Computer Physics Communications 180 (2009) 2563–2573

Contents lists available at ScienceDirect

Computer Physics Communications

www.elsevier.com/locate/cpc

40th Anniversary Issue

A Hardware-Accelerated Quantum Monte Carlo framework (HAQMC) for N-body systems ☆

Akila Gothandaraman [a], Gregory D. Peterson [a,*], G. Lee Warren [b], Robert J. Hinde [c], Robert J. Harrison [c]

[a] Department of Electrical Engineering and Computer Science, University of Tennessee, Knoxville, United States
[b] Department of Chemistry, Yeshiva University, United States
[c] Department of Chemistry University of Tennessee, Knoxville, United States

ARTICLE INFO

Article history:
Received 18 April 2009
Received in revised form 23 June 2009
Accepted 29 June 2009
Available online 2 July 2009

PACS:
02.70.Ss
07.05.Bx
07.05.Tp

Keywords:
Reconfigurable computing
Field-programmable gate arrays
FPGA
Cray XD1
Monte Carlo
Quantum Monte Carlo

ABSTRACT

Interest in the study of structural and energetic properties of highly quantum clusters, such as inert gas clusters has motivated the development of a hardware-accelerated framework for Quantum Monte Carlo simulations. In the Quantum Monte Carlo method, the properties of a system of atoms, such as the ground-state energies, are averaged over a number of iterations. Our framework is aimed at accelerating the computations in each iteration of the QMC application by offloading the calculation of properties, namely energy and trial wave function, onto reconfigurable hardware. This gives a user the capability to run simulations for a large number of iterations, thereby reducing the statistical uncertainty in the properties, and for larger clusters. This framework is designed to run on the Cray XD1 high performance reconfigurable computing platform, which exploits the coarse-grained parallelism of the processor along with the fine-grained parallelism of the reconfigurable computing devices available in the form of field-programmable gate arrays. In this paper, we illustrate the functioning of the framework, which can be used to calculate the energies for a model cluster of helium atoms. In addition, we present the capabilities of the framework that allow the user to vary the chemical identities of the simulated atoms.

Program summary

Program title: Hardware Accelerated Quantum Monte Carlo (HAQMC)
Catalogue identifier: AEEP_v1_0
Program summary URL: http://cpc.cs.qub.ac.uk/summaries/AEEP_v1_0.html
Program obtainable from: CPC Program Library, Queen's University, Belfast, N. Ireland
Licensing provisions: Standard CPC licence, http://cpc.cs.qub.ac.uk/licence/licence.html
No. of lines in distributed program, including test data, etc.: 691 537
No. of bytes in distributed program, including test data, etc.: 5 031 226
Distribution format: tar.gz
Programming language: C/C++ for the QMC application, VHDL and Xilinx 8.1 ISE/EDK tools for FPGA design and development
Computer: Cray XD1 consisting of a dual-core, dualprocessor AMD Opteron 2.2 GHz with a Xilinx Virtex-4 (V4LX160) or Xilinx Virtex-II Pro (XC2VP50) FPGA per node. We use the compute node with the Xilinx Virtex-4 FPGA
Operating system: Red Hat Enterprise Linux OS
Has the code been vectorised or parallelized?: Yes
Classification: 6.1
Nature of problem: Quantum Monte Carlo is a practical method to solve the Schrödinger equation for large many-body systems and obtain the ground-state properties of such systems. This method involves the sampling of a number of configurations of atoms and averaging the properties of the configurations over a number of iterations. We are interested in applying the QMC method to obtain the energy and other properties of highly quantum clusters, such as inert gas clusters.

☆ This paper and its associated computer program are available via the Computer Physics Communications homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).
* Corresponding author.
E-mail addresses: akila@utk.edu (A. Gothandaraman), gdp@utk.edu (G.D. Peterson), gwarren@yu.edu (G.L. Warren), rhinde@utk.edu (R.J. Hinde), robert.harrison@utk.edu (R.J. Harrison).

0010-4655/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.cpc.2009.06.027

Solution method: The proposed framework provides a combined hardware–software approach, in which the QMC simulation is performed on the host processor, with the computationally intensive functions such as energy and trial wave function computations mapped onto the field-programmable gate array (FPGA) logic device attached as a co-processor to the host processor. We perform the QMC simulation for a number of iterations as in the case of our original software QMC approach, to reduce the statistical uncertainty of the results. However, our proposed HAQMC framework accelerates each iteration of the simulation, by significantly reducing the time taken to calculate the ground-state properties of the configurations of atoms, thereby accelerating the overall QMC simulation. We provide a generic interpolation framework that can be extended to study a variety of pure and doped atomic clusters, irrespective of the chemical identities of the atoms. For the FPGA implementation of the properties, we use a two-region approach for accurately computing the properties over the entire domain, employ deep pipelines and fixed-point for all our calculations guaranteeing the accuracy required for our simulation.

1. Introduction

Monte Carlo methods [1] are used to study a number of problems that involve stochastic behavior in science, engineering, and finance. Monte Carlo methods are tools that can be used to calculate high-dimensional integrals and study N-body systems. In physics and physical chemistry, these methods are used to simulate many-particle systems using random numbers. Quantum Monte Carlo (QMC) methods provide a direct and efficient way of solving the many-body Schrödinger equation, the fundamental equation of quantum mechanics. Solving this equation is trivial for small systems; however, analytically solving this equation to study the interactions in a system with hundreds or thousands of atoms is impractical. QMC is a stochastic method that can accurately treat many-body quantum mechanical systems to obtain the ground-state properties of clusters of atoms or molecules.

We use this method to obtain the total ground-state energy for highly quantum clusters such as systems of inert gas atoms. In the QMC algorithm employed in our research, we calculate properties such as the contributions to the potential energy, kinetic energy, and the trial wave function for each pair of atoms in the N-body system. These functions or kernels have a high computational cost and hence represent ideal candidates for acceleration using high performance computing systems. For our target system, we use a reconfigurable computing system, which combines a general-purpose processor with reconfigurable logic in the form of field-programmable gate arrays (FPGAs). We have developed a user-friendly framework that allows a user to simulate a system of inert gas atoms using a hardware-accelerated QMC framework to map these functions onto the FPGA. We use a parallel and pipelined architecture to realize these functions on the reconfigurable hardware. A hardware–software partitioning method is employed with the control-intensive functions of the QMC application retained on the general-purpose processor and the compute-intensive portions accelerated using reconfigurable computing. The computationally intensive tasks such as finding energies and trial wave functions are approximated using a two-region approach followed by quadratic interpolation [2]. We provide the software QMC application, the interpolation coefficients for the evaluation of kernel functions in hardware along with the cut-off value that separates the two regions specified by our two-region approach [2]. Since the Monte Carlo method involves weighted statistical sampling, for each iteration, the processor writes to the FPGA memory $O(N)$ position co-ordinates of the atoms in the N-body system and the FPGA returns $O(N)$ results in the form of energies or wave functions to user registers which can be read back by the host processor. We employ a fixed-point representation, chosen after careful error analysis that guarantees the required accuracy for our calculations. The floating-point to fixed-point transformations and vice versa are done on the host processor.

Our existing QMC framework can be used for studying helium clusters. Advanced users can apply our framework to perform the following functions: (a) change the analytical function used for the energy or wave function to study other inert gas clusters, (b) modify the FPGA implementation to increase the number of systems or change the hardware interfaces to potentially target the application to a different reconfigurable computing platform, or (c) increase the fixed-point precisions or order of interpolation to more accurately approximate the function on FPGAs with more hardware resources.

Our paper is organized as follows. Section 2 provides a background of reconfigurable computing and the Quantum Monte Carlo algorithm. We also provide details of the Cray XD1 high performance reconfigurable computing platform. In Section 3, we describe our model system along with the analytical functions for the energy and wave function. In Section 4, we briefly describe our implementation along with the results from targeting our design to the Cray high performance reconfigurable computing platform. In Section 5, we provide the instructions for installing and using the framework. We provide conclusions and some extensions of our framework in Section 6.

2. Background

Our framework is aimed at accelerating QMC simulations using customized circuitry for compute-intensive kernels. We use a hardware–software partitioning approach and offload the kernels to the reconfigurable logic device connected through a high performance interconnect to the host processor. Here, we provide an overview of reconfigurable computing and the Quantum Monte Carlo (QMC) simulation method. We introduce the Cray XD1 platform that is the target platform of our QMC framework. We also describe the Variational QMC algorithm implemented in our framework.

2.1. Reconfigurable Computing

Reconfigurable Computing (RC) is the combination of reconfigurable logic with a general-purpose microprocessor [3,4] aimed at providing more performance than possible with software-only solutions on general-purpose processors and increased flexibility compared to an application specific integrated circuit (ASIC). Reconfigurable logic devices such as field-programmable gate arrays (FPGAs) consist of a matrix of flexible logic blocks that can be configured to implement the required logic functions by connecting them using programmable routing resources. The FPGA and the microprocessor in an RC system are commonly connected using interfaces such as PCI, VME, or HyperTransport. Present FPGAs have increased gate densities that allow the mapping of complex designs onto these devices. In addition to the two main resources on an FPGA, namely the logic elements and interconnect, current

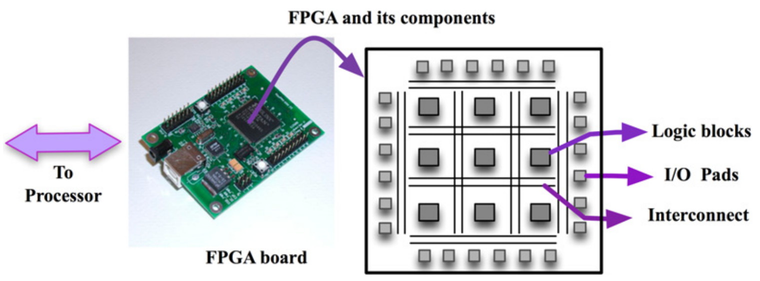

Fig. 1. A reconfigurable computing system and FPGA components.

FPGA vendors have provided embedded resources that are used to achieve high performance for commonly used functions. For example, on the Xilinx FPGAs, these resources include hardware multipliers, on-chip memories known as Block RAMs, digital signal processing blocks, and even PowerPC processors [5]. In [6], the authors provide a survey on reconfigurable architectures and a discussion of their representative applications including signal and image processing, bioinformatics, and supercomputing. Fig. 1 shows a reconfigurable computing system and the components of an FPGA.

2.2. High Performance Reconfigurable Computing

High Performance Reconfigurable Computing (HPRC) is the combination of a High Performance Computing (HPC) system with RC elements to provide increased performance and flexibility. The computing nodes of HPRC systems are connected using a high performance, low latency interconnection network with some or all nodes equipped with one or more reconfigurable elements. Using these systems, we can take advantage of “polygranular” parallelism, i.e., the fine-grained parallelism offered by reconfigurable computing along with the coarse-grained parallelism typically available using parallel computing. There are a number of FPGA-based HPRC solutions such as Cray XD1/XT5$_h$ [7,8], SRC Computers [9], SGI [10] DRC Computers [11], and Maxwell [12]. Our target system for the hardware accelerated QMC framework is the Cray XD1 system. The Cray XD1 is an HPRC platform from Cray Inc., which integrates FPGA coprocessors in their supercomputers [7]. The Cray XD1 system can contain one to hundreds of chassis, which is the architectural unit of the Cray XD1. An XD1 chassis consists of up to six compute blades. Each compute blade consists of two 64-bit AMD Opteron processors connected via AMD’s Hypertransport. A maximum of upto 12 RapidArray Processors (RAPs) are also present to process communications within the chassis. The chassis also contains the application acceleration system with six optional FPGAs, which are tightly coupled to the Opteron’s address space and serve as coprocessors to the Opteron processors. The RapidArray processors provide a 3.2 GB/sec interface for the FPGAs to the processors on the chassis as well as to the RapidArray fabric. The Cray XD1 chassis [7] is shown in Fig. 2.

We target our implementation to *Pacific*, a Cray XD1 single chassis system comprised of six compute blades, where each blade consists of a dual-core dual-processor 2.2 GHz AMD Opteron with 8 GB local memory connected to Cray’s proprietary RapidArray fabric. Each compute blade in a chassis is equipped with an FPGA application accelerator module consisting of either a Xilinx Virtex-II Pro VP50 or Virtex-4 LX160 FPGA that allows user logic to be clocked at up to 199 MHz. The QMC application runs on a single core of the Cray XD1 and the potential and wave function kernels are implemented on the Virtex-4 LX 160 FPGA. The use of a higher gate density FPGA such as the Virtex-4 allows us to map the energy and wave function pipelines on a single FPGA. Cray Inc., also provides the Application Acceleration API, which are a set of functions to access the FPGA within an application on the

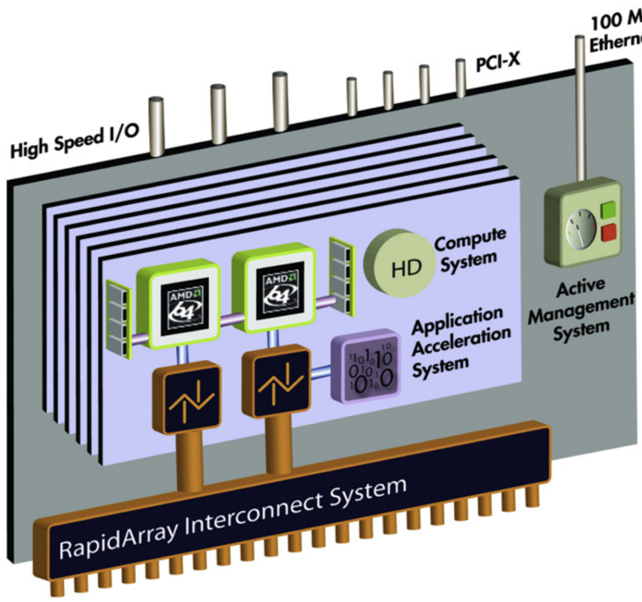

Fig. 2. Cray XD1 system Chassis [7].

Opteron using the Linux operating system. The command line FPGA control utility provided by Cray can also be used to control the FPGA, such as resetting the FPGA or loading the bitstream onto the FPGA. The programming interface provided by Cray for the XD1 system greatly simplifies interfacing an application running on the Opteron with the design implemented on the FPGA.

2.3. Quantum Monte Carlo

Quantum Monte Carlo methods provide a practical and efficient way of solving the many-body Schrödinger equation, the fundamental equation of quantum mechanics [13]. They have the ability to accurately treat many-body quantum mechanical systems to obtain the ground-state properties of clusters of atoms or molecules. Two distinct types of QMC methods are the Variational Monte Carlo (VMC) and Diffusion Monte Carlo (DMC), each possessing its own strengths and weaknesses. VMC is simpler and faster whereas DMC is more accurate but complex. Our hardware accelerated QMC framework is based on the VMC method for a cluster of N rare gas atoms interacting through a pair-wise additive potential. The VMC method applies Monte Carlo integration to calculate the multi-dimensional integrals of expectation values such as the total energy. Schrödinger’s equation is given in Eq. (1). Eq. (2) gives the one-dimensional time-independent Schrödinger equation for a chargeless particle of mass m moving in a potential $V(x)$. Eq. (3) gives the analogous three-dimensional time independent equation. Solving this equation is trivial for small systems, but as the dimensions of the system increase, it is impractical to solve the equation analytically.

$$H\psi = E\psi, \tag{1}$$

$$-\frac{\hbar^2}{2m}\frac{d^2\psi(x)}{dx^2} + V(x)\psi(x) = E\psi(x), \tag{2}$$

$$\left[-\frac{\hbar^2}{2m}\nabla^2 + V(r)\right]\psi(r) = E\psi(r). \tag{3}$$

In the above equations, H refers to the Hamiltonian operator, E represents the energy of the system, ψ is the wave function, and ∇^2 is the Laplace operator.

2.3.1. Variational Monte Carlo algorithm

The VMC method applies the variational method to approximate the ground state of the system. This method employs a set of adjustable parameters to yield a trial wave function, $\psi_T(x)$, that when optimized, best approximates the exact wave function. The VMC method has the advantage of being simple and easy to implement. Also, the method is relatively insensitive to the size of the system and hence can be applied to large systems where some other methods are computationally infeasible. Fig. 3 shows the pseudo-code for the VMC algorithm. Fig. 4 shows a flow chart describing the steps of the algorithm and our hardware–software partitioning approach. The shaded blocks in Fig. 4 correspond to the parts of the algorithm, where we compute the ground-state properties, chosen for hardware acceleration.

The VMC algorithm consists of two phases: Initialization phase and Iterative phase.

(a) Initialization phase: During the initialization phase, we provide the random number generator with an initial seed. In this implementation, we use the Additive Lagged Fibonacci Generator (ALFG) from the Scalable Parallel Random Number Generator (SPRNG) library [14]. The ALFG generator is a recurrence-based generator that has two important properties for MC simulations; the maximum period of the generator is 2×10^{394} and distinctly seeded random number generators can run in parallel with no correlation between the random numbers. Hence, we can run long simulations on scalable parallel systems while retaining statistical

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

Fig. 3. Variational Monte Carlo algorithm.

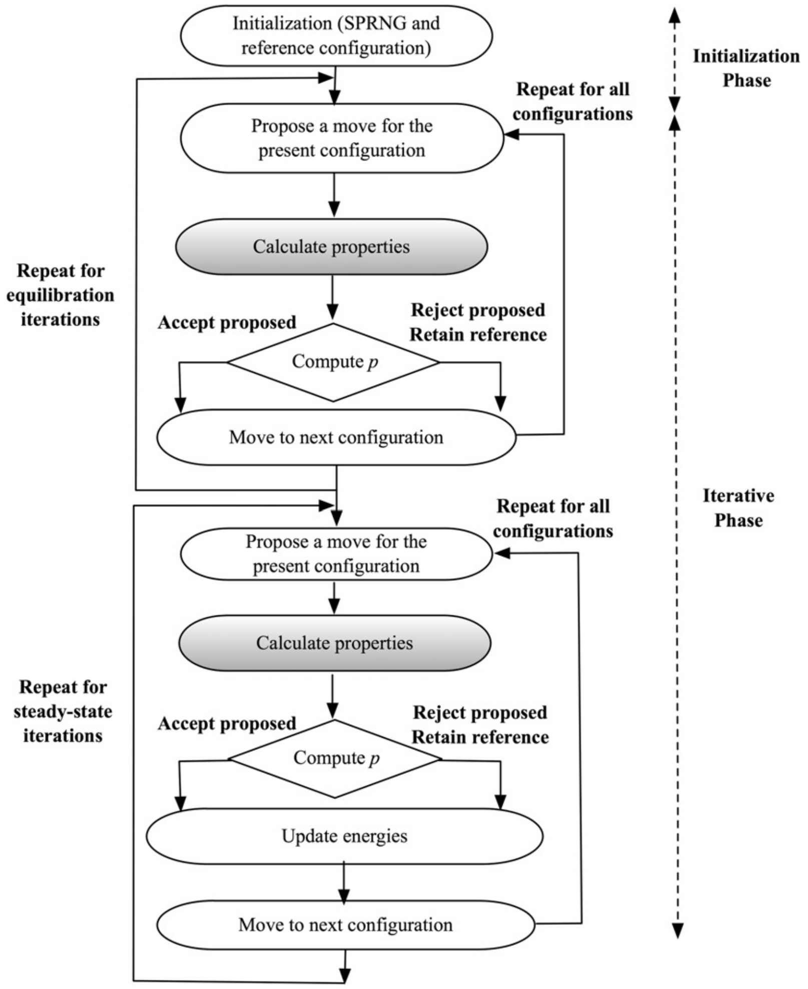
Initialization (SPRNG and reference configuration)
Initialization Phase
Repeat for all configurations
Propose a move for the present configuration
Calculate properties
Repeat for equilibration iterations
Accept proposed
Reject proposed Retain reference
Compute p
Move to next configuration
Iterative Phase
Repeat for all configurations
Propose a move for the present configuration
Calculate properties
Repeat for steady-state iterations
Accept proposed
Reject proposed Retain reference
Compute p
Update energies
Move to next configuration

Fig. 4. Flow chart of the variational Monte Carlo algorithm.

significance of our results. We begin by initializing the *reference configuration*, $R(x, y, z)$ of the system of atoms using a Cartesian co-ordinate system. We compute the properties, potential energy, trial wave function, and the derivatives of the trial wave function to obtain the kinetic energy. The sum of the potential and kinetic energies gives us the total energy of the atoms in the reference configuration.

(b) Iterative phase: In the iterative phase, we have a set of "configurations" or "walkers" that sample the search space. For each configuration, we perturb its atoms to yield a new configuration, called a *proposed configuration*, $R'(x, y, z)$. This is done by adding small random displacements to the co-ordinate positions of the atoms in the current configuration. The next step is to compute the properties for the atoms in this configuration. We perform the following $O(N^2)$ calculations: distance calculation between pairs of particles, pair-wise potential energy, and pair-wise trial wave function computation. After obtaining the trial wave function, we also compute its first and second derivatives in order to compute the kinetic energy. The energies are summed to yield the total energy for this configuration. To ensure that the configurations are asymptotically drawn from the square of the known trial wave function $\psi_T(x)$, we accept or reject this configuration (and associated properties) by determining the fraction, p. The value of p is obtained using the ratio of the square of the trial wave function values of the proposed and the reference configurations. If the value of p is greater than or equal to 1, we accept the proposed configuration. Otherwise, we compare p with a uniformly distributed random number to decide whether to accept or reject the configuration. If the proposed configuration is accepted, we retain its properties; otherwise we keep the properties of the reference configuration. We usually repeat this process for a certain number of equilibration iterations. After we complete the equilibration iterations (EQ), we start updating the properties by accumulating the energies during the steady-state (SS) iterations. This is done for all configurations in the system and repeated for a total of ($I = EQ + SS$) iterations.

3. Model system

First, we provide the analytical functions for the potential energy function and trial wave function for a model system of helium atoms used in our framework. In the sections that follow, we will describe how the user can change the identities of the atoms simulated and extend the usage of this framework. We also describe how the dual-region approach is used to approximate each function for the FPGA implementation. For details of the FPGA hardware implementation, please refer to [2].

We apply the VMC method to investigate the quantum mechanical ground states of a rare gas cluster system, consisting of helium atoms. The extremely weak helium–helium interatomic interactions combined with the small atomic mass make the helium clusters weakly bound and show interesting properties, among which is their ability to attain a superfluid state. By far, the only methods that can accurately model highly quantum clusters such as helium clusters are the quantum Monte Carlo methods. They can be applied to study pure as well as doped helium clusters. In VMC, we choose a form for the trial wave function, $\psi_T(R)$, characterized by a set of parameters and evaluate the energy,

$$E_0 \leqslant \langle E \rangle = \int dR \, \frac{H\psi_T}{\psi_T} \rho(R) \tag{5}$$

where R is a $3N$-dimensional vector composed of the atom co-ordinates, (x, y, z). The trial wave function is a parameterized function, whose parameters can be varied to minimize the integral in Eq. (5) and $\langle E \rangle$ is an upper bound to the exact ground-state energy, E_0. The integral is evaluated using the Monte Carlo method, where we sample the atomic positions from the probability density function, $\rho(R)$ given by Eq. (6). E is estimated using Eq. (7). The probability density function is sampled using the Metropolis algorithm.

$$\rho(R) = \frac{\psi_T^2(R)}{\int dR \, \psi_T^2(R)}, \tag{6}$$

$$\langle E \rangle \approx \frac{1}{M} \sum_{i=1}^{M} \frac{H\psi_T(R_i)}{\psi_T(R_i)}. \tag{7}$$

From the above equations, we can see that the VMC algorithm is composed of the computationally intensive functions such as energy and trial wave function for a number of iterations, M. We present our methodology to accelerate each of these kernels using a general interpolation framework on the FPGA.

Kernel 1: Potential Energy Calculation

The Hamiltonian for the atomic co-ordinates for an N-atom cluster is given by Eq. (8), where r_{ij} is the distance between two helium atoms, and ∇_i^2 is the Laplacian of particle i. m is the atomic mass and $\hbar$ is Planck's constant divided by 2π. We will use a two-body model for our simulations and hence the potential energy, V_{total} is approximated as a sum of the $N(N-1)/2$ pair contributions as shown in Eq. (9).

$$H = \frac{-\hbar^2}{2m} \sum_{i=1}^{N} \nabla_i^2 + V_{total}, \tag{8}$$

$$V_{total} \approx \sum_{i<j}^{N} V(r_{ij}). \tag{9}$$

A number of helium–helium potentials are available in the literature [15,16] to accurately model the He–He interatomic interactions. The *SAPT2* helium–helium potential [16] is employed in this study. The functional form of the *SAPT2* potential is given in Eq. (10).

$$V_{SAPT2} = Ae^{-\alpha r + \beta r^2} - \left[1 - \left(\sum_{k=0}^{6} (\delta r)^k / k!\right) e^{-\delta r}\right] C_6 f(r)/r^6 - \sum_{n=4}^{8} \left[1 - \left(\sum_{k=0}^{2n} (\delta r)^k / k!\right) e^{-\delta r}\right] C_{2n}/r^{2n}. \tag{10}$$

Fig. 5 shows the *SAPT2* potential as a function of the co-ordinate distance, r. We can observe that the potential energy exhibits infinite range for small values of r and an infinite domain for large values of r. We divide the potential energy function with non-identical numerical behavior with respect to the co-ordinate distances into two regions. The cut-off value of r, denoted as *sigma* where the potential energy is zero, is used to differentiate between two regions: region I where $r < sigma$ and region II where $r > sigma$, using a previously described two-region splitting approach [2]. In order to avoid the expensive square root operation on the FPGA following the calculation of squared distances, r^2, we rewrite the potential energy as a function of r^2, thereby effectively precomputing the square root by building it into the interpolation coefficients. We also rescale the potential energy to lie between 0 and 1. We use a quadratic interpolation approach to evaluate the pair-wise potential energies on the FPGA. This means that only the precalculated interpolation coefficients for each region, i.e., region I $\{a, b, c\}$ and region II $\{a, b, c\}$, along with the *sigma* value need to be specified to the HAQMC framework to evaluate the potential energies. Fig. 6 shows the transformed *SAPT2* potential (using

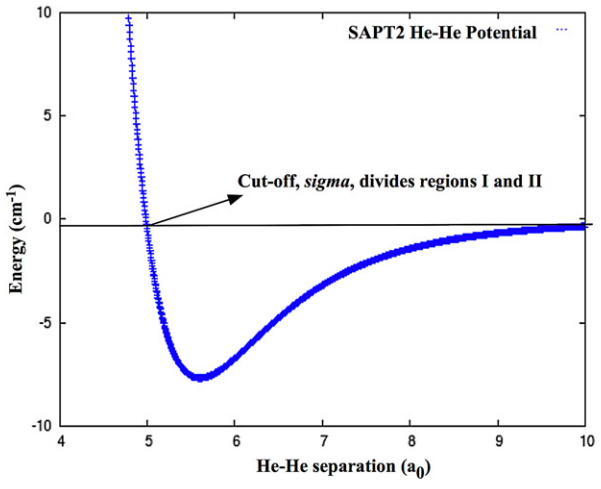

Fig. 5. He–He potential used in our framework.

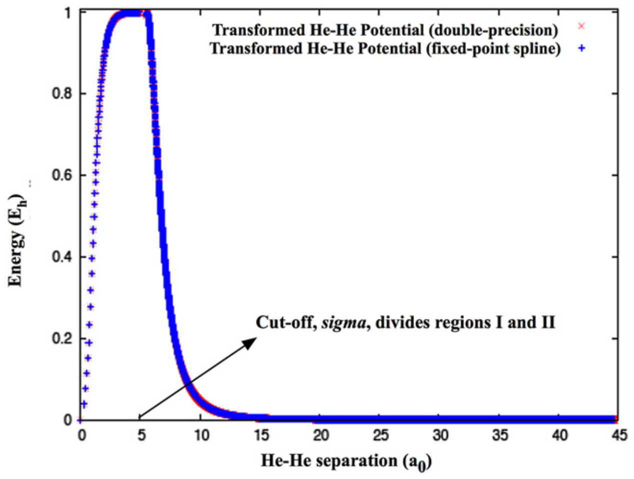

Fig. 6. He–He potential (after transformations and rescaling).

double-precision and fixed-point in software) as a function of the co-ordinate distance between the atoms.

Kernel 2: Wave Function Calculation

We extend the approach discussed for potential energy calculations to offload the trial wave function computations on the FPGA. Here, we provide the analytical function of the trial wave function used in our QMC algorithm. We are interested in the ground-state wave functions of bosonic systems. Wave functions with different parameter values are cited in the literature [17,18]. We implement the wave function proposed in [17]. The wave function is the exponential of the two-body interaction term, $T_2(r)$, which is a function of particle separations as shown in Eq. (11). The two-body term is most commonly sufficient for weakly-bound clusters.

$$\psi(r) = \exp\bigl[T_2(r)\bigr]. \tag{11}$$

We use the analytic form proposed in [17] for ψ which includes the short-range and long-range behaviors and is given by,

$$\psi(r) = \psi_s(r)\psi_l(r), \tag{12}$$

where s and l denote the short- and long-range functions, respectively, shown in Eqs. (13) and (14).

$$\psi_s(r) = \exp\bigl[P(u)\bigr], \quad u = r^{-1},$$

$$P(u) = \sum_{k=0}^{5} a_k u^k, \tag{13}$$

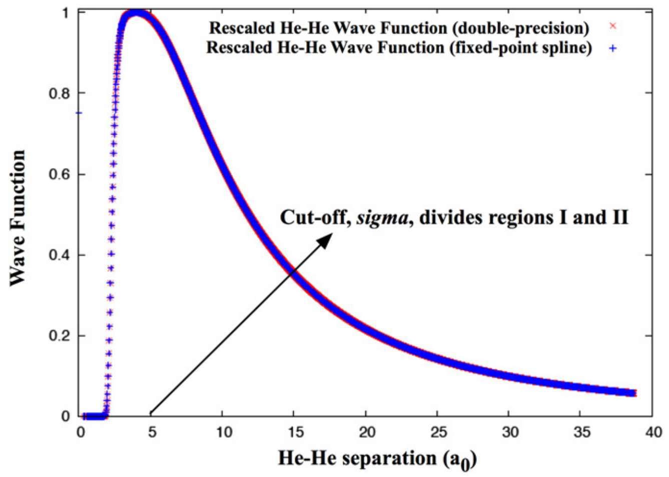

Fig. 7. He–He wave function (after rescaling).

$$\psi_l(r) = r^b \exp\bigl[ar^\alpha\bigr]. \tag{14}$$

Taking the natural logarithm of both sides of the wave function in Eq. (12) and using Eqs. (13) and (14), yields the following:

$$\ln\bigl[\psi(r)\bigr] = \ln\bigl[\psi_s(r)\bigr] + \ln\bigl[\psi_l(r)\bigr], \tag{15}$$

$$\ln\bigl[\psi(r)\bigr] = P(u) + b\ln r + ar^\alpha. \tag{16}$$

Representing the left-hand side of Eq. (16) by $T_2(r)$ yields Eq. (17). We finally express the wave function as the exponential function of $T_2(r)$ given by Eq. (11).

$$T_2(r) = b\ln r + ar^\alpha + \sum_{k=0}^{5} a_k u^k,$$

$$\text{where } u = \frac{1}{r}. \tag{17}$$

Fig. 7 shows the He–He pair-contribution to the trial wave function given by Eq. (11), rescaled to lie between 0 and 1. For the FPGA implementation of this function, we use a two-region splitting approach similar to the one described earlier for $V(r)$ to accurately evaluate the trial wave function for various values of co-ordinate distances, r. The cut-off value of r, denoted *sigma*, where the wave function is maximum divides the two regions, regions I and II. A quadratic interpolation is performed on the rescaled wave function. For the FPGA implementation, we specify *sigma*, Region I $\{a, b, c\}$ coefficients, and Region II $\{a, b, c\}$ coefficients to evaluate the wave function.

Kernel 3: Kinetic Energy Calculation

We further extend the use of our general framework to calculate the sum of the derivatives of the wave function, in three dimensions, which is one of the contributing terms to the total kinetic energy. In Eqs. (18) and (19), we provide the first and second derivatives, namely the gradient and Laplacian functions of the wave function, ψ. The additional contributions to the kinetic energy calculation are from the x-, y- and z-components of the gradient functions, which are calculated in software.

$$\nabla\psi = \frac{d}{dr}\bigl[\exp T_2(r)\bigr]$$

$$= \exp T_2(r)\left[\frac{b}{r} + a\alpha r^{\alpha-1} - \sum_{k=0}^{5} k.a_k u^{k+1}\right],$$

$$\text{where } u = \frac{1}{r}, \tag{18}$$

$$\nabla^2\psi = \frac{d^2}{dr^2}\left[\exp T_2(r)\right]$$
$$= \exp T_2(r)\left[-\frac{b}{r^2} + a(\alpha-1)\alpha r^{\alpha-2} - \sum_{k=0}^{5} k.a_k u^{k+1}\right]$$
$$+ \nabla\psi\left[\frac{b}{r} + a\alpha r^{\alpha-1} - \sum_{k=0}^{5} k.a_k u^{k+1}\right],$$
$$\text{where } u = \frac{1}{r}. \tag{19}$$

Fig. 8 shows the sum of $\nabla^2\psi + 2(\nabla\psi/r)$, chosen for the FPGA implementation rescaled to lie between 0 and 1. For the FPGA implementation, we use the two-region splitting approach as before to accurately evaluate this partial kinetic energy term for various values of co-ordinate distances, r. The cut-off value of r, denoted *sigma*, where this combined function crosses zero and changes sign divides the two regions, regions I and II. As before, a quadratic interpolation is performed on the rescaled wave function. Hence for the FPGA implementation, we specify the *sigma*, Region I $\{a, b, c\}$ coefficients, and Region II $\{a, b, c\}$ coefficients to evaluate this partial kinetic energy term.

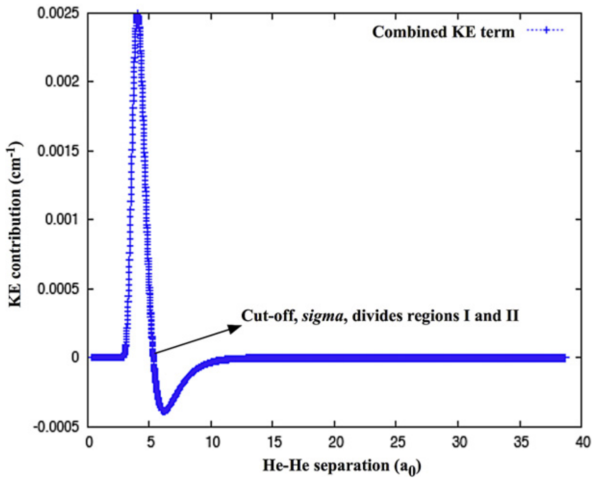

Fig. 8. Combined KE term (*gradient* and *Laplacian* of the wave function).

4. Implementation and results

The QMC application is used to calculate the ground-state energy (i.e., the sum of potential energy and kinetic energy) of a cluster of atoms by sampling a number of configurations and using the wave function to decide whether to accept or reject the configuration. We are able to apply our general interpolation framework to calculate the potential energy, wave function, as well as the sum of the derivatives of the wave function, a contributing term to kinetic energy. The HAQMC framework is targeted to a single core of the dual-core dual-processor 2.2 GHz AMD Opteron with the Xilinx Virtex-4 V4LX160 FPGA. We partition the application so that the shaded blocks in Fig. 4, i.e., the calculation of the kernels is accelerated using the Xilinx Virtex-4 FPGA on the Cray XD1 and the rest of the application runs on the host processor. The function containing the three kernels in the original software application is replaced by the FPGA kernel invocation. We use fixed-point for all our calculations on the FPGA. On the host processor, we perform the following one-time initializations on the FPGA using the Cray XD1 API: load the interpolation coefficients, $\{a, b, c\}$ for regions I and II for the three kernels to the Block RAMs, and (b) load the cut-off values separating the two regions, namely *sigma*, for the three kernels to registers. For every configuration in an iteration, the processor transfers the $\{x, y, z\}$ positions of the current configuration to the embedded Block RAMs using the Cray XD1 API.

Fig. 9 shows the top-level block diagram of our implementation. The FPGA implementation consists of the following blocks: Distance Calculation (*distCalc*), Potential Energy Calculation (*top_PE*), Wave Function Calculation (*top_WF*), and Kinetic Energy Calculation (*top_KE*). We can see that a major portion of our framework is programmable and replicated for all the three kernels (shown as shaded blocks in Fig. 9). Based on whether we employ transformation schemes for our functions, i.e., whether we use an exponential

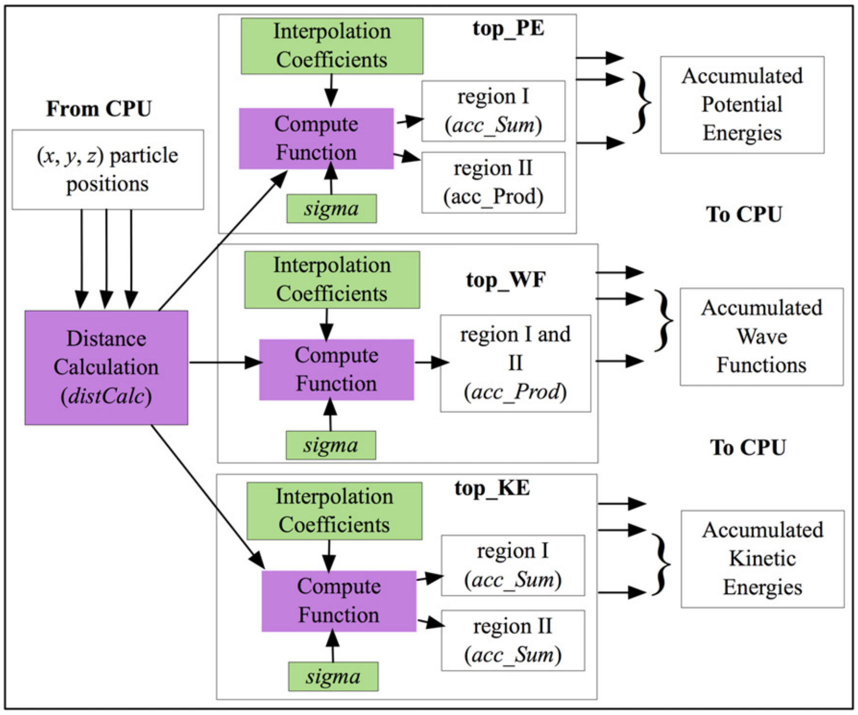
From CPU
(x, y, z) particle positions
Distance Calculation (distCalc)
top_PE
Interpolation Coefficients
Compute Function
sigma
region I (acc_Sum)
region II (acc_Prod)
Accumulated Potential Energies
To CPU
top_WF
Interpolation Coefficients
Compute Function
sigma
region I and II (acc_Prod)
Accumulated Wave Functions
To CPU
top_KE
Interpolation Coefficients
Compute Function
sigma
region I (acc_Sum)
region II (acc_Sum)
Accumulated Kinetic Energies

Fig. 9. Top-level block diagram.

transform or not to limit the range of the function, three accumulator configurations are used. For kernel 1 (potential energy), we form cumulative sums of the pair-wise potential energies from region I (*acc_Sum*) and cumulative products of the potential energies from region II (*acc_Prod*). For kernel 2 (wave function), we form cumulative products of the pair-wise wave functions from both regions I and II (*acc_Prod*). For kernel 3, we form cumulative sums of the pair-wise results from both regions I and II. However, we accumulate these results separately from the two regions due to the non-identical scaling factors in the two regions (*acc_Sum*). The accumulated results are written to registers and the host processor reads these registers upon receiving a control signal from the FPGA indicating that the results are ready. The host processor converts the fixed-point results back to floating-point and removes the rescaling to obtain the floating-point values of the kernels.

Table 1 shows the summary of resources on the Virtex-4 FPGA for kernel 1 alone, kernels 1 and 2, and kernels 1, 2, and 3. Kernel 1 (PE) also includes the resources used for the squared distance calculation. While porting kernels 1 and 2 (PE and WF) and kernels 1, 2, and 3 (PE, WF, and KE term), we are able to share the distance calculation module among the kernels. Column 4 of Table 1 shows that we use roughly 70–80% of the logic slices, DSP multipliers, and Block RAM resources. This means each kernel uses about 25% of the resources. Hence, we port only a part of the KE calculations on to the FPGA and retain the rest of the KE calculations on the host processor, as we cannot fit all the components of the kinetic energy on the present FPGA device.

The software QMC application (in C and compiled using -03 optimization) runs on a single core of the dual-core dual-processor AMD Opteron 2.2 GHz processor. The hardware-accelerated application runs on a single core and offloads the PE, WF, and KE kernels onto a Xilinx Virtex-4 VLX160 FPGA operating at 100 MHz. Table 2 gives the execution times for the simulation with 10 configurations and executed for a total of 400 iterations. The speedup performance for the two potential energies, *SAPT2–He* and *HFDB–He* with the wave function is shown in Fig. 10. In our previous implementation [2], we reported a speedup of 25× for the hardware-accelerated version over the software QMC implementation. We also demonstrated that the fixed-point implementation reproduces the double-precision results over the range of distances that we are interested in, without any loss in accuracy. The speedup reported

Table 1
Summary of resources on the Cray XD1 Xilinx Virtex-4 FPGA V4LX160.

Xilinx FPGA resources (*logic + embedded resources*)	Kernel 1 (PE)	Kernels 1 & 2 (PE and WF)	Kernels 1, 2 & 3 (PE, WF and KE contributing term)
Logic Slices	36%	50%	72%
DSP48s multipliers	26%	52%	78%
18-kbit Block RAMs	31%	51%	70%

Table 2
Execution time (in seconds) of the complete QMC algorithm (10 ensembles, 200+200 iterations).

Implementations	Atoms						
	64	128	256	512	1024	2048	4096
CPU: Kernels 1 and 2 (*HFDB–He* potential)	3.474931	13.94198	55.81017	223.1778	890.9053	3523.418	14015.39
CPU: Kernels 1 and 2 (*SAPT–He* potential)	9.024108	32.42079	130.0553	521.0645	2148.114	8285.269	33185.44
FPGA-accelerated: Kernel 1	4.070662	4.237045	5.397105	9.425678	25.46955	88.98794	341.8981
FPGA-accelerated: Kernel 1 and 2	4.239053	4.534727	5.562009	9.641202	25.66526	89.21081	342.3888

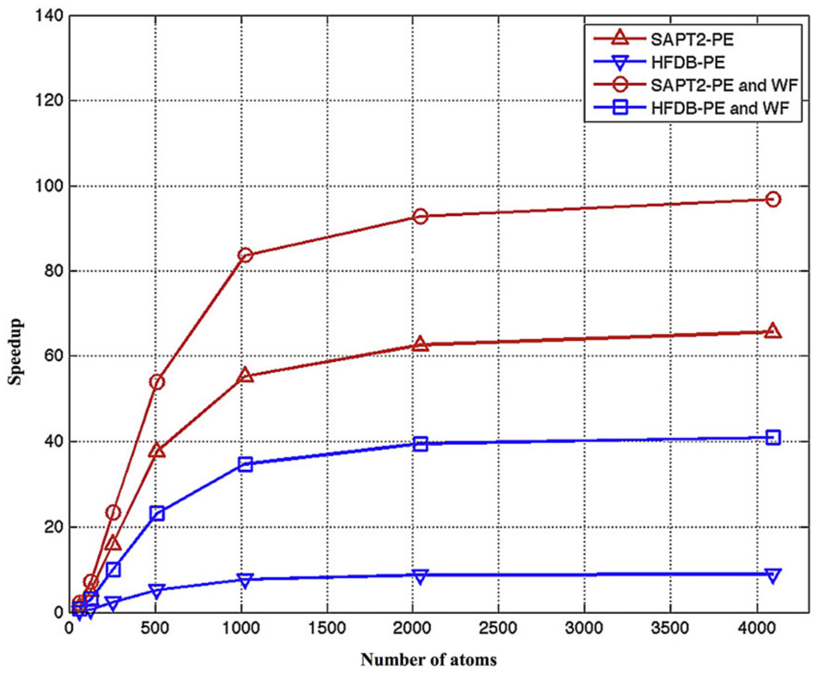

Fig. 10. Speedup performance for kernels 1 and 2 versus the number of atoms.

Table 3
Execution time (in seconds) for a single *compute*() function call on the Cray XD1 HPRC platform.

Kernels	Atoms						
	64	128	256	512	1024	2048	4096
Kernel 1	0.000075	0.000131	0.000373	0.001367	0.005322	0.02108	0.08398
Kernel 1 and 2	0.000137	0.000191	0.000450	0.001450	0.005381	0.021169	0.084141

in [2] is for the calculation of trial wave function and an alternate potential energy function, the *HFDB–He* potential [15], with the FPGA implementation operating at 66 MHz. Here, we use a complicated, and more refined He–He potential [16]. Improvements in the pipelined FPGA architecture allow us to use a clock rate of 100 MHz for our present FPGA implementation. Hence, this implementation shows a speedup approaching 100× for the complete QMC simulation with PE and WF kernels. Inclusion of kernel 3 for kinetic energy calculations, where our approach is to divide the kinetic energy into components that run on the CPU and the FPGA concurrently limits the speedup we can achieve for the overall QMC application. In this case the speedup reduces to 5× over the software implementation.

We provide a preliminary performance analysis for the QMC algorithm without the kinetic energy calculation, as it is not completely ported to the FPGA. An important reason for the improvement in performance for the FPGA design is the use of a deeply pipelined architecture. After a constant number of clock cycles called the pipeline latency, a result is produced every clock cycle. The execution time for the QMC application on the FPGA consists of four parts: initialization time, C_{init}, which accounts for the initializations done once prior to the beginning of the Monte Carlo algorithm including the time for the one-time FPGA configuration. To this, we also include the time to load the interpolation coefficients to the on-chip memories. $C_{sw,i}$, the time taken for the functions performed in software for an iteration, i, $C_{overhead,i}$, which includes the time for input and output data transfer between the Opteron and the FPGA over the RapidArray interconnect for an iteration, i. Finally, the computation time for an iteration, $C_{hw,i}$ is the time taken for the implementation of the concurrent hardware tasks, the energy and wavefunction kernels on the FPGA. The run time for an iteration of the Monte Carlo algorithm is given by Eq. (20).

$$C_i = C_{init} + C_{sw,i} + C_{hw,i} + C_{overhead,i}. \tag{20}$$

The computation time for the kernel implementation on the FPGA for one iteration of the QMC algorithm is approximately given by Eq. (21).

$$C_{hw,i} \approx \frac{L + (N(N-1)/2)}{f}. \tag{21}$$

The numerator of Eq. (21) denotes the number of clock cycles to obtain the results from the energy and wave function calculation modules for a system of N atoms, which have a pipeline depth of L stages and require $N(N-1)/2$ to produce the $O(N^2)$ results and a few clock cycles to produce the accumulated potential energies and wave function values. The pipeline latency in our design, i.e., $L = 58$. f indicates the clock frequency of the implemented FPGA design. We transfer $3N$ 32-bit positions from the SDRAM to on-chip FPGA Block RAM every iteration. The output data consists of only the accumulated potential energies and wave functions, which are written back to registers and read by the processor. We ignore this negligible overhead in our present analysis. The overhead time, $C_{overhead,i}$ is approximated by Eq. (22), where BW is the bandwidth of the RapidArray interconnect between the processor and the FPGA.

$$C_{overhead,i} \approx \frac{3N * 32\ \text{bits}}{BW(\text{bits/sec})}. \tag{22}$$

Table 3 shows the execution time for a *compute*() function call on the FPGA for the PE kernel and for PE and WF kernels respectively. This time includes the components, $C_{hw,i}$ and $C_{overhead,i}$. The times for PE alone in row 2 of Table 3 and PE and WF kernels in row 4 are similar as they are done concurrently in hardware. The execution time predicted by the clock cycle accurate model agrees remarkably with the actual results and can be used to predict the performance of our design as we scale the cluster sizes and on different RC platforms.

Using all the FPGA compute nodes on the Cray XD1 platform, we can distribute the configurations sampled among the compute nodes. We can use a Message Passing Interface (MPI) to utilize multiple nodes of the Cray XD1 platform. There is little communication overhead between the computing nodes, except at the end of the parallel algorithm, when a master processor collects the results from all the processors. Hence, if we have p computing nodes with FPGAs on the XD1, we can expect a speedup approaching $100 * p$ or $40 * p$ while using kernels 1 and 2 for the two potential energy kernels and a speedup of $5 * p$ for the three kernels.

5. Installation and instructions

The HAQMC framework is available as the archive, *haqmc1.0.ar.gz*. Uncompressing the archive results in four subdirectories: *Part1*, *Part2*, *Part3* and *Part4*. *Part 1* of the HAQMC framework contains the software codes used to generate the interpolation coefficients for potential energy, wave function and the kinetic energy calculations. *Part 2* of the HAQMC framework contains the entirely software QMC application in C. This is provided to illustrate the algorithm and used for verification of the results from the hardware-accelerated QMC application and for comparing the execution times of the software versus hardware accelerated QMC application. *Part 3* of the HAQMC framework contains the hardware-accelerated QMC application. In this part of the framework, we replace the software function calls in which we compute the energy and wave function with FPGA function calls. We use a hardware–software partitioning method to offload the computationally intensive functions on the computing node with the FPGA on the *Pacific* Cray XD1 platform and retain the remaining functions on the processor. A small part of the kinetic energy computations is performed on the host processor concurrently with the FPGA execution. *Part 4* of the HAQMC framework contains the source codes for the hardware implementation written using the hardware description language called VHDL.

Part 1: Generating the FPGA interpolation coefficients

Part I of the framework provides the source code initially used to determine the fixed-point representation for the FPGA implementation, for a given functional form of the energy and wave function. We also provide the codes to obtain the interpolation coefficients for the two-regions for the potential energy, kinetic energy and wave function evaluation on the FPGA. Here, we provide the instructions to use this part of the framework for the potential energy calculations. The approach for generating the coefficients for wave function and kinetic energy is identical to that of the potential energy.

The code uses the *SAPT2–He* potential described earlier. This part of the framework approximates the *SAPT2* potential by constructing quadratic splines for the potential energy as a function of squared co-ordinate distance between the atoms. This is done to eliminate the square root operation to obtain the co-ordinate distance following the calculation of the squared distances. We also employ transformation schemes and rescale the function to lie between 0.0 and 1.0 for fixed-point evaluation. The functions *printRegionICoefficients*() and *printRegionIICoefficients*() in "*PotentialEvaluationInterp.cc*" are used to print the interpolation coefficients. The parameter, *sigma* denotes the cut-off value of the squared distance that separates the two regions for potential energy evaluation.

Running *make* followed by *./potential-interp*, gives us the interpolation coefficients. The coefficients are stored in files, {*faI*, *fbI*, *fcI*}*.dat* corresponding to region I coefficients and {*faII*, *fbII*, *fcII*}*.dat* corresponding to region II coefficients respectively. Each of the three files for region I contain 256 coefficients. The three files for region II contain a total of 1344 coefficients. Advanced users can change the analytical function used to realize a different cluster by altering the functional forms for potential energy, wave function and kinetic energy in the files, *potential.h*, *wavefunction.h*, and *kinetic.h*.

Part 2: Software QMC application

This part of the framework consists of three main source files: (a) *VQMC.c* contains the QMC algorithm. The $O(N^2)$ computationally intensive functions (i.e., Step 3 of the algorithm) are contained within the *compute*() function. The rest of the steps (Steps 1, 2 and 4) are performed within the *main*() function. (b) *kernels.h* contains the computationally-intensive functions calculated in Step 3, i.e., the analytical functions for the potential energy, wave function and the first and second derivatives of the wave function. (c) *qmcc.h* contains the simulation settings and parameters. To alter the number of particles, ensembles or iterations, one can vary the parameters in this file. For this part, the Scalable Parallel Random Number Generator (SPRNG) library [12] is used.

Running *make* results in the executable *./vqmc* and the program prints the total average energy (in *microhartree*) averaged over the ensembles and iterations specified in the *qmcc.h* file. Users can change the analytical function used to realize a different cluster by altering the functional forms for potential energy, wave function and kinetic energy in *kernels.h*.

Part 3: Hardware-Accelerated Quantum Monte Carlo (HAQMC)

Part 3 of the framework provides the hardware-accelerated QMC application along with the bitstream that configures the FPGA to calculate the computationally intensive functions on the FPGA. *HAQMC.c* is the source file similar to the *VQMC.c* in Part 2 with the exception that the *compute*() functions are now replaced with the FPGA *compute*() function. Other source files are *hostFunction.c*, which performs a part of the kinetic energy computations on the host processor, *FPGAInit.c*, which is responsible for initializing the sigma and the interpolation coefficients for the FPGA implementation of the functions, and *qmcc.h*, that initializes the simulation settings and parameters. The source files can be directly compiled with the provided Makefile. To use HAQMC, one needs to log on to a computing node of the Cray XD1 with a Xilinx Virtex-4 FPGA. The bitstream is loaded within the program using the Cray XD1 FPGA control functions. To compile and run the source files, run *make* followed by *./haqmc*. This gives the total average energy (in *microhartree*) averaged over the ensembles and iterations specified in the *qmcc.h* file.

We initialize the reference configuration such that the nearest-neighbor distance is 8.0 bohr and we use a $4 \times 4 \times 4$ lattice (64 atoms) in the HAQMC.c provided in the distribution. Referring to the *compute*() function in *HAQMC.c*, first we copy the (x, y, z) co-ordinate positions to the Block RAMs on the FPGA using the Cray XD1 API [7].

The number of particles is stored in the lower 32-bits of the register, USER1_REG.

In the following example, we store 0x00000040, corresponding to 64 atoms.

$$\mathrm{fpga\_wrt\_appif\_val(fp\_id, 0xFFFFFFF100000040UL, USER1\_REG,}$$
$$\mathrm{TYPE\_VAL, \&e);}$$

To simulate a system with 4096 atoms, we would change this to,

$$\mathrm{fpga\_wrt\_appif\_val(fp\_id, 0xFFFFFFF100001000UL, USER1\_REG,}$$
$$\mathrm{TYPE\_VAL, \&e);}$$

The higher 32-bits of the register are used as a control signal to the FPGA to start reading the co-ordinate positions and fill the pipeline. We use the registers to store the results and read the contents within the code, *HAQMC.c* running on the processor. We also reconstruct the floating-point results of the functions back from the fixed-point values. *FPGAInit.c* is used to initialize the block RAMs on the FPGA with the coefficients generated in Part 1 of the framework for the two regions for potential energy, wave function and part of the kinetic energy calculations. We use *USER1_REG* to send control signals to the FPGA to load the respective coefficients for the functions. Registers labeled *USER3_REG*, *USER4_REG*, *USER17_REG* are used to store the cut-off value sigma for the three functions. The registers *USER18_REG*, *USER19_REG* and *USER20_REG* are used to store the inverse of the bin-widths for the region I function evaluation. Unless the user wishes to change the analytical functions, these registers do not need to be modified.

The clock cycle to read the final results is stored in *USER2_REG*, *USER21_REG* and *USER22_REG*. The instructions to modify these values are given in *FPGAInit.c*.

Additionally, we use the following Cray XD1 FPGA control functions [7].

- *fcu -reset* is used to reset the FPGA.
- *fcu -b* is used to create a header file "ufphdr" that is used to convert the binary bit file *.bin* to a loadable ufp, *.ufp* file. We are prompted to enter the Cray Part Number (10 characters) and the clock frequency of the FPGA design in MHz (integer, range 63–199, inclusive). The part number used corresponds to the Xilinx Virtex 4 FPGA part present on the Pacific Cray XD1. The clock frequency used for our implementation is 100 MHz.
- *fcu -c top.bin ufphdr* creates the loadable file *top.bin.ufp*.
- *fcu -l top.bin.ufp* is used to load the bitstream on the Cray XD1 FPGA. To do so, login to the desired node with an FPGA and use this command. This is implemented using an API function within the C program.

Part 4: Source files for FPGA implementation

This part of the HAQMC framework provides the source files (VHDL and edif netlists for the FPGA implementation). We will also provide the Makefile and the scripts that are used to synthesize the design and produce the final configuration bitstream that can be downloaded to the FPGA. The Makefile provided in this part is used to generate the FPGA configuration bitstream, *top.bin*. To run the scripts associated with the Makefile, the installation of Xilinx Integrated Software Environment (ISE) tools are required. These tools are used to synthesize the VHDL design files, place and route the design and create the FPGA binary file that will be downloaded to the FPGA. We use the Xilinx 8.1 ISE/EDK tools for our FPGA development.

This framework presents the following capabilities to the user. Part 1 of the framework can be used to obtain the interpolation coefficients if the user would like to change the analytical functions for the potential energy or wave function as along as the functions can be represented using our two-region approach. Part 3 of the framework gives the capability to vary the number of particles (a number of particles up to 4096 is currently supported by our framework) and repeat the simulations varying the number of configurations and iterations. Using the VHDL sources in Part 4, the user can change the fixed-point representations or additionally fit more functions from the QMC application on a larger FPGA device. The results from this implementation have been compared to the experimental results in [19] which proposes high-quality trial wave functions for helium clusters. We use the proposed trial wave function in [19] and ensure that the ground-state energies obtained using our framework are similar to those proposed in [19] for small helium clusters.

6. Conclusions

We present our Hardware Accelerated Quantum Monte Carlo (HAQMC) framework, which takes advantage of reconfigurable computing in the form of field-programmable gate arrays (FPGAs) along with the general-purpose processor present on the Cray XD1 platform to obtain the total ground-state energy for a cluster of helium atoms. A hardware–software partitioning strategy allows us to accelerate the computationally intensive tasks of the Variational Quantum Monte Carlo algorithm using the FPGA and retain the remaining tasks on the host processor. We use various novel strategies for our FPGA implementation such as deep pipelining, fixed-point evaluation of the functions in hardware, and a two-region splitting approach that is used to approximate the functions accurately on the FPGA. In this paper, we also summarize the analytical functions used to model a cluster of helium atoms. We have categorized the framework into different parts to make it user-friendly and give the user the flexibility to choose the required functionality.

Current extensions of the framework include porting all terms involved in the kinetic energy calculation on the FPGA. We can use our generic HAQMC framework to calculate the other terms required for the kinetic energy. Due to the limitation of the resources on the current target FPGA device, we presently perform part of the computations on the host processor. We are working on optimizing the usage of resources so we can port the additional terms for KE calculations on the FPGA. With this improvement, we would expect a significant improvement in speedup for the entire application.

Acknowledgements

This work was partially 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. We would like to thank Mr. Dave Strenski, Cray Inc., for his timely help in providing us access to the *Pacific* Cray XD1 system and allowing us to share the Cray platform related interfaces in our framework.

References

[1] N. Metropolis, S. Ulam, The Monte Carlo method, J. Amer. Statist. Assoc. 44 (1949).
[2] A. Gothandaraman, G.D. Peterson, G.L. Warren, R.J. Hinde, R.J. Harrison, FPGA acceleration of a quantum Monte Carlo application, Parallel Comput. 34 (2008) 278.
[3] A. DeHon, S. Hauck, Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation, Morgan Kauffman, 2007, p. 908.
[4] K. Compton, S. Hauck, Reconfigurable computing: A survey of systems and software, ACM Comput. Surv. 34 (2002) 171.
[5] Xilinx Virtex-II FPGAs, http://direct.xilinx.com/bvdocs/publications/ds031.pdf,, 2005.
[6] M. Gokhale, P. Graham, Reconfigurable Computing: Accelerating Computation with Field-Programmable Gate Arrays, Springer, 2007, p. 238.
[7] Cray XD1, Cray Inc., http://www.cray.com/products/xd1/index.html.
[8] Cray XT5/XT$_h$, Cray Inc., http://www.cray.com/Products/XT5.aspx.
[9] SRC Computers Inc., http://www.srccomp.com/HardwareSpecs.htm.
[10] SGI Inc., http://www.sgi.com/products/rasc.
[11] DRC Computers Inc., http://www.drccomputer.com/.
[12] R. Baxter, S. Booth, M. Bull, G. Cawood, et al., Maxwell – a 64 FPGA supercomputer, Adaptive Hardware and Systems (2007) 287.
[13] Joseph Thijssen, Computational Physics, Cambridge University Press, 1999, p. 546.
[14] Scalable Parallel Random Number Generator (SPRNG) Library, http://sprng.fsu.edu.
[15] R.A. Aziz, F.R.W. McCourt, C.C.K. Wong, A new determination of the ground-state interatomic potential for He, Mol. Phys. 61 (1987) 1487.
[16] A.R. Janzen, R.A. Aziz, An accurate potential energy curve for helium based on ab initio calculations, J. Chem. Phys. 107 (1997) 914.
[17] R.N. Barnett, K.B. Whaley, Variational and diffusion Monte Carlo techniques for quantum clusters, Phys. Rev. A 47 (1993) 4082.
[18] M.V. Rama Krishna, K.B. Whaley, Wave functions of helium clusters, J. Chem. Phys. 93 (1990) 6738.
[19] R. Guardiola, M. Portesi, J. Navarro, High-quality variational wave functions for small ^{4}He clusters, Phys. Rev. B 60 (1999) 6288.