A first step to performance prediction for heterogeneous processing on manycores

N. Benoit, S. Louise

Procedia Computer Science

Volume 51, 2015, Pages 2952–2956

ICCS 2015 International Conference On Computational Science

A First Step to Performance Prediction for Heterogeneous Processing on Manycores

Nicolas Benoit[1] and Stephane Louise[3]

[1] Bull-Serviware, Croissy Beaubourg, France
[2] CEA, LIST, PC172, 91191 Gif-sur-Yvette, France
stephane.louise@cea.fr

Abstract

The current trends in processor industry opens the way to next generations of microprocessors may count hundreds of independent cores that may differ in their functions and features. As an extensive knowledge of their internals cannot be a prerequisite to their programming and for the sake of portability, these forthcoming computers necessitate the compilation flow to evolve and cope with heterogeneity issues.

In this paper, we lay a first step toward a possible solution to this challenge by exploring the results of SPMD type of parallelism (as a first step) with heterogeneous compute kernels and predicting performance of the compilation results. We show some first experimental results with very good accuracy of the predicted performance with regard to real world measurements.

Keywords: Heterogeneous systems, Performance prediction, Compilation for parallel systems

1 Introduction

During the last decades, the performance growth of microprocessors has been continuously driven by the growth of silicium surface densities and of clock frequencies. Adding multiple independent execution cores is one of the proposed solutions [2]. It is simple and promising but requires a paradigm shift in the field of programming. Instead of writing a single monolithic task, programmers are now required to distribute collaborating tasks to multiple cores.

Current computing trends also suggest that heterogeneous multiprocessors are on their way to conquer a large share of the architectural landscape [2]. In order to take advantage of such designs, a compilation flow must be able to detect the affinity of code portions with the capabilities of the available Processing Elements (PEs, or cores). To this end, the process scheduling part of the compiler must be able to predict the performance resulting from their collaboration.

Therefore, when targeting heterogeneous systems, program characteristics must be checked for matching the target and additional specific information must be gathered. In this paper,

 Selection and peer-review under responsibility of the Scientific Programme Committee of ICCS 2015

doi:10.1016/j.procs.2015.05.493

we consider SPMD-threaded programs and their execution on architectures coupling multiple general-purpose cores to specialized cores. We also show that the proposed process works well to distribute heterogeneous compute kernels on homogeneous targets, as a side effect.

In this paper, we will present a proposition of execution model for performance prediction and how to conduct the evaluation in section 2, then experimental results on benchmark programs and on a simple architecture as first evaluation are presented in section 3, then we present a quick overview of related works in section 4.

2 Program mapping and Execution Model

2.1 Detection of Candidates to Specialization

In previous works we presented a generic characterization of the input program [5] that allows our tools to find program constructs that could be mapped to specialized PEs (SPs). Execution and communication costs are confronted to detailed information about the target architecture in order to build a summary for each class of PEs.

In addition, the selection process can rely on several criteria such as: An execution cost threshold above which the offloading cost of a part of the execution may be amortized, a computational intensity threshold above which extra communication costs due to the offloading may be considered negligible, or the availability on a distinct set of PEs of a category of computation (*e.g.* instructions such as Multiply-Accumulate, vector operations ...) or functionality (*e.g.* OS system calls).

2.2 Execution Model

In this paper we focus on data-parallel parts of applications, as pipelined parts could be handled in a simpler way *e.g.* with Stream Graph Modulo Scheduling technique [8]. Being data-parallel, the application can be split into N independent jobs, each operating on independent sets of data in a SPMD fashion. A section of the application requires to be offloaded to Specialized Processors (SPs).

With these hypotheses, the proposed execution model relies on two work queues Q_G and Q_S. The work queue Q_G, shared by all the available General-Purpose Processors (GPs), manages the general-purpose jobs. It is a priority queue where idling jobs are ordered according to their completion status. The work queue Q_S, shared by all the available Specialized Processors (SPs), manages the requests of specialized sections execution issued during the processing of general-purpose jobs. It is a FIFO queue.

In the synchronous case, a GP is blocked until the specialized job request issued for its current job is done. In the asynchronous case, a GP puts back its current job in Q_G and picks another one. The previous job will be made available to GPs as soon as its specialized job is finished. Jobs of the application are split into 5 sections:

- *GP prologue* is the first section to be executed after a job has been picked by a GP. It contains all the code that must precede the specialized section.
- *GP issue* contains the code necessary for the GP to issue a request to the SPs.
- *SP work* is the specialized section, the code to be executed by a SP.
- *GP work* contains some work that is independent from the specialized section that the GP can perform.
- *GP epilogue* section wraps the work that is dependent from the specialized section.

The execution course and global state of the machine can be described by a Petri net, as shown in Figure 1. On this figure, the Petri net is initialized with one pending job, one idling

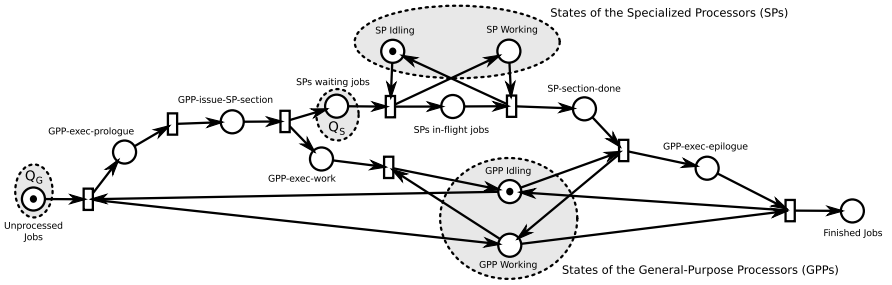

Figure 1: Petri net tracking the machine state in the asynchronous case.

GP and one idling SP.

Each GP picks a job from Q_G. During the processing, each GP issues a query to offload the specialized section, adding a job request to the queue of specialized works Q_S. SPs pick job requests from Q_S in a FIFO order.

As the SPs are shared between the GPs, the number of each must be adjusted to avoid contention. Then, four parameters can be devised to adapt the flow of the pipelining scheme set-up within this model: The number of GPs used, the number of SPs used, the number of partitions of the input dataset, the number of sub-partitions applied to the specialized section of each job. These four parameters constitute a configuration of execution.

2.3 Performance Prediction

To simulate the execution of the application and achieve performance prediction, each task is tagged with two values : a cost and a parallelism ratio. The cost corresponds to an estimation of the execution time of a phase when executed without any partitioning of the application's workset. It can be obtained by profiling or instrumentation, but requires the execution time to be stable and independent of the input dataset. This constraint is often met in data-parallel algorithms, so this is an important first step.

The parallelism ratio (referred to as P ratio in the remainder) of a task is meant in Amdahl's sense [1]. It establishes in which proportion the execution time is reduced when increasing the number of workset partitions with an infinite number of processors.

The *issue* task is distinguished from the *prologue* because its parallelism ratio may be different. Its purpose is to enable the capture of the communication costs between the GPs and the SPs. According to the nature of the coupling between the GPs and SPs, these costs can also be captured by the *SP work* task characteristics. In order to explore the behavior of the execution model proposed and predict performance trends, an event-driven simulator has been developed based on a Petri net model as seen in Figure 1 for the asynchronous case.

3 Experiments

Gomet [4, 5], our extension of GCC, supports code generation in the programming framework described above. Data-parallel loops detected in the input source code are rewritten to use the constructs of our programming framework and become *kernels*.

Currently, data-parallel loops that are transformed to kernels are only checked for exposing a computational cost superior to a constant threshold. In Section 2.2, we introduced an execution model for SPMD-threaded programs which relies on the drawing of two sets of processing elements: General-Purpose Processors (GPs) and Specialized Processors (SPs). This section is a first evaluation of the accuracy of the performance predictor developed to support this model.

The computer used for this evaluation is a quad-processor AMD Opteron 6172 with a total of 48 cores. It is clocked at 2.1 GHz with 64 GB of RAM. The compiler used is GCC 4.6.1 and the system runs a Linux 2.6.32 kernel on a SuSE Linux Enterprise Server 11. The Filterbank and FFT are configured to process a collection of signal slices independently. Table 1 summarizes the costs of the tasks as returned by the runtime characterization phase when processing 2048 signal slices.

name	*purpose*	*cost (ms)*	*P ratio*
prologue	initialization	1596	0.97
issue	specialized job request	23	0.73
GP work	FFT	83702	1.0
SP work	Filterbank	439767	1.0
epilogue	empty	3	0.76

Table 1: Filterbank and FFT characterization on a 48-core homogeneous architecture

Setting the number of GPs to twelve and the number of SPs to twenty-four, Figure 2 shows the evolution of the speed-up against a sequential execution as measured and as predicted by the simulator.

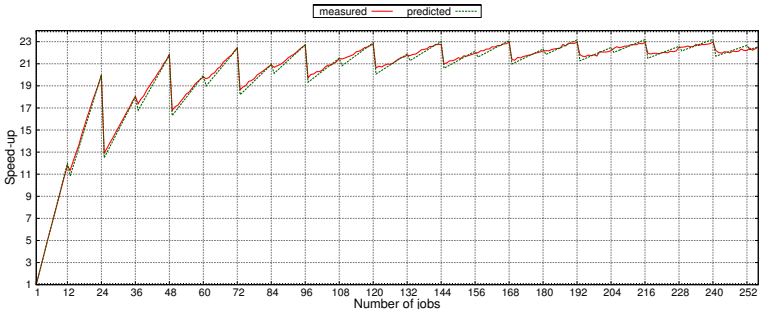

Figure 2: Measured and predicted speed-up on a homogeneous architecture with a varying number of jobs relative to an execution with one GP and one SP.

The simulator correctly captures the effect of using a larger number of work partitions: the performance grows as the balancing between GPs and SPs improves. Also, the prediction mean absolute error is kept very low at 1.59 %.

4 Related Works

There are already various proposals of execution models and programming frameworks for heterogeneous architectures [6, 7, 3]. Regarding strictly the execution model, the main studied issues are often the evaluation of the profitability of offloading computation to a specialized resource, and the relevant code transformations. To the contrary, given a heterogeneous con-

text, our model focuses on the configuration of the execution so that the sharing of multiple specialized resources is not a performance bottleneck.

On the topic of shared resource contention, the work in [9] aims at categorizing the behavior and cohabitation of applications when they compete for the same resources (cache memory, hardware prefetcher, etc.). Our work addresses the same issue, but at a coarser level of specialization. In addition, focusing on SPMD-threaded programs allows us to provide a finer analysis of the execution and enables performance prediction.

5 Conclusion and Outlooks

In this paper, we presented a first step to solve some important problems regarding compilation of applications on manycores, with heterogeneous processings. Our contribution is an automatic tool to map the parallelism onto parallel system with heterogeneous PE, and predict with a very good accuracy their real life performance (less than a few percents). With such results, a parallelizing compiler can map automatically the application on a target without the help of a programmer while reaching close to the best performance for the application on the system.

Future work will extend our results to other systems and apply our technique to parallelize applications.

References

[1] G.M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. *AFIPS Conference Proceedings*, 30(8):483–485, 1967.

[2] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick. The Landscape of Parallel Computing Research: A View from Berkeley. Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, Dec 2006.

[3] C. Augonnet, S. Thibault, R. Namyst, and P.-A. Wacrenier. StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures. *Concurrency and Computation: Practice and Experience, Special Issue: Euro-Par 2009*, 23:187–198, February 2011.

[4] N. Benoit and S. Louise. Extending GCC with a multi-grain parallelism adaptation framework for MPSoCs. In *Proceedings of the 2nd International Workshop on GCC Research Opportunities*, pages 20–33.

[5] N. Benoit and S. Louise. Kimble: a hierarchical intermediate representation for multi-grain parallelism. In Florent Bouchez, Sebastian Hack, and Eelco Visser, editors, *Proceedings of the Workshop on Intermediate Representations*, pages 21–28, 2011.

[6] G. F. Diamos and S. Yalamanchili. Harmony: an execution model and runtime for heterogeneous many core systems. In *HPDC '08: Proceedings of the 17th international symposium on High performance distributed computing*, pages 197–200, New York, NY, USA, 2008. ACM.

[7] J. Enmyren and C. W. Kessler. SkePU: a multi-backend skeleton programming library for multi-GPU systems. In *Proceedings of the fourth international workshop on High-level parallel programming and applications*, HLPP '10, pages 5–14. ACM, 2010.

[8] M. Kudlur and S. Mahlke. Orchestrating the execution of stream programs on multicore platforms. In *PLDI '08: Proceedings of the 2008 ACM SIGPLAN conference on Programming Language Design and Implementation*, pages 114–124. ACM, 2008.

[9] S. Zhuravlev, S. Blagodurov, and A. Fedorova. Addressing shared resource contention in multicore processors via scheduling. In *Proceedings of the fifteenth ASPLOS on Architectural support for programming languages and operating systems*, pages 129–142, New York, NY, USA, 2010. ACM.