Tabu search for partitioning dynamic dataflow programs

Michalska, Malgorzata; Zufferey, Nicolas; Mattavelli, Marco

How to cite

MICHALSKA, Malgorzata, ZUFFEREY, Nicolas, MATTAVELLI, Marco. Tabu search for partitioning dynamic dataflow programs. In: Procedia computer science, 2016, vol. 80, p. 1577–1588. doi: 10.1016/j.procs.2016.05.486

This publication URL: https://archive-ouverte.unige.ch/unige:91053
Publication DOI: 10.1016/j.procs.2016.05.486

Procedia Computer Science

Volume 80, 2016, Pages 1577–1588

ICCS 2016. The International Conference on Computational Science

Tabu Search for Partitioning Dynamic Dataflow Programs

Małgorzata Michalska[1], Nicolas Zufferey[2], and Marco Mattavelli[1]

[1] EPFL STI-SCI-MM, École Polytechnique Fédérale de Lausanne, Switzerland
[2] Geneva School of Economics and Management, University of Geneva, Switzerland

Abstract
An important challenge of dataflow programming is the problem of partitioning dataflow components onto a target architecture. A common objective function associated to this problem is to find the maximum data processing throughput. This NP-complete problem is very difficult to solve with high quality close-to-optimal solutions for the very large size of the design space and the possibly large variability of input data. This paper introduces four variants of the tabu search metaheuristic expressly developed for partitioning components of a dataflow program. The approach relies on the use of a simulation tool, capable of estimating the performance for any partitioning configuration exploiting a model of the target architecture and the profiling results. The partitioning solutions generated with tabu search are validated for consistency and high accuracy with experimental platform executions.

Keywords: tabu search, dataflow, partitioning, design space exploration

1 Introduction

Dataflow programs, as an alternative to the classical sequential programming methods, are a very promising approach for signal processing algorithms implementation on multi-core architectures. They enable the exploration of a rich variety of parallel implementation options and also provide an extensive and systematic implementation analysis [8, 15, 19]. These attractive features rely mostly on the fact that dataflow programs are highly analyzable, platform independent and explicitly expose the potential parallelism of an application. Among several existing dataflow computation models, a dataflow program is, in general, structured as a network of communicating computational kernels, called *actors*, connected by directed, lossless, order preserving point-to-point communication channels (called *buffers*). Data exchanges are only permitted by sending data packets (called *tokens*) over those channels. As a result, the flow of data between actors in such networks is fully explicit.

The dataflow Model of Computation (*MoC*) which enables an implementation of fully dynamic applications, thus can be considered as a very general model, is known in literature as *Dataflow Process Network* (*DPN*) with firings [19]. A *DPN* program evolves as a sequence of discrete steps (called firings) corresponding to the executions of actions that may consume and/or produce a finite number of tokens and modify the internal actor state. At each step,

Selection and peer-review under responsibility of the Scientific Programme Committee of ICCS 2016

doi:10.1016/j.procs.2016.05.486

according to the current actor internal state, only one action can be fired. The processing part of actors is thus encapsulated in the atomic firings completely abstracting from time. The problem of partitioning dataflow programs based on the *DPN MoC* is the object of this work, since apart from allowing to express dynamic algorithms, it also comprises other, less expressive dataflow *MoC*, such as *SDF* and *CSDF* [24].

From the perspective of the dataflow design flow, an important challenge is to find a close-to-optimal partitioning of the program on the target architecture, so that a close-to-maximal possible throughput is achieved. Such result provides information on the parallelism of a dataflow program which can be used for driving the refactoring of the program leading to the better optimized dataflow programs exposing a higher level of parallelism. Using the terminology borrowed from the production field, the objective of partitioning dataflow programs is to find an assignment of n jobs (understood as *action firings*) to m parallel machines (understood as *processors*) so that the overall makespan (completion time of the last performed job among all processors) is minimized. It is assumed that only one job (one action) can be executed at a time on each processor, therefore a scheduling policy must be used to decide about the execution order inside each processor. 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 (i.e., they are the firings of the same action). Between some pairs $\{j, j'\}$ of incompatible jobs (i.e., with $g_j \neq g_{j'}$) there is an associated 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). Among several constraints occurring, the most important ones are: (a) *group constraint* - an actor must be assigned to exactly one processor, therefore all jobs belonging to the same group must be partitioned together, (b) *precedence constraint* - (j, j') involves that a job j (plus the associated communication time) must be completed before job j' is allowed to start, while jobs j and j' can belong to the same or to different groups, (c) *communication channel capacity constraint* - the size of the buffer is bounded by B (indeed, exceeding this value during the execution may introduce serious delays in performance).

Finding an optimal solution to the partitioning problem has been proven to be NP-complete, even when only two processors are considered [34]. Moreover, it is just a set of dimensions in the valid design space. In addition, the scheduling and buffer dimensioning sets add other dimensions to the space in which close-to-optimal solutions are searched [4]. As a result, the size of such a design space is extremely large and very difficult to be efficiently explored. This paper proposes a tabu search approach for finding a close-to-optimal partitioning configuration that involves two important aspects: it evaluates the candidate partitioning configurations by means of simulation (high-accuracy performance estimation) and it uses the execution properties extracted during the simulation as optimization criteria. Since it is possible to extract various properties, an emphasis of the work is to identify which ones (among the four analyzed variants) hold true as the optimization criteria and drive the search so that a high-quality solution is established in the, possibly, shortest time.

The paper is structured as follows: Section 2 presents some examples of a usage of the metaheuristic search (and simulation methodologies) with the purpose of partitioning within/outside the dataflow domain. Section 3 describes the modeling approach used for simulation, properties extraction and search, whereas Section 4 discusses the implemented tabu search approach. Finally, Section 5 reports the results obtained for an example of a dynamic video decoder. They are discussed in Sections 6 and 7, in which directions for future works are also provided.

2 Related work

The problem of partitioning and scheduling dataflow programs, as formulated in the previous section, belongs to the wide class of combinatorial problems. Since the problem is proven to be NP-complete, for realistic instances, it is only possible to develop methods providing close-to-optimal solutions [11]. Usually the fastest approach is to use a constructive heuristic, where a solution is generated from scratch by sequentially adding components to the current partial solution until the solution is complete [2]. However, from the perspective of dataflow application designer, the quality of a partitioning configuration is of high importance. Therefore, the use of more advanced solution methods, namely metaheuristics, appears as a much better choice.

2.1 Metaheuristics: foundations

The term *metaheuristic* was first introduced in [13] and combines the Greek words for *to find* and *beyond, in an upper level*. It is formally defined as an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space [23]. Metaheuristics are usually classified as either *local search* or *population-based* methods. In the first case, the algorithm starts from an initial solution and tries to improve it iteratively, by slightly modifying the current solution. In the second case, it is required to use a central memory to store and operate on certain solutions collected during the search process. Well-known local search methods are: simulated annealing, tabu search, variable neighborhood search, and guided local search. Among the population-based methods, the most popular ones are genetic algorithms, ant colonies, adaptive memory algorithms, and memetic search. The reader desiring more information on metaheuristics can refer to [12, 37].

2.2 Metaheuristics: application examples

Different metaheuristic approaches for solving partitioning problems can be found in literature. Starting from the most general graph partitioning problem, a genetic algorithm can be applied with an opportunity to define multiple objective functions [6]. For the mesh partitioning problem, a multilevel ant colony optimization has provided better results than some evolutionary algorithms or greedy procedures [16]. Another variation of the partitioning problem, namely a multi-dimensional two-way number partitioning problem, can be tackled using variable neighborhood search [17]. There are also several examples of systems/programs partitioning using metaheuristics. The software module clustering can be approached with a genetic algorithm that incorporates multiple different low-level heuristics [18]. Tabu search combined with an intensification routine is developed for efficient partitioning of *Local Area Network* [22]. The partitioning problem is also formulated in an interesting way in [10], since it targets two objectives: reduction of power consumption and traffic optimization. The problem is approached using the ant colony metaheuristic.

Finally, metaheuristics are also used explicitly for partitioning dataflow programs. In [1], simulated annealing is employed for estimating the bounds of the partitioning program. Another example covers the more general problem, which is the design space exploration of dataflow graph-based specifications [32]. The optimization stages include the selection of a target architecture, partitioning, scheduling and design space exploration in order to identify the feasible solutions. The optimizations are performed using the evolutionary algorithm. Multi-objective evolutionary algorithms used for performing an automatic design space exploration are also an objective of work discussed in [28].

2.3 Simulation-based evaluation

The literature is also rich in examples of the methodologies aiming at simulating the program performances on a target platform. In most cases, such methodologies are created around various optimization and design space exploration tools. The trace-based co-simulation is, for instance, employed with the purpose of communication refinement in *KPN* programs [25]. The problem of accuracy of such estimation and the possible approaches to incorporate synthesis methods in order to improve the quality of system exploration are also discussed in [9]. The simulation of the program performance by means of the execution trace analysis is presented in [5]. This work also discusses the advantages of light-weight heuristics over some evolutionary methods in the process of design space exploration.

3 Dataflow program execution modeling

Since the objective is to model the execution of fully dynamic applications (belonging to *DPN* with firings), it is necessary to define a representation of the dataflow program that captures its entire behavior. Such representation can be built by generating a directed, acyclic graph G, called *Execution Trace Graph* (*ETG*). Depending on the application, the *ETG* has to consider a sufficiently large and statistically meaningful set of input stimuli in order to cover the whole span of the dynamic behavior. The size of G varies according to the type of application and to the size of the input stimuli set. Therefore it can be very large for some applications.

The execution of a *DPN* program with firings can be represented as a collection of action executions, called *firings*, that are characterized by intrinsic dependencies of different types: (a) the *internal dependencies* that describe the relations between two *firings* of the same actor; (b) the *token dependencies* that describe the relations between two *firings* of different actors that respectively produce and consume at least one token [4]. Such a model is fully independent from the target architecture. Therefore, in order to model the program execution on a target platform, it is essential to provide an *ETG* with complementary information related to the processing (p_j) and communication ($w_{jj'}$) times valid for this platform. These values are used to weight an *ETG* and to simulate the makespan (total execution time) of a dataflow program.

Such a simulation is performed by an appropriate module in the *TURNUS* co-design framework [39]. The simulation module supports and enables an analysis of different types of architectures with a single platform-independent model of execution [21]. The, generally, high accuracy of the simulation module is sensitive to the properties of the platform because it must rely on the precision of the measurements obtained by executing dataflow program on specific platforms. These measurements, however, might be affected by some uncertainties [35].

4 Tabu search implementation

The partitioning methodology described in this paper relies on tabu search [14], which is still among the most cited and used local search metaheuristics for combinatorial optimization problems. As described in [33], a local search starts from an initial solution, and then explores the solution space by *moving* from the current solution to a neighbor solution, where a *move* involves a slight modification of the current solution. The *neighborhood* $N(s)$ of a solution s is the set of solutions obtained from s by performing each possible move. Unlike in a *descent* algorithm, tabu search avoids getting stuck in the first local optimum by the use of recent memory with a *tabu list*. More precisely, it forbids performing the reverse of the moves done during the last *tab* (parameter) iterations, where *tab* is called *tabu tenure*. At each iteration of

tabu search, the neighbor solution s' is obtained from the current solution s by performing on the latter the best non-tabu move (ties are broken randomly). The process stops, for instance, when a time limit T (parameter) is reached.

The motivation of using tabu search for the considered problem relies on the fact that it has proven to have a good tradeoff according to the following criteria [37]: (1) quality: value of the obtained results, according to a given objective function f; (2) quickness: time needed to get good results; (3) robustness: sensitivity to variations in problem characteristics and data quality; (4) facility of adaptation of the method to a problem; and (5) possibility to incorporate properties of the problem. More generally, tabu search has proven to have a good balance between intensification (i.e., the ability to focus on promising regions of the solution space) and diversification (i.e., the ability to visit various regions of the solution space).

The objective function is defined as the makespan of all firings in the *ETG*. In order to design tabu search for the considered problem, the following features have to be modeled: the way to represent a solution s, the neighborhood structure (i.e., what is a move), the tabu list structure (i.e., what type of information is forbidden), and a stopping condition (i.e., what is the most appropriate time limit).

4.1 Solution encoding and neighborhood structure

A solution for partitioning is represented as a map of actors and processors, where the number of processors is fixed. Each actor can be mapped to only one processor at the time, and each processor must be mapped to at least one actor. Hence, leaving empty processors is not allowed. There are multiple ways to generate neighbor solutions, and they depend on the used definition of a move. In this work, all the neighborhood structures are based on the *REINSERT* operation (i.e., move an actor to another processor) and are generated according to four different criteria:

1. $N^{(B)}$ (for balancing): choose randomly an actor from the most occupied processor and move it to the least occupied processor.

2. $N^{(I)}$ (for idle): for each actor whose idle time is bigger than its processing time, find the most idle processor, different from the one currently mapped. The idle time is defined as the time frame when an actor could execute according to the satisfaction of its firing rules, but since another actor in the same processor is currently working it has to wait to be scheduled.

3. $N^{(CF)}$ (for communication frequency): if an actor has a higher communication frequency (i.e., larger number of token transfers) with actors mapped to a specific processor than with the ones mapped to its current processors, move the actor to this processor.

4. $N^{(R)}$ (for random): choose randomly an actor and move it to a different processor (randomly chosen).

4.2 Tabu list structure

Any time an actor a is moved from a processor q to another processor, it is forbidden to put a back to q for tab iterations, where tab is an integer uniformly generated in interval $[a, b]$ (a and b have been tuned to 5 and 15). Two other sensitive parameters need to be tuned in order to implement tabu search, namely T (the time limit) and p (the proportion of neighbor solutions generated during each iteration). T determines the termination point of the algorithm, whereas p is a random sample involving $p\%$ of the (possibly big) neighborhood of the current solution

s. A large value of p contributes to the intensification ability of the method (indeed, many solutions around s are explored), whereas a small value plays a diversification role (indeed, no focus is put around s).

5 Experiments

One of the objectives of applying a metaheuristic to a partitioning problem is to estimate the upper bound on the speed-up that can be obtained by a dataflow program on different numbers of processors. This information provides an evaluation of a feature of a dataflow program and answers to the question, if the limit of the performance of the dataflow program on a multi-core platform is due to a non-optimal partitioning, or an insufficient level of parallelism implemented in the datflow program. Since the objective of this work is to evaluate and compare different variants of a partitioning metaheuristic, it is important to make an appropriate choice of (a) target platform, (b) analysed algorithm/application implemented by the dataflow program.

The platform chosen for the experiments is based on the *Transport Triggered Architecture* (*TTA*) [38]. This choice has been dictated by several factors, such as: the most complete representation of the architecture model in the simulation module, thoroughly confirmed high accuracy of the simulation, independence of the profiling results from the partitioning configuration. The last property makes the profiling methodology much more reliable, comparing to other platforms, such as, for instance, *Non-Uniform Memory Architectures*, when the results of profiling obtained for one mapping configuration are hardly reproducible for others and require re-profiling with every change of mapping (i.e., a methodology presented in Chapter 5 of [29]). The requirement of an application with much potential parallelism is satisfied on the *TTA* with an MPEG-4 SP decoder design with a total number of 41 actors. The upper bound on the parallelism has been evaluated as a proportion of the critical executions in the overall execution time assuming a full parallel configuration and the unbounded buffer size, as described in Chapter 5 of [3]. For this particular application, the potential parallelism is around 6.28.

5.1 Procedure

Since tabu search requires specifying an initial mapping configuration, the first task is to generate a set of instances that are characterized by possibly different properties. For this purpose, two sets of mapping configurations were generated with the constructive heuristics: one with the total workload balanced in a greedy way, another one with fully random configurations. In each set, the configurations spanned on 2 to 8 processors were generated. Considering the chosen application, 8 processors should already sufficiently convey the potential parallelism. The sets of initial configurations were validated for the consistency between the simulation and platform execution on *TTA* by comparing the simulated clock cycles with the output of the *TTA* cycle-accurate simulator [36].

The second step consisted of tuning the parameters of tabu search, namely T (the time limit) and p (the percentage of neighbor solutions generated in each iteration). In order to objectively compare the four proposed ways of generating the neighborhood, it is necessary that each variant receives exactly the same amount of time to perform the search. First, using $p = 0.5$, each tabu search variant was performed in multiple runs (with different seeds) on each instance. For each run, the search was stopped any time 5 minutes had elapsed without improving the best encountered solution (during the current run) by at least 1%. Parameter T is set as the largest encountered stopping time (minus 5 minutes) among all these experiments. Next, with the selected value of T, all the tabu search variants were tested (again with multiple

runs) with different values of $p \in \{0.2, 0.4, 0.6, 0.8\}$. For each p, average results were computed, which allows easily deducing the parameter p to use.

Using the set of initial configurations and having the parameters tuned, the four variants of tabu search could be accurately compared. For each variant, the best result was computed in terms of the total number of clock cycles simulated and speed-up obtained versus the mono-core execution. For each number of processors (2 - 8), the best variant was chosen, both: for the initially balanced and random configuration, and the final solution was also validated for consistency. Figure 1 summarizes the tools used at different stages of experiments and the information dependencies between them.

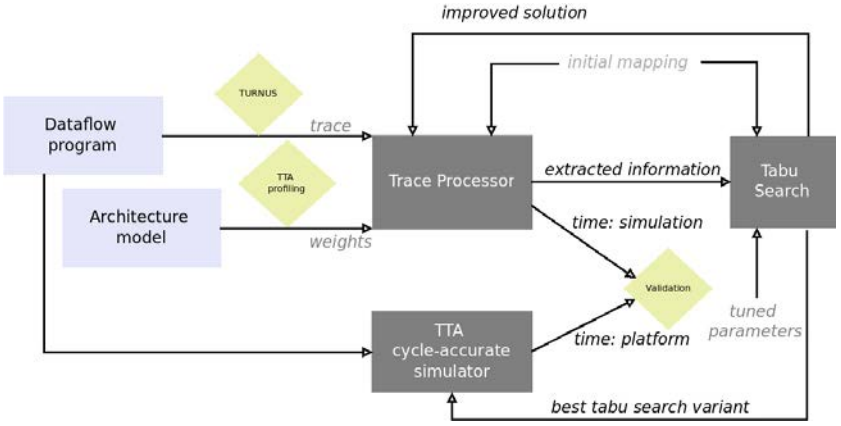


Figure 1: Toolchain.

5.2 Results

The initial validation resulted in a high simulation precision with an average discrepancy less that 1%. It remains consistent with our previous experiments, where the *TTA* platform was validated for the purpose of performance simulation [20], even though the MPEG-4 SP design variant used in the current experiments is much more complex and the same input sequence results in ca. 2 times bigger *ETG* than in the referenced work. The tuning of T was applied to the set of initial configurations and seeds. For each tabu search variant, among all test instances, the minimal, maximal and average values of the stopping time have been recorded and are summarized in Table 1. The tuning of p has been performed with the maximal value of T among the neighborhood types. The following values of p have been deduced to provide with the best solutions:

- $N^{(B)}$: all values of p gave exactly the same result (0.2 was chosen);
- $N^{(I)}$: the best results were obtained for values between 0.4 - 0.8 (0.4 was chosen);
- $N^{(CF)}$: the best results were obtained for values between 0.4 - 0.6 (0.4 was chosen);
- $N^{(R)}$: the best results were obtained for values between 0.2 - 0.4 (0.2 was chosen);

For each neighborhood type, the solution was generated with the tuned values of p and T. For these partitioning configurations (referred as the best ones), the speed-up versus the mono-core was calculated and taken as a final evaluation of the solution. The set of speed-up's

for the initially balanced (resp. random) set of configurations is presented in Table 2 (resp. 3). Next, in Tables 4 and 5, the percentage of improvement versus the initial configuration is given. It denotes which neighborhood type provided the largest improvement.

Table 1: Results of time tuning [minutes].

Neighborhood	Minimal	Maximal	Average
$N^{(B)}$	0	13	6
$N^{(I)}$	19	44	25
$N^{(CF)}$	10	84	48
$N^{(R)}$	24	318	134

Table 2: Best speed-up's - initially balanced configurations.

Processors	$N^{(B)}$	$N^{(I)}$	$N^{(CF)}$	$N^{(R)}$
1	1.00	1.00	1.00	1.00
2	1.85	1.93	1.88	1.92
3	2.67	2.75	2.79	2.70
4	3.21	3.61	3.49	3.49
5	4.04	4.45	4.30	4.27
6	4.83	4.92	4.95	4.91
7	5.4	5.82	5.79	5.64
8	5.76	6.28	6.26	6.27

Table 3: Best speed-up's - initially random configurations.

Processors	$N^{(B)}$	$N^{(I)}$	$N^{(CF)}$	$N^{(R)}$
1	1.00	1.00	1.00	1.00
2	1.75	1.92	1.87	1.94
3	2.08	2.66	2.65	2.65
4	2.77	3.36	3.33	3.30
5	2.34	3.94	4.23	4.01
6	2.61	4.73	3.94	4.23
7	2.48	4.81	4.62	4.04
8	3.24	5.95	4.66	4.56

Table 4: Improvement - initially balanced configurations.

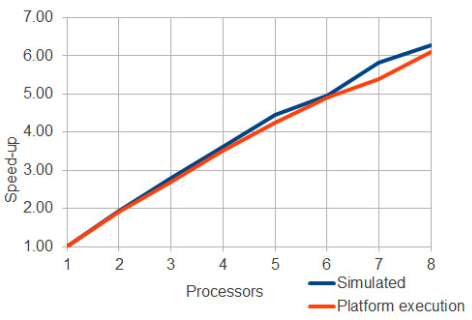
Processors	[%]	Best neighborhood
1	-	-
2	5.38	$N^{(I)}$
3	4.60	$N^{(CF)}$
4	14.79	$N^{(I)}$
5	23.95	$N^{(I)}$
6	17.64	$N^{(CF)}$
7	7.28	$N^{(I)}$
8	8.19	$N^{(I)}$

Table 5: Improvement - initially random configurations.

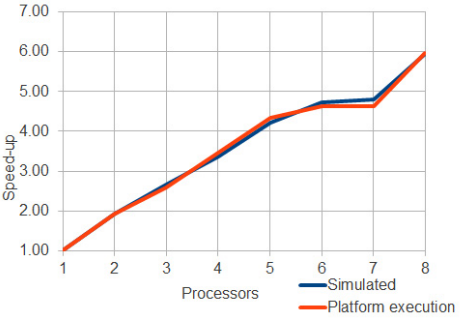
Processors	[%]	Best neighborhood
1	-	-
2	10.10	$N^{(R)}$
3	22.00	$N^{(I)}$
4	17.79	$N^{(I)}$
5	44.63	$N^{(CF)}$
6	44.79	$N^{(I)}$
7	48.49	$N^{(I)}$
8	47.24	$N^{(I)}$

6 Discussion

The comparison of the four tabu search variants leads to an observation that in most cases, $N^{(I)}$ provided the best result. It was occasionally slightly outperformed by the $N^{(CF)}$, and only in one case by the $N^{(R)}$. Due to the properties of the *TTA* related to the negligible communication cost, the $N^{(CF)}$ variant can be considered as quasi-random. It however prevents some moves to



(a) Initially balanced configurations.



(b) Initially random configurations.

Figure 2: Simulation vs platform execution for the best mappings.

be performed and therefore shortens the time of analysis comparing to the $N^{(R)}$. For the $N^{(R)}$ variant, the time required for analysis is multiple times greater than for the other variants, since basically *any* move is allowed for generating a neighbor solution. This freedom of choice, however, does not necessarily correspond to the quality of the results. The $N^{(B)}$ brought a very little improvement and only for certain initial configurations. As a result, the $N^{(I)}$ variant seems to be the most effective and promising one, since it provided the best solution (reasonably considered to be close-to-optimal) in most cases and operated much quicker than the $N^{(R)}$ variant. The results obtained for $N^{(I)}$ and $N^{(CF)}$ also prove that a *smart* choice of moves outperforms random moves.

Further observations can be made, when the quality of the initial configurations obtained by the constructive heuristics is compared with the solutions generated by tabu search variants. First, it can be concluded that the quality of the final solution is sensitive to the quality of the initial solution. Within the assumed time frame, the solutions generated from an initially balanced *(good)* configuration always outperformed the ones generated from an initially random *(bad)* ones. This happened even though the relative improvement of the initially random configurations was much larger than of the initially balanced ones (up to 48% vs 24%). Second, certain numbers of processors offered more room for improvement than some others, for instance, for the initially balanced configurations the largest improvement is observed around 4 - 6 processors, with only slight improvements in other cases.

The bottleneck of the methodology is in the time required for a single simulation of each solution, because in the case of complex applications, even for a very short input sequences, it is expressed in seconds. Such a value might be negligible if an analysis is launched only once (i.e., to evaluate the bottlenecks or extract some properties), but grows in importance for the purpose of a metaheuristic search, when a similar simulation must be executed thousands or millions of times. For this reason, the most time-consuming part of the methodology was, in fact, the fine-tuning of the parameters. Nevertheless, it is not going to be a practical challenge for the dataflow designer, since the amount of time required for refactoring and optimization of the application most likely greatly outstrips the time amount required for partitioning. In any case, the more important aspect of the experiments is the accuracy of the simulated results referred to the platform executions, which is an important issue occurring in various fields [30, 31]. It can be argued which range of differences can be considered negligible. Regarding the discrepancy that we have observed, it must be taken into consideration that for the *TTA*, as well as for other multi-core architectures, a scheduler overhead is present [7]. This kind of overhead is hardly

measurable and may depend not only on the size of actor fraction inside each processor, but also on the types of actors (i.e., how many conditions need to be checked in order to evaluate if an actor is executable). This could explain (to some extent) why in the set of configurations with a little, constant discrepancy, some *picks* of larger differences (i.e., 11% in one of our cases) actually occur.

7 Future works

Several improvements could be considered to the metaheuristic developed in this study. Instead of separately using the four proposed variants for the neighborhood generation, some ongoing experiments involve combining different variants in the same algorithm, for example: mixed types of neighborhood at each iteration or some priorities assigned to the moves, so that the most promising moves are considered first. The latter would involve employing a learning feature as proposed, for instance, in [27]. It could be also interesting to compare tabu search with other metaheuristics, such as different ant approaches, which are usually more complex and sometimes rely on tabu search [26].

From the design perspective, the experience with *TTA*, as well as with other (i.e., NUMA) platforms raises the requirement of introducing further extensions to the used architecture model. The extensions might include: scheduler overheads, communication patterns and an associated cost, caches and memory contention latencies. These extensions should further improve the accuracy of the simulation and, as a consequence, the efficiency of the partitioning metaheuristics, as well as help identify other optimization criteria for the algorithms.

ACKNOWLEDGEMENT

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

References

[1] M. A. Arslan, J. W. Janneck, and K. Kuchcinski. Partitioning and mapping dynamic dataflow programs. *2012 Conference Record of the Forty Sixth Asilomar Conference on Signals, Systems and Computers (ASILOMAR)*, pages 1452–1456, 2012.

[2] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35:268 – 308, 2003.

[3] S. Casale-Brunet. Analysis and optimization of dynamic dataflow programs. *PhD Thesis at EPFL, Switzerland*, 2015.

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

[5] J. Castrillon, R. Velasquez, A. Stulova, W. Sheng, J. Ceng, R. Leupers, G. Ascheid, and H. Meyr. Trace-based KPN composability analysis for mapping simultaneous applications to MPSoC platforms. *Design, Automation and Test in Europe Conference and Exhibition (DATE)*, 2010.

[6] D. Datta and J. Rui Figueira. Graph partitioning by multi-objective real-valued metaheuristics: A comparative study. *Applied Soft Computing*, 11:3976–3987, 2011.

[7] R. I. Davis and A. Burns. A survey of hard real-time scheduling for multiprocessor systems. *ACM Computing Surveys*, 43:1 – 44, 2011.

[8] J. B. Dennis. First version of a data flow procedure language. In *Symposium on Programming*, pages 362–376, 1974.

[9] C. Erbas and A. D. Pimentel. Utilizing synthesis methods in accurate system-level exploration of heterogeneous embedded systems. *IEEE Workshop on Signal Processing Systems*, 2003.

[10] M. Farias, E. Barros, A. G. Silva-Filho, A. Araújo, A. Silva, and J. Melo. An ant colony meta-heuristic for energy aware application mapping on nocs. In *20th IEEE International Conference on Electronics, Circuits, and Systems, ICECS 2013, Abu Dhabi, December 8-11, 2013*, pages 365–368, 2013.

[11] M. Garey and D. Johnson. *A Guide to the Theory of NP-Completeness*. W.H. Freeman, 1979.

[12] M. Gendreau and J.-Y. Potvin. *Handbook of Metaheuristics*. Springer, 2010.

[13] F. Glover. Future paths for integer programming and linkage to artificial intelligence. *Computers and Operations Research*, 13:533–549, 1986.

[14] F. Glover. Tabu Search - part I. *ORSA Journal on Computing*, 1:190–205, 1989.

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

[16] P. Korosec, J. Silc, and B. Robic. Mesh partitioning: a multilevel ant-colony-optimization algorithm. *International Parallel and Distributed Processing Symposium, 2003. Proceedings.*, 2003.

[17] J. Kratica, J. Kojic, and A. Savic. Two metaheuristic approaches for solving multidimensional two-way number partitioning problem. *Computers and Operations Research*, 46:59–60, 2014.

[18] A. C. Kumari, K. Srinivas, and M. P. Gupta. Software module clustering using a hyper-heuristic based multi-objective genetic algorithm. *IEEE 3rd International Advance Computing Conference (IACC)*, pages 813–818, 2013.

[19] E. A. Lee and T. M. Parks. Dataflow process networks. *Proceedings of the IEEE*, 83(5):773–801, May 1995.

[20] M. Michalska, J. Boutellier, and M. Mattavelli. A methodology for profiling and partitioning stream programs on many-core architectures. In *International Conference on Computational Science (ICCS), Reykjavik, Iceland, June 1-3, Procedia Computer Science Ed.*, volume 51, pages 2962–2966, 2015.

[21] M. Michalska, S. Casale-Brunet, E. Bezati, and M. Mattavelli. High-precision performance estimation of dynamic dataflow programs. *IEEE 10th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC-16) (to appear)*, 2016.

[22] M. R. Nusekabel and K. J. Christensen. Using tabu search to find optimal switched LAN configurations. *Southeastcon '98. Proceedings*, pages 298–301, 1998.

[23] I. H. Osman and G. Laporte. Metaheuristics: A bibliography. *Annals of Operations Research*, pages 513–623, 1996.

[24] T. Parks, J. Pino, and E. A. Lee. A comparison of synchronous and cyclo-static dataflow. *Asilomar Conference on Signals, Systems and Computers*, 1995.

[25] A. D. Pimentel and C. Erbas. An IDF-based trace transformation method for communication refinement. In *DAC*, pages 402–407. ACM, 2003.

[26] D. Schindl and N. Zufferey. Optimization by ant algorithms: Possible roles for an individual ant. *Optimization Letters*, 6:963–973, 2012.

[27] D. Schindl and N. Zufferey. A learning tabu search for a truck allocation problem with linear and nonlinear cost components. *Naval Research Logistics*, 61:42–45, 2015.

[28] T. Schlichter, M. Lukasiewycz, C. Haubelt, and J. Teich. Improving system level design space exploration by incorporating sat-solvers into multi-objective evolutionary algorithms. In *2006 IEEE Computer Society Annual Symposium on VLSI (ISVLSI 2006), 2-3 March 2006, Karlsruhe, Germany*, pages 309–316, 2006.

[29] M. Selva. Performance monitoring of throughput constrained dataflow programs executed on shared-memory multi-core architectures. *PhD Thesis at INSA Lyon, France*, 2015.

[30] E. A. Silver and N. Zufferey. Inventory control of raw material under stochastic and seasonal load

times. *International Journal of Production Research*, 43:5161–5179, 2005.

[31] E. A. Silver and N. Zufferey. Inventory control of an item with a probabilistic replenishment lead time and a known supplier shutdown period. *International Journal of Production Research*, 49:923–947, 2011.

[32] J. Teich, T. Blickle, and L. Thiele. An evolutionary approach to system-level synthesis. In *In Proc. of Codes/CASHE 97 - the 5th Int. Workshop on Hardware/Software Codesign*, pages 167–171, Braunschweig, Germany, Mar 1997.

[33] S. Thevenin, N. Zufferey, and M. Widmer. Metaheuristics for a scheduling problem with rejection and tardiness penalties. *Journal of Scheduling*, 18:89–105, 2015.

[34] J. D. Ullman. NP-complete scheduling problems. *Journal of Computer and System Sciences*, 10:384 – 393, 1975.

[35] 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.

[36] H. Yviquel, A. Sanchez, P. Jaaskelainen, 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.

[37] N. Zufferey. Metaheuristics: Some principles for an efficient design. *Computer Technology and Application*, 3:446–462, 2012.

[38] TTA-Based Co-design Environment. http://http://tce.cs.tut.fi/tta.html, Last checked: April 2016.

[39] TURNUS. http://github.com/turnus, Last checked: April 2016.