Comparison of Parallel Programming Models on Intel MIC Computer Cluster

Chenggang Lai†, Zhijun Hao†, Miaoqing Huang†, Xuan Shi†, Haihang You‡
†University of Arkansas, ‡Fudan University, §University of Tennessee
Email: {cl004,mqhuang,xuanshi}@uark.edu, haozhijun@fudan.edu.cn, hyou@utk.edu

Abstract—Coprocessors based on Intel Many Integrated Core (MIC) Architecture have been adopted in many high-performance computer clusters. Typical parallel programming models, such as MPI and OpenMP, are supported on MIC processors to achieve the parallelism. In this work, we conduct a detailed study on the performance and scalability of the MIC processors under different programming models using the Beacon computer cluster. Followings are our findings. (1) The native MPI programming model on the MIC processors is typically better than the offload programming model, which offloads the workload to MIC cores using OpenMP, on Beacon computer cluster. (2) On top of the native MPI programming model, multithreading inside each MPI process can further improve the performance for parallel applications on computer clusters with MIC coprocessors. (3) Given a fixed number of MPI processes, it is a good strategy to schedule these MPI processes to as few MIC processors as possible to reduce the cross-processor communication overhead. (4) The hybrid MPI programming model, in which data processing is distributed to both MIC cores and CPU cores, can outperform the native MPI programming model.

Keywords—parallel programming model; Intel MIC processor; MPI; OpenMP; performance evaluation.

I. INTRODUCTION

Emerging computer architectures and advanced computing technologies, such as Intel’s Many Integrated Core (MIC) Architecture [1] and Graphics Processing Unit (GPU) [2], provide a promising solution to employ parallelism for achieving high performance, scalability and low power consumption. For example, the NSF sponsored Beacon supercomputer [3] contains 192 MIC based Intel Xeon Phi 5110P coprocessors. It is ranked at No. 397 on the Top 500 list [4], and No. 1 on the Green 500 list [5] as of June 2013. The Stampede supercomputer [6] at the Texas Advanced Computing Center, which contains 6,880 Intel Xeon Phi coprocessors, can provide a computing performance of nearly 10 petaflops. The majority of the 10 petaflops comes from the MIC coprocessors.

The current Intel MIC architecture (Knights Corner) contains up to 61 low-weight processing cores, as shown in Figure 1. These cores are connected through a high-speed ring bus. Each core can run 4 threads in parallel. Because each core alone is a classic processor, traditional parallel programming models, such as MPI and OpenMP, are supported by each core. The MIC processors typically co-exist with multicore CPUs, such as Intel Xeon E5, in a hybrid computer node as coprocessors/accelerators. In the remainder of this paper, a single MIC card or device will be called a MIC processor or MIC coprocessor. The constituent processing core on a MIC card will be called a MIC core.

In this work, we conduct a detailed study regarding the performance and scalability of various programming models on Intel MIC processors. In the first model, the MPI process is directly run on each MIC core. In the second model, we try to take advantage of the internal processing parallelism in each MIC core. Therefore, we launch 4 threads in each MPI process using OpenMP. Each MPI process is still run on a MIC core. In the third model, the MPI processes are run on the CPUs. The data processing is offloaded to the MIC processors using OpenMP. We use two geospatial applications, i.e., Kriging interpolation and Cellular Automata, to test the performance and scalability of a single MIC processor and a computer cluster with hybrid nodes. Through this study, we have the following findings. (1) The native MPI programming model on the MIC processors is typically better than the offload programming model, which offloads the workload to MIC cores using OpenMP, on Beacon computer cluster. (2) On top of the native MPI programming model, multithreading inside each MPI process can further improve the performance for parallel applications on computer clusters with MIC coprocessors. (3) Given a fixed number of MPI processes, it is a good strategy to schedule these MPI processes to as few MIC processors as possible to reduce the cross-processor communication overhead when the capacity of the on-board memory is not a limiting factor. (4) We also evaluate a hybrid MPI programming model, which is not officially supported by the Intel MPI compiler. In this hybrid model, the data processing is distributed to both the MIC cores and the CPU cores. The benchmarking results show that the hybrid model outperforms the native
model.

The remainder of this paper is organized as follows. The Intel MIC architecture and the two major programming models are discussed in Section II. We discuss the details of the benchmarks and the experiment platform in Section III. In Section IV, we show the experiment results on a single MIC device. We also compare the performance of a single MIC device with a single Xeon CPU and the latest GPUs. Then we expand the experiment on two geospatial benchmarks to the Beacon cluster using many computer nodes in Section V. Finally, we give the concluding remarks in Section VI.

II. INTEL MIC ARCHITECTURE AND PROGRAMMING MODELS

The first commercially available Intel coprocessor based on Many Integrated Core architecture is Xeon Phi, as shown in Figure 1. Xeon Phi contains up to 61 scalar processing cores with vector processing units. These cores are connected through a high-speed bi-directional, 1024-bit-wide ring bus (512 bits in each direction). In addition to the scalar unit inside each core, there is a vector processing unit to support wide vector processing operations. Further, each core can execute 4 threads in parallel. The communications between the cores can be realized through the shared memory programming models, e.g., OpenMP. Additionally, each core can run MPI to realize communication. Direct communication between MIC processors across different nodes is also supported through MPI.

Figure 2 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 2(a), the MPI processes will be hosted by CPUs, which will offload the computation to the MIC processors. Multithreading programming models such as OpenMP can be used to allocate many MIC cores for data processing in the offload model. The second approach, as shown in Figure 2(b), is to let each MIC core directly host one MPI process. In this way, the 60 cores on the Xeon Phi 5110P, which is used in this work, are treated as 60 independent processors while sharing the 8 GB on-board memory.

III. EXPERIMENT SETUP

A. Benchmarks

Two geospatial applications are chosen to represent two types of benchmarks in high-performance computing: embarrassingly parallel case and intense communication case.

1) Embarrassingly parallel case – Kriging Interpolation: Kriging is a geostatistical estimator that infers the value of a random field at an unobserved location [7]. 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 that reads input point data and returns a raster grid with calculated estimations for each cell. Each input point is in the form \((x, y, Z)\) where \(x\) and \(y\) are the coordinates and \(Z\) is the value. The estimated values in the output raster grid are calculated as a weighted sum of input point values as in (1).

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

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 [8] 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.

In the Kriging interpolation benchmark, the source data set as shown in Figure 3(a) is evenly partitioned among all MPI processes along the row-major order as shown in Figure 3(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 data sets with the respective sizes of 29 MB, 37 MB, 48 MB, and 57 MB. Each data set has 2,191, 4,596, 6,941, and 9,817 sample points, respectively. The output raster grid for each data set has a consistent dimension of 1,440\times 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 data sets are processed in a sequence. For each data set, it is evenly distributed among MPI processes.

2) **Intense communication case – Cellular Automata:**

Cellular Automata (CA) are the foundation for geospatial modeling and simulation. Game of Life (GOL) [9], 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.

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 3(c).

### B. Experiment Platform

We conduct our experiments on the NSF sponsored Beacon supercomputer [3] hosted at the National Institute for Computational Sciences (NICS), University of Tennessee.

The Beacon system (a 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 5110P 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.

The compiler used in this work is Intel 64 Compiler XE, Version 14.0.0.080 Build 20130728.

### IV. EXPERIMENTS AND RESULTS ON A SINGLE DEVICE

Since a single Intel Xeon Phi 5110P processor is a 60-core processor, it is worthwhile to investigate the performance and scalability of the processor alone.

As shown in Figure 2, there are two models to implement parallel applications on a single Intel MIC processor.

- **MPI@MIC:** i.e., native model. In this implementation, each MIC core will directly host one single-thread MPI process, as shown in Figure 2(b).
- **Offload:** In this implementation, the CPU offloads the work to the MIC processor using OpenMP.

#### A. Scalability on a single MIC processor

When MPI programming model is used to implement the Kriging interpolation application, the workload is evenly distributed among MPI processes. In this benchmark, there are 4 data sets. For each data set, the output is a 1,440\times 720 raster grid. In the MPI implementation, we increase the number of MPI processes from 10 to 60 with a stride of 10 processes. The computation of 720 columns of the output grid is evenly distributed. The 50-process configuration is skipped because 720 columns cannot be shared by 50 processes equally. For the offload programming model, we use OpenMP to parallelize the for loops in the program.

### Table I

<table>
<thead>
<tr>
<th>Number of MIC cores</th>
<th>Programming model: MPI@MIC</th>
<th>Programming model: Offload</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Read</td>
<td>Interpolation</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>0.65</td>
<td>0.60</td>
</tr>
<tr>
<td>Interpolation</td>
<td>2734.45</td>
<td>1353.48</td>
</tr>
<tr>
<td>Write</td>
<td>9.44</td>
<td>9.21</td>
</tr>
<tr>
<td>Total</td>
<td>2744.54</td>
<td>1363.30</td>
</tr>
</tbody>
</table>

*The work could not be distributed into 50 cores evenly.*
Table II

PERFORMANCE OF GAME OF LIFE ON A SINGLE MIC PROCESSOR

(UNIT: second)

<table>
<thead>
<tr>
<th>Problem Size</th>
<th>Number of MIC cores</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10</td>
</tr>
<tr>
<td>8192×8192</td>
<td>82.85</td>
</tr>
<tr>
<td>16384×16384</td>
<td>338.57</td>
</tr>
</tbody>
</table>

Programming model: Offload

<table>
<thead>
<tr>
<th>Problem Size</th>
<th>Number of MIC cores</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10</td>
</tr>
<tr>
<td>8192×8192</td>
<td>405.35</td>
</tr>
<tr>
<td>16384×16384</td>
<td>1506.47</td>
</tr>
</tbody>
</table>

The OpenMP APIs will automatically distribute workload to the MIC cores evenly.

The detailed execution times of the Kriging interpolation benchmark under both programming models are listed in Table I. By looking at the time curves in Figure 4, we can find that both models show a good strong scalability for this application. Their performance in terms of interpolation time is very close too. The reason we do not include the write time in Figure 4 is that the write time may become dramatically lengthy when the number of MPI processes increases. In the Kriging interpolation application, each output raster grid is written into a file. When many MPI processes try to write to the same file, their writes need to be serialized. Further, the arbitration takes a lot of time. This effect is not very significant when one MIC processor is used. Later, we will find that the write time can become extremely significant when many MIC processors are allocated.

Game of Life is a generic Cellular Automata program, in which the status of each cell is dependent upon its eight neighbors. Three different grid sizes are tested, i.e., 8,192×8,192, 16,384×16,384, and 32,768×32,768. However, we encounter either out of memory error or runtime error for the 32,768×32,768 when only one MIC processor is used. From the results in Table II, it can be found that the performance of the native model is approximately 5 times of the CPU (Xeon E5-2670) and memory bandwidth become the limiting factors for the performance.

Further, when more cores are allocated, the memory access demand increases as well. Eventually, the communication and memory bandwidth become the limiting factors for the performance.

As an emerging new technology, it is worthwhile to compare the performance of the Intel MIC processor with the other popular accelerators, i.e., GPU. Furthermore, it is a routine to include very powerful multicore CPUs in supercomputers. Therefore, we conduct a comparison among these three technologies at the full capacity of a single device. For Intel Xeon Phi 5110P, we use all 60 cores under two programming models. For the 8-core Xeon E5-2670 CPU on Beacon cluster, we use OpenMP to issue either 8 threads or 16 threads. For GPU, we test two devices, the Nvidia Tesla K20 based on Kepler architecture [11] and the Tesla K20 based on Fermi architecture [10].

The execution times of Kriging interpolation on various devices are listed in Table III. It can be found that the performance of the MIC processor and the CPU is at the same order of magnitude, although the Xeon E5-2670
Figure 6. Performance of Kriging interpolation on single devices.

Table IV
PERFORMANCE OF GAME OF LIFE ON SINGLE DEVICES (UNIT: second).

<table>
<thead>
<tr>
<th></th>
<th>MIC (60 cores)</th>
<th>CPU (Xeon E5-2670)</th>
<th>Nvidia GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MPI</td>
<td>Offload</td>
<td>8 threads</td>
</tr>
<tr>
<td>8192×8192</td>
<td>23.15</td>
<td>112.19</td>
<td>12.03</td>
</tr>
<tr>
<td>16384×16384</td>
<td>56.31</td>
<td>462.87</td>
<td>48.22</td>
</tr>
<tr>
<td>32768×32768</td>
<td>NA</td>
<td>NA</td>
<td>217.33</td>
</tr>
</tbody>
</table>

actually is able to outperform the Xeon Phi 5110P. The 16-thread CPU implementation is almost 2 times faster than the 8-thread implementation. Both GPUs are able to improve the performance by one order of magnitude. Further, K20 is more than 2 times faster than C2075, as shown in Figure 6.

The performance results of Game of Life on three different types of processors are listed in Table IV. The performance of the MPI implementation on the MIC processor is at the same order of magnitude as the implementations on the CPU and the C2075. The OpenMP implementation on the MIC is generally one order of magnitude worse, and the K20 implementation is generally one order of magnitude better in terms of performance.

V. EXPERIMENTS AND RESULTS USING MULTIPLE MIC PROCESSORS

We also conduct the experiments to use multiple MIC processors to demonstrate the scalability of the parallel implementations for those two geospatial applications.

We have the following programming models on the Beacon computer cluster for parallel implementations using multiple nodes.

- **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 single-thread MPI process, as shown in Figure 2(b). Therefore, if \( m \) Xeon Phi coprocessors are used, \( m \times 60 \) MPI processes are created in the parallel implementation.

- **MPI@MIC+OpenMP**: Each MIC core on Intel Xeon Phi 5110P can support up to 4 threads. In this implementation, 4 threads are created in each MPI process running on a MIC core.

- **MPI@CPU+offload**: In this implementation, the MPI processes are running on the CPU. The data processing is offloaded to MIC through OpenMP, as the case shown in Figure 2(a).

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

A. Comparison among three programming models

1) **Kriging Interpolation**: We allocate 2, 4, 8, and 16 MIC processors for 4 different implementation cases. For the MPI@MIC and the MPI@MIC+OpenMP implementations, \( m \times 60 \) MPI processes are created if \( m \) MIC processors are used. For the MPI@CPU+offload programming model, \( m \) MPI processes are created if \( m \) MIC processors are used. As mentioned before, 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, respectively, for both MPI@MIC and MPI@MIC+OpenMP cases.

The detailed results of the three programming models for Kriging interpolation are listed in Table V. We can find that the write time grows dramatically when more MIC processors are used for native MPI programming models. As mentioned before, the serialization of the write and the arbitration among the numerous MPI processes contribute to the lengthy write process. Therefore, we only include the interpolation time, which includes both the time spent on data processing and the time spent on cross-processor
communication, when comparing the performance of the three programming models in Figure 7. It can be found that the MPI@MIC and the MPI@CPU+offload programming models have the very close performance for this benchmark. When the multithreading is applied in each MPI process on the native MPI programming model, the performance can be improved by roughly 3 times. This case shows that it is not enough to only parallelize application to all the cores on MIC processors. It is equally important to increase the parallelism on each MIC core to further improve the performance.

2) Conway’s Game of Life: For Game of Life on multiple MIC processors, three different grid sizes are tested, i.e., $8,192 \times 8,192$, $16,384 \times 16,384$, and $32,768 \times 32,768$. By observing the performance results in Table VI and Figure 8, it can be found that the behavior is quite different from the performance behavior of Kriging interpolation. First, the strong scalability does not hold for all three programming models. Although the offload model is still able to reduce the computation time to half when moving from 2-processor implementation to 4-processor implementation, the performance hangs afterwards. For the two native MPI on MIC programming models, it almost stops scaling when more processors are allocated. Apparently, for this communication dense application, there is not much performance gain when increasing the number of MIC processors from 2 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.

### Table V

<table>
<thead>
<tr>
<th>Number of Processors</th>
<th>Read</th>
<th>Interpolation*</th>
<th>Write</th>
<th>Total</th>
<th>Read</th>
<th>Interpolation*</th>
<th>Write</th>
<th>Total</th>
<th>Read</th>
<th>Interpolation*</th>
<th>Write</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1.24</td>
<td>232.43</td>
<td>12.24</td>
<td>245.90</td>
<td>0.57</td>
<td>60.43</td>
<td>8.82</td>
<td>69.82</td>
<td>0.18</td>
<td>280.83</td>
<td>1.60</td>
<td>282.61</td>
</tr>
<tr>
<td>4</td>
<td>1.27</td>
<td>116.34</td>
<td>16.44</td>
<td>130.50</td>
<td>0.51</td>
<td>36.54</td>
<td>12.23</td>
<td>48.77</td>
<td>0.04</td>
<td>141.03</td>
<td>1.27</td>
<td>142.33</td>
</tr>
<tr>
<td>8</td>
<td>1.23</td>
<td>61.48†</td>
<td>54.43</td>
<td>117.14</td>
<td>0.50</td>
<td>20.43†</td>
<td>240.33</td>
<td>261.26</td>
<td>0.04</td>
<td>74.30</td>
<td>1.19</td>
<td>75.53</td>
</tr>
<tr>
<td>16</td>
<td>1.31</td>
<td>36.74†</td>
<td>300.23</td>
<td>338.28</td>
<td>0.52</td>
<td>12.33†</td>
<td>210.45</td>
<td>222.78</td>
<td>0.04</td>
<td>38.34</td>
<td>5.94</td>
<td>44.28</td>
</tr>
</tbody>
</table>

*The interpolation 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 with 8 or 16 processors, respectively.

### Table VI

<table>
<thead>
<tr>
<th>Number of Processors</th>
<th>8,192×8,192</th>
<th>16,384×16,384</th>
<th>32,768×32,768</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI@MIC</td>
<td>14.56</td>
<td>8.04</td>
<td>7.18</td>
</tr>
<tr>
<td>MPI@MIC+OpenMP(4 threads)</td>
<td>7.99</td>
<td>8.90</td>
<td>8.74</td>
</tr>
<tr>
<td>MPI@CPU+offload</td>
<td>48.39</td>
<td>89.03</td>
<td>82.51</td>
</tr>
</tbody>
</table>

### Table VII

<table>
<thead>
<tr>
<th>Number of Processors</th>
<th>8,192×8,192</th>
<th>16,384×16,384</th>
<th>32,768×32,768</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 threads</td>
<td>7.99</td>
<td>33.11</td>
<td>149.43</td>
</tr>
<tr>
<td>8 threads</td>
<td>8.04</td>
<td>24.06</td>
<td>27.94</td>
</tr>
</tbody>
</table>

B. Experiments on the MPI@MIC+OpenMP programming model

For the implementations using the MPI@MIC+OpenMP programming model in Section V-A, the number of threads running on each MIC core is 4, which is the number of threads a MIC core can physically execute in parallel. We also want to check the potential of performance improvement by running more threads on a single core. Therefore, in addition to the case of 4, we double the number of threads to 8 for the Game of Life benchmark. The results are listed in Table VII. It can be found that the benefit of adding more threads to MIC cores is very marginal. For small problem sizes, e.g., $8,192 \times 8,192$, the 8-thread OpenMP implementation actually has a worse performance than the 4-thread OpenMP implementation for most cases. For this communication-intensive benchmark, partitioning the computation into more threads introduces more cross-thread communication overhead. For large problem sizes, it is still possible to achieve some performance benefit if each MPI process is given a relatively large amount of data, e.g., $32,768 \times 32,768$ partitioned into 120 MIC cores.
For the implementations using the MPI@CPU+offload programming model in Section V-A, the number of OpenMP threads offloaded to the a MIC processor by an MPI process, which runs on the CPU, is 60, i.e., one OpenMP thread per MIC core. In this experiment, we change the number of threads offloaded to the MIC processor by the MPI process from 10 to 60, as shown in Table VIII and Figure 9. In each case, when the number of threads increases from 10 to 30, the scalability holds. When more threads are scheduled, the computation time decreases, however, at a much smaller rate. For most cases, the computation time actually grows when the number of offloaded threads is increased from 50 to 60.

This performance degradation may be due to the increased inter-thread communication overhead.

D. Experiments on the distribution of MPI processes

When an MPI parallel application runs on a computer cluster with nodes consisting of manycore processors such as Xeon Phi, the distribution of MPI processes is not uniform. Some MPI processes are scheduled to the cores on the same processor. The others are scheduled to different processors. Two MPI processes on the same processor are physically close to each other. On the other hand, two MPI processes on two separate processors are distant. The difference in the distances between two MPI processes will cause the disparity in the inter-MPI communication times.

We design a simple benchmark consisting of only 2 MPI processes using MPI@MIC programming model. In this benchmark, MPI process A sends 500 MB data to MPI process B. Then MPI process B returns the 500 MB data back to MPI process A. We have two options to run the benchmark. In Implementation 1, both MPI processes are scheduled to the same MIC processor. In Implementation 2, these two MPI processes are scheduled to two separate MIC processors. It turns out that Implementation 1 and Implementation 2 take 1.59 seconds and 2.81 seconds, respectively. Apparently, the longer distance between the two

---

**Table VIII**

<table>
<thead>
<tr>
<th>Number of Processors</th>
<th># of OpenMP threads offloaded to each MIC processor</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>10779.47</td>
</tr>
<tr>
<td>4</td>
<td>5807.45</td>
</tr>
<tr>
<td>8</td>
<td>6298.11</td>
</tr>
<tr>
<td>16</td>
<td>6923.38</td>
</tr>
</tbody>
</table>

---

**Figure 8.** Performance of Game of Life under various programming models.

**Figure 9.** Performance of Game of Life (32,768×32,768) using MPI@CPU+offload programming model.

**Figure 10.** Performance of Game of Life (32,768×32,768) under different MPI configurations.
MPI processes in Implementation_2 contributes to the more time spent on communication.

The location difference of MPI processes can result in the performance disparity of an application when it is run under different MPI configurations while the total number of MPI processes is the same. Figure 10 illustrates the different performances of the Game of Life benchmark using the various configurations when 120 MPI processes run on 120 MIC cores. In the $2 \times 60$ configuration, 2 MIC processors are allocated, each of which hosts 60 MPI processes. When the number of MIC processors doubles, the number of MPI processes on a processor is halved. The more processors are allocated, the more cross-processor communication, which brings down the performance. Therefore, when the capacity of the on-board memory is not a limiting factor, it is typically a good strategy to schedule as many MPI processes to a single MPI processor as possible to minimize the cross-board communication overhead.

### E. Hybrid MPI vs native MPI

Another programming/execution model that is not officially supported on Beacon computer cluster is the MPI@Hybrid CPU/MIC, i.e., the the MPI processes run on both CPUs and MIC processors. The results in Section IV-B already demonstrate the impressive performance of the latest multicore CPUs. Therefore, it is necessary to use both processors in the applications. We first implement the Kriging interpolation on the 57 MB data set using 16 MPI processes on a single Xeon E5-2670 CPU, which support 16 parallel threads. The total execution time is 46.02 seconds. Then we implement the same application using a 16+14 hybrid MPI model, i.e., 16 MPI processes on a single Xeon CPU and 14 MPI processes on 14 MIC cores of a single card, the total execution time is 24.75 seconds, an almost $2 \times$ speedup.

We also carry out the hybrid MPI programming model on a separate workstation, which contains one Xeon E5-2620 CPU and two Xeon Phi 5110P cards. On this platform, we use the Game of Life ($16,384 \times 16,384$) as the benchmark. The native MPI implementation of 120 MPI processes on two MIC cards takes 30 seconds. The 12+120 hybrid MPI implementation in which the additional 12 MPI processes run on the single CPU takes 27.42 seconds. The $1.1 \times$ speedup aligns with the ratio of number of MPI processes between the hybrid model and the native model.

### VI. Conclusions

In this work, we conduct a detailed study regarding the performance and scalability of the Intel MIC processors under different parallel programming models. Between the two programming models, i.e., native MPI on MIC core and the offload to MIC core, the native MPI programming model typically outperforms the offload model. Particularly for intensely communicating applications, such as Game of Life, the performance improvement can reach 10 times. It is very important to further improve the parallelism inside each MPI process running on a MIC core for better performance. For embarrassingly parallel benchmarks such as Kriging interpolation, the multithreading inside each MPI process can achieve 3 times speedup compared with the native MPI programming model. Due to the fact that the physical distance between two MPI processes may be different under various MPI distributions, it is typically a good strategy to schedule MPI processes to as few MIC processors as possible to reduce the cross-processor communication overhead given the same number of MPI processes. Finally, we evaluate the hybrid MPI programming model, which is not officially supported by the Intel MPI compiler. Through benchmarking, it is found that the hybrid MPI programming model in which both CPU and MIC are used for processing is able to outperform the native MPI programming model.

### Acknowledgments

This research used resources of the Beacon supercomputer, 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


