# Dynamic Scheduling Monte-Carlo Framework for Multi-Accelerator Heterogeneous Clusters

Anson H.T. Tse, David B. Thomas, K.H. Tsoi, Wayne Luk

*Department of Computing*
*Imperial College London, UK*
{htt08,dt10,khtsoi,wl}@doc.ic.ac.uk

***Abstract*—Monte-Carlo (MC) simulation is an effective tool for solving complex problems such as many-body simulation, exotic option pricing and partial differential equation solving. The huge amount of computation in MC makes it a good candidate for acceleration using hardware and distributed computing platforms. In this work, we propose a novel MC simulation framework suitable for a wide range of problems. This framework enables different hardware accelerators in a multi-accelerator heterogeneous cluster to work collaboratively on a single application. It also provides extension interfaces to adaptively balance the workloads according to the cluster status. Two applications are built using this framework to demonstrate its capability and flexibility. A cluster with 8 Virtex-5 xc5vlx330t FPGAs and 8 Tesla C1060 GPUs using the proposed framework provides 44 times speedup and 19.6 times improved energy efficiency over a cluster with 16 AMD Phenom 9650 quad-core 2.4GHz CPUs for the GARCH asset simulation application. The Efficient Allocation Line (EAL) is proposed for determining the most efficient allocation of accelerators for either performance or energy consumption.**

## I. Introduction

To increase the raw computation capacity of a system, one can increase the computational power of the processing unit, or increase the number of processing units. Domain specific processors with specialized instructions or logic blocks usually outperform traditional CPUs due to their more efficient use of silicon area and higher hardware parallelism. So it is common to see Digital Signal Processing (DSP) chips and Field Programmable Gate Array (FPGA) devices used as accelerating co-processors in high performance computing (HPC) systems. Due to recent advances in programming environments, Graphics Processing Units (GPUs) are also attractive to be used as accelerators when building supercomputing systems.

Techniques from distributed computing have been a solution for HPC for many years. The computation task in an application is decomposed into smaller tasks which are performed by computing nodes which communicate through a network. While it is intuitive to combine these two methods to further improve the performance of the system, there are still some key challenges when building practical applications for a multi-accelerator heterogeneous cluster.

The first challenge is the difference in programming models and difference in tools between conventional software programming and these hardware accelerators. Having different types of accelerators within the system makes the situation even more complex as they communicate with the CPU in different ways. This complicated application structure and the high non-recurring engineering (NRE) cost per application become the major barriers when utilizing heterogeneous clusters.

The second challenge is the different types of hardware accelerators are usually customized for specific computation and communication patterns. Thus the performance of them will vary from application to application. Some accelerators may outperform others in computational speed while some accelerators may consume less energy. How to efficiently schedule the tasks for different accelerators is one of the challenging problems. On top of this is the synchronization and data transfer overhead, which increases the uncertainty of the overall achievable performance.

The third challenge for a distributed HPC system is to distribute tasks efficiently. The overhead will become dominant as the number of tasks increases, according to the law of diminishing returns. The communication between distributed tasks will contribute to the overhead. This suggests that applications with a large number of divisible tasks, and a small number of inter-task communications will benefit the most.

Focusing our research on the Monte-Carlo simulation problems enables a better system optimization in a domain specific way. Therefore, we address the above challenges by designing a versatile distributed framework on the heterogeneous cluster architecture targeted in Monte-Carlo problem domain. The main contributions of this work include:

- A scalable distributed Monte-Carlo framework for multi-accelerator heterogeneous clusters is proposed. In this framework, various computational units including CPUs, GPUs and FPGAs work collaboratively to share the workload in the simulation process.
- Various load balancing schemes are modeled and evaluated for the proposed framework. Dynamic runtime scheduling is enabled to improve the utilization efficiency of all available computing resources in the system.
- Two applications are developed and mapped in the proposed framework. The performance of different dynamic scheduling policies in these practical examples is evaluated. The speed and energy consumption trade-off of different accelerator allocations is discussed and analyzed with the Efficient Allocation Line (EAL) approach.

In the paper, Section II presents previous work in the related systems and applications. Then Section III explains the details of our proposed distributed MC framework. Section IV

presents the models and implementations of different dynamic scheduling policies. Section V presents the implementation details of two applications (Asian-Option pricing and GARCH asset simulation) using the proposed framework. Section VI evaluates the measured results of these two applications running on a cluster of accelerated computers. Different dynamic scheduling policies are compared and the speed and energy consumption trade-off between different accelerator allocation policies is discussed. Finally, Section VII summarizes our achievements and future work.

## II. Related Work

Systems with FPGA devices as accelerators have been studied and developed in both academic and industrial fields. In 2004, the Cray XD1 computer [1] achieved 58 GFlops with 12 Opteron CPUs and 6 Xilinx Virtex-II FPGA devices on a single motherboard. In 2007, a cluster with 64 Virtex-4 FPGA devices was built in the Maxwell project [2]. Each FPGA in the Maxwell cluster can achieve up to 2.5 times speedup in a face recognition application, and over 300 times speedup in a Monte-Carlo option pricing application when compared with the software implementations.

GPU computing platforms often have a larger number of floating point units and higher operating frequency than FPGA devices can support. The programming interface of GPU devices, such as CUDA [3], also helps to promote their popularity in HPC systems. In 2008, an updated version of the TSUBAME (Tokyo-tech Supercomputer and Ubiquitously Accessible Mass-storage Environment) system [4], achieved 56.43 TFlops for solving dense linear equations. In addition to custom vector processors, this supercomputer is also equipped with 170 nVidia Tesla C1070 cards.

In 2009, the Quadro Plex (QP) Cluster [5] was built by NCSA in UIUC. For each of the 16 nodes in the QP prototype, there are two AMD Opteron CPUs, four nVidia G80GL GPUs and one Xilinx Virtex-4 LX100 FPGA. This system may achieve 23 TFlops (single precision) theoretically. In 2010, the Axel cluster [6] from Imperial College London demonstrated the collaboration between heterogeneous accelerators. With Xilinx Virtex-5 FPGAs and nVidia C1060 GPUs working together, this 16-node cluster achieved over 22 times speedup in a N-body simulation application over a 16-node CPUs only cluster.

Financial applications in banks usually require supercomputing power for either processing huge amount of raw data or simulating pricing models repeatedly. FPGA platforms have been used to accelerated the Monte-Carlo simulation for financial instruments [7], [8]. In [9] the authors studied the performance of a GPU cluster, which is 2.8 times faster and consumes 28.3 times less energy than a CPU cluster.

## III. Heterogeneous Framework

Flexibility for the application programmers; scalability of the framework; and efficiency of the resource utilization are the three major concerns of our distributed framework. Therefore, we designed our heterogeneous Monte-Carlo framework without creating another layer in programming language level and without altering the original tool chains of each type of accelerators in order to provide the flexibility for the application programmer. The framework provides a unified hierarchical model such that a Monte-Carlo simulation is divided into sub-tasks and distributed to the lower layers recursively. It is highly scalable as a simulation task can be distributed across different accelerators in a single server node, across different server nodes in a cluster or even across several heterogeneous clusters. Extensible dynamic scheduling policies can be designed in the distributor processes such that the sub-tasks can be allocated to the worker process based on the computational performance or even energy consumption.

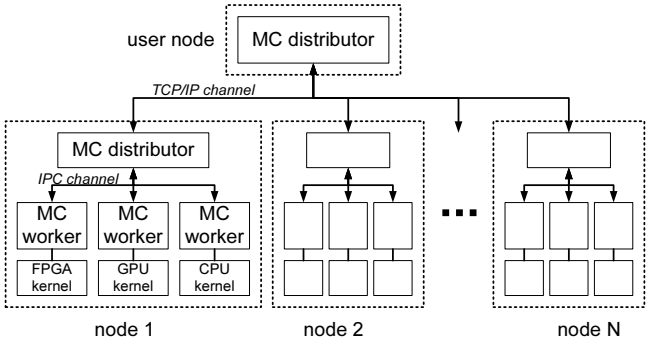


Fig. 1. The overall framework.

### A. Overall hierarchy

The overall framework for distributed Monte-Carlo simulation on a multi-accelerator heterogeneous cluster is shown in Fig. 1. There are two major processes in this framework: MC distributors and MC workers. MC distributors wait for the Monte-Carlo parameters and task size as a form of MC request from their parent MC distributor or from the user. The MC distributor then partitions the task and distributes the sub-tasks to their child MC distributors or MC workers. No simulation is done on the MC distributor. Their functionality is implied by their name – to distribute the Monte-Carlo simulation tasks to their connecting child processes. Each MC worker is responsible for the execution of part of the simulation. They pass the simulation parameters to the underlying "kernel" and get the partial simulation result back from it. In a multi-accelerator environment, each "kernel" holds a specific computational hardware resource such as FPGA, GPU or CPUs.

Fig. 1 only shows a two layer MC distributor network. In fact, the framework is highly scalable since there could be more than two and no upper limit for the number of layers of MC distributors. Additional layers of MC distributors could be inserted between the user node and the cluster. For example, when there are 3 heterogeneous clusters (A,B,C) from different organizations, they could collaborate by inserting a layer of 3 MC distributors namely DA, DB and DC. The MC distributer at the user node distributes the sub-tasks to DA, DB and DC. DA then further partitions and distributes the sub-tasks to the MC distributors of the nodes in cluster A. Similarly for DB and DC.

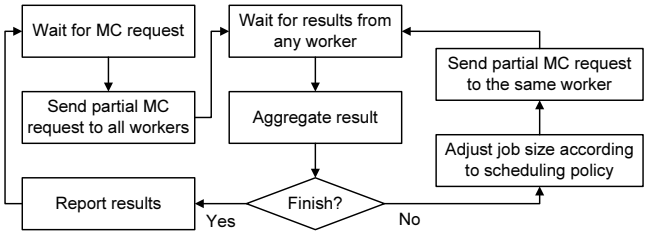


Fig. 2. The work flow of MC distributors.

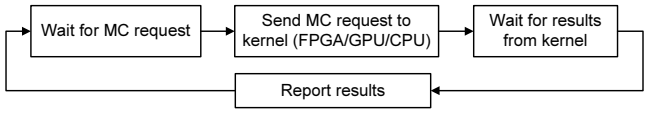


Fig. 3. The work flow of MC workers.

### B. MC processes

The MC workers are the main simulation units. Fig. 3 shows the work flow of the MC worker. The MC workers wait for the MC request (MC parameters and the task size), then forward the MC request to their computation hardware (FPGA, GPU or CPUs) and execute the kernel via the hardware driver. When the computation results are returned, the MC workers report them to their parent MC distributor. The reported results include the aggregated simulation results and the actual completed task size by the kernel. The actual completed task size could differ from the MC request due to hardware specific constraints (e.g. number of cores and memory limit).

The MC distributors are key elements in the distributed Monte-Carlo framework. The work flow of the MC distributor is shown in Fig. 2. The MC distributors wait for the MC request from their parent process or user input, then they partition the MC request to several sub-tasks based on the scheduling policy. The partial MC requests for those sub-tasks are then sent to the child MC distributors or MC workers. When one of the child processes reports the partial results, the MC distributors aggregate the results until the task is completed (the sum of reported task completion size = the required task size in the MC request). When the task is not completed yet, the MC distributor adjusts the sub-task size for the reporting process according to the scheduling policy. Another partial MC request is sent to the reporting process with an updated sub-task size. When the task is completed, the MC distributors report the aggregated result to the parent process (or user). The "task size" discussed here can be the number of simulations, the number of particles or any form of computation tasks of that particular Monte-Carlo simulation.

The intra-node communication between MC distributor and MC workers within the same node is realized by interprocess communication channel (IPC). The inter-node communication between MC distributors of different nodes is realized by the TCP/IP channel with MPI as the session layer.

## IV. Scheduling Policies

In a multi-accelerator heterogeneous cluster, the computational performance differs between different nodes and between different accelerators of the same node as well. Improper task distribution could lead to a drastic performance reduction.

For example, consider a node consisting of one FPGA and one CPU, and the processing speed of FPGA and CPU is 1000 simulations per second and one simulation per second respectively. If 2000 simulations are required and the MC distributor simply distributes 1000 simulations to the MC worker of FPGA and CPU equally at the beginning, the total execution time will be 1000 seconds and the FPGA will be idle for 999 seconds. Such inefficient task allocation leads to poor performance and imbalanced resource utilization. In contrast, if the MC distributor distributes one simulation to both MC workers and distribute another one simulation to them after they reported the result, the execution time is around 2 seconds computation time plus a large amount of message passing overhead and latency between hardware and software.

For the above simple example, one may be able to determine the "optimal task distribution" by pilot running the simulation in each of the devices and distribute the tasks according to their computational performance (1000:1 in this case) provided that the computational time is deterministic for each accelerator. However, such deterministic assumption is often invalid as many Monte-Carlo simulation problems involve non-deterministic run-time (such as solving PDEs). The computation performance for some devices (such as CPU) also depends heavily on the server status.

Therefore, the scheduling policy is a critical factor for the collaborative computing performance in a multi-accelerator heterogeneous cluster. Our solution involves introducing one static and two dynamic scheduling policies. The dynamic scheduling policies enable the task size allocated to the child processes to grow adaptively according to their performance. The performance evaluation of these policies will be discussed in Section VI. The initial task size for all child processes is defined as $TS_{init}$. The task size for child $i$ at the $j$th time of simulation is defined as $TS_j^i$. Therefore, $TS_1^i = TS_{init}$ for all $i$. The remaining uncompleted task size of the MC distributor is defined as $R_d$.

### A. Constant-Size policy

The Constant-Size scheduling policy is the simplest form of static scheduling policy in which the task size stays constant for each child at all times. The Constant-Size scheduling policy is defined as:

$$TS_{j+1}^i = \min(TS_j^i, R_d) \tag{1}$$

The number of $TS_{init}$ ($TS_1^i$) is critical for constant-size scheduling policy. A small value of $TS_{init}$ might cause a large amount of message passing overhead. A large value of $TS_{init}$ might cause the slowest MC worker to affect the overall computation performance.

### B. Linear-Incremental policy

The Linear-Incremental scheduling policy is defined as:

$$TS^i_{j+1} = \min((TS^i_j + c), R_d), \tag{2}$$

where $c$ is a constant. It is a dynamic scheduling policy which increases the task size of the MC worker linearly. Eventually, the task size allocated for the faster child $TS^{i1}_{j1}$ is larger than the slower child $TS^{i2}_{j2}$ as $j1 > j2$. The task size allocated to each child will grow proportionally to their corresponding processing rate slowly.

### C. Exponential-Incremental policy

The Exponential-Incremental scheduling policy is defined as:

$$TS^i_{j+1} = \min((TS^i_j \times m), R_d), \tag{3}$$

where $m$ is a constant. This dynamic scheduling policy increases the task size of the MC worker exponentially with a factor of $m$. Similar to Linear-Incremental policy, the task size allocated for the faster child will be larger than the slower child after a period of time. The task size allocated to the child processed will grow proportionally to their processing rate at a much faster rate.

### D. Other possible policies

Apart from the basic scheduling policies stated above, we can also employ a mixed scheduling policy, such as using Linear-Incremental policy at the beginning and then change the policy to Constant-Size after certain iteration. The scheduling policy in this framework is highly flexible and can be optimized for any goal. It is up to the application engineer to design their own scheduling policies for their own target. For example, If the energy usage of the MC worker can be profiled and fed back to the MC distributor, an Energy-Equal scheduling policy can be defined such that each MC worker consumes the same amount of computational energy. An energy-efficient MC worker keeps computing most of the time, while a less energy-efficient MC worker will be idle occasionally to keep the same amount of energy usage. The idle accelerators can therefore be used in another application.

## V. Applications

We have implemented two applications in our proposed framework, namely

- Asian option pricing using control variate method,
- GARCH asset simulation.

### A. Asian option pricing using control variate method

Asian options are a kind of *derivative*: a financial instrument whose value is dependent on the price of some underlying asset such as a bond or stock. Unlike simpler options, which provide a payoff depending on the instantaneous price of the underlying, arithmetic Asian options provide a payoff depending on the arithmetic average price of the underlying during the option life-time. This averaging makes arithmetic Asian options cheaper and less sensitive to market manipulation, but also means there is no closed-form solution for the pricing.

Monte-Carlo methods provide an accurate way to price Asian options, but have slow convergence, so a huge number of simulations is needed. Therefore, arithmetic Asian options are perfect candidates to be priced using a multi-accelerator heterogeneous cluster.

The *payoff* of an arithmetic Asian call option is:

$$\textit{payoff} = \max\left(0, \frac{1}{n+1}\sum_{i=0}^{n} S_i - K\right). \tag{4}$$

where $S_i$ is the asset price at time $i$, $K$ is the exercise price and $n$ is the number of steps. The estimated *payoff* is calculated using Monte-Carlo simulation.

The control variate method is a variance reduction technique which estimates the target value $x$ using a control variable $y$. The variable $\bar{y}$ is computed using the same set of random data used in the computation of $\bar{x}$. The true expected value of $E(y)$ should be calculable using a closed-form solution. The estimated value of $x$ is adjusted by the difference between true value of $E(y)$ and estimated $\bar{y}$.

Therefore, apart from simulating the *payoff* of the Asian option, the payoff of a correlated European option is also simulated at the same time using the control variate Monte-Carlo (CVMC) algorithm [10], [11].

### B. GARCH asset simulation

Financial equations are often based on many assumptions. The most famous Black-Scholes equation relies on a constant volatility assumption [12]. In fact, it is well-known that the volatility is stochastic. Monte-Carlo simulation is the only way to evaluate the financial derivatives when stochastic volatility is taken into account. One of the most commonly used stochastic volatility models is Generalized Autoregressive Conditional Heteroskedasticity (GARCH) models [13]. One of the GARCH models commonly used is GARCH (1,1) where the volatility ($\sigma$) and the asset price ($\upsilon$) is given by the following equations:

$$\sigma_i = \sqrt{a_0 + a_1\epsilon^2_{i-1} + a_2\sigma^2_{i-1}} \tag{5}$$

$$\epsilon_i = \sigma_i \times W \tag{6}$$

$$\upsilon_i = \upsilon_{i-1} + \mu + \epsilon_i \tag{7}$$

The $W$ is a Gaussian random number. Parameters $a_0, a_1$ and $a_2$ are usually determined empirically using maximum likelihood methods.

### C. FPGA kernels

For both applications, we design the FPGA kernels as shown in Fig. 4. There are two main types of components in the design: one or more identical Monte-Carlo cores, and a single shared Coordination Block (CB). The MC cores contain a Gaussian random number generator (GRNG) core and a simulation core. The GRNG uses the piecewise linear generation method [14] which produces a stream of 24-bit fixed-point random numbers, with a period of $2^{128}$. The quality

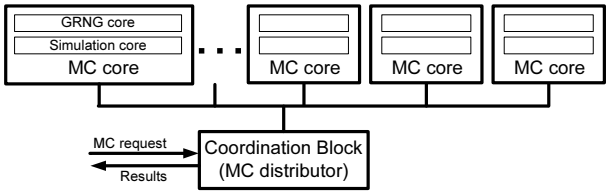


Fig. 4. The hardware design of FPGA kernel.

of the stream has been checked with the Chi-squared test for sample sizes up to $2^{32}$, and shows no significant deviation from the Gaussian distribution.

The MC core in our Asian option pricing application is capable of generating random asset price paths, calculating payoffs of the Asian option and European option, and accumulating the payoffs and payoffs related statistical result. In other words, each MC core is capable of executing the MC part of CVMC Algorithm. Multiple identical MC cores are instantiated to make the maximum use of the device. The required number of simulations is distributed equally to each MC core.

The MC core in our GARCH asset simulation application is responsible for the generation of random numbers, simulation of the stochastic volatility movement, and simulation of the asset movement with respect to the volatility.

The Coordination Block (CB) manages the MC cores, allowing them to work in parallel to price the same option. The CB is also responsible for communicating with the host by accepting MC request and reporting MC results. The Gaussian random number generators in MC cores are also initialized by the CB. Different sequences of bits are connected to different Gaussian random number generators as the random seeds. The CB can also be viewed as a MC distributor employing Constant-Size scheduling policy. Constant-Size scheduling policy is the best choice as all MC cores finish the computation in the exact same cycle.

The hardware architecture of the simulation core for Asian option pricing is shown in Fig. 5. The dynamic input parameter is the Gaussian random number $W$ generated by GRNG. There are many pipelined loops in the hardware. The number of pipelined stages must be identical for all the pipelined loops in order to guarantee a consistent computation schedule. Let $p$ be the maximum number of pipeline stages for these pipelined loops. Pipelined registers are added to ensure the number of pipelined stages of all loops equal to $p$. As the feedback result will reappear only after $p$ stages, we simulate a batch of $p$ paths at the same time by interleaving the computations. The grey boxes in Fig. 5 indicate the pipelined registers.

The hardware architecture of the simulation core for GARCH asset simulation is shown in Fig. 6. The grey boxes in Fig. 6 indicates the pipelined registers inserted to balance the number of pipeline stages for all feedback updating loops for the stochastic volatility and asset prices.

Our FPGA kernels target a Xilinx xc5vlx330t FPGA chip on an Alpha Data ADM-XRC-5T2 card, which contains 51,840 slices, 192 DSP48E and 324 BlockRAM units. We design

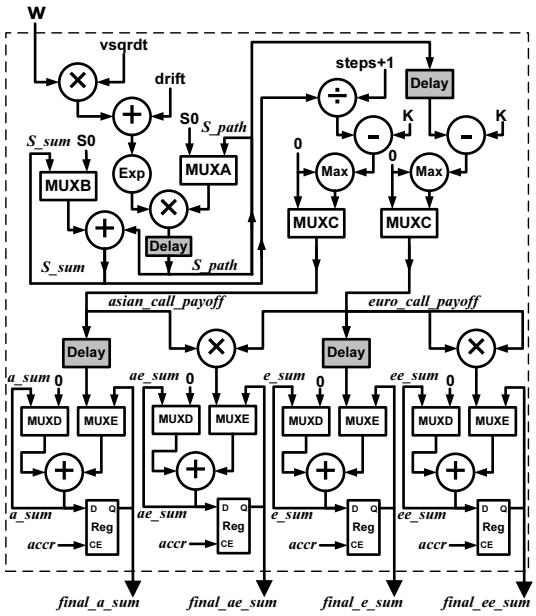


Fig. 5. The hardware architecture of Asian option pricing simulation core.

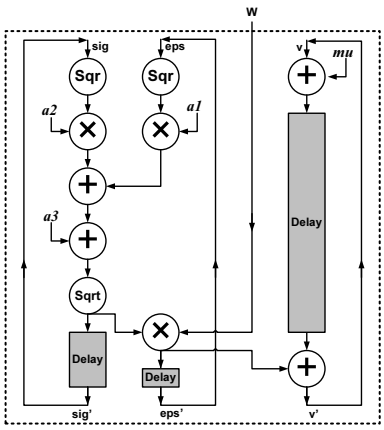


Fig. 6. The hardware architecture of GARCH asset simulation core.

our hardware architectures for both applications manually in VHDL to maximize performance. The design is synthesized, mapped, placed and routed using Xilinx ISE 10.1.03. Single precision floating point arithmetic is used. The number of MC cores for Asian option pricing is 10. The number of MC cores for GARCH asset simulation is 12. The summary of resource consumption for both applications is shown in Table I.

### D. GPU kernels

Graphics Processing Units (GPUs) have been used for acceleration in many application domains. They are Single Instruction Multiple Data (SIMD) computing devices. Parallelizable tasks are executed on the GPU as a "kernel" by a computation grid. The "kernel" is executed by all threads in parallel with the same code, but on different sets of data.

The co-processing flow of GPUs provides a good match to our design framework. The MC request containing MC parameters and task size is firstly copied to the GPU data

TABLE I
XC5VLX330T FPGA RESOURCE CONSUMPTION

| | Asian option pricing | | GARCH simulation | |
|---|---|---|---|---|
| MC Cores | 10 | | 12 | |
| **Resource** | **Used** | **%** | **Used** | **%** |
| Slices | 44,118 | 85% | 37,205 | 71% |
| FFs | 130,195 | 62% | 118,261 | 57% |
| LUTs | 79,587 | 38% | 59,313 | 28% |
| RAM | 10 | 3% | 12 | 3% |
| DSP48Es | 180 | 93% | 192 | 100% |

memory. The MC results are copied back to the memory of the MC worker after the execution of the GPU kernel.

We design our CUDA kernels for Asian options pricing and GARCH asset simulation using two procedures, namely Gaussian random number generator procedure, and a path simulation procedure.

In the Gaussian random number generator procedure, uniform random numbers are first generated and stored in the GPU's global memory space using the Mersenne Twister algorithm in parallel with all threads. Then the uniform random numbers are transformed into Gaussian random numbers using the Box-Muller method [15]. The memory space for storing Gaussian random numbers is allocated by the MC worker once at the beginning. In our target implementation on nVidia Tesla C1060, 2GBytes are allocated, which can accommodate 512MBytes of single-precision Gaussian random number. Such memory constraints may lead to the completed number of simulations to be less than the requested number of simulations, which will be notified by the MC worker to its parent.

In the path simulation procedure of Asian option pricing, each thread simulates the price movement path as in the CVMC Algorithm and computes the Asian and European option result in the shared memory. In the path simulation procedure of GARCH asset simulation, each thread simulates the volatility dynamics as in Equation 5 and updates the asset price accordingly.

### *E. CPU kernels*

In both applications, we implement the CPU kernels in the C language and use the Intel Math Kernel Library (MKL) for the random number generation. The Mersenne Twister algorithm is used as the random number base and Box-Muller is used for Gaussian number transformation. The code is compiled with Intel compiler (icc) 11.1 with -O3 maximum speed optimization options and SSE enabled. OpenMP is used to parallelize the computation with the multi-core capabilities of CPUs. The parallel FOR `#pragma` directive is used to parallelize the main loop, so that loop iterations can be executed in parallel using multiple CPUs.

## VI. PERFORMANCE EVALUATION

In this section, we evaluate the results of the two applications used in our framework. We investigate the effect of different dynamic scheduling policies on the computational performance using the Asian options pricing engine in Section VI-A. The performance, energy consumption and efficient accelerator allocation will be discussed using the GARCH asset simulation example in Section VI-B. We carry out our experiments on an accelerator cluster, which consists of 8 server nodes. Each server node consists of two AMD Phenom 9650 Quad-Core 2.3GHz CPUs, one nVidia Tesla C1060 GPU and one Xilinx Virtex-5 xc5vlx330t FPGA.

TABLE II
PERFORMANCE OF ASIAN OPTION PRICING

| | FPGA | GPU | CPUs | Collarboration |
|---|---|---|---|---|
| Brand | Xilinx | Nvidia | AMD | - |
| Type | Virtex-5 | Tesla | Phenom | - |
| Model | xc5vlx330t | C1060 | 9650 | - |
| Freq. | 200MHz | 1.3GHz | 2.3GHz | - |
| Qty | 1 | 1 | 2 | 1 + 1 + 2 |
| Time | 18.3s | 25.5s | 399.6s | 11.8s |
| Speedup | 21.8x | 15.7x | 1x | 33.8x |

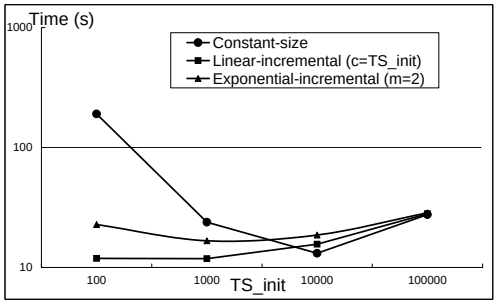


Fig. 7. The performance comparison for different scheduling policies.

### *A. Dynamic scheduling analysis of a single node*

The performance of different accelerator combinations for the pricing of an Asian call option is studied. The computational and load-balancing performance of different dynamic scheduling policies is also presented. We choose a 10-year arithmetic Asian call option with parameters $S_0 = 100$, $K = 105$, $v = 0.15$, $r = 0.1$, $T = 10$ and $steps = 365$. The number of Monte-Carlo simulations is 10,000,000.

The performance comparison for the pricing of Asian option with individual accelerators and multi-accelerator collaboration is shown in Table II. The optimized multi-threaded CPU kernel executed by two AMD Phenom 9650 quad-core CPUs is used as the comparison reference. It can be seen that a speedup of 21.8 times is achieved by the xc5vlx330t FPGA. For the GPU, a speedup of 15.7 times is achieved by the Tesla C1060. For the collaboration with FPGA, GPU and 2 CPUs, a speedup of 33.8 times is achieved using Linear-Incremental policy with $TS_{init} = 1000$.

The collaborative computation time results of using FPGA, GPU and CPU kernels in one node with different scheduling policies are shown in Fig. 7. The Constant-Size, Linear-Incremental and Exponential-Incremental policies are used in the MC distributor with different $TS_{init}$ values. From the figure, when $TS_{init}$ is small, the Constant-Size policy suffers from large overhead and thus long computation time. For large $TS_{init}$, all policies suffer from reduced performance due to the waiting of the completion of the slowest kernel. The shortest computation time achieved is 11.8 seconds when

TABLE III
PERFORMANCE OF THE GARCH ASSET SIMULATION OF DIFFERENT ACCELERATORS AND NUMBER OF COLLABORATIVE NODES

| **Using 2 CPUs per node only** | | | | |
|---|---|---|---|---|
| **Number of nodes** | 1 | 2 | 4 | 8 |
| **Time (ms)** | 1,162,725 | 660,687 | 360,129 | 162,018 |
| **APCC (W)** | 49 | 78 | 146 | 300 |
| **AECC (J)** | 56,973.53 | 51,533.58 | 52,587.83 | 48,605.40 |
| **Using FPGA per node only** | | | | |
| **Number of nodes** | 1 | 2 | 4 | 8 |
| **Time (ms)** | 38,969 | 19,691 | 10,458 | 5,418 |
| **APCC (W)** | 5 | 11 | 21 | 43 |
| **AECC (J)** | 194.85 | 216.60 | 219.62 | 232.97 |
| **Using GPU per node only** | | | | |
| **Number of nodes** | 1 | 2 | 4 | 8 |
| **Time (ms)** | 64,299 | 32,308 | 16,310 | 8,252 |
| **APCC (W)** | 97 | 192 | 350 | 676 |
| **AECC (J)** | 6237.00 | 6203.14 | 5708.50 | 5578.35 |
| **Using FPGA and GPU per node** | | | | |
| **Number of nodes** | 1 | 2 | 4 | 8 |
| **Time (ms)** | 24,706 | 12,822 | 6,825 | 3,636 |
| **APCC (W)** | 102 | 203 | 392 | 683 |
| **AECC (J)** | 2520.01 | 2602.86 | 2675.40 | 2483.40 |
| **Using both FPGA, GPU and 2 CPUs per node** | | | | |
| **Number of nodes** | 1 | 2 | 4 | 8 |
| **Time (ms)** | 24,595 | 12,884 | 7,167 | 4,391 |
| **APCC (W)** | 130 | 270 | 506 | 908 |
| **AECC (J)** | 3197.35 | 3478.68 | 3626.50 | 3987.03 |

Linear-Incremental policy is used with $TS_{init} = 1000$, and it is used as the result for Table II.

In this Asian option pricing application, maximum performance is achieved when $TS_{init} = 1000$ under Linear-Incremental policy. However, other applications may achieve the maximum performance under different policy and with different variables. It is because each application has its particular set of parameters, and different communication overhead. Fig. 7 is just a demonstration of how the performance can differ with different starting task size under different dynamic scheduling policies.

### *B. Performance, energy and efficiency analysis of accelerator allocation of a cluster*

Acceleration performance versus energy consumption is an important factor when considering the efficiency of an accelerator. As a result, it is also one of the main concerns in our evaluation of the proposed framework and we use the GARCH asset simulation application for the evaluation. We study 5 different methods for allocating computational devices in the cluster for collaborative computation:

- CPUs only: use two Phenom CPUs in each node
- FPGA only: use one xc5vlx330t FPGA in each node
- GPU only: use one Tesla C1060 GPU in each node
- FPGA and GPU: use one xc5vlx330t FPGA and one Tesla C1060 GPU together in each node
- FPGA, GPU and CPUs: use one xc5vls330t, one Tesla C1060 and two Phenom CPUs together in each node

We measure the additional power consumption for computation (APCC) with a power monitor. APCC is defined as the power usage during the computation time (run-time power) minus the power usage at idle time (static power). The static power of each cluster node is approximately 210W. In other words, APCC is the dynamic power consumption for that particular computation. The additional energy consumption for computation (AECC) is defined by the following equation:

$$AECC = APCC \times Total\ Computational\ Time. \quad (8)$$

Therefore, AECC measures the actual additional energy consumed for that particular computation.

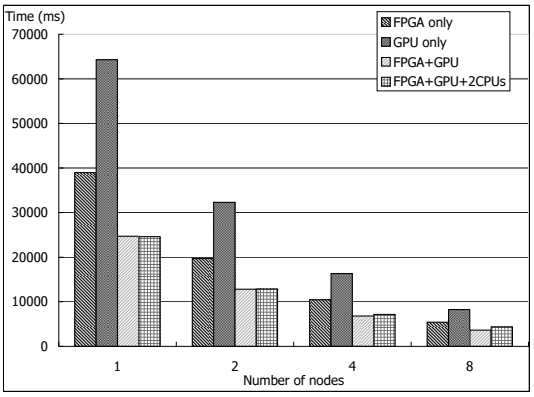


Fig. 8. The computation time of GARCH asset simulation.

The speed and power consumption of the GARCH asset simulation for different accelerator combination in the multi-accelerator cluster is studied. The number of Monte-Carlo simulation is 100,000,000 and one asset is simulated. Linear-Incremental scheduling policy is employed on each MC distributor of the cluster node with $TS_{init} = 1000$. Constant-Size scheduling policy is employed at the higher level MC distributor in the user node with $TS_{init} = 100M$, $50M$, $25M$ and $12.5M$ for a cluster with 1, 2, 4 and 8 nodes. The computation time, APCC and AECC results are shown in Table III.

As expected, an increase in the number of active nodes generally decreases the time for computation. From the results, we can see that the cluster using 8 FPGAs and 8 GPUs is the fastest (3.6s) even when compared with the cluster using all 8 FPGAs, 8 GPUs and 16 CPUs. We believe that the use of the CPUs decreases the response time for the MC distributor and MC worker processes. Therefore, the computational performance gain of using CPUs is offset by the decrease in response time of MC processes, reducing the overall performance. The cluster using 8 xc5vlx330t FPGAs and 8 Tesla C1060 GPUs is 44 times faster than the cluster using 16 AMD Phenom 9650 CPUs. A graphical summary about the computational time is shown in Fig. 8.

The increased number of active nodes increases the APCC proportionally. However, the AECC remains roughly the same level as the computation time is decreased proportionally at the same time. We can see from the result that the cluster using a single FPGA has the lowest AECC. A graphical summary about the AECC is shown in Fig. 9.

We propose a new approach for identifying speed and energy efficient accelerator allocation, called Efficient Allocation

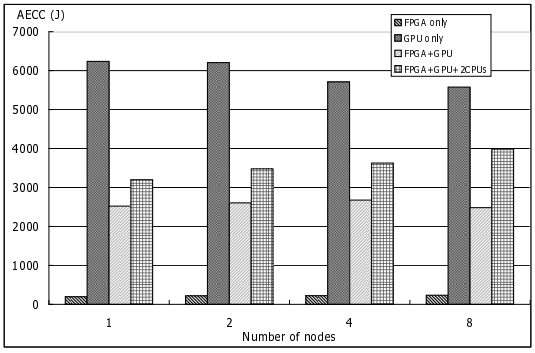


Fig. 9. The AECC of GARCH asset simulation.

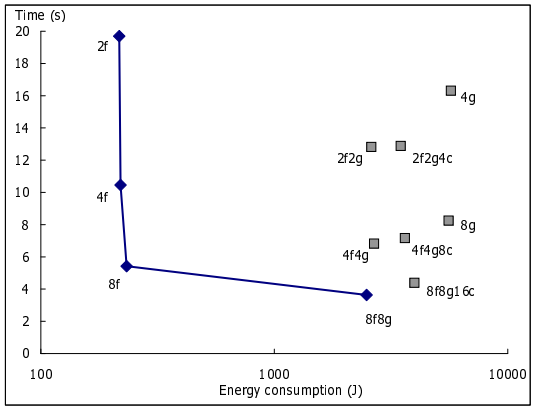


Fig. 10. The computation time and energy consumption for GARCH asset simulation in our cluster. The solid line is the Efficient Allocation Line (EAL). 2f2g4c denotes a design with 2 FPGAs, 2 GPUs and 4 CPUs.

Line (EAL). A scatter plot graph is firstly constructed with the computation time versus energy consumption for all accelerator allocation combinations. The EAL is then constructed by drawing a line linking the leftmost and bottommost allocations. The allocations of computational devices along the EAL are called "efficient" compared with the other allocations, as they are either energy efficient (the lowest energy consumption at a given computational time budget), or speed efficient (the lowest computational time at a given energy budget). Fig. 10 shows the computation time versus the energy consumption (AECC) of different accelerator allocations for the GARCH asset simulation in our 8-node cluster. The solid line is the EAL.

In this GARCH asset simulation application, FPGA is both faster and more energy efficient than the other two computational devices (GPU and CPU). We can simply allocate as many FPGAs as possible in the cluster. However, in the case of one accelerator is more speed efficient, but less energy efficient than the others, identifying the optimized device allocation will be much more challenging. The EAL can then be used for optimizing accelerator allocation. A dynamic scheduling policy based on the EAL could also be developed such that it allocates the tasks to the accelerators based on a certain energy budget or time budget which can vary during run time.

## VII. Conclusion

In this work, we propose a dynamic scheduling Monte-Carlo framework for collaborative computation in a multi-accelerator heterogeneous cluster. The load balancing process is automated by employing dynamic scheduling policies using the proposed framework. The framework is scalable and extensible for a variety of dynamic scheduling policies. We have shown that the proposed framework is viable by mapping two applications involving financial computation.

From our results, the overall performance of a Monte-Carlo simulation can be improved by allowing heterogeneous accelerators to work collaboratively. We explore different schemes of scheduling the workloads to the processing units to better utilize the computing resources. We also explore the speed and energy consumption trade-off for different accelerator allocation, and we propose the Efficient Allocation Line (EAL) as a method to identify the most efficient accelerator allocations.

Future work includes the automation for design development in this framework. More sophisticated dynamic scheduling policies will be designed and more complex Monte-Carlo applications involving data-dependency will be tested in our framework. We also intend to collaborate with other institutes to form a "cluster of heterogeneous clusters" in solving practical scientific problems.

**Acknowledgments.** The support of the Croucher Foundation, UK EPSRC, Alpha Data, nVidia, and Xilinx is gratefully acknowledged.

## References

[1] D. Strenski, "The Cray XD1 computer and its reconfigurable architecture," Cray Inc., Tech. Rep., July 2005.
[2] R. Baxter et al., "Maxwell – a 64 FPGA supercomputer," *Engineering Letters*, vol. 16, no. 3, pp. 426–433, 2008.
[3] nVidia, *nVidia CUDA Programming Guide v2.1*, 2008.
[4] T. Endo and S. Matsuoka, "Massive supercomputing coping with heterogeneity of modern accelerators," in *IEEE International Symposium on Parallel and Distributed Processing, IPDPS'08*, 2008, pp. 1–10.
[5] M. Showerman et al., "QP: A heterogeneous multi-acceleator cluster," in *10th LCI International Conference on High-Performance Clustered Computing*, Boulder, Colorado, March 2009.
[6] K. H. Tsoi and W. Luk, "Axel: a heterogeneous cluster with FPGAs and GPUs." in *Proc. ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA)*, 2010, pp. 115–124.
[7] "Reference is blanked for blind review."
[8] X. Tian, K. Benkrid, and X. Gu, "High performance Monte-Carlo based option pricing on FPGAs," *Engineering Letters*, vol. 16, no. 3, pp. 434–442, 2008.
[9] L. A. Abbas-Turki, S. Vialle, B. Lapeyre, and P. Mercier, "High dimensional pricing of exotic European contracts on a GPU cluster, and comparison to a CPU cluster," in *International Parallel and Distributed Processing Symposium*, 2009, pp. 1–8.
[10] "Reference is blanked for blind review."
[11] J. Hull and A. White, "The use of the control variate technique in option pricing," *Journal of Financial and Quantitative Analysis*, vol. 23, no. 03, pp. 237–251, 1988.
[12] F. Black and M. Scholes, "The pricing of options and corporate liabilities," *Journal of Political Economy*, vol. 81, no. 3, pp. 637–654, 1973.
[13] T. Bollerslev, "Generalized autoregressive conditional heteroskedasticity," *Journal of Econometrics*, vol. 31, no. 03, pp. 307–327, 1986.
[14] D. B. Thomas and W. Luk, "Non-uniform random number generation through piecewise linear approximations," in *Proc. Int. Conf. on Field Programmable Logic and Applications*, 2006, pp. 1–6.
[15] G. E. P. Box and M. E. Muller, "A note on the generation of random normal deviates," *Ann. Math. Statist.*, vol. 29, no. 02, pp. 610–611, 1958.