Adaptive Partitioning for Iterated Sequences of Irregular OpenCL Kernels

Pierre Huchant, Denis Barthou, Marie-Christine Counilh

2018 30th International Symposium on Computer Architecture and High Performance Computing

Adaptive Partitioning for Iterated Sequences of Irregular OpenCL Kernels

Pierre Huchant
Inria / LaBRI
Bordeaux INP, U. of Bordeaux
Talence, France
pierre.huchant@inria.fr

Denis Barthou
Inria / LaBRI
Bordeaux INP, U. of Bordeaux
Talence, France
denis.barthou@inria.fr

Marie-Christine Counilh
Inria / LaBRI
Bordeaux INP, U. of Bordeaux
Talence, France
marie-christine.counilh@inria.fr

***Abstract*—OpenCL defines a common parallel programming language for all devices, although writing tasks adapted to the devices, managing communication and load-balancing issues are left to the programmer. We propose in this paper a static/dynamic approach for the execution of an iterated sequence of data-dependent kernels on a multi-device heterogeneous architecture. The method allows to automatically distribute irregular kernels onto multiple devices and tackles, without training, both load balancing and data transfers issues coming from hardware heterogeneity, load imbalance within the application itself and load variations between repeated executions of the sequence.**

***Index Terms*—OpenCL, Irregular Workload, Load Balancing, Static Analysis, Dynamic Analysis**

I. Introduction

Graphic Processor Units (GPU) are ubiquitous and nowadays most computing nodes of a parallel machine consist in GPUs and multicore CPUs. In terms of programming language, OpenCL has emerged as the programming language for heterogeneous computing, able to define code for GPUs and CPUs alike. However, this introduces new challenges: The application code has to be adapted to the number of devices and the workload has to be balanced equally between these devices. Load balancing is difficult to achieve in general, because the architecture is heterogeneous, the parallel application may not have a constant load and both computation and communication times have to be taken into account.

In this paper, we focus on applications with an iterated sequence of data-dependent OpenCL kernels. This occurs in iterative computations, for instance until a fixed point is reached or for a simulation, where each iteration corresponds to a time step. There is no necessarily enough parallelism between the kernels. The method we propose is to automatically partition each kernel at load time into sub-kernels, one per device. The size of the partition is then adapted at runtime after each iteration, taking into account both execution and communication times. We show that our partitioning method is able to handle sequences of kernels with irregular workload, dynamic load variations and takes into account the communication times between kernels induced by their respective partitioning. Our method automatically transforms a single device multi-kernel application into a portable, heterogeneous multi-device and multi-kernel application.

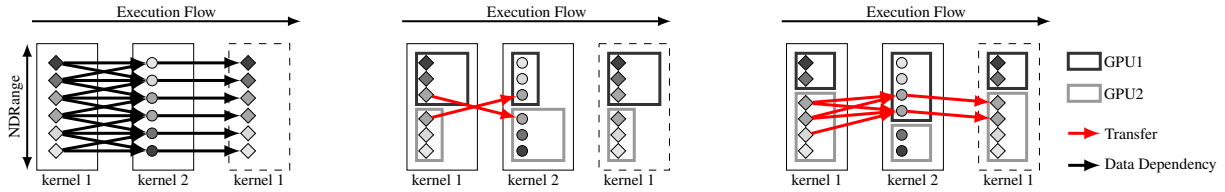
The major contributions of this paper are the following:

- Design and implementation of a framework to automatically adapt a single device application with multiple kernels to any number of devices.
- Automatic partitioning of the data accessed across devices with complex memory access patterns, including indirections.
- Dynamic load balancing for each iteration of the computation, handling irregular workload inside kernels, load variations and communication between devices.

II. Motivation

Figure 1 illustrates two different strategies when distributing a sequence of 2 kernels over 2 devices. The structure of the application is shown Figure 1a. Threads from the iteration spaces of kernels 1 and 2 are respectively represented with diamonds and circles. The shades of gray represent the amount of work of each thread. Both kernels have irregular workload: for kernel 1 (resp. kernel 2), threads from the beginning of the iteration space have more work (resp. less work) than threads at the end of the iteration space. kernel 2 exhibits the structure of a stencil: each thread from its iteration space depends on the data produced by the thread from kernel 1 at the same position and also on the data produced by its two neighbor threads. For kernel 1 however, each thread only depends on the data produced by the thread at the same position in kernel 2. Figure 1b illustrates a uniform partitioning strategy over 2 GPUs, where the two sub-kernels of kernel 1 and kernel 2 have a partitioning ratio of 0.5. A partitioning ratio of 0.5 on a device means that half of the iteration space of the kernel is executed on this device. For this specific application, this partitioning minimizes the amount of data to transfer. However the execution times of the sub-kernels are imbalanced. Figure 1c illustrates another partitioning that minimizes the computation time of each kernel. kernels 1 and 2 must then have different partitioning ratios to balance the execution time of their sub-kernels. However this partitioning implies much more data transfers, and may result in a huge slowdown.

This illustrative example shows the impact of the partitioning of each kernel from the sequence on the volume of data to transfer. Only considering the execution times is not sufficient in order to minimize the overall execution time of



(a) Application Structure (b) Minimizing Transfers (c) Minimizing Computation

Fig. 1: Different Partitioning Strategies

a kernel sequence. The partitioning that minimizes the overall execution time of a kernel sequence is a trade-off between the balancing of the execution times of the sub-kernels and the cost of the data-transfers induced. This partitioning depends on the architecture heterogeneity, the workload of the application, the data-dependencies between kernels, and the iteration number in case of dynamic load variations.

III. Principle of Dynamic Adaptation

This section describes how we automatically adapt OpenCL kernels to heterogeneous multi-device architectures using a combination of static and dynamic approaches.

a) Static Analysis and Transformation: At load-time, our framework computes for each kernel the parametric read and write array regions it accesses. These regions consist of a union of intervals defining the values of indices in the array that may be read/written by the kernel. These intervals may depend on thread ids, on the values of the scalar parameters of the kernels and on the values of other array elements in the case of indirections. We assume here that indirections are monotonous functions, preserving intervals, such as for Compressed Sparse Row format (CSR) or for spatial binning structures. Then all kernels are transformed into *partition-ready kernels* which can be launched with a fraction of the original NDRange as in [3].

b) Runtime Adaptation: At each iteration, a new partitioning of the parallel iteration space defined by the NDRange of each kernel is computed. Each device then executes each kernel on a fraction of its NDRange called *sub-NDRange*. These *sub-kernels* correspond to the partition-ready kernels instantiated with the sub-NDRange resulting from the partitioning computed. For the first iteration, each kernel is partitioned using a Uniform strategy. For the following ones, the partitioning of each kernel is computed by solving a linear system. This linear system, presented in the next section, takes into account both communication and computation times and is based on previous iteration measures. Each time a new partitioning of the kernel sequence is computed, the parametric read and write array regions associated to each sub-kernel are instantiated with the values of its scalar parameters, its current sub-NDRange and possibly the value of some array elements in case of indirections. We then use these instantiated regions to only transfer the data missing on each device.

IV. Dynamic Load Balancing

This section defines a new method to determine how to partition an iterated sequence of m kernels onto n devices. The partitioning of each kernel in the sequence is computed by solving a linear system at each iteration. At the end of iteration t, the linear system computes the partitioning ratios of each kernel in the sequence, in order to minimize the overall execution time of iteration $t+1$.

A. The Adaptive w/o Comm Strategy

The Adaptive w/o Comm strategy finds partitioning ratios for each kernel from the sequence individually in order to minimize their execution times. This strategy does not take into account the transfer times between kernels induced by their respective partitioning. In this case, the linear system relies only on the execution times of the sub-kernels at iteration t to determine the partitioning ratios for iteration $t+1$. The linear system presented below is a generalization of the formulation presented in [3] to a sequence of m kernels.

The Adaptive w/o Comm strategy consists in finding the execution times $T^1, \ldots, T^m$ of each kernel, and the new partitioning ratios y_d^k of each kernel k on each device d for iteration *t+1* such that the following system is fulfilled:

$$\begin{cases} \min T^1 + \ldots + T^m \\ \forall k = 1..m : \\ \quad \forall d = 1..n : f_d^k(x_1^k, \ldots, x_d^k, t) * ng_d * y_d^k \leq T^k, \\ \quad \sum_d y_d^k = 1 \end{cases}$$

The *partitioning ratio* for a kernel k and device d is a value x_d^k in $[0,1]$ (with $\sum_{d=1}^{n} x_d^k = 1$) corresponding to the ratio between the number of work-groups allocated to the device d and the total number of work-groups (ng_k). ng_k is known when kernel k is called. We define $f_d^k(x_1^k, \ldots, x_d^k, t)$ as the mean time to execute one work-group on device d at iteration t, when sub-kernels on device $1, \ldots, d$ have respectively partitioning ratios $x_1^k, \ldots, x_d^k$. The execution time of the sub-kernel of k on device d is $f_d^k(x_1^k, \ldots, x_d^k, t) * ng_k * x_d^k$ and the total execution time of kernel k is $\max_{d=1,n}(f_d^k(x_1^k, \ldots, x_d^k, t) * ng_k * x_d^k)$. The functions f_d^k are not known precisely but we determine the value of f_d^k as the execution time of the sub-kernel of k on device d at iteration t.

B. The Adaptive w/ Comm Strategy

The Adaptive w/ Comm strategy takes into account the data transfer times induced by the partitioning of each kernel from the sequence in order to minimize the overall iteration time. We model the volume of data to transfer between two data-dependent kernels as a function of their partitioning ratios. At each iteration, the parametric read and write regions of all

sub-kernels are instantiated and we can determine the value of this function for the current partitioning ratios. At runtime, using a linear regression, we compute for each pair of data-dependent kernels and for each device a coefficient giving the volume of data to transfer depending on the partitioning ratios of these kernels. Then, using these coefficients we add new constraints to the linear system presented in IV-A modeling the communication times between all data-dependent kernels.

The objective function to minimize becomes:

$$\min T^1 + \ldots + T^m + T^1_{H2D} + T^1_{D2H} + \ldots + T^m_{H2D} + T^m_{D2H}$$

where T^k_{D2H} and T^k_{H2D} are devices-to-host and host-to-devices transfer times before the execution of kernel k. For each kernel k and device d we add the two following linear constraints:

$$T^k_{H2D_d} \leq T^k_{H2D} \quad (1)$$

$$\sum_{h=1}^{m} a_d^{h,k} * S_d(y_1^h, \ldots, y_d^h, y_1^k, \ldots, y_d^k) * \Omega_d \leq T^k_{H2D_d} \quad (2)$$

where: the y_d^k are the unknowns of the system; $a_d^{h,k}$ and Ω_d are coefficients determined at runtime; S_d is a relation on the partitioning ratios y_d^k of kernel k and on the partitioning ratios y_d^h of kernel h on which k depends. (1) means that the transfer time from the host to the n devices before executing kernel k is equal to the longest host to device data transfer time (transfers to different devices are performed in parallel). (2) means that the host to device d transfer time before executing kernel k is the sum of the host to device d transfer times from each kernel h on which k depends. The meaning of the relation S_d and the coefficients $a_d^{h,k}$ and Ω_d are explained in the next paragraph. Similar constraints are added for device to host transfers. If all kernels depend on all kernels, $2 * (m + m * n + m^2 * n)$ constraints are added to the linear system. However in most applications kernels do not depend on all other kernels and we show in the next section that the overhead induced by resolving the system at each iteration is negligible.

Let us now explain our communication modelization. When a kernel h writes to a buffer B that is read by kernel k, data transfers may be required when those kernels are partitioned onto multiple devices. It is the case if a sub-kernel of k executed on device d reads a region of B that is written by a sub-kernel of h executed on another device. This data then comes from another device and a communication is required. Let us assume first that the kernels h and k have a NDRange of size N and that each thread from h (resp. k) writes (resp. reads) buffer B at the index corresponding to its id in the NDRange. The region of B written by the sub-kernel of h on device d and the region of B read by the sub-kernel of k are respectively: $W_d(y_1^h, \ldots, y_d^h) = N * [y_1^h + \ldots + y_{d-1}^h, y_1^h + \ldots + y_d^h]$ and $R_d(y_1^k, \ldots, y_d^k) = N * [y_1^k + \ldots + y_{d-1}^k, y_1^k + \ldots + y_d^k]$.
The data not present on device d before execution of the sub-kernel of k is defined by the region $R_d(y_1^k, \ldots, y_d^k) - W_d(y_1^h, \ldots, y_d^h)$. When the two regions overlap, the amount of data to transfer is:

$$\begin{aligned} S_d(y_1^h, \ldots, y_d^h, y_1^k, \ldots, y_d^k) = {} & N * max(\sum_{i=0}^{d-1} y_i^h - \sum_{i=0}^{d-1} y_i^k, 0) \\ & + N * max(\sum_{i=0}^{d} y_i^k - \sum_{i=0}^{d} y_i^h, 0), \end{aligned}$$

otherwise the amount of data is simply $N * y_d^k$. In real applications, two dependent kernels do not necessary have the same NDRange and threads can write buffers at any location. We model the communication volume from host to device d related to h and k as a function of their partitioning ratios $g_d^{h,k} = a_d^{h,k} * S_d$ where $a_d^{h,k}$ is a coefficient and S_d is over-approximated by always considering that W_d overlaps R_d. At each iteration the parametric regions of h and k are instantiated and we know the value of $g_d^{h,k}$ for their current partitioning ratios. Hence, the coefficients $a_d^{h,k}$ are computed at runtime using a linear regression. Finally the data transfer time from host to device d related to h and k is $a_d^{h,k} * S_d * \Omega_d$ where Ω_d is the time to transfer one byte from host to device d.

V. Evaluation

We evaluate our method on 2 regular benchmarks: Jacobi and FDTD2D, 1 irregular benchmark 2SpMV and 1 application with dynamic load variations: SOTL, on a platform with a 16-core Intel Xeon E5-2650 2.00 GHz and 3 Nvidia Tesla M2075 GPUs. The Jacobi benchmark (2 kernels) consists in stencil kernel followed by a memcpy from the output buffer to the input. FDTD2D consists in a succession of 3 stencil kernels. The 2SpMV benchmark (2 kernels) consists in a Sparse Matrix-Vector Multiplication applied on two different matrices, the output vector of one kernel is the input vector of the following one. Both kernels present irregular workload among threads, due to the sparsity structure of each matrix. SOTL is a N-Body application with 10 kernels, simulating the electromagnetic Coulomb force applied on particles. This force has a cut-off distance, meaning that particles separated by a larger distance have no interaction. The space is divided into bins of equal size and particles are sorted among these bins with a counting sort. This is the role of the first kernels. The last kernel in the sequence computes for each particle the force applied by particles within the distance of one bin. All kernels accesses particles within bins through indirect accesses. Partitioning these kernels, even by hand, is very complex. Figure 2 presents speed-ups compared to the best single device performance for 4 different strategies : Uniform (partitioning ratio of $1/4$ for all sub-kernels), Adaptive w/o Comm (cf. IV-A), Adaptive w/ Comm (cf. IV-B), Oracle. In the Oracle strategy, the kernels in the sequence are directly partitioned with the partitioning ratios found after convergence. For the SOTL application, there is no Oracle since the workload dynamically changes with iteration number and the solver never converges. For the 3 benchmarks, we observe the results of the Adaptive w/ Comm strategy are close to the optimal Oracle strategy. The small difference of performance obtained with these two strategies shows that the overhead of resolving a linear system at each iteration is negligible. The Adaptive w/o Comm strategy obtained poor performance for Jacobi and FDTD2D. Since this strategy only minimizes computation time, a slow down due to data transfers is observed. For 2SpMV the same speedup is obtained with both adaptive strategies since the transfer times induced by the partitioning minimizing the computation time is negligible. For the SOTL

application, the time taken to compute one iteration is mostly taken by the *force* kernel. To avoid penalizing transfers, other kernels from the sequence must be partitioned according to this kernel. This is achieved by the Adaptive w/ Comm strategy and it explains the increase of the speedup from 1.68 to 1.86 when using this strategy instead of Adaptive w/o Comm strategy. We can see on Figure 3 that the load balancing is nearly optimal with the Adaptive w/ Comm strategy since the 4 plots showing the time per iteration on each device are close to each other.

VI. Related Work

Recent works propose approaches to manage the execution of an OpenCL code written for a single GPU on a multi-device heterogeneous platform. In [5], the authors propose to model workload distribution problem as a mixed-integer non-linear programming minimizing the variance of execution times among GPUs. A performance model, built from training runs is required as input to the solver. MKMD [4] uses a two phased approach based on a performance prediction model built from profile data. It first performs a coarse-grain scheduling of kernels and then performs kernel partitioning to offload work-groups of selected kernels to idle devices. In contrast to these works, our approach does not rely on prior training or profiling information. In [1], the authors propose a dynamic load-balancing algorithm for a single kernel. Their approach respond to performance variability among devices. It is limited, nevertheless, to kernels whose relative performance for the small, initial chunks of work-groups may lead to a good prediction of performance for larger chunks. In [8], the authors adapted the OpenMP guided scheduling to partition OpenCL kernels. However this approach does not take into account the data transfers induced by the partitioning of multiple dependent kernels. FluidiCL [7] uses a dynamic work distribution scheme where sub-kernels on the CPU and work-groups on the GPU are executed in a coordinated fashion. Nevertheless, their approach cannot be easily generalized for any number of devices. Sakai et. al [9] propose a data decomposition method for multi-dimensional data that cannot be entirely stored in the GPU memory and aiming at accelerating a single-GPU code on a multi-GPU system. This method uses a sample run and is limited to kernels whose memory references are given as affine functions of the thread indices. Some other works target integrated CPU-GPU systems. There are no explicit communication in this architecture. [2] proposes E-ADITHE for improving performance and energy efficiency of iterative computations. E-ADITHE does not take irregular iterative computations into account. [6] presents LogFit, an adaptive partitioning strategy in the context of parallel loops in applications with irregular data accesses. FinePar [10] relies on fine-grain partitioning and uses a sophisticated performance modeling approach taking both architectural differences between the CPU and GPU and data irregularity in consideration.

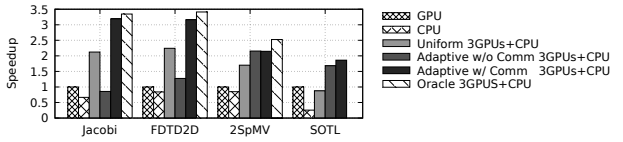


Fig. 2: Overall results obtained

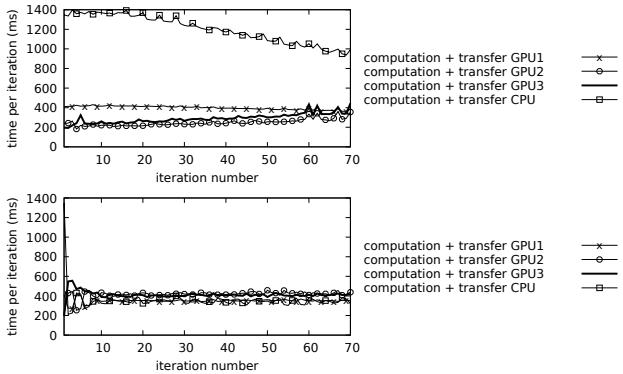


Fig. 3: Total time per iteration when *SOTL* is partitioned on 3 GPUs and 1 CPU using the Uniform strategy (top) and the Adaptive w/ Comm strategy (bottom)

VII. Conclusion

We have presented a novel automatic approach to dynamically partition a multi-kernel OpenCL code for an heterogeneous architecture. The method handles applications with dynamic load variations and takes into account both computation and communication time in order to balance the workload.

References

[1] M. Boyer, K. Skadron, S. Che, and N. Jayasena, "Load Balancing in a Changing World: Dealing with Heterogeneity and Performance Variability," in *Computing Frontiers Conf.*, 2013.

[2] E. M. Garzón, J. J. Moreno, and J. A. Martínez, "An approach to optimise the energy efficiency of iterative computation on integrated gpucpu systems," *The Journal of Supercomputing*, vol. 73, 2016.

[3] P. Huchant, M.-C. Counilh, and D. Barthou, "Automatic OpenCL Task Adaptation for Heterogeneous Architectures," in *Euro-Par*, ser. Euro-Par 2016: Parallel Processing, Grenoble, France, Aug. 2016, pp. 684 – 696.

[4] J. Lee, M. Samadi, and S. Mahlke, "Orchestrating Multiple Data-Parallel Kernels on Multiple Devices," in *Parallel Arch. and Compilation Techniques*. IEEE, 2015.

[5] C.-S. Lin, C.-W. Hsieh, H.-Y. Chang, and P.-A. Hsiung, "Efficient Workload Balancing on Heterogeneous GPUs using MixedInteger Non-Linear Programming," *Journal of Applied Research and Technology*, vol. 12, no. 6, pp. 1176 – 1186, 2014.

[6] A. Navarro, F. Corbera, A. Rodriguez, A. Vilches, and R. Asenjo, "Heterogeneous parallel_for Template for CPU–GPU Chips," *International Journal of Parallel Programming*, Jan 2018.

[7] P. Pandit and R. Govindarajan, "Fluidic Kernels: Cooperative Execution of OpenCL Programs on Multiple Heterogeneous Devices," in *Code Generation and Optimization*. ACM, 2014, pp. 273–283.

[8] B. Pérez, J. L. Bosque, and R. Beivide, "Simplifying Programming and Load Balancing of Data Parallel Applications on Heterogeneous Systems," in *Proceedings of the 9th Annual Workshop on GPGPU*, ser. GPGPU '16. New York, NY, USA: ACM, 2016, pp. 42–51.

[9] R. Sakai, F. Ino, and K. Hagihara, "Towards Automating Multi-dimensional Data Decomposition for Executing a Single-GPU Code on a Multi-GPU System," in *2016 Fourth International Symposium on Computing and Networking (CANDAR)*, Nov 2016, pp. 408–414.

[10] F. Zhang, B. Wu, J. Zhai, B. He, and W. Chen, "FinePar: Irregularity-Aware Fine-Grained Workload Partitioning on Integrated Architectures," in *Proceedings of the 2017 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*, 2017.