Abstract—Recently, many particle physics applications can be parallelized by using multicore platforms such as CPU and GPU. In this paper, we propose a parallel processing approach for Quantum ChromoDynamics(QCD) application by using both CPU and GPU. Instead of distributing the parallelizable workload to either CPU or GPU, we distribute the workload simultaneously into both CPU and GPU by using OpenCL. Based on the experimental results with the lattice QCD code, we confirm that the proposed parallel processing approach can provide better performance than the typical parallel processing approach by utilizing the given resource maximally.

Keywords—QCD, parallel processing, OpenCL.

I. INTRODUCTION

The structure of subnuclear particles like the proton and neutron is dictated by the dynamics of quarks and gluons. The strong force that binds them is described using Quantum ChromoDynamics(QCD). Lattice QCD[1] is a discretized version of this theory that makes it amenable to numerical simulations. The lattice QCD computation is very structured, and many parallel solutions are reported[2-4]. In this paper, we focus on parallelizing the lattice QCD computation by using both CPU and GPU. Recently, OpenCL[5] has been defined as a standard for heterogeneous parallel computing[6]. It provides a cross-platform framework for writing software able to run on different kinds of devices, from multicore CPUs to GPUs. That is, a parallel program written with OpenCL can be executed on either CPU or GPU. Generally, it is true that GPU can provide better performance than CPU for the structured applications such as lattice QCD. However, we consider large lattice sizes where the GPU-only execution is impossible due to the GPU memory limitation. Furthermore, a current multicore CPU is also a powerful processor and thus can reduce the total execution time when used together with GPU.

We propose a load balancing approach which can overcome the performance limit of either CPU-only execution. We first parallelize the lattice QCD computation with OpenCL, and measure the maximum lattice size where GPU can execute. Then, we partition the parallelized workload into two parts, based on the relative performance of GPU over CPU as well as the GPU memory limitation. Finally, we assign the GPU-portion of workload to GPU by using a non-blocking command, and then assign the remaining parallel portion to CPU without waiting for a result from GPU. By reducing the idle time on either CPU or GPU, we can overlap the GPU execution maximally with the CPU execution.

The rest of the paper is structured as follows. Section 2 explains OpenCL[4] and lattice QCD. Section 3 describes our proposed load balancing approach. The experimental results are given in Section 4, and conclusions are provided in Section 5.

II. BACKGROUND

A. OpenCL

OpenCL[5] is an open standard aimed at providing a programming environment suitable to access heterogeneous architectures. In particular, OpenCL allows to execute computational workloads on various multicore processors. Considering the increasing availability of such types of processors, OpenCL is playing a crucial role in enabling portable applications to access a wide range of computational resources. To achieve this aim, various levels of abstraction have been introduced in the OpenCL model.

- **Platform** performs an abstraction of the number and type of computing devices in a hardware platform. At this level are made available to developers the routines to query and to manage the computing devices, to create the contexts and work queues for submission of sets of instructions called **kernels**.

- **Execution** is based on the concept of kernel which is a collection of instructions executed on the computing device, multicore CPU or GPU, called OpenCL **device**. An OpenCL application can be divided in two programs: **host** and **kernel**. The host program is executed on CPU. It defines the context for the kernels and manages their execution. Especially, when a kernel is submitted for execution by the host, an index space
is defined. An instance of the kernel executes for each point in this index space. This kernel instance is called a work-item and is identified by its point in the index space, which provides a global ID for the work-item. Each work-item executes the same code on distinguished data. That is, work-items are organized into work-groups providing a more coarse-grained decomposition of the index space.

**Language** describes the syntax and programming interface for writing kernels (set of instructions that execute on computing device such as multicore CPUs or GPUs).

### B. QCD

One of the main approaches to study High Energy Physics phenomena is the lattice Monte Carlo simulation. For example, a formulation of QCD on a space-time lattice, called a Lattice Gauge Theory (LGT), allowing to compute infinite-dimensional path integral with the procedure of computation of finite sums was proposed[1]. LGT has many important features which makes it possible to study low-energy limit of QCD, which is impossible by analytic methods. In the limit of an infinite number of lattice sites and zero lattice spacing, LGT becomes an ordinary quantum field theory. Numerical results, obtained by means of lattice approximation, depend on the number of lattice sites (denoted as lattice size), and thus computation with large lattice sizes is preferable. Furthermore, some physical phenomena can be observed with large lattice sizes, because small lattices are not sensitive to such effects. However, computation with large lattice sizes increases the required workload, and parallel processing approaches are needed[2-4].

In this paper, we consider the open-source QCDGPU[4] which allows to study SU(2) and SU(3) gauge theories as well as O(N) models. Although QCDGPU was implemented with OpenCL, the code was executed on either CPUs, GPUs, or GPU clusters, not on a combination of CPU and GPU (i.e., heterogeneous computing). We modify QCDGPU such that the code can be executed on a heterogeneous computing configuration by utilizing the given resource maximally.

### III. PARALLEL QCD BY USING OPENCL

GPU can provide high performance for the physics calculations such as Lattice QCD. However, we consider large lattice sizes where the GPU-only execution is impossible due to the memory limitation of GPU. For example, if the lattice size exceeds the memory limitation of GPU, a fatal error occurs. To solve the problem, the way with multi-GPU configurations may overcome the memory limitation of a single GPU. However, we focus on how to utilize the given resource maximally if we have a set of a single GPU and a single CPU (at least one CPU is needed as a host device, regardless of the number of GPUs).

That is, we propose a way to overcome the memory limitation of a single GPU by using GPU-CPU hybrid computing. Furthermore, current CPUs have multiple cores. When an application is running with a single GPU configuration, only one thread is assigned to a host CPU core (i.e., a core of CPU for executing a host program) in order to control the GPU and manage data copy operations; remaining CPU cores are in an idle state while the GPU performs the application-specific tasks. This leads to large amounts of wasted CPU resources. In the CPU-GPU hybrid computing, CPU’s idle state can be reduced by using workload distribution into both CPU and GPU. Therefore, CPU-GPU hybrid computing can not only overcome the memory limitation of GPU, but also improve the performance for the large scale physics such as Lattice QCD.

We first parallelize the lattice QCD computation with OpenCL, and measure the maximum lattice size where a GPU can execute. To parallelize the Lattice QCD efficiently, we should understand the characteristics of lattice QCD computation and CPU-GPU hybrid computing. Due to the data dependency of Lattice QCD, the coarse-grain parallelism can be exploited by using OpenCL, and a typical GPU can provide better performance than a typical CPU.

To find out the memory limitation of a single GPU, however, we conducted simple experiments with a CPU-GPU hybrid computing environment, where CPU was Intel® Core™ i5-2500(4 cores) and GPU was NVIDIA GeForce GTX 550(336 cores). The maximum lattice size with the GPU was 344 (i.e., 4-dimensional space 344 with X-, Y-, Z-, and T-coordinates). If we increase the lattice size to 364, the large-scale problem cannot be executed.

Note that, the CPU can execute the large- scale problem, since the CPU can use the large size host memory, regardless of the data transfer overhead (i.e., host-to-device or device-to-host data transfer). For the 344 lattice size problem, however, the GPU can provide better performance by a factor 8 than the CPU. Therefore, to execute the QCD problem with lattice size larger than 344, we partition the parallelized workload into two parts, based on the relative performance of GPU over CPU as well as the memory limitation of GPU. Note that, to execute the large-scale problem on the CPU-GPU hybrid environment, we modified the ‘big lattice’ mode of the open-source QCDGPU[4] as follows.

In the original big-lattice mode, the workload is divided into multiple sub-lattices to execute it on multi-GPU environments. Fig. 2 shows two sub-lattices, and the lattice is divided based on X-axis. For the maximum performance with the CPU-GPU hybrid environment, however, we decompose the workload into two sub-lattices for each CPU and GPU. Note that, each sub-lattice requires the neighboring lattice points in order to calculate the required operations. Therefore, when we execute a 364 problem, we should divide the lattice into 10 (= 8 + 2 neighborings) × 36 (for CPU) and 30 (= 28 + 2 neighborings) × 36 (for GPU) based on X-coordinate.

![Fig. 1 The lattice table with the partitioned sub-lattices](image)

Finally, we assign the GPU-portion of workload to GPU by using a non-blocking command, and then assign the remaining
By reducing the idle time on either CPU or GPU, we can overlap the GPU execution maximally with the CPU execution.

IV. EXPERIMENTAL RESULTS

For evaluating the proposed approach for the CPU-GPU hybrid environment, we used Intel® Core™ i5-2500 (4 cores) and NVIDIA GeForce GTX 550 (336 cores). Also, the lattice QCD problem was parallelized to execute it on the CPU-GPU hybrid computing environment by using OpenCL. Note that, the maximum lattice size with the GPU-only execution was $34^4$, and we increased the lattice size up to $36^4$ (i.e., the maximum lattice size with the CPU-only execution).

To evaluate the proposed method, we measured the elapsed time of one ‘sweep’ in the lattice QCD problem of size $36^4$, and the performance (i.e., elapsed time) of the proposed approach (i.e., CPU-GPU hybrid computing with a 8:28 workload ratio) was compared with those of three possible scenarios (i.e., CPU-sequential, CPU-only, and CPU-GPU hybrid computing with a 18:18 workload ratio). Fig. 2 shows the comparison of elapsed time. The elapsed times of CPU-sequential and CPU-only were 37.91 sec and 21.06 sec, respectively. Also, the speedup of CPU-only was 1.8. Note that, the CPU-sequential and CPU-only scenarios can execute the lattice QCD problem of $36^4$ lattice size, since the CPU can use the large size host memory. In contrast, the elapsed time of CPU-GPU hybrid computing with a 18:18 workload ratio was 11.25 sec, and the speedup was 3.37. That is, CPU-GPU hybrid computing can overcome the memory limitation of GPU, and provide better performance than CPU-only. Finally, the elapsed time and speedup of the proposed approach were 5.40 sec and 7.02, respectively. Therefore, we confirmed that, the proposed approach can not only overcome the memory limitation of GPU, but also provide the maximum performance by overlapping the GPU execution maximally with the CPU execution.

![Fig. 2 Comparison of elapsed time](image)

V. CONCLUSION

We have proposed an efficient heterogeneous parallel processing approach to use both CPU and GPU in executing the lattice QCD code. The approach, which balances the workload between CPU and GPU by using OpenCL, decreases the total execution time for better performance. The experiment with a single sweep on lattice $36^4$ was impossible with the GPU-only execution due to the limited GPU memory. On the contrary, our heterogeneous parallel processing approach can provide a speedup of 7 over the sequential CPU execution.

ACKNOWLEDGMENT

This research was supported by BK21 Plus Program.

REFERENCES

[2] Cardoso, N. and Bicudo, P., Generating SU(Nc) pure gauge lattice QCD configurations on GPUs with CUDA, Computer Physics Communications 184 (2013) 509-518