J. Parallel Distrib. Comput. 74 (2014) 3228–3239

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

A uniform approach for programming distributed heterogeneous computing systems

Ivan Grasso[a,b,*], Simone Pellegrini[a], Biagio Cosenza[a], Thomas Fahringer[a]

[a] *Institute of Computer Science, University of Innsbruck, Austria*
[b] *Barcelona Supercomputing Center, Barcelona, Spain*

HIGHLIGHTS

- libWater programming model, which extends OpenCL with a simplified interface.
- A lightweight distributed runtime system based on asynchronous command execution.
- A powerful representation that collects and arranges dependencies between commands.
- Dynamic Collective Replacement and Device-Host-Device Copy Removal optimizations.
- A study of the performance of the library on three compute clusters.

ARTICLE INFO

Article history:
Received 15 July 2013
Received in revised form
15 April 2014
Accepted 14 August 2014
Available online 26 August 2014

Keywords:
OpenCL
MPI
Distributed computing
Heterogeneous computing
Programming model
Runtime system

ABSTRACT

Large-scale compute clusters of heterogeneous nodes equipped with multi-core CPUs and GPUs are getting increasingly popular in the scientific community. However, such systems require a combination of different programming paradigms making application development very challenging.

In this article we introduce libWater, a library-based extension of the OpenCL programming model that simplifies the development of heterogeneous distributed applications. libWater consists of a simple interface, which is a transparent abstraction of the underlying distributed architecture, offering advanced features such as inter-context and inter-node device synchronization. It provides a runtime system which tracks dependency information enforced by event synchronization to dynamically build a DAG of commands, on which we automatically apply two optimizations: collective communication pattern detection and device-host-device copy removal.

We assess libWater's performance in three compute clusters available from the Vienna Scientific Cluster, the Barcelona Supercomputing Center and the University of Innsbruck, demonstrating improved performance and scaling with different test applications and configurations.

1. Introduction

Ease of programming and best performance exploitation are two conflicting goals while designing programming models and abstractions for high performance computing (HPC). For instance, when programming a compute cluster, better performance can be obtained directly using low level and error prone communication layers like MPI [27]. Alternatively, high level models like domain specific languages and frameworks can be employed to simplify the programmability and portability of the code. This simplification however brings also a loss of performance due to the level of abstraction that is too far away from the underlying hardware.

The recent arise of multi- and many-core CPUs, next to special purpose hardware and accelerators such as GPUs, made this trade-off even more challenging. In fact, heterogeneous architectures require an intricate and complex mix of programming models such as CUDA, OpenMP and pthreads, in order to handle the diversity of execution environments and programming models.

The Open Computing Language (OpenCL—[21]) is a partial solution to the problem. It introduces an open standard for general-purpose parallel programming of heterogeneous systems, which has been implemented by many vendors such as Adapteva, Altera, AMD, ARM, Intel, Imagination Technologies, NVIDIA, Qualcomm, Vivante and Xilinx. An OpenCL program comprises a *host* program

[*] Corresponding author at: Institute of Computer Science, University of Innsbruck, Austria.
E-mail address: grasso@dps.uibk.ac.at (I. Grasso).

http://dx.doi.org/10.1016/j.jpdc.2014.08.002
0743-7315/© 2014 The Authors. Published by Elsevier Inc. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/3.0/).

and a set of *kernels* intended to run on a compute device. It also includes a language for kernel programming, and an API for transferring data between host and device memory and for executing kernels. Therefore, OpenCL is a big leap forward in order to assure portability between different hardware, potentially replacing standards like OpenMP and CUDA, but it also presents some limitations. A first problem is that it does not allow interactions between different platforms; for example, it is not possible to use event synchronization between devices from different vendors. Secondly, the semantics of OpenCL host applications is somewhat too verbose, as it includes different levels of abstraction (platform, device and context). Moreover, while writing an application targeting e.g. a cluster of heterogeneous nodes, we still require an intricate mix of OpenCL with a communication layer like MPI. Despite OpenCL can be easily extended in order to support remote, distributed devices (attempts in this direction are [22,1,20,11]), the host-device paradigm forces the use of a centralized communication pattern, which is a strong limitation for scaling on large-scale compute clusters. In this article, we introduce *libWater*, a library-based extension of the OpenCL programming paradigm that simplifies the development of applications for distributed heterogeneous architectures. *libWater* aims to improve both productivity and implementation efficiency addressing all the problems listed above. *libWater* does not alter the kernel logic of OpenCL kernels, but replaces the host-side API with a new, simpler and transparent interface which abstracts the underlying distributed architecture.

The main contributions of this article are:

- The *libWater* programming model, which extends the OpenCL standard by replacing the host code with a simplified and concise interface. It defines a novel device query language (DQL) for OpenCL device management and discovery, and introduces new features such as inter- and intra-context synchronization.
- A lightweight distributed runtime environment, which dispatches the work between remote devices, based on asynchronous execution of both communications and OpenCL commands. *libWater* runtime also collects and arranges dependencies between commands in the form of a powerful representation called command DAG.
- Two effective uses of the command DAG in order to improve scalability: (a) a Dynamic Collective Replacement (DCR) optimization, which identify collective communication patterns and replaces them with MPI collective operations; (b) a Device-Host-Device Copy Removal (DHDCR), where device-device communications supersedes device-host-device ones. Both optimizations overcome the limitation of the OpenCL host-device semantic, improving scalability on large-scale compute clusters.
- A study of the scalability of *libWater* on two real production clusters using up to 64 devices. Results show high efficiency and demonstrate the suitability of the presented command DAG optimizations for seven computational application codes. Finally we demonstrate the suitability of *libWater* for a heterogeneous cluster for two codes.

Our approach expands on previous work [13] by adding a new optimization (the DHDCR, in Section 6), new test cases (Section 6), new scalability studies on an additional target architecture, the MinoTauro GPU cluster (Section 7.2) of the Barcelona Supercomputing Center and new studies to test the suitability of libWater to exploit the computational capabilities of a heterogeneous cluster configuration (Section 7.3). With a wider range of applications, test platforms and optimizations, we show how *libWater* effectively improves the overall performance and scalability on large-scale compute clusters while easing the programmability.

The rest of the article is organized as follows. Sections 2 and 3 provide an introduction to OpenCL and *libWater* programming model. Section 4 describes the distributed runtime system and the underlying command DAG representation. The runtime optimizations are treated in Sections 5 and 6. The experimental evaluation is presented in Section 7. Sections 8 and 9 discuss related work and conclusions.

2. The OpenCL programming model

OpenCL is an open industry standard for programming heterogeneous systems. The language is designed to support devices with different capabilities such as CPUs, GPUs and accelerators. The platform model comprises a *host* connected to one or more *compute devices*. Each device logically consists of one or more compute units (CUs) which are further divided into processing elements (PEs). Within a program, the computation is expressed through the use of special functions called *kernels* that are, for portability reason, compiled at runtime by an OpenCL driver. Interaction with the devices is possible by means of *command-queues* which are defined within a particular OpenCL *context*. Once enqueued, commands – such as the execution of a kernel or the movement of data between host and device memory – are managed by the OpenCL driver which schedules them on the actual physical device.

Commands can be enqueued in a blocking or non-blocking way. A non-blocking call places a command on a command-queue and returns immediately to the host, while a blocking-mode call does not return to the host until the command has been executed on the device. For synchronization purpose, within a context, *event* objects are generated when kernel and memory commands are submitted to a queue. These objects are used to coordinate execution between commands and enable decoupling between host and devices control flows.

Despite being a well designed language that allows the access to the compute power of heterogeneous devices from a single, multi-platform source code base, OpenCL has some drawbacks and limitations. One of the major drawbacks is that, because being created as a low-level API, a significant amount of boilerplate code is required even for the execution of simple programs. Developers have to be familiar with numerous concepts (i.e. *platform*, *device*, *context*, *queue*, *buffer* and *kernel*) which make the language less attractive to novice programmers. Another important limitation is that, although it was designed to address heterogeneous systems, in case of devices from different vendors, objects belonging to the context of one vendor are not valid for other vendors. This limitation clearly becomes a problem when synchronization of command queues across different contexts is needed.

3. The libWater programming interface

libWater is a C/C++ library-based extension of the OpenCL programming paradigm that simplifies the development of distributed heterogeneous applications. It inherits the main principles from the OpenCL programming model trying to overcome its limitations. While maintaining the notion of host and device code, *libWater* exposes a very simple programming interface based on four key concepts: *device*, *buffer*, *kernel* and *event*. A *device* represents a compute device, but differently from the original paradigm this single object is an abstraction of the OpenCL platform, device, queue and context concepts. Such simplification reduces the number of source code lines necessary for the initialization of the devices, and thus avoids the boilerplate configuration code that is usually present in every OpenCL program. Furthermore, the library is not restricted to a single node but, taking internally advantage of the message passing model, it provides access to devices on remote nodes as if they were locally available.

Since *libWater* can grant access to a large number of distinct devices, the selection of a particular one can be cumbersome.