Combining OpenCL and MPI to Support Heterogeneous Computing on a Cluster

Ming Li
The University of Tulsa
ming-li@utulsa.edu

Peter Hawrylak
The University of Tulsa
peter-hawrylak@utulsa.edu

John Hale
The University of Tulsa
john-hale@utulsa.edu

ABSTRACT

This paper presents an implementation of a heterogeneous programming model which combines Open Computing Language (OpenCL) and Message Passing Interface (MPI). The model is applied to solving a Markov decision process (MDP) with value iteration method. The performance test is conducted on a high performance computing cluster. At peak performance, the model is able to achieve a 57X speedup over a serial implementation. For an extremely large input MDP, which has 1,000,000 states, the obtained speedup is still over 12X, showing that this heterogeneous programming model can solve MDPs more efficiently than the serial solver does.

KEYWORDS

MDP, MPI, OpenCL, heterogeneous computing, HPC, parallelism

ACM Reference Format:
Ming Li, Peter Hawrylak, and John Hale. 2019. Combining OpenCL and MPI to Support Heterogeneous Computing on a Cluster. In *Practice and Experience in Advanced Research Computing (PEARC '19), July 28-August 1, 2019, Chicago, IL, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3332186.3333059

1 INTRODUCTION

After a long period of development, high performance computing (HPC) and heterogeneous computing have been widely used to solve many computationally intensive problems. The applicable fields include engineering simulation, climate prediction, environment science, data science, life science, etc. The physical hardware platforms that support these computations are also diversified. There are both traditional clusters located in the data center of institutions and cloud-computing based HPC architectures provided by web services. In addition, various models of processors, co-processors, GPUs and field programmable gate arrays (FPGAs) are used as basic components to build such clusters. In the supportive software system, there are multi-thread based POSIX threads (pthreads) and Open Multi-Processing (OpenMP), multi-process based Message Passing Interface (MPI), and Open Computing Language (OpenCL) and CUDA that support heterogeneous computations.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
PEARC '19, July 28-August 1, 2019, Chicago, IL, USA
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-7227-5/19/07…$15.00
https://doi.org/10.1145/3332186.3333059

Many applications running on HPC or heterogeneous clusters use a combination of multiple libraries. For example, the combination of MPI and OpenMP is very popular. Each MPI process often runs on a separate processor. The processors accommodating these MPI processes have multi-core architectures, which allow multiple OpenMP threads being associated with MPI process. Examples of using MPI+OpenMP architecture can be found in [9, 10, 12, 14, 15]. MPI+CUDA is also a common combination. Here MPI processes are mainly used to coordinate the work of many CUDA cores, whereas CUDA cores are the units that undertake the actual computational tasks. Typical application examples using MPI+CUDA can be found in [1, 5, 6, 8, 11].

As a software library that is released relatively later than other HPC libraries, OpenCL provides an alternative method to CUDA to design heterogeneous programs. Compared with CUDA which is dedicated to NVIDIA GPUs, it has the advantage of being able to run on different hardware platforms. Not only does it have hardware support from Intel, AMD and IBM, but NVIDIA as the designer of CUDA also provides OpenCL support for using their graphics cards. On the other hand, although OpenCL has been available for a decade, the architecture of using MPI+OpenCL on application software is not common. Direct experimental data on hybrid of MPI and OpenCL are not widely available. Well-presented examples include [3, 4, 7].

This paper aims to fill the research gap in this area. We choose Markov decision process (MDP) as the general application problem. Specifically we implement a heterogeneous program that combines OpenCL (version 1.2) and MPI to solve the value iteration problem. Related performance tests are conducted on an HPC cluster based with Intel hardware platforms. By comparing the runtimes of the serial and the heterogeneous programs, we evaluate the effectiveness of the MPI+OpenCL architecture in solving such problems.

The remainder of this paper is organized as follows. In Section 2, related research is reviewed. In Section 3, design of the heterogeneous program is described. In Section 4, experimental process is presented and results are analyzed. In Section 5, conclusion is drawn and future work is proposed.

2 RELATED WORK

We investigated several examples that combine OpenCL and MPI in programming to solve specific problems.

In [4], the author proposed a hybrid of MPI and OpenCL to solve the problem of Hyperbolic conservation laws. The original computational domain was split into multiple subdomains, each domain is assigned to a pair of MPI node and GPU device. The performance test was conducted on a hardware platform with Intel Xeon E5-2630 processors, AMD Radeon HD7970 GPUs and NVIDIA K20m GPUs. In [7], an efficient MPI-OpenCL implementation was used to accelerate the LINPACK benchmark for a cluster with mult-GPU nodes. The author combined DTRSM and DGEMM routines, which reduced the amount of data transfer between MPI processes and OpenCL kernels. The performance test was conducted on a cluster with 49 nodes. Each node is composed of two Intel Xeon E5-2630 processors and four AMD Radeon HD7970 GPUs. In [3], the author used the combination of MPI and OpenCL to solve the phase-field equations involved in materials engineering. The original problem domain was decomposed in 1-D and each subdomain was taken by a manually matched pair of MPI process and GPU from a cluster. The performance was evaluated on a cluster with Intel Xeon E5-2670 processors, ATI Radeon HD5870 graphic cards and NVIDIA M2070 GPUs.

Specific to our problem of MDP, implementations using HPC can be found in [2, 13, 16]. In solving the Crowd Simulation problems, Ruiz, *et al.* [13] formulated a MDP solver based on matrix multiplication. In the implementation part, one experiment made use of a 32-bit ARM and a 64-bit Intel Core i7 CPU to run the multi-threaded program, while another experiment worked on GPUs, including the Tesla K40c and the GeForce GTX TITAN, with up to 2000 CUDA cores. The results showed that the GPU based implementation provided a 90X speedup over the CPU based implementation. On the other hand, GPU-based approach represented by [2, 13] is not the only choice to solve MDPs. In searching for the best strategy to win strategy video games, which could also be modeled by MDPs, Zhang, *et al.* [16] proposed Parallel Policy Iteration (PPI) and Parallel Value Iteration (PVI) to parallelize the reward analysis. Their performance test was conducted on an HP Superdone9000, where the program was configured to run with 40 processors, yielding a 20X speedup for the PPI implementation over the serial implementation.

Compared with these aforementioned implementations, our research differs in that the evaluation of our heterogeneous program is conducted on an HPC cluster with both Intel CPUs and co-processors. The OpenCL devices used by our experiments are Intel Xeon Phi Co-Processors. To the best of our knowledge, it is the first time that MDP problem is studied in the context of using such a heterogeneous programming architecture.

3 PROGRAM DESIGN

This section presents the design and complexity analysis of the heterogeneous program for the value iteration method on a MDP.

3.1 Serial Program

In the value iteration method, suppose an MDP has N states and $v_j(n)$ represents the maximum total expected reward (MTER) for initial state j at iteration n. Then the MTER at iteration $(n+1)$ is:

$$v_i(n+1) = \max_k \left(\sum_{j=1}^{N} p_{i,j}^{(k)} r_{i,j}^{(k)} + \sum_{j=1}^{N} p_{i,j}^{(k)} v_j(n)\right) \tag{1}$$

where k is the action taken for state i, $p_{i,j}^{(k)}$ is the probability of state transition $i \longrightarrow j$, and $r_{i,j}^{(k)}$ is the reward obtained from this transition. Suppose there are M actions in the action space, accordingly, there are M pairs of transition probability matrix P and reward matrix R for the MDP. The pseudo code of a serial implementation of this value iteration method is shown in Figure 1.

```
read in M pairs of $P$ and $R$ matrices;
for iteration counter $icnt <= max\_num\_iterations$ do
 for matrix pair counter $dcnt <= M$ do
  for state counter $i <= N$ do
   use $P_{ij}^{(dcnt)}$ and $R_{ij}^{(dcnt)}$ to compute the reward $v_i(n+1)^{(dcnt)}$;
   if $v_i(n+1)^{(dcnt)} >=$ current max $v_i(n+1)$ then
    max $v_i(n+1) = v_i(n+1)^{(dcnt)}$;
    record $dcnt$ as the optimal alternative number for state $i$;
   end if
  end for
 end for
end for
```

Figure 1: The pseudo code of serial implementation of value iteration method.

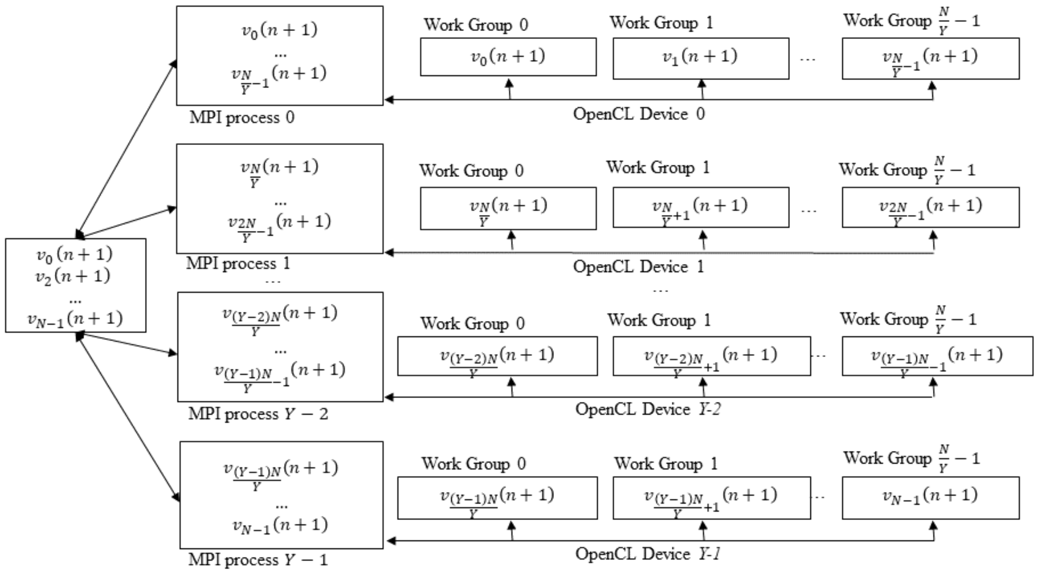
3.2 Heterogeneous Program

The heterogeneous program is designed to run on a high performance computing (HPC) cluster. The cluster provides CPUs to run MPI processes (host code) and OpenCL devices to run OpenCL kernels (kernel code). According to Equation (1), the calculation of the MTER value of different states ($i = 0, 1, ..., N$) at time step $(n+1)$ is independently performed, which favors the use of parallel computing units. As a result, the original problem domain can be partitioned into many subdomains of equal size. Each MPI process takes one subdomain and further offloads it to the parallel units (PUs) in the associated OpenCL device. Once the computation is finished by OpenCL PUs, the partial results are gathered by their master MPI process. All the MPI processes then communicate with one another to obtain a consistent update of METR values. Such partitioning and gathering take place at every MDP time step (shown in Figure 2) until a maximum number of time steps is reached.

The memory capacity of the OpenCL device must be taken into account when designing this heterogeneous program. Because all the computations defined by Equation (1) are performed on OpenCL devices, several rows of the matrices P and R must be put in the global memory of each OpenCL device. As a result, when the number of states is extra large, for example $1,000,000$ states, the capacity of the global memory can be easily exceeded by even a fraction of P and R matrices being uploaded. To address this issue, the matrix rows used by each OpenCL device can be partitioned into chunks, each time only one chunk is uploaded to the global memory as long as the chunk size is small enough. The pseudo MPI host and OpenCL kernel codes based on the chunk operations are shown in Figures 3 and 4, respectively.

3.3 Time complexity analysis

Suppose N, M, K are the number of states, actions, time steps respectively. The serial program has the time complexity of $O(NMK)$. Suppose Y MPI processes are available, and each is associated with an OpenCL device providing Z PUs, the time complexity of the heterogeneous program is $O(NMK/(YZ))$. However, the actual performance is not just determined by the dedicated parallel resources, the following factors also have effects:



$v_0(n+1)$ … $v_{\frac{N}{Y}-1}(n+1)$
MPI process 0
Work Group 0
Work Group 1
Work Group $\frac{N}{Y}-1$
$v_0(n+1)$
$v_1(n+1)$
$v_{\frac{N}{Y}-1}(n+1)$
OpenCL Device 0
$v_{\frac{N}{Y}}(n+1)$ … $v_{\frac{2N}{Y}-1}(n+1)$
MPI process 1
$v_{\frac{N}{Y}+1}(n+1)$
$v_{\frac{2N}{Y}-1}(n+1)$
OpenCL Device 1
$v_0(n+1)$ $v_2(n+1)$ … $v_{N-1}(n+1)$
$v_{\frac{(Y-2)N}{Y}}(n+1)$ … $v_{\frac{(Y-1)N}{Y}-1}(n+1)$
MPI process $Y-2$
$v_{\frac{(Y-2)N}{Y}+1}(n+1)$
$v_{\frac{(Y-1)N}{Y}-1}(n+1)$
OpenCL Device Y-2
$v_{\frac{(Y-1)N}{Y}}(n+1)$ … $v_{N-1}(n+1)$
MPI process $Y-1$
$v_{\frac{(Y-1)N}{Y}+1}(n+1)$
$v_{N-1}(n+1)$
OpenCL Device Y-1

Figure 2: The partitioning and gathering scheme for value iteration method.

```
Input: $N$ states, $M$ actions ($M$ pairs of $P$ and $R$ matrices)
Number of MPI processes: $Y$
Local process id: $y$ ($y \in \{0,1,2,\dots,Y-1\}$)
MTER computed by this process in each time step: $v_{N*\frac{y}{Y}} \sim v_{N*\frac{y+1}{Y}-1}$
Global memory: the memory shared between host and kernel codes
Output: MTER $v_0 \sim v_{N-1}$ and optimal actions $a_0 \sim a_{N-1}$ for
        max_num_iterations time steps

01 read in $M$ pairs of $P$ and $R$ matrices
02 for iteration counter $icnt <= max\_num\_iterations$ do
03   upload $v_0(icnt-1) \sim v_{N-1}(icnt-1)$ to global memory
04   for matrix pair counter $dcnt <= M$ do
05     for chunk counter kcnt <= num of chunks do
06       upload the rows of matrix pair $dcnt$ designated by y and kcnt to
         global memory
07       start kernel computation;
08     end for
09   end for
10   download $v_{N*\frac{y}{Y}}(icnt) \sim v_{N*\frac{y+1}{Y}-1}(icnt)$ calculated by kernel program
     from global memory;
11   MPI allgather with other processes to obtain
     $v_0(icnt) \sim v_{N-1}(icnt)$ and $a_0(icnt) \sim a_{N-1}(icnt)$
12 end for
13 if $y$ is 0 then
14   report final result;
15 end if
```

Figure 3: The host code in heterogeneous program model for value iteration method.

```
Input (in global memory): the $dcnt'th$ pair of $P$ and $R$ matrices,
                          an MTER vector $(v_0(icnt-1), \dots, v_{N-1}(icnt-1))$
Computation task: find MTER values for time step $icnt$ for
                  states $N*\frac{y}{Y} \sim N*\frac{y+1}{Y}-1$, which are designated
                  by host program
Output (in global memory): an MTER sub-vector
                  $\boldsymbol{v\_sub} = \left(v_{N*\frac{y}{Y}}(icnt), \dots, v_{N*\frac{y+1}{Y}-1}(icnt)\right)$
                  and an optimal action sub-vector
                  $\boldsymbol{a\_sub} = \left(a_{N*\frac{y}{Y}}(icnt), \dots, a_{N*\frac{y+1}{Y}-1}(icnt)\right)$

01 define a global memory object to receive $P$ and $R$ from Host;
02 define a global memory object to store the sub-vector $\boldsymbol{v\_sub}$ and $\boldsymbol{a\_sub}$;
03 get the global group ID $i$, which is the index of the state being processed
   by this kernel;
04 fetch row vectors $\boldsymbol{p_i}$ and $\boldsymbol{r_i}$ from $P$ and $R$ to calculate new MTER value $v'$ for
   state $i$ at time step $icnt$;
05 if $v' > v\_sub_i$ then
06   $v\_sub_i = v'$ ;
07   $a\_sub_i = dcnt$
08 end if
```

Figure 4: The kernel code in heterogeneous program model for value iteration method.

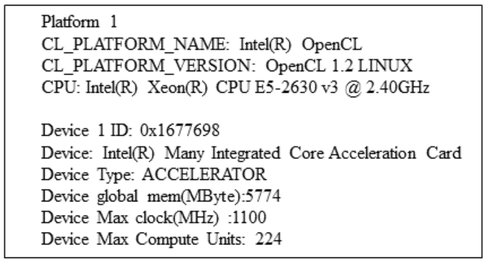
- `OpenCL platform communication`: The global memory of OpenCL device is shared between MPI host and OpenCL kernel. It is the buffer for both input matrices (P and R) and output vectors (MTER values). The overhead of operations on this memory cannot be ignored when it cannot fit in the original input data all at once.
- `OpenCL work group synchronization`: Every execution of OpenCL kernel code finishes when all its work groups wrap up their computations. Only then the results in global memory are effective to be fetched by its MPI process. Therefore, work group synchronization exists within each OpenCL device and the slowest work group determines the runtime of kernel execution.

- `paritition of matrices into chunks`: For an MDP with a large number of states, chunk is necessary and requires extra splitting operations, which increases the workload the MPI host program and the total runtime.
- `MPI process communication`: Partial results gathered by MPI processes are exchanged by invoking MPI allgather communication. This happens at the end of every MDP time step and is an overhead that increases the total runtime.

4 EXPERIMENT AND ANALYSIS

An HPC cluster with 10 available nodes are used for performance tests. Each node has two Intel CPUs and two Intel Xeon Phi Co-Processors. These CPUs run host codes. The enumeration process of OpenCL platform allows us to match each CPU with a co-processor to run OpenCL kernels. As a result, 20 pairs of hosts and kernels are employed. From the viewpoint of hardware, each host-kernel pair is a fully functioning OpenCL platform. The platform's hardware specification is shown in Figure 5. As the clock frequency suggests, the Intel co-processor (Intel Many Integrated Core Acceleration Card) runs slower than the Intel CPU (1100 MHz versus 2400 MHz). However, the former has 224 compute units in a single device, which have the potential to boost performance in the context of heterogeneous computing.

The performance test is conducted in two steps. The first step compares the runtime of the serial and the heterogeneous programs. Speedup values obtained from this step reflect the overall performance improvement brought by using the heterogeneous programming model. The next step compares the runtime of the heterogeneous program and an MPI-only program adpated from it. Speedup values obtained from this step further quantify how much contribution MPI and OpenCL make to accelerating the value iteration.



```
Platform 1
CL_PLATFORM_NAME: Intel(R) OpenCL
CL_PLATFORM_VERSION: OpenCL 1.2 LINUX
CPU: Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz

Device 1 ID: 0x1677698
Device: Intel(R) Many Integrated Core Acceleration Card
Device Type: ACCELERATOR
Device global mem(MByte):5774
Device Max clock(MHz) :1100
Device Max Compute Units: 224
```

Figure 5: The OpenCL platform composed of an Intel CPU and an Intel Xeon Phi Co-Processor.

4.1 Serial Program versus Heterogeneous Program

The runtime values of the serial and the heterogeneous programs are listed in Table 1. For the heterogeous program, these values are obtained by process 0 when the MPI group communication to update MTER values for the last MDP time step is finished. For simplicity, the number of actions is set 4 for all experiments. The number of states is gradually increased in order to examine the scalability of the heterogeneous model. Based on Table 1, the speedup of the heterogeneous program over the serial program is shown in Figure 6.

It can be seen that, when the number of states is smaller than 3200, the serial program runs faster than the heterogeneous program. In this case, the overhead of communication between MPI process and OpenCL kernels accounts for a larger part of the runtime. As more states are added to the MDP, the advantage of using heterogeneous computing outweighs this overhead. The largest speedup of 57.24X is obtained when the MDP has 80,000 states. Due to using multiple chunks, the speedup declines sharply when the number of states is increased to 100,000. By profiling the heterogeneous program, 61.3% of the runtime is found to be used in chunk operations and OpenCL platform communication when the MDP has 80,000 states. This percentage is increased to 84.2% for 100,000 MDP states and 91% for 800,000 states. The more states in the MDP, the higher this percentage. In terms of scalability, the best range of state space size to apply the heterogeneous program is between 10,000 and 400,000 states, where speedup values is around or over 20X. In the extreme case of 1,000,000 states, the speedup is still over 12X, which demonstrates the effectiveness of the heterogeneous program to accelerating the value iteration.

4.2 MPI-only Program versus Heterogeneous Program

In this step, an MPI-only program is adapted from the heterogeous program. Instead of further offloading the assigned problem subdomain to the associated OpenCL kernel, in this program, each MPI process directly performs value iteration using the input data. In addition to the measurement of total runtime, the net computational time, which filters out the impact from chunk operations and OpenCL host-kernel communication, is also measured by both programs. The results are shown in Table 2 and Figure 7.

It can be seen that the speedup of the heterogeneous program over the MPI-only program with respect to total runtime is not significant. This is mainly due to the overhead of chunk operations and OpenCL platform communication. For example, with 6400 MDP states, Table 2 shows that the total runtime of the heterogeneous program is 245 ms, its net computational time is 17 ms, meaning only 7% of the total runtime is effectively used in solving value iteration. By contrast, the MPI-only program uses 92% of the total runtime to accomplish the same computation. As another example, with 80,000 states, the heterogeneous program uses 39% of the total runtime for effective computation while the MPI-only program uses 99%.

Without considering the above overhead, the speedup with respect to net computational runtime is more obvious. 5X speedup is obained at multiple points in Figure 7. The maximum improvement is acquired when 10,000 states are used. A small problem subdomain does not impose enough work load on each OpenCL device. For example, when the number of MDP states is 3200, the number of work groups created for each OpenCL device is 3200/20=160, which is smaller than the number of available compute units (224) on a single Intel co-processor. Accordingly, the device's computational power is not be fully exhibited (speedup is 1.11 for this example). On the other hand, a large problem subdomain increases the cost

Table 1: Serial Program versus Heterogeneous Program (MPI+OpenCL)

Number of States	Serial Runtime (ms)	Number of Chunks in Serial Program	Heterogeneous Runtime (ms)	Number of Chunks in Heterogeneous Program
200	2.4	1	217	1
400	9.2	1	263	1
800	36.8	1	200	1
1600	147	1	175	1
3200	578	1	147	1
6400	2319	1	245	1
10000	5616	1	281	1
20000	2.252+E04	1	502	1
40000	1.351+E05	2	2581	1
80000	5.334+E05	8	9319	1
100000	8.343+E05	10	3.895+E04	2
200000	3.430+E06	40	1.548+E05	5
400000	1.467+E07	160	7.275+E05	20
800000	4.32+E07	640	3.893+E06	80
1000000	8.64+E07	1000	6.889+E06	125

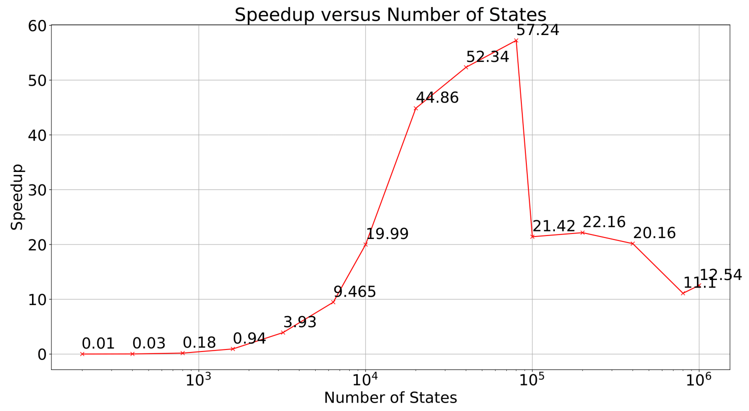


Figure 6: Speedup of the heterogeneous program over the serial program.

of synchronization between OpenCL compute units, which may partially offset the extra computional power obtained by having a large number of such units. As is reflected in Figure 7,the speedup value decreases after the number of states increases to more than 10,000.

5 CONCLUSIONS

This paper designs a heterogeneous program combining OpenCL and MPI to accelerate the value iteration process on MDPs. An HPC cluster with Intel CPUs and co-processors are used to test the performance. The experimental data reveal that the heterogeneous program is able to achieve a 57X speedup over the serial program in best cases. However, its performance is also affected by factors such as chunk operations, synchronization and OpenCL platform communication. In the extreme case of 1,000,000 states in MDP, a speedup of 12X is received. These results demonstrate the efficiency of the heterogeneous programming model to solving large-scale MDP problems.

With respect to future work, the following issues need to be further investigated. First, the communication between MPI process and OpenCL kernel must be properly addressed. This has to do with increasing the scalability of the heterogeneous model. Possible solutions to reduce the amount of such communication include using the queue mechanism defined in OpenCL specifications and designing more independent OpenCL kernel programs. Second, more performance data need to be collected to better evaluate the portability of the heterogeneous programming model. Additional tests can be conducted on other HPC clusters and cloud computing platforms.

ACKNOWLEDGMENTS

This material is based on work supported by the National Science Foundation under Grant No. 1524940. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] David Apostal, Kyle Foerster, Amrita Chatterjee, and Travis Desell. 2012. Password recovery using MPI and CUDA. In *2012 19th International Conference on High Performance Computing*. IEEE, 1–9.
[2] Peng Chen and Lu Lu. 2013. Markov decision process parallel value iteration algorithm on GPU. In *2013 International Conference on Information Science and*

Table 2: Serial Program versus Heterogeneous Program (MPI+OpenCL)

Number of States	MPI-only Program Total Runtime (ms)	MPI-only Program Net Computational Runtime (ms)	Heterogeneous Program Total Runtime (ms)	Heterogeneous Program Net Computational Runtime (ms)
200	0.468	0.114	217	4.72
400	0.878	0.476	263	4.41
800	2.28	1.80	200	5.58
1600	7.785	7.329	175	6.62
3200	30.4	29.2	147	7.80
6400	127	117	245	17.0
10000	286.9	282.1	281	30.3
20000	1134	1128	502	201
40000	4520	4511	2581	790
80000	1.810+E04	1.796+E04	9319	3605
100000	4.762+E04	2.638+E04	3.895+E04	6139
200000	1.923+E05	1.048+E05	1.548+E05	2.103+E04
400000	9.001+E05	4.171+E05	7.275+E05	7.909+E04
800000	4.542+E06	1.666+E06	3.893+E06	3.237+E05
1000000	7.868+E06	2.621+E06	6.889+E06	5.304+E05

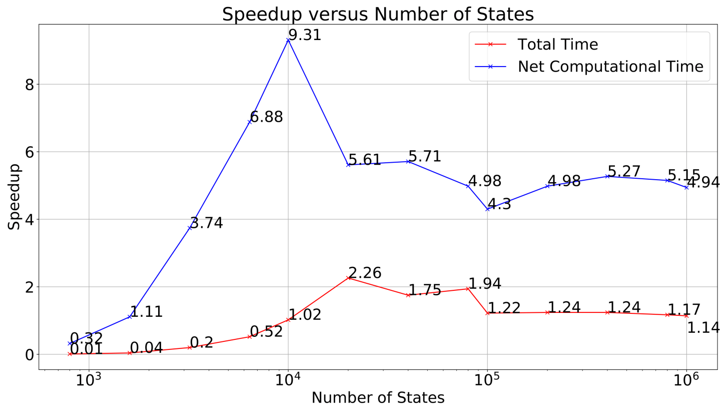


Figure 7: Speedup of the heterogeneous program over the MPI-only program.

Computer Applications (ISCA 2013). Atlantis Press.

[3] Tennyson P Gerald, G. M Karthik, and G Phanikumar. 2015. MPI +OpenCL implementation of a phase-field method incorporating CALPHAD description of Gibbs energies on heterogeneous computing platforms. *Computer Physics Communications* 186 (2015), 48–64.

[4] Philippe Helluy, Thomas Strub, Michel Massaro, and Malcolm Roberts. 2016. Asynchronous OpenCL/MPI numerical simulations of conservation laws. In *Software for Exascale Computing-SPPEXA 2013-2015*. Springer, 547–565.

[5] Miloš Ivanović, Boban Stojanović, Ana Kaplarević-Mališić, Richard Gilbert, and Srboljub Mijailovich. 2016. Distributed multi-scale muscle simulation in a hybrid MPI–CUDA computational environment. *simulation* 92, 1 (2016), 19–31.

[6] Revathi Jambunathan and Deborah A Levin. 2017. Advanced parallelization strategies using hybrid MPI-CUDA octree DSMC method for modeling flow through porous media. *Computers & Fluids* 149 (2017), 70–87.

[7] Gangwon Jo, Jeongho Nah, Jun Lee, Jungwon Kim, and Jaejin Lee. 2015. Accelerating LINPACK with MPI-OpenCL on clusters of multi-GPU nodes. *IEEE Transactions on Parallel and Distributed Systems* 26, 7 (2015), 1814–1825.

[8] Heba Khaled, Hossam El Deen Mostafa Faheem, and Rania El Gohary. 2015. Design and implementation of a hybrid MPI-CUDA model for the Smith-Waterman algorithm. *International journal of data mining and bioinformatics* 12, 3 (2015), 313–327.

[9] Dong Li, Bronis R de Supinski, Martin Schulz, Kirk Cameron, and Dimitrios S Nikolopoulos. 2010. Hybrid MPI/OpenMP power-aware computing. In *2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS)*. IEEE, 1–12.

[10] Vladimir Mironov, Yuri Alexeev, Kristopher Keipert, Michael D'mello, Alexander Moskovsky, and Mark S Gordon. 2017. An efficient MPI/openMP parallelization of the Hartree-Fock method for the second generation of Intel® Xeon Phi™ processor. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, 39.

[11] G Oyarzun, R Borrell, A Gorobets, and A Oliva. 2014. MPI-CUDA sparse matrix–vector multiplication for the conjugate gradient method with an approximate inverse preconditioner. *Computers & Fluids* 92 (2014), 244–252.

[12] Rolf Rabenseifner, Georg Hager, and Gabriele Jost. 2009. Hybrid MPI/OpenMP parallel programming on clusters of multi-core SMP nodes. In *2009 17th Euromicro international conference on parallel, distributed and network-based processing*. IEEE, 427–436.

[13] Sergio Ruiz and Benjamín Hernández. 2015. A parallel solver for Markov decision process in crowd simulations. In *2015 Fourteenth Mexican International Conference on Artificial Intelligence (MICAI)*. IEEE, 107–116.

[14] Bogdan Satarić, Vladimir Slavnić, Aleksandar Belić, Antun Balaž, Paulsamy Muruganandam, and Sadhan K Adhikari. 2016. Hybrid OpenMP/MPI programs for solving the time-dependent Gross–Pitaevskii equation in a fully anisotropic trap. *Computer Physics Communications* 200 (2016), 411–417.

[15] Yunheng Wang, Youngsun Jung, Timothy A Supinie, and Ming Xue. 2013. A hybrid MPI–OpenMP parallel algorithm and performance analysis for an ensemble square root filter designed for multiscale observations. *Journal of Atmospheric and Oceanic Technology* 30, 7 (2013), 1382–1397.

[16] Qi Zhang, Guangzhong Sun, and Yinlong Xu. 2009. Parallel Algorithms for Solving Markov Decision Process. In *International Conference on Algorithms and Architectures for Parallel Processing*. Springer, 466–477.