Accelerating AP3M-Based Computational Astrophysics Simulations with Reconfigurable Clusters

Tianqi Wang†, Tong Geng‡ Xi Jin∗ Martin Herbordt†∗
†Department of Physics; University of Science and Technology of China, Hefei, China
‡Department of Electrical and Computer Engineering; Boston University, Boston, MA, USA
tqwang@mail.ustc.edu.cn, tcheng@bu.edu, jinxi@ustc.edu.cn, herbordt@bu.edu

Abstract—In this paper, we present a case study of using a reconfigurable computing cluster to accelerate AP3M-based computational astrophysics simulations. AP3M is an adaptive particle-particle, particle-mesh method. Many computational astrophysics simulations are based on this method. AP3M can dynamically and adaptively apply computational resources non-uniformly to emphasize regions of interest. Therefore, AP3M can be faster and more energy-efficient than the traditional P3M (particle-particle, particle-mesh) approach. However, the dynamic and pointer-based data structure used by AP3M makes it extremely difficult to accelerate with FPGAs. In this work, we use a custom data structure and hardware kernel to overcome these challenges. All CPU-based dynamic and pointer-based tasks are mapped to FPGAs. Our experiments show that a single FPGA outperforms a Xeon E5-2660 CPU server (8 cores) by from 21x to 23x depending on problem size and data distribution.


I. INTRODUCTION

Over the last decades, great strides have been made in understanding the large-scale structure of the universe. This is largely because of high-resolution computational astrophysics simulations based on the ubiquitous attraction of the gravitational force. The structures in these simulations are highly non-uniform: Galaxies, for example, are typically $10^4$ times denser than their surrounding environment, and substructures within them can be orders of magnitude denser still [1].

Direct methods simply add up the contributions of all particles to the gravitational potential. This Particle-Particle (PP) method has good simulation resolution but suffers from high computation complexity of $O(N^2)$. The Particle Mesh (PM) method treats the force as a field by computing it on a mesh. This approach is implemented on a uniform grid and uses a spectral method, like the Fourier Transform, to solve the problem. Even though the resolution is lower than that of the PP method, PM reduces the complexity to $N_g \log(N_g)$, where $N_g$ is the number of grid nodes. For contemporary computational astrophysics, modeling systems with high-density contrast are difficult with either fixed grid PM or direct PP method. Thus hybrid solvers, such as Particle-Particle, Particle-Mesh (P3M) methods have been utilized extensively. P3M combines the use of PM for the surrounding environment with the use of PP for high-resolution regions. Crucially, substantial improvement in the performance of P3M algorithms can be achieved by allowing spatially adaptive mesh refinement (AMR) in regions of high particle density. The AP3M method was proposed by Couchman [2] and is widely used [3].

FPGA is a promising candidate for computational astrophysics-specific accelerators [4], [5]. For astrophysics simulations using FPGAs, A3PM rarely been used due to the dynamic and pointer-based data structures and the expensive and inefficient irregular off-chip memory access requests. To the best of our knowledge, there has been no previous work on accelerating A3PM algorithm using FPGAs. The contributions of this paper include:

- We use an FPGA-centric cluster to accelerate A3PM-based computational astrophysics simulations;
- The hardware resources are allocated according to the workloads of adaptive particle-mesh and particle-particle, so that the computation latencies of these two parts can be overlaid;
- Experiments show that a single FPGA outperforms a Xeon E5-2660 CPU by 20x to 23x depending on cluster scale and simulation initial conditions. Compared with an NVIDIA Tesla P100 GPU, this design provides comparable per-socket performance with much better power consumption. Also, we demonstrate experimentally that, using the secondary network, performance scales perfectly to eight FPGAs and is likely to scale similarly to much larger FPGA clusters.

II. BACKGROUND

A. AP3M

For computational astrophysics, the dynamics of all particles is described by the Poisson function. We assume periodic boundary condition for a cube of size $L^3$. P3M algorithms get high resolution by solving the long-range part in Fourier space with the Particle-Mesh method and the short-range part in real space with the Particle-Particle method.

The P3M algorithm has an operation cost that is approximately $O(\alpha N + \beta L^3 \log L + \gamma N^2_{pp})$ where $\alpha$, $\beta$ and $\gamma$ are constants. This operation cost is the summation of mass assignment, PM’s 3D Fourier transformation, and short-range force calculation. P3M suffers the drawback that, under heavy gravitational clustering, the short-range sum (used to supplement the PM force) slows the calculation dramatically.

As shown in Figure 1, AP3M remedies the slow-down under clustering by isolating regions where the $N^2_{pp}$ term dominates. It then solves for the short range force in these regions using FFT methods on a sub-grid, which is then supplemented by short range calculations involving fewer neighbors. This process is a repeat of the P3M algorithm on the selected regions. Therefore the operation count is now approximately shown in Equation 1.

$$O(\alpha N + \beta L^3 \log L + \sum_{j=1}^{n_{ref}} [\alpha_j N_j + \beta_j L^3_j \log L_j + \gamma_j N^2_{pp,j}])$$

(1)
B. FPGA-centric clusters

In traditional HPC clusters, the accelerators communicate through their hosts, which results in substantial latency overhead. For FPGA-centric clusters, such as Catapult I and many other recent systems, FPGAs are interconnected directly through their transceivers. As shown in Figure 2(A), the FPGAs can exchange data without detouring through the host CPU. The direct communication network not only shortens the physical distance but also reduces the overhead of the CPUs’ software stack.

C. Challenge and Motivation

To take advantage of FPGA-centric clusters’ strengths, we need to reduce CPU-based computation and communication. For AP3M, the optimized hardware kernels are designed to handle the PM and PP functions. However, the AP3M algorithm uses tree-based dynamic data structures. For computational astrophysics simulation, the size of these data structures is usually hundreds of MB. Therefore these pointer-based dynamic data structures must be stored off-chip. The large number of random off-chip memory accesses makes the accelerator inefficient, so existing systems use host CPUs to handle these adaptive particle mesh operations. Our previous work [6] provide an FPGA framework called FP-AMR. FP-AMR can offload all CPU-based pointer-based adaptive particle mesh (APM) operations to FPGAs, including particle mesh mapping, mesh refinement, and coarsening.

During the whole astrophysics simulation, the distribution of particle evolves. Thus, the workload allocation among nodes needs adjust at runtime. Moreover, the evolution of particles’ density changes the ratio of PM to PP operations. So the ratio of these hardware kernels should change dynamically. In a word, we need to design an efficient workload/resource allocation algorithm for AP3M simulation.

III. ACCELERATOR ARCHITECTURE

A. Overview

Figure 2(A) shows the architecture of the AP3M accelerator. A router module handles FPGA-to-FPGA communication through the transceivers. Intra-node, the FPGA and CPU exchange data via the PCIe bus. APM, PM, and PP calculations are instantiated as specific hardware kernels. The related data structures are too large (usually several GB) to be stored on-chip; therefore, each hardware kernel has a private on-chip cache.

B. Potential Calculation Kernel

The design of the PP kernel is shown in Figure 3(A). The i-th PP kernel receives the particle position $\text{Pos}_i$ and the particle information list $\text{Par}_j$, and evaluates the potential $\varphi$. For multiple PP kernels ($PP_1 \sim PP_n$), we fix their position sequence as $\text{Pos}_1 \sim \text{Pos}_n$ and the sorted particle information list is sequentially broadcast to these kernels’ particle input port. Because of the spatial locality of the sorted particle information list, we can improve the kernels memory access efficiency.

The PM method’s main operation is the 3D FFT. The $N^3$ points 3D FFT can be computed by executing three sets of $N^2$ N-point 1D FFTs consecutively in the three dimensions. As shown in Figure 3(B), the design has three main parts: RAMs, Crossbars, and FFT pipelines. The RAMs’ primary purpose is simply to store the data throughout the computation. The Crossbars work in conjunction with the RAMs to select the flow of data so as to effect transpose and un-transpose as needed.
need to allocate FPGA resource rationally to make sure that their time consumption in each timestep is almost the same. According to the hardware design described above, there are several constant parameters: 1) Performance of one APM kernel $Perf_{APM}$, 2) Performance of one PM kernel $Perf_{PM}$, and 3) Performance of one PP kernel $Perf_{PP}$. Therefore, we can estimate the execution time of each kernel type.

During the simulation, PM and PP workloads vary over time. As shown in Section III, the PM and PP kernels can be reconfigured at run time. Even that the reallocation can reduce resource idleness, reconfiguration is time-consuming. Therefore the host CPU needs to monitor the change in workload and weigh gains versus losses.

The cluster scheduling of $Node_m$ and $Node_n$ is shown in Figure 5. For timestep $i$, at $T_0$, the sorted particle information list and adaptive mesh have been generated by the APM kernel of $Node_m$. After that, the particle distribution status is sent to the local CPU and the PP/PM kernels start work at the same time. At $T_2$, the local CPU finishes the workload estimate and sends the information to the host node of the cluster for partitioning the whole cluster's workload. The PM and PP calculation finishes at $T_4$ and $T_6$. Motion update can only start after $T_6$. During $T_4 \sim T_6$, the PP kernels are idle. After motion update finishes at $T_7$, $Node_m$ and $Node_n$ exchange particle information via the direct network.

Using the monitored data of the previous timestep, at $T_8$ the host node CPU decides whether to adjust the FPGA resource allocation in the next timestep. Reconfiguration lasts from $T_9 \sim T_{10}$.

V. EXPERIMENTAL EVALUATION

We implement three different systems as controls: 1) CPU-only cluster, 2) CPU-GPU cluster (GPUs as co-processors), and 3) CPU-FPGA cluster (FPGAs as co-processors). These three versions are based on AMReX, which is a publicly available software framework designed for building massively parallel adaptive mesh applications.

A. Experimental Setup

We use NVIDIA Tesla P100 GPUs and Xilinx VCU118 Boards as platforms. These devices all use the 16nm process. The host CPU is a one-socket Intel Xeon E5-2660 with 8 cores running multithreaded. We currently have resources to test all four configurations for up to eight nodes with one accelerator per node. The control clusters all use CPUs to handle AMReX-based adaptive mesh operations and use CPU-side Ethernet for communication between nodes. For the Poisson solver, the CPU-only version uses the Intel AVX-enhanced FFTW library; the CPU-GPU version uses cuFFT; and the two FPGA versions both use Xilinx FFT IP. For the CPU-FPGA version’s accelerator, the ratio of PM/PP components is decided based on the simulation initial conditions. This ratio is static during the simulation.

In the FP-AP3M version, the CPU only handles program initialization and workload scheduling. All adaptive mesh and Poisson solver tasks are completed on the FPGA. All other communication is direct FPGA-to-FPGA through the FPGAs’ transceivers.

The performance of the cluster depends on the initial conditions of the system. For example, the particles’ kinetic energy can influence the number of refine/coarsen operations, the frequency of particle information exchanges, and workload/resource reallocate. We use NGenIC, a widely used initial conditions generator for cosmological simulations. We generate two different initial conditions: strong interaction and weak interaction. Both have the same number and distribution of particles, but the kinetic energy of the first is twice that of the second.

B. Performance

To measure performance we run the system for 500 timesteps after a 50 timestep warm-up. Figure 6 shows average node performance for different initial conditions and node count. Table I shows the systems’ performance. All of the accelerators do very well compared with the CPU-only system: CPU+GPU is $35 \times -37 \times$ better, CPU+FPGA $17 \times -20 \times$, and FP-AP3M $20 \times -23 \times$ better.

<table>
<thead>
<tr>
<th>TABLE I</th>
<th>PERFORMANCE OVERVIEW</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CPU-only</td>
</tr>
<tr>
<td>Performance</td>
<td>1x</td>
</tr>
<tr>
<td>Energy Consumption</td>
<td>1x</td>
</tr>
</tbody>
</table>

C. Contribution Breakdown

Compared with the CPU+FPGA system, FP-AP3M’s performance is 20%-23% better. This improvement originates from three effects. First, at the end of each timestep, nodes transfer particles as they move through boundaries. FP-AP3M benefits from the direct FPGA-to-FPGA connections. Second, FP-AP3M offloads adaptive mesh related tasks to the FPGAs. Thirdly, with the help of partial reconfiguration (PR), FP-AP3M can reallocate FPGA resources dynamically and reduce hardware idleness. If the initial condition has higher kinetic energy, it will cause more particle information exchange and influence the particles’ spatial locality, which has a negative effect on the particle map. As shown in Figure 6, for the strong interaction the initial time consumption of each timestep is 22.8% larger.

For the strong interaction case, the AP3M algorithm costs more time. Therefore, we use this initial condition to analyze these three effects. Figure 7 gives a break-down of time consumption for the 8-node versions of CPU-GPU, CPU-FPGA, FP-AP3M (PR), and FP-AP3M (No PR).

Contribution of APM offload: For the FP-AP3M(PR/No PR) system, FPGA-based adaptive mesh operation cost 50.7%-51.8% time compared with CPU-GPU/FPGA systems.

Contribution of the direct network: For the CPU-FPGA and CPU-GPU clusters, their inter-node particle information exchange operations are handled by the CPU. Hence the times for these operations are similar. The FP-AP3M(PR/No PR) system’s inter-node particle exchange operation costs 3.4% -4.8% more time compared with CPU-GPU/FPGA systems.

Contribution of PR: The CPU-FPGA cluster and the FP-AP3M(No PR) cluster have similar times for the Poisson solver. The GPU cluster’s Poisson solver is three times faster. The FP-AP3M(PR) version’s Poisson solver is 5.7% faster than the other two FPGA clusters.
D. Utilization Breakdown

Figure 8 shows the FPGA resource utilization. The FP-AP3M design contains three parts: basic utility (DRAM controller, Transceivers controller, etc), AMP kernel, and Poisson solver (PM/PP kernel). For FP-AP3M, the FPGA resource bottleneck is DSP slices. The hardware APM operations cost less than 5% of DSP slices.

Overall we find that both FPGA-based versions of APM substantially improve performance over CPU-based versions, with factors of $17 \times 20$ and $20 \times 23$, respectively. Energy consumption is improved by a much greater factor. The limiting factor on FPGA performance is the number of DSP units.

When compared with a high-end GPU of similar process generation, we find that the performance of the GPU is $1.46 \times 1.67 \times$ better than FP-AP3M. This appears to be entirely due to the floating point resources available on the GPUs for the execution of the intra-iteration payload computations. Clearly having more DSPs would allow FP-AP3M to have equal or better performance. The energy consumption of the FPGA-based systems is about $6 \times$ less than that of the GNU-based systems.

VI. DISCUSSION AND FUTURE WORK

In this work, we use an FPGA-centric cluster to accelerate AP3M-based computational astrophysics simulations. We believe that this is the first such study. We create two versions, one where the FP3Ms execute the computations that are traditionally offloaded to the accelerator, but the CPUs retain control and data structures; and a second, FP-AP3M, where the entire computation (after initialization) is executed on the FP3Ms. For the FP3Ms to execute the balance of the CPU-based tasks—including particle mesh mapping, mesh refinement and coarsening—requires using custom data structures. Another advantage of FP-AP3M is that inter-iteration particle exchanges are executed using direct FPGA-FPGA connections. The FP-AP3M enhancements lead to a 20% to 23% performance improvement over the CPU+FPGA version. Moreover, partial reconfiguration technology is used to help FP-AP3M dynamically reallocate resources among different hardware kernels. With this, the Poisson solver’s performance can improve 5.7×.

When compared with a high-end GPU of similar process generation, we find that the performance of the GPU is $1.46 \times 1.67 \times$ better than FP-AP3M. This appears to be entirely due to the floating point resources available on the GPUs for the execution of the intra-iteration payload computations. Clearly having more DSPs would allow FP-AP3M to have equal or better performance. The energy consumption of the FPGA-based systems is about $6 \times$ less than that of the GPU-based systems.

REFERENCES