Ivan Grasso, Klaus Kofler, Biagio Cosenza, Thomas Fahringer

# Automatic problem size sensitive task partitioning on heterogeneous parallel systems



**Suggested Citation**
Grasso, I.; Kofler, K.; Cosenza, B.; Fahringer, T.: Automatic problem size sensitive task partitioning on heterogeneous parallel systems. - In: ACM SIGPLAN Notices. - ISSN: 0362-1340 (online), 1558-1160 (print). - 48 (2013), 8. - pp.281-282. - DOI: 10.1145/2517327.2442545. *(Postprint version is cited.)*

# Automatic Problem Size Sensitive Task Partitioning on Heterogeneous Parallel Systems

Ivan Grasso Klaus Kofler Biagio Cosenza Thomas Fahringer
Institute of Informatics, University of Innsbruck, Austria
{grasso, klaus, cosenza, tf}@dps.uibk.ac.at

## Abstract

In this paper we propose a novel approach which automatizes task partitioning in heterogeneous systems. Our framework is based on the Insieme Compiler and Runtime infrastructure [1]. The compiler translates a single-device OpenCL program into a multi-device OpenCL program. The runtime system then performs dynamic task partitioning based on an offline-generated prediction model. In order to derive the prediction model, we use a machine learning approach that incorporates static program features as well as dynamic, input sensitive features.
Our approach has been evaluated over a suite of 23 programs and achieves performance improvements compared to an execution of the benchmarks on a single CPU and a single GPU only.

***Categories and Subject Descriptors*** D.3.2 [*Programming Languages*]: Language Classifications; D .3.4 [ *Programming Languages*]: Processors; C.1.3 [*Processor Architectures*]: Other Architecture Styles

***General Terms*** Languages, Algorithms, Performance

***Keywords*** heterogeneous computing, compilers, GPU, task partitioning, code analysis, machine learning, runtime system

## 1. Introduction

The transition from homogeneous to heterogeneous architectures is challenging with respect to the efficient utilization of the hardware resources and the reuse of the software stack. This problem has drawn great interest from researchers and industry, leading to the proposal of several programming models including HMPP, OpenACC, CUDA and OpenCL [6]. OpenCL (Open Computing Language) is the first open standard for cross-platform parallel computing that supports a wide range of hardware through a low-level high performance abstraction layer.

As heterogeneous computing opens many new opportunities for developing parallel algorithms, our work is motivated by the additional challenges and complexity that it also introduces. One of the challenges is the distribution of tasks (i.e. task partitioning) among the available OpenCL devices in order to maximize the system performance. Task partitioning defines how the total workload is distributed among several computational resources. It is important to understand that the best-performing task partitioning changes with different applications, different (input) problem sizes, and different hardware configurations.

Another important aspect of heterogeneous computing is the difficulty of writing *multi-device* programs (i.e. a single program which can be executed on multiple devices concurrently).

In this paper we present an automatic, problem size sensitive method for task partitioning of OpenCL programs on heterogeneous systems. Our work is based on machine learning which effectively combines compile time analysis with runtime feature evaluation to predict the optimal task partitioning for every combination of program, problem size and hardware configuration.

## 2. Framework Overview

The proposed approach, based on the Insieme Compiler and Runtime infrastructure [1], is composed by two main phases: *training* and *deployment*.

The goal of the training phase is to build a task partitioning prediction model. To build the model, a set of OpenCL programs are provided to the system and translated by the code analyzer into the *Insieme Parallel Intermediate Representation* (INSPIRE). From this representation, the features of the program (*static program features*) are extracted and stored in a database. The intermediate representation of the program is then passed to the backend which generates multi-device OpenCL code. Once generated, the new program will be executed with various problem sizes and the available task partitionings. The obtained performance measurements, together with the problem size dependent features of the program (i.e. *runtime features*), are collected and added to the database. After these steps have been accomplished for all programs, a machine learning task partitioning model is generated using the features and the performance measurements stored in the database.

In the deployment phase a new OpenCL program is provided to the analyzer, the static features are extracted and the intermediate representation is passed to the backend which generates a multi-device OpenCL program. When the program is executed, the static code features along with the runtime features are provided to the previously trained model, which predicts the best task partitioning for the current program with the selected problem size. Finally, the runtime system executes the program on the given hardware using the predicted task partitioning.

### 2.1 Predicting the Optimal Partitioning

Our overall approach requires building a model using machine learning in order to predict a task partitioning $p$ from a vector of features that describes the essential characteristics of a program as well as the current problem size. The predicted task partitioning $p$ should be as close as possible to the best task partitioning in terms

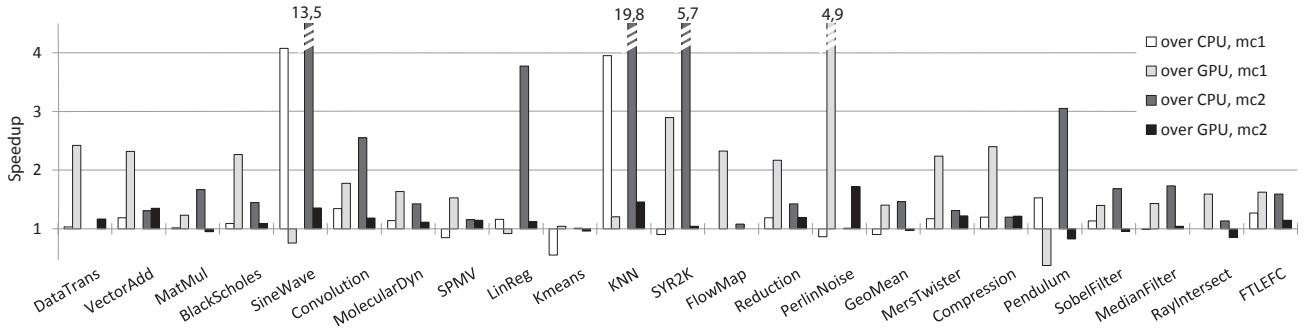


**Figure 1.** Speedup of our machine leaning guided task partitioning approach over execution on CPU/GPU only for different programs on two target architectures with different problem sizes.

of performance. $p$ is selected from a discretized partitioning space with a stepsize of $10\%$.

## 3. Evaluation

To evaluate the performance of our approach we used a selection of 23 programs drawn from OpenCL vendors' example codes, applications from our department or partner universities, and benchmark suites [2–4]. After processing the OpenCL input program with the Insieme source-to-source compiler, the Gnu Compiler Collection (GCC) version 4.6.3 was used to convert the resulting code to binary.

In order to examine the impact of problem sizes on task partitioning we executed each benchmark with varying problem sizes on two heterogeneous target platforms composed of three OpenCL devices: two GPUs and two multi-core CPUs in a dual-socket infrastructure. While both GPUs represent a separate device, the two CPUs are reported as a single OpenCL device. The first platform, *mc1*, consists of two AMD Opteron CPUs and two Ati Radeon HD 5870 GPUs, while the second, *mc2*, holds two Intel Xeon CPUs and two NVIDIA GeForce GTX 480 GPUs.

Each training pattern consists of the static features of a program, its runtime features for a certain problem size as well as the best task partitioning for the given program with the current input size. To ensure a fair comparison between different task partitionings, we measured the execution time of the kernels including the memory transfer overhead [5].

In Figure 1 we compare the performance of the task partitionings predicted by the Insieme framework based on a machine learning approach, with the performance delivered by the CPU/GPU-only strategy for each code and each target architecture individually. Of these two default strategies, in almost all test cases, the CPU-only strategy delivers a higher performance on *mc1*, while on *mc2* the GPU-only strategy usually performs better. This is a result of the weaker performance of the GPU in *mc1*. The VLIW architecture with a very wide instruction width and high branch miss penalty would require specific fine-tuning of each code to perform well [7]. However, none of our test cases was tuned for a specific device.

On average considering both target architectures, our approach delivers a significantly better performance than the two default strategies for most test cases. Our models are capable of representing the target architecture's characteristics in order to find performance efficient task partitionings and determine which device is to be favored on a specific target architecture.

## 4. Conclusion

In this paper we proposed a novel approach which can automatically distribute OpenCL programs on heterogeneous CPU-GPU systems. It consists of a source-to-source compiler, which translates a single-device OpenCL program into a multi-device OpenCL program and a runtime system which distributes the workload over all heterogeneous resources using a machine learning based, offline generated prediction model.

Our measurements demonstrate that the optimal task partitioning does depend on the program, the target architecture, as well as the problem size. To accommodate this observation, we use two classes of features: static program features, whose values can be extracted from the source code at compile time, and problem size dependent runtime features, whose values are collected during program execution.

To demonstrate the portability and the performance of the system, all tests were performed on two different target architectures showing that our approach outperforms the two default strategies that use only the CPU or only the GPU, respectively.

## Acknowledgment

This project was funded by the FWF Austrian Science Fund as part of project TRP 220-N23 "Automatic Portable Performance for Heterogeneous Multi-cores".

## References

[1] Insieme compiler and runtime infrastructure. - Distributed and Parallel Systems Group, University of Innsbruck. `http://insieme-compiler.org`, 2012.

[2] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron. Rodinia: A benchmark suite for heterogeneous computing. In *IISWC*, pages 44–54, 2009.

[3] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford, V. Tipparaju, and J. S. Vetter. The scalable heterogeneous computing (shoc) benchmark suite. In *GPGPU*, pages 63–74, 2010.

[4] S. Grauer-Gray, L. Xu, R. Searles, S. Ayalasomayajula, , and J. Cavazos. Auto-tuning a high-level language targeted to gpu codes. In *InPar*, 2012.

[5] C. Gregg and K. M. Hazelwood. Where is the data? why you cannot debate cpu vs. gpu performance without the answer. In *ISPASS*, pages 134–144, 2011.

[6] Khronos OpenCL Working Group. The OpenCL 1.2 specification. `http://www.khronos.org/opencl`, 2012.

[7] P. Thoman, K. Kofler, H. Studt, J. Thomson, and T. Fahringer. Automatic opencl device characterization: guiding optimized kernel design. In *Euro-Par*, pages 438–452, 2011.