ACCELERATED CODE GENERATOR FOR PROCESSING OCEAN COLOR REMOTE SENSING DATA ON GPU

Jae-Moo Heo¹, Gangwon Jo², Hee-Jeong Han¹, Hyun Yang¹

¹Korea Ocean Satellite Center, Korea Institute of Ocean Science and Technology, Pusan, South Korea
²Seoul National University and ManyCoreSoft Co., Ltd., Seoul, South Korea

ABSTRACT
As the satellite data gradually increases, the processing time of the algorithm for remote sensing also increases. It is now essential to make efforts to improve the processing performance in various accelerators such as Graphics Processing Unit (GPU). However, it is very difficult to use programming models that make programs to run on accelerators. Especially for scientists, easy programming and performance are often more important than providing many optimization functions and directives. To meet these requirements, we developed the accelerated code generator based on Open Computing Language (OpenCL) for processing ocean color remote sensing data. We easily applied our generator to atmospheric correction program for GOCI data as a case study and the program’s performance achieved about 29.2x that is as good as the hand-written OpenCL program. We look forward to many scientists that want to find a tool mentioned above taking advantage of our generator.

Index Terms— GOCI-II, GPU, Ocean color data processing, OpenCL, Source-to-source translator

1. INTRODUCTION
Amount of ocean color satellite data is rapidly increased in recent years. This increase is due to the development of multiple satellite sensors and improvement of sensor performance. There are many studies using improved computation devices for high-resolution and multi-sensor data processing [1] and it is now commonplace to improve the processing performance of the algorithm for remote sensing through an accelerator such as GPU [2], [3]. Korea Ocean Satellite Center (KOSC) has been operating Geostationary Ocean Color Imager (GOCI) since 2010 and is also developing ground system for GOCI-II launched in 2019. KOSC with many universities and research institutes is being developed 26 kinds of ocean, air, and land products. In addition, amount of data is increased by about 24 times due to 1.5 times the number of observation bands, 4 times the spatial resolution, and full-disk observation, etc. Therefore, we completed the critical design review of GOCI-II ground system in July 2017 and decided to process algorithms on the Graphics Processing Unit (GPU) in order to process large amounts of data. Nowadays, we can take advantage of computation devices of various architectures: Many Integrated Core (i.e., Intel Xeon Phi), GPU, and Filed Programmable Gate Array (FPGA). These accelerators are constantly improving and changing. And, parallel programming models that are compatible with these various accelerators are being developed at the same time. However, algorithm source codes have been generally improved for several years or several decades and it is not easy to develop a large amount of these source codes into the existing parallel programming models. OpenCL [4] and Compute Unified Device Architecture (CUDA) [5] are extensions to existing programming languages (i.e., C/C++ and Fortran) and the most popular parallel programming models among programmers. These models are the best option if performance improvement is the most important. Conversely, these are difficult to use because it is a low-level programming model and we should also have good knowledge of hardware architecture. Open Multi-Processing (OpenMP) [6] and OpenACC [7] are directives provided additional information to compiler for parallelization and have relatively recently been designated for a various computing devices and are currently undergoing vigorous improvements. Both programming models are aimed at portability and easy programming. However, these are relatively lower performance improvement than the OpenCL or CUDA [8] and we have to be familiar with the various directives in order to achieve good performance. As a result, these models are still difficult for general scientists to use.

We concentrated on producing a good performance as well as parallelizing scientific algorithms easily. In addition, these generated programs must be available in future accelerators. OpenCL is not only a portable programming model, it also has a good performance. Therefore, we decided to base on OpenCL 1.2 and support only C programming [9]. Finally, we developed the accelerated code generator for processing ocean color remote sensing data and named it as OCAccel. It is implemented based on
annotations for easier programming rather than compiler directives or C/C++ extensions. OCAccel generates automatically parallel code from serial code. It internally translates the given serial code into an OpenCL code. Therefore, scientists do not need to use a complicated parallel programming model because it is automatically done during compilation. In this paper, the atmospheric correction algorithm for GOCI data was used for case study and the presented experiment has been based on NVIDIA GTX Titan X Pascal GPU. Consequently, most of parallel sections showed good performance and it is much easier to programming.

2. FRAMEWORK

OCAccel has a source-to-source translator that automatically offloads parallelizable loops (i.e., loops with no loop-carried dependences) in a C/C++ program to an accelerator (e.g., a GPU). OCAccel internally translates the given program into an OpenCL program, and then executes it on top of the vendor-provided OpenCL driver. OpenCL is an open standard for heterogeneous computing and many industry-leading hardware vendors including AMD, ARM, IBM, Intel, NVIDIA, and Xilinx support OpenCL for their processors and accelerators. Thus, OCAccel also provides portability between different accelerators.

Fig. 1 shows an example code that is given to OCAccel. A for-loop (or perfectly nested for-loops), that is annotated by a label whose name starts with parallelize, is offloaded to an accelerator by OCAccel. We call this a kernel loop. Multiple work-items (i.e., threads running on an accelerator) execute different loop iterations simultaneously [10]. All other parts of the program are executed on a CPU. The body of a kernel loop contains only a single function call. It may pass arrays, memory blocks allocated by malloc() or new, and scalar values as arguments to the function. The target function is defined with the keyword KERNEL. In addition, every pointer parameter passed from the kernel loop is classified into one of INPUT, OUTPUT, and INOUT (Line 1 of Fig. 1). They indicate that the given array or memory block is read-only, write-only, or read-write. A KERNEL function should be written in C99, even if the other parts of the program are written in C++. It may call other KERNEL functions, C mathematical functions (e.g., sin, cos, and so on), and built-in functions defined by OCAccel. The program translated by OCAccel calls only a small number of designed runtime functions, and then internally calls vendor-provided OpenCL libraries with the runtime library. All of the user annotations rid OCAccel of the need for loop dependence analysis and pointer analysis that are impractical for real-world applications. Note that if we erase four keywords (i.e., KERNEL, INPUT, OUTPUT, and INOUT) using the C preprocessor, the program becomes a valid C/C++ source code and can be executed solely on a CPU sequentially. This helps the scientists to easily verify the behavior of the program before parallelizing it.

We implement the source-to-source translator by modifying clang 3.7.1 and writing a python script. OCAccel consists of an extract program, a patch program, and a runtime library. We split the extract program and the patch program to make it easier for users to understand the program translated by OCAccel and to handle complex programs with multiple source codes. Fig. 2 shows how a program is executed on a heterogeneous system using OCAccel. The source-to-source translator of OCAccel replaces every kernel loop in a given source code with an OpenCL host code that copies all INPUT/INOUT arrays and memory blocks to the accelerator, executes all iterations of the kernel loop on the accelerator, and copies all OUTPUT/INOUT arrays and memory blocks from the accelerator. In addition, the source-to-source translator aggregates all KERNEL functions into an OpenCL
3. CASE STUDY AND EVALUATION

In ocean color remote sensing, atmospheric correction program must be necessarily performed before processing other ocean color algorithms. Although the atmospheric correction algorithms are not exactly identical due to the differences of satellite sensor performance or trajectory, the overall procedure is similar. For this reason, we evaluate OCAccel with the atmospheric correction algorithm for GOCI data that has a theoretical basis for standard atmospheric correction methods applied to NASA’s polar orbiting satellites, Sea-Viewing Wide Field-of-view Sensor (SeaWiFS) and Moderate-Resolution Imaging Spectroradiometer (MODIS) [11] and it is included in GOCI Data Processing System (GDPS) [12] 2.0 version. This algorithm program is divided into 4 parallel sections. Section 1 generates and store information corresponding to each slot of spectral bands (for GOCI only) and section 2 calculates the geometric angles of the sun and the satellite (depending on satellites). And then section 3 generates Rayleigh corrected reflectance. Finally, section 4 generates remote sensing reflectance [13], [14].

Case study was on the system that has dual-socket Intel Xeon E5-2630 v3 CPUs (total 16-Cores) and an NVIDIA Titan X Pascal GPU. We compared the OCAccel program with the serial program, the OpenMP program (multi-core CPU version), and the hand-written OpenCL program (GPU version). In this experiment, we used GOCI L1B data consisting of bands and each band has 5567 x 5685 pixels. Each experiment took an average time of 10 runs and the execution time of each run was almost the same within several tens of milliseconds. During the case study, we confirmed that there are some limitations for a good performance. ACACcel does not provide detailed information about the memory copy to the GPU. Therefore, if the same kernel is repeatedly executed continuously, meaningless copy transfer may occur. This issue is also present in OpenACC [15]. Nevertheless, it is difficult to solve with an abstracted directive or an annotation base. Thus, we need to configure the kernel loop in the outermost for-loop so that each kernel can be executed only once.

Fig. 3 shows the execution time of each program. As you can see, the OCAccel program has almost the same performance as the hand-written OpenCL program and it also showed much better performance than the multi-core CPU version OpenMP program. We achieved the OCAccel’s total performance improvements of 29.2x and 2.3x compared to a serial program and OpenMP program running on a multi-core CPU, respectively. For the OpenMP program performs best with 32 threads. The performance was evenly measured 13-14x since the number of cores of the multi-core CPU is physically 16. At this point, we founded that the GPU-based method is more efficient than the multi-core CPU in the atmospheric correction algorithm.

Table 1. Internal execution time of each section of the OCAccel program

<table>
<thead>
<tr>
<th>Section</th>
<th>CPU host</th>
<th>Memory copy</th>
<th>GPU kernel</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.42</td>
<td>0.61</td>
<td>0.68</td>
<td>1.71</td>
</tr>
<tr>
<td>2</td>
<td>0.28</td>
<td>0.85</td>
<td>0.75</td>
<td>1.88</td>
</tr>
<tr>
<td>3</td>
<td>0.30</td>
<td>1.23</td>
<td>1.34</td>
<td>2.87</td>
</tr>
<tr>
<td>4</td>
<td>1.55</td>
<td>3.51</td>
<td>15.43</td>
<td>20.49</td>
</tr>
</tbody>
</table>

Table 1 summarizes the internal execution time of each section of the OCAccel program. It measures the time inside of each for-loop that is annotated by a label `parallelize`. The time of CPU host is included before calling the “clEnqueueNDRangeKernel” OpenCL API function [16]. We also used the “clGetEventProfilingInfo” function to measure the time that spent in the command queue and defined as the time of GPU kernel. The remaining time is defined as the memory copy between CPU and GPU. This time, of course, includes the time it takes to call OpenCL API functions. Because it takes 0.5 seconds to call the OpenCL function even if no memory copy occurs, there is little overhead due to memory copy. The pure computational time within the GPU kernel is 212x, 202x, 90x, and 58x faster for each section compared to the serial program, respectively. The reason for the relatively low performance of section 3 and 4 is that all of code branches are internally performed in the GPU kernel [17]. Therefore, the smallest performance improvement is shown in section 4 since the amount of computation deviates greatly according
to the branch statement and the computation amount are basically large.

4. CONCLUSION

We have presented the accelerated code generator for processing ocean color satellite data and call it as OCAccel. It is based on OpenCL, programming model considering portability, scalability, easy programming, and performance. OCAccel internally translates a serial C/C++ program into a program that can operate in accelerators through a few lines of simple annotations. We easily applied OCAccel to atmospheric correction program for GOCI data and the experimental result showed the performance as good as the hand-written OpenCL program. In the future, we will apply it to the ocean color algorithms that have been developed for GOCI-II ground system. Then we will develop the common parts of remote sensing algorithms as OCAccel runtime API functions and release it.

ACKNOWLEDGMENT

This research was supported by the ‘Development of the integrated data processing system for GOCI-II’ funded by the Ministry of Ocean and Fisheries, Korea. This research was also supported by the ‘Operation in Korea Ocean Satellite Center’ funded by the Korea Institute of Ocean Science and Technology (KIOST).

5. REFERENCES


