




Porting CUDA-Based Molecular Dynamics Algorithms to AMD ROCm Platform Using HIP Framework: Performance Analysis

Evgeny Kuznetsov¹ and Vladimir Stegailov^{1,2,3} 

¹ National Research University Higher School of Economics, Moscow, Russia
v.stegailov@hse.ru

² Joint Institute for High Temperatures of RAS, Moscow, Russia

³ Moscow Institute of Physics and Technology, Dolgoprudny, Russia

Abstract. The use of graphics processing units (GPU) in computer data processing tasks has long ceased to be an unusual event. However, now GPU computing is nearly synonymous with CUDA, a proprietary framework for developing applications for Nvidia's GPU devices. It provides comprehensive documentation and excellent development tools. Meanwhile, the main competitor of Nvidia in the market for the production of GPU devices, the AMD company is developing its own Radeon Open Compute (ROCm) platform that features an application programming interface compatible with CUDA. The primary objective of this work is to investigate whether ROCm provides a worthy alternative to CUDA in the field of GPU computing. The work has two sub-objectives: the description of the programmers experience investigation during porting classical molecular dynamics algorithms from CUDA to ROCm platform and performance benchmarking of initial and resulting programs on GPU devices with modern architectures (Pascal, Vega10, Vega20).

Keywords: GPU computing · Parallel computing · CUDA · ROCm · Molecular dynamics

1 Introduction

Parallelization of computationally intensive algorithms on graphics accelerators is an integral part of the development of high-performance software. There are quite a number of libraries for developing code for graphics processors, such as OpenMP, OpenCL, OpenACC. They allow using the same code to compile programs for both the GPU from different manufacturers and the CPU. However, for the most part, they provide too high-level abstractions, and in tasks where performance is critical, the only choice is specialized GPGPU language-based programming models, such as CUDA and ROCm. Only they, due to low-level optimizations and the use of features of each platform, allow achieving the highest possible performance.

CUDA is a platform for writing applications for general-purpose computing on graphics processing units (GPGPU) designed by Nvidia. It was released about 11 years ago and made a long path. It has mature driver and runtime support, great debugging and profiling tools, detailed documentation and samples. Almost every algorithm that has high computational complexity and parallelization possibility was implemented in CUDA.

In contrast, ROCm is a part of the AMD’s “Boltzmann Initiative” announced in 2015. At the moment, ROCm is barely known platform for developing GPGPU applications that may run only on the specific subset of AMD graphics processing units [1]. It has a limited operating system (OS) support too: only a few Linux based OS are supported [1]. Besides, it has insufficient documentation and debugging support. So, ROCm is just at the beginning of its path and is developing rather quickly (e.g., version 1.9 has been announced on 15 September 2018 and version 2.3 has been announced on 12 April 2019). However, already now AMD GPUs have enough computational power to compete on equal terms with Nvidia GPUs [2].

An appealing aspect of the ROCm platform is its open-source character that improves portability and corresponds to the best practices of community codes development.

Moreover, AMD’s graphics accelerators have another important advantage. Scientific calculations mainly use double-precision floating point arithmetic, whose performance in graphics chips is artificially limited by manufacturers to encourage the purchase of much more expensive devices. For most AMD GPUs, this limitation is significantly lower than that of Nvidia, which makes it possible for enthusiasts to use devices of the middle price segment to develop high-tech software.

This work aims to perform a readiness review of the ROCm platform to production development by porting one real-world CUDA application on the ROCm platform and evaluating performance differences between them.

As an example of a real-world CUDA application CoMD-CUDA is taken. CoMD [3] is a mini-application that represents a reference implementation of conventional classical molecular dynamics algorithms and workloads. CoMD-CUDA [4] is a CUDA implementation of this mini-application.

The choice of this mini-application is not accidental. The tasks of classical molecular dynamics provide about 20–30% of the load of the largest supercomputers in the world [5], a significant part of which have a heterogeneous architecture and use GPUs [6]. Therefore, this class of algorithms deserves a focused consideration.

2 Literature Review

According to Jon Peddie Research press release [7], discrete AMD GPUs occupy a significant market share (Table 1). However, they are mainly used only for games and for solving specific tasks, such as cryptocurrency mining and 3D rendering, while they have significant potential in the field of general-purpose computing.

Table 1. Discrete GPU market share

Supplier	Q2'18	Q3'18	Q3'17-Q3'18
AMD	25.7%	36.1%	27.2%
Nvidia	74.3%	63.9%	72.8%

The ROCm platform as a relatively new technology is a rare subject in the articles devoted to performance studies of parallel algorithms on GPU. No one has yet made a thorough comparison of the performance of the ROCm platform with the CUDA platform. Sometimes (e.g. see [8]) this tends to be caused by the complexity of the installation and testing processes.

When it comes to cross-platform analysis, most authors direct their attention to comparing the performance of the CUDA platform and the Open Computing Language (OpenCL) heterogeneous computing framework. A thorough research [9] was made to find out if OpenCL technology can provide efficiency comparable to CUDA. The answer is yes, but it will require the developer to make complex adjustments to the program execution parameters. Besides, the process of porting CUDA application to OpenCL may be even more complicated, but this aspect of the study was not reported.

Another limitation of the OpenCL technology is the inability to use architectural features such as AMD's global data share and Nvidia's inline PTX assembly. It is a fairly common practice to have independent, optimized for each platform implementation of bottlenecks in the program.

Interestingly enough for the purposes of this paper, the article [2] is devoted to comparing the performance of all three GPU programming frameworks supported by the ROCm platform: HIP, OpenCL, and HC++. According to it, the following conclusions can be drawn:

- HIP is the best performing high-level framework from choices that ROCm support.
- HIP does not add any noticeable overhead to the workloads and the execution times comparing to the corresponding CUDA implementation.

In addition, the authors compare the speed of the two vendor implementations of deep neural networks for the CUDA and ROCm architectures: cuDNN and MIOpen, respectively. The implementations of both manufacturers on devices similar in their reported single precision floating point calculation capabilities showed approximately equivalent timings. In this work we present similar results for hybrid MD algorithms.

3 Methods

A vast number of parallel algorithms and applications have been developed using the CUDA platform. To facilitate their porting process, ROCm provides a HIP

framework [10], which provides CUDA-compatible API, as well as the *hipify* tool for semi-automatic translation of CUDA runtime library calls to ROCm calls. Moreover, the HIP platform allows executing the resulting code on both AMD devices and Nvidia graphics accelerators. In the latter case, the functions of the HIP library are simple wrappers over the corresponding functions of CUDA, which allows developing code for CUDA-compatible devices with near-zero overhead [2].

The first phase of this work is porting the CoMD-CUDA application to the ROCm platform using the HIP library. It includes several sub-steps:

1. Using the *hipify* tool to translate CUDA runtime library calls.
2. Revision of the GPU kernels under the architectural features of the ROCm platform.
3. Creation a cross-platform (AMD/Nvidia) CMake build project.

The second phase consists in analyzing the performance of the resulting application on several graphics accelerators from AMD and Nvidia manufacturers of the same class. The list of used video cards and their brief characteristics presented in Table 2.

Table 2. The comparison of main features of GPUs considered

	RX480	Vega 56	Radeon VII	GTX 1070
Manufacturer	AMD	AMD	AMD	Nvidia
Stream processors	2304	3584	3840	1920
Base frequency	1120 MHz	1156 MHz	1750 MHz	1506 MHz
Memory size	4 Gb	8 Gb	16 Gb	8 Gb
Memory type	GDDR5	HBM2	HBM2	GDDR5

In case the results have an inexplicable variation in execution time, a more thorough study of the performance drop-down by profiling is needed. Moreover, the CoMD-CUDA application is hard-wired for optimal performance for the GK110 graphics chip. Thus, the comparison will be incomplete without optimization for the architecture features of AMD GPUs. This step, in turn, requires some sophisticated register and shared memory usage analysis [11], which is supposed to be performed using the AMD architecture assembly code investigation.

ROCm uses Heterogeneous Compute Compiler (HCC) for compilation both host and device code. It is based on the open-source Clang C++ compiler and Low Level Virtual Machine (LLVM) framework [12] that provides powerful optimization possibilities. For the purposes of this study, it is interesting to compare HCC code generation quality with NVIDIA proprietary CUDA Compiler (NVCC). To accomplish this goal, a side by side examination of the generated NVCC and HCC codes are made.

The selected CoMD-CUDA application is best suited for all phases of work. It implements several different approaches to perform modeling in molecular dynamics problems [4]. The source code contains a wide range of operations, such as working with shared memory, double precision arithmetic, atomic operations, synchronization of threads, cross warp data exchange and asynchronous execution, which fully covers all typical scenarios of using GPU. Thus, a comparison of the performance of the two platforms using the CoMD-CUDA can be considered more relevant than when conducting synthetic tests.

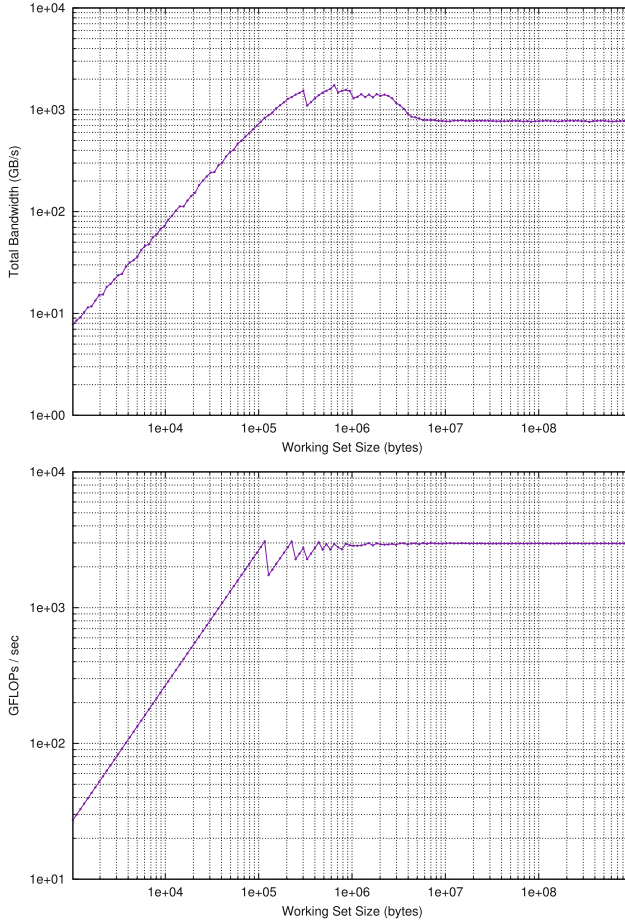


Fig. 1. ERT memory bandwidth and ERT FLOPS/sec for the case of 64 threads/640 blocks.

4 Results

Since the HPC community started to pay more attention not only to peak performance, but also to such parameters as memory throughput and caches, many other metrics and models became available to evaluate the performance effectiveness. In this paper we use the Berkeley lab's Empirical Roofline Toolkit [13]. This intuitive model bounds the floating point peak computation performance, the memory bandwidth and the arithmetic intensity, also taking into account such effects as caching, non-uniform memory access and instruction-level parallelism. The results for the newest Radeon VII GPU are presented on Figs. 1 and 2.

The CUDA-version of the ERT kernel has been hipified without any problems. However the current version of HCC compiler (ROCm version 2.2) can not properly detect FMA operation in the ERT kernel and only manual inclusion of these assembler instruction in the kernel allowed us getting full computational performance of Radeon VII (3 TFLOPS/sec). In this case the memory bandwidth is maximized as well and reaches 77% of the theoretical value of 1 TB/sec.

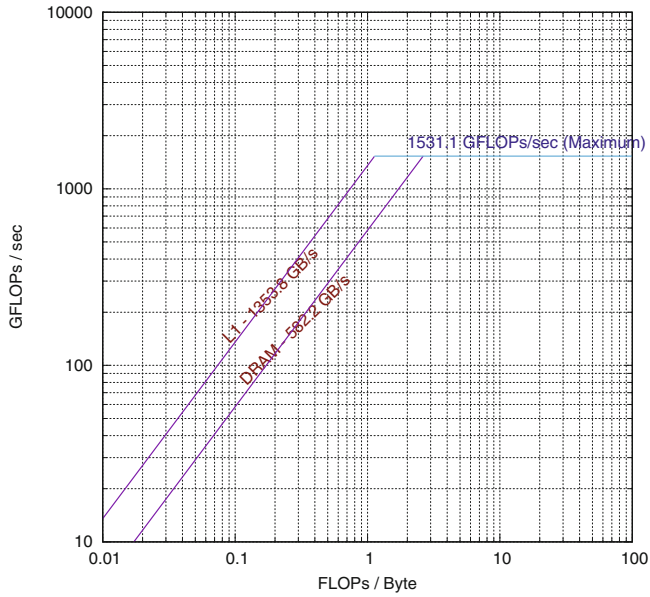
This paper addresses the gap in the area of developing high-performance GPGPU applications by comparing two modern GPGPU platforms: CUDA and ROCm. Due to the novelty and insufficient prevalence of the ROCm platform, this work also aims at examining the process of migrating existing CUDA applications to a new platform.

Despite the stated simplicity of porting CUDA applications to the ROCm platform, some problems have been met due to the lack of full-fledged examples, insufficient documentation and the presence of a large number of GPU kernels optimized for the Nvidia architecture as part of CoMD-CUDA.

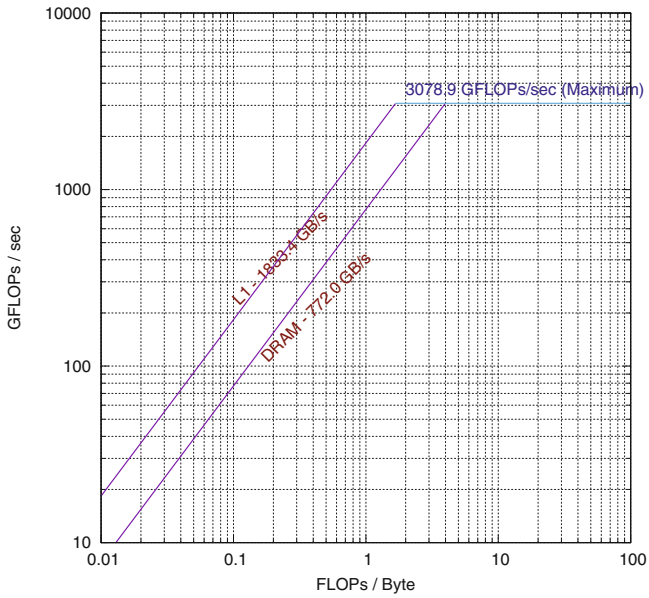
The process of porting a CUDA application to the ROCm platform, as well as identification of the points that should be paid attention to, are illustrated on Fig. 3.

CoMD-CUDA contains two interatomic potential models for MD simulations: the standard Lennard-Jones pair potential and the EAM manybody potential. The `thread_atom` and `cta_cell` methods for neighbor search are considered for the LJ model, the `wrap_atom` method is considered for the EAM model as well.

The performance of CoMD algorithms on AMD video cards without corresponding optimizations is expected to be lower than on the Nvidia platform. When the optimal occupancy of all stream processors of the Vega 56 accelerator reached, its results can exceed those of GTX 1070. The results are summarized on Figs. 4 and 5.



(a) implicit FMA



(b) explicit FMA

Fig. 2. Roofline obtained on AMD Radeon VII using implicit and explicit FMA operations.

```

grid = (s->n_boundary_cells + (block/WARP_SIZE)-1)/(block/WARP_SIZE);
- UpdateBoundaryList<<<grid, block>>(s->gpu, s->gpu.b_list, s->n_boundary_cells, cell_offsets1, s->boundary_cells);
+ hipLaunchKernelGGL((UpdateBoundaryList), dim3(grid), dim3(block), 0, 0, s->gpu, s->gpu.b_list, s->n_boundary_cells, cell_offsets1, s->boundary_cells);
CUDA_GET_LAST_ERROR

grid = (n_interior_cells + (block/WARP_SIZE)-1)/(block/WARP_SIZE);
- UpdateBoundaryList<<<grid, block>>(s->gpu, s->gpu.i_list, n_interior_cells, cell_offsets2, s->interior_cells);
+ hipLaunchKernelGGL((UpdateBoundaryList), dim3(grid), dim3(block), 0, 0, s->gpu, s->gpu.i_list, n_interior_cells, cell_offsets2, s->interior_cells);
CUDA_GET_LAST_ERROR

- cudaMemcpy(&s->gpu.b_list.n, cell_offsets1 + s->n_boundary_cells, sizeof(int), cudaMemcpyDeviceToHost);
- cudaMemcpy(&s->gpu.i_list.n, cell_offsets2 + n_interior_cells, sizeof(int), cudaMemcpyDeviceToHost);
+ hipMemcpy(&s->gpu.b_list.n, cell_offsets1 + s->n_boundary_cells, sizeof(int), hipMemcpyDeviceToHost);
+ hipMemcpy(&s->gpu.i_list.n, cell_offsets2 + n_interior_cells, sizeof(int), hipMemcpyDeviceToHost);

- cudaFree(partial_sums);
- cudaFree(cell_offsets1);
- cudaFree(cell_offsets2);
+ hipFree(partial_sums);
+ hipFree(cell_offsets1);
+ hipFree(cell_offsets2);

bool flag = r2 <= rCut2 && r2 > 0.0;
- unsigned int x;
+ unsigned long long int x;
int n;
if(x == __ballot(flag))
{
//Scan
- x = x & mask2;
- n = __popc(x);
+ n = __popcll(x);
- x = x & mask;
- const int p = __popc(x);
+ const int p = __popcll(x);
const int place = nNeighbors + p;
- if (flag) mytemp[(place/memoryPackSize) * 32 * memoryPackSize + (place & (memoryPackSize-1))] = joff;
+ if (flag) mytemp[(place/memoryPackSize) * WARP_SIZE * memoryPackSize + (place & (memoryPackSize-1))] = joff;
nNeighbors += n;
}

```

Fig. 3. Examples of the CUDA to ROCm translations made by the hipify tool. Corrections of the GPU kernel due to different warp sizes of Nvidia and AMD GPUs.

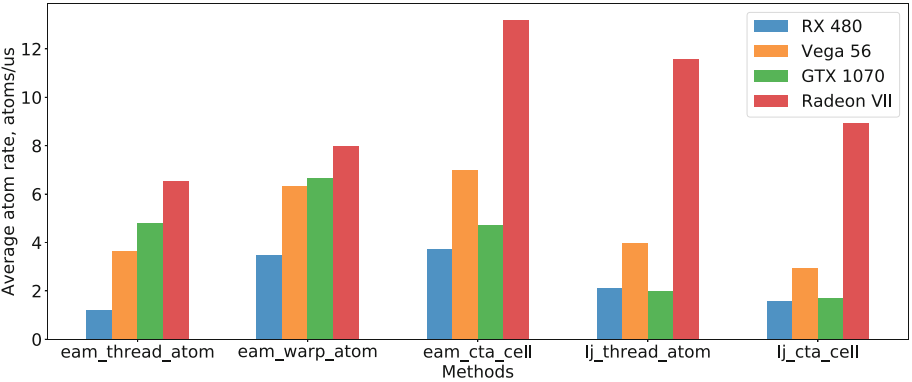


Fig. 4. Performance of different methods (problem size $50 \times 50 \times 50$)

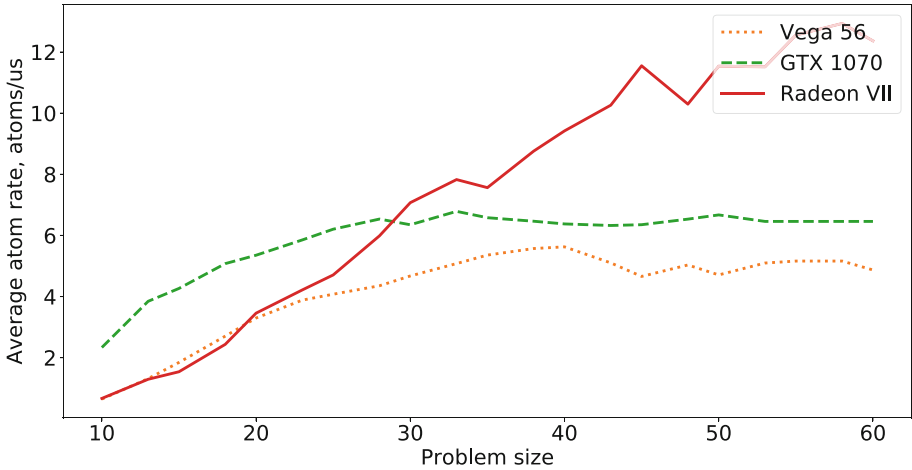


Fig. 5. Performance of the warp atom method on different problem sizes

5 Conclusion

New technologies require attention from software developers. It is essential to provide the necessary feedback, this allows to quickly correct occurring errors and develop the product.

AMD is the second-largest supplier of CPU and discrete GPU in the world. Its processors have considerable potential in the field of high performance computing. The significance of AMD's ROCm platform is hard to overestimate - it provides tools for developing cross-platform GPGPU applications that can run on both AMD video accelerators and Nvidia devices.

ROCm is still under development, so far there have been too few examples of its successful application. Therefore, it is necessary to discuss this technology, develop methods for and share experiences and results of its use, as well as track its production readiness.

To this aim, this work is going to start such a process by providing a comprehensive review of cross-platform ROCm framework and investigation of its performance in a real-world scenario.

Our results show that the transfer of a real-world application from CUDA to HIP does not require major modifications of the code and gives the possibility to run the application considered on AMD GPUs without performance degradation.

References

1. ROCm Install. <https://rocm.github.io/ROCmInstall.html#hardware-support>. Accessed 15 Apr 2019
2. Sun, Y., et al.: Evaluating performance tradeoffs on the radeon open compute platform. In: 2018 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS) (2018)

3. Mohd-Yusof, J.: Codesign molecular dynamics (CoMD) proxy app. In: Los-Alamos National Lab, Technical report (2012)
4. Mohd-Yusof, J., Sakharnykh, N.: Optimizing CoMD: a molecular dynamics proxy application study. In: GPU Technology Conference (GTC) (2014). <http://on-demand.gputechconf.com/gtc/2014/presentations/S4465-optimizing-comd-molecular-dynamics.pdf>. Accessed 15 Apr 2019
5. Norman, G., et al.: Why and what supercomputers of the exaflops class are needed in the natural sciences. *Program Syst.: Theory Appl.* **6**(4), 243–311 (2015)
6. November 2018 — TOP500 Supercomputer Sites. <https://www.top500.org/lists/2018/11/>. Accessed 15 Apr 2019
7. Peddie, J., Dow, R.: Jon Peddie Research releases its Q3, 2018 add-in board report. <https://www.jonpeddie.com/press-releases/jon-peddie-research-releases-its-q3-2018-add-in-board-report>. Accessed 15 Apr 2019
8. Turner, D., Andresen, D., Hutson, K., Tygart, A.: Application performance on the newest processors and GPUs. In: Proceedings of the Practice and Experience on Advanced Research Computing - PEARC (2018)
9. Fang, J., Varbanescu, A., Sips, H.: A comprehensive performance comparison of CUDA and OpenCL. In: International Conference on Parallel Processing (2011)
10. ROCm-Developer-Tools/HIP. <https://github.com/ROCm-Developer-Tools/HIP>. Accessed 15 Apr 2019
11. Aaltonen, S.: Optimizing GPU occupancy and resource usage with large thread groups. <https://gpuopen.com/optimizing-gpu-occupancy-resource-usage-large-thread-groups/>. Accessed 15 Apr 2019
12. The LLVM Compiler Infrastructure Project. <https://llvm.org/>. Accessed 15 Apr 2019
13. Lo, Y., et al.: Roofline model toolkit: a practical tool for architectural and program analysis. In: Jarvis, S.A., Wright, S.A., Hammond, S.D. (eds.) PMBS 2014. LNCS, vol. 8966, pp. 129–148. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-17248-4_7