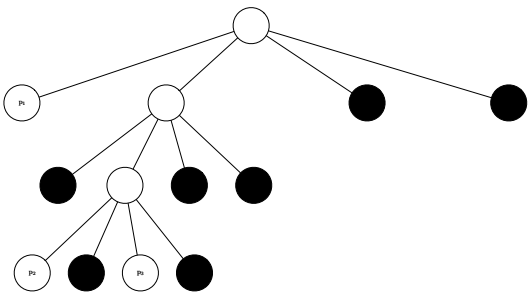

Fig. 4. State-of-the-art approach - 3

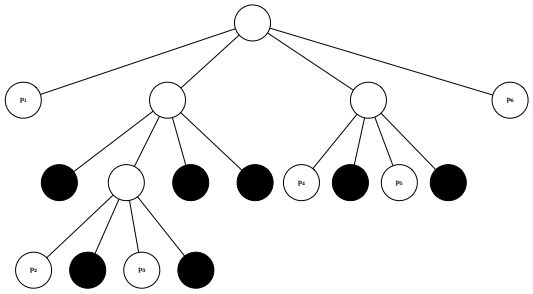

Fig. 5. State-of-the-art approach - 4

Our approach is different as we first sort all bodies from an upper node and put them into the direct sub-nodes as shown in figures 6-8, without subdividing the sub-nodes immediately. After having sorted all the bodies from the upper node, we calculate and store the m_c location and it's gravity for each sub-node to save some computation. Immediately after having calculated the last part, it is highly recommended to deallocate the memory where the bodies are stored from the node. After this we may recursively repeat this procedure with all the 4 sub-nodes until every sub-node is a leaf which contains 0 or 1 body.

This approach needs temporarily more memory to allocate the bodies into their correspondent quadrant/sub-nodes. On the other hand, we have less check conditions to compute if the sub-node has to be split again or that the body has to continue its path or to leave it at this node. An other major advantage is that this allows us to parallelize the sorting of bodies more easily as there will be less critical sections, which will be shown later.

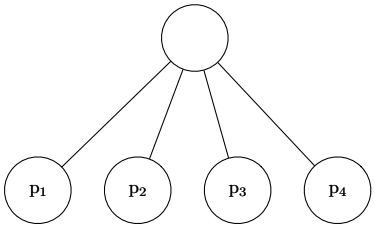

Fig. 6. Our approach - 1

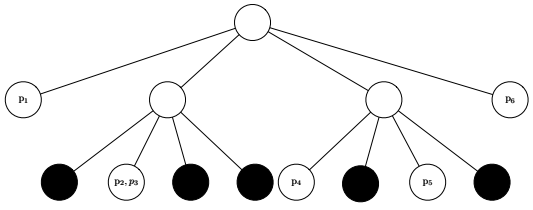

Fig. 7. Our approach - 2

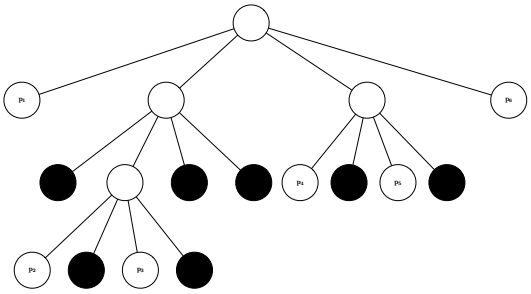

Fig. 8. Our approach - 3

Furthermore to have a more realistic simulation, the root's bounding box is dynamic and contains at any moment all bodies. At every iteration the minimum and maximum of the x and y coordinates of each body are picked to build the smallest possible bounding box around all the bodies. Current state of the art often have a static bounding box, even though the whole universe of the bodies may move outside the bounding box. Other known Barnes-Hut N-Body simulators ignore outliers and also ignore their force impact on the universe, as shown in figure 9.

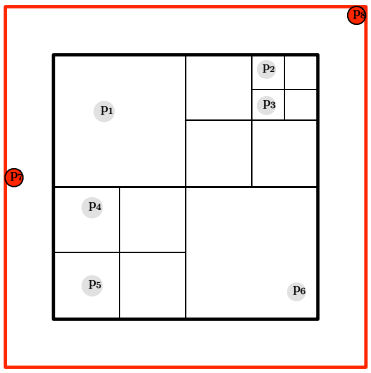

Fig. 9. Universe - Bounding boxes

B. Force Calculations

The force calculation is not different from the current state of the art. However we added a dynamic θ, which defines when a body is far enough from a given node / quadrant to immediately calculate the force with the upper node or if the body needs to go further into the sub-nodes.

C. Parallelization

In this section we describe the parallelization methods used for our approach. We are using the Message Passing Interface (MPI) for parallelization.

1) Master – Slave Model: In order to maintain an uniform distribution, we make use of a dedicated processor (usually the first with rank 0) which we call the *master* and the other processors, the slaves, which are following the instructions of the master.

In this program we have 2 parts which are parallelized using MPI with Master-Slave / Server-Client architecture.

2) Parallel Tree Construction: The first part is parallelizing the construction of the tree while using the new approach. Starting from the root the master / server counts the amount of existing bodies, including the bounding box. The server splits the bodies into equally distributed chunks. Every client receives one chunk, and calculates the position where the bodies belongs to and sends these information back to the server. The server collects this information and modifies the tree himself by putting the bodies into the corresponding sub-nodes. This procedure will be repeated recursively until every body has reached a leaf. However it is not recommended to use the master-slave architecture for every node as the amount of bodies to sort shortens, the time to send messages between processors takes longer than actually sorting the bodies from the server.

3) Parallel Force Calculations: The second master-slave MPI part is for the force calculation. This one is more complicated as every computing unit needs to have the knowledge of the constructed tree. Multiple processors don't share the memory, therefore the data of the tree needs to be send from the server to the clients. As we used C, it is not possible to send nested structures / trees over MPI. Therefore we implemented our own serialization of the tree, but the server only sends those data which are needed to compute the forces. Next, as every processor now has the knowledge of the tree, the server has to send again the information of the bodies. After, the clients calculate the forces for each body they were assigned to and send this data to the

V. Implementation and Experimental Setup

A. Initial simulation setup

G	$6.67 \cdot 10^{-11}$
I	1200
Initial position dimensions	4000×2080
Number of particles	10^6
Min. m_i	$30 \cdot 10^6$
Max. m_i	$50 \cdot 10^6$
Initial θ	0.5

B. Program

The program is implemented in pure C. The program exports a JSON file containing all the bodies positions at each step. This data is used by another program as input in order to enable a visual simulation which is also used as verification of correctness. The C-program makes use of the MPI library. For our simulation, the OpenMPI version were used.

C. Visualization

In order to gain a visual insight into our generated simulation and as a proof of concept, we have developed a browser based tool that takes as input the computation data of the

simulation and produces a visualization of the whole relative behaviour between all the bodies, i.e. rendering all particles to their corresponding position. In order to provide a good and satisfying simulation no matter how many bodies are rendered, the tool needs to have a high performance when displaying more than 100k different bodies.

The challenge was definitely not to reinvent the wheel but more on finding a good Javascript Framework that provides such performance out of the box. The choice after all fell on the Three.js framework which enables the easy use of WebGL in the browser and provides the possibility to create particle systems in a three dimensional space. This flexibility given by the framework allows to simply use the available structures and map the read in data on those particles. After all the simulator should be as easy usable as possible without doing to many changes when changing from 2D to 3D.

Furthermore the construction of the tool in the browser allows to run it without the big need of a setup, simply drop in the name of the json file and you are ready to go. In addition the simulator is ready to display the boxes of the bodies, but is disabled in the current version of the tool.

The following figures show our simulator and a running simulation with 10^4 bodies.

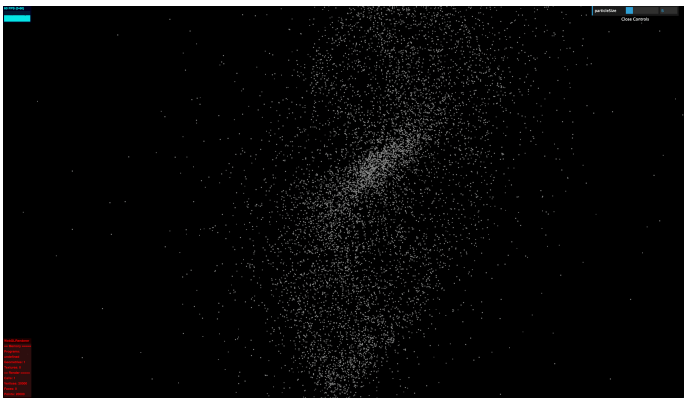

Fig. 10. Our Simulator - 1/4

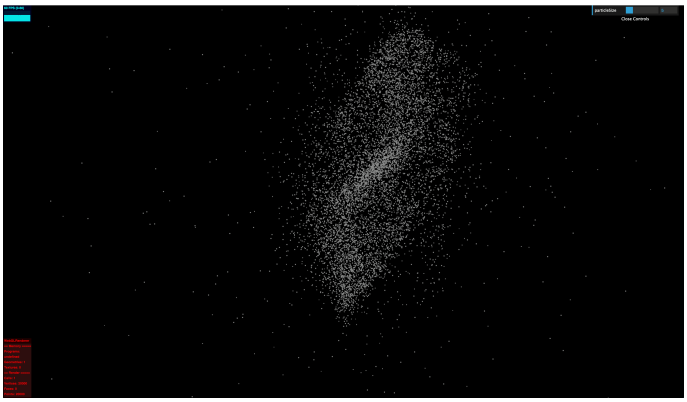

Fig. 11. Our Simulator - 2/4

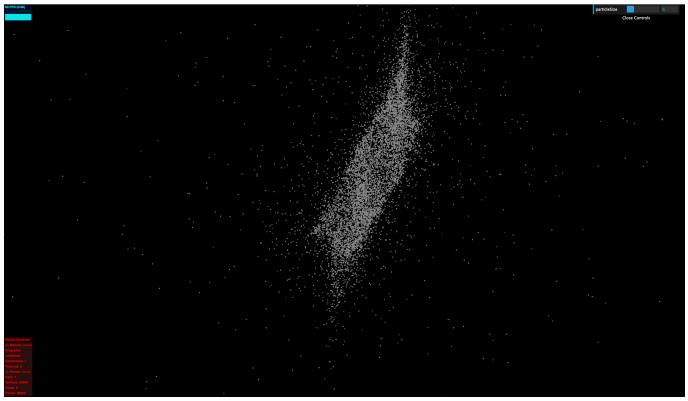

Fig. 12. Our Simulator - 3/4

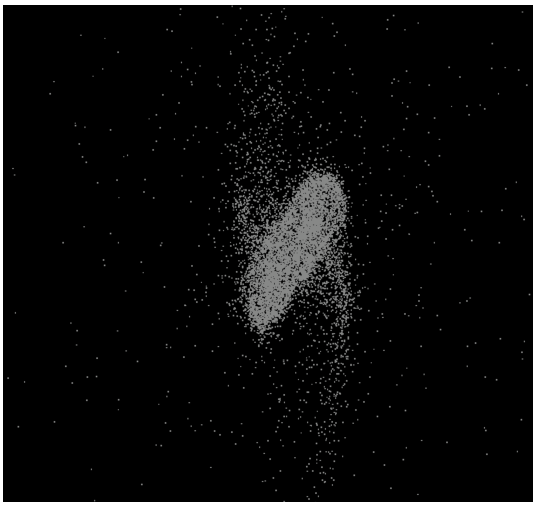

Fig. 13. Our Simulator - 4/4

Figure 10 shows the simulation of particles in an early stage where each particle is randomly positioned in the 2D-plance. In figure 11 we can observe that the force impacts work, thus attracting the particles to each other. Figure 12 and 13 finally show a rotation of the particle positions according to the center. In the top right corner one can adjust the particle radius size in order to enlarge or shrink the objects in the simulation.

D. Cluster setup

We were using up to 32 nodes with each 1 core, i.e., 32×1 processors. The nodes that were available are composed of the following hardware:

[32] 2 Xeon L5640@2.26 GHz, 24GB RAM
[16] 2 Xeon L5640@2.26 GHz, 48GB RAM
[1] 2 Xeon X7560@2.26 GHz, 1024GB RAM
[16] 2 Xeon E5-2660@2.2 GHz, 32GB RAM
[16] 2 Xeon E5-2660@2.2 GHz, 32GB RAM

In this setup we explicitly reserved nodes with each one core to illustrate a distributed network communication flow with real message passing.

VI. Validation and Experimental Results

In order to test and validate the implementation of our sequential and parallelized algorithm, it should be tested in different kind of setups and compare it also to the n-squared version. In order to get an insight in the overall behaviour of all the different performances a benchmarking diagram is provided including all the algorithms such that it is easy to spot significant differences in the computation time depending of course on the amount of simulated bodies in the universe.

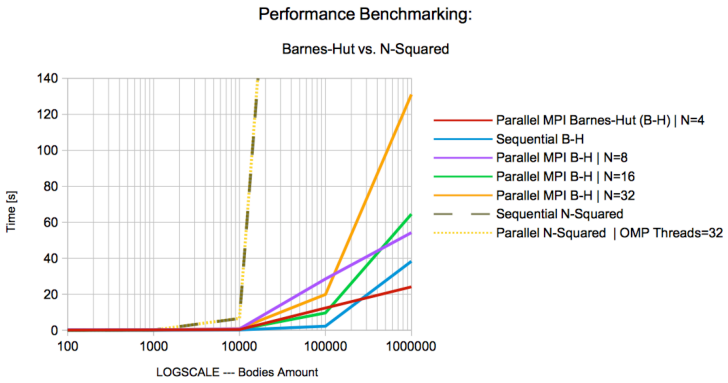

Fig. 14. Performance Benchmark

For the overall experimental setup we went with a total of 4 different parallel setups to test our approach on the Barnes-Hut algorithm. Those tests are performed on 4 up to 32 different nodes with 1 core each. In addition we tested the sequential version of the Barnes-Hut algorithm as well as a sequential and parallel version in OpenMP of the n–squared approach. This allowed us to provide a widespread view and compare the results with each other gathered on the HPC. As expected from the n-squared approach a polynomial growth in time is observed, what really was rather unexpected was that the inclusion of OpenMP in the n-squared approach made no significant difference, it nearly stayed the same. In the case of our sequential Barnes-Hut approach we could observe the complexity of $O(n \log n)$, it was now interesting to see how the parallel approach on the Barnes-Hut algorithm performed in comparison. A rather unexpected picture was to observe, compared to our sequential Barnes-Hut approach, the parallel versions with nodes greater equal 8 performed less the higher the amount of bodies went. We suppose that this is due to the way the parallel version was implemented, since we have heavily rely on the usage of MPI Send and Receive, the more processes we have the more messages have to be passed around. Therefore a golden number has to be found for the best behaviour, in our case this seemed to be 4 different nodes, it was performing better than the sequential approach when we exceeded the number of 200k bodies. But still the best behaviour would be if we increase the number of nodes more then the overall time should go down.

VII. Future Work

As our program should be considered as a proof of concept and a prototype, we still need to optimize several parts of the code.

First of all we need to reduce the amount of messages being send from the MPI, in case the time to send the tasks to clients and receive back the answers take more time than actually compute the tasks from the server himself. This can be done by improving MPI communication groups to reduce communication overhead. Moreover instead of choosing all available processors to compute we only take part of them to reduce the amount of passing messages.

In order to further reduce the amount of messages send from each process, the server could assign each process some fix bodies. Then, the server needs only to send the bodies at the beginning of the program including the initial velocity. As the clients calculate the force impact of each body, it is then for themselves possible to know the exact location of their bodies, thus reducing the messages send from the server as every client knows their assigned bodies locations.

Another approach is to deploy multiple servers, each maintaining its own slaves. Then, building a inter-server communication for constructing the whole tree using each sub-tree. This method is quit interesting and we keep it in memory for later tests.

Also, we would like to perform simulations on nodes with more cores. In the current simulation, we performed the executions on up to 32 nodes each having only one core. This implies bigger communication times since the nodes are *physically* separated. Therefore, we would like to combine the processors locally on each node by using more cores. We are convinced that this will positively impact our performance.

As check conditions are computational intensive, we would like to reduce them even more.

Furthermore we plan to implement CUDA to further speed up the calculations and benchmark it on the HPC cluster.

Additionally we would like to implement the system into C++, which allows us to be more flexible and compare the current version with C++ using other tools such as Boost.

Last but not least we also plan to implement a 3D version of our approach by maintaining an Oct-tree instead of a Quad-tree.

VIII. Conclusion

In order to pass some light on the executed work, we propose a different approach on the Barnes-Hut Algorithm by providing what we consider a much more efficient approach to construct the overall tree. This different approach enables a much more efficient manner to sort the bodies in to the tree, which will later be helpful for successfully parallelizing the tree construction. The current version of our algorithm can simulate up to more than 1 million bodies in less time than the sequential version of our Barnes-Hut approach. As

an additional feature, we have created an easy to use and very performance based simulator which does not require much of a setup such that one can easily test and run simulations.

This simulator can be seen as a proof of concept of the correct functioning of the algorithm as well as a visual feedback. After all, our parallelization relies on the heavy use of MPI methods for sending and receiving purposes, therefore the need for serialization was emerged, which we again tackled on our own. The custom serialization allows us to only rely on the information we need for a process to know upon reception.

The execution of the overall parallelization is accompanied by some difficulties on how someone shall approach the development of good functioning and yet parallelized Barnes-Hut algorithm. We can conclude that the Barnes-Hut algorithm is definitely not the easiest algorithm to parallelize due to it overall reappearing of recursion in it, this had also the side effect that the processes needed to know about the tree to do calculations, this as a side effect means that we had to deal with shared memory concerning the tree. The serialization as mentioned beforehand and the shared memory problem lead to an overall exhaustive use of MPI send and receives messages which as side effect are leading to a message overhead which can possibly explain why the parallel version performs less good the more nodes are used.

As a final word, the approximation delivered by Barnes-Hut looks stunning but the implementation of a reliable and parallel version can be a hard task to achieve due to the many mentioned obstacles. Nonetheless, we went for a new approach to construct the tree, succeeded in implementing a parallel version, that delivers useful results for a continuation on even faster and much more performance based approaches in the future.

IX. Acknowledgements

The experiments presented in this paper were carried out using the HPC facilities of the University of Luxembourg [12] see http://hpc.uni.lu.

References

[1] V. S. et al., "Simulations of the formation, evolution, and clustering of galaxies and quasars," *Nature*, vol. 435, no. 7042, pp. 629–636, 2005.

[2] Z. E. Musielak and B. Quarles, "The three-body problem," *Reports on Progress in Physics*, vol. 77, no. 6, p. 065901, Jun. 2014.

[3] D. C. Heggie, "The Classical Gravitational N-Body Problem," *ArXiv Astrophysics e-prints*, Mar. 2005.

[4] M. A. K. Mayez A. Al-Mouhamed, Nazeeruddin Mohammad, "Enhancing the performance of parallel n-body simulation," *Nature*, vol. 324, 1986.

[5] T. Ishiyama, K. Nitadori, and J. Makino, "4.45 Pflops Astrophysical N-Body Simulation on K computer – The Gravitational Trillion-Body Problem," *ArXiv e-prints*, Nov. 2012.

[6] Yu Hu and S. Lennart Johnsson, "A Data–Parallel Implementation of O(N) Hierarchical N–body Methods," *ACM/IEEE Conference on Supercomputing*, 1996.

[7] J. Barnes and P. Hut, "A hierarchical o(n log n) force-calculation algorithm," *Advance Computing Conference (IACC), 2017 IEEE 7th International*, Jul. 2017.

[8] M. D. S. Emanuele Del Sozzo, Lorenzo Di Tucci, "A highly scalable and efficient parallel design of n-body simulation on fpga," *Computer Physics Communications 183 (2012)*, Aug. 2012.

[9] H. H. L. A. R. K. Mathias Winkel, Robert Speck and P. Gibbon, "A massively parallel, multi-disciplinary barneshut tree code for extreme-scale n-body simulations," *Advance Computing Conference (IACC), 2017 IEEE 7th International*, vol. 183.

[10] M. Burtscher and K. Pingali, "An efficient cuda implementation of the tree-based barnes hut n-body algorithm," 2011.

[11] J. Bédorf, E. Gaburov, and S. Portegies Zwart, "A sparse octree gravitational N-body code that runs entirely on the GPU processor," *Journal of Computational Physics*, vol. 231, pp. 2825–2839, Apr. 2012.

[12] S. Varrette, P. Bouvry, H. Cartiaux, and F. Georgatos, "Management of an Academic HPC Cluster: The UL Experience," in *Proc. of the 2014 Intl. Conf. on High Performance Computing & Simulation (HPCS 2014)*. Bologna, Italy: IEEE, July 2014, pp. 959–967.

Appendix