POSTER: Towards OmpSs-2 and OpenACC Interoperation

Orestis Korakitis
Barcelona Supercomputing Center (BSC)
orestis.korakitis@bsc.es

Simon Garcia De Gonzalo
Barcelona Supercomputing Center (BSC)
simon.garcia@bsc.es

Nicolas Guidotti
INESC-ID, Instituto Superior Técnico, University of Lisbon
nicolas.guidotti@tecnico.ulisboa.pt

João Pedro Barreto
INESC-ID, Instituto Superior Técnico, University of Lisbon
joao.barreto@tecnico.ulisboa.pt

José C. Monteiro
INESC-ID, Instituto Superior Técnico, University of Lisbon
jcm@inesc-id.pt

Antonio J. Peña
Barcelona Supercomputing Center (BSC)
antonio.pena@bsc.es

Abstract

The increasing demand in HPC to utilize accelerators has motivated the development of pragma-based directives to target these devices. OmpSs-2 and OpenACC are both directive-based solutions that allow application programmers to utilize accelerators. The two leverage distinct types of parallelism: task parallelism and data parallelism, respectively. Non-trivial scientific applications can benefit from both types of available parallelism. However, the combination of pragma-based models is difficult to coordinate, as both assume full control and are unaware of each other at runtime. We propose an interoperation mechanism to enable novel composability across pragma-based programming models. We study and propose a clear separation of duties and implement our approach by augmenting the OmpSs-2 programming model, compiler and runtime to support OmpSs-2 + OpenACC programming.

Keywords Programming Productivity, Data-flow Paradigm, Runtime Scheduling, Code Transformation, Parallelism, GPU

1 Introduction

The OpenACC Standard [6] was introduced in 2011 to address many of the above-mentioned limitations and facilitate the development of heterogeneous applications. OpenACC is a directive-based programming model that allows compiling plain C/C++ or Fortran code regions, annotated by the user, interchangeably targeting different accelerator devices. However, in the scope of OpenACC, the main (*host*) part of the program is running sequentially, while utilization of multiple devices, as well as asynchronous execution of device kernels, comes at the cost of additional programming complexity.

Meanwhile, task-based programming models are receiving increasing attention for being especially tailored to harness heterogeneous architectures. Since these models are asynchronous by nature, they inherently overcome the above-mentioned limitations of OpenACC. One such example is OmpSs-2 [2], a task-parallel programming model, that aims to ease the development of parallel programs. Using OmpSs-2, programmers annotate their programs as a series of tasks and indicate data dependencies among those. Then, the runtime system can discover and enable potential parallelism among the tasks execution. Additionally, it is designed with heterogeneity in mind, as it can support running tasks based on different programming models, for instance, CUDA. However, as the name suggests, OmpSs-2 + CUDA programs require kernels that will be offloaded to a device to be written using the CUDA programming model, requiring a substantial programming effort by developers, breaking the productivity benefits and performance portability across different architectures offered by pure pragma-based models.

Collaboration between two pragma-based programming environments is difficult to coordinate effectively, as both assume full control during application execution and are agnostic to each other. We propose a mechanism of inter-operations across both pragma-based programming models that enables programmers to compose OmpSs-2 tasks with OpenACC accelerator kernels. Our proposal ensures a clear separation of duties between the two models, and is implemented by augmenting the OmpSs-2 compiler and runtime to support hybrid OmpSs-2 + OpenACC programming.

2 Background and Motivation

Table 1 exposes limitations that different pragma-based models suffer from. Currently the most widely available and popular directive-based parallel programming models lack the ability to express and manage multi-device asynchronous execution effectively without a substantial effort from developers. For example, OpenACC forces the programmer to keep track

PPoPP '22, April 2–6, 2022, Seoul, Republic of Korea

ACM ISBN 978-1-4503-9204-4/22/04…$15.00
https://doi.org/10.1145/3503221.3508401

Progr. Model	Async. Execution	Pragma Only	Device Support	Re-target	Multi-Device
OpenMP	Manual	Yes	Yes	Yes	Manual
OpenACC	Manual	Yes	Yes	Yes	Manual
OmpSs-2	Auto	Yes	No	No	No
OmpSs-2 +CUDA	Auto	No	Yes	No	Auto
OmpSs-2 +OpenACC	Auto	Yes	Yes	Yes	Auto + Affinity

Table 1. Summary of programming models capabilities.

of all asynchronous queues and to be responsible of how code regions map to GPU stream/queues. OmpSs-2 has the ability to annotate and schedule asynchronous tasks at an application-wide scope, but it lacks the capabilities to offload computation to devices without using low-level languages. OpenMP's [7] limited scope and implicit barriers for asynchronous tasks, as well as the opaqueness of how offloaded tasks are mapped to asynchronous GPU streams/queues, preclude developers from effectively scheduling device tasks at application wide scope. These models also lack the capacity to inter-operate with each other beyond a superficial level, increasing the programming complexity if more than one model is needed or desired. The goal of this work is to provide an interoperation mechanism for pragma-based programming languages and at the same time lower the programming effort needed to target multi-device systems.

3 OmpSs-2 + OpenACC Interoperation

We propose a hierarchy of programming models where OmpSs-2 is the "dominant" model, responsible for orchestrating the program execution. Then, OpenACC, as a supportive model, can be used to accelerate appropriately-annotated, computation-heavy, data-parallel regions internally within the tasks. Our proposal delineates a clear separation of responsibilities for each model.

Device Kernel Execution: The task of targeting data-parallel kernels to accelerator devices falls on the OpenACC programming model. This is accomplished by the programmers' use of a convenient subset of the OpenACC programming standard, limited to *compute* directives. These regions are processed by an OpenACC compiler.

Data Movement: The responsibility for managing and transferring data across the host and different number of devices follows the current general OmpSs-2 implementation decision and falls on the Unified Memory [3] abstraction layer for NVIDIA devices. This maintains OmpSs-2's requirement for a single address space and simplifies programmability for application developers, since manual transfer of data is not needed. This choice features important implications for the work scheduling heuristics.

Host and Device Work Scheduling: Work scheduling for both host and device is managed by the OmpSs-2 runtime. In terms of programmability, application developers only need to mark which tasks are meant for accelerator devices with a new clause. Work scheduling of OpenACC kernels is performed through source–to–source transformation by the OmpSs-2 compiler and by the OmpSs-2 runtime by managing OpenACC device queues.

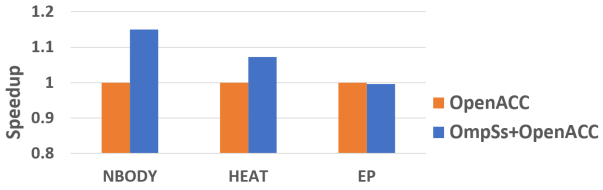

Figure 1. Single-GPU benchmarks.

4 Evaluation

The evaluation was performed on an IBM AC922 cluster based on POWER9 processors and NVIDIA Volta GPUs [5]. OmpSs-2 official release v2.3 has been extended with the work proposed in this paper and used for compilation. NVIDIA HPC SDK 20.11 (with CUDA 10.2) was used for providing the OpenACC support and native compiler. We present preliminary results on a set of single-device benchmarks that comprises: (1) a 2D heat equation solver using an iterative Gauss-Seidel method in blocks (HEAT) [1]; (2) an NBody benchmark simulating dynamic particle systems (NBODY) [1]; and (3) a SPEC-Accel [4] OpenACC adaptation of the Embarrassingly Parallel benchmark from the NAS benchmark suite (EP). Preliminary benchmarks' results, shown in Figure 1, indicate that NBODY and HEAT benefit from the automatically enabled asynchronous execution, showing a speedup of 15% and 7% respectively, with a single directive addition in the code. For EP, where each iteration comprises multiple and increasingly complex kernels inside the OpenACC task region, resulting in a much coarser-grain approach, there is no performance benefit; however, it shows that our proposal does not pose a negative impact on a code that is not inherently suitable for adaptation.

References

[1] David Álvarez, Kevin Sala, Marcos Maroñas, Aleix Roca, and Vincenç Beltran. 2021. Advanced synchronization techniques for task-based runtime systems. In *PPoPP'21*. 334–347.

[2] Barcelona Supercomputing Center. [n. d.]. OmpSs-2 Specification. ([n. d.]). https://pm.bsc.es/ftp/ompss-2/doc/spec/ Accessed: 2021-03-30.

[3] Mark Harris. [n. d.]. Unified Memory in CUDA 6. ([n. d.]). https://developer.nvidia.com/blog/unified-memory-in-cuda-6

[4] Guido Juckeland, William Brantley, Sunita Chandrasekaran, Barbara Chapman, Shuai Che, Mathew Colgrove, Huiyu Feng, Alexander Grund, Robert Henschel, Wen-Mei W Hwu, et al. 2014. SPEC ACCEL: A standard application suite for measuring hardware accelerator performance. In *PMBS'14*. Springer, 46–67.

[5] NVIDIA Corporation. [n. d.]. V100 Datasheet. ([n. d.]). https://images.nvidia.com/content/technologies/volta/pdf/volta-v100-datasheet-update-us-1165301-r5.pdf

[6] OpenACC Organization. 2019. The OpenACC Application Programming Interface. Version 3.0. (November 2019). Available at: https://www.openacc.org/specification/ (2021-03-28).

[7] OpenMP Architecture Review Board. 2020. OpenMP Application Programming Interface. Version 5.1. (November 2020). Available at: https://www.openmp.org/ (2021-03-28).