GPGPU n-body simulation for 2-D games in Java

Jaan Janno, Supervisor: Eero Vainikko

Abstract—The aim of this work is writing a system for 2-D games for doing n-body simulations efficiently in parallel. N-body simulation is a method that is used to study interactions of particles. Quite commonly it is used to simulate the effect of gravity. It is often used in cosmology to predict the evolution of different structures in space, but it can also have uses in media and games. N-body simulation can be very demanding to compute. However, in essence the problem consists of a large number of small calculations, which makes it a prime candidate for parallel computing. GPGPU computing is becoming ever more relevant and with its highly parallel capabilities, it could be utilized for solving a problem such as the n-body problem. The result of this work is demonstrated by using it on a simple simulation with visualization. Some performance tests are also shown.

Index Terms—N-body problem, GPGPU computing, Barnes-Hut algorithm, Java, Aparapi, OpenCL.

1 Introduction

THE goal of this seminar work is creating a system for efficiently conducting n-body simulations with the intent of using this for 2-D games in Java. The two main methods that the system tries to make use of for efficiency are GPGPU computing and an advanced algorithm for n-body simulation.

GPGPU support is added through using the *Aparapi* library, which is a Java library for parallel computing. As the problem consists of a large number of small calculations, it could make excellent use of parallel computing.

The algorithm that will be used is the Barnes-Hut algorithm, which offers an $n \times log(n)$ complexity instead of the classic n^2 complexity of computing each pair's interactions.

The result is demonstrated by showing a simple visualization of what the system is capable of and performance test results are given.

April 26, 2016

2 N-body problem

The problem referred to as the *n-body problem* is the problem of computing the interaction between a number of distinct particles. The interactions could take into account gravity and other physical forces. [1]

Figure 1 illustrates an example of an n-body interaction. There are two larger massive bodies in the centre with smaller low-mass bodies in orbit around them. Each body in the system has a gravitational effect on every other body in the simulation.

If the number of particles we are studying is 2, then it is possible to analytically compute the evolution of the system. If, however, the set of bodies is larger, then the problem is no longer analytically solvable. Instead it must be solved by numerical integration by running simulations. [1]

The type of interaction this work models is the effect of gravity. Given that the number of particles we wish to simulate is definitely larger than one, the approach is to create a simulation.

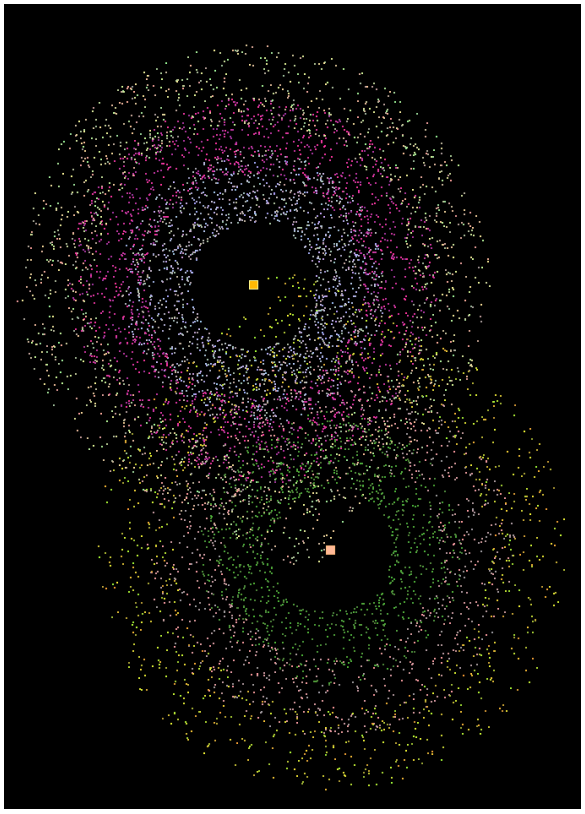

Fig. 1. A number of particles in orbit around 2 massive bodies.

3 Aparapi

Aparapi is a library for Java, which can be used to develop code that can be executed in parallel. It can utilize GPUs for conducting the computation. It interprets Java bytecode and translates it into OpenCL. This translation happens in runtime. After the translation the library offloads the computation onto GPUs when possible. However, it is also able to run the computation on the CPU if no required

hardware is detected. [3]

Java and OpenCL have significantly different architecture. Therefore there are some difficulties when writing code for Aparapi. The most significant are the following. [3]

1) No object orientation and non primitive data types are limited.
2) Recursion is not supported.
3) Arrays are maximally only 1-dimensional.

4 Barnes-Hut algorithm

The Barnes-Hut algorithm is a method for modelling the n-body problem with $n \times log(n)$ computation and $n \times log(n)$ memory complexity. This improvement compared to the classic n^2 approach comes from the fact that instead of computing the force contributions for each particle separately, the contributions from clusters of particles in the distance are aggregated. [2]

The algorithm can be divided into 2 tasks.

1) Constructing a Barnes-Hut tree.
2) Computing the forces affecting each particle.

The Barnes-Hut tree is a data structure which recursively divides the space the bodies exist in into 4 (for the 2-D case) until each subdivision contains exactly one body. This means the leaves of the tree are the bodies themselves and inner nodes describe clusters of bodies in the same subsections of space. Each node in the tree contains its mass centre coordinates, total mass, subsection width and 4 child nodes for each subsection of its space. [2]

This tree can be used to compute the affecting forces for each body in the system. The basic idea is to recursively go down in the tree until a given precision parameter is reached. Clusters of objects that are far away from another object don't have their contribution of gravity computed separately, but from an upper node that sums their mass and has their centre of mass. [2]

Figure 2 shows an example of a number of bodies in 2-D space and their representation in a Barnes-Hut tree. It can be seen that each body in the space is represented in the tree as a leaf node and each subsection in the space corresponds to an internal node in the tree.

5 Implementation

The implementation makes use of the Aparapi library and implements solutions to the problem using the n^2 algorithm on CPU and GPU and the Barnes-Hut algorithm on the GPU.

The link to the implementation repository can be found in Appendix A.

As mentioned in chapter 3, Aparapi has some limitations compared to native Java code. For this reason it was not possible to traverse the tree recursively, because recursion is not supported. Furthermore, due to limited support for custom data types, it was not possible to represent the tree traditionally with nodes existing in memory separately with pointers to child nodes.

For representing the tree data structure the *tree* was placed in an array of floating point numbers with each node

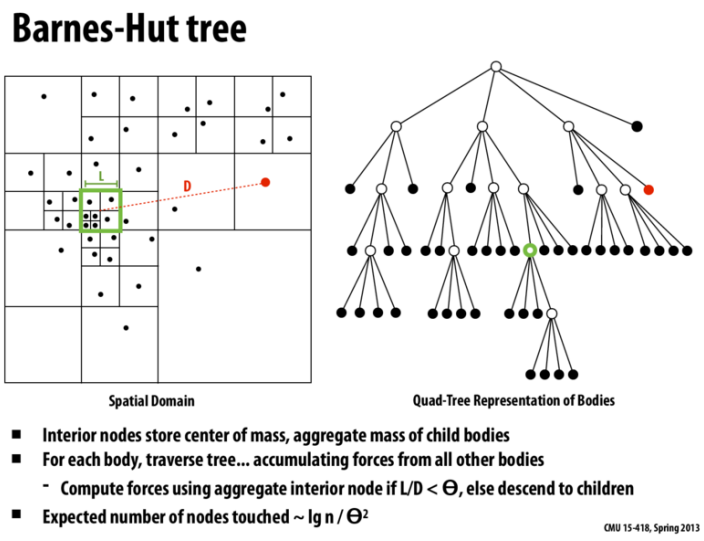

Fig. 2. Example of a Barnes-Hut tree. [4]

being represented by a number of consecutive floating point numbers.

To get around the lack of recursion, the algorithm traverses the tree iteratively by making use of reverse pointers to parent nodes and remembering highest visited index. The nodes are indexed depth-first, this allows us to know where the algorithm has to move in the tree.

To demonstrate the capabilities, the system also includes a simple visualization for the algorithms. It supports showing the bodies simulation in real time, visualizing the Barnes-Hut tree, adding new bodies, focusing on bodies, zooming and displaying various metrics. These metrics are zoom level, compute time, fps and node count in the tree when tree visualization is turned on. Figure 3 shows the result of simulating the bodies seen in Figure 1.

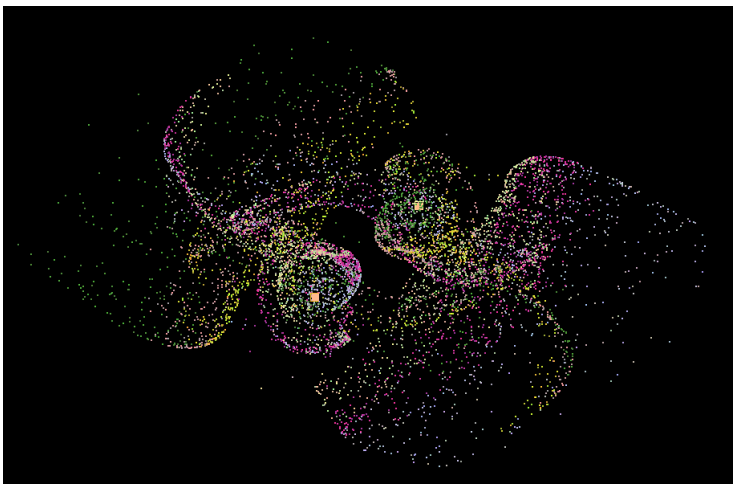

Fig. 3. Simulation example

As mentioned, the visualization can show the Barnes-Hut tree structure. This is shown in Figure 4. It can be seen, that space with a higher density of bodies contains more subdivisions in the tree. As mentioned in chapter 4, this comes from the fact the subdivisions are created until

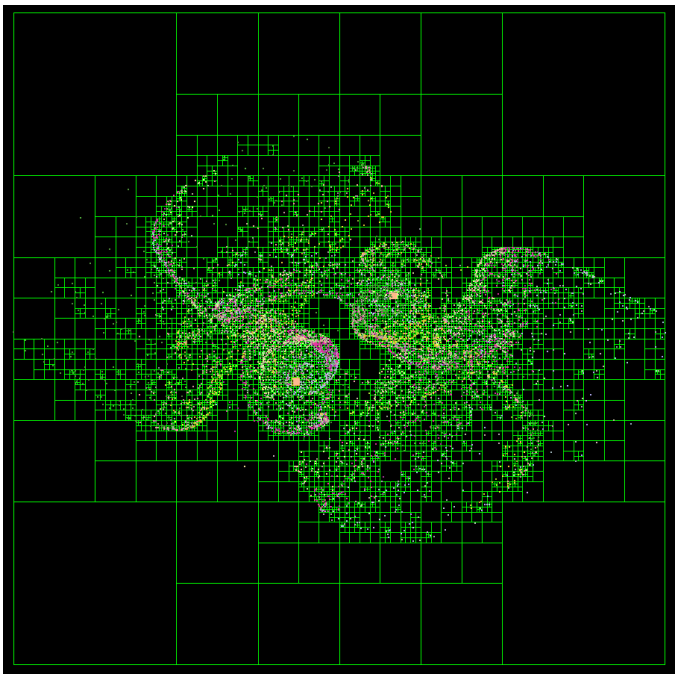

Fig. 4. Simulation Barnes-Hut tree example

each body is in an unique subsection in the space the tree represents.

6 Performance tests

The performance testing was conducted on a computer with the following hardware.

1) CPU: Intel i7 4770.
2) GPU: Nvidia 770.
3) VRAM: 4GB.
4) RAM: 16GB DDR3.

The benchmark on Figure 5 shows the performance scaling of the Barnes-Hut algorithm on the GPU. It can be observed that the algorithm is scalable to a high number of interacting bodies.

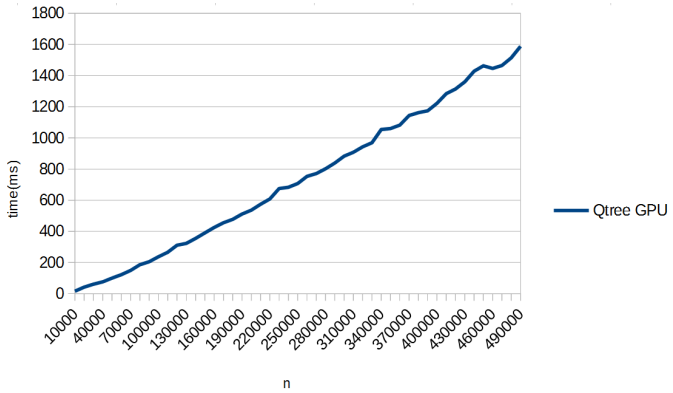

Fig. 5. Benchmark 1

The benchmark on Figure 6 compares the performance of Barnes-Hut on GPU to the n^2 algorithm on CPU and GPU. It can be seen that CPU does not scale very well in this benchmark. The GPU n^2 implementation, however, still seems quite competitive at this point. Furthermore, the GPU n^2 implementation would even be preferred for a lower number of bodies over the tree based solution.

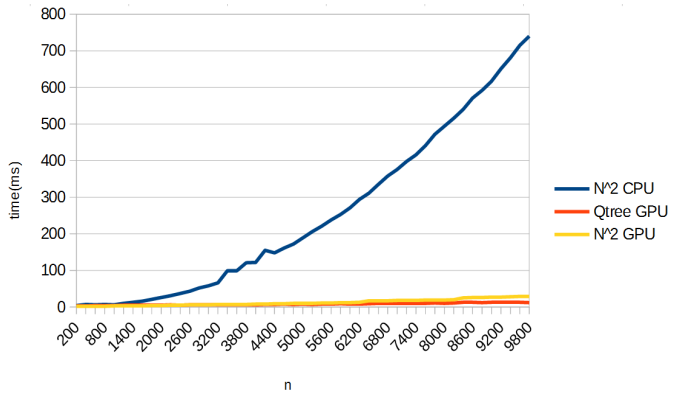

Fig. 6. Benchmark 2

The benchmark on Figure 7 further compares the performance of CPU n^2 algorithm to GPU Barnes-Hut. As expected, for a higher n the brute force approach is far from optimal.

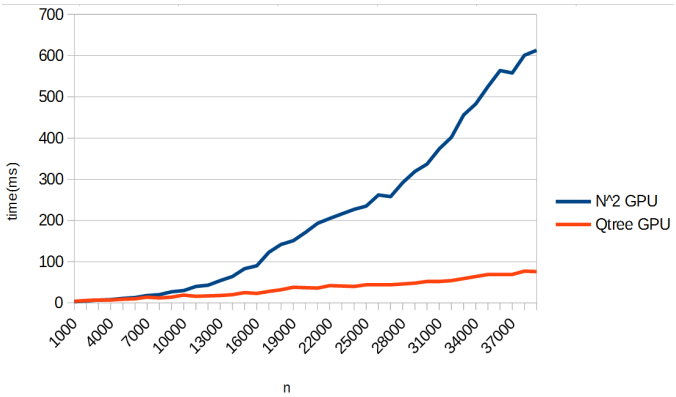

Fig. 7. Benchmark 3

7 Possible future additions

At this moment the Barnes-Hut implementation on the GPU has to do a lot of operations, which are actually more efficient on a CPU. This means a considerable number of branching operations and fine grained memory access - most with respect to traversing the Barnes-Hut tree. The implementation could perhaps be improved by traversing the tree on the CPU and aggregating the information needed for the computations, so the GPU could apply its raw performance better without the need for branching and fine grained memory access. This could combine the strong properties of both the CPU and GPU.

8 Conclusion

A system was implemented for n-body simulation.

The computation was moved from the CPU to the GPU, which showed good performance gains for the n^2 approach to simulation.

The Barnes-Hut algorithm was used to more efficiently compute the interaction of bodies with some approximation for bodies that are farther apart.

A visualisation package was added to the system which is capable of showing particle interactions and the Barnes-Hut tree that the bodies in the simulation generate. The user can interact with the visualization by creating new bodies and navigating the space.

APPENDIX A
PROJECT REPOSITORY

The repository of the solution can be found in the following link.

https://github.com/JaanJanno/NBody-GPGPU-Simulation

REFERENCES

[1] Trenti, Michele, and Piet Hut. "Gravitational N-body Simulations." arXiv preprint arXiv:0806.3950 (2008).
[2] Barnes, Josh, and Piet Hut. "A hierarchical O (N log N) force-calculation algorithm." nature 324.6096 (1986): 446-449.
[3] Aparapi documentation, https://github.com/aparapi/aparapi/blob/master/doc/README.md, accessed 26th April, 2016
[4] Parallel Programming Case Studies slides, Carnegie Mellon University, http://15418.courses.cs.cmu.edu/spring2013/lecture/casestudies, accessed 26th April, 2016