# A hybrid parallization approach for N-body problem

XULIN YANG, The University of Melbourne

Deriving an efficient parallel algorithm for the *N-body problem* is one of the most challenging and classic topics in high-performance computing (HPC) topic. The major bottleneck is to come up with efficient communication pattern to exchange information between processors. We present two hybrid algorithms for *the naive approach* and *Barnes-Hut algorithm* to solve the *N-body problem*. Both of them are implemented in *OpenMPI* (Open Message Passing Interface) and *OpenMP* (Open Multi-Processing) and tested on the *Spartan* HPC facility. Comparative analysis of both algorithms is performed. The contributions are, we showed that the hybrid naive approach can achieve a speedup up to 60 for 2000 bodies on 12 nodes each with 16 threads. It has good scalability. The hybrid Barnes-Hut algorithm can achieve a speedup up to 10 with the same setting. And it has a faster execution time in practice.

CCS Concepts: • **Computing methodologies** → **Parallel computing methodologies**.

Additional Key Words and Phrases: OpenMPI, OpenMP, N-body

**ACM Reference Format:**
Xulin Yang. 2020. A hybrid parallization approach for N-body problem. 1, 1 (October 2020), 14 pages. https://doi.org/

## 1 INTRODUCTION

The computation power provided by a central processing unit (CPU) is limited. Running independent activities in a program on separate CPU cores or computer nodes can reduce the program's execution time [23]. Parallel computing is a way to achieve high performance for such heavy computation required programs. In this article, we focus on parallelizing the classic *N-body problem* with the naive $O(N^2)$ approach and the $O(NlogN)$ Barnes-Hut algorithm.

### 1.1 Problem definition

The problem we focused here to be parallelized is the ***N-body problem*** **in** $\mathbb{R}^3$. It is defined as follows: Given the initial state of bodies (position and velocity), what is its final state after a period of time? Assuming that bodies in the system won't collide.

The initial inputs are $N$ bodies $b_1, b_2, ..., b_N$ where each body $b_i$ has a mass $m_i$, a position $p_i$ in $(x, y, z)$ coordiantes and a velocity $v_i$ in $x, y, z$ directions. $d_{ij}$ is the euclidean distance between $b_i$ and $b_j$ as shown in the equation 1. The gravitational force on $b_i$ by $b_j$ is written as $f_{ij}$. The total force on a body $b_i$ is written as $f_i$.

Euclidean distance between $b_i$ and $b_j$ in terms $(x, y, z)$ coordinates in $\mathbb{R}^3$

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2} \tag{1}$$

One body's state is calculated from its previous state at discrete times, $t_0$, $t_1$, ..., $t_T$ with time intervals $\Delta t$, where $T$ is the number of iterations of updates. Calculation for $b_i$'s next state after $\Delta t$ from its previous state can be done in the following steps:

Author's address: Xulin Yang, xuliny@student.unimelb.edu.au, The University of Melbourne, 168, Grattan St, Parkville, Melbourne, Victoria, VIC 3052.


XXXX-XXXX/2020/10-ART $15.00
https://doi.org/

(1) Firstly, compute the forces $f_{ij}$ for all pairs of bodies in each direction by the equation 2, where the gravitational constant is $G = 6.67 * 10^{-11} m^3 kg^{-1} s^{-2}$ and $p_i$, $p_j$ are the position coordinate.

$$f_{ij} = \frac{G m_i m_j (p_i - p_j)}{{d_{ij}}^3}, i \neq j \tag{2}$$

(2) Secondly, compute the total force on each body $b_i$ in each direction by the equation 3.

$$f_i = \sum_{j=0, j\neq i}^{N} f_{ij} \tag{3}$$

(3) Thirdly, update the velocity by the acceleration calculated by the Newton's second law $F = m * a$ as shown in the equation 4 as well as the position as shown in the equation 5 in each direction.

$$v_i' = v_i + \frac{f_i}{m_i} * \Delta t \tag{4}$$

$$p_i' = p_i + v_i * \Delta t \tag{5}$$

where $v_i'$ and $p_i'$ are updated velocity and updated position respectively.

Above all, the sequential naive approach is summarized in the algorithm 1.

**Algorithm 1:** sequential naive approach $O(N^2)$

```
Input: N bodies, T, Δt;
Result: Updated N bodies for T iterations
for step ← 0 to T do
    for i ← 0 to N do
        compute force on body_i in each direction ;        // using equation 3
        i = i + 1 ;
    end
    for i ← 0 to N do
        update velocity for body_i in each direction ;     // using equation 4
        update position for body_i in each direction ;     // using equation 5
        i = i + 1 ;
    end
    step = step + 1 ;
end
```

## 1.2 Related work

The **naive approach** for N-body problem requires $O(N^2)$ time complexity for calculations mentioned in the section 1.1 in each time step because it computes pairwise forces.

**Barnes-Hut algorithm** [17] requires $O(Nlog(N))$ time complexity in each time step. It uses a tree-structured hierarchical subdivision of space into cubic cells (as shown in the figure 1) to approximate nearby bodies as a pseudo body. Each of the cubic cells has eight subcells for eight directions in $3^{rd}$ dimension space.

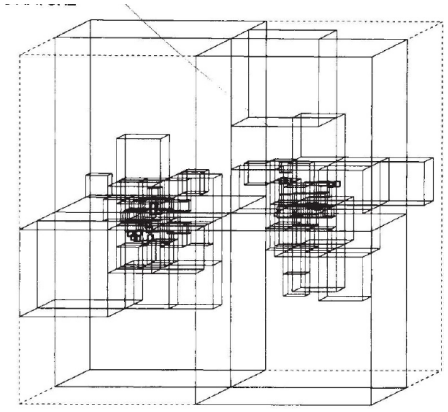

Fig. 1. Hierarchically divided cells (bodies) in octree for N-body in $\mathbb{R}^3$ [17]

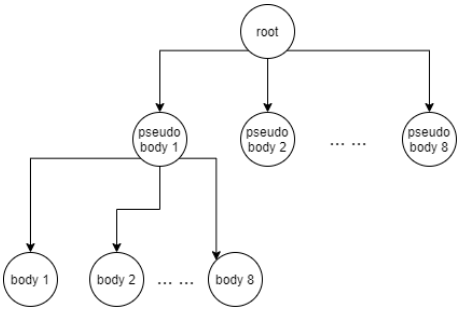


Fig. 2. an example constructed octree

It has following steps:

(1) Firstly, it recursively constructs an octree by inserting a body into one of eight children nodes from the root node until there is only one body in a leaf node as shown in the figure 2.

**Algorithm 2:** Barnes-Hut algorithm construct_octree()

```
Input: N bodies, root_bound: (x, y, z);
Result: root node
initialize root node with root_bound, 0 children, no pseudo body stored
for body_i in N bodies do
    cell <- root
    while cell has children do
        ith child <- identify which of eight subcell (space) that body_i's position located in ;
        cell <- cell's ith child ;
    end
    if cell already has body then
        construct a new cell with two bodies inserted to its children ;
    else
        add body_i to cell ;
    end
end
```

(2) Secondly, calculate each pseudo body's mass and position from leaf node to root recursively and stored in non-leaf node. The pseudo mass is calculated by $m_{pseudo} = \sum_i^{close\ bodies} m_i$. The pseudo position is calculate by $\frac{\sum_i^{close\ bodies} m_i * p_i}{m_{pseudo}}$.

**Algorithm 3:** Barnes-Hut algorithm compute_pseudo_bodies()

```
Input: a node cell;
if cell has no children then
    return body stored in the cell ;                                  // leaf
else
    for child in cell's children do
        child <- compute_pseudo_bodies(cell) ;
        if child not NULL then
            cell's pseudo body's mass += child's mass ;               // m_pseudo
        end
    end
    compute centroid from its children ;                          // pseudo position
    return cell ;
end
return NULL ;
```

(3) Thirdly, for $b_i$'s total force calculation, if the pseudo body stored in a node in the octree has euclidean distance $D$ such that $I/D < \theta$ where $I$ is the bound of the cell, $\theta$ is a fixed accuracy parameter $\sim 1$, the pseudo body can be used instead of its children node (close bodies). The updates are the same as the naive approach.

**Algorithm 4:** Barnes-Hut algorithm compute_total_force()

```
Input: root: octree, body_i;
if cell has no children then
    if body in cell != body_i then
        compute force between them ;                              // using equation 2
    end
else
    D <- euclidean_distance(body in cell, body_i) ;               // using equation 1
    if I/D < θ then
        compute force between them ; // approximate close bodies using equation 2
    else
        compute_total_force(for each root's child, body_i)
    end
end
```

Thus, the force and update calculation is simplified from $O(N)$ to $O(logN)$ by using the octree while sacrificing some precision.

Above all, the sequential Barnes-Hut algorithm is summarized in the algorithm 5.

**Algorithm 5:** sequential Barnes-Hut algorithm $O(NlogN)$

```
Input: N bodies, T, Δt;
Result: Updated N bodies for T iterations
for step ← 0 to T do
    max_bound ← (0, 0, 0) ;
    for i ← 0 to N do
        max_bound = max(max_bound, body_i) ;
    end
    root <- construct octree(max_bound, bodies) ;          // using algorithm 2
    compute_pseudo_bodies(root) ;                          // using algorithm 3
    for i ← 0 to N do
        compute force on body_i using octree ;  // using equation 3 with algorithm 4
        i = i + 1 ;
    end
    for i ← 0 to N do
        update velocity for body_i in each direction ;     // using equation 4
        update position for body_i in each direction ;     // using equation 5
        i = i + 1 ;
    end
    step = step + 1 ;
    delete octree ;
end
```

**Fast Multiple Method (FMM)** [15] is another way to calculate the *N-body problem* requiring $O(N)$ time complexity in each time step. Compared to the Barnes-Hut algorithm, calculations are performed from a cell (a cluster of close bodies) to cells-wise manner rather than a single body to cells-wise manner. Although FMM has a better time complexity, we choose not to investigate FMM because it is complicated, hard to determine the optimal N in practical and less accurate compared to the Barnes-Hut algorithm and naive approach [5].

| Literature | Description | Parallel platform | | | Peer-reviewed |
|---|---|---|---|---|---|
| | | OpenMPI | OpenMP | CUDA | |
| (Simon et al., 1991) [20] | Parallelize Barnes-Hut algorithm with a speedup of 380 on a 512-processor Ncube system. | Y | | | Y |
| (Singh et al., 1992) [22] | Parallelize Barnes-Hut algorithm on Stanford SPLASH | | Y | | Y |
| (Belleman et al., 2008) [3] | For $N \geq 105$, the 8800GTX outperforms the host CPU by a factor of 100. | | | Y | Y |
| (Green et al., 2010) [14] | Implement the naive approach on CUDA. | | | Y | Y |
| (Burtscher et al., 2011) [6] | An effective implementation of Barnes-Hut Algorithm on CUDA. | | | Y | Y |
| (Berczik et al., 2011) [4] | A parallel MPI/CUDA code on large GPU clusters for 6 million bodies. | Y | | Y | Y |
| (Miki et al., 2013) [19] | Hides the global memory access latency. | | | Y | Y |
| (Duy et al., 2012) [10] | The hybrid approach outperforms the pure MPI approach by a factor of 1.52 on 4-way cluster. | Y | Y | | N |
| (Nicholas et al., 2016) [7] | Parallelize Barnes-Hut algorithm. | Y | Y | | N |

Table 1. N-body literatures

As summarized in table 1, there are plenty of literatures on parallelizing N-body problem with single parallel platform, but the investigation on OpenMPI+OpenMP approach is insufficient. This motivates us to research such a hybrid approach.

## 2 PARALLEL ALGORITHM

### 2.1 Parallelization platform

The algorithms are implemented in C++ with distributed-memory parallelization OpenMPI [13] and shared-memory parallelization OpenMP [9] as a hybrid approach. So, it is in the Single Program Multiple Data (SPMD) category using CPU. OpenMPI is chosen as it can make parallel programs have good scalability by adding more nodes if there are enough independent tasks not yet parallelized. The only disadvantage is that the time reduced might be less than the communication time added. Unlike OpenMPI, OpenMP parallelize program by executing independent tasks in multiple threads on one node. So, it won't slow down the program if there is a heavy communication time cost. Usually, we don't have so many nodes available for OpenMPI. So, using multiple cores (threads) in one node can achieve another level of parallelization by adding more cores to each node. However, it might suffer from issues like false sharing and heavy context switching by fork and join threads in a loop if not carefully implemented. Another choice is to use Compute Unified Device Architecture (CUDA) [8] to parallel the program on the Graphics processing unit (GPU). However, we choose not to use CUDA as it is difficult to code as well as hard to debug compared to OpenMPI and OpenMP.

## 2.2 Parallel algorithms

The proposed two algorithms 6 and 7 for the N-body problem can be classified as synchronous and data-parallel. For both algorithms, the outer for loop (time step) cannot be parallelized because there is a dependency between each discrete time frame. So, parallelization takes place inside the outer for loop. Communication pattern such as *Bcast*, *AllReduce* and *AllGather* are used to synchronize data between processors.

**Algorithm 6:** Hybrid naive approach $O(\frac{N^2}{PQ})$

```
1  Input: N bodies, T, Δt, P: nodes, Q: threads;
   Result: Updated N bodies for T iterations
2  MPI_Bcast(N bodies, T, Δt) ;
3  p_start, p_end ← divide N bodies evenly for P nodes ;
4  for step ← 0 to T do
5      #pragma omp parallel for num_threads(Q)
6      for i ← p_start to p_end do
7          compute force on body_i in each direction ;        // using equation 3
8          i = i + 1 ;
9      end
10     MPI_ALLGather(computed forces) ;
11     #pragma omp parallel for num_threads(Q)
12     for i ← p_start to p_end do
13         update velocity for body_i in each direction ;     // using equation 4
14         update position for body_i in each direction ;     // using equation 5
15         i = i + 1 ;
16     end
17     MPI_ALLGather(updated bodies) ;
18     step = step + 1 ;
19 end
```

The hybrid OpenMPI+OpenMP approach for the naive algorithm is demonstrated in the algorithm 6. Compared to the sequential version, it broadcasts initial state and relevant information to all nodes from the root node (line 2). Then each node calculates the range of bodies it needs to update (line 3). After that, it uses threads (OpenMP) to parallelize the allocated bodies at two inner for loops (line 6 and 12) in each time step. MPI_ALLGather is used to synchronize calculated results in each step across all nodes (line 10 and 17). By using $P$ nodes and $Q$ threads, the time complexity of the naive algorithm is reduced from $O(N^2)$ to $O(\frac{N^2}{PQ})$ as two inner loops are parallelized in each time step.

**Algorithm 7:** Hybrid Barnes-Hut algorithm $O(NlogN)$

```
1  Input: N bodies, T, Δt, P: nodes, Q: threads;
   Result: Updated N bodies for T iterations
2  MPI_Bcast(N bodies, T, Δt) ;
3  p_start, p_end ← divide N bodies evenly for P nodes ;
4  for step ← 0 to T do
5      max_bound ← (0, 0, 0) ;
6      #pragma omp parallel for num_threads(Q) reducion(max:max_bound)
7      for i ← p_start to p_end do
8          max_bound = max(max_bound, body_i) ;
9      end
10     MPI_ALLReduce(max_bound, MPI_MAX) ;
11     root <- construct octree(max_bound, bodies) ;          // using algorithm 2
12     compute_pseudo_bodies(root) ;                          // using algorithm 3
13     #pragma omp parallel for num_threads(Q)
14     for i ← p_start to p_end do
15         compute force on body_i using octree ; // using equation 3 with algorithm 4
16         i = i + 1 ;
17     end
18     MPI_ALLGather(computed forces) ;
19     #pragma omp parallel for num_threads(Q)
20     for i ← p_start to p_end do
21         update velocity for body_i in each direction ;      // using equation 4
22         update position for body_i in each direction ;      // using equation 5
23         i = i + 1 ;
24     end
25     MPI_ALLGather(updated bodies) ;
26     step = step + 1 ;
27     delete octree ;
28 end
```

The hybrid OpenMPI+OpenMP approach for Barnes-Hut algorithm is demonstrated in the algorithm 7. It is similar to the algorithm 6 except it has a sequential $O(NlogN)$ time complexity octree construction (line 11), $O(logN)$ pseudo body calculation (line 12) and tree destruction (line 27). And different total force calculation for $b_i$ by using the octree in each time step as described in the section 1.2 (line 14) with time complexity decreases from $O(N^2)$ to $O(NlogN)$. A max_bound is needed for the root node in the octree. It is calculated and synchronized (MPI_ALLReduce) on line 5-10 with $O(\frac{N}{PQ})$. Compared to the hybrid naive approach, its overall complexity is still $O(NlogN)$ in each time step as there is a sequential octree construction. However, for two loops of the force calculation and bodies update respectively, the complexity is reduced from $O(\frac{N^2}{PQ})$ to $O(\frac{NlogN}{PQ})$ which has heavy constant time calculation as investigated in the section 3.2.

# 3 EXPERIMENTS

## 3.1 Dataset

Although we can use real galaxy dataset [21], these real-world dataset are too large to be uploaded to the Spartan. The data set is randomly generated with uniform distribution as summarized in the table 2. The parameters are summarized in table 3.

| Body data | Range | Unit | Reason |
|---|---|---|---|
| mass | [0, 1.0e30] | kg | The sun has 1.0e30 kg. |
| position (x, y, z) | [0, 5.9e12] | m | size of the solar system |
| velocity (x, y, z) | [-50000, 50000] | m/s | max orbit speed in the solar system |
| N | [10, 100, 500, 1000, 2000] | body | If too many, each test requires too much execution time |

Table 2. Randomly generated body

| Parameter | Value | Reason |
|---|---|---|
| $\Delta t$ | 60 | If too small, no updates observed (s) |
| $T$ | 1440 | 24h * 60 minutes |
| $G$ | 6.67e-11 | gravitational constant ($m^3 kg^{-1} s^{-2}$) |
| $\theta$ | 1.0 | precision parameter as mentioned in the section 1.2 |

Table 3. Program parameters

## 3.2 Result & Discussion

The parallel algorithms' performance are measure in terms of $speedup = \frac{sequential\ runtime}{parallel\ runtime}$ [11]. The results are generated by running programs on the HPC facility Spartan's snowy partition [18].

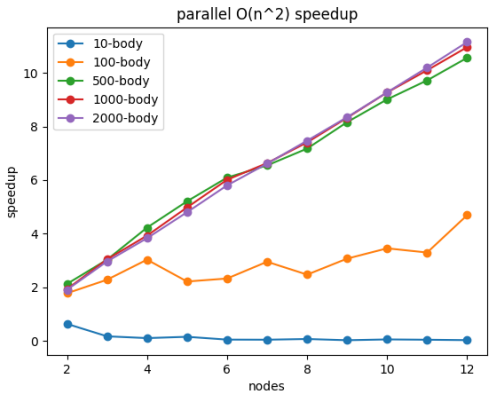


Fig. 3. Speedup for parallel naive approach by pure OpenMPI

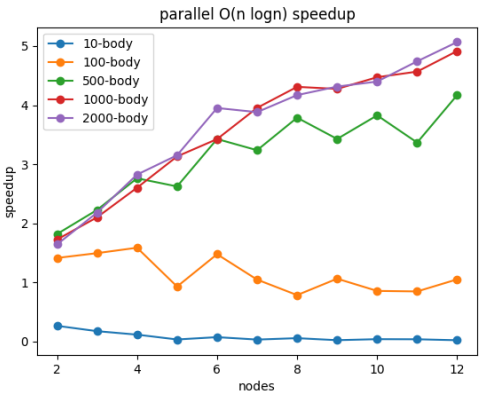


Fig. 4. Speedup for parallel Barnes-Hut algorithm by pure OpenMPI

Firstly, we consider only using OpenMPI to observe algorithms' behaviors. By the figure 3 and 4, speedup increases as nodes used increases. The parallelized naive approach and parallelized

Barnes-Hut algorithm can achieve linear and half linear speedup respectively if the number of bodies is at least 50 times of nodes (enough data to be parallelized). These observations support the Gustafson's Law [16]: the parallel program's speedup increases as the processors used increases. Additional communication overhead is introduced for both algorithms if there are insufficient bodies ($N$ = 10 and 100) to be parallelized. This observation supports the Amdahl's Law [1]: no matter how we increase the number of available processors, the maximum speedup is limited by the non-parallelizable parts in the program.

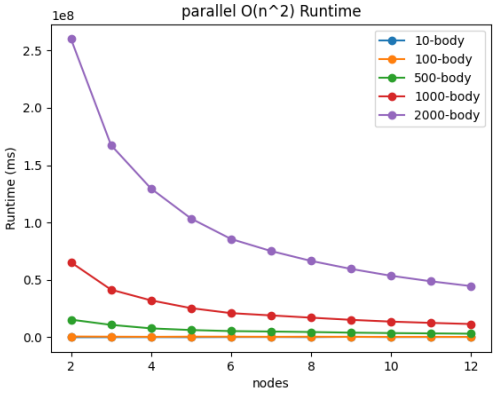


Fig. 5. Runtime for parallel naive approach by pure OpenMPI

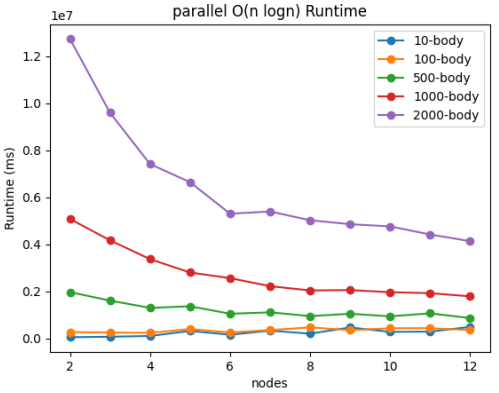


Fig. 6. Runtime for parallel Barnes-Hut algorithm by pure OpenMPI

By comparing figure 5 with 6 and 3 with 4, although the parallel Barnes-Hut algorithms achieves a smaller speedup, it has a faster execution time (10 times faster) than the parallel naive approach in all cases.

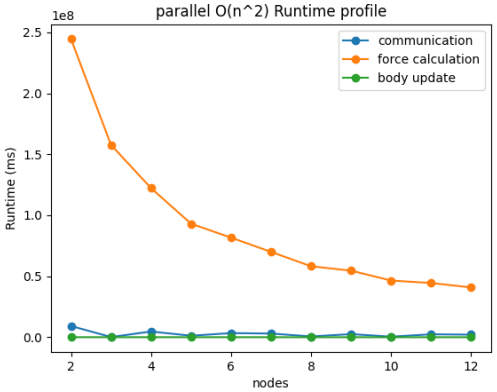


Fig. 7. Runtime profile for parallel naive approach by pure OpenMPI for 2000 bodies

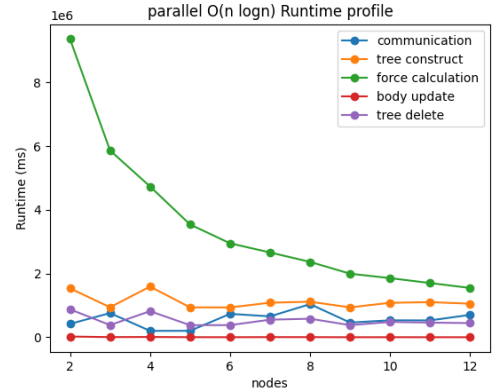


Fig. 8. Runtime profile for parallel Barnes-Hut algorithm by pure OpenMPI for 2000 bodies

By the figure 7 and 8, only the force calculation time decreased when nodes used increases. So, the major bottleneck for both algorithms is the force calculation. In other words, although

we have achieved some parallelism by dividing bodies over nodes, there still are several not parallelized bodies on each node shown by both figures when $nodes = 12$. What's more, the OpenMPI's communication time does not contribute to a large proportion of the program's execution time. In other words, the implementation of both algorithms achieves a good granularity ($granularity = \frac{execution\ time}{communication\ time}$). Similarly, the body update calculation doesn't cost too much execution time. For the parallel Barnes-Hut algorithm, although the tree construction takes $O(NlogN)$ time complexity, in the runtime, this part does not consume too much execution time compared to the force calculation. So, these motivate us to further parallelize bodies' calculation on each node by using multiple threads (OpenMP).

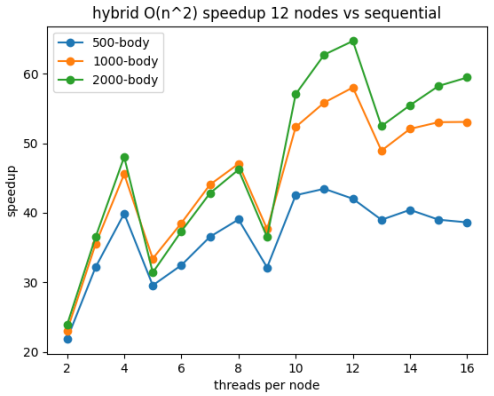


Fig. 9. Speedup of hybrid naive approach with fixed 12 nodes

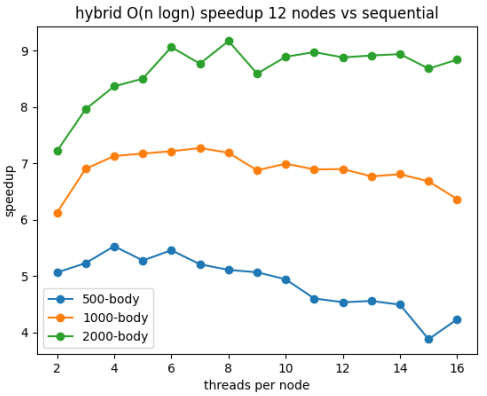


Fig. 10. Speedup of hybrid Barnes-Hut algorithm with fixed 12 nodes

Secondly, we consider the use of the hybrid approach by OpenMPI+OpenMP for $N = 500, 1000, 2000$ because for $N = 10, 100$ no parallelization can be achieved. By the figure 9, 2 to 6 times of speedup has been further achieved compared to the figure 3 by using more threads on each node. The speedup is not linear concerning the number of threads used because we have execution time variance on the Spartan. Above all, more resources (nodes or threads) can be used for the hybrid naive approach to achieve a better speedup. The speedup limit has not been achieved yet.

By the figure 10, 1 to 2 times of speedup has been further achieved compared to the figure 4 by using more threads on each node. However, speedup remains unchanged for 2000 bodies when threads used are larger than 10. This observation suggests that the maximum speedup has been achieved according to Amdahl's Law. The speedup decreases for 1000 and 500 bodies as threads used increases (when >8 threads in the figure 10). This is caused by the time saved by parallelizing calculation is limited while the time for threads fork-join operation time increases when more threads used. This makes sense because there are only 8 and 5 bodies per thread for 10 and 16 threads on each node respectively (limited parallelization).

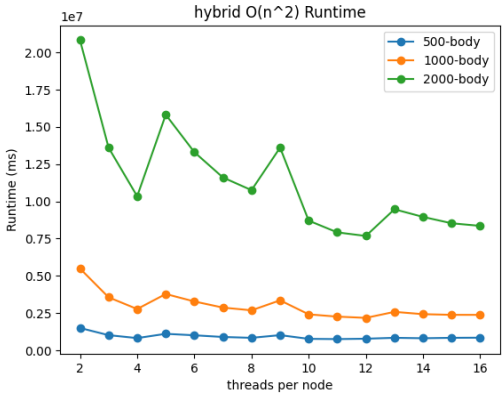


Fig. 11. Runtime for hybrid naive approach by OpenMPI+OpenMP

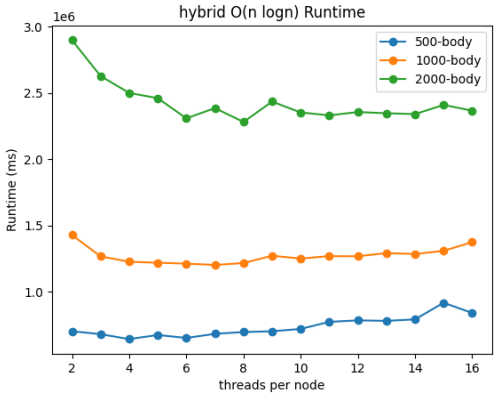


Fig. 12. Runtime for hybrid Barnes-Hut algorithm by OpenMPI+OpenMP

By figure 11 and 12, although the hybrid naive approach can achieve a higher speedup than the hybrid Barnes-Hut algorithm, its execution time is still longer than the hybrid Barnes-Hut algorithm in all cases.

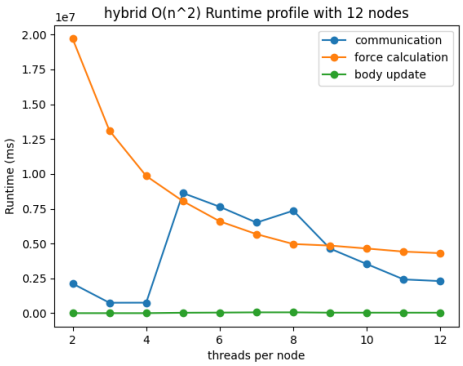


Fig. 13. Runtime profile for hybrid naive approach by OpenMPI+OpenMP with 2000 bodies

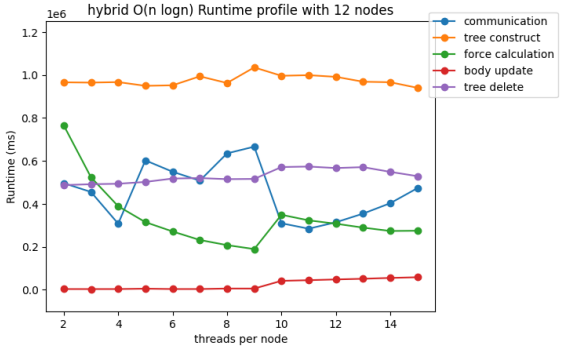


Fig. 14. Runtime profile for hybrid Barnes-Hut algorithm by OpenMPI+OpenMP with 2000 bodies

By the figure 13, the force calculation time decreases as threads used increases which means further parallelization is achieved by using more threads.

By the figure 14, the force calculation is parallelized so that its time consumption is smaller than the octree construction which matches our observation that the maximum speedup is achieved for hybrid Barnes-Hut algorithm.

For both of the figure 13 and 14, the communication time is not linear at presented y-scale because of the variance on the Spartan.

## 4 CONCLUSION

Firstly, a great amount of parallelism is achieved while not introducing large communication time for both proposed algorithms. A large granularity shows that an efficient communication pattern is derived when using OpenMPI. In other words, both algorithms' implementation are effective and feasible (achieve good speedup while not using too many resources) for solving the N-body problem. Secondly, the hybrid naive approach has a better speedup than the hybrid Barnes-Hut algorithm. However, hybrid Barnes-Hut algorithm is more suitable in practice than the hybrid naive approach as it has a smaller execution time. Thirdly, our findings suggest that two proposed hybrid algorithms should be used when there are enough bodies that can be parallelized. And both of them have good scalability when using more resources. Fourthly, the major time consumption to solve the N-body problem is the total force calculation. And it is shown that both algorithms are effective to parallelize this part. Fifthly, the hybrid naive approach can always achieve a higher speedup by adding more resources. The hybrid Barnes-Hut algorithm has a speedup limit. To further parallelize this algorithm, we can use *orthogonal recursive bisection* [12] to construct a balanced octree. Or we can distribute the octree construction on multiple nodes [2] to achieve further speedup. Finally, we conclude that the hybrid Barnes-Hut algorithm is better than the hybrid naive approach.

## REFERENCES


[1] Gene M Amdahl. 1967. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, spring joint computer conference*. 483–485.

[2] Mazzocca N. Venticinque S. Aversa R., Di Martino B. 2005. Performance Analysis of Hybrid OpenMP/MPI N-Body Application. *International Workshop on OpenMP Applications and Tools* 3349 (2005). https://doi.org/10.1007/978-3-540-31832-3

[3] Robert G Belleman, Jeroen Bédorf, and Simon F Portegies Zwart. 2008. High performance direct gravitational N-body simulations on graphics processing units II: An implementation in CUDA. *New Astronomy* 13, 2 (2008), 103–112.

[4] Peter Berczik, Keigo Nitadori, Shiyan Zhong, Rainer Spurzem, Tsuyoshi Hamada, Xiaowei Wang, Ingo Berentzen, Alexander Veles, and Wei Ge. 2011. High performance massively parallel direct N-body simulations on large GPU clusters. In *International conference on High Performance Computing, Kyiv, Ukraine*. 8–18.

[5] G Blelloch and Girija Narlikar. 1997. A practical comparison of N-body algorithms. 30 (1997), 81–96. https://doi.org/10.1090/dimacs/030/06

[6] Martin Burtscher and Keshav Pingali. 2011. An efficient CUDA implementation of the tree-based barnes hut n-body algorithm. In *GPU computing Gems Emerald edition*. Elsevier, 75–92.

[7] Nicholas J Carugati. 2016. The Parallelization and Optimization of the N-Body Problem using OpenMP and OpenMPI. (2016).

[8] Shane Cook. 2012. *CUDA Programming: A Developer's Guide to Parallel Computing with GPUs* (1st ed.). Morgan Kaufmann Publishers Inc.

[9] Leonardo Dagum and Ramesh Menon. 1998. OpenMP: an industry standard API for shared-memory programming. *Computational Science \& Engineering, IEEE* 5, 1 (1998), 46–55.

[10] Truong Vinh Truong Duy, Katsuhiro Yamazaki, Kosai Ikegami, and Shigeru Oyanagi. 2012. Hybrid MPI-OpenMP paradigm on SMP clusters: MPEG-2 encoder and n-body simulation. *arXiv preprint arXiv:1211.2292* (2012).

[11] Derek L Eager, John Zahorjan, and Edward D Lazowska. 1989. Speedup versus efficiency in parallel systems. *IEEE transactions on computers* 38, 3 (1989), 408–423.

[12] Florian Fleissner and Peter Eberhard. 2008. Parallel load-balanced simulation for short-range interaction particle methods with hierarchical particle grouping based on orthogonal recursive bisection. *Internat. J. Numer. Methods Engrg.* 74, 4 (2008), 531–553.

[13] Squyres J.M. Graham R.L., Woodall T.S. 2006. Open MPI: A Flexible High Performance MPI. *Parallel Processing and Applied Mathematics* (2006). https://doi.org/10.1007/11752578

[14] Simon Green. 2010. Particle simulation using cuda. *NVIDIA whitepaper* 6 (2010), 121–128.

[15] L. Greengard and V. Rokhlin. 1987. A fast algorithm for particle simulations. *J. Comput. Phys.* 135, 73 (1987), 325–348. https://doi.org/10.1006/jcph.1997.5706

[16] John L Gustafson. 1988. Reevaluating Amdahl's law. *Commun. ACM* 31, 5 (1988), 532–533.

[17] Barnes Josh and Hut Piet. 1986. A hierarchical O(N log N) force-calculation algorithm. *Nature* 324, 4 (1986), 446–449. https://doi.org/10.1016/j.nmd.2014.06.252

[18] Lafayette Lev, Sauter Greg, Vu Linh, and Meade Bernard. 2016. Spartan Performance and Flexibility: An HPC-Cloud Chimera. *OpenStack Summit* (2016). https://doi.org/doi.org/10.4225/49/58ead90dceaaa

[19] Yohei Miki, Daisuke Takahashi, and Masao Mori. 2013. Highly scalable implementation of an N-body code on a GPU cluster. *Computer Physics Communications* 184, 9 (2013), 2159–2168.

[20] John Salmon. 1991. Parallel N log N N-body algorithms and applications to astrophysics. In *COMPCON Spring'91 Digest of Papers*. IEEE, 73–78.

[21] J A Sellwood. 2014. GALAXY package for N-body simulation. *arXiv preprint arXiv:1406.6606* (2014).

[22] Jaswinder Pal Singh, Wolf-Dietrich Weber, and Anoop Gupta. 1992. SPLASH: Stanford parallel applications for shared-memory. *ACM SIGARCH Computer Architecture News* 20, 1 (1992), 5–44.

[23] Bulić P. Robič B. Trobec R., Slivnik B. 2018. *Why Do We Need Parallel Programming. In: Introduction to Parallel Computing*. Springer, Cham. https://doi.org/10.1007/978-3-319-98833-7