Automatic Pruning of Autotuning Parameter Space for OpenCL Applications

Ahmet Erdem, Gianluca Palermo[6], and Cristina Silvano[6]

Department of Electronics, Information and Bioengineering
Politecnico di Milano

Abstract. OpenCL standard reaches more wider audience due to increasing the number of devices supporting it. This situation puts developers who want performance on large range of platforms in a difficult position. To solve this problem, autotuning frameworks are deployed. But the problem of design exploration space is seriously large because of OpenCL parameters. In this work, we introduce an approach which uses constraint programming to prune the design space before employing intelligent or exhaustive techniques to explore.

1 Introduction

The recent advances in computer architecture made heterogeneous computer systems available to not only data centers and supercomputers but also to commercial personal computers. Especially, with the advent of AMD APUs and Intel CPUs which include integrated GPUs, the heterogeneity of modern machines has increased. Furthermore, enabling discrete GPUs for general purpose computing has added another type of computation device to the system. While each system has provided different granularity of parallelism which needs to be properly exploited, the communication between various computation units must also be handled according to the needs of application as well.

Open Computing Language (OpenCL) which is maintained by Khronos consortium [3] is an open standard for developing parallel applications on heterogeneous systems by abstracting the underlying compute machine. OpenCL adopts data parallel approach by describing the parallel computations as a group of *work-items*, called *work-groups*. This hierarchical parallelism has been realized by launching kernel functions with a number of work-groups including a set of work-items. Kernel function describes how each work-item defines the operations that is to be carried out on a single data. Therefore the collection of work-items under all work-groups together expresses the data parallelism for an application. Although OpenCL defines the execution of the application that is portable between the devices conforming the OpenCL standard, it does not guarantee the performance to be optimal. Especially, moving applications to different types of architectures like from CPU to GPU may result significant loss of performance, this is the reason why OpenCL is not considered performance portable. On heterogeneous performance portability represents a challenging research issue.

One naive solution to performance portability is to develop separate kernel functions for each device the application is supposed to run. This solution makes development of application dramatically complicated when the system is heterogeneous, because of explicit management of multiple command queues and contexts in the presence of multiple vendors on the system.

Performance portability problem of OpenCL applications has been approached either by tuning of significant parameters described as in [6] or by introducing Domain-specific languages to annotate kernel and OpenCL code generation [2].

From another perspective, it is not always possible to access these parameters to tune if they are not being exposed by developers. The work of [1] tackles this problem by coalescing *work-groups* using compiler transforms while preserving the correctness of application.

In this work, we introduce an automation of extraction of OpenCL platform parameters and usage of the information that is gathered to aid the tuning process described in [6].

2 Proposed Methodology

The procedure of autotuning of an OpenCL application in order to get optimum performance without concerns of underlying architecture of the platform requires a set of parameters that define characteristics of the machine. In the case of OpenCL, these platform specific parameters are stated by the OpenCL standard itself. Furthermore, it is possible to gather them using the querying framework which is provided by the OpenCL standard. With these information gathered, it is possible to determine the size of the exploration space and then using intelligent methods for searching optimum design space.

Outside of platform parameters, there might be also application specific parameters that can be tightly related to platforms capabilities. An example of this situation is well-known tiled version of the matrix multiplication. Size of the tiles are considered as an application parameters and due to nature of the algorithm there is a sharing of information between work-items on the elements of the same tile. Due to OpenCL architecture design, this kind of communication requires local memory to be used. Therefore tile size is directly related to local memory usage which is a limited resource of the platforms.

There are some problems regarding with this approach; design space is larger for even simple applications, for instance, Nvidia Fermi architecture allows up to 1024 work-items for first and second dimensions and 64 work-item for the third dimension, resulting a 2^{26} different configurations already. Most of the configurations are not feasible in the sense that the kernel may not even launch or may fail during execution, due to illogical configurations parameters. Moreover these failed attempts of kernel launches do not provide any information about the sample that has been taken from design space. Hence effort and time are wasted on these ill-advised configurations.

In order to address this issue, the work in [6] presented a design space exploration flow that includes constraint programming to prune the design space

and eliminate infeasible solutions. This helps reduction of space while only using samples which makes sense within the scope of OpenCL standard. Fig. 1 demonstrates this idea simply.

Our work aims to improve the pruning phase by automating the extraction of platform specifications, to find constraints that are valid for all platforms, so that application programmer only needs to insert constraints related to application itself. For constraint programming, we used MiniZinc [4] constraint modelling language as it is used in [6]. Using OpenCL querying framework, for each OpenCL compliant device available on the machine we generated MiniZinc data files which include the following information about the device:

- maximum *work-group* size for each three dimensions.
- maximum number of total *work-group* a kernel launch may contain.
- number of compute units on the device.
- local memory size of the device.

In addition to these, a set of constraints that can be deduced from standard [3] has been used to generate platform constraint model, thus together with application constraints provided by programmer can prune the design space effectively. The generated platform constraints are as follows:

- total number of *work-groups* launched must be less than or equal to maximum *work-group* size.

$$workgroup_x * workgroup_y * workgroup_z <= max\_total\_wg \tag{1}$$

- each global work-item dimensions must be multiple of corresponding *work-group* dimension size.

$$\begin{aligned} global_x \% workgroup_x &== 0 \\ global_y \% workgroup_y &== 0 \\ global_z \% workgroup_z &== 0 \end{aligned} \tag{2}$$

- total number of work-groups should be equal or greater than number of compute units. Otherwise there will be idle compute units.

$$\begin{aligned} &global_x/workgroup_x + global_y/workgroup_y + \\ &global_z/workgroup_z >= num\_compute\_units \end{aligned} \tag{3}$$

3 Use Case Example

In order to test our approach, we used a machine with Intel i7-2630QM which is a quad-core CPU at 2.0Ghz and Nvidia GeForce GT 550M which is a mobile GPU with 96 CUDA cores. For testing purposes, tiled version of matrix multiplication is used and tile size is given as a application parameter and it is been set the

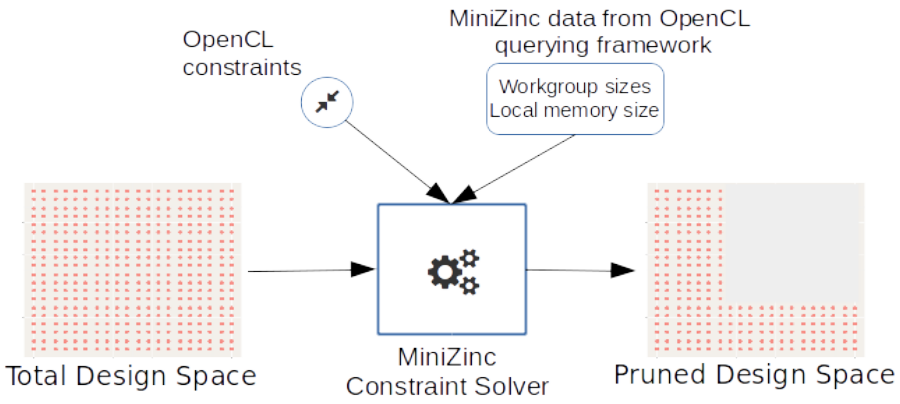

Fig. 1. Pruning of configuration design space

interval of [1 : 256]. Furthermore, we have taken into account that application global work size and set it to 4096x4096 matrix for this experiment.

Given this setting, the design space for CPU is 2^{34} and for GPU it is 2^{47}. But after the pruning of infeasible configurations, there are only 367 configurations left for CPU and 421 configurations for GPU. Considering the result of pruning for the use case given, it is even possible to search exhaustively all possible configurations left to find the optimal one. However the given example is too elementary to deduce this conclusion for broader range of applications. Hence, the next step is to introduce tools to at least one of the industry-proven benchmarks like OpenDwarfs [5], Rodinia [7] and shoc [8]. Moreover, testing with more diverse and recent hardware platforms is necessary to prove generality of the work. Additionally, besides platform specific and application specific parameters, there is a possibility of adding compiler-supported parameters like coalescing factor that has been explored in [1].

References

1. G. Agosta, A. Barenghi, G. Pelosi, and M. Scandale. Towards transparently tackling functionality and performance issues across different opencl platforms. In *In proceedings of the Second International Symposium on Computing and Networking Across Practical Development and Theoretical Research (CANDAR 2014)*, Dec. 2014.
2. N. Chaimov, B. Norris, and A. Malony. Toward multi-target autotuning for accelerators. In *Parallel and Distributed Systems (ICPADS), 2014*, pages 534 – 541.
3. Khronos Group. The open standard for parallel programming of heterogeneous systems. [Online; Accessed: Nov. 2015].
4. MiniZinc. Medium-level constraint modelling language minizinc. [Online; Accessed: Dec. 2015].
5. OpenDwarfs. Opendwarfs. [Online; Accessed: Jan. 2016].
6. E. Paone, F. Robino, G. Palermo, V. Zaccaria, I. Sander, and C. Silvano. Customization of OpenCL applications for efficient task mapping under heterogeneous

platform constraints. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, pages 736–741. EDA Consortium, 2015.
7. Rodinia. Rodinia:accelerating compute-intensive applications with accelerators. [Online; Accessed: Jan. 2016].
8. Shoc. Scalable heterogeneous computing (shoc). [Online; Accessed: Jan. 2016].