Procedia Computer Science

Volume 51, 2015, Pages 2962–2966

ICCS 2015 International Conference On Computational Science

A Methodology for Profiling and Partitioning Stream Programs on Many-core Architectures

Małgorzata Michalska[1], Jani Boutellier[2], and Marco Mattavelli[1]

[1] École Polytechnique Fédérale de Lausanne, Switzerland
[2] University of Oulu, Finland

Abstract

Maximizing the data throughput is a very common implementation objective for several streaming applications. Such task is particularly challenging for implementations based on many-core and multi-core target platforms because, in general, it implies tackling several NP-complete combinatorial problems. Moreover, an efficient design space exploration requires an accurate evaluation on the basis of dataflow program execution profiling. The focus of the paper is on the methodology challenges for obtaining accurate profiling measures. Experimental results validate a many-core platform built by an array of Transport Triggered Architecture processors for exploring the partitioning search space based on the execution trace analysis.

Keywords: TTA, execution trace graph, dataflow, partitioning, profiling

1 Introduction

A natural method to handle applications characterized by high computational complexity is to implement parallel programs on high performance machines. Traditional approaches with concurrency evolving directly from sequential programs present several drawbacks. For instance, there is no guarantee that the resulting program would end in exactly the same way as the starting sequential program. The threads behaviour cannot be properly profiled nor simulated by appropriate test vectors and the risk of unpredicted race conditions is always present. Furthermore, threads depend on the platform making any changes a troublesome interference in the design [11]. Programming paradigm used by the dataflow computational model presents an interesting alternative. Apart from some attractive features such as platform independence and explicit exposition of parallelism it provides several design space analysis opportunities [9].

In spite of many features that make dataflow programs efficient and analyzable, the designer usually has to deal with the problem of transferring those features onto a specific architecture. At this stage several narrowing factors occur that are relative to the number of available units, constraints on power consumptions or architectural obstacles to track the application execution. There are several ways of dataflow design evaluation among which the throughput is particularly important, as it contributes to the improvement of other design factors [8]. One method of

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

doi:10.1016/j.procs.2015.05.498

increasing the throughput is an optimal partitioning of computational kernels on the set of processing units available within the platform. It requires an appropriate definition of the execution model of a network partitioned on a target architecture and the optimization function relying on it. Describing with sufficient accuracy the various steps occurring in the computation platform is very challenging because it has to rely on the profiling of several computational properties of the target architecture that involve computations and data access measures.

This paper describes a methodology based on profiling a single execution of a dataflow program on a target platform and capable of simulating with a very high level of accuracy the performance obtainable by executions corresponding to different partitioning configuration. Its main purpose is to make preliminary experiments to validate a new platform to explore profiled based partitioning.

2 Notation

Dataflow computation models can be defined as (hierarchical) networks of computational kernels, called *actors*, connected by directed, lossless, order preserving point-to-point communication channels, called *buffers* and with the algorithmic part specified inside the so-called *actions*. We consider a very general dataflow Model of Computation (MoC), namely, the Dataflow Process Network (DPN) with firings. This model enables the representation of a DPN program as a sequence of discreet action executions called *firings*. A description of DPN actors is directly captured in the CAL actor language [6]. Execution of a dataflow program can be characterized with different types of dependencies [5] that establish the precedence orders. The set of firings can be represented with a directed and acyclic graph called Execution Trace Graph (further referenced as ETG), where each single firing is represented by a node and each dependence by a directed arc.

Expressing the partitioning problem in the theoretic notation of scheduling problems, its most generic version can be described as $P|prec, groups|C_{max}$, which is the scheduling of a set of tasks on identical processors under arbitrary constraints such that the total execution time is minimized [7]. A further assumption implies that for tasks assigned to the same processing unit only one can be executed at one time. However, although we identify the tasks with firings, as multiple firings constitute an algorithmic part of one actor, we cannot partition them freely on different processing units. This constraint strongly narrows the partitioning search space.

3 Related work

Simulation, evaluation and finally optimal mapping of an application onto target architecture requires obtaining some profiling metrics in a form of cycle-accurate platform execution results. However, retrieving such values in a reliable way is an extremely troublesome task. Real-world hardware counters usually do not live up to the ideal ones. The undesired deviation from the expected result is usually due to non-determinism (different values for identical runs) and overcount (counting some instructions multiple times) [14]. There are various external sources of these variations including program layout, measurement overhead, multi-processor variations and uncertainty of compilers that may leave many unexplored corner cases.

Independently from the accuracy and reliability of profiling, the partitioning problem itself has been proven to be NP-complete, even when only 2 processing units are considered [3]. Several heuristic approaches defining different optimization functions exist in the literature. Most often they aim at cuts cost minimization [10] or workload balancing [12]. Apart from

greedy approaches, there are some more sophisticated methods based on integer programming or genetic algorithms [13], to list just a few.

Separating the problem of accurate profiling and efficient partitioning requires defining a high-level design flow to drive the exploration. A dataflow design flow is usually composed of several phases: specification (entirely agnostic of the target platform), design space exploration (functional validation by a set of behavioural simulations) and implementation (transferring the design onto a target architecture) that all together build a design cycle [4]. Accurate simulation of the program execution on the analysed platform that is a contribution of this work is based on the Trace Processor tool (further referenced as TP) developed as part of the TURNUS co-exploration environment [2] that is compatible with such a design flow.

4 TTA Architecture and Profiling

Transport Triggered Architecture (further referenced as TTA) is a processor architecture where program description contains only the operands transfers between the computational resources, what makes this embedded platform easily measurable and highly deterministic [1]. As far as we are concerned, this is the only multiprocessor platform with no significant inter-processor communication penalty. Hence, it is ideal for validating the partitioning algorithm at the first stage, where communication cost is not yet taken into account. For the experiments presented in this paper, profiling for execution time takes place on the cycle-accurate TTA simulator [15], where the application is executed one actor at a time, on a single processor core. This execution uses an input data for each input buffer of the actor that is currently in execution. In this setting, each actor is executed in isolation from other actors, i.e. it executes in optimal conditions.

For the purpose of profiling, the processor core of the simulator is equipped with a special *timestamp* hardware operation that creates a record to an external file every time the operation is executed. This hardware operation is minimally intrusive, as it executes in one clock-cycle on a processor that is capable of executing multiple operations every clock-cycle. The timestamp operations are placed right before and after the invocation of each action's computations and hence provide accurate information on each action's computation time. Furthermore, timestamps are placed to the entry and exit points of each actor's state machine structure, which allows to measure the time spent in action scheduling.

5 Methodology

A TP-based simulation processes the (platform independent) *ETG* generated for a specific input stimulus and assigns the nodes with weights profiled on the TTA platform. A weight consists of two components: computational load, profiled for each action separately and scheduling cost, profiled globally for an actor and distributed among actions according to their frequency of execution. Due to the TTA properties, we may neglect the communication cost and assume that the architecture provides us with n identical units with full connectivity and a Round Robin scheduling policy.

5.1 Experimental setup

All tests have been performed with an MPEG SP decoder design, consisting of 17 actors and adapted to the requirements of TTA platform. The execution time in clock-cycles simulated

by the TP has been compared with the values produced by the cycle-accurate multicore TTA simulator for different numbers of units. As starting points, we chose the mapping configurations generated by the partitioning algorithm that we are currently developing.

5.2 Results

We applied simple random modifications (single moves, swaps and multiple moves) and observed how they contribute to the change of execution time for the TP simulation and whether they affect the real platform execution in the same way. We present the results obtained for 4 and 5 units, as for these numbers a sufficient level of variation has been observed. Figure 1 presents normalized values of clock-cycles obtained for the simulation and the platform execution for the initial configurations. Table 1 comprises the experiments performed on 4 units. For each operation (first column), we calculated how the execution time was affected (in %) for the TTA (second column) and TP simulation (third column). A negative value corresponds to a decrease of the execution time so an improvement of throughput at the same time. The opposite applies to a positive value. Table 2 presents similar values obtained for modifications of partitioning configuration spanned on 5 units.

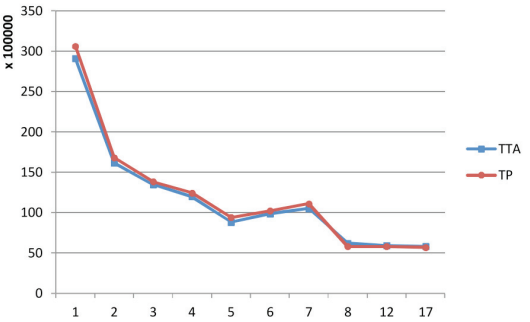

Figure 1: Clock-cycles comparison between TTA and TP for various numbers of units.

Operation	TTA[%]	TP[%]
single move (I)	1	-1
single move (II)	8	7
single move (III)	-1	-1
swap (I)	-4	-3
swap (II)	26	31
swap (III)	-1	-3
multiple move (I)	8	9
multiple move (II)	-6	-6
multiple move (III)	-9	-12

Table 1: Execution time changes (4 units).

Operation	TTA[%]	TP[%]
single move (IV)	0	0
single move (V)	6	1
single move (VI)	2	1
swap (IV)	6	2
swap (V)	7	1
swap (VI)	1	-1
multiple move (IV)	25	9
multiple move (V)	74	74
multiple move (VI)	25	15

Table 2: Execution time changes (5 units).

Taking into account all analysed cases it can be noticed that the TP corresponds very well to the TTA simulator. The average difference between number of cycles obtained for both is only around 4%. This difference may be be partially due to the lower efficiency of the profiling-enabling code and the simplified distribution of the scheduling cost among action firings. As for the random modifications, the evaluation of each move was very similar in most cases.

6 Conclusion

The experiments fully validated the TTA platform for the purpose of partitioning search space exploration. The Trace Processor tool that we have provided enabled multiple emulation of the program execution on the platform, using the results of a single profiling. Described methodology is an initial step on the way to find a close-to-optimal partitioning heuristic. The most important future work should focus on the extension of the model in order to make the methodology valid for other emerging many-core and multi-core platforms.

ACKNOWLEDGEMENT

This work is supported by the Fonds National Suisse pour la Recherche Scientifique, under grant 200021.129960 and grant 200021.138214.

References

[1] TTA-Based Co-design Environment. `http://http://tce.cs.tut.fi/tta.html`, Last checked: December 2014.

[2] TURNUS. `http://github.com/turnus`, Last checked: March 2015.

[3] L. Benini, M. Lombardi, M. Milano, and M. Ruggiero. Optimal resource allocation and scheduling for the CELL BE platform. *Annals of Operations Research*, 184:51 – 77, 2011.

[4] E. Bezati, R. Thavot, G. Roquier, and M. Mattavelli. High-level dataflow design of signal processing systems for reconfigurable and multicore heterogeneous platforms. *Journal Of Real-Time Image Processing*, 9(1):251–262, 2014.

[5] S. Casale-Brunet, A. Elguindy, E. Bezati, R. Thavot, G. Roquier, M. Mattavelli, and J. W. Janneck. Methods to explore design space for MPEG RVC codec specifications. *Signal Processing: Image Communication*, 28(10):1278 – 1294, 2013.

[6] J. Eker and J. Janneck. *CAL Language Report: Specification of the CAL Actor Language*. University of California-Berkeley, December 2003.

[7] A. Elguindy, M. Matavelli, S. Hendseth, and E. Amaldi. Multiprocessor scheduling of grouped task graphs. *internal report*.

[8] M. Hirzel, R. Soulé, S. Schneider, B. Gedik, and R. Grimm. A catalog of stream processing optimizations. *ACM Computing Surveys*, 46(4), 2014.

[9] G. Kahn. The semantics of simple language for parallel programming. *IFIP Congress*, 1974.

[10] B.W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49:291 – 307, 1970.

[11] E. A. Lee. The problem with threads. *Computer*, 39:33–42, May 2006.

[12] S. Miguet and J.-M. Pierson. Heuristics for 1d rectilinear partitioning as a low cost and high quality answer to dynamic load balancing. *High-Performance Computing and Networking Lecture Notes in Computer Science*, 1225:550 – 564, 1997.

[13] M. Pelcat, J. Piat, M. Wipliez, S. Aridhi, and J-F. Nezan. An open framework for rapid prototyping of signal processing applications. *EURASIP Journal on Embedded Systems*, 2009.

[14] V. Weaver, D.Terpstra, and S. Moore. Non-determinism and overcount on modern hardware performance counter implementations. *IEEE International Symposium on Performance Analysis of Systems and Software, Austin*, 2013.

[15] H. Yviquel, A. Sanchez, P. Jääskeläinen, J. Takala, M. Raulet, and E. Casseau. Embedded multi-core systems dedicated to dynamic dataflow programs. *Journal of Signal Processing Systems*, pages 1–16, 2014.