Constructing a Gravitational Simulator for the Cell Processor

Undergraduate Senior Thesis by John French
Department of Computer Science
University of Colorado
Fall 2009

Advisor: Michael Main, Associate Professor, University of Colorado

Thesis Committee: Michael Main, Associate Professor, University of Colorado
Elizabeth Bradley, Professor, University of Colorado
Dirk Grunwald, Professor, University of Colorado
Introduction

Numerical simulations of physical systems provide insight to their underlying dynamics. Moreover, such incites may allow for advances in the areas of engineering and science associated with those systems. The study of how these simulations might be improved to offer better performance, in so far as the insight provided by these simulations relies on ever more resource intensive simulations, motivates this study.

Numerical simulation of gravitational systems is a well studied area of Astrophysics. The problem this study intends to approach, the N-body problem, is also well studied with several numerical simulation methods being represented in literature. Three of these methods are particularly prominent.

First, so-called tree or hierarchal methods partition the input data into a nested tree of cells and then compute the force of each cell in the tree on each body. The force calculation of most tree methods relies on a multipole expansion for the larger cells of the tree, those closer to the root, and direct summation to compute the force of cells close to the body in the tree. Such methods are ordinarily of $O(N \log N)$ time complexity [1].

Second, harmonic expansion methods calculate the potential for N bodies by expanding it to a set of basis functions. While this approach has some disadvantages, the efficiency of the expansion method provides $O(N)$ time complexity [1].

Finally, a conceptually simpler approach to numerical simulation of the N-body problem, and the focus of this study, is the method of direct summation. Direct summation involves calculating the force between every pair of bodies in the system using Newton's Law of Gravitation, summing the forces on each body, and performing a numerical integration step. Although this method is constrained with time complexity $O(N^2)$, it is none the less still used. Despite its complexity, the method of direct summation typically has a greater accuracy than other approaches [1].

This study makes no attempt at implementing or analyzing other N-body approaches besides the direct approach. Indeed, the above discussion of other simulation methods is neither complete nor exhaustive of methods presented in literature. The above discussion should only serve as a comparison for the study of the approach at hand, the direct summation method.
The chosen approach to the N-body simulation, the method of direct summation, presents an opportunity to demonstrate the applicability of the Cell processor.

Physical Treatment

Newton's Law of Gravitation defines the force of gravity on mass 1 exerted by mass 2, a 'force pair interaction':

$$ F = -\frac{Gm_1m_2}{|r|^3} \hat{r} $$

In addition, every mass in space acts on every other mass. Therefore, in a system of N masses, the following diagram represents the possible N²-N pairings:

$$
\begin{bmatrix}
\emptyset & F_{12} & \ldots & F_{1(n-1)} & F_{1n} \\
F_{21} & \emptyset & \ldots & F_{2(n-1)} & F_{2n} \\
\ldots & \ldots & \ldots & \ldots & \ldots \\
F_{(n-1)1} & F_{(n-1)2} & \ldots & \emptyset & F_{(n-1)n} \\
F_{n1} & F_{n2} & \ldots & F_{n(n-1)} & \emptyset \\
\end{bmatrix}
$$

The elements of the above table represent the calculations of the force between 2 bodies that must be executed to find the component forces on each body.

The diagonal elements of the above table are null because the particle masses being considered have no gravitational effect on themselves. However, the above table can be further simplified. Newton's third law of motion states that whenever a body exerts a force on another, the second body exerts a force equal in magnitude and opposite in direction on the first body. Simply, $F_{ij} = -F_{ji}$. Therefore, $F_{ji}$ can easily be obtained by a calculation of $F_{ij}$, by taking the inverse of $F_{ij}$. By only executing the calculations enumerated by the lower triangular section of the above table the number of force pair interactions that must be computed is reduced by half to $1/2*(N^2 - N)$. By Newton's third law, the upper triangular sector can be easily computed by taking the inverse of the corresponding operation in the lower triangular sector.

To construct a gravitational direct summation simulator requires the use of Newton's Law of Gravitation. However, due to numerical relaxation effects, and for the simplicity of calculations, Newton’s Law of Gravitation is reformulated as such:

$$ F = -\frac{m_1m_2}{(x^2 + y^2 + z^2 + \epsilon^2)^{3/2}} \hat{r} $$
The units of the law of gravitation are taken to be such that $G = 1$. In the original formulation, were 2 masses to become very close together $|r| << 1$, the denominator of the equation could become as small as to lead to numerical errors. Therefore, epsilon is introduced to enforce a 'collision-less' condition.[1]

**Cell Architecture Overview**

The Cell Broadband Engine Architecture (CBEA) includes several features which motivates interest in its performance on scientific codes, and N-body direct sum simulations in particular. The Cell processor is a heterogeneous processor consisting of two types of processing elements. Rather than consisting of one or more identical 'core's with associated pipelines and logic units, the Cell incorporates 2 specialized types of computing elements. The first is the PowerPC Processing Element (PPE), which is exposed to the programmer as a 64 bit PowerPC processor with a 32 Kbyte L1 instruction cache and a 32 Kbyte L1 data cache. The second type of computing element on the Cell is the so called Synergistic Processing Element (SPE). An SPE consists of a Memory Flow Controller (MFC), which supports DMA access to the main memory, a Local Storage (LS) area of approximately 256Kbytes in size, and an SPU, a 128-bit Single Instruction Multiple Data (SIMD) vector processor. A Cell chip is typically clocked at 3.2 GHz.

These elements communicate over an on-chip bus known as the Element Inter-Connect (EIB).

**Hardware**

The hardware available for testing during this study consists of a Sony Playstation 3 (PS3) video game console. The PS3 provides a low cost, off-the-shelf hardware environment for testing programs written for the Cell processor. The pre-installed Sony PS3 operating system allows the installation of a 3rd party OS to a separate partition of the console's disk. However, this 3rd party OS does not have access to the full feature set of the PS3 system.

The Cell processor deployed in the PS3 system differs in a significant way from a commodity Cell processor. First, of the 8 SPE's available to commodity Cell processors, only 6 are available to a 3rd-party operating system running on a PS3. Of the 2 'forbidden' SPE's, 1 is involved in executing a hypervisor to the 3rd party operating system. The second either acts as a redundant check to the first, or is disabled by a manufacturing defect.

**Build Environment**
The N-body simulator developed as a part of this study ran on a PS3 running Yellowdog Linux 6.1 as a 3rd party guest operating system. All of the code requires the IBM Cell SDK version 3.1 and the SPE runtime library version 2.1 (libspe2).

**Programming for the Cell**

Input for the implementation of the N-body Gravitational Simulator (NGS) is defined as a flat text file. The first line of the file contains an integer representing the number of bodies to be simulated. Every subsequent line is defined as 7 floats and an integer, representing 6 initial conditions on the body, its position and velocity in 3 dimensions, the mass of the body, and a unique body identifier number. There are $N + 1$ lines in the input file, where $N$ is the number on the first line.

Using an input file, the NGS generates a master list of bodies in the simulation. Given the size of the master list, the memory requirements per body on an SPE, and the upper limit on the size of the SPE's LS, the NGS calculates the correct number of SPE contexts to create such that the memory requirements of the partitioned list segment size do not exceed the memory available to each SPE.

It is possible that the NGS code will generate more SPE contexts than there are hardware SPEs, making context switching on the SPE inevitable. However, there seems to be no other option that does not involve communication with the PPE. Where as a normal processor might have the option of paging its memory to disk when the demand for resources becomes larger than available physical memory, an SPE has no such built in ability. In addition, the SPE LS acts as a circular memory area and provides no memory access protection[3], allowing running programs to over-write themselves. Therefore, allocating large amounts of memory on the SPE will result in unpredictable behavior, as the stack wraps around to over write the executing program. One possible solution, unexplored in this study, would be to attempt to 'page' sections of memory from the SPE to the PPE, manually or through the use of an API. However, in the interest of simplicity, this study allows the hardware SPE's to be over-subscribed by SPE contexts to prevent a context from handling an input so large as to over flow the LS available to the hardware SPE.

Once the number of SPE contexts, $K$, is determined, NGS partitions the input as evenly as possible between the $K$ contexts. NGS creates a circular, singly linked list of $K$ parameter structures to carry information about each context's section of the master list.

After the creation of the $K$ parameter structures, one for each SPE context, NGS opens a new
thread for each context, passing the corresponding parameter structure. After this point in the NGS the program flow on the PPE is limited to supporting the threads which contain the SPE contexts, recording the results of the calculations that occur on the SPEs and acting as a storage server for the DMA requests from the SPEs.

Each SPE context is initialized by a thread running on the PPE. The SPE context, now executing on a physical SPE, has no direct access to the PPE’s memory, where the corresponding parameter structure, and the input master list is stored, but is passed an effective memory address to its parameter structure in the PPE's memory area. The SPE context must declare space for the parameter structure, and then perform a DMA request to transfer the data at the memory address on the PPE, to the address of the newly declared parameter structure in LS.

Given the parameter structure in memory the SPEs can perform the first phase of computation, copying the section of the master list defined in the parameter structure from the PPE to the LS via a DMA request, then calculating the force pair interactions only between the bodies defined in the allotted section of the list, and the acceleration on each body as a result of those interactions.

The second phase of processing requires a second parameter structure, acquired by traversing the linked-list structure of the parameter structures using DMA requests to the main memory. The SPE transfers the segment of the master list defined by this second parameter structure into memory. Because all segments of the master list defined by the linked-list of parameter structures are sequential and have no overlap, this phase is referred to as the ‘disjoint’ computation phase. During this phase an SPE calculates the force pair interactions between the elements of either list, but not the interactions between elements of the same list. The results of these force pair interactions are accumulated in the same way as the results of the previous phase. SPEs perform disjoint computation phases until they have reached the initially assigned parameter structure in the linked list, as this will be a signal that the SPE has computed the force pair interactions between all the elements on its original segment of the master list and all the other segments.

The completion of the disjoint compute phase concludes the force pair calculations. There after an SPE would be concerned with the numerical integration of the acceleration caused by all the force pair interactions on the bodies on its segment of the master list. However, this study has fallen short of its goal to provide a full gravitational simulator, and does not support the originally planned Runge-Kutta-Fehlberg integration step.
Results

While this study has fallen short of its initial goal of constructing a full gravitational simulator with a numerical integrator with a high order of accuracy (Runge Kutta Fehlberg 45), it has produced a mechanism which performs the necessary dispatch and direct summation force calculation.

During this study, various difficulties associated with programming for the Cell processor were encountered, such as the dispatch and computation on large datasets, requiring large amounts of memory on the SPEs.

Conclusions

While this study has fallen short of the initial goal of producing a full gravitational simulator for the Cell processor, using the method of direct summation, it has produced a starting point for future work on such a program, demonstrated the applicability of the Cell to the problem in the form of computational parallelism, and enumerated some of the problems involved with programming for the Cell processor.
References


