2013 IEEE International Conference on High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing

Accelerating Geospatial Applications on Hybrid Architectures

Chenggang Lai, Miaoqing Huang
CSCE Department
University of Arkansas
Fayetteville, AR 72701, USA
Email: {cl004,mqhuang}@uark.edu

Xuan Shi
Department of Geosciences
University of Arkansas
Fayetteville, AR 72701, USA
Email: xuanshi@uark.edu

Haihang You
National Institute for Computational Sciences
University of Tennessee
Oak Ridge, TN 37831, USA
Email: hyou@utk.edu

***Abstract*—Accelerators have become critical in the process to develop supercomputers with exascale computing capability. In this work, we examine the potential of two latest acceleration technologies, Nvidia K20 Kepler GPU and Intel Many Integrated Core (MIC) Architecture, for accelerating geospatial applications. We first apply a set of benchmarks under 3 different configurations, i.e., MPI+CPU, MPI+GPU, and MPI+MIC. This set of benchmarks include embarrassingly parallel application, loosely communicating application, and intensely communicating application. It is found that the straightforward MPI implementation on MIC cores can achieve the same amount of performance speedup as hybrid MPI+GPU implementation when the same number of processors are used. Further, we demonstrate the potentials of hardware accelerators for advancing the scientific research using an urban sprawl simulation application. The parallel implementation of the urban sprawl simulation using 16 Tesla M2090 GPUs can realize a 155× speedup compared with the single-node implementation, while achieving a good strong scalability.**

I. Introduction

Performance acceleration technologies have been adopted in many supercomputers, e.g., Titan, Stampede, and Tianhe-2, mainly for two purposes: (1) improving the performance, and (2) reducing the overall power consumption [1]. As GPUs are becoming ubiquitous in HPC, numerous applications have been ported to GPU-based systems over the past several years, including large scale scientific applications on GPU clusters [2]–[4]. Various performance optimization techniques have been developed in the past, including tiling, privatization, scatter to gather conversion, binning, regularization, compaction, data layout transformation, granularity coarsening, etc [5]. In the meantime, a couple of languages have been developed to support the programming on GPU, such as Nvidia CUDA [6] and OpenCL [7]. In order to remove the obstacle for porting parallel application to hybrid CPU/GPU system, OpenACC Application Program Interface [8] provides a collection of compiler directives to specify loops and regions of code in standard C, C++ and Fortran to be offloaded from a host CPU to an attached accelerator, providing portability across operating systems, host CPUs and accelerators.

In geospatial applications, a few initiatives utilized a single GPU to build spatial index [9], [10] or calculate the viewshed to determine the locations over digital terrain surface visible to a set of observer features [11]. In processing satellite images, most of prior works deployed a single GPU, such as unsupervised image classification [12], segmentation [13], and hyperspectral data analytics [14], [15]. In geospatial simulation, agent based modeling has been implemented on individual GPUs [16] or multiple GPUs without communication between the GPUs [17]. Spatial computation by deploying multiple GPUs through combined MPI and CUDA programs were reported with a focus on interpolation using IDW and Kriging algorithms [18], [19].

High-resolution geospatial data has become increasingly available along with an accelerating increase in data volume. For example, satellites now can generate high-quality images with a resolution comparable to aerial photos, i.e., < 0.5 meter. While Light Detection And Ranging (LiDAR) technology has been extensively used in survey, digital elevation model (DEM) derived from LiDAR has dramatically increased the resolution to represent and analyze the terrain surface. Such high-resolution data is critical in a variety of applications, such as transportation planning, hydrological network and watershed analysis, environmental modeling and surveillance, emergency response, and military operations. However, high-resolution data also raises a variety of challenges and complexities along with the increasing data scalability. Although it is expected that data can be exploited effectively for timely delivery of accurate information and for knowledge discovery, traditional desktop-based geographic information system (GIS) and remote sensing softwares have become inefficient and impossible to process large-scale high-resolution geospatial data.

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 parallelism to achieve scalability with high performance for data intensive computing over high-resolution spatial data. In this work, we demonstrate that hybrid computer clusters equipped with the latest GPU and Intel MIC processors can achieve a significant performance improvement for a range of important geospatial applications, such as Kriging interpolation, ISODATA, Cellular Automata, and urban sprawl simulation. It is also found that the simple MPI+accelerator programming model is very efficient for both GPU and Intel MIC when porting traditional MPI based applications to hybrid platforms.

The remainder of this paper is organized as follows. The latest accelerator architectures, including Nvidia Kepler GPU and Intel MIC, and their programming models are discussed in Section II. A comprehensive comparison of hybrid computer

978-0-7695-5088-6/13 $31.00 © 2013 IEEE
DOI 10.1109/HPCC.and.EUC.2013.217

(a) Kepler architecture. (b) SMX architecture.

Fig. 1. Nvidia's Kepler GPU architecture.

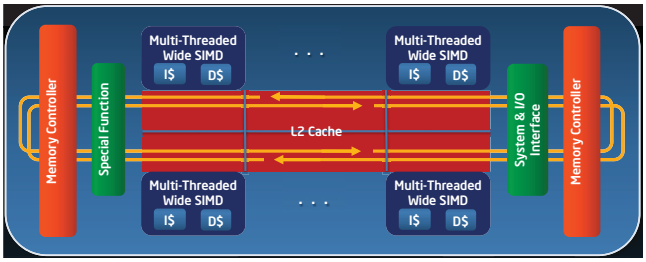


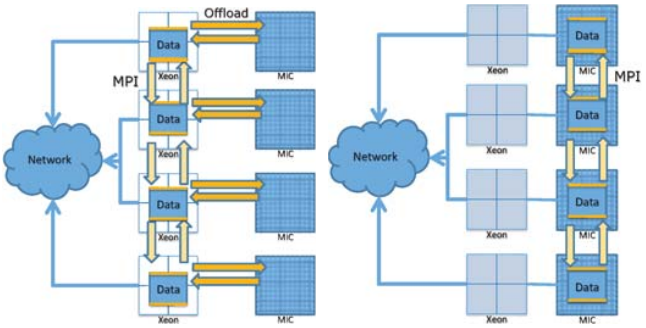
Fig. 2. The architecture of Intel Xeon Phi coprocessor (MIC).

cluster using different acceleration technologies is presented in Section III, in which three representative geospatial applications are adopted as benchmarks. In Section IV we further demonstrate the benefit of the latest acceleration technologies by presenting an urban sprawl simulation case. Finally, we give the conclusion marks in Section V.

II. Coprocessor Architecture

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 (i.e., Hyper-Q), (2) flexibly create new kernels on GPU (i.e., Dynamic Parallelism), and (3) reduce communication overhead across GPUs through GPUDirect [20]. GPUs are typically used as accelerators in high-performance computer clusters. In a typical MPI-based parallel application, the MPI process executes on a host CPU, which allocates the computation to one or more client GPUs.

The first commercially available Intel coprocessor based on Many Integrated Core architecture is Xeon Phi, as shown in Figure 2. Xeon Phi contains up to 61 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 3(a), the MPI processes will be hosted by CPUs, which will offload the computation to the MIC processors. Multithreading programming models



(a) Offloading. (b) Direct host.

Fig. 3. Two different approaches to implementing parallelism on MIC cluster.

such as OpenMP can be used to allocate many cores for data processing. The second approach, as shown in Figure 3(b), is to let each MIC core directly host one MPI process. In this way, the 60 cores on the same die are treated as 60 independent processors while sharing the 8 GB on-board memory on the Xeon Phi 5110P.

III. A Direct Comparison of Hybrid Architectures

A. Benchmarks

Three use cases in geospatial applications are adopted as three different types of benchmarks in this pilot study.

1) Embarrassingly parallel case: Kriging Interpolation: Kriging is a geostatistical estimator that infers the value of a random field at an unobserved location [21]. Kriging is based on the idea that the value at an unknown point should be the average of the known values at its neighbors.

Kriging can be viewed as a point interpolation which 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 as in (1).

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

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 so that the summation is over k nearest neighbors of the estimated point. This reduction in calculation is due to the fact that the farther the sampled point is from the estimated point, the less impact it has in the summation. For example, the commercial software ArcGIS [22] uses 12 nearest points (i.e., $k = 12$) in the Kriging calculation by default. In this benchmark, embarrassingly parallelism can be realized since the interpolation calculation over each cell has no dependency on the others.

2) Loose communication case: ISODATA: The Iterative Self-Organizing Data Analysis Technique Algorithm (ISODATA) is one of the most frequently used algorithms for unsupervised image classification algorithms in remote sensing applications [23]. 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. When multiple processors are utilized, only one summation from all processors is required in each iteration.

3) Intense communication case: Cellular Automata: Cellular Automata (CA) are the foundation for geospatial modeling and simulation. Game of Life (GOL) [24], invented by British mathematician John Conway, is a well-known generic Cellular Automaton that consists of a collection of cells that can live, die or multiply based on a few mathematical rules.

The universe of the Game of Life is a two-dimensional orthogonal grid of square cells, each of which is in one of two possible states, alive ('1') or dead ('0'). Every cell interacts with its eight neighbors, which are the cells that are horizontally, vertically, or diagonally adjacent. At each step in time, the following transitions occur:

- Any live cell with fewer than two live neighbors dies, as if caused by under-population.
- Any live cell with two or three live neighbors lives on to the next generation.
- Any live cell with more than three live neighbors dies, as if by overcrowding.
- Any dead cell with exactly three live neighbors becomes a live cell, as if by reproduction.

When the data are partitioned and processed by multiple processors, data at the boundary rows have to be exchanged between different processors in each iteration of Game of Life calculation.

B. Experimental Platform

We conducted our experiments on two platforms, the NSF sponsored Keeneland supercomputer [25] and Beacon supercomputer [26].

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 2 6-core 2.8 GHz Xeon CPUs and 3 Tesla M2090 GPUs. The Nvidia 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 compute nodes and 6 I/O nodes joined by FDR InfiniBand interconnect providing 56 Gb/s of bi-directional bandwidth. Each compute node is equipped with 2 Intel Xeon E5-2670 8-core 2.6 GHz processors, 4 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. Thus, 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, in aggregate.

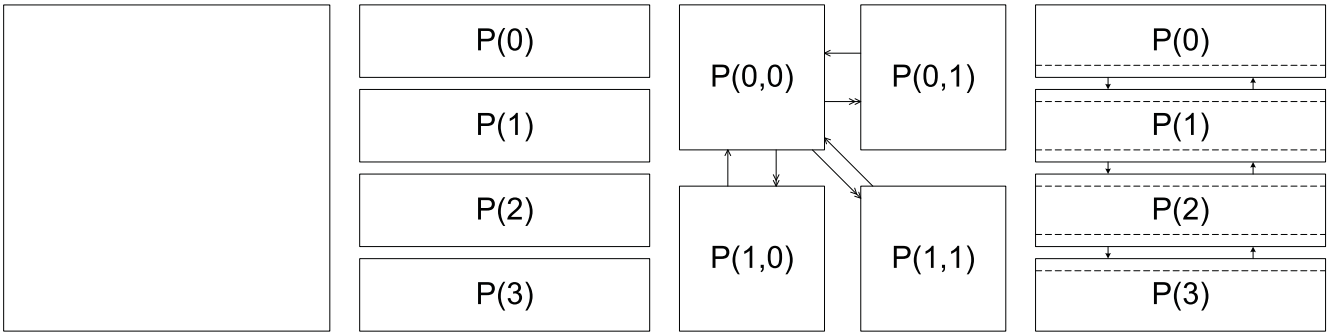
Both platforms are equipped with Lustre parallel distributed file system, in which a technique called *file striping* can be applied to significantly improve the I/O performance. File striping is a process in which Lustre automatically divides data into chunks and distributes them across multiple object 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 OST 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 that a file is split into and distributed across OSTs.

C. Implementation and Results

For each benchmark, we have three parallel implementations on two clusters as follows.

- MPI+CPU: MPI-based parallel implementation on Keeneland. The Intel Xeon E5 8-core CPU is used for data processing. Each MPI process runs on a single CPU. The process on CPU is a single-thread process. If m MPI processes are created in the parallel application, m CPU processors are allocated.
- MPI+GPU: MPI-based parallel implementation 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 m MPI processes are created in the parallel application, m CPU processors and m GPU processors are allocated. The host CPU is responsible for the MPI communication; and the client GPU is in charge of data processing. We do not try to use the GPUDirect technique to reduce the cross-GPU communication so that the MPI+GPU implementation has the best portability.
- MPI+MIC: MPI-based parallel implementation on Beacon. The Intel Xeon Phi 5100P is used for data processing. In this implementation, each MIC core will directly host one MPI process, as shown in Figure 3(b). Therefore, if m Xeon Phi coprocessors are used, $m \times 60$ MPI processes are created in the parallel implementation. This programming model on Intel MIC cluster has the better portability than the offloading model shown in Figure 3(a). Legacy parallel code using MPI can be directly compiled and executed on systems with Intel MIC without much tuning.

We want to show the strong scalability of the parallel implementations. Therefore, the problem size is fixed for each benchmark while the number of participating MPI processes is increased.



(a) Source data. (b) Kriging interpolation. (c) ISODATA. (d) Game of Life.

Fig. 4. Data partition and communication in three benchmarks.

TABLE I. PERFORMANCE OF KRIGING INTERPOLATION OF THREE PARALLEL IMPLEMENTATIONS (UNIT: *second*).

Number of Processors*	Keeneland KIDS								Beacon			
	MPI+CPU				MPI+GPU				MPI+MIC			
	Read	Comp.†	Write	**Total**	Read	Comp.†	Write	**Total**	Read	Comp.†	Write	**Total**
2	0.03	1,054.99	1.13	**1,056.15**	0.03	6.77	0.90	**7.69**	1.24	232.43	12.24	**245.90**
4	0.03	528.43	0.98	**529.44**	0.03	3.92	0.99	**4.93**	1.27	116.34	16.44	**134.05**
8	0.03	260.60	0.95	**261.57**	0.03	2.76	0.94	**3.73**	1.23	46.48	54.43	**102.14**‡
16	0.03	129.61	1.16	**130.69**	0.03	2.17	1.11	**3.31**	1.31	36.74	300.23	**338.28**‡

*The processor can be a CPU, a GPU, or Intel MIC in a corresponding implementation.
†The computation time includes both the time spent on data processing and the time spent on communication.
‡Only 360 or 720 MIC cores are used in the computation for 8 or 16 processors, respectively.

1) Kriging Interpolation: In the Kriging interpolation benchmark, the source dataset as shown in Figure 4(a) is evenly partitioned among all MPI processes along the row-major order as shown in Figure 4(b), in which we use 4 processes as an example. The computation in each MPI process is purely local, i.e., there is no cross-process communication.

The problem size of this benchmark is 171 MB, consisting of 4 datasets with the respective sizes of 29 MB, 37 MB, 48 MB, and 57 MB. Each dataset has 2,191, 4,596, 6,941, and 9,817 sample points, respectively. The output raster grid for each dataset has a consistent dimension of 1,440×720. The value of the remaining unsampled locations in the output grid needs to be estimated using those sample points. In our experiments, the value of an unsampled location will be estimated using the values of the 10 closest sample points, i.e., $k = 10$.

These 4 datasets are processed in a sequence. For each dataset, it is evenly distributed among MPI processes. The total execution time of the benchmark under various cases is listed in Table I. As mentioned at the beginning of Section III-C, the number of MPI processes is different while the same number of processors are used. On Keeneland, if m processors (either CPU or GPU) are allocated, m MPI processes are generated correspondingly. On Beacon, each MIC core will directly host an MPI process. Therefore, $m \times 60$ MPI processes are generated. For each output raster grid, the generation of the 720 columns is evenly distributed among the MPI processes. Therefore, only 360 or 720 MPI processes, which execute on 360 or 720 MIC cores, are created when 8 or 16 MIC processors are allocated for the computation, respectively.

We do not have access to a computer cluster with Nvidia Kepler GPU. In order to project the performance of a hybrid cluster with Kepler GPU (e.g., K20), we compare the performance of the same single-GPU implementation of Kriging interpolation on both M2090 and K20. The results are shown in Table II. Based on the performance on the single-node implementation, we assume that the K20 is able to achieve a 2.39×

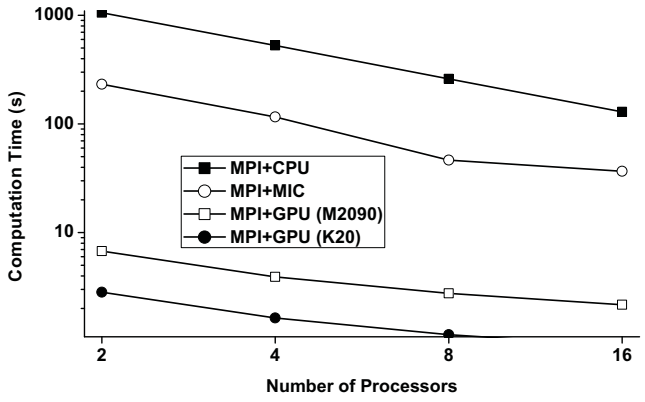


Fig. 5. Performance of Kriging interpolation on four different configurations.

TABLE II. COMPUTATION TIME COMPARISON OF SINGLE-GPU IMPLEMENTATION ON M2090 AND K20 (UNIT: *second*).

GPU	Kriging Interpolation	ISODATA*	Game of Life		
			8,192×8,192	16,384×16,384	32,768×32,768
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**

*The image size is reduced to 10,000×10,000×3 to fit the on-board memory of the GPU.

TABLE III. PERFORMANCE OF ISODATA OF THREE PARALLEL IMPLEMENTATIONS (UNIT: *second*).

Number of Processors	Keeneland KIDS						Beacon		
	MPI+CPU			MPI+GPU			MPI+MIC		
	Read	Comp.	**Total**	Read	Comp.	**Total**	Read	Comp.	**Total**
36	6.04	232.22	**238.26**	3.91	100.26	**104.17**	NA*		
64	12.60	140.00	**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 parallel implementation on Beacon cluster with 36/64 MIC processors cannot be done due to the limit of on-board memory of MIC processor.

speedup without any particular tuning for this benchmark.

Figure 5 illustrates the performance comparison among 4 different implementations for the Kriging interpolation. The performance of the GPU cluster with K20 is projected based on the speedup of the single K20 vs. M2090 and we assume that the other specifications of the K20 GPU cluster is same to the Keeneland KIDS. We exclude the read time for source data and the write time for result data in the comparison due to the fact that both times can become unstable on both Keeneland and Beacon for three benchmarks. Therefore, we only include the computation time, which includes both the time spent on data processing and the time spent on cross-processor communication. From this figure, it can be found that all hybrid implementation can easily outperform the parallel implementation on CPU. For this embarrassingly parallel benchmark, the stream processing architecture on Nvidia GPU works quite well and is able to outperform the parallel implementation on Intel MIC, which employs relatively low-performance cores.

2) ISODATA: The input of the ISODATA is a high-resolution image of 18 GB with a dimension of 80,000×80,000 for three bands. The objective of this benchmark is to classify the image into n classes. In this work, $n = 15$. 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. In the example shown in Figure 4(c), the whole image will be processed by 4 processors, therefore, it is divided into 4 blocks.

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 (shown as $\rightarrow$ in Figure 4(c)) their local means to the head MPI process (i.e., P(0,0)). After the head MPI process collects all the local means, it calculates the global means for the 15 classes and returns (shown as $\twoheadrightarrow$) them to all other MPI processes. Then all the MPI processes start the computation of the next iteration.

The performance of the ISODATA benchmark on three different platforms is shown in Table III. When the MPI-direct-host 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 for ISODATA.

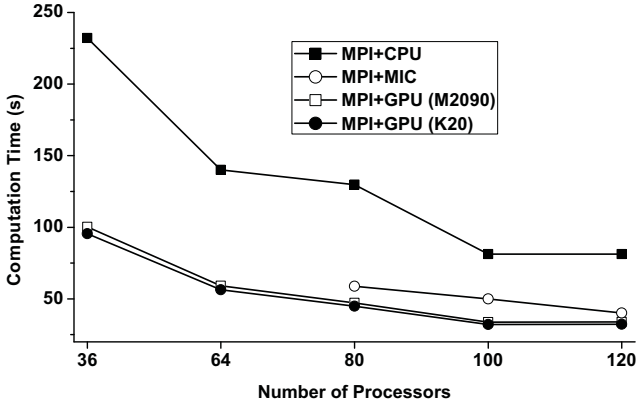


Fig. 6. Performance of ISODATA on four different configurations.

From the results in Table II, it is found that K20 barely outperforms M2090 with a 1.05× speedup.

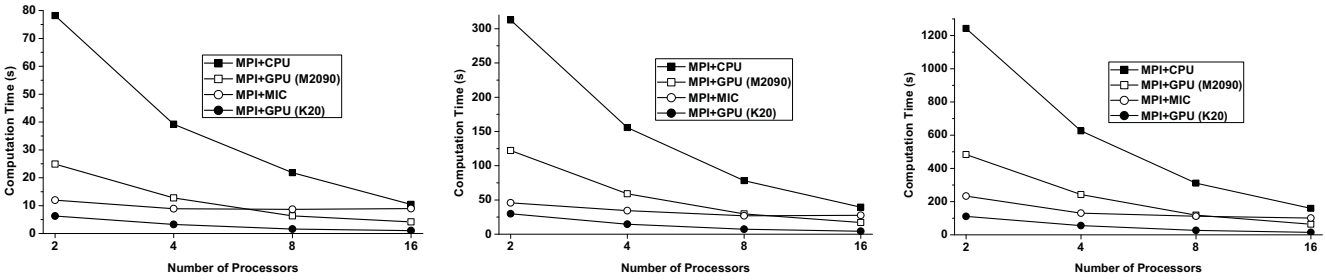
The performance comparison of various implementations of ISODATA is shown in Figure 6. It can be found that the gap between the MIC processor and GPUs becomes quite small. One reason is that FDR InfiniBand network on Beacon provides much higher bandwidth than the QDR InfiniBand network on Keeneland KIDS. The advantage of more efficient communication network on Beacon is further demonstrated when the number of participating processors is increased from 100 to 120. On Keeneland, the computation time using 120 processors is almost same to the time using 100 processors because the decrease in the data processing time is offset by the increase in the communication time. On Beacon, we still observe the strong scalability when 120 MIC processors are used in computation.

3) Conway's Game of Life: Game of Life is a generic Cellular Automata program, in which the status of each cell is dependent upon its eight neighbors. In this benchmark, the status of each cell in the grid will be updated for 100 iterations. In each iteration, the statuses of all cells are updated simultaneously. In order to parallelize the updating process, the cells in the square grid are partitioned into stripes along the row-wise order. Each stripe is handled by one MPI process. At the beginning of each iteration, each MPI process needs to send the statuses of the cells along the boundaries of each stripe to its neighbor MPI processes and receive the statuses of the cells of two adjacent rows as shown in Figure 4(d).

Three different grid sizes are tested, i.e., 8,192×8,192, 16,384×16,384, and 32,768×32,768. By observing the performance results in Table IV, it can be found that the strong

TABLE IV. PERFORMANCE OF GAME OF LIFE OF THREE PARALLEL IMPLEMENTATIONS (UNIT: *second*).

Number of Processors	8,192×8,192			16,384×16,384			32,768×32,768		
	MPI+CPU	MPI+GPU	MPI+MIC	MPI+CPU	MPI+GPU	MPI+MIC	MPI+CPU	MPI+GPU	MPI+MIC
2	78.15	24.92	11.97	312.69	122.19	45.87	1242.19	483.58	233.28
4	39.20	12.79	8.91	155.64	59.14	34.43	625.92	242.43	130.55
8	21.82	6.30	8.65	78.14	29.66	26.88	311.34	118.31	112.24
16	10.41	4.15	8.93	39.35	17.20	27.54	159.05	63.83	100.50



(a) 8,192×8,192. (b) 16,384×16,384. (c) 32,768×32,768.

Fig. 7. Performance of Game of Life on four different configurations.

scalability is demonstrated for MPI implementations on both CPUs and GPUs. For this benchmark, the K20 can consistently outperform the M2090 by $\sim$4$\times$ based on the results in Table II. For the MPI+MIC implementation, it is found that the performance does not scale quite well due to the communication overhead among MPI processes. The number of MPI processes on m MIC processors is $m \times 60$. Based on the results shown in Table IV and Figure 7, there is not much performance gain when increasing the number of MIC processors from 4 to 8 and 16. When the grid is partitioned into $m \times 60$ MPI processes on m MIC processors, the performance gain from the reduced workload on each MIC core is easily offset by the increase of the communication cost among the cores. Therefore, it is critical to keep a balance between computation and communication for achieving the best performance.

Through the above three cases, it has been clearly illustrated that the parallel implementations of geospatial applications on hybrid platforms have an evident advantage than the parallel implementations on CPUs. In the following section, we use a urban sprawl simulation application to further demonstrate the benefit of hybrid architectures. Because the benchmarking results have shown that the MIC cluster can provide the computing capability equivalent to the GPU cluster when the same number of processors are used, we only implement the urban sprawl simulation on the GPU cluster.

IV. ACCELERATING URBAN SPRAWL SIMULATION

Among varieties of approaches to simulating urban growth, Cellular Automata (CA) have been extensively applied to understand the dynamics of land use and land cover change. This pilot study is based on a prior work [27] that developed a stochastic CA model for urban sprawl simulation. According to the original model designer, the stochastic CA model was implemented by a procedure that calibrates the initial global probability surface from sequential land use data and then modifies the global probability with the local probability, which is updated in each iteration of simulation.

The purpose of calibration is to extract the coefficients

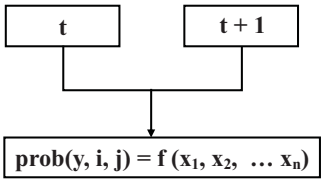


Fig. 8. Calibration of the global probability surface from sequential land use data.

or parameter values of the rules from the observation of land use pattern at time t and $t+1$, as shown in Figure 8. Mathematically, this is generalized as the estimation of the probability of particular state transition y occurring at a particular location (i, j) through a function of development factors $(x_1, x_2, \ldots, x_n)$.

Furthermore, the probability of site selection is incorporated into the joint probability to constrain the quantity of simulated conversions to projected land demand. In the case of land use changes, for each year of simulation, a logistic model can be developed to calculate the probability of whether a pixel is developed into urban land use type or remaining in the current state. Thus the probability of a site experiencing land conversion can be computed as (2).

$$p_g(S_{ij} = urban) = \frac{exp(Z)}{1 + exp(Z)} = \frac{1}{1 + exp(-Z)}, \quad (2)$$

where p_g is the observed global probability, S_{ij} is the state of the cell (i, j). The joint probability can be calculated as the product of global probability, cell constraint, and neighborhood potentiality. Cell constraint refers to factors that exclude land development on the cells such as a body of water, a mountainous area and planning restriction zones. It is possible to use an evaluation score of land suitability instead of a binary one (suitable/unsuitable). The joint probability is stated in (3).

$$p_c^t = p_g con(S_{ij}^t = suitable)\Omega_{ij}^t, \quad (3)$$

where $con()$ converts the state of suitable land into a binary

TABLE V. PERFORMANCE OF URBAN SPRAWL SIMULATION (UNIT: *second*).

Platform: Keeneland KIDS (problem size: 23,852×40,461×7)															
No. of	1993–2002			1993–2012			1993–2022			1993–2032			1993–2042		
Proc.	Read	Comp.	**Total**	Read	Comp.	**Total**	Read	Comp.	**Total**	Read	Comp.	**Total**	Read	Comp.	**Total**
4	38.65	72.85	**111.47**	27.24	123.67	**150.88**	76.13	210.53	**286.63**	15.89	304.80	**320.66**	28	355.42	**383.38**
8	4.06	37.12	**41.16**	14.01	72.81	**86.79**	94.50	107.99	**202.47**	7.49	143.42	**150.88**	6.38	178.46	**184.82**
16	2.75	19.13	**21.86**	11.29	37.43	**48.69**	1.86	54.40	**56.24**	11.57	72.56	**84.09**	4.27	87.59	**91.83**
Single-CPU Implementation (problem size: 23,852×40,461×7)															
Core i7	47.20	2,492.69	**2,539.89**	47.26	5,636.26	**5,683.52**	46.38	8,283.61	**8,329.99**	47.05	11,061.32	**11,108.37**	46.97	13,600.04	**13,647.01**
Single-GPU Implementation (problem size: 10,000×10,000×7)															
M2090	7.18	25.35	**32.44**	5.28	45.76	**51.04**	7.24	74.18	**81.42**	5.24	86.95	**92.19**	5.14	118.10	**123.24**
K20	4.40	27.65	**32.05**	4.40	54.57	**58.97**	4.41	78.63	**83.04**	4.43	103.65	**108.08**	4.42	134.76	**139.18**

variable. Again, please note that the joint probability p_c is denoted with time t, indicating it changes along with iterations.

In this model, the neighborhood function is calculated in a conventional ad hoc way. As a neighborhood evaluation function, the neighborhood potentiality of cell transition is defined in (4).

$$\Omega_{ij}^t = \frac{\sum_{3\times 3} con(s_{ij} = urban)}{3 \times 3 - 1}. \quad (4)$$

Based on the joint probability, a Monte Carlo process is applied in the calculation of the scaling of probability defined in (5).

$$p_s^t(ij) = \frac{qp_t^t(ij)}{\sum_{ij} p_t^t(ij)}, \quad (5)$$

where q is the number of cells to be converted according to projected land conversion at each iteration. The value of the right-hand should be limited at 1.0 to ensure that the probability is within the range of 0 to 1. This transformation in fact constitutes an additional constraint to the joint probability. As a result, the scaled probability is composed of three probabilities: (1) the probability of development measured on global factors, (2) the probability of development measured on local factors, and (3) the probability of cell selection according to the projected land demand.

The grid $p_s^t(ij)$ is then directly compared with a random grid with uniform distribution from 0 to 1 to decide whether the cell is to be converted at time $t+1$ as in (6).

$$S^{t+1}(ij) = \begin{cases} urban, & p_s^t(ij) > rand(ij) \\ rural, & p_s^t(ij) \leqslant rand(ij) \end{cases}, \quad (6)$$

where $rand(ij)$ is a uniform 0-1 random distribution grid as the output result.

When large scale datasets are applied, it is time consuming to complete such a complex model, or it could be an impossible task if the computer does not have sufficient memory to hold the input data, intermediate processing outcome, and the final output results.

A. Implementation and Results

The input image to this application is southern California taken in 1993. It has a dimension of 23,852×40,461 for 7 bands. In order to project the urban growth in n years, n iterations of simulation need to be carried out. When multiple processors are used in the parallel implementation, the data partition is similar to the case in Figure 4(d), i.e., each image is divided into multiple stripes along the row-major order. It is also a densely communicating parallel application.

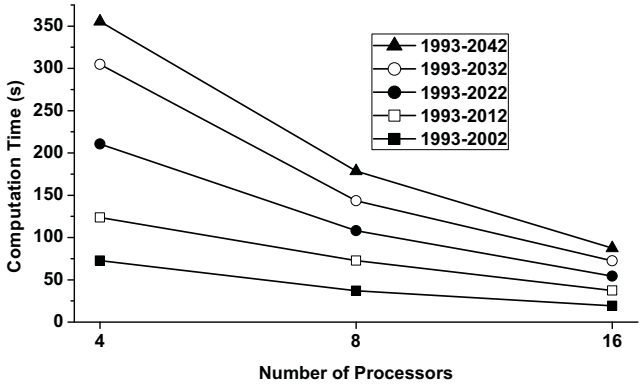


Fig. 9. Performance of parallel implementations of urban sprawl simulation.

This urban sprawl simulation is implemented on Keeneland using MPI+GPU mode. 5 different simulations are carried out from 10 years projection to 50 years projection. In each simulation, we test the scalability of the parallel implementation by using 4, 8, and 16 M2090 GPUs. The results are shown in Table V. Like the case before, it is found that the source image read time is not stable on Keeneland, varying dramatically from one run to another run. Therefore, we verify the strong scalability using the computation time only, which includes both data processing and data communication time. As shown in Figure 9, the MPI+GPU parallel implementation demonstrates a good scalability for all cases. For this application, we also conduct a performance comparison between K20 and M2090 by running single-GPU implementation. The results in Table V show that K20 is even slower than M2090 for this benchmark.

Many current geospatial analysis applications and softwares are still implemented on a single CPU. Therefore, we also execute the urban sprawl simulation on a single workstation, which hosts an Intel Core i7-930 quad-core CPU at 2.8 GHz with 18 GB main memory. It is found that the parallel implementation with 16 GPUs can achieve a 155× speedup than the implementation on the workstation. The significant performance improvement of geospatial applications from the parallel implementation on hybrid architectures will dramatically advance the scientific research efficiency and accuracy in this domain.

V. Conclusions

In this work we demonstrate the potential for accelerating geospatial applications using parallel implementation on hybrid computer clusters. MPI+GPU and MPI+MIC parallel implementations of representative geospatial applications achieve significant performance improvement compared with the traditional MPI+CPU parallel implementation as well as single-CPU implementation. It is also found that the simple MPI-direct-host programming model on Intel MIC cluster can achieve a performance equivalent to the MPI+GPU model on GPU clusters when the same number of processors are allocated. An efficient cross-node communication network is still the key to achieve the strong scalability for parallel applications running on multiple nodes.

It is also worth mentioning that the latest Kepler GPU is able to outperform the Fermi GPU for most applications without special performance tuning. However, the direct cross-GPU communication will be critical for GPU to outperform Intel MIC processor when dealing with densely communicating applications. On Intel MIC architecture, although the direct support of MPI on each MIC core makes it straightforward to port MPI+CPU code to MIC cluster while achieving significant performance improvement, a lot of on-board memory is used for OS and MPI support. Therefore, a detailed comparison between the offloading model and direct host model is worthwhile.

Acknowledgments

This 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 GPU donations.

References

[1] B. Dally, "Power, programmability, and granularity: The challenges of exascale computing," in *Proc. 2011 IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, May 2011, p. 878.

[2] C. Yang, F. Wang, Y. Du, J. Chen, J. Liu, H. Yi, and K. Lu, "Adaptive optimization for petascale heterogeneous CPU/GPU computing," in *Proc. IEEE International Conference on Cluster Computing (Cluster 2010)*, Sep. 2010, pp. 19–28.

[3] P. Jetley, L. Wesolowski, F. Gioachin, L. V. Kalé, and T. R. Quinn, "Scaling hierarchical N-body simulations on GPU clusters," in *Proc. 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC'10)*, Nov. 2010, pp. 1–11.

[4] S. S. Hampton, S. R. Alam, P. S. Crozier, and P. K. Agarwal, "Optimal utilization of heterogeneous resources for biomolecular simulations," in *Proc. 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC'10)*, Nov. 2010, pp. 1–11.

[5] J. Stratton, N. Anssari, C. Rodrigues, I.-J. Sung, N. Obeid, L. Chang, G. D. Liu, and W. mei Hwu, "Optimization and architecture effects on GPU computing workload performance," in *Proc. 2012 Innovative Parallel Computing: Foundations & Applications of GPU, Manycore, and Heterogeneous Systems (InPar2012)*, May 2012, pp. 1–10.

[6] *NVIDIA CUDA C Programming Guide 5.0*, NVIDIA Corporation, Oct. 2012.

[7] Khronos OpenCL Working Group, *OpenCL 1.2 Specification*, Khronos Group, Nov. 2012.

[8] *http://www.openacc.org/*.

[9] J. Zhang, S. You, and L. Gruenwald, "Indexing large-scale raster geospatial data using massively parallel GPGPU computing," in *Proc. 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS'10)*, Nov. 2010, pp. 450–453.

[10] ——, "Parallel quadtree coding of large-scale raster geospatial data on GPGPUs," in *Proc. 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems (GIS'11)*, Nov. 2011, pp. 457–460.

[11] Y. Zhao, A. Padmanabhan, and S. Wang, "A parallel computing approach to viewshed analysis of large terrain data using graphics processing units," *International Journal of Geographical Information Science*, vol. 27, no. 2, pp. 363–384, Feb. 2013.

[12] F. Ye and X. Shi, "Parallelizing isodata algorithm for unsupervised image classification on gpu," in *Proc. International Workshop on Modern Accelerator Technologies for GIScience (MAT4GIScience 2012)*, Sep. 2012.

[13] L. Men, M. Huang, and J. Gauch, "Accelerating mean shift segmentation algorithm on hybrid CPU/GPU platforms," in *Proc. 2012 International Workshop on Modern Accelerator Technologies for GIScience*, Sep. 2012.

[14] S. Bernabé, S. Sánchez, A. Plaza, S. López, J. A. Benediktsson, and R. Sarmiento, "Hyperspectral unmixing on GPUs and multi-core processors: A comparison," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 6, no. 3, pp. –, Jun. 2013.

[15] S. Bernabé, S. López, A. Plaza, and R. Sarmiento, "GPU Implementation of an Automatic Target Detection and Classification Algorithm for Hyperspectral Image Analysis," *IEEE Geosci. Remote Sens. Lett.*, vol. 10, no. 2, pp. 221–225, Mar. 2013.

[16] W. Tang and D. A. Bennett, "Parallel agent-based modeling of spatial opinion diffusion accelerated using graphics processing units," *Ecological Modelling*, vol. 229, pp. 108–118, Mar. 2012.

[17] B. G. Aaby, K. S. Perumalla, and S. K. Seal, "Efficient simulation of agent-based models on multi-GPU and multi-core clusters," in *Proc. 3rd International ICST Conference on Simulation Tools and Techniques (SIMUTools'10)*, Mar. 2010, pp. 29:1–29:10.

[18] F. Ye, X. Shi, S. Wang, Y. Liu, and S. Y. Han, "Spherical interpolation over graphic processing units," in *Proc. 2nd International Workshop on High Performance and Distributed Geographic Information Systems*, Nov. 2011, pp. 38–41.

[19] X. Shi and F. Ye, "Kriging interpolation over heterogeneous computer architectures and systems," *GIScience and Remote Sensing*, pp. –, Apr. 2013.

[20] NVIDIA Corporation, "NVIDIA's next generation CUDA compute architecture: Kepler gk110," White paper V1.0, 2012, available online on http://www.nvidia.com.

[21] J. R. Jensen, *Introductory Digital Image Processing: A Remote Sensing Perspective (3rd ed)*. Upper Saddle River, New Jersey: Prentice Hall, May 2004.

[22] *http://www.esri.com/software/arcgis/*.

[23] G. H. Ball and D. J. Hall, "ISODATA: a method of data analysis and pattern classification," Stanford Research Institute, Menlo Park, California, Tech. Rep., 1965.

[24] M. Gardner, "Mathematical games - the fantastic combinations of John Conway's new solitaire game "life"," *Scientific American*, no. 223, pp. 120–123, Oct. 1970.

[25] *http://keeneland.gatech.edu/*.

[26] *http://www.jics.tennessee.edu/aace/beacon*.

[27] F. Wu, "Calibration of stochastic Cellular Automata: the application to rural-urban land conversions," *International Journal of Geographical Information Science*, vol. 16, no. 8, pp. 795–818, 2002.