Procedia Computer Science

Volume 51, 2015, Pages 1443–1452

ICCS 2015 International Conference On Computational Science

Execution Trace Graph Based Multi-Criteria Partitioning of Stream Programs

Małgorzata Michalska, Simone Casale-Brunet, Endri Bezati, and Marco Mattavelli

École Polytechnique Fédérale de Lausanne, Switzerland

Abstract

One of the problems proven to be NP-hard in the field of many-core architectures is the partitioning of stream programs. In order to maximize the execution parallelism and obtain the maximal data throughput for a streaming application it is essential to find an appropriate actors assignment. The paper proposes a novel approach for finding a close-to-optimal partitioning configuration which is based on the execution trace graph of a dataflow network and its analysis. We present some aspects of dataflow programming that make the partitioning problem different in this paradigm and build the heuristic methodology on them. Our optimization criteria include: balancing the total processing workload with regards to data dependencies, actors idle time minimization and reduction of data exchanges between processing units. Finally, we validate our approach with experimental results for a video decoder design case and compare them with some state-of-the-art solutions.

Keywords: execution trace graph, dataflow, partitioning

1 Introduction

For several signal processing fields, the use of dataflow programs as a methodology for the implementation of the processing algorithms constitutes an interesting approach versus the classical sequential programming methods. Dataflow programming enables to explore a rich variety of parallel implementation options and also provides more extensive and systematic implementation analysis [6, 11, 13]. These attractive features rely mostly on the fact that dataflow programs are highly analyzable, platform independent and explicitly expose the potential parallelism of the application. Several dataflow computation models are structured as (hierarchical) networks of communicating computational kernels, called *actors*. Actors are connected by directed, lossless, order preserving point-to-point communication channels, called *buffers*. As a result the flow of data between actors in such a network is fully explicit and data sharing is only permitted by sending data packets, called *tokens*.

This work considers a very general dataflow Model of Computation (MoC) called Dataflow Process Network (DPN) with firings. A specific property of this MoC is that actors executions

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

doi:10.1016/j.procs.2015.05.334

are performed as a sequence of discrete *firings*. During each firing an actor can (a) consume a finite number of input tokens, (b) produce a finite number of output tokens, and (c) modify its own internal state if it has any. The algorithmic part of a single actor firing is specified inside the so-called *actions*. At one time, according to the internal state of the current actor, only one action can be fired. The resulting absence of race conditions makes the behaviour of a dataflow program more robust to different execution policies.

The purpose of this work follows the general objective of a dataflow program implementation onto a many-core architecture, that is, achieving the maximum data throughput exploiting the smallest possible number of processing units. In such perspective, the problem of optimal resources allocation and the partitioning problem are explicitly connected. Finding the optimal assignment of actors or, using the more general terms, tasks to the processing units has been proven to be an NP complete problem, even when only 2 cores are considered [18]. Additionally, it strongly affects the opportunities of developing scheduling policies [2, 4].

In the theoretic notation, the partitioning problem under study in this work can be described as $P|prec, groups|C_{max}$, which is the assignment of a set of tasks $\mathcal{T}$ to identical processors under arbitrary constraints with no pre-emption such that the make-span (total execution time) is minimized [9]. A further assumption implies that for tasks assigned to the same processing unit only one can be executed at one time. Following such a problem definition, the state of the art of partitioning heuristics developed for various platforms is very rich. The approaches described in literature range from simple, greedy placements, up to integer programming based methodologies [2] or refinements using genetic algorithms [16]. In all cases, the crucial problem is to identify the subset of measures and parameters that leads to quality results and good performance independently from the application of a program. As a dataflow design can be perceived as a weighted graph, a plenty of classical approaches for graph partitioning can be applied. The most common optimization criteria for this purpose include either communication cost minimization [10, 12], workload balance [7, 15] or multiple properties [17, 19].

Transferring the generally formulated partitioning problem to the dataflow programming paradigm requires applying some further constraints related to the tasks. The most influencing ones are the intrinsic dependencies between action firings and their affiliation to the specific actor that cannot be violated during the partitioning process. This property makes the execution of load balancing more specific regarding the dependencies and yields some modifications of the objective functions. Furthermore, the approximation of program execution must be thorough enough to capture the input dependent behaviour within the firings.

The main novelty of the presented partitioning approach is to handle the mentioned earlier properties of dataflow programs execution. First of all, an Execution Trace Graph (further referenced as ETG) is generated for a given input stimulus to capture the complete behaviour of the design. Secondly, the execution is profiled in order to obtain the clock-cycles required for each action to proceed. Finally, information related to actions executions, their influence and the communication volume is being extracted and used as a metric for the partitioning algorithm. The proposed algorithm aims at achieving the general balance of the workload, however with regards to the dependencies. The algorithm also makes an attempt to find a trade-off between the program performance and resources utilization. It defines metrics capable of establishing the number of units which is optimal from the processor occupancy balance perspective.

The paper is structured in the following way: first, Section 2 gives an overview of the basic properties of the execution trace graph. Then, Section 3 describes details of the partitioning heuristic with an emphasis put on the, mentioned earlier, trade-off estimation. Next, Section 4 describes the experiments and results obtained by the performance simulation and a direct platform execution. Finally, conclusions are drawn in Section 5.

2 Execution Trace Graph

The execution of a *DPN* program with firings can be represented as a collection of action executions called *firings* as described in the introduction. Consequently, it is possible to explicitly characterize the intrinsic dependencies between two firings. For example, if a firing consumes tokens, it must depend on the execution of the actor that produces those tokens. The same can be stated if an execution firing makes use of a state variable that was previously modified by another firing. Different types of dependencies can be identified and used to characterize the execution of a dataflow program (i.e. Finite State Machine dependencies, State Variable dependencies, Guard dependencies, Port and Tokens dependencies [5]). Defining dependencies between executed firings establishes precedence orders: if firing f_2 depends on firing f_1, then f_1 has to be executed and completed before f_2 can be started. The transitive hull of the dependencies is the precedence relation $\prec$, so the precedence constraint among f_1 and f_2 can be expressed as $f_1 \prec f_2$. The set of firings $\mathcal{F}$ and dependencies $\mathcal{D}$ can be represented with a directed and acyclic graph $\mathbf{G}(\mathcal{F},\mathcal{D})$ called *ETG*, where each single firing is represented by a node and each dependence by a directed arc. It is possible to group different dependency types into two categories: (a) the *internal dependencies* (i.e. Finite State Machine, State Variable, Guard and Port) and (b) the *token dependencies*. The first group describes the relations between two firings of the same actor, while the second one describes the relation between the firings that respectively produce and consume at least one token. It must be noted that in the domain of dynamic dataflow programs the size of $\mathbf{G}$ could vary according to the execution input stimulus. For systems which implement several classes of signal processing applications (e.g. video or audio codecs, packet switching in communication networks), probabilistic approaches are meaningful representations of the underlying processing model. In order to generate statistically representative *ETGs* it is essential to be able to provide such sets of input stimuli that sufficiently cover the whole span of the application behaviour.

3 Trace-based partitioning approach

The proposed partitioning approach requires delivering the *ETG* of a dataflow program obtained for a specific input stimulus. A complementary information that is necessary to model the program execution on the target platform is the number of clock-cycles required for each action to execute. This information (further referenced as *weights*) is being profiled and incorporated into the *ETG* as described in [5]. All information being used for the partitioning heuristic is extracted from the trace by analysing the properties of the firings, such as their number, load, precedence/succession relation, token exchange volume.

3.1 Assumptions

The most important assumption made in this work is the predefinition of the scheduling policy. We assume a simple Round Robin strategy to be used on every processing unit to perform the scheduling of the subnetwork of actors that is mapped on it, if it contains multiple. Although the close-to-optimal scheduling solution that might be identified can be different from the Round Robin scheduling used here, we assume the solution of the partitioning problem dominant. It is supported by experimental observations and some previous works on identifying quasi-static scheduling policies [3].

Another important assumption of the approach presented is that in a first stage the costs of data exchange between processing units are neglected. Therefore the weights assigned to the

firings contain only the computational load of their algorithmic part. In order to consider the data exchange costs in a further stage of the algorithm, we analyse number of tokens that need to be exchanged between partitions. Additional measures related to data transfers, such as, for instance, read/write overhead are not considered.

The partitioning algorithm provides two different use cases. First, when the target number of available identical processing units is fixed and the mapping can span on them. This process is depicted in Fig. 1. In the second case, in order to establish the most optimal number of units, the heuristic algorithm proceeds until certain conditions on execution time and resources utilization are fulfilled.

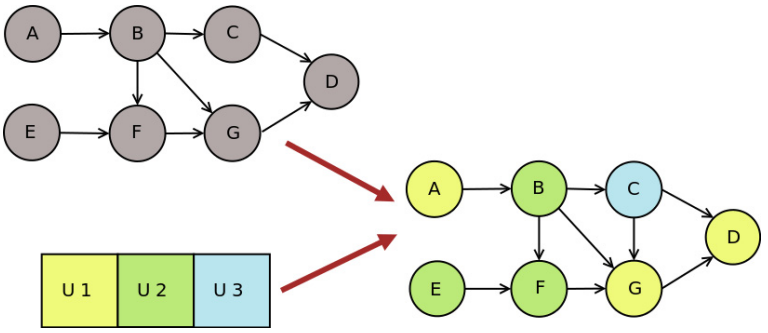

Figure 1: Example of partitioning.

3.2 Load Balancing Algorithm

The heuristic defines some parameters evaluated directly from the weighted ETG. These are the following:

1. **Actor Workload** (AW) is the sum of weights assigned to each action firing belonging to a given actor A: $\sum w_{f_i} \in A$;

2. **Actor Preceding Workload** (APW) is defined as the maximal sum of weights of each firing of each actor that precedes the given actor A in the network in terms of topological order: $max \sum w_{f_i \in A_j} A_j \prec A$. An example of calculation is presented in Figure 2;

3. **Actors Common Predecessors** (ACP) number is evaluated for each pair of actors and denotes number of actors appearing on the topological list of predecessors for both from the pair.

The initial assumption places all actors on separate processing units. A reduction applied iteratively (Algorithm 1) involves removing units one-by-one by attaching them to those with most appropriate values of APW and ACP. Our goal is to join units where overall APW is small with those with big APW so that the actors about to fire at the similar time within the execution do not block each other. Additional criterion favours a high ACP value between actors inside one unit, as most likely there is a kind of pipeline between them that would disable their parallelisation anyway.

After each reduction the ETG is processed in order to simulate the new execution time. Although this operation might be time consuming, especially for big traces, it helps to keep

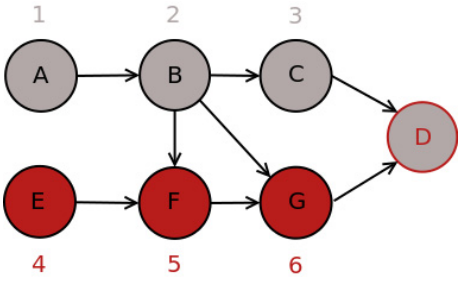

Figure 2: Simple example of preceding workload calculation for actor D. The weight of each actor is indicated in the picture. A branch with the maximal sum of weights goes through actors E, F and G, therefore preceding workload for D = 15.

track on the performance decrease and adjust a further phase of the algorithm. Once the close-to-optimal execution time cannot be maintained any more, we expect the value of *increase* to exceed the *iLIMIT* and further reductions are performed on a different basis. This stop condition could be reached at different numbers of units, depending on the granularity of the dataflow design. As for the analysed networks and the experiments that have been carried out, we have set those factors to 66% (remains of 1/3 of *uINITIAL* - initial number of processing units) and 10% (allowed *increase* for each reduction). When this point is reached, some partitions usually have a remarkable workload. In such case, joining them is not profitable any more, because it contributes to a greater workload imbalance. Therefore, a further reduction involves a separate reassignment of actors from the partition with lowest utilization to other units, using the same *ACP* criterion as previously.

```
Data: INPUT: Trace, Weights; OUTPUT: Mapping
while units number > uLIMIT AND increase < iLIMIT do
    take partition with lowest average APW;
    find partition with highest average ACP;
    join partitions;
    if units number new < uINITIAL/3 OR user condition then
        while units number > uLIMIT AND decrease < dLIMIT do
            take partition with lowest average workload;
            while actors remaining do
                find partition with highest ACP;
                attach actor;
            end
        end
    end
end
```

Algorithm 1: Basic partitioning procedure

Defining the stop condition of the algorithm requires emphasising its two modes. In the first one, that is, when the target number of units (*uLIMIT*) is given, the reduction simply proceeds till the expected number is achieved. For the second application case, when the optimal number of units is unbounded and needs to be established by the algorithm, we introduce two additional parameters. These are: (a) the **Average Partitioning Occupancy** (*APO*), calculated as an average value of processing time of each unit expressed in percent; (b) the **Standard Deviation of Occupancy** (*SDO*), calculated as a statistical standard deviation for the population of units processing times expressed in percent. These parameters are calculated after each iteration.

Our observations lead to characterize the balanced workload of a partitioning configuration with a high value of average occupancy and, at the same time, a low value of standard devi-

ation. With such distribution of values, in the ideal case, all partitions should be equally and maximally occupied. Therefore, we use the ratio of *APO* to *SDO* as an evaluation of partitioning configuration. As the reduction procedure continues, this ratio quite naturally increases. If the opposite occurs, it usually means that with a reduction we have introduced a strong inequality of the workload among units. Small decreases are, however, acceptable as verified experimentally. Hence, if the decrement exceeds the allowed value (*dLIMIT*) the algorithm reaches its stop point. Just like other tuning parameters of the algorithm the value of allowed decrement may vary from application to application basing on its granularity properties. After some experimental work it has been observed that this parameter can be appropriately set in the range of a few %.

3.3 Idleness Optimization

The representation of execution provided by the *ETG* enables the analysis of different states of the actors taken during the execution. Analysis of these states may provide us with some useful indications on the search for more efficient configurations. The following actor states consider the occurrences for which an actor is not processing and has not yet terminated: (a) the **Blocked writing** takes into account the situation where the actor could fire, but the buffer it is expecting to write to is full. Therefore, it is necessary that the actor waits for the available space; (b) the **Blocked reading** considers the situation where the actor has not yet received the required input tokens and therefore cannot be executed; (c) the **Idle** finally corresponds to the situation where even though the actor has necessary tokens and required space in a buffer, it cannot be fired due to the limitations of the scheduling policy (another actor on the same unit is currently processing).

With the currently used model of simulation, the most urgent state to handle and eliminate/reduce is the Idle one. Therefore, a further stage of the optimization procedure focuses on moving actors with the highest overall idle time to the least occupied unit. If a move did not decrease the idle time of an actor or it increased the idle time of any actor of the target unit it joined in, the move is reversed and the actor is labelled as *unmovable*. This procedure continues until all actors are either moved or unsuccessfully attempted to move. For the analysed networks, we consider the idleness of an actor to be reducible, if it is among 10% of actors with the longest idle time.

3.4 Communication Volume Optimization

The second optimization strategy attempts to conform in a simple way with the initially neglected cost of token exchange. Processing the information about token dependencies within the trace, we calculate how many tokens need to be exchanged between each pair of partitions. If for a certain actor on any processing unit, the number of tokens to exchange with an actor on another unit takes a remarkable part of overall token exchange between these two units, the actor is moved to the corresponding unit in order to cancel this cost out. Granularity of the applications that were analysed lead this level to be set experimentally to 2%.

4 Experimental Results and Discussion

The testbench we have used to test the partitioning approach consisted of two different dataflow network designs: RVC CAL [8, 14] implementations of JPEG and MPEG4-SP decoder. While the first one is a small network of only 6 actors, that has been used mostly to validate the

algorithm, the second one consists of 34 actors and is a relatively large design, challenging from the partitioning perspective. The *ETGs* have been generated using the TURNUS co-exploration environment [1], developed by the authors. The weights assigned to each firing have been obtained using C generator retrieving the clock-cycles by the hardware counters of the processor [20]. The platform used for experiments was the 4-cores Intel i7-3770 3.4 GHz. The first part of the results is based on the simulation of the execution times expressed in clock-cycles obtained for the mappings spanned between 6 to 1 unit. These results (normalized values) are presented in Tables 1 and 2. Additional information calculated for each configuration is the average occupancy (*APO*) and the standard deviation (*SDO*), that follow the definitions introduced in the previous Section. Finally, the last column presents a calculated speed-up vs the monocore configuration.

Table 1: JPEG Decoder: Simulation

No. of units	Execution Time [clc]	APO	SDO	Speed-up
6	1054	21%	35%	1.27
5	1057	25%	38%	1.27
4	1057	32%	40%	1.27
3	1057	42%	41%	1.27
2	1058	63%	36%	1.27
1	1340	100%	0%	1

Table 2: MPEG4-SP Decoder: Simulation

No. of units	Execution Time [clc]	APO	SDO	Speed-up
6	4368	31%	31%	1.87
5	4368	37%	32%	1.87
4	4368	47%	32%	1.87
3	4368	62%	29%	1.87
2	4514	90%	6%	1.8
1	8148	100%	0%	1

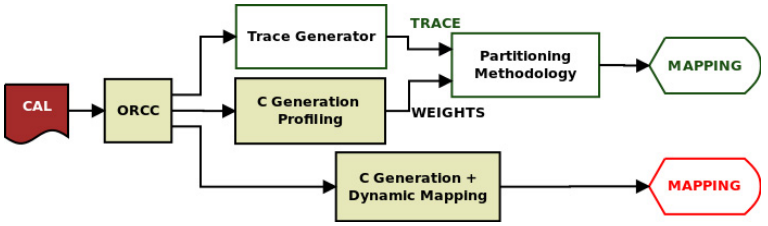

Figure 3: Schematic representation of the experiment procedure.

The measure of performance in case of platform execution experiments is the number of displayed frames per second. The throughput achieved has been compared with the mappings generated with different strategies provided by Orcc [21] (Fig. 3). Special attention was paid to partitioning configurations spanned on up to 4 units, as enforced by the properties of the used machine. Table 3 contains performances for the mappings generated by the algorithm with and without applying additional optimization approaches for JPEG Decoder. Table 4 comprises

performances for mappings generated by different Orcc strategies for the same design. In the same way, Tables 5 and 6 present results for MPEG4-SP Decoder. Missing values in the tables denote strategies the experiments were not possible for due to various uncertainties on the software side.

Table 3: JPEG Decoder: Platform Execution

No. of units	Basic [fps]	+ Min-Idle [fps]	+ Min-Communication [fps]
1	2217	2217	2217
2	3631	2239	3462
3	3843	2085	3298
4	3481	3498	3448

Table 4: JPEG Decoder: ORCC

No. of units	KLR [fps]	MR [fps]	RR [fps]	WLB [fps]
1	2217	2217	2217	2217
2	2229	2208	2258	3767
3	-	2634	2641	3396
4	-	3082	3419	3358

Considering the speed-up relative to the maximal speed-up achievable for an application for certain number of units, simulation proved the results to be on the way to close-to-optimal. As for the platform execution, a speed-up of 1.9 for 2 units and 2.5 for 3 units has been achieved. Moreover, the algorithm along with its optimization procedures outperformed the state-of-the art solutions provided by Orcc, although the communication minimization seemed to improve the performance in a pretty selective way. The major strength of the developed approach over the referenced one is the stability of the solution, as the analysis of execution trace does not introduce any notion of randomness. On contrary, for the referenced Orcc strategies it could be easily noticed when running the same procedure several times for the identical input stimuli, that the resulting configuration could differ from the one competitive to the proposed algorithm to the one way less efficient than a simple Round Robin placement. A good example of this occurrence is for example the MKEC strategy run for 3 units, where the rate spanned from 1303 fps (the best result, as presented in Table 6) to only 857 fps.

It can be clearly observed that for the simulation results a *saturation* of speed-up occurs, so the execution time does not decrease any more as we increase the number of available units. This phenomenon is most likely due to the limit of parallelism achievable for an application. It also falls on the point where a proportion of *APO* and *SDO* is relatively good. It could correspond to the mentioned earlier trade-off of performance and resources utilization that the algorithm aims at, in general. On the other hand, platform execution experiments point the decrease of performance beyond the *saturation* level what reveals an additional cost that outstrips the potential parallelism of an application. To make the simulation fully resemble the platform execution it would be necessary to extend the assumed model by communication cost and possible further overheads (due to i.e. scheduling within a partition). A model fully corresponding to the platform would enable getting rid of most of the fixed tuning parameters and replace them with dynamic boundaries obtained directly from platform simulation.

Some values in the tables also reveal border effects of the optimization procedures. For example, for the platform execution of JPEG, Idle Optimization decreases the throughput for 2 and 3 units. It might be a corner case of the optimization, when although we reduce the idle

time of a partition, we affect the execution of some critical actors what in the end contributes to a decrease in the performance. A similar situation might occur for the Communication Volume Optimization, where a move may reduce one heavy communication channel, but increase other at the same time. These limitations definitely need to be overcome in next versions of the algorithm.

Table 5: MPEG4-SP Decoder: Platform Execution

No. of units	Basic [fps]	+ Min-Idle [fps]	+ Min-Communication [fps]
1	536	536	536
2	868	984	1015
3	1327	1332	1075
4	1130	1222	1036

Table 6: MPEG4-SP Decoder: ORCC

No. of units	KLR [fps]	MKCV [fps]	MKEC [fps]	MR [fps]	RR [fps]	WLB [fps]
1	536	536	536	536	536	536
2	997	995	979	985	729	968
3	-	1191	1303	1309	887	1195
4	-	1145	1212	1145	873	1214

5 Conclusion

Partitioning is an indispensable element of porting dataflow programs onto many-core architectures. This work proposed an efficient methodology based on the analysis of the execution trace. It has been validated by means of two design examples and the results obtained experimentally visibly outperformed currently available solutions. The proposed choice of optimization measures and functions the heuristic relies on appeared to follow the correct line of investigation. The future work involves 2 different directions. First, a research should be continued on the exploration of the partitioning search space. It should lead to development of further heuristics that would guarantee stronger improvement of the performance independently from the application type. Its complementary aspect is an extension of the design case classes. On the other hand, a work should be continued on the model side, especially on the introduction of further target functions related to the extensive cost model of the data exchanges among processing elements. Other model extension could also involve simulation of memory accesses and shared caches.

ACKNOWLEDGEMENT

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

References

[1] TURNUS. `http://github.com/turnus`, Last checked: June 2014.

[2] 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.

[3] J. Boutellier, Ch. Lucarz, S. Lafond, V. M. Gomez, and M. Mattavelli. Quasi-static scheduling of CAL actor networks for reconfigurable video coding. *Journal of Signal Processing Systems*, 2009.

[4] S. Casale-Brunet, E. Bezati, C. Alberti, M. Mattavelli, E. Amaldi, and J. W. Janneck. Partitioning And Optimization Of High Level Stream Applications For Multi Clock Domain Architectures. In *2013 Ieee Workshop On Signal Processing Systems (Sips)*, IEEE Workshop on Signal Processing, pages 177–182, Taipei, 2013. Ieee.

[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 RMC codec specifications. *Signal Processing: Image Communication*, 28(10):1278 – 1294, 2013.

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

[7] R. Diekmann, B. Monien, and R. Preis. Load balancing strategies for distributed memory machines. *Multi-Scale Phenomena and their Simulation. World Scientific*, 1997.

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

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

[10] C.M. Fiduccia and R.M. Mattheyses. A linear-time heuristic for improving network partitions. *19th Conference on Design Automation*, pages 175 – 181, 1982.

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

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

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

[14] M. Mattavelli. MPEG reconfigurable video representation. In Leonardo Chiariglione, editor, *The MPEG Representation of Digital Media*, pages 231–247. Springer New York, 2012.

[15] 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.

[16] 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.

[17] M. Tanaka and O. Tatebe. Workflow scheduling to minimize data movement using multi-constraint graph partitioning. *12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, pages 65 – 72, 2012.

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

[19] L. Wang, J. Liu, J. Hu, Q. Zhuge, and E.H.-M. Sha. Optimal assignment for tree-structure task graph on heterogeneous multicore systems considering time constraint. *IEEE 6th International Symposium on Embedded Multicore Socs (MCSoC)*, pages 121 – 127, 2012.

[20] V. Weaver, D. Terpstra, H. McCraw, M. Johnson, K. Kasichayanula, J. Ralph, J. Nelson, P. Mucci, T. Mohan, and S. Moore. Papi 5: Measuring power, energy, and the cloud. *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2013.

[21] H. Yviquel, A. Lorence, K. Jerbi, G. Cocherei, A. Sanchez, and M. Raulet. Orcc: multimedia development made easy. *The 21st ACM International Conference on Multimedia, France*, 2013.