A Multithreaded N-Body Simulation

Sam Moore
20503628@student.uwa.edu.au

Contents

Abstract

We have implemented a Newtonian gravity simulation for N-Bodies using Euler's method. Parallelism using threads has been implemented in seperate versions of the program using POSIX Threads and OpenMP. The performance of the different implementations have been compared for different numbers of threads on a range of architectures including: Intel i5; Intel Core Duo; Intel Pentium IV; Intel Pentium; and ICRAR's Plieade's cluster at UWA. All implementations have been programmed in C.

1 Introduction

An N-Body simulation's purpose is to solve for the positions and velocities of an arbitrary number of interacting bodies using numerical methods. The number of bodies that can be simulated is limited by the real time available, as well as the desired amount

Complex systems involving large numbers of bodies, interacting over long periods of time, are often of more interest than simpler systems. Because of this, there is considerable motivation to improve the speed (amount of time simulated over real time required) of N-Body simulations. This report focuses on reducing the running speed of an N-Body simulation by introducing parallelism using threads.

2 Euler's Method - Overview

A template single threaded program was already provided for this assignment by UWA. Some modifications were made in order to simplify the process of introducing threading to the program, and in one case to correct an error in the program.

Euler's method (first order finite differences) has been used for this simulation. This method approximates a continuous time interval by a series of steps of length Δt, during which bodies undergo motion at constant velocity.

The algorithm in pseudo-code is:

```
FOR each time interval of duration dt LOOP
        FOR each body A // can be multithreaded
                COMPUTE the force acting on A
        DONE

        FOR each body A // can be multithreaded
                COMPUTE the velocity of A as:
                        V += (F / m) * dt // m is mass of A, F is force on A
                COMPUTE the position of A as:
                        X += V * dt
        DONE
DONE
```

This method can be parallelised by dividing the loops indicated between a number of concurrently running threads. It may seem that the two loops could be combined into a single loop. However, in the case where bodies are interacting, the force on any given body will depend upon the positions of other bodies. Consider a case involving two bodies A and B interacting in a single time step. If the force on A is computed, and A's position is updated before the force on B is computed, then the force experienced by B will not be equal and opposite to that experienced by A - voilating Newton's third law.

The template program provided by UWA makes the mistake of placing force and position updates for each body in the same loop. This can be simply fixed by moving the position updates to a seperate loop executed after the first. In the multithreaded programs, more complexity is required to ensure that *all* forces are updated before positions of bodies are altered.

3 Multi Threaded Version

3.1 POSIX Threads (pthreads)

3.2 OpenMP

3.3 Barne's Hut Algorithm for OpenMP

4 Performance Comparison