Hardware acceleration of a Quantum Monte Carlo application

Akila Gothandaraman[1], G. Lee Warren[2], Gregory D. Peterson[1], Robert J. Harrison[3]
{akila, gwarren, gdp, robert.harrison}@utk.edu
[1] *Department of Electrical and Computer Engineering, University of Tennessee, Knoxville*
[2] *Department of Chemistry, University of Delaware*
[3]*Department of Chemistry, University of Tennessee, Knoxville*

Abstract

We are currently exploring the use of reconfigurable computing using Field Programmable Gate Arrays (FPGAs) to accelerate kernels of scientific applications. Here, we present a hardware architecture targeted towards the acceleration of two scientific kernels in a Quantum Monte Carlo (QMC) application applied to N-body systems. Quantum Monte Carlo methods enable us to determine the ground-state properties of atomic or molecular clusters. Here, we focus on two key kernels of the QMC application: acceleration of potential energy and wavefunction calculations. Our current platform consisting of a dual processor Intel Xeon 2.4 GHz augmented with two reconfigurable FPGA development boards provides a 3x speedup over the equivalent software only implementation. Targeting our design onto a High Performance Computing (HPC) system like the Cray XD1 or XT4 platform with high gate-density FPGAs will allow us to operate multiple instances of our design thereby providing additional parallelism.

1. Introduction

Scientific computing is characterized by applications that have an ever-increasing demand for tremendous processing power. Traditional High Performance Computing (HPC) systems have involved the use of supercomputers and cluster-based computing systems to boost the performance of computationally demanding scientific applications. Current HPC systems incorporate hardware accelerators onto their processing nodes to speed up the critical portions of these applications. Hardware accelerators using reconfigurable logic, such as Field Programmable Gate Arrays (FPGAs) are now forming major components of HPC systems to leverage the coarse-grained parallelism of the microprocessors with the fine-grained parallelism provided by FPGAs. These FPGA-based solutions, presently offered by vendors like Cray (*Cray-XD1* [1]), SRC Computers (*MAPstations* [2]), SGI systems (*SGI Reconfigurable Application Specific Computing – RASC* [3]) and DRC Computers (RPU – Reconfigurable Processor Unit [4]), which couple FPGAs with high-end conventional microprocessors, are of increasing interest to the computational science and engineering community. Previously, FPGAs have been used in applications where a function in its entirety is mapped onto reconfigurable hardware. The emerging hybrid HPC systems allow us to partition the applications such that critical components can be mapped onto hardware and the remainder of the application retained in software, thus providing significant performance gains over an entirely software implementation running on a general-purpose processor. FPGAs have been used in various computational science and engineering applications such as computational fluid dynamics [5] and molecular dynamics simulations [6].

In our work, we apply reconfigurable computing using FPGAs to accelerate a Quantum Monte Carlo (QMC) chemistry application. We present a parallel and pipelined architecture to calculate the properties of the particles in an *N*-body system. We employ unique schemes that will easily allow us to transform and interpolate a given function and compute it using reconfigurable hardware. Our design presently computes the energies and wavefunctions over pairs of homogeneous or heterogeneous particles using a general interpolation framework. We aim to develop a user-friendly design framework that can be used to obtain the properties of large many-body systems and thus serve as a useful tool for studying these systems. In our earlier work [7], we provided preliminary results with estimated speedups while accelerating the potential energy calculation alone using FPGAs. In this work, we provide the actual speedups of the potential energy and wavefunction kernels implemented on the FPGA versus the software implementation.

The reason why the reconfigurability feature of FPGAs is desirable in our application is because a change in the nature (chemical identity) of the particles (atoms) of the system requires the use of a different set of interpolation parameters. This will also provide us with the freedom to evaluate any function of the position co-ordinates of particles, such as external or confinement potentials or custom potential energy surfaces beyond the simple Lennard-Jones form. Hardware-software partitioning methods [8] are critical for mapping a scientific application, which typically consists of many compute-intensive kernels, onto these architectures. The kernels in our QMC application that rely on additions, subtractions, and multiplications represent ideal candidates for implementation on present generation FPGAs with abundant logic and embedded hardware multipliers. It also gives us the flexibility to retain in software, rest of the calculations or for computing new properties which will not fit on our current FPGA device. Our implementation uses fixed-point arithmetic for all calculations, which is faster than floating-point calculations and requires less FPGA resources. Our error analysis shows that a 52-bit fixed-point format can deliver an accuracy on the order of or better than a standard double-precision floating-point representation for our application.

Our current platform consists of a dual-processor Intel Xeon with two Amirix Systems AP130 FPGA development boards [9]. Each board consists of a XC2VP30 Virtex II Pro [10] and 64 MB DDR SDRAM and is devoted entirely to the computation of potential energy or the wavefunction. The communication between the processor and the FPGA occurs via a 66MHz PCI interface. The Amirix control program is used for controlling the FPGA board. The overall application demonstrates a 3x speedup (on the second generation V2P FPGAs) over the software only version with no compromise in accuracy. This shows that a substantial improvement in speedup performance can be expected on platforms like the Cray-XD1 consisting of processors and several high-end FPGAs over a low-latency interconnect.

The reminder of the paper is organized as follows. Section 2 presents a background of Monte Carlo (MC). In Section 3, we present the related work on the special purpose engines and hardware accelerators developed for scientific simulations. Section 4 provides a description of the potential and wavefunction kernels chosen for hardware acceleration. In Section 5, we describe the building blocks of our complete MC architecture. In Section 6, we provide the performance results of our current implementation. In Section 7, we offer conclusions and some directions for future research.

2. Monte Carlo Background

Current parallel computing systems have provided us the sheer computing power required to conduct simulations of large *N*-body systems. These simulations could be anything from simulations of quantum systems consisting of interacting atomic or subatomic particles to astrophysical *N*-body systems. The tremendous processing power available today with these sophisticated parallel systems have aided in identifying interesting properties and phenomena in otherwise intractable systems. Two simulation techniques widely used in physics and physical chemistry are the Molecular Dynamics (MD) and Monte Carlo (MC) methods. MC methods are used in chemistry to study the structural and energetic properties of clusters consisting of a group of atoms or molecules close enough to experience interatomic or intermolecular attraction [11]. MD simulations are of a deterministic nature and used to simulate the classical time evolution of a system given the initial positions and velocities of all particles in the system. These methods use Newton's laws of motion to generate the successive configurations for the *N*-body system. On the other hand, MC methods are stochastic and rely on high quality random number generators and a Markov process to generate the configurations. MC simulations are inherently parallel and thus lend themselves well to implementation on parallel computer architectures. There also exist hybrid techniques, which switch between MC and MD techniques for various parts of the simulation.

We are interested in studying the ground-state properties of quantum many-body problems and use Quantum Monte Carlo (QMC) methods for modeling our *N*-body system. These methods can compute accurately many-body quantum mechanical properties of atomic clusters. These methods sample the *N*-body quantum mechanical wavefunction for the purpose of computing observable properties, such as potential energy. The statistical uncertainty in the computed properties shrinks with the square root of the number of samples; thus accurate estimates of properties require a large number of samples. Two flavors of QMC methods are, Diffusion Monte Carlo (DMC) and Variational Monte Carlo (VMC). DMC is a technique for numerically solving the many-body Schrödinger equation and can compute properties of the exact ground state of a bosonic quantum system. VMC method employs a set of adjustable parameters to yield a trial wavefunction, $\psi_T(x)$ that, when optimized, best approximates the exact wavefunction. We employ the VMC method for our *N*-body simulation. Figure 1 highlights the various steps of this algorithm.

```
REPEAT (for N iterations)
  1. Select a reference configuration, $R(x, y, z)$ at random
  2. Obtain a new configuration, $R'$ by adding a small random displacement to one of the particles in the above configuration
  3. Compute the ground-state properties (energy, wave-function, etc.) of the particles in the current configuration, $R'$
  4. Accept or reject the present configuration using the ratio of the wave function values,
       $p = \left| \frac{\psi_T(R')}{\psi_T(R)} \right|^2$
     If $p \geq 1$, $R'$ is accepted. If $p < 1$, $R'$ is rejected
     and $R$ is retained.
UNTIL finished
```

Figure 1. VMC algorithm

Step 1 of the algorithm consists of choosing a reference configuration, *R* for the system of atoms using a Cartesian co-ordinate system. We add a small random displacement to one of the atoms to obtain a new configuration, *R'* in step 2. In step 3, we compute the properties of the particles in the configuration, *R'*. Necessary properties include the local energy and the value of the trial wavefunction. To ensure that configurations are asymptotically drawn from the square of the known trial wavefunction $\psi_T(x)$, we accept or reject this configuration (and associated properties) by determining the fraction, *p* in step 4. If the new position, *R'* is accepted, then we also retain the corresponding properties, otherwise we keep the properties corresponding to the original configuration, *R*. Steps 2-4 are repeated until asymptotic behavior is attained (typically thousands of iterations). Additional samples (typically millions) are then drawn to compute properties of interest. Due to the limited resources on our target FPGA, we implement Step 3 of the algorithm presently using reconfigurable computing. The remaining steps of the algorithm are implemented in software on the general-purpose processor. We use the software Scalable Parallel Random Number Generator (SPRNG) [12] to provide the random configurations to the reconfigurable hardware in our MC simulation.

3. Related Work

First, we summarize the work related to the application of special-purpose hardware in Monte Carlo and Molecular Dynamics simulations. A reconfigurable MD simulator is proposed in [6] where all of the MD simulation tasks are mapped onto FPGAs. Reconfigurable hardware is also proposed in [13] where a hardware-software approach is used to map specific tasks of the MD simulation onto FPGAs and the remaining tasks executed on general-purpose processors. Reconfigurable computing has been used to accelerate Monte Carlo simulations in various applications. [14] describes a random variable accelerator for Monte Carlo simulation in finance. A hardware design has been proposed for generating random numbers from arbitrary distributions and applied to a pi estimator, a Monte Carlo integrator and a stochastic simulator for chemical species [15]. In [16], computationally intensive portions of a Monte Carlo application that simulates radiative heat transfer in a 2-D chamber have been mapped on Virtex-II and Virtex-II Pro FPGAs. A reconfigurable Monte-Carlo clustering processor (MCCP) is proposed in [17] where a number of MC algorithms used in statistical physics are implemented using reconfigurable hardware.

Previous research in accelerating scientific applications has resulted in a number of special purpose computers namely GRAPE (Gravity Pipe) systems [18] used to accelerate gravitational *N*-body simulations and MD simulations. These special purpose engines are used for computationally demanding long-range force calculations. The Protein Explorer with MDGRAPE-3 chip [19] performs the force calculations in MD simulations with potential target applications including drug design, protein analysis, and material sciences. PRO-GRAPE [20] overcomes the inflexibility of ASICs and reduces the initial development cost by using FPGAs to realize the pipeline processor.

Different schemes have been developed to provide high-quality random numbers required in Monte Carlo simulations. Hardware-accelerated random number generators based on the Scalable Pseudo Random Number Generator (SPRNG) library have been developed in [21]. [22] describes a splitting approach for parallel random number generation for parallelizing Monte Carlo simulations.

4. Description of kernels

Our original software QMC application computes the ground-state properties of homogenous or heterogeneous atomic clusters. The two computationally intensive kernels, potential energy and wavefunction calculation are accelerated using reconfigurable hardware. The numerical behavior of these functions requires us to use unique transformation schemes and then evaluate them using hardware. We describe the techniques used to transform these functions below.

4.1. Potential energy calculation

Figure 2 shows the interatomic potential as a function of distance. The N-atom potential energy function V_{total} is approximated as a sum of $N(N-1)/2$ pair contributions given by equation 1.

$$V_{total} \approx \sum_{i<j}^{N} V(r_{ij}) \tag{1}$$

The pair-wise potential energy function, $V(r_{ij})$ is characterized by an exponentially repulsive region at small values of r_{ij} and an attractive region at intermediate values of r_{ij} that asymptotically reaches zero. We define the following parameters: ε, the depth of the well region and σ, the cut-off value of r_{ij} where the potential function equals zero. The general shape of the potential energy function that Figure 2 depicts is universally applicable in describing non-Coulombic atomic or molecular interactions. However, slightly different potential energy functions are required depending on the exact chemical identities of the interacting atoms. Two problems posed by the potential energy function necessitate the use of special techniques to transform the functions.

First, the potential energy is a function of the distance, r_{ij}. At first glance, this calls for the instantiation of an expensive square root core on the FPGA device. However, if we rewrite each pair-wise potential $V(r_{ij})$ as a function of r_{ij}^2, we can eliminate the need for the square-root operation following the calculation of the squared distance values. This is largely a cosmetic distinction as all we have done is shift the burden of taking the square root of r_{ij}^2 inside the potential function. This is advantageous in conjunction with a spline-based evaluation method since we can effectively pre-compute the square root by building it into the supplied coefficients.

Second, we can observe the problematic range and domain of this functional shape. The potential has a finite domain and infinite range until it reaches a zero value, at the point $r_{ij}^2 = \sigma^2$ and an infinite domain and a finite range thereafter. We divide these two regions with non-identical numerical behavior as regions I and II. The original expression of equation 1 for the total potential is now rewritten as a sum of two terms given by equation 2. Region I is defined on the domain, $0 \le r_{ij}^2 < \sigma^2$ and within region I, $V_I(r_{ij}^2)$ is positive taking on values from zero to positive infinity. This dynamic range is clearly undesirable as we implement all

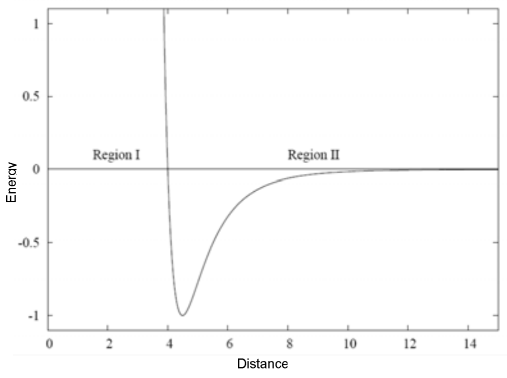

Figure 2. Plot of potential energy

the operations using fixed-point due to space limitations on our current FPGA device. Region II of the potential is defined on the region, $r_{ij}^2 \ge \sigma^2$ and ranges in value from zero to $-\varepsilon$. The finite range of the function in region II is an attractive property as it bounds the sum in equation 2 albeit the infinite domain remains a problem.

$$V_{total} = V_I + V_{II} = \sum_{(i<j)\in I} V_I(r_{ij}) + \sum_{(i<j)\in II} V_{II}(r_{ij}) \tag{2}$$

The exponential transform used in region I is given by equation 4. This transformation provides several advantageous properties. Firstly, the transformed region I potential, $V_I^{'}$ is restricted to take values between zero and one, inclusive. The sum of the pair-wise potentials for region I (the first term in equation 2) can now be expressed as a product of the transformed pair-wise potentials (equation 4). The relationship between V_I and $V_I^{'}$ is now given by equation 5. Another key advantage here is that the expensive final transformation involving the natural logarithm can now be delegated to the host processor.

$$V_I^{'}(r_{ij}^2) = e^{-V_I(r_{ij}^2)} \tag{3}$$

$$V_I^{'} = \prod_{(i<j)\in I} V_I^{'}(r_{ij}) \tag{4}$$

$$-\ln V_I^{'} = V_I = \sum_{(i<j)\in I} V_I(r_{ij}) \tag{5}$$

An approximate logarithmic binning scheme is used to cope with infinite domain in region II. We first introduce a cutoff at large distance that coincides with the largest value of r_{ij}^2 allowed by its fixed-precision format. Next, we partition the whole region into smaller regions such that the end points of each region correspond to consecutive powers of two. Thus, the size of each sub-region will increase stepwise with r_{ij}^2. This partitioning takes advantage of the fact that the curvature of the potential is largest at smaller values of r_{ij}^2 and asymptotically approaches zero or flattens out at large values of r_{ij}^2. Finally, we partition each sub-region into several intervals of equal size. This scheme allows us to effectively take advantage of a logarithmic transformation of the coordinate r_{ij}^2 without the need to compute base two logarithms to determine the correct set of coefficients for interpolation. We also re-scale the region II potential by a factor of $-1/\varepsilon$ so that it always takes a value between zero and one.

It is also worthwhile to point out here the many-body effects that manifest themselves in the higher order terms of the potential energy of a given configuration. In atomic clusters, the potential energy function can be described as the summation of two-, three- and higher many-body terms as in equation 6.

$$V_{total} = \sum_{i<j} V_{i,j} + \sum_{i<j<k} V_{i,j,k} + ... \qquad (6)$$

The two-body terms, $(V_{i,j})$ encompass all pairs of particles and three-body terms, $\left(V_{i,j,k}\right)$ encompass all unique sets of three particles, the number of two- and three-body terms scales as N^2 and N^3, respectively. Often, the higher order terms are small compared to the two-body terms and can be neglected or approximated by other means. Also for large N, it becomes unmanageable to handle the higher many-body terms. Thus, we will restrict the scope of our work to the fully pair-wise (two-body) model and ignore the contribution of the higher terms, which is a reasonable physical approximation in the study of weakly interacting atomic clusters.

4.2. Wavefunction calculation

Figure 3 shows the general-shape of the wavefunction applicable in atomic and molecular clusters. The wavefunction is generally taken as the product of one-body (T_1), two-body (T_2) and three-body (T_3) terms as given in equation 7.

$$\psi = \prod_i T_1(r_i) \prod_{i<j} T_2(r_{ij}) \prod_{i<j<k} T_3(r_{ij}, r_{ik}, r_{jk}) \qquad (7)$$

We can observe that the many-body effects also result in higher-order terms here as with the case of potential energy function. For our purposes, we ignore the three-body correlation functions and work with the two-body interactions to evaluate the wavefunction. The one-body terms are unnecessary since they may be represented by the pair-wise terms.

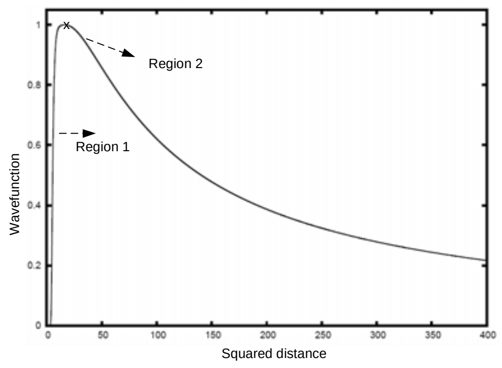

Figure 3. Plot of wavefunction

The wavefunction requires no transformation techniques. However, we rescale the wavefunction so that the maximum is less than 1. We use a similar region classification approach for the wavefunction consisting of regions I and II. The cut-off, σ computed as the maximum value of the wavefunction by setting its first derivative to zero, serves as a good dividing point so that we can use our existing binning schemes and yet accurately approximate the function. A quadratic polynomial interpolation will be performed on this rescaled wavefunction. The advantage of our approach lies in the fact that we can re-use the same hardware developed for the energy calculations for the wavefunction. This significantly reduces our design time as we now only need to calculate new constants and a different set of interpolation coefficients.

5. Architecture

Figure 4 shows the overall top-level block diagram of our design. The software Quantum Monte Carlo application runs on the host processor. The Potential Energy (PE) and WaveFunction (WF) calculations are performed on the FPGA. As a result, the host must provide the inputs to the FPGA and read results from the FPGA so that the functions implemented on the FPGA can be integrated with the host application. The design implemented on each FPGA consists of the following modules: Position Memory, {`PE,WF`} calculation engines, {`PE,WF`} Coefficient Memory. The {`PE, WF`} engines consist of the following pipelined components: `CalcDist`, `CalcFunc` and `AccFunc`. `CalcDist` computes the squared distance between a pair of atoms and sends it to the `CalcFunc` modules that produces the intermediate potentials and wavefunctions every clock cycle. The `AccFunc` accumulates these intermediates to their final values and sends them to the host processor. The host processor scales the results back to their original values and reconstructs the floating-point values of the functions.

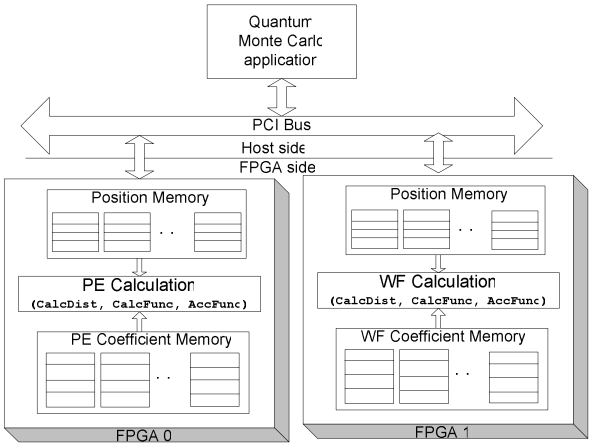

Figure 4. Top-level Block diagram

5.1. Binning methods

Here we discuss the binning schemes employed in the two regions and the memory platform used to accommodate the interpolation coefficients for the two regions.

5.1.1. Region I and II Binning Schemes

We use a quadratic polynomial interpolation on the transformed potential and wavefunction which are both functions of the squared distance. The schemes to look up the interpolation coefficients are different for the two regions due to their non-identical numerical behavior. We divide the region I function to accommodate 256 bins. Determining the interpolation constants for region I is straightforward and a single stage lookup is sufficient to lookup the constants from a table of coefficients at uniformly spaced intervals ranging from $r_{ij}^2 = 0$ to $r_{ij}^2 = \sigma^2$. The width of each bin is used to choose from the 256 values corresponding to the squared distances. The bin lookup scheme for region I is shown in Figure 5. The lower 8-bits form the address that is used to fetch the interpolation coefficients from the memory.

We employ a logarithmic binning scheme in order to represent region II function accurately. As discussed before, the region II function is divided into subregions called regimes. In our case, we divide the region II into 21 regimes. Each regime is divided into 64 bins for a total of 1344 coefficients. Hence we have a total of [256 + 21 * 64] * 3 coefficients (for quadratic polynomial interpolation). A two-stage lookup procedure is used, first to determine the regime by performing a leading zero count and then determining the set of coefficients to be used in the interpolation. The reciprocal of the bin widths is stored eliminating the need for a division operation. The block diagram of the first stage of the lookup scheme for region II is shown in Figure 6. The difference between the squared distances and the σ^2 value is used by the leading zero count detector (LZCD) to compute the regime. The LZCD logic is implemented using a set of three priority encoders (Pr1, Pr2, Pr3). Figure 7 shows the second stage of the lookup scheme to obtain the actual set of interpolation coefficients after the regime lookup is complete. To compute the bin location for region II, we require additional constants which are obtained from memory addressed using regime. We can use the computed address to retrieve the coefficients for region 2.

5.1.2. Memory Platform

The availability of on-chip Block RAMs (BRAMs) on the present FPGAs allows us to store the various parameters needed by our system. We use the BRAMs for the following purposes: `Position Memory`, which stores the positions of the atoms, `PE Coefficient Memory`, which stores the interpolation coefficients for regions I and II for potential energy calculation, `WF Coefficient Memory`, which stores the interpolation coefficients for regions I and II for wavefunction calculation. The BRAMs used to store the system's configurations are dual-ported and thereby allow simultaneous memory access to write configurations sent by the host processor and also access by the hardware modules to consume these configurations. Presently, our hardware block is attached to the 32-bit On-Chip Peripheral Bus (OPB) of the target FPGA. Hence, we use a memory bank structure consisting of dual-port BRAMs to implement the memory platform.

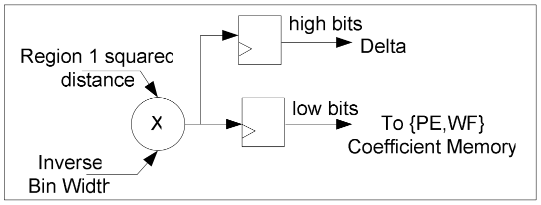

Figure 5. Region I Bin lookup scheme

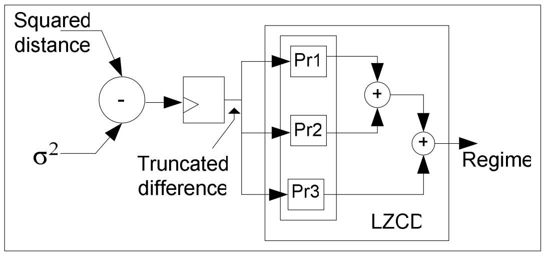

Figure 6. Region II bin lookup (1st stage)

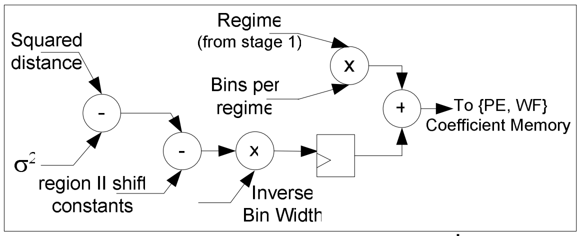

Figure 7. Region II bin lookup (2nd stage)

5.2. Description of pipelines

Here, we describe the details of the {PE, WF} calculation engines. These two calculation engines are nearly identical with the exception of the final accumulation step. We use the following pipelined components: `CalcDist` to compute the co-ordinate distances between the atoms, the generic `CalcFunc` to compute potential and wavefunction and an `AccFunc` module to accumulate the resulting values.

5.2.1. Calculate Distance (`CalcDist`)

The `CalcDist` block calculates the squared distances between pairs of atoms. The host processor stores the *R* (*x*, *y*, *z*) configurations of atoms for every iteration onto the off-chip memory, which are then transferred to the on-chip `Position Memory`. An address generator is used to provide read addresses to the `Position Memory` to read the pair positions, (x_i, y_i, z_i) and (x_j, y_j, z_j) every clock cycle and provide them to the distance calculation module. Given *N* particles, there are *N* (*N*-1)/2 coordinate distances. The state machine repeats the address generation for all the *N* particles. The data path of the `CalcDist` block is shown in Figure 8 with the latencies shown in clock cycles. This module uses a 32-bit fixed-point representation for the positions and produces a 52-bit squared distance value per clock cycle. The resulting squared distances are compared to σ^2 and classified as region I or region II r^2 values for the respective functions.

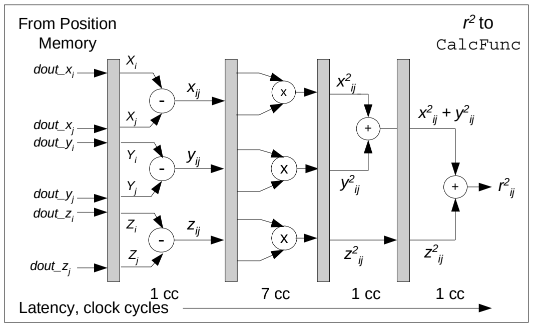

Figure 8. CalcDist pipeline

5.2.2. Calculate Function (`CalcFunc`)

Figure 9 shows the `CalcFunc` pipeline. This generic pipeline that can compute potential and wavefunction consumes the squared distances every clock cycle. We use the lookup schemes discussed earlier to fetch the interpolation coefficients and a delta value pertinent to the squared distances. These are processed by the `CalcFunc` module to produce a result every clock cycle once the pipeline is full. The latency of the pipeline is 49 clock cycles.

5.2.3. Accumulate Function (`AccFunc`)

The `AccFunc` module accumulates the energy and wavefunction intermediate values from the `CalcFunc` pipeline. Since we interpolate the transformed potential, we accumulate the potential as a running product in region I and running sum in region II. The transformed potential in region I is no larger than unity in value. Hence, repeated multiplication during accumulation of these potential values results in a potential that will tend to zero. In a fixed-precision register, the appearance of leading zeros results in a loss of precision. To guarantee that we do not lose precision during accumulation in region I, we introduce a bit shift to the left after computing each product (if it is less than 2^{-1}) and incrementing an initially denormalized exponent. To take care of overflow issues associated with an accumulator, the region II potentials are accumulated in a register of fixed-precision large enough to hold *N* evaluations of the maximum value (1.0) of the rescaled potential. The results will be delivered to the host, which removes the scaling and combines the results from region I and II to reconstruct the floating-point value of the total potential energy.

In the case of potential function, transformation schemes were employed which necessitated the use of both sum and product accumulators for the respective regions. However, we may recall that the wavefunction did not undergo any transformation. The accumulation step is necessary only due to the functional form taken by the wavefunction. Hence, we bypass the region II sum accumulator and route all the results to the region I product accumulator.

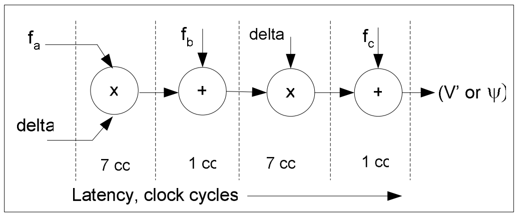

Figure 9. `CalcFunc` pipeline

6. Implementation and Results

The overall system consists of 2.4 GHz dual processor Intel Xeon with two AP130 boards from Amirix Systems. The entire chemistry application runs on the Linux based host and compiled using the GNU gcc version 3.3.5 compiler with –O3 optimizations. The kernels of the chemistry application are coded in VHDL and the design synthesized using Synplicity Synplify tools and integrated into the Xilinx tool flow. The place and route is performed using Xilinx 7.1 ISE/EDK tools. Since we use a hardware/software approach in our design, we only need to replace the wavefunction and potential energy function calls in software with the FPGA function calls. Transferring the random configurations from the host processor to the FPGA and the FPGA-host communication is accomplished through the SDRAM on the FPGA over the 66 MHz PCI interface using the Amirix control program. A polling-based approach is used to transfer data and status signals through SDRAM between the host and the FPGA. The user cores are attached to the On-Chip Peripheral Bus (OPB) which operates at 40 MHz for the Amirix baseline platform. The PowerPC processor initializes the interpolation coefficients.

6.1. Error Analysis

Scientific applications commonly use floating-point representation as they allow for high precision arithmetic and higher dynamic range compared to fixed-point representation. The QMC software application employs 64-bit double precision floating-point representation. However, floating-point representation on the hardware is more expensive and slower than fixed-point representation. The fixed-point representation used for the squared distances fed to the pipelines and potential and wavefunction results (prior to accumulation) are shown in the Table 1. Using a fixed-point representation allows us to save area and increase the speed of operation. Since the potential and wavefunction values are rescaled to be less than or equal to 1, their fixed-point representations use 52 bits after the decimal point. These representations deliver the required accuracy for our application.

Table 1. Fixed-point formats

Parameter	Fixed-point formats (s-signed, u-unsigned)
(x, y, z) positions	s12.20
Squared distances	u27.26
Potential energy and wave-function	s0.51
Interpolation coefficients	s0.51

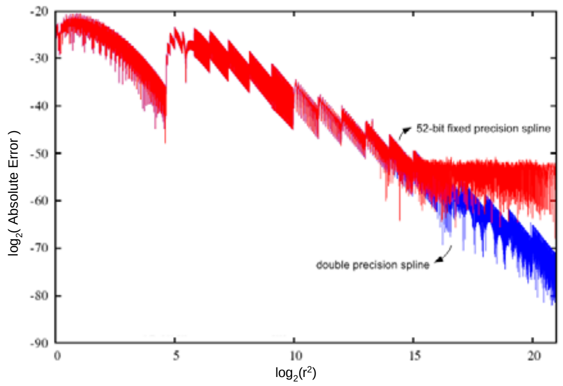

Figure 10. Absolute error potential function

Figure 10 shows the absolute error of potential, of the 52-bit hardware spline versus the same spline expressed in 64-bit double precision (52-bit mantissa) floating point representation. The plot is shown on a log (to base 2) scale so that we can equate the error with the bits of precision. The plots show that the error for fixed-point spline levels out at our 52-bit maximum fixed-point limit, although the floating-point representation has smaller error for large r^2 and is more precise. Comparing our potential values to a floating-point version, we can see that the fixed-point version accurately reproduces the floating-point results over almost the entire range. Since the squared distances in our application are confined within 2^{16}, we can safely ignore the degradation of accuracy in this poor region.

6.2. Performance

Table 2 shows the usage of resources of the PE and WF kernels targeted to Xilinx Virtex II-Pro, VP30 on the Amirix AP130 FPGA board. Table 3 compares the actual usage of resources (SLICEs, BRAMs, 18x18 MULTs) on the XC2VP30 and the estimated usage on XC2VP50 present on the Cray XD1 platform and the usage of SLICEs, BRAMs, DSPs on Virtex-4 FPGA [23] device for the PE and WF engines. In our current baseline platform, we are limited by available resources, which prohibits realizing additional pipelines. Each core occupies only 28% of the SLICEs on XC2VP50, but we can fit another potential pipeline or wavefunction pipeline to provide additional parallelism. On the Virtex-4 part, we could place additional pipelines and they would work independently using different configurations. After deploying multiple pipelines on these high end FPGAs, we could use the available resources to calculate derivatives of these functions and other related ground-state properties, which are often of utmost importance for studying *N*-body systems. Table 4 shows the execution profile (units in seconds) for the FPGA accelerated QMC application and the software only QMC application running on an Intel Xeon, 2.4 GHz dual-processor with the PCI FPGA boards. Our design can presently simulate a system of up to 4000 atoms. The simulation is performed for 100 iterations, although increasing the iterations reduces the statistical uncertainty in the computed properties.

From table 4, we observe that a significant amount of time is spent on overheads consisting of data-processing and transfer to the FPGA over the 66 MHz PCI. This is because we are presently using a development board with a slow interface; use of a system with a better communication interface could easily provide an order of magnitude improvement. The rest of the application consists of creating a reference configuration and using SPRNG to add a random displacement to the above configuration. For the current system, we can observe a speedup of 3x over the software only QMC application. Since we typically deal with a large number of particles in quantum chemistry simulations, our reconfigurable hardware implementation can provide potential speedups greater than 5x while accelerating the potential and wavefunction kernels in the absence of communication bottlenecks. Also, we can speedup our overall application by identifying critical portions remaining that could be implemented using reconfigurable hardware.

Table 2. Resource usage on AP130 VP30

Resource type	Potential energy (on FPGA0)	Wavefunction (on FPGA1)
SLICEs (13696)	11386 (83%)	9920 (72%)
BRAMs (136)	100 (73%)	100 (73%)
18x18 multipliers (136)	85 (62%)	109 (80%)

Table 3. Resource usage (post place and route) on Xilinx FPGAs (VP30, VP50, Virtex 4)

Resource type / Target FPGA	SLICEs (%)		BRAMs (%)		18x18 MULTs/ DSPs (%)	
	PE	WF	PE	WF	PE	WF
Virtex-II Pro XC2VP30	48	46	50	50	62	80
Virtex-II Pro XC2VP50 (estimated)	28	27	29	29	36	47
Virtex4 XC4VFX140 (estimated)	14	11	7	7	33	33

Table 4. Execution profile of a 4000 atom simulation over 100 iterations (in seconds)

	PE + WF kernels	Rest of the application	Overhead	Total
FPGA accelerated application	61.8	108.73	68.4	238.93
Software only QMC application	664.51	108.88	--	773.39

7. Conclusions and Future Work

We present a novel architecture to calculate pairwise functions of N-body systems using Quantum Monte Carlo simulations. We have carefully partitioned our software application such that the computationally intensive potential and wavefunction calculations are performed on the reconfigurable hardware and the remaining calculations are performed on the host processor. The design uses fixed-point representations for all values within the system thus reducing the area and complexity compared to floating-point implementation. This design allows us to compute any function of the position co-ordinates using a general interpolation framework. Based on the results reported, we will be able to fit additional wavefunction or potential pipelines on the next-generation FPGAs. Presently, quadratic interpolation empirically provides a good balance between numerical accuracy and usage of resources. However, additional block memories available on the current high-end FPGAs will enable us to choose a different order of interpolation, e.g. cubic splines to provide increased accuracy. We will also identify possible functions from the rest of the application which can be accelerated using FPGAs.

Partitioning the data suitably will allow us to operate multiple copies of each pipeline on a single FPGA or place copies of each pipeline on multiple FPGAs, thus taking advantage of the coarse-grained parallelism. We are working on extending our present prototype platform to include multiple processing nodes and employing a Message Passing Interface (MPI) model for communication between the nodes. Each processor will have its own FPGA board. Each slave FPGA on the host processor can work on its own set of configurations. There would be no communication required among the FPGAs themselves. We will use MPI at the software level to co-ordinate the host processors. With such a setup, the total speedup would be linear in the number of FPGAs times the speedup of a single FPGA. Following this, we will port our design onto the Cray XD1 located at the Oak Ridge National Laboratory. This Cray XD1 system, known as Tiger, consists of 144 64-bit Opteron processors arranged in

72 nodes of two processors each. Six of the nodes include application acceleration processors (FPGA) providing a tight integration of FPGA resources and host microprocessors over the system's high bandwidth and low latency interconnect. A platform like the Cray XD1 with multiple FPGAs and compute nodes will allow us to place multiple copies of the calculation pipelines, thereby maximizing the achievable performance.

Acknowledgements

This work was supported by the National Science Foundation grant, NSF CHE-0625598, and the authors gratefully acknowledge prior support for related work from the University of Tennessee Science Alliance.

References

[1] Cray Inc., http://www.cray.com/products/xd1/index.html

[2] SRC Computers, Inc. http://www.srccomp.com/HardwareSpecs.htm

[3] SGI, http://www.sgi.com/products/rasc/

[4] DRC Computer Corporation, , http://www.drccomputer.com/

[5] W. D. Smith and A. R. Schnore, "Towards an RCC-based accelerator for computational fluid dynamics applications," *Intl Conf on Engineering Reconfigurable Systems and Algorithms*, pp. 226-232, June 2003.

[6] N. Azizi, I. Kuon, A. Egier, A. Darabiha, P. Chow, "Reconfigurable Molecular Dynamics Simulator," *IEEE Intl Symp on Field-Programmable Custom Computing Machines*, pp. 197-206, April 2004.

[7] A. Gothandaraman, G. L. Warren, G. D. Peterson, R. J. Harrison, "Reconfigurable Accelerator for Quantum Monte Carlo Simulations in N-body Systems," *Supercomputing* 2006.

[8] J. L. Tripp, A. A. Hanson, M. Gokhale, H. Morteveit, "Partitioning Hardware and Software for Reconfigurable Supercomputing Applications: A Case Study," *Supercomputing* 2005.

[9] Amirix, http://www.amirix.com/

[10] Virtex-II Platform FPGAs: complete data sheet, http://direct.xilinx.com/bvdocs/publications/ds031.pdf

[11] J. Doll, D. L. Freeman, "Monte Carlo Methods in Chemistry," *IEEE Computational Science and Engineering*, vol.1, issue 3, pp. 22-32, Spring 1994.

[12] The Scalable Parallel Random Number Generators Library (SPRNG), Florida State University, http://sprng.cs.fsu.edu/

[13] R. Scrofano, M. Gokhale, F. Trouw, V. K. Prasanna, "A Hardware/Software Approach to Molecular Dynamics on Reconfigurable Computers," *IEEE Symp on Field-Programmable Custom Computing Machines*, 2006.

[14] G. L. Zhang, P. H. W. Leong, C. H. Ho, K. H. Tsoi, et al., "Reconfigurable Acceleration for Monte Carlo based Financial Simulation," *IEEE Intl Conf on Field-Programmable Technology*, pp. 215-222, Dec 2005.

[15] J. M. McCollum, J. M. Lancaster, D. W. Bouldin and G. D. Peterson, "Hardware acceleration of pseudo-random number generation for simulation applications," *Proc of the 35th Annual Southeastern Symposium on System Theory*, pp. 299–303, March 2003.

[16] M. Gokhale, J. Frigo, C. Ahrens, J. L. Tripp, R. Minnich, "Monte Carlo Radiative Heat Transfer Simulation on a Reconfigurable Computer," *Intl Conf on Field-Programmable Logic and Applications*, 2004.

[17] C. P. Cowen and S. Monaghan, "A reconfigurable Monte-Carlo clustering processor (MCCP)," *Proc. of the IEEE Symposium on Field Programmable Custom Computing Machines (FCCM)*, pp. 59-65, 1994.

[18] J. Makino, "The GRAPE project," *Computing in Science & Engineering*, vol. 8, pp. 30-40, Jan.-Feb. 2006.

[19] M. Taiji, T. Narumi, Y. Ohno, "Protein Explorer: A Petaflops Special-Purpose Computer System for Molecular Dynamics Simulations," *SC*, Nov 2003.

[20] N. Nakasato, T. Hamada, "Astrophysical Hydrodynamics Simulations on a Reconfigurable System," *IEEE Symposium on Field-Programmable Custom Computing Machines*, pp. 279-280, 2005.

[21] Y. Bi, G. D. Peterson, G. L. Warren and R. J. Harrison, "Hardware Acceleration of Parallel Lagged-Fibonacci Pseudorandom Number Generation," *Intl Conf on Engineering Reconfigurable Systems and Algorithms*, June 2006.

[22] Roy Wikramaratna, "Pseudo-random Number Generation for Parallel Monte Carlo – A Splitting Approach," *SIAM News*, vol. 33, Number 9, 2000.

[23] Virtex-IV FPGAs: Overview , http://direct.xilinx.com/bvdocs/publications/ds112.pdf