Design Space Pruning and Computational Workload Splitting for Autotuning OpenCL Applications

Ahmet Erdem Davide Gadioli

Gianluca Palermo Cristina Silvano

Department of Electronics, Information and Bioengineering
Politecnico di Milano
e-mail: name.surname@polimi.it

ABSTRACT

Recently, OpenCL standard reached much wider audiences due to the increasing number of devices supporting it. At the same time, we have observed an increase of differences among devices that support OpenCL. This situation offers to developers, who want to get high performance, a large spectrum of platforms. Given the additional OpenCL platform parameters alongside application specific parameters, the design space for exploration is seriously large. Furthermore, availability of more than one kind of device allows distribution of computation on the heterogeneous platform. Automatic design space exploration frameworks are one of the recent approaches to address these problems and to reduce the burden of programmers. In this work, we present our automatic and efficient technique to prune the design space before moving on to the exploration phase and we propose a new method for splitting the computational tasks to computing devices on heterogeneous platforms.[1]

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 devices must also be handled according to the requirements of application as well.

[1]This work is partially funded by the EU H2020 FET-HPC program under grant 671623 ANTAREX.

© 2018 ACM. ISBN 978-1-4503-6417-1.
DOI: https://doi.org/10.1145/3180665.3180669

Open Computing Language (OpenCL), maintained by the Khronos consortium [4], 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*, named *work-groups*. In this hierarchical mechanism, a kernel function is executed in parallel by each work-item in the work-group. A kernel function describes how each work-item defines the operations 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 guarantees that the execution of the application 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. Heterogeneous platforms 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. Moreover, this approach has more design flaws:

- The application developer must have knowledge of all the device's architectures.
- The application must be manually split between existing devices on the platform.
- The developer should have access to all the devices in order to test and profile on them.
- The number of devices targeted by the application is limited, consequently future architectures are impossible to target in a performance optimal way without modifying the application code.

The performance portability problem of OpenCL applications has been approached either by tuning significant parameters as described in [6] or by introducing Domain-specific languages to annotate kernel, to generate more specialized OpenCL code[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 transformations while preserving the correctness of application.

In Glinda framework presented by [8], a specific application with possible imbalanced workload is analyzed and used as a case study for their load balancing and autotuning framework.

Sharing similar vision with [8], Alok Prakash et al. demonstrate in their work [7] how they approached the problem of utilization of heterogeneous computing platform on an embedded device.

The open source library named Maestro [9], which is introduced by Kyle Spafford et al., employs autotuning techniques to find optimal work-group size and load balancing between multiple devices on heterogeneous platforms.

Similarly, in the work of [3], adaptive ways of selecting faster architecture using source-to-source polyhedral compiler explored. Concurrent runs on devices are not used to accelerate the execution, but rather to use the fastest device that finishes the execution.

In this work, we introduce an automation of extraction of OpenCL platform parameters and usage of the information that are gathered to aid the tuning process. Furthermore, we propose a new method to split the computational workload to different OpenCL devices on heterogeneous systems.

2. PROPOSED METHODOLOGY

2.1 Parameter Space Pruning

The procedure of autotuning an OpenCL application in order to get optimum performance without any concerns of the 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.

Besides platform parameters, there might be also application specific parameters that are tightly related to platforms capabilities. An example of this situation is the well-known tiled version of the matrix multiplication. The size of the tiles are considered as 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 related to searching for the optimum configuration, design space is larger for even simple applications. For instance, Nvidia Fermi architecture has limitations on work-group sizes for each dimension allowing up to 1024 work-items for first and second data-dimensions, while 64 work-items for the third dimension, resulting in 2^{26} different configurations. Most of these configurations are not feasible (e.g. the total number of work-items may exceed devices capabilities), in the sense that the kernel may not even launch or may fail during execution, due to unfeasible 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 unfeasible configurations.

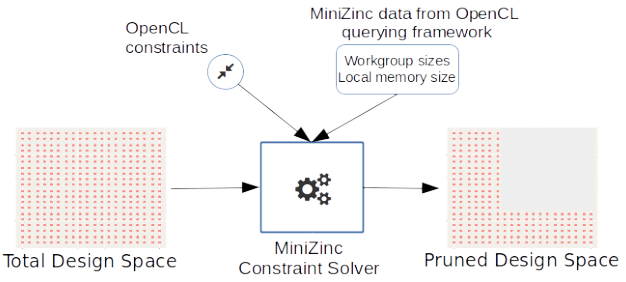


Figure 1: Design Space Pruning

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 unfeasible solutions. This helps the reduction of the design space. Moreover, it only uses configurations which make sense within the scope of OpenCL standard. Fig. 1 demonstrates this idea. Constraint Solver shown in Figure 1, eliminates the samples which do not comply with the constraints from the design space. And as output of the solver, a pruned design space is generated such that all configuration samples are compliant with the constraints.

Our work aims at improving the pruning phase by automating the extraction of platform specifications, to find constraints that are valid for all the OpenCL devices in the target system. Therefore, application programmer only needs to insert constraints related to application itself. For constraint programming, we use the MiniZinc [5] constraint modelling language. Using *clGetDeviceInfo* function provided by OpenCL querying framework, for each OpenCL compliant device available on the machine we generate MiniZinc data files which include the following information about the device:

- maximum *work-group* size for each dimension.
- maximum total number of work-groups considering all dimensions.
- number of compute units on the device.
- local memory size of the device.

In addition to these device parameters, a set of constraints that can be deduced from the rules defined by the standard [4], has been used to generate platform constraint model using MiniZinc constraint programming language. Thus, together with application constraints provided by programmer, it is possible to prune the design space effectively. The generated platform constraint model contains the following rules:

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

$$workgroup_x * workgroup_y * workgroup_z <= max\_total\_wg \quad (1)$$

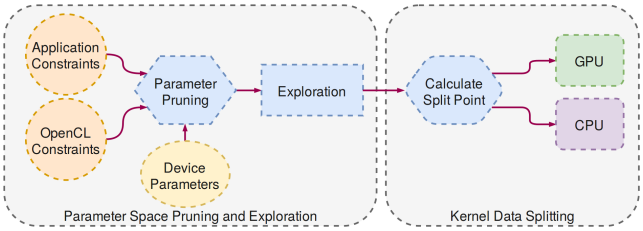


Figure 2: The proposed methodology

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

$$\begin{aligned} global_x \% workgroup_x == 0 \\ global_y \% workgroup_y == 0 \\ global_z \% workgroup_z == 0 \end{aligned} \quad (2)$$

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

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

Using both constraints coming from platform specifications and application domain, the total design space will be reduced to a collection of configurations that satisfy these constraints. This will reduce the exploration of the space, which is crucial for the next phase.

2.2 Computational Workload Splitting

Heterogeneous computing architectures are widely adopted, thus being able to utilize this heterogeneity is crucial for achieving higher performance systems. Therefore, in this work we propose a new method that splits huge OpenCL computing kernels into smaller chunks. Then it maps those partial computations to different devices in order to make use of all the existing OpenCL devices.

Since the development of OpenCL framework has been inspired from GPUs, which have been conceived for data-parallel computing, we are focusing on data-parallel computations by using the *NDRange* functionality of OpenCL framework. When using *NDRange*, it is possible to launch kernels on an iteration space where each iteration processes one element of a set of data. The goal is to be able to prune that iteration space and distribute the portions of it to the available devices in a efficient manner.

To achieve this goal, we should address the following issues:

- It is required the performance knowledge of a kernel on each device of the heterogeneous platform in order to find the right split point.
- After splitting, the chosen work-group sizes may become ill-advised for kernel execution.

In order to find a balanced splitting point, we needed performance information of the target kernel on each device. To acquire this information, we measured execution time of the remaining configurations after parameter space pruning described in Section 2.1. After the exploration, the candidate

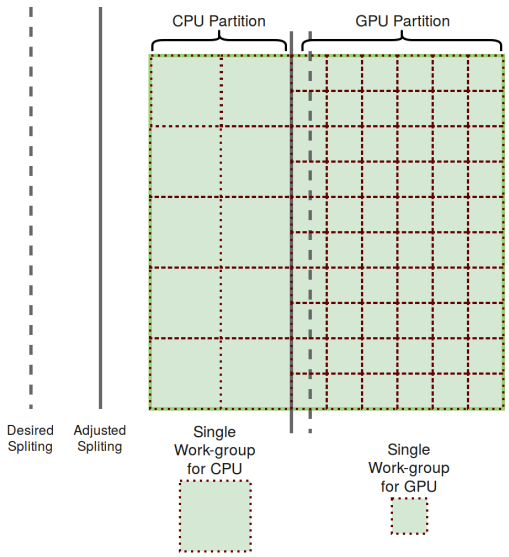


Figure 3: Global work space splitting

configurations with the lowest execution time for each device are chosen for splitting decision. In Figure 2, the proposed methodology has been shown to give high level perspective.

First, execution times are converted to speed values for each device by taking multiplicative inverse. Then, these speed values are used for calculating *split factors* for each device using Equation 4; where $split\_factor_i$ is the *split factor* of i^{th} device and N is the number of devices.

$$split\_factor_i = exe\_speed_i / (\sum_{j=0}^{N-1} exe\_speed_j) \quad (4)$$
$$\text{where} \quad i = 0, \ldots, N-1s$$

Since split factors define how much computation will take place in each OpenCL device, the sum of all the split factors must be equal to 1.0. Otherwise, there would be residue computation that is missing from the output of the task. It is fairly easy to observe that $\sum_{i=0}^{N-1} split\_factor_i = 1.0$ is satisfied by substituting $split\_factor_i$ by the Equation 4.

While split factors are calculated from performance measurements to give good hints about how to load balance between different devices in heterogeneous environments, dividing the global work sizes using a generic split factors, may easily result in partitions with fractional work sizes which are invalid for the OpenCL standard.

For instance, let us consider a global work size of 512 work-items and split factors of 0.4 and 0.6 for the two devices. In this case, the share of device 1 would be 204.8 work-items while device 2 has a share of 307.2 work-items. Because the concept of work-items is intrinsically indivisible, the splitting factors are impractical.

Moreover, it is important to keep the optimum work-group sizes found by exploration for each device, since the performance is strongly related to work-group sizes. This situation is problematic because different devices usually have different optimal work-group sizes. In [7], a single work-group size has been used for both GPU and CPU. In contrast, this work introduces a new technique to overcome this limitation by adjusting the splitting factors minimally while taking into

account the optimal work-group sizes discovered in the exploration phase.

Equation 5 defines the global work size. where wg_i is the work-group size and γ_i is the number of work-groups for i^{th} device. For simplicity, it only considers one dimension of the iteration space. Moreover, it can be extended for the multi-dimensional case, since splitting operation can be applied to each dimension separately.

$$G = \sum_{i=0}^{N-1} wg_i * \gamma_i \tag{5}$$

The number of work-groups that are needed for each device is calculated using devices' optimal work-group sizes and splitting factor S_i (Eq. 6).

$$\gamma_i = \frac{G * S_i}{wg_i} \tag{6}$$

It is important to note that at this stage, the numbers of work-groups γ_i are real values, hence OpenCL framework will not launch successfully. To deal with this issue, the number of work-groups are reduced to an integer number for all devices (Eq. 7).

$$\hat{\gamma_i} = \lfloor \gamma_i \rfloor \tag{7}$$

With the integer number of work-groups $\hat{\gamma_i}$, the splitting factors are recalculated (Eq. 8a) in order to find the residue (Eq. 8b).

$$\hat{S_i} = \frac{wg_i * \hat{\gamma_i}}{G} \tag{8a}$$

$$S_R = \sum_{i=0}^{N-1} S_i - \sum_{j=0}^{N-1} \hat{S_j} \tag{8b}$$

Having a leftover part of the computation ($G * S_R$), we determine how much work each device needs to process, in order to compute residue (Eq. 9a and Eq. 9b). Then we choose the device that requires the minimum amount of work to cover the remaining computations.

$$\tilde{\gamma_i} = \lceil \frac{G * S_R}{wg_i} \rceil \tag{9a}$$

$$\tilde{S_i} = \frac{wg_i * \tilde{\gamma_i}}{G}, \tag{9b}$$

Using this method, it is possible to preserve the desired work-group sizes while adjusting split factor minimally. We achieve this by reducing the number of work-groups conservatively per device, then choosing the appropriate device for the residue work to be computed on. An example of splitting and partitioning of global work space into work-groups is illustrated in Figure 3.

An interesting feature of this method is that, when work-group sizes of devices are not multiple of each other, there will be overlap between the partitions of data that are computed on devices. However the redundant computations will be minimum due to choice of the device which requires least amount of work for residue. In case the work-group sizes are multiple of each other, there will be no overlap at all. This outcome is observed because when work-group sizes are multiple of each other, larger work-groups can perfectly be accommodated by smaller work-groups.

Algorithm 1 Calculates the best theoretical performance

```
function TheoreticalHeteroPerf(splitFactors, exeTimes)
    devicePerfs ← empty list
    i ← 0
    while i < number of devices present do
        devicePerfs[i] ← splitFactors[i] * exeTimes[i]
    return MAX(devicePerfs)
```

Figure 4 shows the details of the splitting phase. In this phase, we also measure how well the methodology stands against theoretically best-case performance. After calculation the best split location, the theoretical heterogeneous performance is calculated using Algorithm 1. The difference between the calculated performance and the actual heterogeneous performance is the overhead of our methodology.

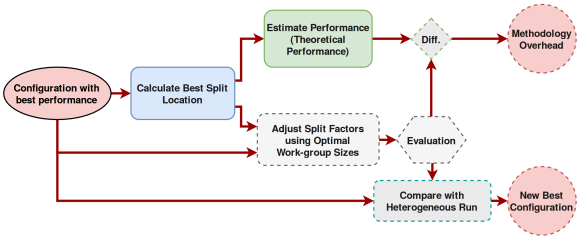


Figure 4: Proposed splitting phase in detailed

3. EXPERIMENTAL SETUP

In order to test our approach, we used two platforms. Platform 1(PLT1) consists of an Intel i7-4770 quad-core at 3.4Ghz and Intel HD Graphics 4600 with 20 Execution Units. Platform 2(PLT2) has an Intel i7-2630QM quad-core CPU at 2.0Ghz and a Nvidia GeForce GT 550M which is a mobile GPU with 96 CUDA cores.

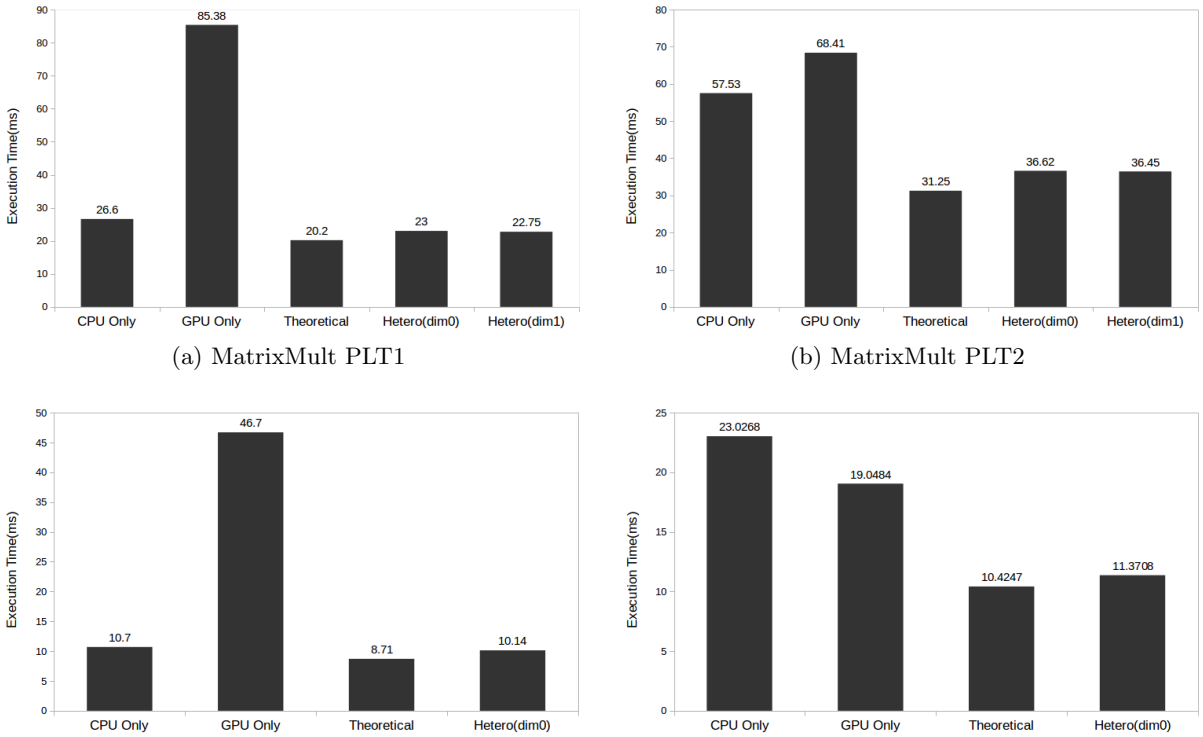
In order to validate our methodology, we have chosen two application case studies; one dimensional convolution and matrix multiplication. Both implementations make use of local memory provided by OpenCL framework as cache.

Matrix multiplication is implemented as tiled version and the size of the considered tiles is an application parameter and it is used to calculate the local memory consumption of the OpenCL kernel (Eq. 10). As a constraint to the exploration space, we fixed the workgroup sizes equal to the tile sizes.

$$local\_mem = 2 \times tile\_size^2 \tag{10}$$

OpenCL version of the convolution operation has been implemented in a fashion that work-items, in the same work-group, share as much data as possible. This is possible by using local memory to load all the neighbouring data required for the whole work-group. Therefore, the number of work-items, as well as the mask size defined in the convolution operation, affects the local memory consumption according to (Eq. 11).

$$local\_mem = workgroup\_size + mask\_size - 1 \tag{11}$$



(a) MatrixMult PLT1

(b) MatrixMult PLT2

(c) Convolution PLT1

(d) Convolution PLT2

Figure 5: Execution times of heterogeneous runs

4. EXPERIMENTAL RESULTS

In this section, we present how much the design space is reduced by using our methodology. The amount of acceleration achieved due to utilization of both CPU and GPU as well as the limitations of the techniques that are introduced is examined. In order to get repeatable outcomes, we evaluated all the configurations 10 times and took the mean of the best five results for all the experiments.

4.1 Design Space Pruning

Without our pruning phase implementation, the required amount of kernel runs for the explorations are shown in Table 1. The matrix multiplication used as test case is a 1024x1024 multiplication, while the convolution operation has a mask size 625 over 2^{16} elements.

The numbers in Table 1, are generated considering only the dimensions that are used by the kernel. Matrix multiplication has a 2 dimensional iteration space, while convolution is 1 dimensional in this case. This is the reason behind the huge difference of the exploration size of the two applications. In Table 2, the number of feasible configurations that require evaluation, are dramatically reduced. The amount of reduction of the space allows us to explore all the remaining configurations.

4.2 Computation Workload Splitting

Figure 5 shows the theoretical value, execution time of the heterogeneous evaluations, along with only CPU and only GPU configurations. For the matrix multiplication case two different heterogeneous experiments are shown as dim0 and dim1 in Figure 5. It is important to notice that on PLT1 the GPU is an integrated GPU and on PLT2 the GPU is a mobile variant. Moreover, both CPUs and GPUs use the same kernel with parameters tuned to the specific device. For all the execution types, we show the configuration with the minimum execution time among the explored ones.

Table 1: Exploration space size without pruning

	PLT1		PLT2	
	CPU	GPU	CPU	GPU
Matrix Mult.	2^{20}	2^{18}	2^{20}	2^{20}
Convolution	8192	512	8192	1024

Table 2: Exploration space size after pruning

	PLT1		PLT2	
	CPU	GPU	CPU	GPU
Matrix Mult.	7	5	7	6
Convolution	24	17	24	19

We may observe that the performance differences between the CPU and the GPU on PLT1 lead to less than 15% improvement for the two cases. Even the theoretical value does not suggest much better speed up (Fig. 5a, Fig. 5c). Contrarily, on PLT2 with matrix multiplication and convolution applications, there is a 54% and a 68% speed up respectively (Fig. 5b, Fig. 5d).

We have also investigated the effect of kernel data splitting on different dimensions of the iteration space. In order

Table 3: Data split factors (CPU,GPU)

	PLT1		PLT2	
	Adjusted	Optimal	Adjusted	Optimal
MatMul.	(0.762, 0.238)	(0.75, 0.25)	(0.55, 0.45)	(0.5, 0.5)
Conv.	(0.814, 0.186)	(0.812, 0.188)	(0.45, 0.55)	(0.45, 0.55)

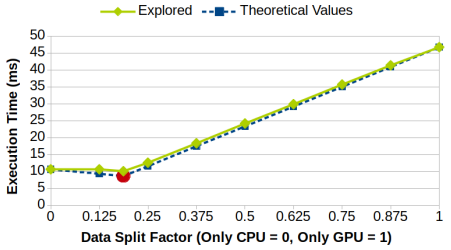


Figure 6: Split Factor Evaluations(Convolution, PLT1)

to test it, splitting technique has been applied to both dimensions (dim0, dim1) of matrix multiplication. According to the experiment results, there is no significant improvements for this case study due to the symmetric distribution of the workload.

Table 3 shows the calculated optimal split factors and split factors that are adjusted according to optimal work-group sizes explored. For each case, the first value is the computational share of CPU and second one is the share of GPU. It is important to note that for all cases, it sums up to 1.0, meaning that the computation covers all the iterations space. Overall, the difference between the optimal split point and the split point adjusted by our methodology is smaller in convolution application, due to the fact that the size of the work-groups are much smaller compared to global work size. This situation provides much a finer granularity when adjusting the splitting point, therefore leading to closer to optimal splitting. In contrast to this, the matrix multiplication work-group sizes are not as insignificant to the global work size as convolution, although differences between the adjusted and the optimal are still limited.

In order to further evaluate our methodology, we tried different split points to compare with our split point calculation (Fig. 6, Fig. 7). The splitting factor, indicated with a circle, is the factor computed by our method. On Platform 2 it is obvious that our splitting factor is the balancing point between the two devices. While it is also the case on Platform 1, the shape of the graph indicates the performance gap between two computing devices used on the platform. Additionally, the difference between the calculated execution times (dashed blue line) and the explored execution times (solid green line) is the overhead of the technique we introduced. This overhead includes adjustment of splitting point and runtime management of multiple OpenCL devices.

5. CONCLUSION & FUTURE WORK

The contribution of this work is twofold. The first contribution is a more automatic way of collecting and using OpenCL parameters as an improvement on [6]. Secondly, a new technique on how to split data between OpenCL devices while respecting work-group sizes has been introduced. Compared to [7], it is not required to have the same work-

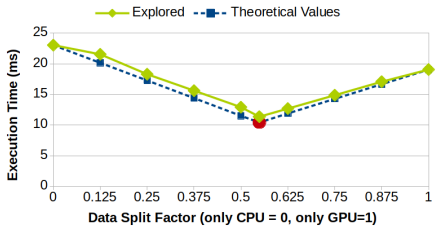


Figure 7: Split Factor Evaluations(Convolution, PLT2)

group sizes for all the devices. Removing this limitation enables the usage of better suited work-group sizes for each device.

Experimental results have shown that it is possible to significantly reduce exploration space and moreover by utilizing all the devices available on the heterogeneous platform, our methodology improves the performance.

6. 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] J.-F. Dollinger and V. Loechner. Adaptive runtime selection for gpu. In *Proceedings of the 2013 42Nd International Conference on Parallel Processing*, ICPP '13, pages 70–79, Washington, DC, USA, 2013. IEEE Computer Society.

[4] Khronos Group. The open standard for parallel programming of heterogeneous systems. [Online; Accessed: Nov. 2015].

[5] MiniZinc. Medium-level constraint modelling language minizinc. [Online; Accessed: Dec. 2015].

[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, 2015.

[7] A. Prakash, S. Wang, A. E. Irimiea, and T. Mitra. Energy-efficient execution of data-parallel applications on heterogeneous mobile platforms. In *Computer Design (ICCD), 2015 33rd IEEE International Conference on*, pages 208–215. IEEE, 2015.

[8] J. Shen, A. L. Varbanescu, H. Sips, M. Arntzen, and D. G. Simons. Glinda: a framework for accelerating imbalanced applications on heterogeneous platforms. In *Proceedings of the ACM International Conference on Computing Frontiers*, page 14. ACM, 2013.

[9] K. Spafford, J. Meredith, and J. Vetter. *Maestro: Data Orchestration and Tuning for OpenCL Devices*, pages 275–286. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.