Efficient scheduling policies for dynamic dataflow programs executed on multi-core

Małgorzata Michalska[1], Nicolas Zufferey[2], Jani Boutellier[3], Endri Bezati[1], and Marco Mattavelli[1]

[1] EPFL STI-SCI-MM, École Polytechnique Federale de Lausanne, Switzerland
[2] Geneva School of Economics and Management, University of Geneva, Switzerland
[3] Department of Computer Science and Engineering, University of Oulu, Finland

Abstract. An important challenge for dataflow program implementations on multi-core platforms is the partitioning and scheduling providing the best possible throughput when satisfying multiple objective functions. Not only it has been proven that these problems are NP-complete, but also the quality of any heuristic approach is strongly affected by other factors, such as: buffer dimensioning, influence of an established partitioning configuration and scheduling strategy on each other, uncertainties of a compiler affecting the profiling information. This paper focuses on the adaptation of some alternative scheduling policies to the dataflow domain and observation of their properties and behavior when applied to different partitioning configurations. It investigates the impact of the scheduling on the overall execution time, and verifies which policies could further drive the metaheuristic-based search of a close-to-optimal partitioning configuration.

1 Introduction

In the emerging field of massively parallel platforms, it is highly requested to develop efficient implementations exploiting the available concurrency. Dataflow programs, characterized by some interesting properties providing a natural way of dealing with parallelism, seem to posses the necessary features to successfully handle this requirement. For the purpose of mapping a dataflow program on a target architecture, the program should be treated as a set of non-decomposable components (connected to each other by a set of buffers) that can be freely placed. This implies that dataflow programs are portable to different architectures with making only two decisions: (1) assign the components to the processing units; (2) find the execution order (sequencing) inside every unit. For static dataflow programs the problem of partitioning and scheduling has been studied very well and the whole class of compile time algorithms is proven valid [14]. In the case of dynamic dataflow programs the problem becomes much complicated, because it requires creating a reliable model of execution that could sufficiently cover and capture the entire application behavior, which depends on the input data.

The exploration of the design space should take into consideration three dimensions: partitioning of dataflow components, scheduling inside each partition and dimensioning of the buffers that connect the components with each other [8]. Exploring one dimension requires making an assumption or at least narrowing the setup of other two dimensions. On the other hand, such assumptions usually strongly influence the outcome of the exploration. For instance, in the case of scheduling exploration, the quality of applied partitioning and buffer dimensioning determines the size of the space of possible admissible scheduling configurations. Still it can be stated that if the whole dynamic behavior of an application is properly captured for a given input sequence and the non-blocking buffer dimensioning is applied, the scheduling problem for dataflow programs is always feasible, in comparison to some other Models of Computation (*MoCs*) [2]. Under the circumstances the challenge becomes to find such a scheduling configuration that optimizes the desired objective. For the case of signal processing systems, among various possible objective functions, the most natural is the maximization of the data throughput, since it contributes to the improvement of other objective functions [12]. Providing such an optimal solution to the partitioning-scheduling problem has been, however, proven to be NP-complete even if only a platform with two processors is considered [25].

After discussing the related work in Section 2, the contribution of this paper starts in Section 3 from a proper formulation of the partitioning and scheduling problem, specifically in the dataflow domain. Indeed, to the best of our knowledge, such a formulation is still missing in the dataflow related literature. Furthermore, the Section 4 presents the methodology of experiments which involves modeling of the program execution, target architecture, simulation an verification of some different scheduling policies described in Section 5. The main objective is the analysis of these policies and their performance potential for different partitioning configurations. Section 6 contains the results of simulation supported with experiments conducted on a real platform. Finally, the results, observations, advantages and drawbacks of the applied methodology clarify a direction of future work discussed in Section 7.

2 Related work

Among several existing dataflow computation models, a dataflow program is in principle structured as a network of communicating computational kernels, called *actors*. They are in turn connected by directed, lossless, order preserving point-to-point communication channels (called *buffers*), and data exchanges are only permitted by sending data packets (called *tokens*) over those channels. This model is presented in Fig. 1. As a result, the flow of data between actors in such a network is fully explicit. The most general dataflow *MoC* is known in literature as "Dataflow Process Network (*DPN*) with firings" [15]. A *DPN* evolves as a sequence of discrete steps by executing actors firings (called *actions*) that may consume and/or produce a finite number of tokens and modify the internal actor state. At each step, according to the current actor internal state, only one action

can be executed. The processing part of actors is encapsulated in the atomic firing completely abstracting from time.

The problem of partitioning and scheduling of parallel programs in general has been widely described in literature in numerous variants [22]. In the dataflow domain, in particular, the programs are usually treated as graphs that need to be optimally partitioned [29]. According to the commonly used terminology, the partitioning can be defined as a mapping of an application in the spatial domain (binding), whereas scheduling takes place in the temporal domain (sequencing) [24]. It is also usually emphasized that the partitioning is performed at compile time, whereas scheduling occurs at run-time and is subject to the satisfaction of firing rules, as well as to the scheduling policy for the sequential execution of actors inside each processor [9]. Although the partitioning and scheduling problems seem to rely and impact each other, much more attention has been paid so far to the partitioning problem. Several experiments lead to consider finding a solution to the partitioning problem dominant over the scheduling problem [2, 7].

Since some dataflow models can be very general and therefore difficult to schedule efficiently, an interesting idea comes along with the concept of a flow-shop scheduling [1]. The asynchronous dataflow models can be, in some cases, transformed into simpler synchronous ones, where the partitioning and scheduling can be applied directly to the actions. After the partitioning stage (which is an assignment of all actions to the processing units), the scheduling is performed first in the offline phase (schedules are computed at compile time), and then in the run-time phase where a dispatching mechanism selects a schedule for data processing [3].

Another approach for simplifying the scheduling problem is to reduce the complexity of the network and control the desired level of granularity. This can be achieved by actor merging, which can be treated as a special transformation performed on the sets of actors [13]. Recent research shows that actor merging is possible even in the case of applications with data dependent behavior and in the end can act quasi-statically [4]. This, however, does not solve the scheduling problem entirely, since even for a set of merged actors, if multiple merged actors are partitioned on one processor, a scheduling approach needs to be defined.

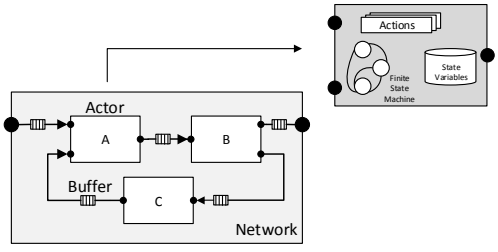

Fig. 1. Construction of a dataflow network and actor.

3 Problem formulation

The problem formulation described here should: cover the dataflow *MoC* in a very generic way, and avoid any limitations of the allowed level of dynamism in the application. It is used as a starting point for any experiment on the partitioning and scheduling. Following the production field terminology [17], a goal is to find an assignment of n jobs (understood as *action firings*) to m parallel machines (understood as *processing units*) so that the overall makespan (completion time of the last performed job among all processing units) is minimized. Assuming the processing units with no parallel execution, only one job (one action) can be executed at a time on each machine. When all jobs are assigned to the machines, the next decision is about their order of executions within each machine, as it is restricted that each job may only consists of one and exactly one stage.

Each job j has an associated processing time (or weight) p_j and a group (or actor) g_j. There are k possible groups, and each one can be divided into subgroups where all jobs have the same processing time. This division can be easily identified with actors and associated firings, which can be different executions of the same action. Between some pairs $\{j, j'\}$ of incompatible jobs (i.e., with $g_j \neq g_{j'}$) is associated a communication time $w_{jj'}$. The communication time is subject to a fixed quantity $q_{jj'}$ of information (or the number of tokens) that needs to be transferred. The size of this data is fixed for any subgroup (i.e., an action always produces/consumes the same amount of data). Due to the structure of dataflow programs, the following constraints need to be satisfied:

- **Group constraint.** All jobs belonging to the same group have to be processed on the same machine (an actor must be entirely assigned to one processing unit). A fixed relative order is decided within each group (it can be assumed that this order is established based on the program's input data).
- **Precedence constraint** (j, j') means that a job j (plus the associated communication time) must be completed before job j' is allowed to start.
- **Setup constraint.** It requires that for each existing connection (j, j') involving jobs from different groups, a setup (or communication) time $w_{jj'}$ occurs. More precisely, let C_j (resp. B_j) be the completion (resp. starting) time of job j. Then, $B_{j'} \geq C_j + w_{jj'}$.
- **Communication channel capacity constraint.** The size of a communication channel (buffer) the information (tokens) is being transmitted through, is bounded by B. That is, the sum of the $q_{jj'}$'s assigned to this buffer cannot exceed B. If it occurs, it might affect the overall performance.

The range of values for the p_j's and the $w_{jj'}$'s fully depends on the targeted architecture. In homogeneous platforms, p_j is constant no matter how a group (actor) is actually partitioned. In heterogeneous platforms, this value can vary according to the processor family the processing unit belongs to (i.e., software or hardware). Let m_j be the machine assigned to job j. Then $w_{jj'}$ is a product of two elements: the number of tokens $q_{jj'}$ and the variable time $c_{jj'}(m_j, m_{j'})$ needed to transfer a single unit of information from m_j to $m_{j'}$. For two given

jobs j and j', the largest $c_{jj'}$ can be significantly larger than the smallest $c_{jj'}$. In theory, every connection (j, j') can have as many different $c_{jj'}$'s as the number of different possible assignments to the machines, but in practice this number can be usually reduced to few different values, depending on the internal structure of the target platform (i.e., multiple NUMA nodes) [18].

4 Methodology of experiments

The goal of the methodology is to deal with the most general dataflow *MoC*, *DPN*, which is considered to be fully dynamic and capable of covering all classes of signal processing applications, such as audio/video codecs or packet switching in communication networks. In order to provide a framework for analysis and simulation of such dynamic applications, several models need to be introduced. A starting point is in the execution model of the application that is going to be partitioned and scheduled, and the model of the target architecture (set of machines). The next step is the profiling of an application on a target architecture in order to provide the model with weights assigned to every job. Next, the results of the profiling are exploited by the simulation tool in order to calculate the makespan for various partitioning and scheduling configurations. Finally, as in [21], the simulated results are verified and compared with the actual execution times obtained on the platform for a given set of configurations.

4.1 Program execution modeling

A representation of the dataflow program that captures its entire behavior, as extensively studied to solve other optimization problems of dynamic dataflow implementations [8], can be built by generating a directed, acyclic graph G, called *Execution Trace Graph (ETG)*, which has to consider a sufficiently large and statistically meaningful set of input stimuli in order to cover the whole span of dynamic behavior. The execution of a *DPN* program with firings can be represented as a collection of action executions, called *firings*, which are characterized by intrinsic dependencies. The dependencies are either due to the data that is exchanged over communication channels or to the internal properties, such as *Finite State Machine* or *State Variable*. In the first case, the firings (jobs) belong to different actors (groups) and the setup constraint occurs. In the second case, the firings belong to the same actor and the dependency contributes to the precedence constraint.

4.2 Target platform

The platform used in this work is built by an array of *Transport Triggered Architecture* processors (Fig. 2), further referenced as *TTA*. It resembles the *Very Long Instruction Word* (VLIW) architecture with the internal datapaths of the processors exposed in the instruction set. The program description consists only

of the operand transfers between the computational resources. A *TTA* processor is made of functional units connected by input and output sockets to an interconnection network consisting of buses [31]. Among several strengths of the *TTA* architecture, the property of highest importance for the sake of this work is a simple instruction memory without caches [11, 28]. As far as we are concerned, this is also the only multiprocessor platform with no significant interprocessor communication penalty. Thus, it allows a validation of an execution - and architecture model with regards to different partitioning and scheduling configurations, before the methodology is extended by a figure of merit for the possibly complex communication time.

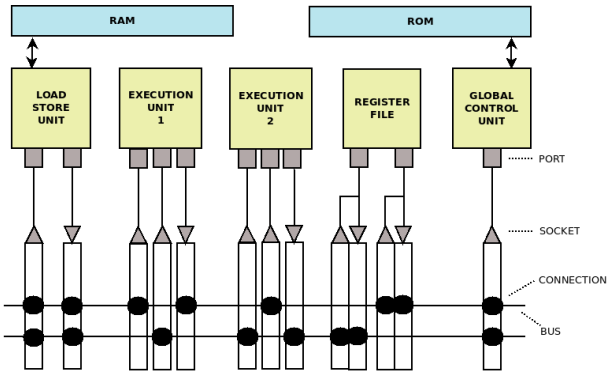

Fig. 2. Transport Triggered Architecture model.

The choice of the *TTA* as a target platform has been also dictated by the quest for the measurable and deterministic processing time of an application with possibly negligible overheads. In fact, the applied profiling methodology operates on actors executed in isolation, that is, one actor at a time on a single processor core. Thanks to that, it is possible to apply the profiling only once and explore its results in various configurations. It is a precious property of the *TTA* architecture comparing to profiling of other platforms, where the results of profiling usually depend on the partitioning configuration and may turn to be invalid when other configurations are approached [18, 26].

The profiling information is obtained using a minimally intrusive *timestamp* hardware operation taking place on the cycle-accurate TTA simulator [30]. The location of *timestamp* calls allows to measure the execution time in clock-cycles for every action inside the application as well as the overall time spent inside every actor outside the actual algorithmic parts (actions). This additional time can be identified with the internal scheduling overhead of an actor. Such a profiling seems to be a unique opportunity comparing to other platforms (i.e., NUMA architectures), where especially the communication time profiling is a highly troublesome process [18].

4.3 Performance simulation

A simulation tool developed as a part of the TURNUS co-design framework [6] is used to simulate the performance for different partitioning and scheduling configurations. It is able to compute in a deterministic way the makespan (execution time) for any given set of partitioning, scheduling and buffer dimensioning configurations, with the input of *ETG*, the p_j's and the $w_{jj'}$'s. The simulation tool considers the constraints specified in the problem formulation, monitors the events occurring on every processor in parallel, and throughout the execution it follows the model of behavior defined for *DPN* actors. In the situation when multiple actors could be executed at one time, it makes a choice basing on the specified internal scheduling policy. The simulation finalizes a tool chain used for the experiments, depicted in Fig. 3. Our previous experiments have proven that the simulation tool can be effectively and reliably used to simulate the performance of an application running on the *TTA* platform exploiting the results of a single profiling. Different partitioning configurations can be simulated with the maximal discrepancy between the simulated and real execution time of less than 5% [16].

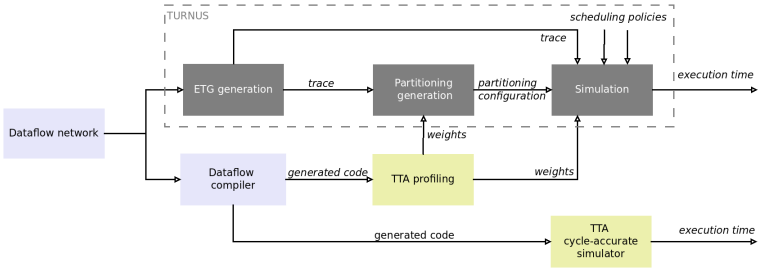

Fig. 3. Methodology of experiments: toolchain.

4.4 Analyzed application

All experiments have been performed using an MPEG4 SP decoder network, which is an implementation of a full MPEG-4 4:2:0 Simple Profile decoder standard written in CAL Actor Language [10]. The main functional blocks include a parser, a reconstruction block, a 2-D inverse discrete cosine transform (IDCT) block and a motion compensator. These functional units are hierarchical compositions of actors in themselves. The decoding starts from the parser (the most complicated actor in the network consisting of 71 actions) which extracts data from the incoming bitstream, through reconstruction blocks exploiting the correlation of pixels up to the motion compensator performing a selective adding of blocks. The whole network is presented in Fig.4.

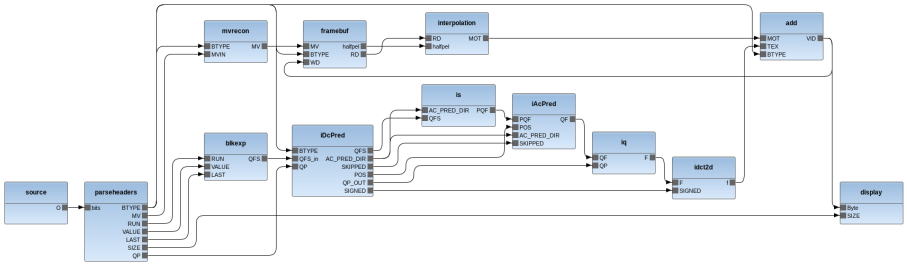

Fig. 4. MPEG4 SP decoder network

5 Scheduling policies

This work validates six different scheduling policies for actors partitioned on one core. The first three are direct implementations of existing techniques described in literature and widely used in systems of multiple types:

- **Non Preemptive** (*NP*): one actor is executed as long as as the firing conditions are satisfied, that is, it has the necessary input tokens and available space in outgoing buffers. It can be considered analogous to the *FCFS* scheduling, known also as Run-to-Completion [20]. The scheduler moves to the execution of the next actor on the list only if its firing conditions are not satisfied any more. The expression "preemptiveness" refers here to the change of the target actor after a successful firing and not to the interruption of a single task, which is, by nature, not allowed in dataflow programs.
- **Round Robin** (*RR*): after a successful firing of an actor, the scheduler moves to another one and verifies its firing conditions. It is not allowed to execute an actor multiple times in a row if there are other ones executable at the same time. This policy follows directly the standard *RR* procedure used in operating systems and described in [20].
- **NP/RR swapped** (*NP/RR*): it is similar to the concept of Round Robin with credits scheduling, where each task (actor, in this case) can receive a different number of "cells" for execution in each round [19]. In this case the choice of cells number is binary: either equal to one or the number determined by the *NP* policy. The choice is made basing on the criticality of an actor, which is represented as a percentage of its executions belonging to the *critical path* (*CC*). *CC*, defined as the longest time-weighted sequence of events from the start of the program to its termination, is evaluated using multiple algorithms as described in [8].

The other three policies are the extensions of these strategies with an introduction of different types of priorities (priority scheduling [20]). Unlike the existing approaches they exploit the information obtained at the level of action firings, not actors (i.e., jobs, not groups). As a result, although only the actors can be chosen by the scheduler, the system of priorities changes from firing to

firing throughout the execution. Extracting the information at this level is performed with the simulation tool operating within the TURNUS framework [6].

- **Critical Non Preemptive** (*CNP*): as long as the next firing of an actor is in *CC*, it is executed on a *NP* basis. For the non-critical executions, a *RR* approach is applied instead. It is a similar strategy to the *NP/RR*, but the priority is resolved independently for each action firing. In this case only the actual critical firings are given the priority, not actors as such.
- **Critical Outgoings Workload** (*COW*): priority is assigned to actors according to different properties. The highest priority goes to the actor whose next firing is critical. If multiple actors await to execute a critical firing, the next level of priority is given to the one, where the firing has outgoing dependencies in other partitions. If the decision cannot be made basing on these two criteria, the heaviest firing is chosen.
- **Earliest Critical Outgoings** (*ECO*): priority is assigned to the actor where the next firing occurs the earliest in *CC*, or if no critical firing is currently available, to a firing with the highest number of outgoing dependencies in other partitions. Non-resolved cases are handled on a *RR* basis.

6 Experimental results

In order to explore the design space in the dimension of scheduling, a fixed setup of partitioning and buffer dimensioning must be specified. In all experiments, the two sets of partitioning configurations spanned on up to 8 processors have been compared. The first set contained configurations where the overall workload of each partition is balanced, whereas the second one was created out of some random configurations. The idea behind that was to verify whether a certain tendency in the performance for different scheduling policies occurs independently from the quality of partitioning. As for the buffer dimensioning, in order to minimize its influence on the results, we would ideally aim at considering infinite buffer sizes. From practical purposes, as experimentally verified, a buffer size of 8192 is already a good approximation of an infinite buffer, because blocking at the outputs is not likely to happen. This value has been used for profiling, platform execution and performance simulation.

The first part of the analysis was the execution time simulated for the *NP* strategy, which is originally used by the *TTA* backend of ORCC [27]. The accuracy obtained for the simulation tool was very high, for instance, for the random set of partitioning configurations the difference between the *TTA* platform execution and emulated results was less than 1.8%. This makes the accuracy even higher comparing to our previous work [16]. This improvement might be due to the more convenient buffer size (8192 vs 512 used previously) and using of a longer input sequence (30 frames vs 5 frames used previously). Secondly, for each partitioning configuration, the simulation tool estimated the execution times for 6 different scheduling policies and calculated the speed-ups versus the mono-core execution. The results for the balanced (resp. random) partitioning configurations are presented in Table 1 (resp. 2).

Table 1. Estimated speed-ups: balanced partitioning configurations

No. of units	*NP*	*RR*	*NP/RR*	*CNP*	*COW*	*ECO*
1	1.00	1.00	1.00	1.00	1.00	1.00
2	1.78	1.99	1.79	1.70	1.89	1.99
3	2.27	2.84	2.36	2.31	2.30	2.79
4	2.72	3.57	2.75	2.68	3.28	3.46
5	3.14	4.20	3.29	3.62	3.85	4.14
6	4.41	4.67	4.43	4.67	4.72	4.68
7	5.04	5.12	4.99	5.14	5.10	5.12
8	5.41	5.46	5.41	5.47	5.49	5.46

Table 2. Estimated speed-ups: random partitioning configurations

No. of units	*NP*	*RR*	*NP/RR*	*CNP*	*COW*	*ECO*
1	1.00	1.00	1.00	1.00	1.00	1.00
2	1.61	1.64	1.61	1.54	1.59	1.62
3	2.30	2.48	2.31	2.19	2.47	2.48
4	2.59	2.97	2.68	2.45	2.73	2.97
5	2.84	3.21	2.87	3.03	2.97	3.21
6	2.73	2.86	2.72	2.82	2.67	2.87
7	2.83	2.98	2.83	2.85	2.98	2.98
8	4.47	5.01	4.46	4.70	4.63	5.02

It can be clearly observed that some policies tend to perform much better than the others for almost *any* set of configuration. For example, *RR* outperforms *NP* by more than 10% on average, and up to even 25%. The strategies relying on changing the actor after every execution (*RR, COW, ECO*) are also in general more efficient than *NP* and its derivatives. Surprisingly, *CNP* does not perform really well. It can be due to the fact that, as for the scheduling policy, when the critical firings were given a priority to fire, the critical path might have been modified by the concurrent decision of the scheduler. At the higher processor count all policies start to perform very similarly. It can be due to the fact that as the average number of actors in one processor decreases, the possible choice of scheduler becomes limited and less sensitive to the strategy it is using.

Another observation is that the balanced partitioning configurations resulted in much more diversity in the results than the random ones. This can lead to a conclusion that the partitioning problem should be in fact considered dominant over the scheduling problem, as it is responsible for a room for improvement available for scheduling policies. The same kind of observation was made in order and acceptance scheduling problems [23]. For further experiments, the two relatively extreme strategies *RR* and *NP* have been chosen. The scheduler inside the *TTA* backend has been modified to perform the scheduling on both *NP* and *RR* basis, so that a comparison of performances is possible. The execution times are presented in Fig. 5 (resp. 6) for balanced (resp. random) configurations.

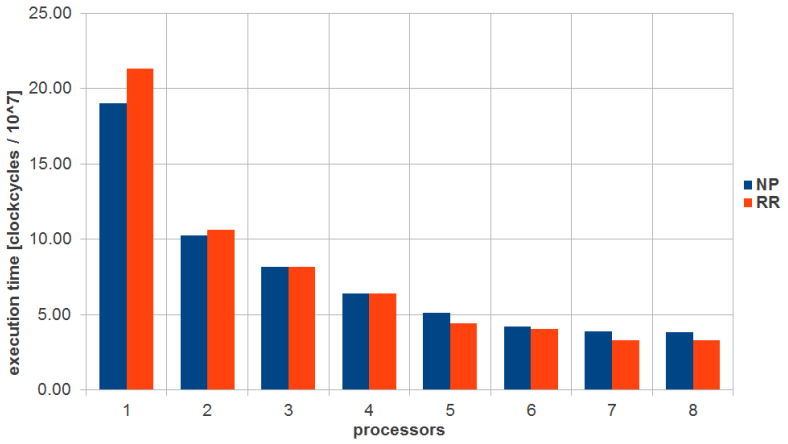

Fig. 5. TTA platform execution: balanced partitioning configurations

The same tendency can be again observed in both sets of partitioning configurations. It thus confirms the legitimacy of the partitioning setup applied to the design space for the exploration of scheduling. Since "good" as well as "bad" partitioning configurations behave in the same way for different scheduling policies, using the simulation tool in order to tune the scheduling policy for the meta-heuristic search of optimal partitioning configuration seems to be a promising direction.

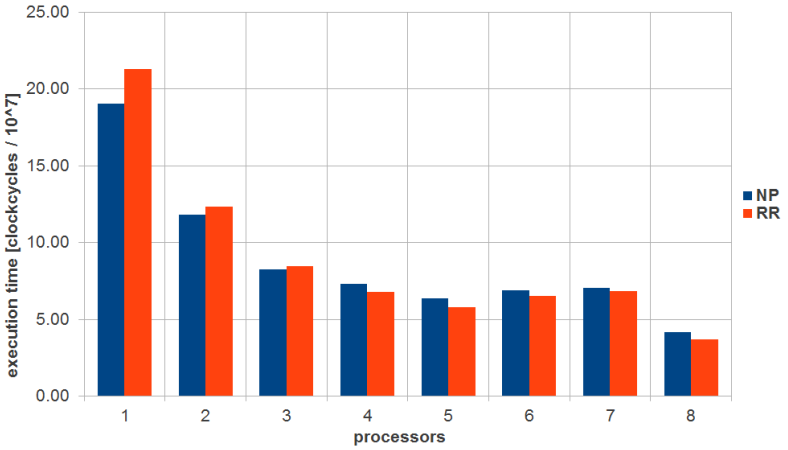

Fig. 6. TTA platform execution: random partitioning configurations

At the beginning, that is, up to 3 units, *NP* outperforms *RR*. However, the difference between them gradually decreases. From 4 units, *RR* achieves a better performance. This phenomenon can by explained by the presence of intra-partition scheduling overhead. This overhead is not measurable according to the current profiling methodology, but we would logically expect it to be proportional to the number of actors in one partition, since if there are more actors, more conditions need to be checked at every scheduling decision. Nevertheless, even in the presence of this unfavorable overhead, the modified scheduler *RR* brought up to 14.5% of improvement.

7 Future work

The most promising aspect of our current work is the expansion of different scheduling approaches to platforms different than *TTA* with an emphasis on the NUMA architectures and various heterogeneous platforms. This involves much more advanced profiling methodology and the introduction of the probability model since a bigger notion of uncertainty is present in the architecture, especially regarding the caches. On the other hand, it is highly important to understand the differences between the estimated execution times and platform results and, in particular, investigate if the intra-partition scheduling overhead can be measured or at least approximated. For this purpose, the goal would be to extend the simulation tool to keep track on the schedulers decision in a more detailed way, especially in terms of the overall number of firing conditions that are checked before a successful execution.

In this work, the scheduling strategies are evaluated globally, that is, the same strategy is defined for every processing unit (partition). It might be useful to analyze also the opportunity of defining a different scheduling policy for each partition, depending on the level of dynamism occurring in the sequencing for every subset of actors. Finally, having the model extended to cover different architectures in a generic way, the target will be to use the simulation tool in order to improve the algorithms for partitioning of dataflow applications. Exploring the properties and performance potential of different scheduling policies should help drive the metaheuristic search for a close-to-optimal partitioning.

References

1. Baker, K. R., Trietsch, D.: Principles of Sequencing and Scheduling. Wiley (2009).
2. Benini, L., Lombardi, M., Milano, M., Ruggiero. M: Optimal resource allocation and scheduling for the CELL BE platform. Annals of Operations Research, 51–77 (2011).
3. Boutellier, J., Sadhanala, V., Lucarz, C., Brisk, P., Mattavelli, M.: Scheduling of dataflow models within the reconfigurable video coding framework. IEEE Workshop on Signal Processing Systems, Washington, DC, 182–187 (2008).
4. Boutellier, J., Ersfolk, J., Lilius, J., Mattavelli, M., Roquier, G., Silven, O.: Actor Merging for Dataflow Process Networks. IEEE Transactions on Signal Processing, vol. 63, 2496–2508 (2015).

5. Casale-Brunet, S., Elguindy. A., Bezati, E., Thavot, R., Roquier, G., Mattavelli, M., Janneck, J. W.: Methods to explore design space for MPEG RMC codec specifications. Signal Processing: Image Communication, vol. 28, 1278–1294 (2013).
6. Casale-Brunet, S., Alberti, C., Mattavelli, M., Janneck, J. W.: TURNUS: a Unified Dataflow Design Space Exploration Framework for Heterogeneous Parallel Systems. Conference on Design and Architectures for Signal and Image Processing (DASIP), Cagliari, Italy (2013).
7. Casale-Brunet, S., Bezati, E., Alberti, C., Mattavelli, M., Amaldi, E., Janneck, J. W.: Partitioning And Optimization Of High Level Stream Applications For Multi Clock Domain Architectures. IEEE Workshop on Signal Processing, Taipei, Taiwan, 177–182 (2013).
8. Casale-Brunet, S.: Analysis and optimization of dynamic dataflow programs. PhD Thesis at EPFL, Switzerland (2015).
9. Eisenring, M., Teich, J., Thiele, L.: Rapid Prototyping of Dataflow Programs on Hardware/Software Architectures. Proc. of HICSS-31, Proc. of the Hawai Int. Conf. on System Sciences, 187–196 (1998).
10. Eker, J., Janneck, J. W.: CAL Language Report. Tech. Memo UCB/ERL M03/48, UC Berkeley (2003).
11. Esko, O., Jääskeläinen, P., Huerta, P., de La Lama, C. S., Takala, J., Martinez, J. I.: Customized exposed datapath soft-core design flow with compiler support. 15th Annual IEEE International ASIC/SOC Conference, 87–91 (2002).
12. Hirzel, M., Soulé, R., Schneider, S., Gedik, B., Grimm, R.: A catalog of Stream Processing Optimizations. ACM Computing Surveys, vol. 46 (2014).
13. Janneck, J. W.: Actors and their composition. Formal Aspects Comput., vol. 15, 349–369 (2003).
14. Lee, E. A., Messerschmitt, D. G. : Static Scheduling of Synchronous Data Flow Programs for Digital Signal Processing. IEEE Transactions on Computers, vol. C-36, 24–35 (1987).
15. Lee, E. A., Parks, T. M.: Dataflow process networks. Proceedings of the IEEE, 773–801 (1995).
16. Michalska, M., Boutellier, J., Mattavelli, M.: A methodology for profiling and partitioning stream programs on many-core architectures. International Conference on Computational Science (ICCS), Procedia Computer Science Ed., 2962–2966 (2015).
17. Pinedo, M.: Scheduling: Theory, Algorithms, and Systems, third edition. Prentice Hall (2008).
18. Selva, M.: Performance Monitoring of Throughput Constrained Dataflow Programs Executed On Shared-Memory Multi-core Architectures. PhD Thesis at INSA Lyon, France (2015).
19. Singh, S.: Round-robin with credits: an improved scheduling strategy for rate-allocation in high-speed packet-switching. Global Telecommunications Conference, GLOBECOM (1994).
20. Silberschatz, A., Galvin, P., Gagne, G: Operating System Concepts. Wiley (2005).
21. Silver, E. A., Zufferey, N.: Inventory Control of an Item with a Probabilistic Replenishment Lead Time and a Known Supplier Shutdown Period. International Journal of Production Research 49 (4), 923–947 (2011).
22. Sinnen, O.: Task scheduling for parallel systems. Wiley Series on Parallel and Distributed Computing (2007).
23. Thevenin, S., Zufferey, N., Widmer, M.: Metaheuristics for a Scheduling Problem with Rejection and Tardiness Penalties. Journal of Scheduling 18 (1), 89–105 (2015).

Thiele, L., Bacivarov, I., Haid, W., Huang, K.: Mapping Applications to Tiled Multiprocessor Embedded Systems. Seventh International Conference on Application of Concurrency to System Design, 29–40 (2007).
24. Thiele, L., Bacivarov, I., Haid, W., Huang, K.: Mapping Applications to Tiled Multiprocessor Embedded Systems. Seventh International Conference on Application of Concurrency to System Design, 29–40 (2007).
25. Ullman, J. D.: NP-complete scheduling problems. Journal of Computer and System Sciences, 384–393 (1975).
26. Weaver, V., Terpstra, D., Moore, S. Non-Determinism and Overcount on Modern Hardware Performance Counter Implementations. IEEE International Symposium on Performance Analysis of Systems and Software, Austin (2013).
27. Yviquel, H., Lorence, A., Jerbi, K., Cocherel, G.: Orcc: Multimedia Development Made Easys. Proceedings of the 21st ACM International Conference on Multimedia, 863–866 (2013).
28. Yviquel, H.: From dataflow-based video coding tools to dedicated embedded multicore platforms. PhD Thesis at Université Rennes, France (2013).
29. Yviquel, H., Casseau, E., Raulet, M., Jääskeläinen, P., Takala, J.: Towards runtime actor mapping of dynamic dataflow programs onto multi-core platforms. 8th International Symposium on Image and Signal Processing and Analysis (2013).
30. Yviquel, H., Sanchez, A., Jääskeläinen, P., Takala, J., Raulet, M., Casseau, E.: Embedded Multi-Core Systems Dedicated to Dynamic Dataflow Programs. Journal of Signal Processing Systems, 1–16 (2014).
31. TTA-Based Co-design Environment, http://http://tce.cs.tut.fi/tta.html, Last checked: December 2014.