Performance Optimization on Intel Xeon Phi Through Load Balancing

Chenggang Lai[†], Xuan Shi[‡], Miaoqing Huang[†]
[†]*Department of Computer Science and Computer Engineering,* [‡]*Department of Geosciences*
University of Arkansas, Fayetteville, Arkansas 72701, USA
{cl004, xuanshi, mqhuang}@uark.edu

***Abstract*—Many-core accelerators, such as Xeon Phi coprocessors and GPUs, have been adopted in many high-performance computer clusters to improve the overall performance. Performance improvement requires software to effectively partition workload and hardware resource to maximize the utilization of host and accelerator. The MPI programs may synchronize frequently. When the workload is evenly distributed, the faster ranks will idle at the synchronization point waiting for the slowest rank to finish. In order to achieve a good performance, load imbalance has to be minimized in the heterogeneous environment. However, determining the right workload partition and task parallelism is challenging. This is because many factors need to be taken into account, including the computing capabilities of the processors, the communication bandwidth between devices, etc.**

In this paper, we present a smart data distribution model (DDM) to determine workload partition for applications running on Intel Xeon Phi accelerated clusters. This model relies on the accurate performance profiling as the parameter to allocate the amount of data to different types of processors and coprocessors based on their computing capabilities to achieve the load balance. We conduct a detailed study on the performances of 4 programming models (i.e., native, symmetric, hybrid, and DDM models) on heterogeneous environments. The proposed DDM model achieves a 2× speedup than the native model.

***Index Terms*—Parallel programming model, heterogeneous computing, load balance, MIC.**

I. Introduction

Emerging computer architectures and advanced computing technologies, such as Intel's Many Integrated Core (MIC) Architecture [1] and graphics processing units (GPUs) [2], provide a promising solution to employ parallelism for achieving high performance, scalability and low power consumption. However, programming heterogeneous architectures requires a careful consideration of the computing capabilities of underlying devices and the memory bandwidth between the host CPUs and the accelerators. Otherwise, it may have the problem of load imbalance, which leads to an unsatisfactory performance. MPI programs may synchronize frequently. The faster rank will stay idle at the synchronization point waiting for the slowest rank to finish if a load imbalance happens. Many applications [3] [4] have employed various data partitioning and task partitioning schemes, but many of them run the applications only on accelerators without considering the simultaneous execution on the CPUs. To achieve the maximum gain and resource utilization on heterogeneous computer clusters, the computation workload in an application should be distributed across accelerators and host CPUs according to their computing capabilities.

An ideal load balancing scheme should be able to minimize the total amount of execution time. Two factors are critical to achieve such an ideal balancing in the heterogeneous architectures, (1) the accurate measurement of computing capabilities of devices, and (2) the ideal size of a data partition. In this work, we conduct a detailed study regarding the performance of MIC accelerated computer clusters under different programming models, including the proposed smart data distribution model (DDM). We use cellular automata (CA) as the benchmark to test the performances and scalabilities of different programming models on the Beacon computer cluster with MIC coprocessors. We also develop an urban sprawl simulation and do a performance comparison between one-MIC-plus-one-CPU and one-GPU. In the implementations of both CA and urban sprawl simulation on Intel MIC coprocessors, the vector processing unit (VPU) is leveraged to achieve the most performance potential of the Xeon Phi coprocessors.

II. Related Work

Heterogeneous multiprocessors have been drawing increasing attentions from both the hardware and software research communities. Load balancing in heterogeneous environment is a critical task in order to get high performance. The hardware heterogeneity makes it difficult to ensure reasonably uniform resource utilization, thus leading to performance losses due to load imbalance. In the heterogeneous programming systems, static and dynamic models are used to solve the problem of load imbalance.

Static methods [5], which are based on data partitioning, need information about the application and heterogeneous platform. This information can be gathered at both compilation time and execution time. Static methods depend on accurate performance model to predict the future execution of the application and it can lead to very high performance only after several runs of the benchmark. Dynamic methods [6], which are based on task scheduling and work stealing, balance the load by moving fine-grained tasks between processors and coprocessors during the calculation. These methods do not require the information of applications and heterogeneous platforms. Dynamic methods often use static partitioning for their initial step due to its communication cost, bounded tiny load imbalance, and smaller scheduling overhead [7].

On multicore platforms, parallel processes interfere with each other through the shared memory so that the speed of individual cores cannot be measured independently, and independent performance models cannot be defined for cores. In our previous work [6], we conduct a detailed study regarding the performance and scalability of native and offload models on Intel MIC processors. We proposed a dynamic distribution mechanism to resolve the imbalance of workload among many cores in the offload model.

III. Experiment setup

A. Cellular Automata (CA): an intensive communication benchmark

A cellular automata (CA) model is set up based on a set of cells with identical shape and size. Each cell is associated with a state of a finite set, indicating its condition or attribute. The CA model can be updated in discrete time steps or iterations. The status of each cell is changed according to a collection of transition rules, synchronously applied over the cell space at every step. The next state of a cell depends on its current state and the states of its neighbors. Cellular automata provide a straightforward approach for spatiotemporal dynamic simulations, and have been widely adopted in geospatial studies and spatial decision-making applications, such as land-use and land-cover change.

In this work, we simulate a two-dimensional cellular automaton called Game of Life in which every element becomes a 'cell'. Each cell has two possible states, 0 or 1. One cell has 8 neighbors adjacent to it. A set of rules will decide the center cell in a 3×3 window to become 1 or 0 in the next time stage. As the simulation proceeds, each stage of Game of Life can get fewer live cells than the previous stage.

In this implementation, the status of each cell in the grid will be updated for 100 iterations (representing 100 stages). In each iteration, the states of all cells are updated simultaneously. The pseudo code is illustrated in Algorithm 1. We divide a square matrix (representing the space of Game of Life) into stripes in a row-wise order. Each stripe is handled by an MPI process. Each MPI process needs to send the states of the cells along the boundaries of each stripe to its neighbor MPI processes and receive the states of the cells of two adjacent rows. 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 Game of Life benchmark can be applied to a variety 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.

ALGORITHM 1: Game of Life

```
Function Transition($Cell, t$)
    n = number of alive neighbors of cell at time t;
    if cell is alive at time t then
        if $n > 3$ then
            Cell dies of overcrowding at time t+1;
        end
        if $n < 2$ then
            Cell dies of under-population at time t+1;
        end
        if $n = 2$ or $n = 3$ then
            Cell survives at time t+1;
        end
    end
    if $n = 3$ and Cell is dead at time t then
        Cell becomes a live cell by reproduction at time t+1;
    end
end
```

B. Experiment platform

We conduct our experiments on the NSF sponsored Beacon supercomputer hosted at the National Institute for Computational Sciences (NICS), University of Tennessee. The experiments were conducted on one compute node configured with 2 Xeon E5-2670 CPUs, 256 GB of RAM, and 4 Xeon Phi 7120P coprocessors. Each Xeon Phi 7120P coprocessor contains 61 1.24 GHz MIC cores and 16 GB GDDR5 onboard memory.

IV. Emerging Heterogeneous Computer Architectures and Programming models

A. Intel Many Integrated Core (MIC) Architecture

The commercially available Intel coprocessor based on Many Integrated Core architecture is Xeon Phi. 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. An important component of each Intel Xeon Phi processing core is its vector processing unit (VPU), which significantly improves the computing power. Each VPU supports a 512-bit SIMD instruction and provides a method of data parallelism. Three approaches [1] are introduced to parallelize applications on computer clusters equipped with MIC coprocessors, such as native model, offload model and symmetric model.

In our previous work [6], we proposed a method of dynamic distribution to solve the imbalance of workload on offload model. In this paper, we develop a data distribution model to minimize the load imbalance of the symmetric programming model.

B. Programming models

1) Native Model (MPI-based implementation): Native execution occurs when an application runs entirely on Intel Xeon Phi coprocessors and no job is dispatched on the host CPUs. Applications that are already implemented by MPI can use this model by distributing MPI ranks across the coprocessors natively. The MPI library is designed to support a program running on a heterogeneous set of nodes. In the native model, MPI can run on the coprocessors without modifying any source code. Each MIC core directly hosts up to 4 MPI processes. The native model avoids the complex architectural heterogeneity on a heterogeneous supercomputer. However, the MPI program only runs on Xeon Phi coprocessors. It does not take advantage of the compute capacity of the host CPUs and is likely to give up too much performance potential of the whole system.

2) Symmetric Model: In the symmetric model, the program runs on both the host processors and coprocessors. The MPI processes/ranks reside on the host CPU and the MIC coprocessors. The work load is evenly distributed among all MPI processes. If m MIC (Xeon Phi) coprocessors, in which each MIC core directly hosts n (up to 4) MPI processes, are used in addition to k MPI processes on the CPUs, $m \times n \times 60 + k$ single-thread MPI processes are created in the parallel implementation.

Many MPI programs were written with the implicit assumption that they will run on homogeneous systems in which each MPI rank computes at the same rate. These programs decompose the problem into even parts to compute so that all MPI processes can synchronize at some point without waiting for a long time. However, when some ranks are hosted on processors/coprocessors that have a stronger computing capability, the slowest rank will determine the overall computation rate. This is a typical load imbalance problem.

3) Hybrid Model: On top of the symmetric programming model, a hybrid model makes multithreading inside each MPI process on MIC. When MPI processes are scheduled to both the host Xeon processors and the MIC coprocessors, a single-thread MPI process on the MIC coprocessor has a weaker performance than a single-thread MPI process on the Xeon processor. Given the same amount of data, the MPI process on MIC will take a longer time than the MPI process on the Xeon processor to finish. In order to improve the performance of MPI processes on MIC, multiple internal threads are launched. This model can solve the load imbalance problem and improve the performance for parallel applications by carefully adjusting the extent of multi-threading inside the MPI processes on the MIC coprocessors.

4) Data Distribution Model (DDM): In above three models, each MPI process will receive the same amount of data no matter it is single-threaded or multiple-threaded. In the hybrid model, a careful tuning is required to have equivalent performances from MPI processes on different processors/coprocessors.

In the Data Distribution Model (DDM), single-thread MPI processes are scheduled to both the host processors and the coprocessors. Based on the computing capabilities of the host processors and the coprocessors, a proportional amount of data will be given to an MPI process so that all MPI processes will finish the data processing in more or less the same amount of time. The high performance of parallel applications on heterogeneous platforms can be achieved by partitioning the computational load unevenly across processors/coprocessors. This model relies on an optimal distribution to partition data among MPI processes with different computing capabilities, as shown in Figure 1.

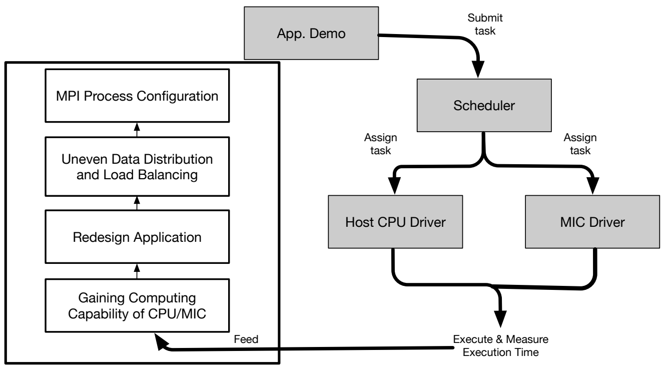

App. Demo
Submit task
Scheduler
Assign task
Assign task
Host CPU Driver
MIC Driver
Execute & Measure Execution Time
Feed
MPI Process Configuration
Uneven Data Distribution and Load Balancing
Redesign Application
Gaining Computing Capability of CPU/MIC

Fig. 1. Data Distribution Model (DDM) workflow.

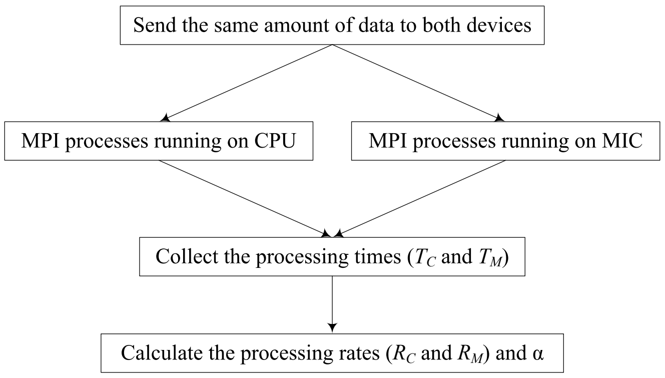

Send the same amount of data to both devices
MPI processes running on CPU
MPI processes running on MIC
Collect the processing times (T_C and T_M)
Calculate the processing rates (R_C and R_M) and α

Fig. 2. The process to calculate α in the DDM model.

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. One issue in data communication is the amount of data exchanged, which may have a significant impact on the total performance. Therefore, we cannot simply decide the distribution of data via the peak performance of processors/coprocessors. Instead, a performance profiling process by running the benchmark on the real platform is required to decide the precise data distribution among various types of processors/coprocessors.

Given an MPI application using the DDM model, (1) all MPI processes are single-threaded, and (2) the ratio between the number of MPI processes on the host processors and the number of MPI processes on the coprocessors is fixed (e.g., 8 MPI processes on the Xeon CPU vs. 240 MPI processes on the Xeon Phi 7120P). For achieving a load balance among heterogeneous nodes, optimal distribution of data between different processors is typically based on their computing capabilities to the kernel. In this work, we define the percentage of data distributed to the MIC coprocessors as α. If we use N to represent the amount of overall data, then $N \times \alpha$ data is scheduled to run on the MIC coprocessors and $N \times (1 - \alpha)$ data is sent to the host processors.

Figure 2 shows the work flow for calculating α. The same amount of data is given to a single host processor and a single MIC coprocessor, each of which executes certain number of MPI processes (e.g., 8 MPI processes on the Xeon CPU and 240 MPI processes on the Xeon Phi 7120P). The host processor and the MIC coprocessor process the same amount of data independently and the respective processing times are collected, i.e., T_C and T_M. The processing rates of both devices can be computed as $R_C = \frac{1}{T_C}$ and $R_M = \frac{1}{T_M}$, respectively.

Given the overall data of N and the parameter α, in order for both types of devices to finish the computation in the same amount of time, we have the following equation.

$$\frac{N \times \alpha}{R_M} = \frac{N \times (1 - \alpha)}{R_C}$$

Then α can be derived in the following formula.

$$\alpha = \frac{R_M}{R_M + R_C}$$

C. The implementation of Game of Life by different models on heterogeneous clusters with MIC coprocessors

In the native model, Game of Life runs entirely on Intel Xeon Phi coprocessors. The cells in the n×n matrix grid are partitioned into 240 stripes in the row-wise order and each stripe is handled by one MPI process. At the beginning of each iteration, the states of the cells along the boundaries of each stripe have to be exchanged with its neighbors through the MPI `send` and `receive` commands. In other words, each MPI process needs to send the states of the cells along the boundaries of each stripe to its neighbor MPI processes and receive the states of the cells of two adjacent rows. For each iteration, the states of all cells are updated simultaneously.

For the symmetric model, the MPI processes run on both the MIC cores and the host Xeon CPU cores. On the compute node on Beacon supercomputer we use in this work, there are 2 8-core Xeon CPUs, running up to 32 MPI processes, and 4 Xeon Phi 7120P coprocessors. On the MIC coprocessor, we run 240 single-thread MPI processes to process 240 stripes. Accordingly, for each MIC coprocessor, we run 8 single-thread MPI processes on the host CPU to process 8 stripes.

In the hybrid model, the MPI processes running on the MIC coprocessor carry out the communication and serial parts; and the OpenMP threads inside the MPI processes carry out the computation part. For each MIC coprocessor, we schedule 8 single-thread MPI processes on the host CPU and then determine the optimal combination of the number of MPI processes on the MIC coprocessor and the number of OpenMP threads in each MPI process. Because there are

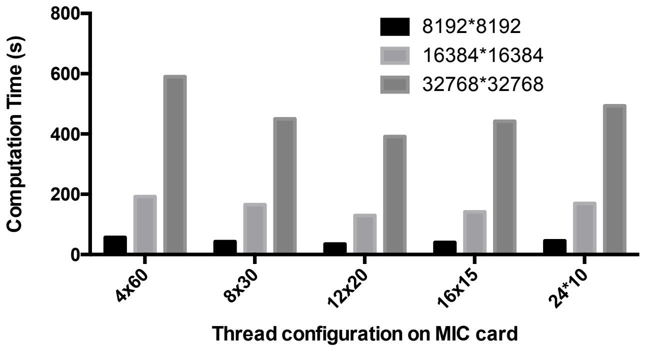


Fig. 3. Performance comparison among different MPI/OpenMP configurations on a single MIC card. M×N: M is the number of MPI processes running on a MIC card, N is the number of OpenMP threads spawned in each MPI process.

multiple combinations of the number of MPI processes on a MIC coprocessor and the number of threads spawned by each MPI, we carried out an experiment for figuring out the performances of various combinations. The results of 5 combinations are shown in Figure 3. It can be found that we can achieve the best performance when the MIC card issues 12 MPI processes and each MPI spawns 20 threads. This result agrees with the performance model we obtain in the DDM, i.e., the performance of a single thread on the host CPU is equivalent to the combined performance of 20 threads on the MIC coprocessor. Therefore, this model issues 8 single-thread MPI processes to a CPU and 12 MPI processes to a MIC coprocessor. A single MPI process running on the MIC coprocessor spawns 20 threads using OpenMP.

In the DDM, we use two iterations of Game of Life as a test to determine the computing capabilities of various processors/coprocessors. We find the computing capability of a MIC coprocessor is about 1.5 times the combined performance of 8 threads on the host processor. We redesign the implementation of Game of Life based on this test. We schedule 8 single-thread MPI processes on the host processor to process 8 stripes. A MIC coprocessor runs 240 single-thread MPI processes to process 240 stripes. The program decomposes the universe of the Game of Life into uneven stripes based on the calculated α. A stripe processed on the CPU contains 20 times cells as a stripe processed on the MIC coprocessor. Through the uneven data distribution, all MPI processes can finish the data processing in roughly the same time.

The implementation details under different parallel programming models are summarized in Table I.

D. Result and discussion

We conduct the experiments to use Xeon Phi 7120P coprocessors to demonstrate the performance of the Game of Life with different programming models. Up to 4 Xeon Phi 7120P coprocessors are used in the experiments. In all of

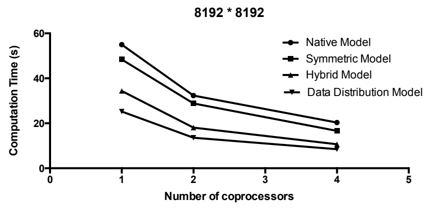
TABLE I
IMPLEMENTATION DETAILS OF 4 PARALLEL MODELS FOR GAME OF LIFE.

Model	Number of MPI processes on one MIC coprocessor	Number of MPI processes on the host CPU corresponding to one MIC coprocessor	Work load distribution among MPI processes
Native	240 single-thread MPI processes	None	Even
Symmetric	240 single-thread MPI processes	8 single-thread MPI processes	Even
Hybrid	12 MPI processes; each contains 20 threads	8 single-thread MPI processes	Even
DDM	240 single-thread MPI processes	8 single-thread MPI processes	Proportional

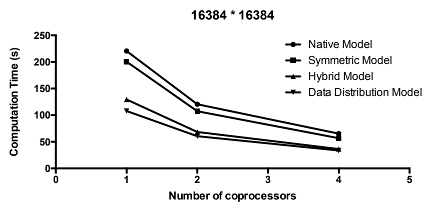
TABLE II
PERFORMANCE OF GAME OF LIFE USING MULTIPLE MIC COPROCESSORS. (UNIT: *second*).

Number of	8,192×8,192				16,384×16,384				32,768×32,768			
MIC coprocessors	N	S	H	D	N	S	H	D	N	S	H	D
1	54.98	48.37	34.38	25.21	220.45	200.46	129.53	107.79	819.45	788.35	391.42	378.76
2	32.34	28.82	18.07	13.56	120.42	107.34	68.35	60.45	450.34	423.90	230.02	209.45
4	20.34	16.63	10.66	8.45	65.34	56.67	36.36	33.56	240.34	220.45	130.47	120.45

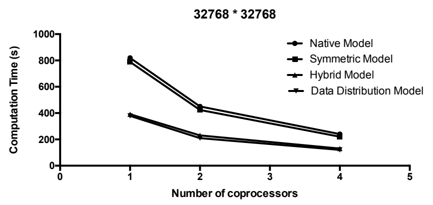
N: Nave model; S: Symmetric model; H: Hybrid model; D: DDM.



(a) 8,192×8,192 cells in the universe of Game of Life.



(b) 16,384×16,384 cells in the universe of Game of Life.



(c) 32,768×32,768 cells in the universe of Game of Life.

Fig. 4. Performance of Game of Life on three different configurations.

the implementations, the vector processing units on the MIC coprocessors have been leveraged.

Three different grid sizes are tested, i.e., 8192×8192, 16,384 ×16,384, and 32,768×32,768. From the results in Figure 4 and Table II, it can be found that the hybrid model and DDM consistently outperform the native model and the symmetric model for this intense communication case. The native model does not take advantage of the compute capacity of the host Xeon CPU when the MPI program only runs on Xeon Phi coprocessors. By observing the performance results of symmetric model, it can be found that the symmetric model is still able to reduce the computation time compared with the native model. However, the improvement is very marginal. A Xeon CPU core has a stronger computing capability than a MIC core. Therefore, a single thread on the Xeon CPU core has a much higher capability than a single thread on the MIC core. But the symmetric model decomposes the problem into even parts so that the MPI processes executing on Xeon CPUs have to wait for the MPI processes on the MIC coprocessors at the synchronize point. Although the symmetric model takes advantage of the compute capacity of the host Xeon CPUs, it demonstrates a load imbalance problem when the MPI programs are written for homogeneous systems.

Both the hybrid model and DDM also take advantage of the compute capacity of the host Xeon CPU. From the results in Table II, they have a similar performance and achieve about 2 times speedup than the native model. However, compared with the hybrid model, the DDM model has the following advantages. (1) In order for the hybrid model to achieve the best performance, the combination of MPI/OpenMP on the MIC coprocessor has to be carefully tuned in advance. In the DDM model, we design the application to figure out the computing capabilities of host processors and coprocessors automatically and then distribute the work load proportionally. (2) The amount of code modification from a pure MPI application to the hybrid model is significantly greater than the amount of code revision from the pure MPI application to the DDM model. In the DDM model, all the MPI processes are all single-threaded.

V. ACCELERATING URBAN SPRAWL SIMULATION

Among varieties of approaches to simulating urban growth, Cellular Automata (CA), an important spatiotemporal simulation approach and an effective tool for spatial optimization strategies, has been extensively applied to understand the dynamics of land use and land cover change. This study aims to utilize the MIC/CPU heterogeneous architecture to enable complex CA simulations for urban growth simulation over

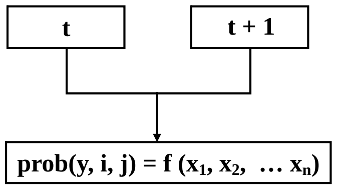


Fig. 5. Calibration of the global probability surface from sequential land use data.

massive datasets to provide quick and scalable support for land allocation.

According to the original model designer [8], 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 or parameter values of the rules from the observation of land use pattern at time t and $t + 1$, as shown in Figure 5. 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 $(x_1, x_2, \ldots, x_n)$.

Besides, two factors, the probability of site selection and the joint probability, 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 [9], computed as (1), In the other word, it can make a decision whether a pixel is developed into urban land use type or remaining in the current state .

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

where p_g is the observed global probability y for a cell to convert to urban, ranging within [0, 1], 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 (2).

$$p_c^t = p_g con(S_{ij}^t = suitable)\Omega_{ij}^t, \quad (2)$$

where $con()$ converts the state of suitable land into a binary 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. The neighborhood potentiality of cell transition, as defined in (3), calculates the urban density of the cells 3 $\times$ 3 neighborhood at time t.

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

Based on the joint probability, a Monte Carlo process is applied in the calculation of the scaling of probability defined in (4).

$$p_s^t(ij) = \frac{q p_t^t(ij)}{\sum_{ij} p_t^t(ij)}, \quad (4)$$

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 the urbanization status of a cell at time t, and $rand(ij)$ generates a random number with uniform distribution within the range of [0, 1]. Comparison between the grid $p_s^t(ij)$ and a random grid will decide whether the cell is to be converted at time $t + 1$ as in (5).

$$S^{t+1}(ij) = \begin{cases} urban, & p_s^t(ij) > rand(ij) \\ rural, & p_s^t(ij) \leqslant rand(ij) \end{cases}, \quad (5)$$

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. We use a part of image, a dimension of 10,000$\times$10,000 for 7 bands, as an input. 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 of Game of Life, i.e., each image is divided into multiple stripes along the row-major order. It is also a densely communicating parallel application.

This urban sprawl simulation is implemented on a single Xeon Phi 7120P coprocessor with a host CPU (Intel Xeon E5-2603 v3) using the DDM parallel model. 240 single-thread MPI processes are scheduled on the MIC coprocessor in addition to 6 single-thread MPI processes on the host CPU. Five different simulations are carried out from 10-year projection to 50-year projection. For this application, we also conduct a performance comparison between a single Nvidia K20 GPU [10] and a single MIC coprocessor (with the host CPU).

The results in Table III show that the implementation of DDM model using one Xeon Phi 7120P (with the host

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

Platform: Single MIC and Single GPU (problem size: $10{,}000\times10{,}000\times7$)															
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**
Single-MIC DDM Implementation															
7120p	6.32	9.4	**15.72**	6.31	18.85	**25.16**	6.11	28.6	**34.71**	6.23	40.45	**46.68**	6.26	49.4	**55.66**
Single-GPU Implementation															
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**

CPU) can outperform the implementation on a single Nvidia K20 GPU by 2 times. GPU solution does not get an ideal performance, because this simulation contains four kernels of CA transition. CA may lead to a problem of warp divergence on GPU because the value of each cell need to be decided by their neighbors, as shown in Algorithm 1. Besides, after completing each CA kernel, operations of $joint$ and sum are implemented to decide the rule of next kernel. As a result, data exchange between GPU and CPU has to be done for a better performance.

VI. CONCLUSION

In this work, we conduct a detailed study regarding the load imbalance problem on heterogeneous computing environment including Intel Xeon Phi coprocessors. Among the four parallel programming models, the native model only schedules the MPI processes to the MIC coprocessors, therefore wasting the computing capability of the host processors. The remaining three programming models schedule the MPI processes to both the MIC coprocessors and the host processors, which may have different computing capabilities. The symmetric model may have a load imbalance problem because the MPI processes running on different types of processors/coprocessors may spend different amount of time to process the evenly distributed work load. The hybrid model and the DDM model try to address the load imbalance problem using different ways. In the hybrid model, the work load is still evenly distributed. However, the MPI processes running on MIC coprocessors will spawn multiple OpenMP threads to match the performance of MPI processes running on the host CPUs. However, the combination of MPI/OpenMP implementation has to be carefully adjusted to achieve the balance between the host processors and the coprocessors. In the DDM model, single-thread MPI processes are deployed on both host processors and coprocessors. However, the work load distribution among the MPI processes is proportional to the computing capabilities of the processors/coprocessors. The DDM model automatically carries out a performance profiling step to determine the computing capabilities of various processors and coprocessors in the heterogeneous environments. Based on the computing capabilities of the hosting processing cores, a proportional amount of data will be distributed to MPI processes so that they finish the data computation in roughly the same amount of time. Among the four parallel programming models, the hybrid model and DDM outperform the native model and symmetric model by about 2 times for the Game of Life benchmark.

ACKNOWLEDGMENTS

This research was partially supported by the National Science Foundation through the award SMA-1416509. This research used resources of the Beacon supercomputer, which is a Cray CS300-AC$^{\text{TM}}$ Cluster Supercomputer. The Beacon Project is supported by the National Science Foundation and the State of Tennessee.

REFERENCES

[1] *Intel Many Integrated Core Architecture. http://www.intel.com.*

[2] D. B. Kirk and W. W. Hwu, *Programming Massively Parallel Processors: A Hands-on Approach (2nd ed).* Burlington, MA: Morgan Kaufmann, Dec. 2012.

[3] R. Scrofano, M. B. Gokhale, F. Trouw, and V. K. Prasanna, "Accelerating molecular dynamics simulations with reconfigurable computers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 6, pp. 764–778, 2008.

[4] M. Huang, C. Lai, X. Shi, Z. Hao, and H. You, "Study of parallel programming models on computer clusters with intel mic coprocessors," *The International Journal of High Performance Computing Applications*, vol. 31, no. 4, pp. 303–315, 2017.

[5] Z. Zhong, V. Rychkov, and A. Lastovetsky, "Data partitioning on heterogeneous multicore platforms," in *Cluster Computing (CLUSTER), 2011 IEEE International Conference on*. IEEE, 2011, pp. 580–584.

[6] C. Lai, M. Huang, and G. Chen, "Towards optimal task distribution on computer clusters with intel mic coprocessors," in *High Performance Computing and Communications (HPCC), 2015 IEEE 7th International Symposium on Cyberspace Safety and Security (CSS), 2015 IEEE 12th International Conferen on Embedded Software and Systems (ICESS), 2015 IEEE 17th International Conference on.* IEEE, 2015, pp. 811–814.

[7] M. D. Linderman, J. D. Collins, H. Wang, and T. H. Meng, "Merge: a programming model for heterogeneous multi-core systems," in *ACM SIGOPS operating systems review*, vol. 42, no. 2. ACM, 2008, pp. 287–296.

[8] 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.

[9] Q. Guan, X. Shi, M. Huang, and C. Lai, "A hybrid parallel cellular automata model for urban growth simulation over gpu/cpu heterogeneous architectures," *International Journal of Geographical Information Science*, vol. 30, no. 3, pp. 494–514, 2016.

[10] C. Lai, M. Huang, X. Shi, and H. You, "Accelerating geospatial applications on hybrid architectures," in *High Performance Computing and Communications & 2013 IEEE International Conference on Embedded and Ubiquitous Computing (HPCC_EUC), 2013 IEEE 10th International Conference on.* IEEE, 2013, pp. 1545–1552.