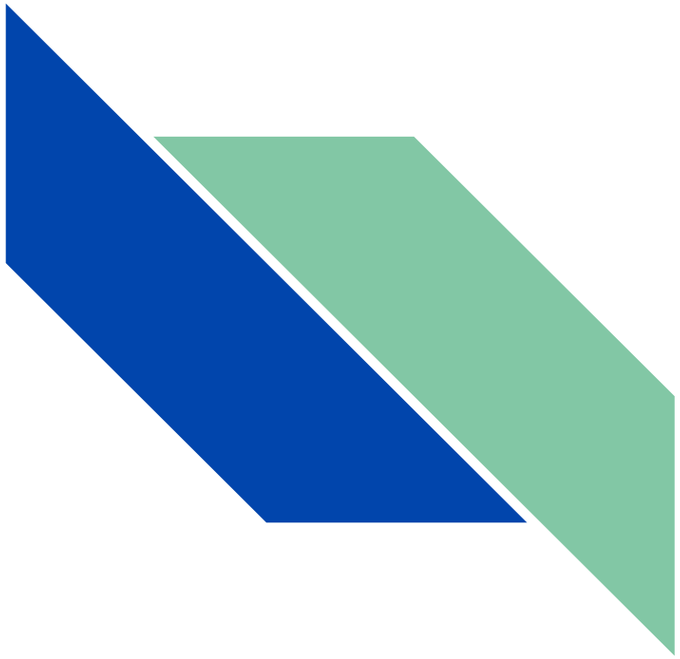


Parallel OpenMP and CUDA Implementations of the N-Body Problem

Tushaar Gangavarapu, Himadri Pal, Pratyush Prakash , Suraj Hegde, Geetha V
Email: geethav@nitk.edu.in

Please cite this article as: Gangavarapu T., Pal H., Prakash P., Hegde S., Geetha V. (2019) Parallel OpenMP and CUDA Implementations of the N-Body Problem. In: Misra S. et al. (eds) Computational Science and Its Applications – ICCSA 2019. ICCSA 2019. Lecture Notes in Computer Science, vol 11619. Springer, Cham, doi.org/10.1007/978-3-030-24289-3_16

Introduction

- Astrophysics problem of predicting individual motions of a group of celestial bodies interacting *gravitationally or otherwise*
  - Applications: Biology, Molecular and Fluid Dynamics and Semiconductor device simulation
- Gravitational N-body problem aims to compute the states of N bodies at a given time T based on their initial states of velocity and position
- Naive implementation of N body problem has a time complexity of $O(N^2)$ and O(N logN) for Barnes Hut algorithm

Please cite this article as: Gangavarapu T., Pal H., Prakash P., Hegde S., Geetha V. (2019) Parallel OpenMP and CUDA Implementations of the N-Body Problem. In: Misra S. et al. (eds) Computational Science and Its Applications – ICCSA 2019. ICCSA 2019. Lecture Notes in Computer Science, vol 11619. Springer, Cham, doi.org/10.1007/978-3-030-24289-3_16

Key Contributions

- Design of OpenMP and CUDA implementations of All-Pairs algorithm to parallelize force computation
- Design of OpenMP and CUDA implementations of Barnes-Hut algorithm to parallelize both force computation and mass distribution
- To present a detailed evaluation of the performance of the parallel algorithms in terms of speedup and running time using galactic datasets with number of bodies ranging from 5 to 30,002

Please cite this article as: Gangavarapu T., Pal H., Prakash P., Hegde S., Geetha V. (2019) Parallel OpenMP and CUDA Implementations of the N-Body Problem. In: Misra S. et al. (eds) Computational Science and Its Applications – ICCSA 2019. ICCSA 2019. Lecture Notes in Computer Science, vol 11619. Springer, Cham, doi.org/10.1007/978-3-030-24289-3_16

Problem Statement

Given the initial states (velocities and positions) of N bodies, compute the states of those bodies at time T using the instantaneous acceleration on every body at regular time steps.

We implement two algorithms to solve this problem,

- All Pairs - Suited for smaller number of bodies
- Barnes Hut - Scales efficiently for large number of bodies

Please cite this article as: Gangavarapu T., Pal H., Prakash P., Hegde S., Geetha V. (2019) Parallel OpenMP and CUDA Implementations of the N-Body Problem. In: Misra S. et al. (eds) Computational Science and Its Applications – ICCSA 2019. ICCSA 2019. Lecture Notes in Computer Science, vol 11619. Springer, Cham, doi.org/10.1007/978-3-030-24289-3_16

All-Pairs Algorithm

- Traditional algorithm is an exhaustive Brute-Force algorithm that computes instantaneous acceleration between each body and every other body
- For two bodies, B_i and B_j with masses m_i and m_j respectively, the force between them is inversely proportional to the square of the distance between them.
  - $F_{ij} = G * m_i m_j / r_{ij}^2$
  - $a_i = G * m_j / r_{ij}^2$

Please cite this article as: Gangavarapu T., Pal H., Prakash P., Hegde S., Geetha V. (2019) Parallel OpenMP and CUDA Implementations of the N-Body Problem. In: Misra S. et al. (eds) Computational Science and Its Applications – ICCSA 2019. ICCSA 2019. Lecture Notes in Computer Science, vol 11619. Springer, Cham, doi.org/10.1007/978-3-030-24289-3_16

Sequential All Pairs

Algorithm 1: Sequential All-Pairs Algorithm (2 Dimensions)

```
1 Function calculate_force() is
2     foreach i: body do
3         find_force(i, particles)
4 Function find_force(i: body, particles) is
5     foreach j in particles do
6         if j ≠ i then
7             d_sq = distance(i, j)
8             force[i].x += d_x * mass(i) / d_sqˆ3
9             force[i].y += d_y * mass(i) / d_sqˆ3
```

Computes pair-wise accelerations and updates the forces acting on each body.

Doubly nested for-loop resulting in time complexity $O(N^2)$

Please cite this article as: Gangavarapu T., Pal H., Prakash P., Hegde S., Geetha V. (2019) Parallel OpenMP and CUDA Implementations of the N-Body Problem. In: Misra S. et al. (eds) Computational Science and Its Applications – ICCSA 2019. ICCSA 2019. Lecture Notes in Computer Science, vol 11619. Springer, Cham, doi.org/10.1007/978-3-030-24289-3_16

Parallel All-Pairs Algorithm

- The traditional brute-force All-Pairs algorithm, with $O(N^2)$ time complexity serves as an interesting target for parallelization. This approach can be easily parallelized, as it is known in advance, precisely how much work-load balancing is to be done.
- The work can be partitioned using a simple block partition strategy since the number of bodies is known and updating each body takes the same amount of calculation .We assign each process a block of bodies each number of Planets/number of Processors in size, to compute forces acting on those bodies
- All Processes perform the same number of computations ,thus the workload is equally and efficiently partitioned among process

Please cite this article as: Gangavarapu T., Pal H., Prakash P., Hegde S., Geetha V. (2019) Parallel OpenMP and CUDA Implementations of the N-Body Problem. In: Misra S. et al. (eds) Computational Science and Its Applications – ICCSA 2019. ICCSA 2019. Lecture Notes in Computer Science, vol 11619. Springer, Cham, doi.org/10.1007/978-3-030-24289-3_16

OpenMP Implementation of All Pairs

Algorithm 3: OpenMP Implementation of the All-Pairs Algorithm

```
1  Function calculate_force() is
2      #pragma omp parallel for
3      foreach i: body do
4          find_force(i, particles)
5  Function find_force(i: body, particles) is
6      #pragma omp parallel for reduction (+ : force[i].x, force[i].y)
7      foreach j in particles do
8          if j ≠ i then
9              d_sq = distance(i, j)
10             force[i].x += d_x * mass(i) / d_sqˆ3
11             force[i].y += d_y * mass(i) / d_sqˆ3
```

Please cite this article as: Gangavarapu T., Pal H., Prakash P., Hegde S., Geetha V. (2019) Parallel OpenMP and CUDA Implementations of the N-Body Problem. In: Misra S. et al. (eds) Computational Science and Its Applications – ICCSA 2019. ICCSA 2019. Lecture Notes in Computer Science, vol 11619. Springer, Cham, doi.org/10.1007/978-3-030-24289-3_16

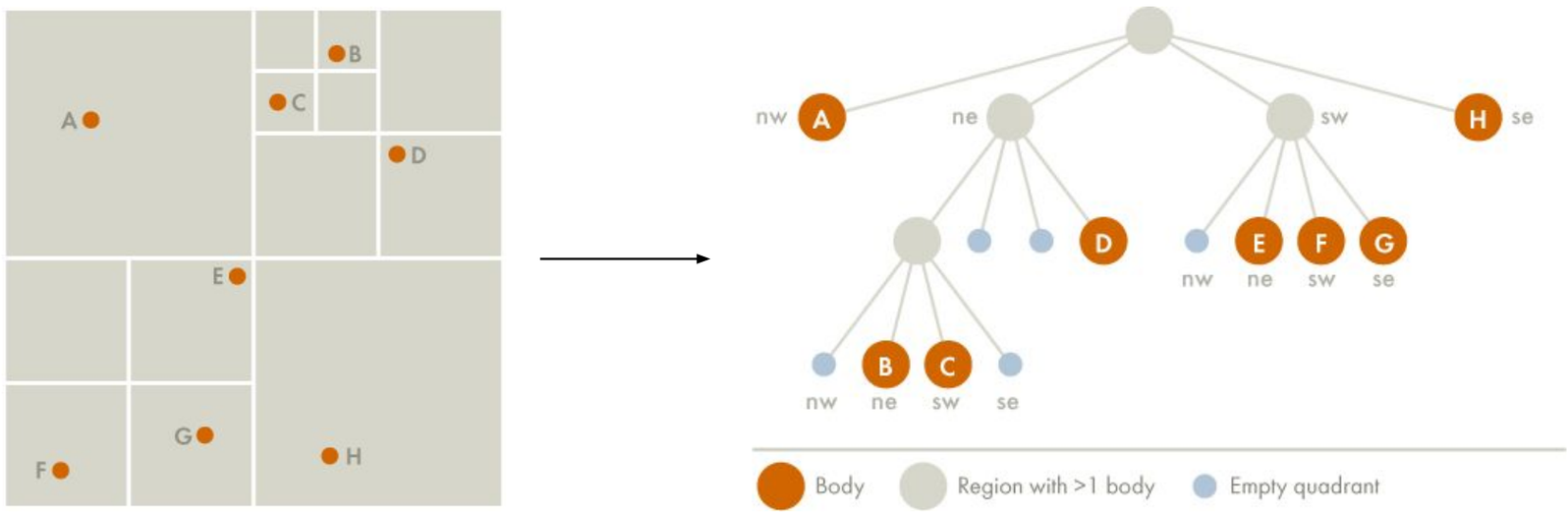
CUDA Implementation of All Pairs

Algorithm 4: CUDA Implementation of the All-Pairs Algorithm

```
1 Function calculate_force() is
2     foreach i: body do
3         find_force <<< BLOCKS, THREADS_PER_BLOCK >>>
            (index, particles, force, size)
4 Function find_force(i: body, particles, force, size) is
5     j = particles[treadIdx.x + blockIdx.x * blockDim.x]
6     if j ≠ i then
7         d_sq = distance(i, j)
8         force[i].x += d_x * mass(i) / d_sqˆ3
9         force[i].y += d_y * mass(i) / d_sqˆ3
```

Please cite this article as: Gangavarapu T., Pal H., Prakash P., Hegde S., Geetha V. (2019) Parallel OpenMP and CUDA Implementations of the N-Body Problem. In: Misra S. et al. (eds) Computational Science and Its Applications – ICCSA 2019. ICCSA 2019. Lecture Notes in Computer Science, vol 11619. Springer, Cham, doi.org/10.1007/978-3-030-24289-3_16

Representing points in a quad tree



Please cite this article as: Gangavarapu T., Pal H., Prakash P., Hegde S., Geetha V. (2019) Parallel OpenMP and CUDA Implementations of the N-Body Problem. In: Misra S. et al. (eds) Computational Science and Its Applications – ICCSA 2019. ICCSA 2019. Lecture Notes in Computer Science, vol 11619. Springer, Cham, doi.org/10.1007/978-3-030-24289-3_16

Inserting into the tree

```
Function insert_to_node(new_particle) is
    if num_particles > 1 then
        quad = get_quadrant(new_particle)
        if subnode(quad) does not exist then
            create subnode(quad)
        subnode(quad)→insert_to_node(new_particle)
    else if num_particles == 1 then
        quad = get_quadrant(new_particle)
        if subnode(quad) does not exist then
            create subnode(quad)
        subnode(quad)→insert_to_node(existing_particle)
        quad = get_quadrant(new_particle)
        if subnode(quad) ≠ NULL then
            create subnode(quad)
        subnode(quad)→insert_to_node(new_particle)
    else
        existing_particle ← new_particle
    num_particles++
```

Please cite this article as: Gangavarapu T., Pal H., Prakash P., Hegde S., Geetha V. (2019) Parallel OpenMP and CUDA Implementations of the N-Body Problem. In: Misra S. et al. (eds) Computational Science and Its Applications – ICCSA 2019. ICCSA 2019. Lecture Notes in Computer Science, vol 11619. Springer, Cham, doi.org/10.1007/978-3-030-24289-3_16

Recursive calculations on the tree

```
Function compute_mass_distribution() is
    if new_particles == 1 then
        center_of_mass = particle.position
        mass = particle.mass
    else
        forall child quadrants with particles do
            quadrant.compute_mass_distribution
            mass += quadrant.mass
            center_of_mass = quadrant.mass * quadrant.center_of_mass
        center_of_mass /= mass
```

```
Function calculate_force(target) is
    Initialize force ← 0
    if num_particles == 1 then
        force = gravitational_force(target, node)
    else
        if l/D < θ then
            force = gravitational_force(target, node)
        else
            forall node : child nodes do
                force += node.calculate_force(node)
```

Please cite this article as: Gangavarapu T., Pal H., Prakash P., Hegde S., Geetha V. (2019) Parallel OpenMP and CUDA Implementations of the N-Body Problem. In: Misra S. et al. (eds) Computational Science and Its Applications – ICCSA 2019. ICCSA 2019. Lecture Notes in Computer Science, vol 11619. Springer, Cham, doi.org/10.1007/978-3-030-24289-3_16

Parallelization of the Force calculation

Algorithm 5: Force Computation Parallelization of Barnes-Hut Algorithm

```
1  Function compute_force() is
2      #pragma omp parallel for
3      forall particles do
4          force = root_node.calculate_force(particle)
5  Function calculate_force(target_body) is
6      force = 0
7      if num_particles == 1 then
8          force = gravitational_force(target_body, node)
9      else
10         if l/D < θ then
11             force = gravitational_force(target_body, node)
12         else
13             #pragma omp parallel for
14             forall node : child nodes do
15                 #pragma omp critical
16                 force += node.calculate_force(node)
```

Please cite this article as: Gangavarapu T., Pal H., Prakash P., Hegde S., Geetha V. (2019) Parallel OpenMP and CUDA Implementations of the N-Body Problem. In: Misra S. et al. (eds) Computational Science and Its Applications – ICCSA 2019. ICCSA 2019. Lecture Notes in Computer Science, vol 11619. Springer, Cham, doi.org/10.1007/978-3-030-24289-3_16

Parallelization of the Mass calculations

Algorithm 6: Mass Distribution Parallelization of Barnes-Hut Algorithm

```
1  Function compute_mass_distribution() is
2      if new_particles == 1 then
3          center_of_mass = particle.position
4          mass = particle.mass
5      else
6          #pragma omp parallel for
7          forall child quadrants with particles do
8              quadrant.compute_mass_distribution
9              #pragma omp critical
10             mass += quadrant.mass
11             center_of_mass = quadrant.mass * quadrant.center_of_mass
12         center_of_mass /= mass
```

Please cite this article as: Gangavarapu T., Pal H., Prakash P., Hegde S., Geetha V. (2019) Parallel OpenMP and CUDA Implementations of the N-Body Problem. In: Misra S. et al. (eds) Computational Science and Its Applications – ICCSA 2019. ICCSA 2019. Lecture Notes in Computer Science, vol 11619. Springer, Cham, doi.org/10.1007/978-3-030-24289-3_16

Evaluation and Results

- All tests for sequential and OpenMP implementations were performed on nearly identical machines with the following specifications:
  - Processor : i5 7200U @ 4 × 3.1 GHz
  - Memory: 8 GB DDR3 @ 1333 MHz
  - Network: 10/100/1000 Gigabit Local Area Network (LAN) Connection
- All tests for CUDA implementations were performed on a server with the following specifications:
  - Processor : Intel Xeon Processor @ 2 × 2.40 GHz
  - Memory: 8 GB RAM
  - Tesla GPU : 1 × TESLA C-2050 (3 GB Memory)
- Datasets used : Princeton.edu galactic datasets
- Speed up = Time for sequential algorithm / Time for parallel algorithm

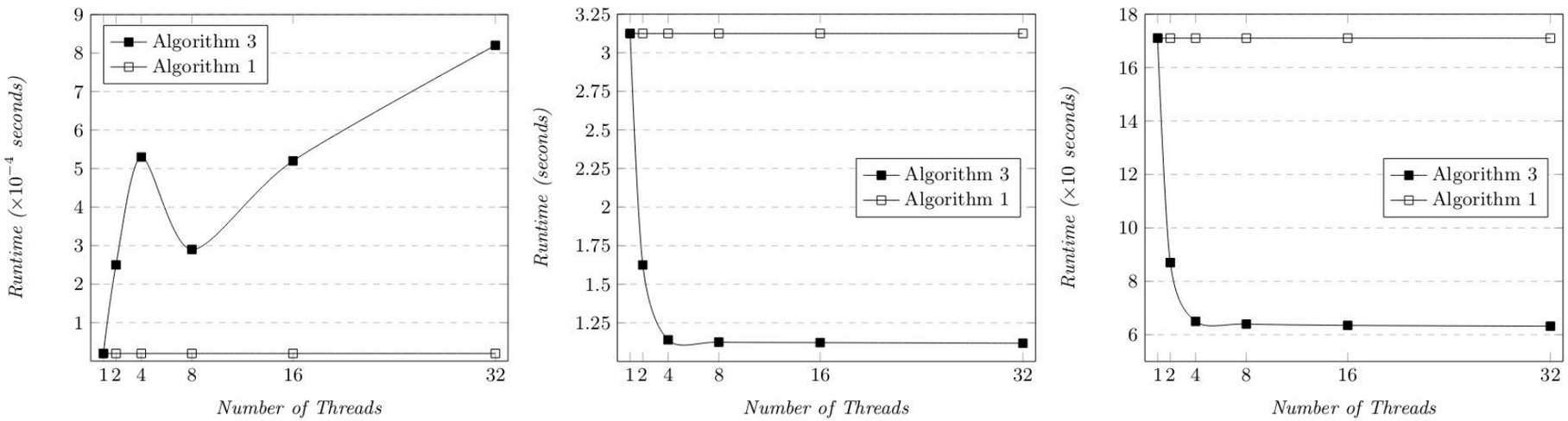
Please cite this article as: Gangavarapu T., Pal H., Prakash P., Hegde S., Geetha V. (2019) Parallel OpenMP and CUDA Implementations of the N-Body Problem. In: Misra S. et al. (eds) Computational Science and Its Applications – ICCSA 2019. ICCSA 2019. Lecture Notes in Computer Science, vol 11619. Springer, Cham, doi.org/10.1007/978-3-030-24289-3_16

Evaluation and Results

- The execution times collected to measure the performance and impact of parallelization is collected five times to overrule bias caused by any other system processes that are not under the control of the experimenter.
- In every run for time measurement, the order of experimentation for a given dataset is shuffled. The individual measurements are then averaged to represent the running time taken by the parallel algorithm accurately.

Please cite this article as: Gangavarapu T., Pal H., Prakash P., Hegde S., Geetha V. (2019) Parallel OpenMP and CUDA Implementations of the N-Body Problem. In: Misra S. et al. (eds) Computational Science and Its Applications – ICCSA 2019. ICCSA 2019. Lecture Notes in Computer Science, vol 11619. Springer, Cham, doi.org/10.1007/978-3-030-24289-3_16

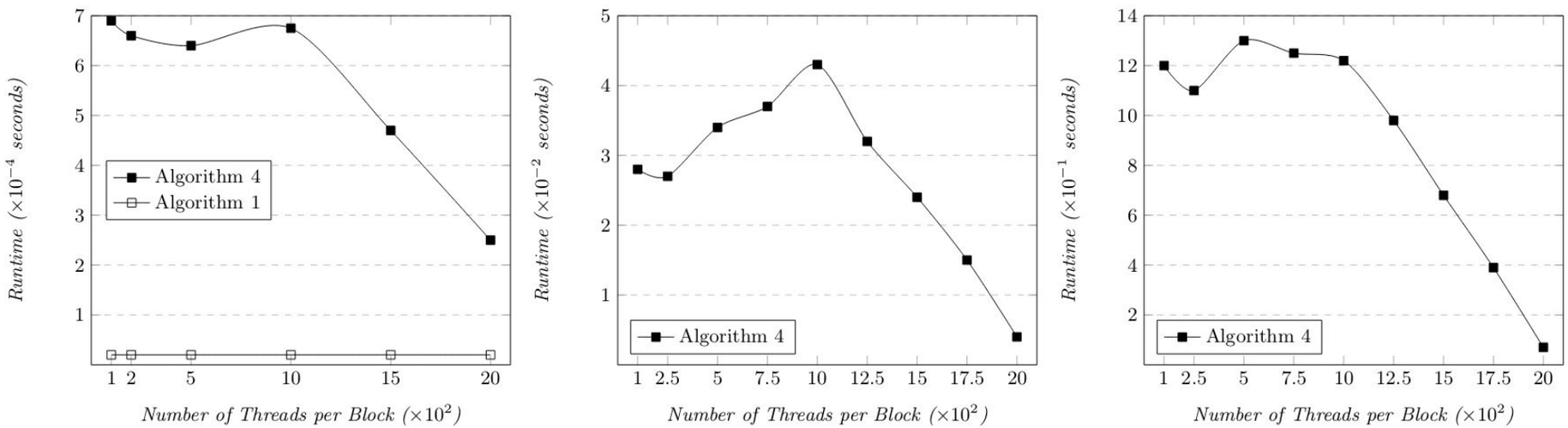
Evaluation and Results



Parallel vs. Sequential runtimes for All Pairs algorithm implemented in OpenMP for datasets of 5 bodies, 4000 bodies, and 30002 bodies respectively

Please cite this article as: Gangavarapu T., Pal H., Prakash P., Hegde S., Geetha V. (2019) Parallel OpenMP and CUDA Implementations of the N-Body Problem. In: Misra S. et al. (eds) Computational Science and Its Applications – ICCSA 2019. ICCSA 2019. Lecture Notes in Computer Science, vol 11619. Springer, Cham, doi.org/10.1007/978-3-030-24289-3_16

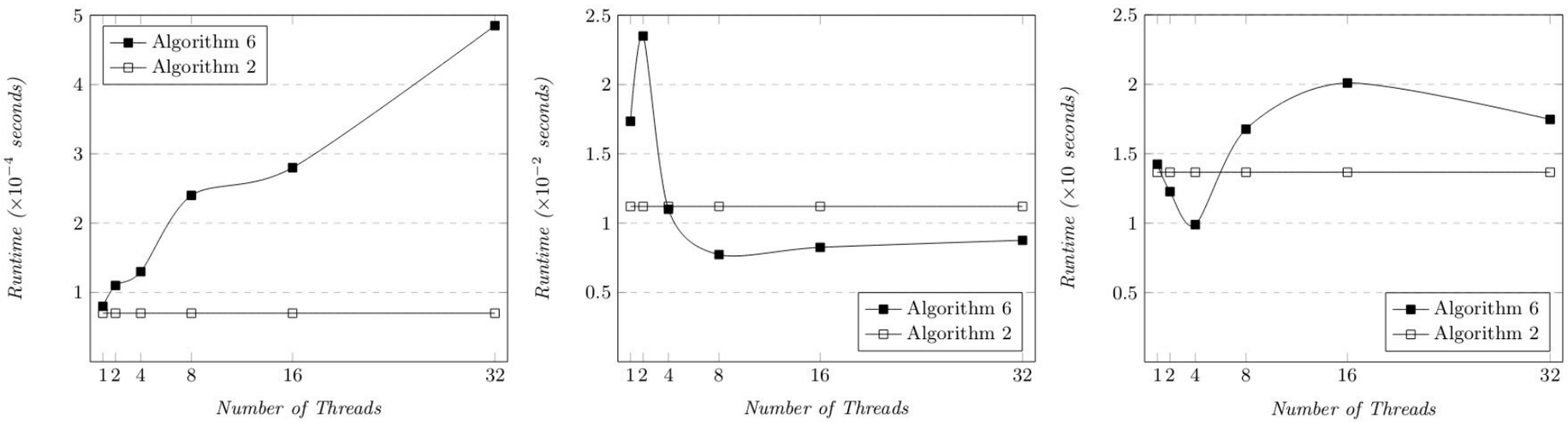
Evaluations and Results



Parallel vs. Sequential runtimes for All Pairs algorithm implemented with CUDA for datasets of 5 bodies, 4000 bodies, and 30002 bodies respectively

Please cite this article as: Gangavarapu T., Pal H., Prakash P., Hegde S., Geetha V. (2019) Parallel OpenMP and CUDA Implementations of the N-Body Problem. In: Misra S. et al. (eds) Computational Science and Its Applications – ICCSA 2019. ICCSA 2019. Lecture Notes in Computer Science, vol 11619. Springer, Cham, doi.org/10.1007/978-3-030-24289-3_16

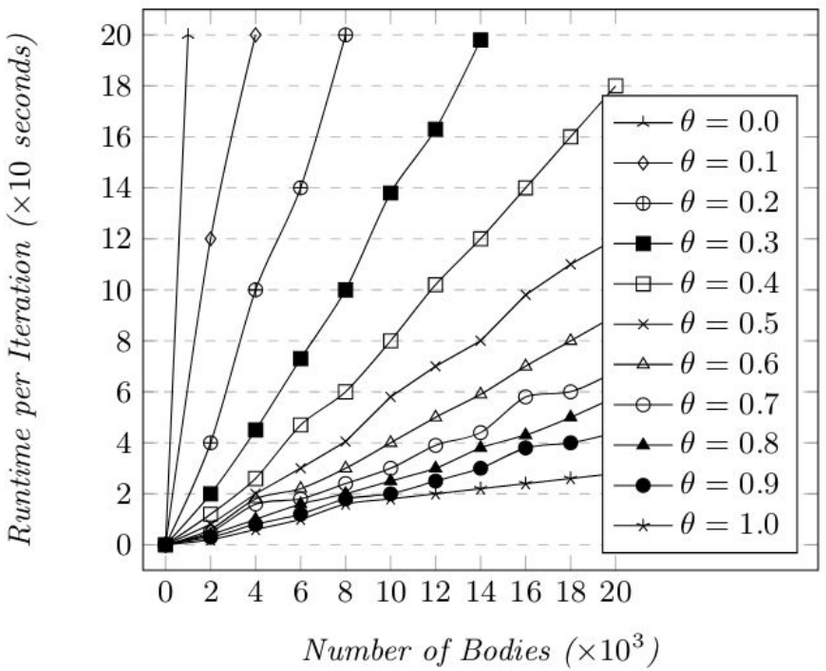
Evaluation and Results



Parallel vs Sequential runtimes for Barnes Hut algorithm implemented in OpenMP for datasets of 5 bodies, 4000 bodies, and 30002 bodies respectively

Please cite this article as: Gangavarapu T., Pal H., Prakash P., Hegde S., Geetha V. (2019) Parallel OpenMP and CUDA Implementations of the N-Body Problem. In: Misra S. et al. (eds) Computational Science and Its Applications – ICCSA 2019. ICCSA 2019. Lecture Notes in Computer Science, vol 11619. Springer, Cham, doi.org/10.1007/978-3-030-24289-3_16

Choosing the right θ



A smaller value of θ implies a higher accuracy since the depth of the force computations in the quad-tree are increased.

Conversely, a large value of θ reduces the accuracy and increases the number of computations.

Please cite this article as: Gangavarapu T., Pal H., Prakash P., Hegde S., Geetha V. (2019) Parallel OpenMP and CUDA Implementations of the N-Body Problem. In: Misra S. et al. (eds) Computational Science and Its Applications – ICCSA 2019. ICCSA 2019. Lecture Notes in Computer Science, vol 11619. Springer, Cham, doi.org/10.1007/978-3-030-24289-3_16

Analysis

- Since the All-Pairs algorithm can be broken down into hundreds of threads, CUDA proves to be the best solution. GPUs use massive parallel interfaces to connect with their memory and are approximately ten times faster than a typical CPU-to-memory interface.
- For inputs with a smaller number of celestial bodies, we observe that the sequential execution is faster than parallelized OpenMP code in case of both All-Pairs and Barnes-Hut algorithms.

Please cite this article as: Gangavarapu T., Pal H., Prakash P., Hegde S., Geetha V. (2019) Parallel OpenMP and CUDA Implementations of the N-Body Problem. In: Misra S. et al. (eds) Computational Science and Its Applications – ICCSA 2019. ICCSA 2019. Lecture Notes in Computer Science, vol 11619. Springer, Cham, doi.org/10.1007/978-3-030-24289-3_16

Analysis

- While the OpenMP implementations have a bottleneck over the number of cores on the test machine, parallelization with CUDA outperforms any such limitations. It can be observed that CUDA implementation provides an exponential decrease in the running time, which is because GPUs has an exponentially larger thread pool as compared to CPUs. It can be seen from that, for a smaller number of bodies, the parallel running time gradually decreases with the increase in the number of threads per block
- For 4, 000 bodies , the speedup of parallel implementation was observed to be 100 and for 30, 002 bodies , the speedup was approximately 250. These results establish the potential of CUDA in parallel programming and multi-processing support over traditional CPUs.

Please cite this article as: Gangavarapu T., Pal H., Prakash P., Hegde S., Geetha V. (2019) Parallel OpenMP and CUDA Implementations of the N-Body Problem. In: Misra S. et al. (eds) Computational Science and Its Applications – ICCSA 2019. ICCSA 2019. Lecture Notes in Computer Science, vol 11619. Springer, Cham, doi.org/10.1007/978-3-030-24289-3_16

Conclusions

- It was observed that until a certain level of parallelization the running time decreases and then increases afterward. The performance analysis of these methods establishes the potential of parallel programming in big data applications. We achieved a maximum speedup of approximately 3.6 with OpenMP implementations and about 250 with CUDA implementation.
- The OpenMP implementations experienced a massive bottleneck of the number of cores on the testing machine. Also, we experimentally determined the optimal value of the fixed accuracy parameter for an efficient trade-off between the running time and accuracy.

Please cite this article as: Gangavarapu T., Pal H., Prakash P., Hegde S., Geetha V. (2019) Parallel OpenMP and CUDA Implementations of the N-Body Problem. In: Misra S. et al. (eds) Computational Science and Its Applications – ICCSA 2019. ICCSA 2019. Lecture Notes in Computer Science, vol 11619. Springer, Cham, doi.org/10.1007/978-3-030-24289-3_16