Parallel Hardware/Software Implementation of the N-body Problem on a Supercomputer

CS 213 Report

Guangdeng Liao and Scott Siowry

Abstract

The marriage of a highly parallel supercomputer with FPGA presents new opportunities for high performance computing. We implement and analyze an important scientific application called the N-body problem on an SGI Altix 4700 machine, a supercomputer which is coupled with two Xilinx Virtex 4 LX200 FPGAs. We analyze the tradeoff space for implementing the N-body problem as a highly parallel algorithm versus adding hardware support. We also consider combining a parallel implementation with hardware support. Our results show that a new computation workload assignment policy on software and hardware should be taken into consideration to fully take advantage of both CPU and FPGA computation power. We propose one hybrid approach to combine computational capabilities of the SGI CPU cores and the FPGAs together to calculate gravitational forces to improve the latency of the N-body simulation. Results show that our hybrid scheme can achieve 36X and 42X speedup respectively comparing to 8X and 13X with only a pure hardware offload.

1. Introduction

Computing technology has increased exponentially in complexity and power over the last 50 years. In accordance with Moore's law, the size of a computer chip shrinks by roughly half about every eighteen months, meaning we can theoretically pack double the amount of power and performance in the same size chip. Modern desktops now have more raw power than the powerful supercomputers of the 1970s and 1980s.

However, supercomputing hasn't taken a backseat to the more prevalent desktop and laptop market. Supercomputers are still needed to run very complex and computationally expensive applications ranging from stock market simulations to protein folding.

Consequently, much research has been performed to be able to deal with the increased complexity in the software being run. Modern supercomputers can support thousands of processors, using a number of different interconnection schemes, bus structures, memory hierarchies, and topologies. There are limits to the amount of performance that can be achieved by simply adding more processors to the computer running the problem.

Taking cues from the hardware/software partitioning community, modern supercomputers are starting to integrate hardware support in the form of field programmable gate arrays (FPGA) to allow offloading of computationally intensive kernels and functions. Extensive research has shown that partitioning critical software functions and loops to hardware can result in substantial performance and energy benefits[3,5,6,8,9,10]. The SGI Altix 4700 [20], a 64 processor supercomputer, includes two Xilinx Virtex 4 LX200 FPGAs that are directly connected to the memory substructure of the main system. The integration of a large number of processors and FPGA on a single platform presents unprecedented and untapped opportunities for high performance computing.

There exist many problems domains which would benefit greatly from both a highly parallel computing cluster coupled with sufficient FPGA resources. In this paper, we investigate the performance benefits from implementing the N-body problem, a technique that is widely used to study formation and evolution of particles in galaxies, molecular biology, cellular automata, etc. We analyze the feasibility of combining parallel implementation of the N-body problem with hardware support. We show that to fully take advantage of the computational power on a highly parallel supercomputer, a new computation workload assignment policy on software and hardware should be taken into consideration. We will propose one hybrid approach to accelerate the gravitation force calculation (the heart of the N-body computation), which combines computational power of both the SGI's CPU cluster and the FPGA, instead of leaving CPU cluster idle to waste their cycles. Results from our conducted experiments show that our hybrid scheme can achieve 36X and 42X speedup respectively comparing to 8X and 13X with pure hardware offload. Our results also show that to accelerate parallel scientific applications more efficiently in future, either more FPGAs or larger FPGAs directly connected to the memory system might be required. The main contributions of our paper mainly consist of:

- Implement and analyze the scalability of N-body problem on SGI Altix 4700, which is a cache coherent non-uniform memory access (CC-NUMA) machine.
- Identify the highly computational function (gravitation force calculation) for hardware implementation and implement using Virtex 4 FPGAs, associated with the SGI machine.
- Design a hybrid algorithm for the N-body problem and analyze the scalability for fully utilizing both CPU and FPGA cores.

The rest of the paper is organized as follows: Section 2 presents a brief overview of the N-body problem. Section 3 discusses related work in the domain of the N-body problem, as well as hardware/software partitioning. Section 4 presents an overview of the SGI Altix 4700 supercomputer. Section 5 presents techniques for accelerating the N-body problem with hardware. Section 6 discusses experiments and results of N-body on the SGI Altix 4700 supercomputer. Section 7 presents a hybrid design to fully take advantage of CPU and FPGA computation power. Section 8 concludes.

2. N-body Problem Description

N-body simulation is one of the most widely used techniques to investigate formation and evolution of particles in various fields of science, including physics, astronomy, chemistry, and molecular biology [12][13]. A wide range of physical systems have been studied by modeling those systems with the N-body problem.

The N-body problem simulates the evolution of a system of *N* bodies, where the force exerted on each body arises due to its interaction with all the other bodies in the system. The simulation proceeds over time steps, each time computing the net force on every body and thereby updating its position and other attributes. Currently there are many computation methods and implementations to simulate the N-body problem. The straightforward approach is to directly compute pair-wise forces between bodies in each time step, described in the Figure 1. In each step, the net force on each body is calculated through combining all Newton gravitational forces between that body and all the other bodies in the system. The system is then updated by moving each body to it's new position.

Figure 1 Pseudocode for the Naive algorithm of N-body

```
For each timestep
   For each body i
      For all other bodies j
      Calculate Pairwise force F(I,j)
      F(i, j) = G * Mi * Mj * ((Xi - Xj) / r^3, (Yi - Yj) / r^3, (Zi - Zj) / r^3)
      Move body for each step
   End
End
```

The time complexity of naïve approach is $O(n^2)$. Unfortunately, most problems modeled as the N-body problem usually consist of a large volume of particles. Consequently, the naïve algorithm incurs a heavy performance overhead and will not suffice for any large model, which accounts for most of the astronomical, biological, and physical problems. Fortunately, hierarchical tree-based algorithms [14][15][16] reduce the complexity from $O(n^2)$ to $O(nlogn)$. In the following section, we introduce and summarize one of the more popular one of hierarchical tree-based schemes, the Barnes Hut algorithm.

3. Related Work

In order to speed up the particles simulation, hierarchical tree-based algorithms [14,15,16,21] and their parallel implementations have been developed to reduce the calculation cost, including Barnes-Hut algorithm, Greengard's Fast Multipole algorithm and the Multipole Tree algorithm. Furthermore, the emergence of FPGA technology has spawned efforts to integrate hardware support into the different

N-body implementations. In this section, we will present an overview of one of the more popular hierarchical tree-based algorithms, Barnes Hut. We will also summarize recent work being done in the field of hardware/software partitioning.

3.1 Barnes Hut

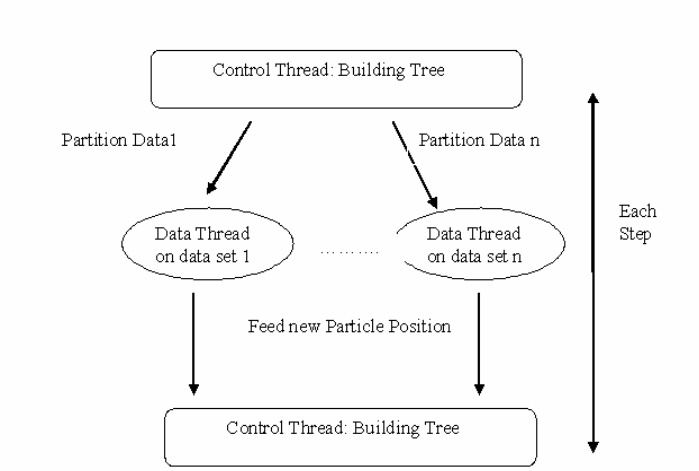
Barnes Hut algorithm [14], one of hierarchical tree-based algorithms, reduces the calculation cost from $O(n^2)$ to O(nlogn). Barnes-Hut works by organizing particles in the form of a tree, where each node of the tree represents a group of particles. The force from a distance node is replaced by the force from its center of mass.

Barnes-Hut uses the divide-and-conquer method to find clusters of particles in the N-body problem. If we consider the particles within an N-body simulation being contained within a 3 dimensional cube, we can imagine constructing an *octtree* by recursively subdividing the cube into eight smaller cubes. If a cube does not contain a particle then that cube is discarded. A cube that contains exactly one particle becomes a leaf cube. All other cubes are further subdivided. This subdivision process results in an incomplete tree, where each node in the tree has up to 8 children. Leaf nodes in the tree are cubes that contain exactly one particle. After building the octtree, the Barnes-Hut algorithm computes the center of mass and total mass for all the particles it contains for each sub-cube in the octtree, through a post order traversal. For each particle, the algorithm traverses the tree to compute the force on it for each particle. We note that the the Barnes Hut algorithm saves time because it only needs to compute the force due to all the particles in the box just by using the mass and center of mass of the particles in the box if the ratio of diameter of box and distance from particle to center of mass of box is below a defined threshold. Otherwise, particle-to-particle force using Newton gravitation law should be computed.

3.2 Parallel Barnes Hut

Since most of scientific applications operate on large data sets, they are often rewritten as parallel applications running on large CPU clusters with parallel computation capability. Singh et. al. [17] parallelizes the Barnes-Hut algorithm, attempting to parallelize the execution of each of the phases within a time step, and not exploiting the parallelism across time-steps. Although the force calculation for a particle requires the communication of position and mass information from other particles, the information is not modified during the force calculation phase. Therefore, task of force computation can be parallelized without extra communication in each time step [18]. The parallel implementation of Barnes-Hut is illustrated in Figure 2.

Figure 2 Parallel Barnes Hut



The parallel implementation employs two types of threads: control and data. The control threads are responsible for global tree-building and data set partition, and the data threads are responsible for the net force computation on the data assigned to it by the control thread. The control thread is bound to one physical node and is initialized to build up the octtree according to the current particles' position. The control thread then partitions the data set and assigns sub-sets to each independent data thread. In each time step, the data threads operate concurrently to compute net forces on its particles. Finally, the control thread updates the octree after all the data threads have finished computation on their own data set, and the process is repeated for the next time step.

3.3 Hardware/Software Co-design

Extensive research has been published regarding taking critical software loops and converting them to hardware representations, in a process known as hardware/software partitioning. Such partitioning has been shown to achieve substantial performance and energy benefits, often orders of magnitude better than the original software application [2,3,5,6,7,8,9,10].

Hardware/software partitioning typically takes two forms in the literature, one concentrating on multi-processing oriented software, and one concentrating on sequential software. In the multi-processing oriented form, often labeled system synthesis, maps a task graph to a set of concurrently executing (and communicating) microprocessors and coprocessors. The sequential form creates custom circuits to execute commonly executed functions (or loops) found in a single sequential program running on one microprocessor[10].

We focus on the sequential form of hardware/software partitioning. In this form, the custom circuits can be thought of as accelerators, which can dramatically improve the performance of the software application. In the sequential form of partitioning, the microprocessor will call the accelerator when it reaches the function call within the software application. The microprocessor will block remaining execution until the accelerator generates a result.

The close proximity of FPGA fabric on state-of-the-art platforms [3] has made hardware/software partitioning even more important. The FPGA can be placed on the FPGA bus to communicate with the host microprocessor, and current architectures (like the SGI Altix) give the onboard FPGA access to the entire memory space through shared memory. The marriage between CPU and FPGA presents a new opportunity for extraordinary performance gains by partitioning critical software functions and kernels onto the FPGA.

While much research has focused on developing tools to automatically partition an application for the best performance and energy consumption, many others are focusing on other aspects, including developing novel hardware implementations of software algorithms that couldn't be automatically synthesized or that resulted in inferior hardware.

4. SGI Altix 4700 Architecture Overview

The SGI Altix 4700 series is a family of multiprocessor distributed shared memory (DSM) computer systems that initially scale from 16 to 512 Intel 64-bit processors as a cache-coherent single system image (SSI). The system uses a global-address-space, cache-coherent multiprocessor that scales up to sixty four Intel 64-bit processors in a single rack. The system architecture for the Altix 4700 system is a fourth-generation NUMAflex DSM architecture known as NUMAlink 4. The SHub(Super-Hub)ASIC is the heart of the processor and memory node blade technology. This specialized ASIC acts as a crossbar between the processors, local SDRAM memory, and the network interface. The SHub ASIC memory interface enables any processor in the system to access the memory of all processors in the system.

The SGI Altix 4700 also adopts SGI RASC™ (Reconfigurable Application Specific Computing) technology and is coupled with the SGI RASC RC100 blade[19]. As illustrated in Figure 3, the RASC hardware blade contains two high performance Xilinx Virtex 4 LX200 FPGA chips with 160K logic cells, two TIO ASICs, and a loader FPGA for loading bitstreams onto the FPGAs. The FPGAs connect directly into the NUMAlink fabric via RASC Scalable System Port called SSP on the TIO ASICS and gives the FPGA access to the entire memory space with a bandwidth 3.2GB/s. The new RASC blade with two FPGAs also increases memory resources with 10 synchronous static RAM dual in-line memory modules. Each of SRAM is 16M bytes. The memory modules also provide a high-speed bandwidth of 6.4GB/s to each FPGAs.

Figure 3 RASC Blade Hardware

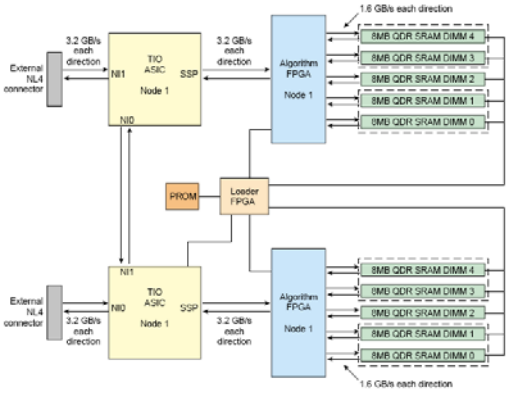


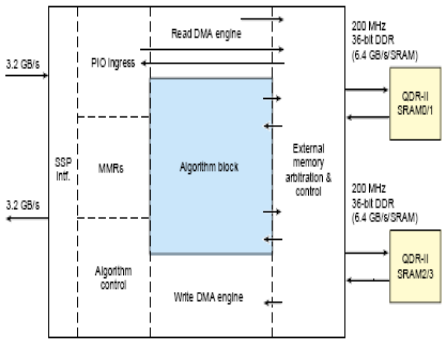
Figure 4 shows a block diagram of RASC Scalable System Port (SSP) Field Programmable Gate Array (FPGA). The FPGA contains two major functional blocks: The reprogrammable *Algorithm Block* and the *Core Services Block* that facilitates running the algorithm. The *Algorithm Block* sees memory resources, each with independent read and write ports: up to 1M words deep and 128-bits wide per bank (up to 16MBtotal per port per bank). The *Core Service Block* offers the following services to facilitate the FPGA algorithm block: independent direct memory access (DMA) engines for read and write data, host and FPGA process synchronization capabilities, including interrupts and atomic memory operations (AMOs), and two sets of independent read and write ports to each of the two random access memories (SRAMs) etc.

5. Implementation of N-Body Acceleration

Because of the utility of the N-body problem across different domains, much research has been developed to increase its performance. As earlier sections described, the research community developed the Barnes Hut algorithm which solves the problem in O(*nlogn*), trading off accuracy for greater performance. Similarly, researchers have devised methods to parallelize the Barnes Hut algorithm, so it might be mapped onto a highly parallel architecture like the SGI Altix.

Perhaps the best opportunity for N-body speedup is with the force calculation itself. The complex gravity force equation is computed for every single pair of bodies in the simulation, which accounts for a large percentage of the execution time. Since the gravity force equation includes several multiplications, a division, and a square root, a typical software implementation of the equation might consist of hundreds, if not thousands of cycles on a typical microprocessor architecture.

Figure 4 Block Diagram of the RASC algorithm FPGA



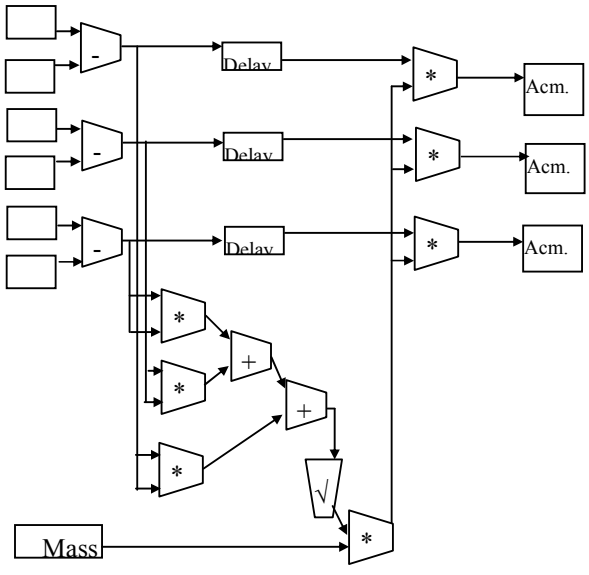
The notion of offloading the gravity equation to hardware on an FPGA thus becomes very attractive. The host CPU would feed the hardware gravity equation data in the form of coordinates in the x, y, and z planes, and the hardware would handle calculating the forces for the data points. A well-designed circuit could be implemented as a long "gravity pipeline" such that the circuit is able to sustain one force calculation/clock cycle. This is possible since each force calculation is completely independent of all other force calculation within the system. Such throughput would result in a performance gain of up to three orders of magnitude.

Figure 5 shows the gravity pipeline we designed to implement the force gravity equation: $F(i,j)=G*M*M_j*((X_i-X_j)/r^3,(Y_i-Y_j)/r^3,(Z_i-Z_j)/r^3)$. Although similar design has been introduced in the [1], it only considers particles with unit mass and simply ignored mass parameter. Out component consists of seven different memory structures where the mass and x, y, and z coordinates are maintained for each body in the simulation. On each clock cycle, the memories output values that are then fed through a tree of multipliers. Each multiplier is in fully pipelined, meaning delays need to be inserted to ensure the correct values of the subtraction are matched to the result of the tree of multipliers and adders. The square root component is also fully pipelined, and accounts for thirty-four stages of the entire sixty-seven stage pipeline. We omit the control logic required to initialize the particle and mass memories for brevity and clarity. The control logic would be responsible for communicating with a host processor in determining where the required data is to complete the force calculation.

6. Experiments and Results

In this section, we implemented two algorithms of N-body problem on SGI Altix 4700 supercomputer: the naïve N-body algorithm with quadratic time complexity and Barnes-Hut algorithm with time complexity of O(*nlogn*). We also implemented the hardware gravity pipeline described in Section 5 using Xilinx ISE 9.1 design suite, and targeted it towards the FPGAs available on the SGI machine. We integrated the gravity pipelined into the two algorithms and ran both on SGI machine. We present results for running the naïve algorithm (for purposes of comparison) and the Barnes-Hut algorithm on the SGI Altix 4700 machine both with and without hardware support. We note that although the SGI machine is furnished with sixty-four powerful Itanium CPUs, our experiment environment consisted of only thirty-two configured CPUs to minimize the impact from other applications running on SGI machine.

Figure 5 N-body "Gravity Pipeline"



6.1 N-body Naïve Algorithm Results

We first ran the naïve implementations of both the sequential and parallel versions of the N-body problem. Recall that the naïve version calculates pair-wise interactions between every single body in

the system. We ran the sequential naïve algorithm for purposes of comparison only. The sequential implementation on SGI machine took over 35 hours to complete computing gravitation forces of 524288 bodies for just one time step because the naive algorithm has a quadratic time complexity.

Figure 7 Parallel Naive Algorithm Speedup.

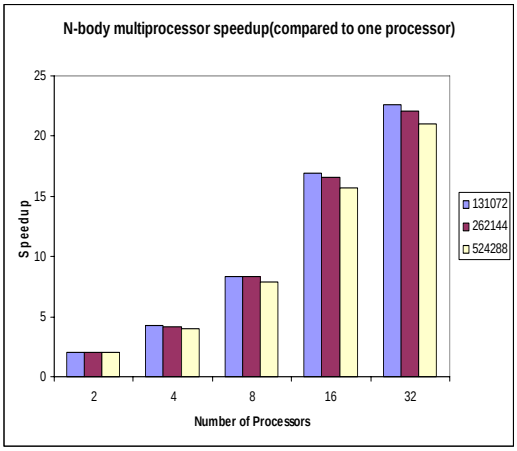


Figure 7 shows speedups attained when we implemented a parallel version of the N-body problem varying the number of threads. Even for the smallest test case, the parallel naïve algorithm was able to achieve almost a 23X speedup over the single-threaded sequential version implementation.

The onboard FPGAs presented even more opportunity for speedup over the naïve single threaded implementation. We implemented and integrated the N-body gravity pipeline into the single threaded and multithreaded versions using either one or both of the onboard SGI FPGAs. The naïve algorithm coupled with a single board of FPGA support attained 7.5X speedup over the pure software algorithm. When both FPGAs were configured with the gravity pipeline, the algorithm ran 12X faster.

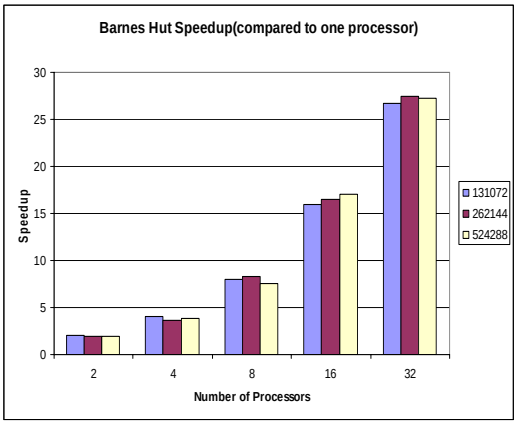
The multithreaded naïve algorithm implemented on the SGI Altix 4700 without hardware support and with hardware support were able to attain 23X and 13X speedups respectively, which means that computing power from CPUs on SGI machine is as roughly twice as powerful as completely offloading all computation to the FPGA. Thus, simply offloading all the entire workload to the FPGA heavily underutilizes the rest of the SGI's highly parallel resources.

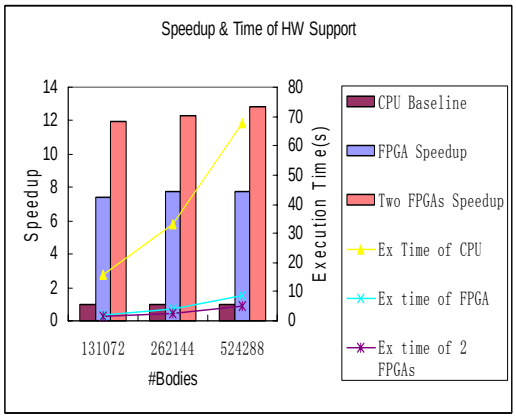
6.2 Barnes Hut Algorithm Results

Although the naïve algorithm with hardware support attained more than 13x speedups, the computation still took more than one hour real time to compute gravity force at each time step. To speed up the calculation, we also implemented the Barnes-Hut algorithm on the SGI machine, one of the hierarchical tree-based algorithms with O(*nlogn*) time complexity.

Figure 8 Barnes Hut Speedups

Figure 9 Speedup & Time of HW support

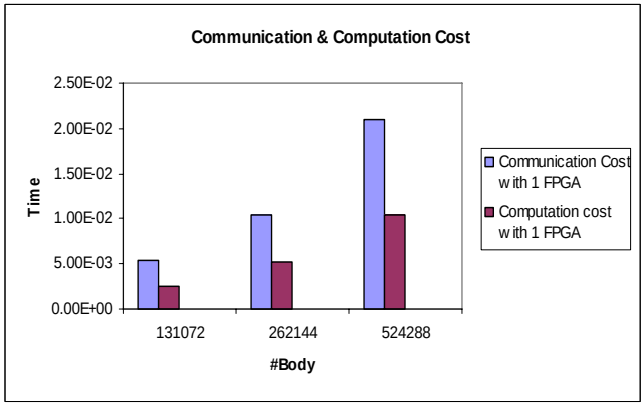




We also implemented a parallel implementation of Barnes-Hut, and then augmented both versions to work with the support of the FPGA gravity pipeline. Figure 8 illustrates the preliminary experiment results.

While not shown, the single threaded version of the Barnes Hut algorithm was superior to the naïve implementation of the N-body problem, as expected. For instance, Barnes Hut algorithm took 138 seconds to compute gravitational forces of 524288 bodies at each time step while the naïve algorithm took 35 hours. As illustrated in the Figure 9, the integration of one gravity pipeline into the Barnes-Hut algorithm yielded 12X performance speedups comparing to 27X speedup of parallel Barnes-Hut implementation, which indicates that FPGA is also an attractive approach to accelerate Barnes Hut algorithm on a highly parallel supercomputer. However, as mentioned in the section 6.1, a offloading the entire workload to FPGA might sacrifice massive CPU computing power on SGI machine.

Figure 10 Time Cost Analysis



In order to gain some insight into the makeup of the total execution time with FPGA support, we measured the data communication cost from the main memory system to FPGA and the force computation cost on the FPGA. Figure 10 illustrates that the time delay transferring data between FPGA and main memory is comparable to FPGA computation time and almost takes up to 70% of total execution time due to state-of-the-art data transfer bandwidth limit between memory and FPGA. Therefore although our system does not suffer from the costs of data transfer overhead heavily, special care of data transferring cost must be taken to design a more effective parallel

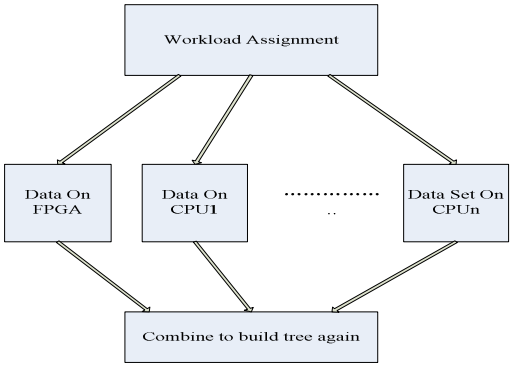
algorithm on the SGI machine. In addition, since execution time on the FPGA is only relevant to the number of bodies, the parallel algorithm with hardware support would not gain much more speedup than a sequential version when all gravitation force calculation workload is offloaded to hardware. Thus, massive CPU computing power on highly parallel supercomputer would be wasted significantly.

We have already shown that both a pure software approach and pure hardware offload approach do not fully utilize all of the computing resources to accelerate N-body problem aggressively. To further accelerate parallel scientific applications, we believe that both a combination of more FPGAs directly connected to memory systems might be require, as well as a new workload assignment policy to fully utilize the thirty-two CPUs and the 2 FPGAs on the SGI machine. In the next section, we propose a hybrid computation of the two algorithms to achieve even greater performance and utilization on the SGI supercomputer.

7. Hybrid Computation of CPU and FPGA

We have shown that while the inclusion of FPGA support presents significant performance speedup over pure software implementations, offloading the entire computational workload to shared FPGA not only makes FPGAs a potential bottleneck, but also leaves the all the CPUs underutilized, especially when pure parallelized software speedup is comparable to hardware acceleration, a complex system like the SGI would be heavily underutilized. In order to fully take advantage of the massive computing power of CPUs and FPGAs, we propose a new hardware/software co-design for highly parallelizable scientific applications which balances the workload between processors and FPGAs.

Figure 11 Implementation of hybrid design



Given the speedup X from hardware support, n processors and m FPGAs on a machine, we deduce a workload assignment policy for the machine, which is an approximate approach to optimal assignment for a highly parallel scientific application and can be used to guide us to design a more efficient highly parallel algorithm. Suppose D_c with C percent of workload is assigned to CPU and others D_f are assigned to FPGA. And also assume that their computing time per unit is t_c

and t_f when workload is assigned to CPU and FPGA respectively, and the data transfer latency per byte from memory to SRAM attached to FPGA is represented by l.

We denote the execution time on CPU and FPGA as T_c and T_f. Additionally, we denote the speedup from n processors and m FPGAs as $Sc(n)$ and $Sf(m)$. Therefore we can formulate them as followings:

$$T_c = D_c t_c / Sc(n) \qquad T_f = D_f t_f / Sf(m) + lD_f$$

In order to fully utilize both the CPU and FPGA computing power, we attempt to achieve the following object function:

$$Min\{Max\{D_c t_c / Sc(n), D_f t_f / Sf(m) + lD_f\}\}$$

In order to achieve the above object function, T_c should be equal to T_f, which means it balances the load between processors and FPGA. Through the above equation, it is easy to deduce the following workload assignment policy.

$C = Sc(n)(t_f + Sf(m)l)/(Sc(n)(t_f + Sf(m)l) + Sf(m)t_c)$ This means C percent of computational workload is still in CPU, and other (*1-C*) percent would be offloaded to FPGA. Although not all systems suffer from a heavy data transfer overhead, we can't simply ignore memory-to-FPGA latency in the N-body problem in our case, since memory-to-FPGA latency in N-body problem costs more than half of total hardware execution time. In order to assign the workload based on the equation described above, speedup of hardware and multi-processors, execution time of CPU and FPGA and data transfer cost are required to partition the data set. We varied the parameters to match the specifications of the SGI Altix 4700 machine and illustrated them using an N-body simulation of 131072 bodies scenario in the Table 1. Table 2 shows the results of partitioning the workload based on the equation above and using the particular parameters of the SGI machine, shown in Table 1.

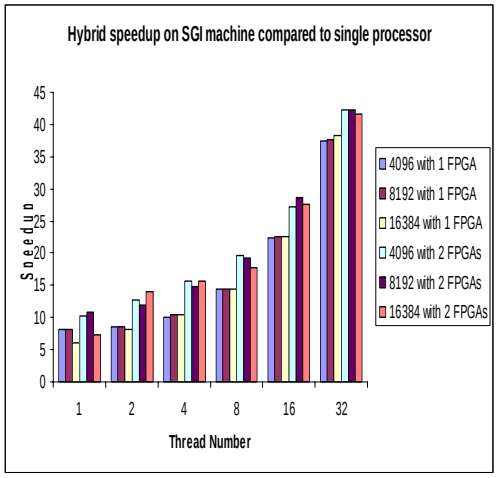
Table 1 Parameters on SGI machine

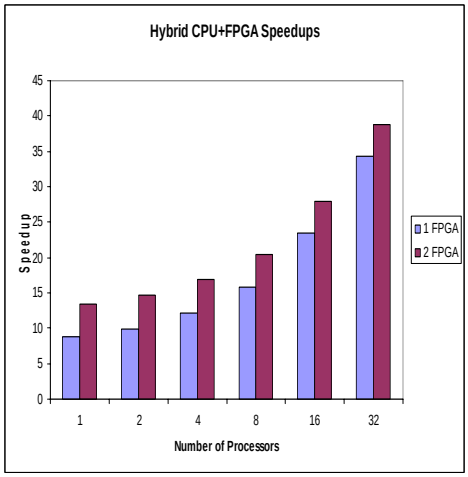
Parameters	l	t_c	t_f
Time(us)	0.0004	5.96E+04	8.0E+3

Table 2 Workload Assignment on SGI machine

#CPU	CPU Wordload with 1 FPGA	CPU Workload with 2FPGA
1	11.49%	7.49%
2	20.13%	13.57%
4	33.07%	23.55%
8	50.71%	39.05%
16	68.17%	57.18%
32	77.8%	68.7%

Figure 12 Hybrid design on Naive algorithm **Figure 13** Hybrid Design on Barnes Hut algorithm





It is clear that the partition is highly dependent on the number of CPUs and FPGAs so as to produce minimum execution time. Based on the proposed hybrid design and data set partition, we modified our implementation described in the previous section and ran experiments configured with one FPGA and two FPGAs. In order to eliminate the contention cost of share FPGAs on SGI machine, we implemented the hybrid design where only one thread was assigned to access and manage gravitation force computation on FPGAs, and all the others ran their computation work on their corresponding microprocessor. The results are shown in Figure 12. Since our hardware acceleration design on one FPGA was able to attain around 8X speedup, the workload assignment policy guides the naïve algorithm to leave 73% of computational workload on 32 processors configured with one FPGA and 66% for two FPGAs. From Figure 12 the new hybrid design with a FPGA and two FPGAs was able to attain upward of 36X and 42X speedup over the sequential versions. Note this is a dramatic speedup compared to our original implementation where we offloaded all the computation to the FPGA gravity pipeline. This is most likely because our hybrid design guided gravitation force computation to approximately fully-utilize all of the computing resources on SGI machine rather than leave all CPUs idle while the FPGA operated on the data set.

We also integrated our workload assignment policy and hybrid design into our Barnes-Hut algorithm implementation. The workload assignment policy left 78% and 70% of computing workload on all 32 processors when one FPGA and two FPGAs were configured respectively. Figure 13 illustrates that the new hybrid design was able to attain 40X and 45X speedup utilizing one and two FPGAs over the sequential implementation. Again this is a dramatic increase in speedup over the initial Barnes-Hut implementations which only attained 8X and 13X performance speedups with a pure hardware offload of the gravity computation. Obviously, a hybrid approach can offer an implementation that will come closer to fully utilizing all of the resources on a highly parallel supercomputer such as SGI Altix 4700, and thus is a more attractive approach to accelerate high performance computing.

8. Conclusions and Future Work

We have presented results that show we can attain speedups over 40X when implementing the N-body problem on a highly parallel SGI Altix 4700 machine. We implemented a gravity pipeline on the onboard FPGAs capable of sustaining one calculation per cycle that resulted in over 10X speedup over pure software sequential and parallel implementations of the N-body problem. However, we observed that many of the resources on the SGI machine were being underutilized with a pure offload of the gravity calculations. Therefore, we suggested that a new computation workload assignment policy on software and hardware should be taken into consideration to fully take advantage of both CPU and FPGA massive computation power. In order to further improve N-body implementation on SGI Altix 4700 machine coupled only with two FPGAs, we proposed one hybrid approach to combine computation capabilities of FPGA and CPU together to calculate gravitation force to take into account our new workload policy. The marriage of a large CPU cluster with FPGAs presents a large opportunity for applications requiring very high performance, but we've shown that in order to gain the best performance, both resources must be fully utilized.

9. References

[1] Tsoi, K.H.; Ho, C.H.; Yeung, H.C.; Leong, P.H.W. An arithmetic library and its application to the N-body problem Field-Programmable Custom Computing Machines, 2004. FCCM 2004

[2] Chattopadhyay, A. and Z. Zilic. GALDS: A Complete Framework for Designing Multiclock ASICs and SoCs. IEEE Trans. on Very Large Scale Integration (VLSI) Systems, Vol. 13, No. 6, June 2005

[3] [3] Eles, P., Z. Peng, K. Kuchcinsky, and A. Doboli. System Level Hardware/Software Partitioning Based on Simulated Annealing and Tabu Search. Design Automation for Embedded Systems, vol2, no 1, 5-32 January 1997.

[4] Excalibur. Altera Corp., http://www.altera.com

[5] Galanis, M.D, A. Milidonis, G. Theodoridis, D. Soudris,and C. E. Goutis. A Partitioning Methodology for Accelerating Applications in Hybrid Reconfigurable Platforms. Design Automation and Test in Europe (DATE), pp. 247-252, 2005.

[6] Gupta, R. and G. De Micheli. Hardware-Software Cosynthesis For Digital Systems. IEEE Design and Test of Computers. Pages 29-41, September 1993

[7] Hauser, J.R. and J. Wawrzynek. Garp: A MIPS Processor with a Reconfigurable Accelerator. FPGAs for Custom Computing Machines, 1997. Proceedings., The 5th Annual IEEE Symposium on16-18 April 1997 Page(s):12 – 21

[8] Henkel, J. A low power hardware/software partitioning approach for core-based embedded systems. In Proceedings of the 36th ACM/IEEE Design Automation Conference, 122–127.1999

[9] Kalavade, A. and Subrahmanyam, P. A. 1997. Hardware/software partitioning for multi-function systems. In Proceedings of the 1997 IEEE/ACM international Conference on Computer-Aided Design

[10] Stitt, G., F. Vahid, and S. Nematbakshi. Energy Savings and Speedups From Partitioning Critical Software Loops to Hardware in Embedded Systems. IEEE Transactions on Embedded Computer Systems, January 2004.

[11] Suresh, D. C., W.A Najjar,, F. Vahid,., J. Villarreal., and G. Stitt.. Profiling tools for hardware/software partitioning of embedded applications. In Proceedings of the 2003 ACM SIGPLAN Conference on Language, Compiler, and Tool For Embedded Systems (San Diego, California, USA, June 11 - 13, 2003). LCTES '03. ACM Press, New York, NY, 189-198. 2003.

[12] J. Makino and M. Taiji. Scientific simulation with specialpurpose computers - the GRAPE systems. pages 41–48. John Wiley & Sons Ltd, 1998.

[13] T. Narumi, R. Susukita, T. Ebisuzaki, G. McNiven, and B. Elmegreen. Molecular dynamics machine: Special purpose computer for molecular dynamics simulations. In Molecular Simulation, pages 401–415, 1999.

[14] J.E. Barnes and P. Hut. A hierarchical O(N Log N) force calculation algorithm. Nature, 324(4):446-449, December 1986

[15] J.A. Board, Z.S. Hakura, W.S. Elliot, D.C. Gray, W.J. Blanke, and J.F. Leathrum Jr. Scalable implementations of multipole-accelerated algorithms for molecular dynamics. Technical Report 94-002, Duke Univaersity, 1994

[16] L. Greengard. The rapid evaluation of potential fields in particle systems. The MIT Press, 1987.

[17] J.P. Singh, W.D. Weber, and A. Gupta. Splash: Stanford parallel applications for shared-memory. Computer Architecture News, 20(1):5-44, March 1987

[18] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In Proceedings of the 22nd International Symposium on Computer Architecture, pages 24-36, Santa Margherita Ligure, Italy, June 1995. © 1995 by the ACM

[19] SGI SGI® RASC™ RC100 Blade: http://www.sgi.com/products/rasc/

[20] SGI Altix 4700 servers and supercomputers : http://www.sgi.com/products/servers/altix/4000/

[21] J.A. Board, Z.S. Hakura, W.S. Elliot, D.C. Gray, W.J. Blanke, and J.F. Leathrum Jr. Scalable implementations of multipole-accelerated algorithms for molecular dynamics. Technical Report 94-002, Duke Univaersity, 1994