Exploring the Suitability of Remote GPGPU Virtualization for the OpenACC Programming Model Using rCUDA

Adrián Castelló, Rafael Mayo, Enrique S. Quintana-Ortí
Universitat Jaume I de Castelló
Castelló de la Plana, Spain
{*adcastel,mayo,quintana*}*@uji.es*

Antonio J. Peña, Pavan Balaji
Argonne National Laboratory
Argonne, IL 60439
{*apenya,balaji*}*@anl.gov*

***Abstract*—OpenACC is an application programming interface (API) that aims to unleash the power of heterogeneous systems composed of CPUs and accelerators such as graphic processing units (GPUs) or Intel Xeon Phi coprocessors. This directive-based programming model is intended to enable developers to accelerate their application's execution with much less effort. Coprocessors offer significant computing power but in many cases these devices remain largely underused because not all parts of applications match the accelerator architecture. Remote accelerator virtualization frameworks introduce a means to address this problem. In particular, the remote CUDA virtualization middleware rCUDA provides transparent remote access to any GPU installed in a cluster. Combining these two technologies, OpenACC and rCUDA, in a single scenario is naturally appealing. In this work we explore how the different OpenACC directives behave on top of a remote GPGPU virtualization technology in two different hardware configurations. Our experimental evaluation reveals favorable performance results when the two technologies are combined, showing low overhead and similar scaling factors when executing OpenACC-enabled directives.**

***Keywords*-GPUs; OpenACC; remote virtualization; rCUDA**

I. Introduction

The use of coprocessor accelerators has grown continuously in the past several years. The generalized adoption of coprocessors to accelerate general-purpose pieces of code started with the emergence of the CUDA ecosystem for NVIDIA GPUs [1]. Following CUDA, OpenCL [2] emerged as an attempt to offer a cross-vendor solution. The third-generation programming model for accelerators, based on compiler directives, started with OpenACC [3]. This standard defines a collection of directives to facilitate platform-independent accelerator usage.

The traditional approach for accelerator-enabled clusters has been to furnish every compute node with one or more of these devices. In many scenarios, however, these configurations yield a low utilization of the computational resources available in the hardware accelerators, because of mismatches between the application's type of parallelism and the coprocessor architecture. Remote virtualization was proposed as a technique to address this problem. Among the virtualization frameworks, the most prominent is rCUDA [4], [5], which enables cluster configurations with fewer GPUs than nodes. Compared with other CUDA and OpenCL virtualization frameworks (e.g. DS-CUDA [6], vCUDA [7], and VOCL [8]), rCUDA is a mature production-ready framework that offers support for the latest CUDA revisions and provides wide coverage of current GPGPU APIs.

The combination of both technologies—namely, execution of OpenACC-enabled applications on remote accelerators—is obviously appealing. In this paper we explore the open question of whether remote virtualization is suitable for applications accelerated via OpenACC directives.

In summary, the contributions of this paper are as follows: (1) we explore the challenges of integrating the directive-based OpenACC programming model for accelerators and the rCUDA remote GPU virtualization framework, (2) we analyze the performance of representative OpenACC directives when operating over a remote accelerator, and (3) we evaluate different OpenACC compiler options to find optimized configurations for remote accelerations.

The rest of the paper is structured as follows. Section II provides background information on the technologies explored in this paper. Section III discusses the integration of the OpenACC and rCUDA technologies. Section IV introduces our testbed in terms of hardware and test code. Section V reviews our experimental evaluation, and Section VI closes the paper with a brief summary.

II. Background

In this section we review the two technologies targeted in this paper: OpenACC and rCUDA.

A. The OpenACC Programming Standard

OpenACC [3] is an open programming standard developed by PGI, Cray, and NVIDIA that enables programmers to easily leverage heterogeneous CPU plus coprocessor systems from their C, C++, or Fortran codes.

OpenACC comprises a collection of compiler directives that the programmer employs to identify the pieces of code to be accelerated by a coprocessor. Although the performance attained by current compilers still shows some gap with respect to that obtained when directly leveraging

GPGPU APIs, the development productivity is clearly much higher [9], [10]. We have selected the widely used and mature PGI OpenACC compiler (version 14.9) for our study.

B. The rCUDA Framework

The rCUDAframework [4], [5] is middleware that enables seamless access to any CUDA-compatible device present in a cluster from all compute nodes. It is structured following a client–server distributed architecture. The GPUs can be shared between nodes, and a single node can use all these graphic accelerators as if they were local. These properties aim to attain higher accelerator utilization rates in the overall system while simultaneously reducing resource, space, and energy requirements [11], [12]. The rCUDA client exposes the same interface as the regular NVIDIA CUDA 6.5 release [1] does, including the runtime and driver APIs. Hence, applications are unaware that they are executing on top of a virtualization layer. We base our study on the current rCUDA public release (version 5.0).

III. Integrating OpenACC and rCUDA

In this section we discuss the OpenACC and rCUDA integration challenges.

A. Compilation Requirements

The integration of these two technologies is possible because current OpenACC compilers generate calls to public GPGPU APIs. Since we target a distributed environment where the GPGPU clients do not necessarily feature actual GPUs, among the different options the PGI compiler offers to generate separate GPU modules, we choose to generate PTX files (`--keepptx`) —that is, low-level source code GPU files—that are compiled just in time on the GPGPU server and optimized for the specific target GPU architecture.

The PGI compiler uses the following module management functions from the low-level CUDA driver API, which the current rCUDA release does not support.

`cuModuleLoadData`: This call loads an appropriate module for the target GPU architecture, comprising a set of GPU kernel functions, and makes it available for subsequent kernel executions.

`cuModuleGetFunction`: This function searches within the module loaded by the previous call for the code implementing a given kernel name and makes it available for subsequent use.

We have implemented both functions in rCUDA and carefully tuned them for the distributed environment. Our `cuModuleLoadData` implementation allows the client to send all the GPU modules from the module repository of the executed application to the GPGPU server upon the first intercepted call to this function, and this mechanism is executed only once. These images are stored in contiguous memory addresses in the server in order to attain efficient `cuModuleGetFunction` call responses.

B. PGI and rCUDA Data Transfers

The PGI compiler performs data transfers between the host and the device exploiting a double buffer mechanism. A similar mechanism is implemented by rCUDA when requested to perform transfers from pageable memory, but it performs a direct transfer from the pinned buffers otherwise, avoiding the augmented latency and memory stress of an additional pipeline stage [5]. We set the internal rCUDA buffers to their optimal sizes (FDR: 1 MB; QDR: 2 MB).

IV. Experimental setup

In this section we introduce our testbed. First we present the systems and describe our test cases.

A. Hardware Systems

We have selected two systems with different GPU and network transfer rates, as well as different computational power, in order to avoid biasing our study for a particular hardware configuration.

System `A` is composed of two compute nodes, each equipped with two Intel Xeon E5520 quadcore processors running at 2.27 GHz and 24 GB of DDR3-1866 RAM. One of the nodes is connected to an NVIDIA C2050 GPU. Internode communications are accomplished via an InfiniBand QDR fabric.

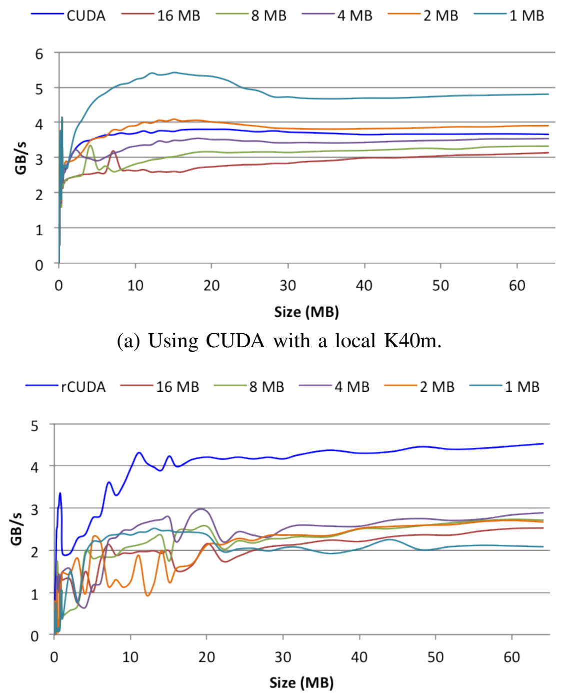
System `B` consists of two compute nodes, each equipped with two Intel E5-2687W v2 8-core processors running at 3.40 GHz and 64 GB of DDR3-1866 RAM. The GPGPU server is endowed with an NVIDIA Tesla K40m GPU. Both nodes are connected via an InfiniBand FDR interconnect.

B. Test Cases

We have developed a set of microbenchmarks to measure the execution time of the most frequently used OpenACC directives. Each directive is executed 10 times, and the dataset size ranges from 1 MB to 64 MB in data transfer directives and to 128 MB in execution directives.

The common pattern in an accelerated application involves at least one data movement from host memory to device memory, before the code is executed in the coprocessor, and one data transfer from device memory to host memory, after the execution in the accelerator is completed. In OpenACC, the data movement is controlled by different `copy` directives. We have implemented two simple copy microbenchmarks to evaluate the performance of these transfers. The "copyin" test shows the bandwidth from host memory to device memory, and the "copyout" test evaluates it from device memory to host memory.

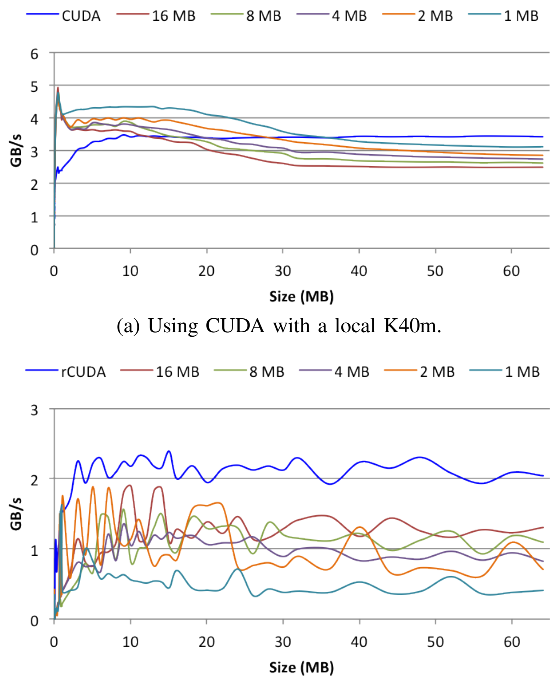
In OpenACC, the programmer can use the directives `kernels` or `parallel` to instruct the OpenACC compiler that the next structured code block has to be executed in the coprocessor. To study these directives, we have developed microbenchmarks that encode a simple scaling vector kernel using both options. The "nested" test measures the



(a) Using CUDA with a local K40m.

(b) Using rCUDA with a remote K40m.

Figure 1: Performance of `copyin` directive on system `B`.



(a) Using CUDA with a local K40m.

(b) Using rCUDA with a remote K40m.

Figure 2: Performance of the `copyout` directive for each buffer size on system `B`.

performance when more than one for loop is present by implementing a matrix-matrix product.

V. Experimental evaluation

1) Data Transfer Directives: The size of each PGI intermediate buffer is 16 MB but it can be configured by setting the `PGI_ACC_BUFFERSIZE` environment variable. Figures 1 and 2 report the performance attained by the `copy` directives in system `B`, for a range of buffer sizes.

Figure 1a illustrates the performance attained by the `copyin` directive with locally executed clauses. The throughput with respect to bare CUDA calls is as much as 1 GB/s lower for small data payloads using the default buffer size. This gap is reduced as the payload size increases and also if the buffer size is reduced. Using 1 MB buffers,we obtain the highest performance.The combination of pinned memory and asynchronous transfers enables this OpenACC directive to perform better than its homologous CUDA counterpart. For rCUDA (Figure 1b), we note a higher performance with bare CUDA calls than with native CUDA because the rCUDA implementation leverages a mechanism similar to that of the PGI compiler based on internal pinned buffers. The performance drop of the OpenACC directives with respect to those executed on a local accelerator is around 1 GB/s. In this case, the optimal buffer size starting at 8 MB transfers is 4 MB.

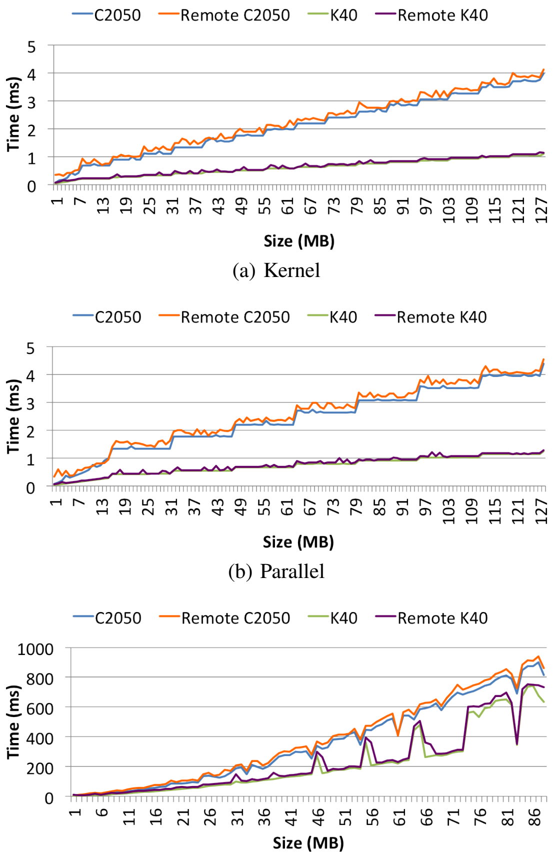
Figure 2 shows the performance obtained by the `copyout` directive. With local CUDA (Figure 2a), as in the `copyin` case, the best choice for the buffer size is 1 MB. When rCUDA is used (Figure 2b), a more irregular behavior appears, caused by the polling mechanism used to determine the finalization of device-to-host transfers [4], [5]. In this case, larger buffers benefit sufficiently large data transfers (starting at 8 MB). The performance difference with respect to locally executed directives is in this case more than 1.5 GB/s.

The main conclusion from this evaluation is that the throughput difference for `copy` directives is considerable when they are executed on top of virtualized remote accelerators instead of their local counterparts. Tuning the PGI buffer size for the particular system and underlying GPGPU implementation (CUDA vs. rCUDA) yields non-negligible benefits in terms of data transfer performance.

2) Computational Directives: The performance attained by the `kernels` (Figure 3a) and `parallel` (Figure 3b) directives is similar. The differences are caused by different implementations of the GPU kernels generated by the PGI compiler. On the other hand, Figure 3c reports the execution time when loops are nested in order to implement a matrix-matrix product. In both scenarios, the use of a remote GPU does not lead to a significant performance difference with respect to local acceleration.

VI. Conclusions

We analyzed the suitability of remote accelerator virtualization technologies for the OpenACC programming model.



(a) Kernel

(b) Parallel

(c) Nested

Figure 3: Performance of loop directives.

Our evaluation of the performance of the major OpenACC directives indicates that remote accelerations may add a significant overhead with respect to their local counterpart only if `copy` directives dominate the overall execution time, since the rest of the directives do not exhibit a large performance penalty. Our main conclusion from this study is that in spite of the OpenACC programming model being designed to leverage fine-grained parallelism, remote accelerator virtualization technologies may still provide considerable benefits for production scenarios.

Acknowledgments

Researchers at UJI were supported by MINECO, by FEDER funds under Grant TIN2011-23283, and by Universitat Jaume I (Grant P11B2013-21). This work was partially supported by the U.S. Dept. of Energy, Office of Science, Office of Advanced Scientific Computing Research (SC-21), under contract DE-AC02-06CH11357. We gratefully acknowledge the computing resources provided and operated by the Joint Laboratory for System Evaluation (JLSE) at Argonne National Laboratory. We are also grateful for the generous support provided by Mellanox Technologies. The initial version of rCUDA was jointly developed by Universitat Politècnica de València (UPV) and Universitat Jaume I de Castellón (UJI) until year 2010. This initial development was later split into two branches. Part of the UPV version was used in this paper. The development of the UPV branch was supported by Generalitat Valenciana under Grants PROMETEO 2008/060 and Prometeo II 2013/009.

References

[1] NVIDIA, *CUDA API Reference, Version 6.5*, 2014.

[2] A. Munshi, Ed., *The OpenCL Specification Version 2.0*. Khronos OpenCL Working Group, Mar. 2014.

[3] "OpenACC directives for accelerators," http://www.openacc-standard.org, 2015.

[4] A. J. Peña, "Virtualization of accelerators in high performance clusters," Ph.D. dissertation, Universitat Jaume I, Castellón, Spain, Jan. 2013.

[5] A. J. Peña, C. Reaño, F. Silla, R. Mayo, E. S. Quintana-Ortí, and J. Duato, "A complete and efficient CUDA-sharing solution for HPC clusters," *Parallel Computing*, vol. 40, no. 10, pp. 574–588, 2014.

[6] A. Kawai, K. Yasuoka, K. Yoshikawa, and T. Narumi, "Distributed-shared CUDA: Virtualization of large-scale GPU systems for programmability and reliability," in *The Fourth International Conference on Future Computational Technologies and Applications*, 2012, pp. 7–12.

[7] L. Shi, H. Chen, J. Sun, and K. Li, "vCUDA: GPU-accelerated high-performance computing in virtual machines," *IEEE Transactions on Computers*, vol. 61, no. 6, 2012.

[8] S. Xiao, P. Balaji, Q. Zhu, R. Thakur, S. Coghlan, H. Lin, G. Wen, J. Hong, and W. Feng, "VOCL: An optimized environment for transparent virtualization of graphics processing units," in *Innovative Parallel Computing*. IEEE, 2012.

[9] T. Hoshino, N. Maruyama, S. Matsuoka, and R. Takaki, "CUDA vs OpenACC: Performance case studies with kernel benchmarks and a memory-bound CFD application," in *13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, May 2013, pp. 136–143.

[10] S. Lee and J. S. Vetter, "Early evaluation of directive-based GPU programming models for productive exascale computing," in *International Conference on High Performance Computing, Networking, Storage and Analysis (SC)*, 2012.

[11] A. Castelló, J. Duato, R. Mayo, A. J. Peña, E. S. Quintana-Ortí, V. Roca, and F. Silla, "On the use of remote GPUs and low-power processors for the acceleration of scientific applications," in *The Fourth International Conference on Smart Grids, Green Communications and IT Energy-aware Technologies (ENERGY)*, April 2014, pp. 57–62.

[12] S. Iserte, A. Castelló, R. Mayo, E. S. Quintana-Ortí, C. Reaño, J. Prades, F. Silla, and J. Duato, "SLURM support for remote GPU virtualization: Implementation and performance study," in *International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, Oct. 2014.