Transactions in GIS

Research Article

Transactions in GIS, 2014, 18(S1): 3–24

Geocomputation over the Emerging Heterogeneous Computing Infrastructure

Xuan Shi*, Chenggang Lai†, Miaoqing Huang† and Haihang You‡

*Department of Geosciences, University of Arkansas
†Department of Computer Science and Computer Engineering, University of Arkansas
‡Institute of Computing Technology, Chinese Academy of Sciences

Abstract

Emerging computer architectures and systems that combine multi-core CPUs and accelerator technologies, like many-core Graphic Processing Units (GPUs) and Intel's Many Integrated Core (MIC) coprocessors, would provide substantial computing power for many time-consuming spatial-temporal computation and applications. Although a distributed computing environment is suitable for large-scale geospatial computation, emerging advanced computing infrastructure remains unexplored in GIScience applications. This article introduces three categories of geospatial applications by effectively exploiting clusters of CPUs, GPUs and MICs for comparative analysis. Within these three benchmark tests, the GPU clusters exemplify advantages in the use case of embarrassingly parallelism. For spatial computation that has light communication between the computing nodes, GPU clusters present a similar performance to that of the MIC clusters when large data is applied. For applications that have intensive data communication between the computing nodes, MIC clusters could display better performance than GPU clusters. This conclusion will be beneficial to the future endeavors of the GIScience community to deploy the emerging heterogeneous computing infrastructure efficiently to achieve high or better performance spatial computation over big data.

1 Introduction

High-resolution geospatial data has become increasingly available along with an accelerating increase in data volume. Consequently spatial data may be stored as separate tiles for efficient data exchange, integration, processing, analysis and visualization. The distribution or separation of data has raised a variety of challenges and complexities in spatial computation for knowledge discovery. Traditional geographic information systems (GIS) and remote sensing software may only work on one tile of data at a time for computation or analysis. If all tiles are merged or mosaicked into a single piece, its size typically becomes too huge to be processed on a desktop computer. If all the data are maintained as separate tiles, however, the analytic result may not be correct or consistent among multiple separated tiles. In the case of unsupervised image classification, the classes' mean values are determined by the *global* information derived from all tiles. When individual tiles of images are processed by separate procedures, the class mean values are calculated based on the *local* information

Address for correspondence: Xuan Shi, Department of Geosciences, University of Arkansas, Fayetteville, AR 72701, USA. E-mail: xuanshi@uark.edu

Acknowledgments: This research was partially supported by the National Science Foundation through the award OCI-1047916. The research used resources of the Keeneland Computing Facility at the Georgia Institute of Technology, which is supported by the National Science Foundation under Contract OCI-0910735. This research also used Beacon, which is a Cray CS300-AC Cluster Supercomputer. The Beacon Project is supported by the National Science Foundation and the State of Tennessee. The authors would also thank Nvidia Corporation for K20 GPU donations.

© 2014 John Wiley & Sons Ltd

doi: 10.1111/tgis.12108

of each tile. For this reason, even if each tile of the image can be classified into the same number of classes, the result is not consistent among different tiles when the spectral statistics of the pixels in different tiles are different. In the case of viewshed analysis, when observational points are near the border of the Digital Elevation Model (DEM) data, the visible areas should also cover the neighboring tiles of data. If viewshed calculation is implemented over each individual tile at a time, the result is not correct.

In the problem-solving and decision-making processes, the performance of geospatial computation is severely limited when massive datasets are processed. Heterogeneous geospatial data integration and analytics obviously magnify the complexity and the operational time-frame. Many large-scale geospatial problems may not be processible at all if the computer system does not have sufficient memory or computational power. For example, simulating urban growth has been a challenging theme, since the difficulty of modeling urban growth can be aggravated due to the massive computational intensity caused by complicated algorithms and large data volume that are often required in the simulation. It was reported that the comprehensive calibration process of the SLEUTH model, one of the most widely used urban land-use change models in the world, over a medium size dataset might need 1,200 CPU hours (Clarke 2003) to accomplish – if the system could run the program successfully. In emergency response practice, both the HAZUS module (FEMA 2013) and the MAEviz (NCSA 2009) have been suffering the performance and scalability constraints in handling large datasets. In public health research, Centers for Disease Control and Prevention (CDC) professionals were looking for alternative technologies that outperform ArcGIS due to the long processing times for geospatial analysis. In the Extreme Science and Engineering Discovery Environment (XSEDE) community, a request was discussed about geospatial data mining on XSEDE when very large data was applied. In service computation, although Web Processing Service (WPS) was released in 2006, it has not been feasible in real world practice since the server cannot handle dynamic spatial data processing in a service-oriented computing environment when big data is involved. In network computing, origin-destination (OD) cost matrix calculation is time-consuming when large datasets are applied.

Scalable and high performance solutions for geospatial computation are increasingly required since GIS has been employed in a variety of domain applications that have broader impacts for societal benefit. Emerging computer architectures and advanced computing technologies, such as Intel's Many Integrated Core (MIC) Architecture and Graphics Processing Unit (GPU), provide a promising solution to employ massive parallelism to achieve scalability with high performance for data intensive computing over large spatial data. In geospatial applications, a few initiatives utilized a single GPU to build a spatial index (Zhang et al. 2010, 2011) or calculate the viewshed to determine the locations over a digital terrain surface visible to a set of observer features (Zhao et al. 2013). In processing satellite images, most prior works deployed a single GPU, such as unsupervised image classification (Ye and Shi 2013), segmentation (Huang et al. 2013), and hyperspectral data analytics (Bernabe et al. 2013a, b). In geospatial simulation, agent based modeling has been implemented on individual GPUs (Tang and Bennett 2012) or multiple GPUs without communication between the GPUs (Aaby et al. 2010). Spatial computation by deploying multiple GPUs through combined MPI and CUDA programs were reported with a focus on interpolation using IDW and Kriging algorithms (Ye et al. 2011; Shi and Ye 2013). Currently there is no prior work on deploying MIC for geospatial computation.

From a perspective of efficiently utilizing the heterogeneous computer architecture for parallelizing geospatial computation, three categories of geospatial applications can be

© 2014 John Wiley & Sons Ltd

identified. Embarrassingly parallelism is the *first category* of applications in which spatial calculation can be implemented on distributed computing nodes and there is *no communication* between nodes. For example, in spatial interpolation, each cell in the output grid can be calculated by an individual thread and there is no dependency between the threads (Shi et al. 2013). Data conversion from one data type or projection to other types or projections can be another use case (Li et al. 2010). In spatial analysis, many functions for map algebra calculation over raster datasets can be accomplished through embarrassingly parallelism.

In parallel and distributed computation, data communication between computing nodes may be required in different scenarios. Before the computation is implemented, partial data may have to be shared or exchanged among the distributed nodes. For example, in Focal Statistics calculations, the value of each output cell is derived based on the input cells in a specified neighborhood. In surface analysis, aspect or slope is calculated based on the values of the cell's eight neighbors. For this reason, data at the boundary of the tiles has to be exchanged before the geospatial computation is executed. In the other scenario, when spatial data is processed as separate tiles by distributed and parallel computing nodes, geospatial computation may need to retrieve the global information from separated datasets to complete the task. In spatial statistics calculations, for example, the mean and standard deviation have to be derived from the entire datasets. As a result, the *second category* of spatial computation refers to those applications that either need data exchange *before* the computation is started, or need post-processing of the result *after* the computation is accomplished on distributed nodes. In many applications, such as agent-based modeling, data communication may have to be executed multiple times in order to complete the computational processes. In the *third category* of spatial computation, communication among distributed computing nodes is required *both* before *and* after the computation. One other significant issue in data communication is the amount of data exchanged that may have significant impact on the total performance. In general, data can be exchanged between the computing node in regular forms for the same amount of data, such as a given number of columns or rows, although irregular data exchange can be performed that will increase the difficulty in handling load balance.

In this pilot study, the CPU and GPU clusters on Keeneland and the MIC cluster on Beacon were utilized to implement three benchmarks that could be representative of three categories of geospatial computation, as discussed above. Kriging interpolation is applied as a use case of embarrassingly parallelism. Our prior works (Ye et al. 2011; Shi and Ye 2013) on interpolation were conducted on GPU clusters, but CPU or MIC clusters were not applied. Unsupervised image classification is selected as a representative benchmark for the second category because communication has to be implemented in each iteration of the classification processes when local summation information has to be integrated and synthesized to derive the global result. In the prior work, we successfully implemented unsupervised image classification on a single GPU (Ye and Shi 2013), but multiple GPUs and MICs were not applied yet for image classification. Finally, Cellular Automata (CA) is selected as a representative benchmark of the third category that has *intensive data communication* in geospatial calculation over distributed computing nodes. Since the status of each cell is dependent upon the status of its neighbors, for each iteration in neighborhood dependent operations, the value of the neighboring cells along the boundary of the data has to be exchanged before the computation is implemented. After the computation, the status of the cells on distributed nodes has to be summed up. For this reason, communication is required before and after the calculation. In spatial modeling, CA have been applied extensively. A generic CA applied as the solution is applicable and extensible to all CA-based simulations.

© 2014 John Wiley & Sons Ltd

In the following sections, this article first reviews the main characteristics of the emerging heterogeneous computer architectures and systems that are utilized as the computing infrastructure for this pilot study. As a result, the chosen approach can be applied commonly in different systems for comparison. The algorithms of the three benchmarks are summarized before the implementation details are introduced. The result and performance comparison are discussed in detail to demonstrate that the emerging advanced computer architecture and computing technologies have the potential to transform research in geospatial science. The solutions exemplified in this study can be extended to other geospatial computations with a broader impact in the future.

2 Emerging Heterogeneous Computer Architectures and Systems

2.1 Graphic Processing Unit (GPU)

The Graphics Processing Unit (GPU) was traditionally built for the purpose of efficiently manipulating computer graphics. For this reason, image processing has been a prime application that benefits from the marvelous parallelism inherent in graphic processing. Today's GPU has evolved rapidly to support General-Purpose computing (thus called GPGPU) through many-core processors capable of intensive computation and data throughput. Thus a modern GPU enables massively parallel computing for scientific computation and is not limited to processing computer graphics (Preis et al. 2009; Kalyanapu et al. 2011; Steinbach and Hemmerling 2012; Simion et al. 2012; Emeliyanenko 2013). By simultaneously executing tens of thousands of threads on hundreds of specialized GPU cores, GPGPU can achieve high performance computing over desktop and laptop computers.

GPU architecture has been evolving for many years. Taking the GPUs developed by NVIDIA as examples, it has gone through many generations, such as G80 → GT200 → Fermi → Kepler. NVIDIA's latest Kepler GPU architecture contains 15 streaming multiprocessors (SMXes), each of which consists of 192 single-precision cores and 64 double-precision cores, as shown in Figure 1. The Kepler architecture provides three advanced features to: (1) efficiently share the GPU resources among multiple host threads/processes; (2) flexibly create new

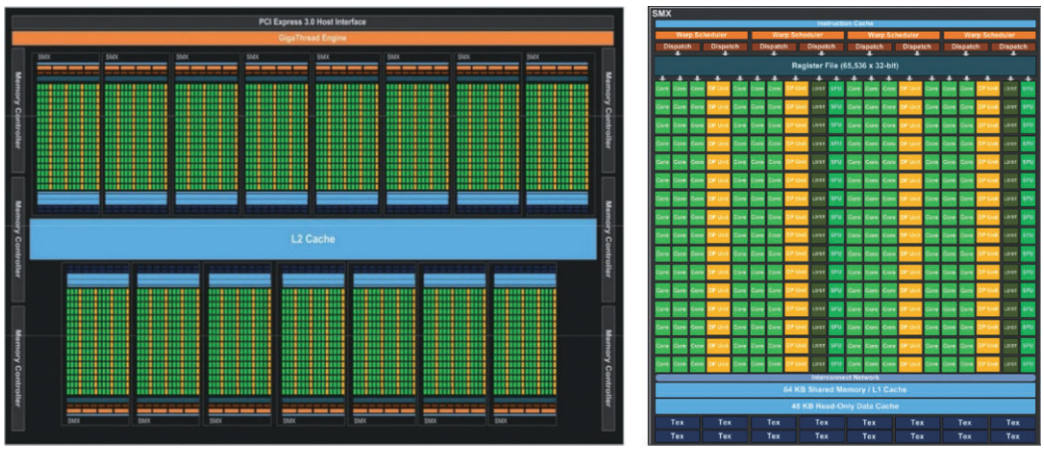


Figure 1 Nvidia's Kepler GPU architecture

© 2014 John Wiley & Sons Ltd

kernels on GPU (i.e. Dynamic Parallelism); and (3) reduce communication overhead across GPUs through GPUDirect. Although such advanced features cannot be tested in this pilot study since there are no Kepler clusters that can be accessed, the advantages of the latest generation of GPU are exemplified in comparison with the old generation of GPUs.

2.2 Intel's Many Integrated Core (MIC)

The first commercially available Intel coprocessor based on Many Integrated Core architecture (Jeffers and Reinders 2013) is Xeon Phi, as shown in Figure 2. Xeon Phi contains 60 scalar processors with vector processing units. Direct communication between MIC coprocessors across different nodes is also supported through MPI. Figure 3 shows two approaches to parallelizing applications on computer clusters equipped with MIC processors. The first approach is to treat the MIC processors as clients to the host CPUs. As shown in Figure 3a, the MPI processes will be hosted by CPUs, which will off-load the computation to the MIC processors. Multithreading programming models such as OpenMP can be used to allocate many cores for data processing. The second approach, as shown in Figure 3b, is to let each MIC core directly host one MPI process. In this way, the 60 cores on the same chip are treated as 60 independent processors while sharing the 8 GB on-board memory on the Xeon Phi 5110P. Although a few works on MIC-based computation have been reported (Crimi et al. 2013; Schmidl et al. 2013; Heinecke et al. 2013), few applications in geospatial sciences have been explored yet.

2.3 Comparison between Tesla K20 GPU and Intel MIC

Table 1 provides a comparison of the architecture of Nvidia's Tesla K20 GPU with Intel's MIC. Both of them deploy GDDR5 on-board memory to provide a high memory bandwidth. In terms of processing cores, the Nvidia K20 GPU contains thousands of low-performance cores running at 706 MHz, including both single precision (SP) and double precision (DP) cores. On the other hand, the Intel Xeon 5110P consists of 60 comparably more powerful cores running at 1.053 GHz. Therefore, if the application is fine-grain parallel, i.e. the workload can be divided into a large amount of small, however, concurrent pieces, it will fit the

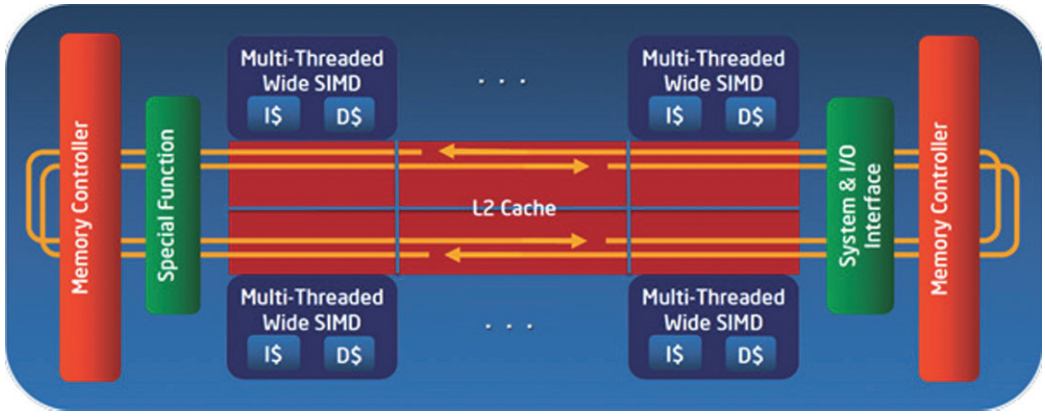


Figure 2 The architecture of Intel MIC

© 2014 John Wiley & Sons Ltd

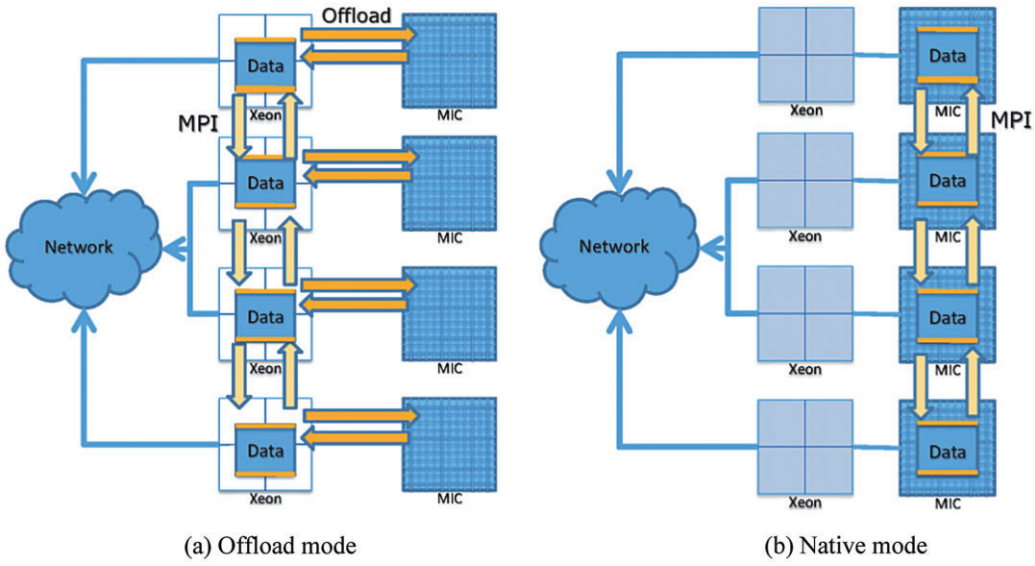


Figure 3 Two different approaches to implementing parallelism on an MIC cluster

Table 1 Architecture comparison between Tesla K20 GPU and Intel MIC

Devices	On-board Memory Capacity	Memory Bandwidth	No. of Cores	Core Frequency
Nvidia Tesla K20	5 GB GDDR5	208 GB/s	2496 (SP) +832 (DP)	706 MHz
Intel Xeon 5110P	8 GB GDDR5	320 GB/s	60	1.053 GHz

GPU architecture quite well. Otherwise, if the application is coarse-grain parallel, i.e. the workload needs to be distributed into dozens of parallel pieces, it will fit the MIC architecture. Because Xeon 5110P provides a higher memory bandwidth than that of the Nvidia K20 GPU, MIC has the potential to provide a better performance for memory-intensive applications while, on the other hand, computation-intensive applications may run on Nvidia K20 GPU better because of the large amount of processing cores on the GPU.

2.4 Keeneland and Beacon

We conducted our experiments on two platforms, the NSF sponsored Keeneland supercomputer (Keeneland 2014) and Beacon supercomputer (Beacon 2014). The Keeneland Initial Delivery System (KIDS) is a 201 Teraflop, 120-node HP SL390 system with 240 Intel Xeon X5660 CPUs and 360 Nvidia Fermi GPUs, with the nodes connected by a QDR InfiniBand network. Each node has two 6-core 2.8 GHz Xeon CPUs and three Tesla M2090 GPUs. The NVIDIA Tesla M2090 GPU contains 512 CUDA cores and 6 GB GDDR5 on-board memory. The Beacon system (Cray CS300-AC Cluster Supercomputer) offers access to 48

© 2014 John Wiley & Sons Ltd

compute nodes and six I/O nodes joined by a FDR InfiniBand interconnect providing 56 Gb/s of bi-directional bandwidth. Each compute node is equipped with two Intel Xeon E5-2670 8-core 2.6 GHz processors, four Intel Xeon Phi (MIC) coprocessors 5110P, 256 GB of RAM, and 960 GB of SSD storage. Each I/O node provides access to an additional 4.8 TB of SSD storage. Each Xeon Phi coprocessor contains 60 1.053 GHz MIC cores and 8 GB GDDR5 on-board memory. As a result, Beacon provides 768 conventional cores and 11,520 accelerator cores that provide over 210 TFLOP/s of combined computational performance, 12 TB of system memory, 1.5 TB of coprocessor memory, and over 73 TB of SSD storage.

Both platforms are equipped with a Lustre parallel distributed file system (Cluster File Systems, Inc. 2002), in which a technique called file striping can be applied to significantly improve the I/O performance. File striping is a process in which Lustre automatically divides data into chunks and distributes them across multiple object storage targets (OSTs). It plays an important role in running large scale computation by reducing the time spent on reading or writing a big data file to significantly improve the I/O performance. By setting the stripe count and stripe size, which are the tunable parameters of the Lustre file system, the I/O performance of a particular benchmark can be optimized. Stripe count is the number of OSTs into which a file is stored. For example, if the stripe count is set to 10, the file will approximately be partitioned in equal portions of 10 different OSTs. Stripe size is the chunk size into which a file is split to be distributed across OSTs.

3 Overview of the Algorithms for the Selected Three Benchmarks

Geocomputation is fundamental in GIS and remote sensing applications. Commercial and open source GIS and remote sensing software could have hundreds of computational modules. As the first geospatial application implemented on supercomputers Keeneland and Beacon, although varieties of options could be applied to conduct such a pilot study, three benchmarks were selected to explore the generic routines for developing geocomputation over the emerging heterogeneous computing infrastructure. From the perspective of experimental design for this study, examining the scalability of the computational solution over the hybrid environment is of particular interest as many big problems could hardly be resolved by a desktop computer or by a single GPU or MIC. Utilizing multiple GPUs and MICs should have the potential to handle large scale data and computational problems.

Considering the complexity of the interpolation algorithm, Kriging was selected as a typical case of embarrassingly parallelism, though many other functional modules in GIS software could be more interesting to a different audience. Unsupervised image classification was selected as a representative use case because a single CPU or GPU could hardly process a large image. Game of Life is selected for this pilot study since it helps to develop a generic routine to deal with data communication in a heterogeneous computing environment, while the solution can be easily adapted to other Cellular Automata based spatial simulation. This section briefly reviews and summarizes the algorithms of the three benchmarks, followed by sections about the implementation and the comparison of the results.

3.1 Kriging Interpolation: The Use Case of Embarrassingly Parallelism

Kriging is a geostatistical interpolation method that has a complex implementation and a large computation load (Oliver and Webster 1990; Cheng et al. 2010; Srinivasan et al. 2010). It can

© 2014 John Wiley & Sons Ltd

be viewed as a point interpolation that reads input point data and returns a raster grid with calculated estimations for each cell. Each input point is in the form (x_i, y_i, Z_i) where x_i and y_i are the coordinates and Z_i is the value. The estimated values in the output raster grid are calculated as a weighted sum of input point values:

$$\hat{Z}(x, y) = \sum_{i=1}^{k} w_i Z_i \tag{1}$$

where w_i is the weight of the i-th input point. Theoretically the estimation can be calculated by the summation through all input points. In general, users can specify a number k and sum over k nearest neighbors of the estimated point because the farther the sampled point is from the estimated value, the less impact it has over a certain distance. By default, ArcGIS uses 12 nearest points ($k = 12$) in the Kriging calculation.

A set of weights is calculated separately by solving the following matrix equation for each output point:

$$V \cdot W = D \tag{2}$$

where V is the semivariogram matrix, W is a vector of the weights, and D is a vector in which the semivariogram values for distances from k input points to the output point are filled in. This can also be expressed as:

$$\begin{bmatrix} 0 & \gamma(h_{12}) & \gamma(h_{13}) & \cdots & \gamma(h_{1k}) & 1 \\ \gamma(h_{21}) & 0 & \gamma(h_{23}) & \cdots & \gamma(h_{2k}) & 1 \\ \gamma(h_{31}) & \gamma(h_{32}) & 0 & \cdots & \gamma(h_{3k}) & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & 1 \\ \gamma(h_{k1}) & \gamma(h_{k2}) & \gamma(h_{k3}) & \cdots & 0 & 1 \\ 1 & 1 & 1 & \cdots & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} w_1 \\ w_1 \\ w_1 \\ \vdots \\ w_1 \\ \lambda \end{bmatrix} = \begin{bmatrix} \gamma(h_{p1}) \\ \gamma(h_{p2}) \\ \gamma(h_{p3}) \\ \vdots \\ \gamma(h_{pk}) \\ 1 \end{bmatrix} \tag{3}$$

This means the size of the semivariogram matrix V is $(k+1)*(k+1)$. When a Kriging algorithm is implemented on a single GPU, the size of V may lead to the memory issues when a large amount of data is applied. In this pilot study, the Gaussian elimination method is used to solve Equation (2). In a parallel computing environment, each thread can be used to calculate one cell of the output grid. Since there is no dependency between each node, interpolation can be a typical case of embarrassingly parallelism.

3.2 *ISODATA for Unsupervised Image Classification: The Use Case of Light Communication*

Unsupervised image classification is the foundation for other classification approaches, such as supervised and object oriented classifications. The Iterative Self-Organizing Data Analysis Technique Algorithm (ISODATA) (Jenson 1999) is one of the most frequently used algorithms for unsupervised image classification in remote sensing applications. In general, ISODATA can be implemented in three steps: (1) calculate the initial mean value of each class; (2) classify each pixel to the nearest class; and (3) calculate the new class means based on all pixels in one class. The second and third steps are repeated until the change between two iterations is small enough.

To perform ISODATA classification, several parameters need to be specified (see Table 2). The initial class means are derived by the statistics of the original data sets, although the initial means can be assigned arbitrarily. Accordingly, the initial class means are evenly distributed in a multi-dimensional feature space along a vector between two points ($\mu_1 - \sigma_1$, $\mu_2 - \sigma_2$, . . . , μ_k

© 2014 John Wiley & Sons Ltd

Table 2 Prerequisites for ISODATA implementation

Symbol	Definition
C	The number of classes to be created
T	The convergence threshold, which is the maximum percentage of pixels whose class values are allowed to be unchanged between iterations
M	The maximum number of iterations to be performed

$- \sigma_k$) and ($\mu_1 + \sigma_1, \mu_2 + \sigma_2, \ldots, \mu_k + \sigma_k$) in the first iteration, where μ_i denotes the mean of the i-th band ($i = 1, 2, \ldots, k$) and k is the total number of bands in the dataset, and σ_i is the standard deviation of band i. When the image is classified into five classes, the initial five cluster means are marked as the dots in the chart and are evenly distributed between ($\mu_A - \sigma_A$, $\mu_B - \sigma_B$) and ($\mu_A + \sigma_A$, $\mu_B + \sigma_B$).

Throughout the iteration procedures, while the maximum number of iterations (M) and the convergence threshold (T) are not reached, the means of all classes are recalculated, causing the class means to shift in the feature space. During the iterative calculations, each pixel is compared with the new class means and will be assigned to the nearest class mean. During the process of classification, each class is labeled as a certain type of object. The change between two consecutive iterations can be either the percentage of pixels whose class labels have been changed between two iterations, or the accumulated distances of the class means that have been changed in the feature space between two iterations. The iterative process will not be terminated until either the maximum number of iteration or the convergence threshold (T) is reached or exceeded, which means the change between two iterations is *small* enough, that is, the *maximum* percentage of pixels whose class values that are *unchanged* between iterations.

In a parallel computing environment, each computing node can be used to calculate one section of the entire image. In order to obtain the cluster mean value, the local summation value derived from each node has to be integrated by a single node to derive the global mean value and the new cluster centers. For this reason, data exchange happens at the end of each iteration and thus can be a light communication use case.

3.3 Game of Life: The Use Case of Intensive Communication

Conway's Game of Life (GOL) is a well-known classical Cellular Automata model (Gardner 1970). According to the transition rules, a cell can live or die based on the condition of its 3 × 3 neighborhood. The pseudo code of such transition rules can be described as follows:

```
FUNCTION Transition (cell, t)
n = number of alive neighbors of cell at time t
IF cell is alive at time t
  IF n > 3
    THEN cell dies of overcrowding at time t+1
  IF n < 2
    THEN cell dies of under-population at time t+1
  IF n = 2 OR n = 3
    THEN cell survives at time t+1
```

© 2014 John Wiley & Sons Ltd

ELSE (*i.e. cell is dead at time t*)
IF $n = 3$
THEN *cell becomes a live cell by reproduction at time t+1*

In the parallel computing environment, when the entire matrix is cut into several segments, for example, the rows at the boundary of each segment have to be exchanged before the simulation can be implemented. After each simulation, the total living cells have to be counted and the summation has to be done by aggregating the local values into a global value. For this reason, data exchange and communication have to be implemented before and after each iteration, which is a typical use case of intensive communication. The solution on the GOL benchmark can be applied in varieties of geospatial modeling, such as in a CA-based simulation of urban sprawl study, infectious disease spread in public health research, and environmental change upon possible socioeconomic consequences.

4 Implementation of the Benchmarks

The benchmarks were *initially* developed in serial C programs in order to establish a standard for quality control ensuring the output results from all different solutions will be the same. *Consequently* the GPU programs running on a single GPU were first developed before the benchmarks were implemented on Keeneland using multiple GPUs. Since the desktop GPUs are not comparable to those on Keeneland, the performance of the same programs implemented on the latest version of GPU on Keeneland is obviously much better than that from the old version of GPUs installed on the desktop machine. *Lastly*, the MPI + MIC solutions were developed for implementing three benchmarks on Beacon. Result and performance comparison of the experimental design are discussed in Section 5.

4.1 Serial C Programs for Quality Control

For each use case, we first developed serial C programs to establish a standard for quality control, understand the computation and performance limits for data, and compute intensive applications implemented on a desktop machine. In this pilot study, we used a desktop computer in which Windows 7 is the operating system. The desktop machine has an Intel Pentium 4 CPU with 3.00 GHz frequency and 4 GB RAM. We built the serial program in C within Visual Studio .NET 2010 to implement the Kriging interpolation. The output grid is a matrix that has 1,440 x 720 = 1,036,800 cells. When 9,817 sample points were used as the input, it took 2,771 seconds (i.e. more than 46 minutes) to complete the Kriging calculation in the serial C program when 10 nearest neighbors were used. In the case of the unsupervised image classification, one tile of three-band imagery data was used. That image has a resolution of 0.5 m and the file size is18 GB ($80{,}000 \times 80{,}000 \times 3$). Commercial remote sensing software would take 3 hours, 44 minutes, 37 seconds (13,477 seconds in total) to read and classify this image into five classes. In the case of the Game of Life computation, a 100-iteration simulation over a $10{,}000 \times 10{,}000$ cell space was accomplished in about 100 minutes.

4.2 Benchmark in CUDA C Programs on a Single GPU

The serial C programs were further transformed into CUDA programs tested on a single GPU over small datasets to guarantee that the output results are the same as that derived from the

© 2014 John Wiley & Sons Ltd

Table 3 Computation time comparison of single-GPU implementation on M2090 and K20 (with time counted in seconds). K20 is able to achieve a better performance than M2090 in the Kriging calculation and Game of Life simulation without any particular tuning in this pilot study

			Game of Life		
GPU	**Kriging**	**ISODATA**	8,192 x 8,192	16,384 x 16,384	32,768 x 32,768
M2090	24.64	49.19	12.86	51.29	204.99
K20	10.32	46.86	3.25	12.58	46.99
Speedup	**2.39**	**1.05**	**3.96**	**4.08**	**4.36**

serial C programs. The experiments were initially conducted on a desktop PC with a NVIDIA GeForce GTS 450. In the case of the Kriging interpolation, when the abovementioned same size of data was used as the input, it took 52 seconds to complete the task. In the case of unsupervised image classification, due to the memory limit on such an old GPU, a small dataset was used to validate the parallel program and the quality of the result. The file size of the small data is 89.6 MB. In this case, the GPU on a desktop computer can complete the classification in 5.43 seconds and achieve a speedup of about 42 when the image is classified into five classes. In the case of the Game of Life computation, the simulation at a size of 10,000×10,000 for 100 iterations took about 6 minutes to complete, achieving a speedup of 16.7 on a single GPU.

Since we do not have access to a computer cluster with NVIDIA Kepler GPUs, in order to project the performance of a hybrid cluster with Kepler (K20) GPU, we compare the performance of the same single-GPU implementation of the Kriging interpolation using 10 nearest neighbors on both the Tesla M2090 and K20 GPUs. Table 3 displays the result. Due to the on-board memory limit of the single GPU, the image size is reduced to $10{,}000 \times 10{,}000 \times 3$ in the unsupervised image classification test. Based on the performance on the single-node implementation, it seems that the K20 is able to achieve a better performance than M2090 in the Kriging calculation and the Game of Life simulation without any particular tuning in this pilot study. Obviously both the Tesla M2090 and K20 GPUs displayed much better performance than the old generations of GPUs, such as GTS 400 series and GTX 200 series.

4.3 Benchmarks in Hybrid MPI Programs on Keeneland and Beacon

We developed hybrid MPI programs to deploy multiple CPUs and GPUs on Keeneland and multiple MICs on Beacon to implement the three benchmarks. The Intel Xeon E5 8-core CPUs on Keeneland are used to implement MPI + CPU programs for the three benchmarks. Each MPI process runs on a single CPU. The process on the CPU is a single-thread process. When a number of MPI processes are created in the parallel application, the same number of CPU processors are allocated.

MPI + GPU programs are implemented on Keeneland. Each MPI process runs on a host CPU, which conveys all the data processing to one client GPU processor. On the NVIDIA M2090 GPU, all 512 CUDA cores are deployed for computation. If a number of MPI processes are created in the parallel application, the same number of CPU processors and the same number of GPU processors are allocated. The host CPU is responsible for the MPI communi-

© 2014 John Wiley & Sons Ltd

cation, and the client GPU is in charge of data processing. Since we did not have access to K20 GPU clusters, the GPUDirect technique was not tested yet.

Lastly, the Intel Xeon Phi 5110P on the Beacon is used for such benchmarks through the MPI + MIC programs. Each MIC core will directly host one MIC process, as shown in Figure 3b. In this case, when a given number (e.g. N) of Xeon Phi coprocessors are used, $N \times 60$ MPI processes are created in the parallel implementation. This programming model on the Intel MIC cluster has better portability than the offload model shown in Figure 3a. Legacy parallel code using MPI can be directly compiled and executed on systems running on the Intel MIC without much tuning.

5 Results and Performance Comparison of the Experimental Benchmarks

In this pilot study, the problem size is fixed for each benchmark when the number of participating MPI processes is increased so that the strong scalability of the parallel implementations can be examined. In the case of the Game of Life simulation, since the random processes on different nodes may generate different results, we create an initialization file through the serial C program as the starting point for the simulation in a distributed computing environment, to ensure the result is the same as that derived from the serial C program or the CUDA program on a single GPU.

5.1 Result of the Kriging Interpolation on Keeneland and Beacon

In the Kriging interpolation benchmark, the calculation is evenly partitioned among a given number of MPI processes. Each MPI process will perform the interpolation calculation on a number of rows in the $1{,}440 \times 720$ output grid. In this experiment, the value of an unsampled location will be estimated using the values of the 10, 12, and 15 nearest sample points. Since the computation in each MPI process is purely local, then there is no cross-processor communication.

From Table 4 and Figure 4, it can be found that the hybrid implementation on the GPU and MIC clusters can easily outperform the parallel implementation on CPU clusters. For this embarrassingly parallel benchmark, the stream processing architecture on the NVIDIA GPU works quite well and is able to outperform the parallel implementation on the Intel MIC, which employs relatively low-performance cores.

5.2 Result of ISODATA on Keeneland and Beacon

The input of the ISODATA is a high-resolution image of 18 GB with a dimension of $80{,}000 \times 80{,}000$ for three bands. The objective of this benchmark is to classify the image into n classes. In this benchmark test, $n = 15$. In order to parallelize the computation, the whole image is partitioned into blocks of the same size. Each block is sent to a different processor.

The whole classification process will go through many iterations until: (1) the number of pixels that do not switch classes between two consecutive iterations is above a threshold (i.e. 95%); or (2) the preset maximum number of iterations (i.e. 15) is reached. During each iteration, each MPI process first calculates the local means of 15 classes. Then all MPI processes send their local means to the head MPI process (i.e. MPI rank = 0). After the head MPI process collects all the local means, it calculates the global means for the 15 classes and returns them to all other MPI processes. Then all the MPI processes start the computation of the next iteration.

© 2014 John Wiley & Sons Ltd

Table 4 Kriging interpolation on Keeneland and Beacon (with time counted in seconds)

		Keeneland KIDS								Beacon			
		MPI + CPU				MPI + GPU				MPI + MIC			
No. of processors	No. of neighbors	Read	Comp.	Write	Total	Read	Comp.	Write	Total	Read	Comp.	Write	Total
2	10	0.03	**1,054.99**	1.13	1,056.15	0.03	**6.77**	0.9	7.69	0.24	**25.4**	2.16	27.79
	12	1.09	**960.12**	1.10	962.30	0.12	**7.11**	7.84	15.06	1.09	**188.18**	8.29	197.56
	15	0.03	**983.13**	1.29	984.45	0.03	**14.96**	16.65	31.64	0.93	**190.06**	8.27	199.25
4	10	0.03	**528.43**	0.98	529.44	0.03	**3.92**	0.99	4.93	0.22	**16.06**	1.97	18.25
	12	0.03	**477.87**	1.12	479.03	0.03	**3.81**	11.71	15.55	0.93	**93.80**	8.37	103.10
	15	0.03	**488.42**	1.01	489.46	0.03	**5.30**	26.37	31.70	0.91	**94.92**	7.99	103.81
8	10	0.03	**260.6**	0.95	261.57	0.03	**2.76**	0.94	3.73	0.26	**6.93**	12.97	20.15
	12	0.03	**237.30**	0.92	238.25	0.03	**2.54**	7.60	10.17	0.90	**31.25**	229.57	261.71
	15	0.03	**242.54**	0.90	243.47	0.03	**2.94**	15.92	18.89	0.97	**31.78**	219.58	252.34

© 2014 John Wiley & Sons Ltd

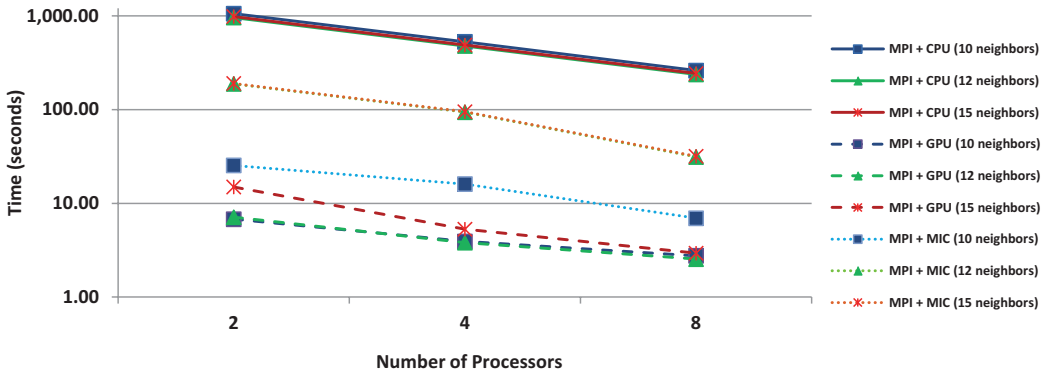


Figure 4 Performance chart of the computational time for the Kriging interpolation on different configurations. GPU cluster seems to have better performance than CPU cluster and MIC cluster in resolving embarrassingly parallel computing problems due to the massively parallel computing nodes and threads in the GPU cluster

Table 5 ISODATA (15 classes) on Keeneland and Beacon (with time counted in seconds)

	Keeneland KIDS						Beacon		
	MPI + CPU			MPI + GPU			MPI + MIC		
No. of processors	Read	Comp.	Total	Read	Comp.	Total	Read	Comp.	Total
36	6.04	**232.22**	238.26	3.91	**100.26**	104.17	N/A		
64	12.6	**140**	152.59	3.51	**59.18**	62.69			
80	15.03	**129.72**	144.74	21.11	**47.12**	68.24	41.27	**58.75**	100.02
100	1.29	**81.31**	82.59	36.35	**33.81**	70.16	27.01	**50.01**	77.02
120	0.98	**81.34**	82.39	22.29	**33.95**	56.24	32.32	**40.08**	72.40

The performance of the ISODATA benchmark on three different platforms is shown in Table 5. When the MPI-native programming mode is used to implement this benchmark on Beacon, it is found that this 18 GB image cannot be handled by 36 or 64 MIC processors. This is due to the fact that there is a full software stack on each MIC core to support MPI communication, which consumes a lot of memory and leaves not much space for data. Therefore, we allocate more MIC processors, i.e. 80, 100, and 120, on Beacon to implement ISODATA.

Figure 5 displays the performance comparison of various implementations of ISODATA on Keeneland and Beacon. It can be found that the performance gap between the MIC processors and GPUs becomes quite small when more GPUs and MICs are used for computation. Further, the implementation on Beacon keeps the scalability while the number of processors increases. One reason is that the Fourteen Data Rate (FDR) InfiniBand network on Beacon provides much higher bandwidth than the Quad Data Rate (QDR) InfiniBand network on Keeneland KIDS, i.e. 56 Gb/s (FDR) vs. 32 Gb/s (QDR). The advantage of the more efficient communication network on Beacon is further demonstrated when the number of participating processors is increased from 100 to 120. Such an advantage is further

© 2014 John Wiley & Sons Ltd

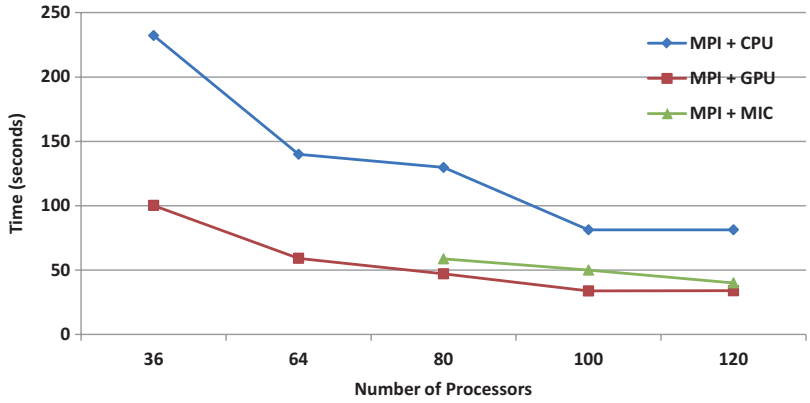


Figure 5 Performance chart of the computational time for ISODATA (15 classes) on different configurations. The performance gap between the MIC processors and GPUs becomes quite small when more GPUs and MICs are used for computation

Table 6 Classifying the image into five, 10, 15, and 20 classes on Keeneland and Beacon

		Keeneland KIDS			Beacon		
		MPI + GPU			MPI + MIC		
No. of processors	No. of Classes	Read	Comp.	Total	Read	Comp.	Total
80 GPUs vs. 80 MICs	5	2.56	**11.47**	14.02	27.62	**11.71**	39.34
	10	2.97	**36.39**	39.37	15.37	**39.07**	54.44
	15	2.63	**77.60**	80.23	41.27	**58.75**	100.02
	20	2.62	**102.05**	104.67	41.79	**90.00**	131.79
100 GPUs vs. 100 MICs	5	54.09	**9.49**	63.58	36.10	**9.14**	45.24
	10	74.11	**32.88**	107.00	41.99	**31.52**	73.51
	15	36.35	**33.81**	70.16	27.01	**50.01**	77.02
	20	62.46	**82.19**	144.65	22.66	**72.04**	94.70

observed in different scenarios when the image is classified into five, 10, 15, and 20 classes using 100 and 120 GPUs on Keeneland and 80 and 100 MICs on Beacon. The result is illustrated in Table 6 and Figure 6. If we compare the classification time used on classifying the image into different classes, the performance is pretty close on Keeneland and Beacon.

5.3 Result of the Game of Life runs on Keeneland and Beacon

Game of Life (GOL) is a generic Cellular Automata program, in which the status of each cell is dependent upon its eight neighbors. In this benchmark test, the simulation is implemented for 100, 150, and 200 iterations. For each iteration, the status of all cells is updated simultaneously. In order to parallelize the updating process, the cells in the $n \times n$ matrix grid are partitioned into stripes in the row-wise order. Each stripe is handled by one MPI process. At the

© 2014 John Wiley & Sons Ltd

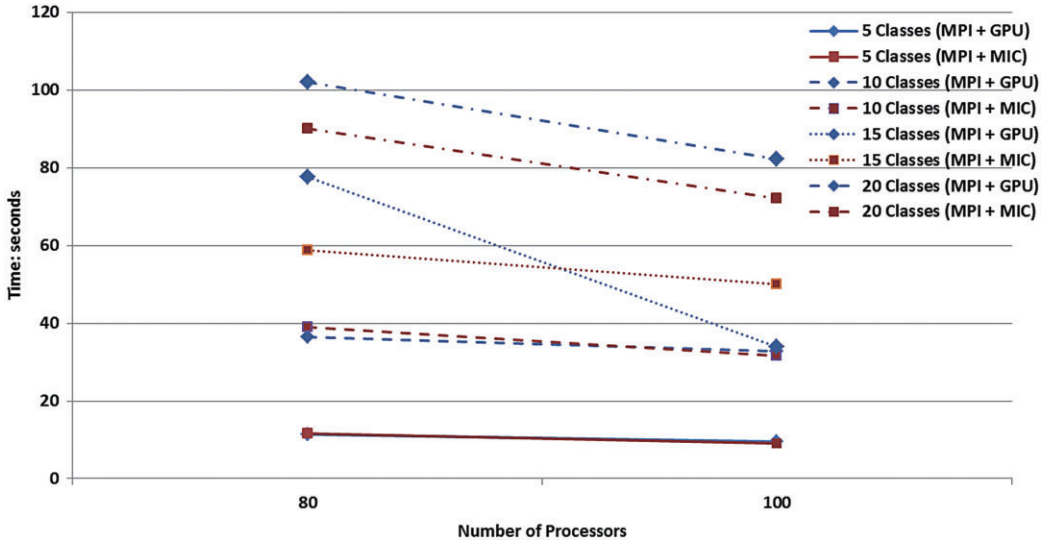


Figure 6 Performance chart of the computational time for classifying the image into five, 10, 15, and 20 classes on Keeneland and Beacon. The performance gap between the MIC processors and GPUs becomes quite small when the image is classified into five or 10 classes. The performance on Beacon seems not stable and consistent in this study when the image is classified into 15 classes

beginning of each iteration, the status of the cells along the boundaries of each stripe has to be exchanged with its neighbor through the MPI send and receive command. In the case of running the Game of Life in GPU clusters on Keeneland, applying atomicAdd and shared memory techniques can further improve the performance.

GOL is tested by three different grid sizes of 8,192 × 8,192, 16,384 × 16,384, and 32,768 × 32,768. By observing the performance results in Table 7, it can be found that the strong scalability is demonstrated for all three cases. The exception happens when 16 MIC processors are allocated in the implementation as the performance of 16 MIC processors is worse than the performance of 8 MIC processors. This is due to the fact that the grid is partitioned into 960 MPI processes on 16 MIC processors. At this scale, the performance gain from the reduced workload on each MIC core is offset by the increase of the communication cost. For this benchmark, the K20 can consistently outperform the M2090 by ~4 times speedup based on the results in Table 2.

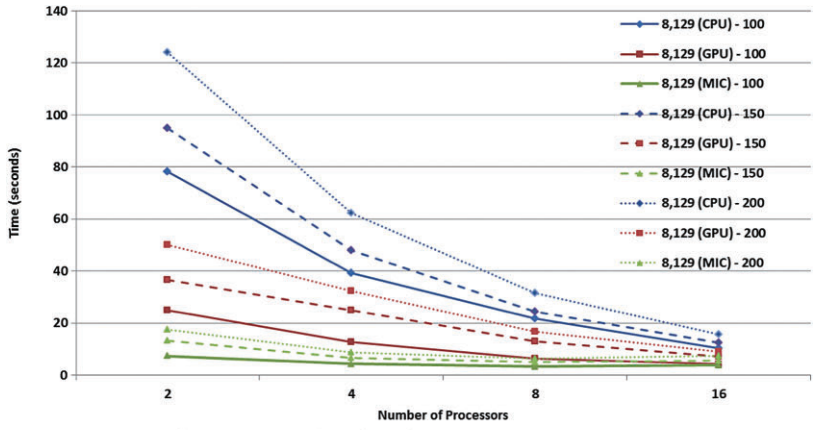
The performance comparison of various implementations of Game of Life is shown in Figure 7. The FDR Infiniband network on Beacon clearly demonstrates its advantage for this intensive communication benchmark. Because the FDR provides the higher bandwidth than the QDR, it takes less time to communicate on the Beacon cluster than on the Keeneland cluster. As a result, the implementation on MIC clusters outperforms the implementations on GPU clusters for most cases. This is because for each iteration, the status of the cells along the boundaries of each stripe has to be exchanged with its neighbor through the MPI send and receive command. When GPUs are utilized in this case, data has to be copied back and forth between the CPUs and GPUs over distributed computing nodes. In order to explore the potential of whether the hybrid implementation with GPU could outperform the implementation on the MIC processors, the direct GPU communication capability of Kepler GPUs would be worth investigating and validating.

© 2014 John Wiley & Sons Ltd

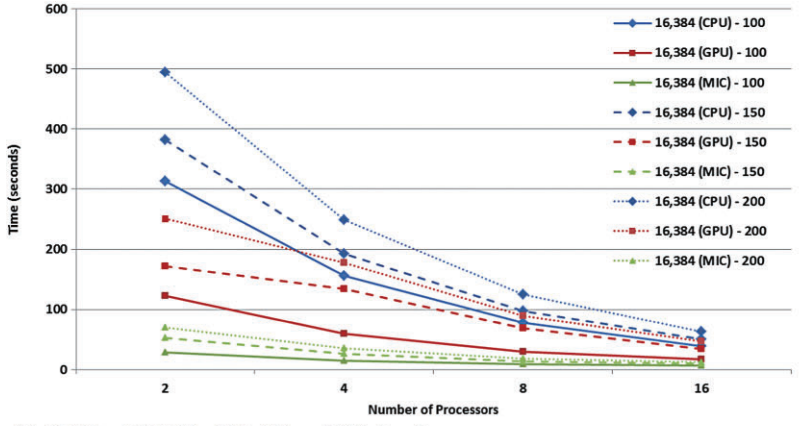
Table 7 Game of Life on Keeneland and Beacon (with time counted in seconds)

		8,192 x 8,192			16,384 x 16,384			32,768 x 32,768		
No. of processors	No. of iterations	MPI + CPU	MPI + GPU	MPI + MIC	MPI + CPU	MPI + GPU	MPI + MIC	MPI + CPU	MPI + GPU	MPI + MIC
2	100	78.15	24.92	7.33	312.69	122.19	28.96	1242.19	483.58	116.93
	150	95.01	36.62	13.23	381.33	172.37	52.90	1527.32	745.42	213.53
	200	124.02	50.23	17.40	494.39	250.63	69.89	1995.89	1011.08	288.87
4	100	39.2	12.79	4.27	155.64	59.14	14.65	625.92	242.43	58.57
	150	48.13	24.82	6.64	192.62	134.52	26.14	771.46	546.92	107.29
	200	62.31	32.37	8.71	249.08	177.99	34.84	1012.17	725.01	141.22
8	100	21.82	6.3	3.30	78.14	29.66	8.47	311.34	118.31	30.98
	150	24.35	13.07	4.92	97.74	68.20	13.28	392.07	274.81	54.00
	200	31.66	16.71	6.50	124.40	89.76	17.66	505.85	362.27	70.66
16	100	10.41	4.15	3.75	39.35	17.2	6.56	159.05	63.83	17.16
	150	12.44	7.21	5.63	50.00	34.65	9.70	201.91	138.56	26.91
	200	15.74	9.12	7.45	62.89	46.30	12.84	258.23	181.37	35.50

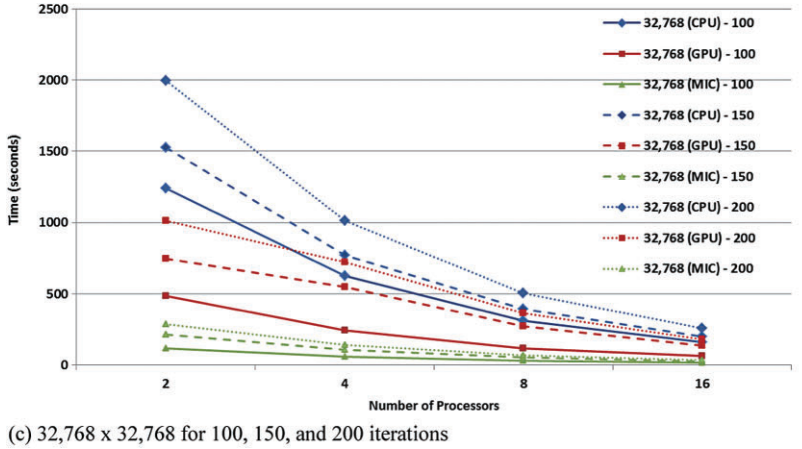
© 2014 John Wiley & Sons Ltd



(a) 8,192 x 8,192 for 100, 150, and 200 iterations



(b) 16,384 x 16,384 for 100, 150, and 200 iterations



(c) 32,768 x 32,768 for 100, 150, and 200 iterations

Figure 7 Performance chart of Game of Life on different configurations. In all cases over different sizes of data, MIC clusters on Beacon achieved better performance than GPU clusters on Keeneland

© 2014 John Wiley & Sons Ltd

6 Conclusions

While many geospatial applications are data and computation intensive, geocomputation over the emerging heterogeneous computing infrastructure could be a promising solution to the big data challenge. Understanding the main features of the emerging advanced computer architectures and systems will help efficiently deploy the latest computing infrastructure and technology to advance GIScience research. Through this pilot study, three categories of geospatial computation benchmarks were implemented on CPU clusters, GPU clusters, and MIC clusters. The conclusions derived from this research could be beneficial to GIScientists who will build other geospatial applications with similar characteristics.

It seems that GPU clusters would have exemplified the advantage in the category of embarrassingly parallelism. This is reasonable, since GPU clusters could have sufficient and more computing threads to perform the calculation when the scale of data and computation could be within the memory limit. The latest generations of GPUs may have more memory on each GPU card, such as Tesla M2090 and K20/K20X that have up to 6 GB of memory. When the large data and the computation problem can be partitioned into multiple GPUs, the GPU cluster could have more potential to accelerate geospatial applications to achieve significant performance improvement compared with the traditional MPI + CPU parallel implementation as well as single-CPU implementations.

In the case of geospatial computation that has simple communication between the distributed computing nodes, especially when larger data are involved in iterative computation procedures, the simple MPI-native programming model on the Intel MIC cluster can achieve a performance equivalent to the MPI + GPU model on GPU clusters when the same number of processors are allocated. It is implied that an efficient cross-node communication network will be the key to achieve the strong scalability for parallel applications running on multiple nodes.

In the case of geospatial computation that has intensive data exchange and communication between the distributed computing nodes, it seems that the simple MPI-native programming model on the Intel MIC cluster can achieve better performance than the MPI + GPU model on GPU clusters when the same number of processors is allocated. This is because data has to be copied back and forth between the host CPUs and the GPUs. When the spatial computation and simulation have to be implemented for a lot of iterations, the communication overhead between CPUs and GPUs could be prominent.

While NVIDIA's latest Kepler GPU (K20) is able to outperform its Fermi GPU for most applications without special performance tuning, K20's capability of direct cross-GPU communication may have the potential for GPU to outperform Intel's MIC processor when dealing with communication-intensive applications. On the Intel MIC architecture, although the direct support of MPI on each MIC core makes it straightforward to port MPI + CPU code to the MIC cluster while achieving significant performance improvement, a large amount of on-board memory is used for OS and MPI support. For this reason, a detailed comparison between the offload model and native model is worthwhile. Both the direct cross-GPU communication mechanism and the offloading model of MIC will be future research directions, while other types of geospatial computation should be explored and examined in such emerging heterogeneous computing infrastructure.

Appendix

1. URLs of system configuration and access of Keeneland and Beacon:
   Beacon: http://www.nics.tennessee.edu/beacon
   Keeneland: http://keeneland.gatech.edu/kids-quick-start

© 2014 John Wiley & Sons Ltd

2. Sample Make file and PBS script for the CA benchmark on Keeneland using 16 GPUs

Make file:
```
#!/bin/bash
nvcc -c kernel.cu
mpicc -o main main.c kernel.o -lcudart -limf -lm -L /sw/keeneland/cuda/4.0/linux
_binary/lib64 -I /sw/keeneland/cuda/4.0/linux_binary/include/
```
PBS script:
```
#!/bin/sh
#PBS -N xshi
#PBS -j oe
#PBS -A UT-NTNL0102
#PBS -M xuanshi@uark.edu
#PBS -l nodes=8:ppn=2:gpus=2,walltime=00:30:00

date
cd $PBS_O_WORKDIR

echo "nodefile="
cat $PBS_NODEFILE
echo "=end nodefile"

#which mpirun
mpirun -np 16 ./main

date

# eof
```

3. Sample Make file and shell script for the CA benchmark on Beacon using 16 MICs

Make file:
```
CC = mpiicc -mmic

MIC_PROG = kernel
MPI_PROG = main.c

ca_test : main.o kernel.o
   $(CC) main.o $(MIC_PROG).o -o ca_32768_200

main.o: $(MPI_PROG)
   $(CC) -c $(MPI_PROG) -o main.o

$(MIC_PROG).o: $(MIC_PROG).cpp
   $(CC) -c $(MIC_PROG).cpp -o $(MIC_PROG).o

clean:
   rm *.o
```
Shell script:
```
generate-mic-hostlist micnative 60 > machines
allmicput -t ca_32768_200
time micmpiexec -genvlist
LD_LIBRARY_PATH=/global/opt/intel/impi/4.1.0.024/mic/lib:$LD_LIBRARY_PATH
-machinefile machines -n 960 $TMPDIR/ca_32768_200 $PBS_O_WORKDIR/32768 >
200-960mic-32768.log
```

© 2014 John Wiley & Sons Ltd

References

Aaby B G, Perumalla K S, and Seal S K 2010 Efficient simulation of agent-based models on multi-GPU and multi-core clusters. In *Proceedings of the Third International ICST Conference on Simulation Tools and Techniques*, Torremolinos, Malaga, Spain

Beacon 2014 Beacon. WWW document, http://www.nics.tennessee.edu/beacon

Bernabe S, Lopez S, Plaza A, and Sarmiento R 2013a GPU implementation of an automatic target detection and classification algorithm for hyperspectral image analysis. *IEEE Geoscience and Remote Sensing Letters* 10: 221–5

Bernabe S, Sanchez S, Plaza A, Lopez S, Benediktsson J A, and Sarmiento R 2013b Hyperspectral unmixing on GPUs and multi-core processors: A comparison. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing 6: 1386–98

Cheng T, Zhang Y, Li D, and Wang Q 2010 A component-based design and implementation for parallel kriging library. In Proceedings of the Second International Conference on Information Science and Engineering, Nanjing, China

Clarke K C 2003 Geocomputation's future at the extremes: High performance computing and nanoclients. *Parallel Computing* 29: 1281–95

Cluster File Systems 2002 Lustre: A Scalable, High Performance File System. WWW document, http://www.cse.buffalo.edu/faculty/tkosar/cse710/papers/lustre-whitepaper.pdf

Crimi G, Mantovani F, Pivanti M,Schifano S F, and Tripiccione R 2013 Early experience on porting and running a lattice Boltzmann code on the Xeon-phi co-processor. *Procedia Computer Science* 18: 551–60

Gardner M 1970 Mathematical games: The fantastic combinations of John Conway's new solitaire game "life". *Scientific American* 223: 120–3

Emeliyanenko P 2013 Computing resultants on graphics processing units: Towards GPU accelerated computer algebra. *Journal of Parallel and Distributed Computing* 73: 1494–505

FEMA 2013 Limitations of the HAZUS-MH 2.0 Software. WWW document, http://www.fema.gov/media-library-data/20130726-1759-25045-8653/hazus2_limitations.pdf

Heinecke A, Vaidyanathan K, Smelyanskiy M, Kobotov A, Dubtsov R, Henry G, Shet A G, Chrysos G, and Dubey P 2013 Design and implementation of the linpack benchmark for single and multi-node systems based on Intel Xeon phi coprocessor. In *Proceedings of the Twenty-seventh IEEE International Symposium on Parallel and Distributed Processing*, Boston, Massachusetts: 126–37

Huang M, Men L, and Gauch J 2013 Accelerating mean shift segmentation algorithm on hybrid CPU/GPU platforms. In Shi X; Kindratenko V, and Yang C (eds) *Modern Accelerator Technologies for Geographic Information Science*. Berlin, Springer: 157–68

Jeffers J and Reinders J 2013 *Intel Xeon Phi Coprocessor High Performance Programming*. Oxford, UK, Elsevier

Jenson J R 1999 *Introductory Digital Image Processing: A Remote Sensing Perspective* (Second Edition). Englewood Cliffs, NJ, Prentice-Hall

Kalyanapu A J, Shankar S, Pardyjak E R, Judi D R, and Burian S J 2011 Assessment of GPU computational enhancement to a 2D flood model. *Environmental Modelling and Software* 26: 1009–16

Keeneland 2014 Keeneland: National Institute for Experimental Computing. WWW document, http://keeneland.gatech.edu/

Li J, Humphrey M, Agarwal D, Jackson K, and van Ingen C, and Ryu Y 2010 eScience in the cloud: A MODIS satellite data re-projection and reduction pipeline in the Windows Azure platform. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium*, Atlanta, Georgia

NCSA 2009 MAEviz Building Damage Tutorial. WWW document, https://wiki.ncsa.illinois.edu/display/MAE/MAEviz+Building+Damage+Tutorial

Oliver M A and Webster R 1990 Kriging: A method of interpolation for geographical information systems. *International Journal of Geographical Information Science* 4: 313–32

Preis T, Virnau P, Paul W, and Schneider J J 2009 GPU accelerated Monte Carlo simulation of the 2D and 3D Ising model. *Journal of Computational Physics* 228: 4468–77

Schmidl D, Cramer T, Wienke S, Terboven C, and Müller M S 2013 Assessing the performance of OpenMP programs on the Intel Xeon Phi. In Wolf F, Mohr B, and Mey D (eds) *Euro-Par 2013: Parallel Processing*. Berlin, Springer Lecture Notes in Computer Science Vol. 8097: 547–58

Shi X and Ye F 2013 Kriging interpolation over heterogeneous computer architectures and systems. *GIScience and Remote Sensing* 50: 196–211

Simion B, Ray S, and Brown A D 2012 Speeding up spatial database query execution using GPUs. *Procedia Computer Science* 9: 1870–79

Steinbach M and Hemmerling R 2012 Accelerating batch processing of spatial raster analysis using GPU. *Computers and Geosciences* 45: 212–20

© 2014 John Wiley & Sons Ltd

Srinivasan B V, Duraiswami R, and Murtugudde R 2010 Efficient kriging for real-time spatio-temporal interpolation. In *Proceedings of the Twentieth Conference on Probability and Statistics in the Atmospheric Sciences*, Atlanta, Georgia

Tang W and Bennett D A 2012 Parallel agent-based modeling of spatial opinion diffusion accelerated using graphics processing units. *Ecological Modelling* 229: 108–18

Ye F and Shi X 2013 Parallelizing ISODATA algorithm for unsupervised image classification on GPU. In Shi X, Kindratenko Y, and Yang C (eds) *Modern Accelerator Technologies for Geographic Information Science*. Berlin, Springer: 145–56

Ye F, Shi X, Wang S, Liu Y, and Han S Y 2011 Spherical interpolation over graphic processing units. In *Proceedings of the Second International Workshop on High Performance and Distributed Geographic Information Systems*, Chicago, Illinois: 38–41

Zhang J, You S, and Gruenwald L 2010 Indexing large-scale raster geospatial data using massively parallel GPGPU computing. In *Proceedings of the Eighteenth SIGSPATIAL International Conference on Advances in Geographic Information Systems*, San Jose, California: 450–3

Zhang J, You S, and Gruenwald L 2011 Parallel quadtree coding of large-scale raster geospatial data on GPGPUs. In *Proceedings of the Nineteenth ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, Chicago, Illinois: 457–60

Zhao Y, Padmanabhan A, and Wang S 2013 A parallel computing approach to viewshed analysis of large terrain data using graphics processing units. *International Journal of Geographical Information Science* 27: 363–84

© 2014 John Wiley & Sons Ltd