Classification of OpenCL Kernels for accelerating Java Multi-agent Simulation

Pitipat Penbharkkul and Worawan Marurngsith
Department of Computer Science, Thammasat University, Pathum Thani, Thailand,

Abstract

Java-based multi-agent simulation (MAS) can be offloaded to graphical processing units (GPU) and other OpenCL accelerators to achieve many hundred-fold speedups. However, the performance gain from the accelerated code depends strongly on whether the computation (kernels) have been scheduled to the appropriate devices. Thus, accelerating Java MAS may not lead to a sustainable speedup. This paper proposes a method for a kernel classifier to specify suitable devices to execute OpenCL kernels. The classifier can identify suitable OpenCL devices for kernels based on the static and dynamic characteristics of the code of the kernels. Kernels are grouped by their suitability for particular devices using the multiclass support virtual machine technique. After that, kernels are scheduled to an appropriate task queue. Kernel scheduling based on the proposed technique is compared against the firstcome-first-serve (FCFS) technique and against oracle scheduling when handling eight kernels. Our results show that, using the proposed method, all kernels finished execution 45 percent sooner than using the FCFS technique. However, the overall execution time was 22.5 percent longer than with oracle scheduling. Our results seem to confirm that kernel classification techniques might contribute towards sustainable high performance in accelerated Java-based MAS models.

Keywords: GPGPU, OpenCL, multi-agent simulation, performance, acceleration, SVM, MASON

1 Introduction

Java is a powerful platform for developing multi-agent simulation (MAS) models due to its portability and the huge amount of functionality available in development kits. However, limitations in performance and scalability make Java-based MAS a target for performance acceleration on heterogeneous platforms *e.g.*, multicore CPUs, graphical processing units (GPUs), accelerated processing units (APUs), or coprocessors such as the Intel Xeon Phi (Aaby et al., 2010; Hayashi et al., 2013; Ho et al., 2015; Li et al., 2016). Offloading a cellular automata simulation, *e.g.* the Conway's Game of Life, to a cluster of GPUs could gain a 100x speedup if proper latency hiding techniques are used (Aaby et al., 2010). Well optimised MAS models based on the MASON library, a legacy Java-based MAS framework, could be accelerated from 100x up to 468x (Ho et al., 2015). Recent work proposing AgentPool and agent management techniques has shown that accelerated models can outperform CPU and GPU implementations based on the MASON and FLAME simulation frameworks (Li et al., 2016).

The OpenCL language layer (Group, 2013) has been widely used to exploit data parallelism on heterogeneous systems because of its portability and acceptable performance (Sachetto Oliveira et al., 2012). Research has shown that a single version of OpenCL code can be executed on three different platforms with less than a 12% performance loss, providing that parameters are well chosen (Dolbeau et al., 2013). Programmers can offload data-parallel fragments of Java code to heterogeneous platforms supporting standard OpenCL in two ways: by using an auto-parallelisation tool (AMD Developer Central, 2011; Hayashi et al., 2013) or by manually specifying fragments of parallelisable code using Java to OpenCL bindings such as JOCL (JOCL, 2011).

However, a well-known limitation of OpenCL is that the performance gain from accelerated code depends strongly on scheduling computation (kernels) to appropriate devices. This limitation also applies to accelerated Java MAS code. Hence, its speedup can be a hundredfold or zero. Several ways to predict performance of OpenCL kernels for different devices have been mentioned in three extensive surveys (Mokhtari and Stumm, 2014; Rossbach et al., 2013; Yan et al., 2009). Kernel profiling is a key technique used to get information about kernels to be classified, *e.g.* retrieving from history with profile data (Sato et al., 2011) or developing a framework for profiling a shared library (Matoga et al., 2013). A compiler framework to collect and classify kernels suitable for different devices was proposed in (Lopez-Novoa et al., 2015; Wen et al., 2014). These reports have confirmed that by using classification techniques, a kernel speedup on different OpenCL device can be predicted in advance at up to 87 percent accuracy (Wen et al., 2014)

Table 1. Parallelisation Techniques for ABS.

ABS Class	Parallelisation Techniques
Homogeneous	Agent's computation is implemented as kernel (Ho et al., 2015; Li et al., 2016).
Heterogeneous	Agent's computation is implemented as kernel. Parallel task partition or computing pipeline can be used to speed up complex calculations (as surveyed in (Lopez-Novoa et al., 2015)).
Communicate	Use bitwise operations for checking communication and latency hiding technique for data transfer (Aaby et al., 2010).
Non-communicate	Data partitioning on agent's address space (as surveyed in (Lopez-Novoa et al., 2015)).

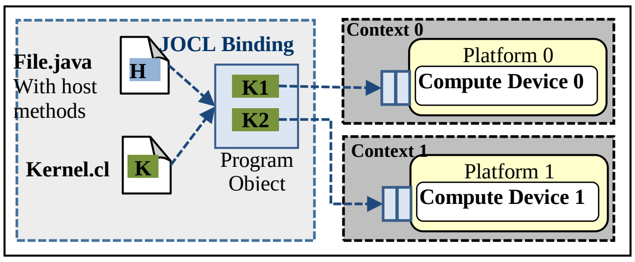

Figure 1. The architecture of the OpenCL execution model with Java code.

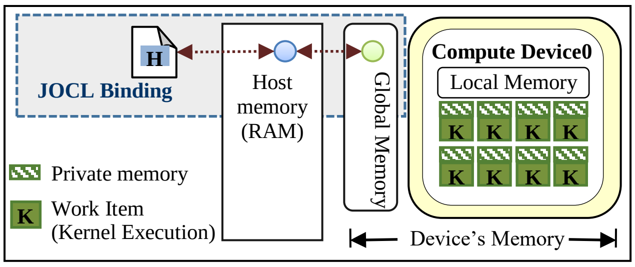

Figure 2. OpenCL memory mapping architecture.

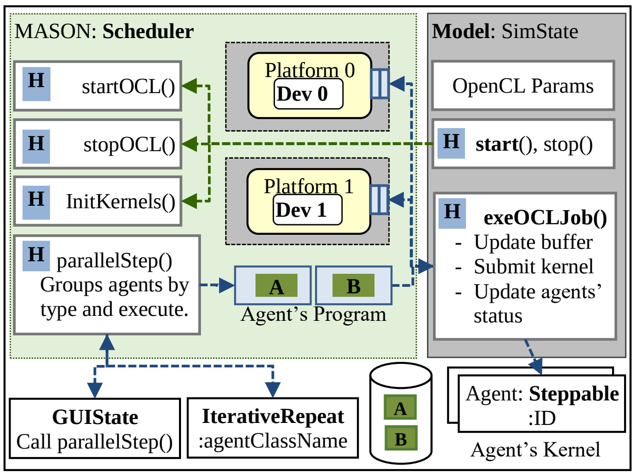

Figure 3. Interaction of components to run OpenCL in MASON's models.

In this paper we propose a technique to classify multiple OpenCL kernels for heterogeneous devices to obtain sustainable overall speedup. Our main contributions are threefold.

(1) We have developed an agent-parallelisation technique on the MASON framework (Luke, 2015) to accelerate Java-based MAS models developed on multiple OpenCL devices. The technique allows modellers to execute the OpenCL computation side by side with an existing multicore computation.

(2) We present a classification technique implemented in the agent scheduler to guide the scheduler as to which OpenCL device is most suitable for a kernel. Central to the classification is performing static and dynamic profiling on the kernel code and using the information obtained to feed the multiclass support vector machine (SVM) classifier.

(3) We show that using the proposed classification technique in the scheduler, the overall speedup of eight kernels used in our experiment outperform the traditional first-come-first-serve (FCFS) scheduler.

The rest of this paper is organised as follows. The next section introduces a new agent-parallelisation technique. Section 3 presents the proposed classification technique. Section 4 analyses the speedup gained from scheduling eight kernels with the proposed technique in comparison to the FCFS and oracle scheduling counterparts. Finally, Section 5 brings this paper to a conclusion.

2 Accelerating Java-Based MAS using OpenCL

Agent-based simulations (ABS) can be grouped into four classes according to the heterogeneity of the agents and how agents interact with each other (Stone and Veloso, 2000). These classes are: (1) homogeneous non-communicating, (2) heterogeneous non-communicating, (3) homogeneous communicating, and (4) heterogeneous communicating. As shown in Table 1, ABS classes can be parallelised on heterogeneous systems using different techniques. A key technique common to both homogeneous and heterogeneous ABS models is agent parallelisation. In agent parallelisation, agents' behaviours are implemented in separate functions that are ready to be offloaded either to GPUs or other accelerators. In OpenCL, these functions are called kernels. A kernel can be scheduled in parallel to be executed on any OpenCL supported device.

Most existing MAS models are implemented on legacy simulation frameworks, *e.g.*, MASON, Repast, FLAME, JADE or NetLogo (as reviewed in (Marurngsith, 2014; Parry and Bithell, 2012; Railsback et al., 2006)). Many of these frameworks have been developed in Java (MASON, Repast, JADE) or Scala

(NetLogo). (Ho et al., 2015) successfully modified the MASON library to support CUDA GPU computing. They achieved a speedup of 187x using the JCUDA binding. However, the target accelerator for CUDA computing is limited to the Nvidia GPUs. The next subsection presents an agent parallelisation alternative technique for OpenCL accelerators.

2.1 The OpenCL Execution Model

The architecture of the OpenCL execution model in Java using the JOCL binding (JOCL, 2011) is shown in Figure 1. An OpenCL project comprises a host application and kernel functions. The execution environment is set by the host application in four steps: (1) getting the number of OpenCL platforms available in a machine, (2) getting the number of available devices for each platform, (3) creating a work space (context) for each platform, and (4) creating a job submission queue (command queue) for each device. The host application also prepares tasks (kernels) ready to be offloaded to available devices. The latter process involves creating a program object, reading kernel files (.cl) and invoking the OpenCL compiler to create binaries for all the kernel functions.

The host application allocates memory buffers in the machine's main memory (called host memory), and also manages data transfers and memory-address mapping between the host memory and the devices' memory (see Figure 2). The memory buffers are used to transfer data to/from a device.

2.2 Implementing OpenCL-Enabled MAS on MASON

The components used to implement the OpenCL-enabled MAS on MASON are shown in Figure 3. Three classes of the MASON simulation engine were modified: Scheduler, GUIState and IterativeRepeat. Four methods
were also added to the Scheduler class to perform the tasks of the host application. The handlers of OpenCL's context, devices, command queues, program, and kernels are all kept in the Scheduler. These handlers can be accessed from the model. This model is a Java class derived from the SimState class which should be manually modified by the modeller. The model accelerated in this work, the Student Schoolyard Cliques model, has been taken from the MASON tutorial. This model consists of student agents and one anonymous agent. Users can enable OpenCL support and select the OpenCL devices via parameters. If the OpenCL flag is set, the start method will invoke a method in the Schedule class to initialise the OpenCL environment. The start method also allocates host memory buffers and maps them to the memory of the OpenCL devices. When a kernel is invoked, data in the host memory buffers is transferred to the memory of the device associated with the kernel.

We use an agent-parallelisation technique to accelerate the execution of student's behaviour at every time step. Our technique consists of rewriting the behaviour of a Student agent as kernel function in OpenCL. Therefore, if the OpenCL flag is set, at every simulation step, the kernel implementing the agents' behaviour is scheduled to be executed on a suitable OpenCL device. If the OpenCL flag is off, the original scheduling method of the MASON framework, called step, is invoked to perform multicore execution. To facilitate the OpenCL scheduling process, we added a method, called parallelStep, to the MASON scheduler. Similar to the original step method, the parallelStep method gathers objects based on their timestamp and puts them into a list. However, the parallelStep method differs from the step method in that the former method splits objects into two lists, an execution list for CPUs and another list for OpenCL devices. In this process, the IterativeRepeat class is used to get objects' class names that are checked against the list of kernels' class names. When a match is found, the object is removed from the CPU list and added to the list of objects for OpenCL devices. Objects in the CPU list are executed using traditional multicore execution. Simultaneously, the objects for OpenCL devices are scheduled on the OpenCL command queue handled by the model. We implemented the method executeOpenCLJob in the model to carry out kernel submission. The executeOpenCLJob method updates memory buffers, submits kernels to the command queue and reads results back from OpenCL devices before updating the status of all agents accordingly. This parallel scheduling process is iterated until the end of the simulation. It is important to note that the scheduler uses the kernel classification to identify which command queue (OpenCL device) is suitable to execute a kernel (see Figure 4).

3 OpenCL kernel classification

The technique for kernel classification has been modified from (Wen et al., 2014) to identify which device is suitable for executing a kernel. In (Wen et al., 2014) the binary SVM classifier gives a more accurate prediction than that of the neuron network technique. The binary SVM classifier was used to classify kernels into low and high speedup groups to identify as a suitable device either a CPU or a GPU. Nevertheless, after collecting the execution time of 21 workloads for training the classifier, we noticed that the execution time obtained from two different GPUs could be significantly different. Thus, in this work, we adopted the multiclass SVM classification technique (Chih-Wei and Chih-Jen, 2002) to support selection from more than two OpenCL devices. Kernels are classified into k groups, where k is the number of available OpenCL devices so that guided results can be used not only with a GPU or CPU, but also with specific devices. The connection of the proposed SVM kernel classifier with the MASON framework is shown in Figure 4.

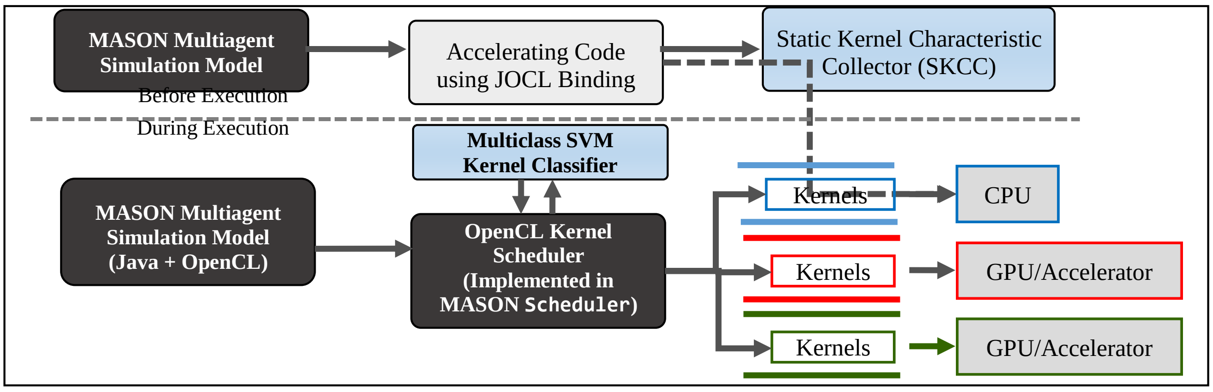
MASON Multiagent Simulation Model
Accelerating Code using JOCL Binding
Static Kernel Characteristic Collector (SKCC)
Before Execution
During Execution
Multiclass SVM Kernel Classifier
MASON Multiagent Simulation Model (Java + OpenCL)
OpenCL Kernel Scheduler (Implemented in MASON Scheduler)
Kernels
CPU
Kernels
GPU/Accelerator
Kernels
GPU/Accelerator

Figure 4. Overall connection of the proposed SVM kernel classifier with the MASON agent-based simulations to offload agents to OpenCL devices.

3.1 Workload Characteristics Profiling

We captured fifteen features of host and kernel code from both a static profiling tool and by inserting profiling functions into the original code. The features collected are shown in Table 2. Twenty-one workloads (available on the Nvidia and Intel websites) were used for training the classifier (see Table 3). The features of kernels from these workloads were collected and passed to the SVM module for training.

3.2 Support Vector Machine Multiclass

The SVM is a well-known supervised binary classifier. The characteristics of the workloads were used to supervised the classifier. Trained workloads were executed on OpenCL devices available in our target machine (see Table 4) to collect the execution times of the workloads. Each workload was labelled to the device with the fastest execution time.

To allow the classifier to work with more than two classes, the well-known One-against-ALL or One Versus the Rest method (Chih-Wei and Chih-Jen, 2002) was used. First, the number of classes was defined as the number of OpenCL devices available in the system *i.e.*, k classes. Second, the binary classifier was constructed by separating one class from the rest. For example, if k=3, the pair wise of binary classifiers are 0 with (1, 2), 1 with (0, 2) and 2 with (0, 1). After that, training data for the classifier were re-labelled and trained for each possible pair of classes. The results were combined to get a multi-class classification according to the maximum output. Note that the SVM classifier from the Intel Data Analytics Acceleration Library was used in this work.

Table 2. Features used in the classification.

At	**Collected Features** (# = number of)
Host	#iteration, workgroup dimension, global size, local size, input buffer size, output buffer size
Kernel	#parameters, #barriers, #math functions, #int and float scalar operations, #int and float vector operations, #atomic, #control

Table 3. List of workloads used in the experiment.

Workload		Input size	#KN[a]	Dim[b]
BlackScholes	Nvidia	1.1M	2	1
ConvolutionSeparable		150M	4	2
DCT8x8		50M	2	2
DotProduct		25M	1	1
FDTD3d		452M	1	2
HiddenMarkovModel		404	2	2, 1
MatrixMul		128K	1	2
MatVecMul		440M	6	1
MersenneTwister		96M	2	1
Reduction		67M	2	1
Scan		54M	4	1
Transpose		33M	5	2
VectorAdd		137M	1	1
BitonicSort	Intel	134M	1	1
GEMM		188M	1	2
GodRays		61M	1	1
MedianFilter		134M	1	2
ProcGraphicsOpt		8M	3	2
SimpleOptimizations		134M	1	1
ToneMapping		61M	1	2
ToneMappingMultiDevice		122M	1	1

Table 4. Experimental Platform Information.

Detail	OpenCL	Platform
Host Machine	Dev 0	Intel Core i7-4710HQ (2.50 GHz, 6 MB L3 Cache, up to 3.50 GHz), 8GB RAM
Accelerator	Dev 1	NVIDIA GeForce GTX 850M (4GB GDDR3), 640 cores, 902MHz, Memory of 4096MB, OpenCL1.2
	Dev 2	Intel(R) HD Graphics 4600 (No dedicated Memory, using max of 1.7 GB RAM), 100 Effective SPUs Count, 400MHz, OpenCL1.2
OS		Windows 10 Pro

4 Experimental results and discussion

Two experiments, using three OpenCL devices, were carried out on a Windows-based machine to confirm the performance of the accelerated MASON MAS model and the proposed kernel classification technique. The specification of the experimental platform is listed in Table 4.

4.1 Performance of the Accelerated MAS Model

In our first experiment, we analysed the performance of the accelerated Student Schoolyard Cliques by comparing the OpenCL execution time against the original multicore execution time as baseline. The baseline execution of the model using one thousand to a quarter of a million students is shown in Figure 5. The execution time shows a linear growth in agents.

The OpenCL-enable model was executed on the Nvidia GPU (Dev1) and on the host CPU (Dev0). We collected the execution time and calculated the speedup relative with the baseline performance (see Figure 6). The results show a similar speedup obtained from two computing devices. However, when the number of agents (students) is small, the OpenCL execution is slower than the multicore execution (1,000 to less than 5,000 agents). Furthermore, the accelerated model outperforms the original model when there are more than 5,000 agents. The highest speedup obtained is 27.6x faster with a constraint of 250,000 agents.

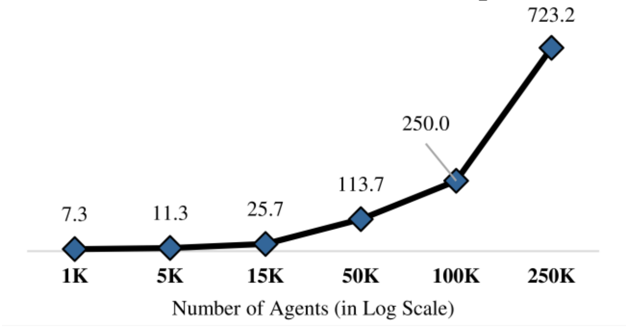

Figure 5. Baseline execution time (in sec.) of the Students model on CPU.

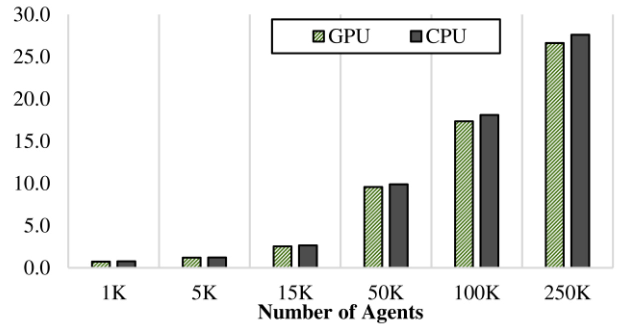

Figure 6. Speedup of the OpenCL execution over the baseline.

Certainly, the available memory space was an issue for the accelerated MAS models on the MASON framework. This is because, in the framework, many key data are defined as double *e.g.*, the coordinate representing agent's position. In our accelerated model, agents resided on a 2D continuous space. Each agent requires 128 bits to store the x, y coordinate. Therefore, the size of the input and output buffers grows with the number of agents. The memory available on our experimental platform reached its limit of devices at 250,000 agents (students). It is also important to note that the double data type is not supported in some OpenCL devices. Consequently, compatibility of the available devices must be verified before executing OpenCL code.

4.2 Performance of the Classifier

The aim of our second experiment was to quantify the effectiveness and accuracy of the classifier. We used the classifier with accelerated MAS kernels. However, MAS models can only generate a limited number of kernels that might not be representative of the key computation load of general kernels. Consequently, only eight kernels and twenty-one scientific workloads were used to test the SVM classifier. The list of kernels used in this experiment and their execution times (in millisecond) measured on the OpenCL devices of the target platform are shown in Table 5. The BitonicSort workload shows the longest execution time, and the most aggressive acceleration on Dev1 (the Nvidia GPU). Note that the execution time obtained from two different GPUs (Dev1 and 2) are very different.

A classification of kernel features is shown in Table 5. The output prediction is used for scheduling the kernels combining the FCFS technique with the results of the classification. The results of all four scheduling techniques, oracle scheduling, the proposed method (FCFS + Classification), all kernels scheduled on the fastest device, and FCFS, are shown in Figure 7 – Figure 10, respectively. The obtained results show that in terms of overall execution time (in milliseconds), the proposed method performs similarly to the fastest device technique, but is 25% slower than oracle scheduling. Scheduling based on the proposed method outperforms the FCFS-only technique by over 45%. In this experiment, suitable devices for seven kernels were correctly identified but one failed. The classifier identified an incorrect device for kernel B (Median Filter) *i.e.*, Dev2 was selected instead of Dev0. Thus, the prediction accuracy of our proposed classifier is 87.5%. However, this accuracy rate cannot be generalised yet as the number of tested kernels was small.

The classifier was also used with the kernels created in the accelerated Student Schoolyard Clique model. In this case, the classifier identified the correct device. However, as the model has only one type of kernel, a very small speedup is observed when using the FCFS + Classification method. Moreover, the kernel execution time on CPU and GPU is very similar (see Figure 6). Thus, more accelerated MAS models should be used to confirmed the effectiveness of the classifier on the MAS acceleration.

Table 5. Kernel Execution Time (in Milliseconds).

Kernels		Classify	Dev0	Dev1	Dev2
A	VectorAdd	Dev2	0.04	4.33	0.02
B	MedianFilter	Dev2	14.29	34.39	35.91
C	BitonicSort	Dev1	4,641.23	2,709.99	4,834.55
D	SobelGraphic	Dev1	3.42	1.26	1.98
E	MT Naïve	Dev1	483.36	198.31	602.09
F	MT Simple Copy	Dev1	377.40	135.43	215.87
G	MT Shared Copy	Dev1	483.48	135.43	305.85
H	Matrix Transpose	Dev1	575.98	138.60	319.77

5 Conclusions and future work

In this paper, an OpenCL-kernel classification technique to identify a suitable OpenCL device for a kernel, has been proposed. An agent-parallelisation technique for multiple OpenCL devices to accelerate a JAVA-based MAS model on the MASON framework has been discussed. A modified SVM for multiclass classification has been used to guide the scheduler to offload kernels to suitable devices. The proposed classification could achieve 87.5 percent of accuracy on tested workloads.

The accelerated Java MAS achieved a 27x speedup in comparison to the original multicore execution using a maximum of 250K agents. The proposed classifying kernel technique arranged the MAS kernel correctly as is demonstrated by scheduling eight different computational kernels. The results show that our classification technique is slower than oracle scheduling, but outperforms FCFS scheduling. The latter suggests that kernel classification can be an alternative option for sustainable speedup in accelerated Java-based MAS models. However, in order to achieve an effective scheduler in the MAS engine, future work must focus on classifying kernels that has been generated from a wider variety of MAS models.

Acknowledgements

We thank the reviewers for their valuable comments. We thank contributors to the MASON, JOCL and OpenCL community website for discussion and lesson learned. We thank Professor Roland Ibbett and JC Diaz Carballo for improving the readability of this paper.

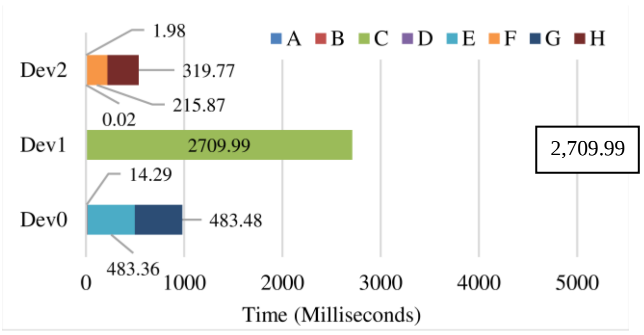

Figure 7. Oracle Scheduling.

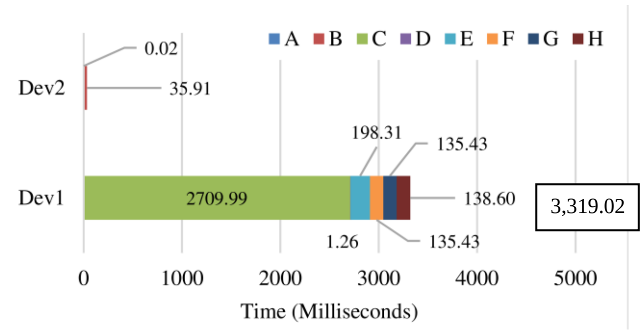

Figure 8. FCFS + Classification Scheduling.

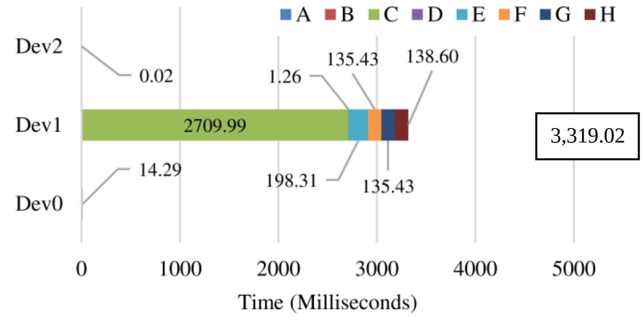

Figure 9. All kernels scheduled on the fastest device.

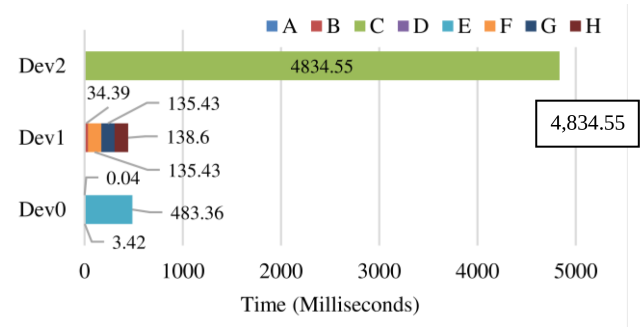

Figure 10. FCFS Scheduling.

References

Brandon G. Agby, Kalyan S. Perumalla, and Sudip K. Seal. Efficient simulation of agent-based models on multi-GPU and multi-core clusters. In *the 3rd International ICST Conference on Simulation Tools and Techniques*, SimuTools, 2010.

AMD Developer Central. *APARAPI: An Opensource API for Expressing Parallel Workloads in Java*. Available via http://developer.amd.com/tools-and-sdks/opencl-zone/aparapi/, 2011.

R. Dolbeau, F. Bodin and G. C. de Verdiere. One OpenCL to rule them all? In Proceedings of *Multi-/Many-core Computing Systems (MuCoCoS), 2013 IEEE 6th International Workshop on*, 2013.

Khronos Group. *OpenCL - The open standard for parallel programming of heterogeneous systems*. Available via http://www.khronos.org, 2013.

A. Hayashi, M. Grossman, J. Zhao, J. Shirako and V. Sarkar V. Accelerating Habanero-Java programs with OpenCL generation. In Proceedings of *ACM International Conference Proceeding Series*, 2013.

N. M. Ho, N. Thoai and W. F. Wong. Multi-agent simulation on multiple GPUs. *Simulation Modelling Practice and Theory, 57*: 118-132, 2015.

Chih-Wei Hsu and Chih-Jen Lin. A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks, 13*(2): 415-425, 2002.

JOCL. *jocl.org: Java bindings for OpenCL*. Available via http://www.jocl.org/, 2011.

X. Li, W. Cai and S. J. Turner. Supporting efficient execution of continuous space agent-based simulation on GPU. *Concurrency Computation*, 2016.

U. Lopez-Novoa, A. Mendiburu and J. Miguel-Alonso. Survey of performance modeling and simulation techniques for accelerator-based computing. *IEEE Transactions on Parallel and Distributed Systems, 26*(1): 272-281, 2015.

Sean Luke. *Multiagent Simulation and the MASON Library*. Available via https://cs.gmu.edu/~eclab/projects/mason

Worawan Marurngsith. Computing platforms for large-scale multi-agent simulations: The niche for heterogeneous systems. *Vol. 8669 LNCS. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pages 424-432, 2014.

A. Matoga, R. Chaves, P. Tom and N. Roma. A flexible shared library profiler for early estimation of performance gains in heterogeneous systems. In Proceedings of *High Performance Computing and Simulation, HPCS*, 2013.

R. Mokhtari and M. Stumm. BigKernel - High performance CPU-GPU communication pipelining for big data-style applications. In *Proceedings of The International Parallel and Distributed Processing Symposium, IPDPS*, 2014.

Hazel R. Parry and Mike Bithell. Large Scale Agent-Based Modelling: A Review and Guidelines for Model Scaling. In *Agent-Based Models of Geographical Systems*, pages 271-308, 2012.

Steven F. Railsback, Steven L. Lytinen and Stephen K. Jackson. Agent-based Simulation Platforms: Review and Development Recommendations. *SIMULATION, 82*(9): 609-623, 2006.

C. J. Rossbach, Y. Yu, J. Currey, J. P. Martin and D. Fetterly. Dandelion: A compiler and runtime for heterogeneous systems. In *Proceedings of the 24th ACM Symposium on Operating Systems Principles, SOSP*, 2013.

Rafael Sachetto Oliveira, Bernardo Martins Rocha, Ronan Mendonça Amorim, Fernando Otaviano Campos, Wagner Meira, Jr., Elson Magalhães Toledo and Rodrigo Weber Santos. Comparing CUDA, OpenCL and OpenGL Implementations of the Cardiac Monodomain Equations. In *Parallel Processing and Applied Mathematics,* Vol. 7204, pages 111-120, 2012.

K. Sato, K. Komatsu, H. Takizawa and H. Kobayashi. A History-Based Performance Prediction Model with Profile Data Classification for Automatic Task Allocation in Heterogeneous Computing Systems. In *Proceedings of Parallel and Distributed Processing with Applications, ISPA*, 2011.

Peter Stone and Manuela Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots, 8*(3): 345-383, 2000.

Y. Wen, Z. Wang and M. F. P. O'Boyle. Smart multi-task scheduling for Open CL programs on CPU/GPU heterogeneous platforms. In Proceedings of *2014 21st International Conference on High Performance Computing, HiPC*, 2014.

Y. Yan, M. Grossman and V. Sarkar. JCUDA: A programmer-friendly interface for accelerating java programs with CUDA. *Vol. 5704 LNCS. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pages 887-899, 2009.