QCDGPU: an Open-Source OpenCL Tool for Monte Carlo Lattice Simulations on Heterogeneous GPU Cluster

Vadim Demchik[1], *Natalia Kolomoyets*[1]

[1]Dnipropetrovsk National University, Gagarin ave. 72, 49010 Dnipropetrovsk, Ukraine

DOI: http://dx.doi.org/10.3204/DESY-PROC-2014-05/29

QCDGPU is an open-source tool for Monte Carlo lattice simulations of the SU(N) gluodynamics and O(N) models. In particular, the package allows to study vacuum thermodynamics in external chromomagnetic fields, spontaneous vacuum magnetization at high temperature in the SU(N) gluodynamics and other new phenomena. The QCDGPU code is implemented in the OpenCL environment and tested on different OpenCL-compatible devices. It supports single- and multi-GPU modes as well as GPU clusters. Also, the QCDGPU has a client-server part for distributed simulations over VPN. The core of Monte Carlo procedure is based on the PRNGCL library, which contains implementations of the most popular pseudorandom number generators. The package supports single-, mixed- and full double-precision including pseudorandom number generation. The current version of the QCDGPU is available online.

1 Introduction

Intensive development of computational methods, along with a rapid progress of computer technology has opened up the possibility of studying many problems of quantum field theory that can not be resolved within the analytical approach. One of such computational methods is lattice Monte Carlo (MC) method, which significantly expanded our understanding of many high-energy phenomena. Lattice MC simulations are based on the procedure of infinite-dimensional path integral calculation with the finite sums on a discrete space-time lattice. Due to locality of basic algorithms, the method is well suited for massively parallel computational environment. Therefore the advent of graphics processing units (GPU) as affordable computational platform reasonably causes a great interest.

Historically, NVIDIA CUDA has become the first widespread platform for general purpose computing on GPUs. However, the requirement of cross-platform compatibility yields to the development of the new open standard computational language OpenCL. Currently the share of OpenCL usage reaches 30% of all scientific researches involving GPUs [1].

Nowadays, there are several software packages to provide MC lattice simulations on GPUs (for example, [2], [3], [4], [5] and [6]). Nevertheless, existing software tools cannot cover many modern high-energy physics problems like investigations of O(N) scalar models or study of SU(N) vacuum in (chromo)magnetic fields. Therefore, we have developed a new software package, QCDGPU to provide the possibility of such kind of explorations. The general aim of the QCDGPU software [7] is the production of lattice gauge field configurations for O(N) and SU(N) (with or without (chromo)magnetic fields) models with further statistical processing of measurable quantities on GPUs.

2 Structure of QCDGPU

A full platform-independence principle is taken as a basis while constructing the QCDGPU package. Correspondingly, OpenCL was chosen as a GPGPU platform for the package to provide independence on GPU hardware vendors. Host code is implemented in C++ language with minimal external library dependence, which ensures it runs equally well on Windows and Linux operating systems.

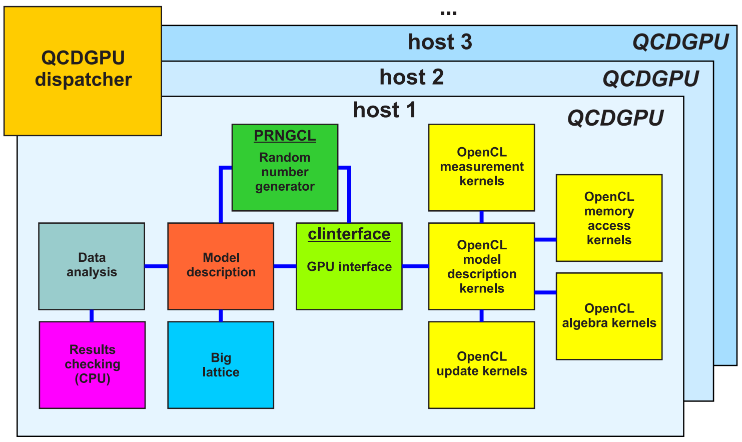

Figure 1: Structure of the QCDGPU package

From the programming point of view the QCDGPU package is a set of independent modules that provide different functions. The structure of QCDGPU is shown as a schematic diagram in Fig. 1.

The core of the package is the CLInterface module which unifies the interaction of the host code with different computing devices. It performs initialization and finalization of computing devices, prepares and adjusts memory objects and OpenCL-kernels for execution, controls and schedules the program flow, etc. All references to memory objects and kernels are organized through integer identifiers. This approach removes direct dealing with OpenCL types and structures, which essentially simplifies programming.

Another important function of the CLInterface module is a caching of previously compiled OpenCL programs to reduce startup time of their execution by creating .bin-files with compiled code for a particular device with distinct computational parameters. The reason to implement an external caching is caused by the fact that not all OpenCL SDK vendors provide correct caching and tracking of dependent source code files. Compilation-specific and additional parameters (like computing platform and device, program options, etc.) are written to the corresponding .inf-files. The uniqueness and code modification is controlled by MD5-hash calculation for each OpenCL source code. Any code or compiler parameters changing leads to a new rebuilding of the program with corresponding .bin- and .inf-files generation. These files are created for a new startup parameters set. If MD5 and other parameters fit the existing .inf-file, the CLInterface module uses the previously built program from .bin-file. If only MD5-hash is changed (that means kernel source code update), old .bin and .inf files are overwritten. Such a technique provides significant runtime speedup for real MC simulations, because usually a limited set of parameters is used to perform such computer experiments.

Moreover, instant profiling of kernels and host-device data transfer are also performed with the module CLInterface. Due to a slightly slowing up of the package by profiling, it is turned off by default. Device errors handling is performed with CLInterface too. All errors are collected in .log-file. In case of critical errors the program stops.

Model description block is represented by the SUNCL and ONCL modules, which describe SU(N) and O(N) models, correspondingly. The quantum fields in the SU(N) case are represented by $N \times N$ complex matrices living on lattice links. Not all the components of these matrices are independent, so it reduces desired storing data. In the O(N) model case the scalar fields are N-vectors placed in lattice sites. While organizing such matrices and vectors, it is taken into account a common GPU memory architecture that is optimized for the storage of 4-vectors. Such storage format provides additional significant speedup for a program on all computing devices. The whole lattice is a multidimensional set of data that is presented in the QCDGPU package in the following way. The first dimension (fastest index) enumerates lattice site. The next dimension contains the spatial directions of lattice links (in the case of O(N) models this is absent). The last dimension (slowest index) is connected with the gauge group.

The lattice update is realized by decomposing sites in odd and even ones (checkerboard scheme), and, if necessary, into separate parts to handle big lattices. For SU(N) link update a (pseudo)heat-bath algorithm [8, 9] is used, while for O(N) update an improved Metropolis algorithm [10] is implemented.

Pseudorandom numbers for MC simulations are produced with the PRNGCL library [11]. The most widely used generators in HEP lattice simulations are realized in it (RANMAR, RANECU, XOR7, XOR128, MRG32K3a and RANLUX with various luxury levels). Since almost all realized generators (apart from MRG32k3a) have not a general scheme for multi-thread execution, parallelization is made by the random seeding method – each computing thread uses its own seed table, which is uniquely initialized during startup. Initialization of PRNG is made by one integer parameter `RANDSERIES`. If it is set to zero, system time is chosen for its value. PRNGCL library provides very important feature for MC simulations – repeatability of results on different hardware and operating systems. PRNGCL possesses the possibility to produce pseudorandom numbers with single or double precision to study tiny effects on a lattice.

The possibility to work with big lattices is implemented in QCDGPU through dividing the whole lattice into several parts (Big lattice module). Each part can be simulated on one or several devices. In the case of a multi-GPU system or heterogeneous GPU cluster, dividing the lattice into unequal parts provides the possibility to hide the differences in performance of the different devices. The slowest device gets the smallest lattice part, while the most powerful device operates with biggest lattice portion. The division is implemented along the x spatial axis, because the package is designed mainly for investigation of finite temperature effects, where $L_t < L_s$, $L_{t/s}$ being the lattice size in the temporal and spatial directions, respectively.

In order to hide data transfer among computing devices, additional checkerboard decomposition is used: the odd and even parts of the lattice are updated alternately. In this case, the information exchange of lattice boundary layers is performed in asynchronous mode – while even sites are being updated, transfer of the information at the boundary layer sites takes place and vice versa. That is why it is preferable to divide the lattice into an even number of parts.

In the Data analysis module the statistical analysis of the data collected during one program run is performed. The averages and variances over each run of measured quantities are calculated here. The Data analysis module also provides data distribution analysis for particular measurements.

The Results checking module serves to control the correctness of obtained results. It compares the measurements on the last gauge configuration obtained on GPU with results of traditional sequential approach on CPU. For deep debugging and dynamical comparison of results the Results checking module can independently produce lattice gauge configurations on the same pseudorandom numbers as the device. This module is turned off by default to speed up the program and can be activated.

One of the most widely spread error while developing lattice-based programs is incorrect linking among different lattice sites. Obviously, such an error may bring unnecessary lattice connections and lead to nonphysical results. To overcome such problems the QCDGPU package provides additional lattice initialization with the so-called GID update. Unlike "cold" (ordered) and "hot" (disordered) starts, in this case the lattice is filled with predefined numbers (GID start), which are uniquely determined with each computing thread. Update is also performed with some predefined numbers instead of PRNs. To exclude rounding errors in the heat-bath procedure, the expression for the coefficients of the identity matrix of SU(2) subgroups is changed and the obtained configuration is always accepted. This method provides a good possibility to compare MC simulation results with some alternative realization of the key procedures (for example, mathematical packages).

3 Run description

In actual calculations it is convenient to run QCDGPU in pair with an external program GPU-dispatcher, which prepares the init.dat file with run parameters (like lattice geometry, gauge group, precision and so on). init.dat is an ordinary text file containing those parameters in the form `PARAMETER = value`. The complete list of run parameters may be found in the poster [12].

The program QCDGPU is run in the standard way, `QCDGPU_root_directory> ./QCDGPU init.dat`. During execution it creates several files program$i$.bin and program$i$.inf, where i is index number of the binary file. The .bin files contain compiled OpenCL kernels. Additional information needed for those programs is written in the .inf files.

The possibility of regular saving of the program state, including the state of pseudorandom number generators, is realized in QCDGPU for resuming the interrupted simulation. It allows to interrupt the simulation at any time, and provides basic fault tolerance (power or hardware failure). Saving frequency can be set manually. Saved .qcg-file with the computational state is portable – the computation can be continued on another device.

The result of the program execution is written to the model$n$-yy-Month-dd-hh-mm-ss.txt file, which contains run parameters, average over run values of measured quantities, their variances over run, the table of averaged over configuration quantities. If Results checking module is on then CPU comparison is included in the output file.

After program execution the file finish.txt is created. It prevents QCDGPU from premature run. To start the program one more time, this file must be removed. GPU-dispatcher creates a new init.dat file, removes finish.txt, and QCDGPU runs with new startup parameters. Such asynchronous task scheduling via operating system tools causes extremely low CPU load in stand-by mode.

Using QCDGPU in pair with GPU-dispatcher allows to perform MC simulations on several hosts at the same time. Each copy of the main computational program is launched on the corresponding host, while parameters passing and launch control are carried out by GPU-dispatcher. It sequentially looks through folders and sets a new task for the first free host.

4 Performance results

To estimate the performance of the QCDGPU package, several devices were used: NVIDIA GeForce GTX970, NVIDIA GeForce GTX980 (NVIDIA CUDA SDK 7.0), AMD Radeon R9 270X, AMD Radeon HD7970 (AMD APP SDK 3.0 Beta), Intel Xeon Phi 31S1P and Intel Xeon CPU E5-2609 (Intel OpenCL SDK 1.2 4.6.0.92). In Table 1 the timings for one sweep in seconds are presented for SU(3) gauge theory for three lattice sizes for single, mixed and double precisions. The mixed precision means that all calculations are done with double precision except for PRNs production. Each sweep consists of the lattice update by multihit heat-bath method (10 hits are set) and measurement of the Wilson action.

	Single			Mixed			Double		
Lattice / Device	16^4	24^4	32^4	16^4	24^4	32^4	16^4	24^4	32^4
NVIDIA GTX980	0.01	0.04	0.13	0.02	0.08	0.25	0.03	0.20	0.69
NVIDIA GTX970	0.01	0.06	0.18	0.02	0.10	0.34	0.04	0.27	0.94
AMD R9 270X	0.02	0.10	0.32	0.05	0.22	0.68	0.14	0.82	2.65
AMD HD7970	0.02	0.08	0.26	0.03	0.14	0.46	0.12	0.66	2.28
E5-2609 CPU	0.19	0.93	2.89	0.23	1.14	3.56	0.59	3.60	11.77
Intel Xeon Phi	0.47	2.29	7.18	0.57	2.65	8.20	1.81	11.3	36.9

Table 1: Timings (in seconds) of one sweep for SU(3) gluodynamics

It can be seen that the computing time depends linearly on the lattice volume for almost all devices. Despite of significant difference in the peak performance for modern GPUs, QCDGPU shows much better ratio of double/single precision timings. The run on Intel Xeon Phi is performed just to test compatibility of the QCDGPU package, the program is not optimized for Intel MIC architecture.

5 Conclusions

QCDGPU package is a new software tool for MC lattice simulations on OpenCL-compatible computing devices. It provides a possibility to study vacuum thermodynamics in extreme conditions and measuring of nonconventional quantities like Polyakov loop and action spatial distribution, Cartesian components of SU(N) electromagnetic field tensor. QCDGPU is also applicable for simulating scalar O(N) model in the quantum field theory on GPUs.

Underlying full platform-independence principle makes it possible the usage of the QCDGPU package on different OpenCL-compatible computing hardware as well as for most popular operating systems without any source code modifying. Additionally, the package almost does not load the central processor. Together with platform-independence it provides a great opportunity to use the package in local networks as a background software for lattice simulations.

The package demonstrates the best performance results in multi-GPU simulations for trivial parallelization of a task. This approach consists of whole lattice simulation with certain set of adjustable parameters by each involved computing device. The main bottleneck of GPU computing is still relatively low throughput of host-to-device connection. Moreover, interconnection between computing nodes in GPU cluster possesses even lower throughput then host-to-device connection. Obviously, it is better to carry out simulations directly in device memory to eliminate data transfer to/from device. However, for large lattices this is impossible due to the lack of device memory and the only way is to separate lattices into several parts on one host system.

The package instantly provides data production with single, mixed and double precision. The usage of double precision on a regular basis may lead to unnecessary waste of computing time. To overcome this issue internal pseudorandom number generators are built to provide double precision numbers from numbers with single precision by applying a special procedure [13].

Undoubtedly, QCDGPU package is continuously developing to provide additional features that can improve performance and extend the class of tasks solvable with the package. One of such features is autotuning of start-up parameters with internal micro-benchmarks of involved computing devices.

The source codes of the current version of the QCDGPU package and some examples of result files are publicly available [7].

References

[1] High performance computing on graphics processing units, `http://hgpu.org/`.

[2] M. A. Clark, R. Babich, K. Barros, R. C. Brower and C. Rebbi, Comput. Phys. Commun. **181**, 1517 (2010) [arXiv:0911.3191 [hep-lat]].

[3] N. Cardoso and P. Bicudo, Comput. Phys. Commun. **184**, 509 (2013) [arXiv:1112.4533 [hep-lat]].

[4] M. Schrock and H. Vogt, PoS LATTICE **2012**, 187 (2012) [arXiv:1209.4008 [hep-lat]].

[5] M. Bach, V. Lindenstruth, O. Philipsen and C. Pinke, Comput. Phys. Commun. **184**, 2042 (2013) [arXiv:1209.5942 [hep-lat]].

[6] M. Di Pierro, PoS LATTICE **2013**, 043 (2013).

[7] QCDGPU, `https://github.com/vadimdi/QCDGPU`

[8] A. D. Kennedy and B. J. Pendleton, Phys. Lett. B **156** 393 (1985).

[9] N. N. Cabibbo and E. Marinari, Phys. Lett. B **119** 387 (1982).

[10] M. Bordag, V. Demchik, A. Gulov and V. Skalozub, Int. J. Mod. Phys. A **27** 1250116 (2012).

[11] V. Demchik, ch. 12 in "Numerical Computations with GPUs," ed. V. Kindratenko, Springer (2014).

[12] N. Kolomoyets, V. Demchik, GPUHEP (2014), `https://agenda.infn.it/getFile.py/access?contribId=8&sessionId=15&resId=0&materialId=poster&confId=7534`.

[13] V. Demchik and A. Gulov, arXiv:1401.8230 [cs.MS], 1–4 (2014).