Memory and communication driven spatio-temporal scheduling on MPSoCs

Zubair Wadood Bhatti*, Narasinga Rao Miniskar§, Davy Preuveneers*, Roel Wuyts§,
Yolande Berbers* and Francky Catthoor§
*DistriNet, KU Leuven, Celestijnenlaan 200A, B-3001, Leuven, Belgium.
{zubairwadood.bhatti,davy.preuveneers,yolande.berbers}@cs.kuleuven.be
§IMEC, Kapeldreef 75, B-3001, Leuven, Belgium.
{miniskar,wuytsr,catthoor}@imec.be

***Abstract*—Scheduling and executing software efficiently on contemporary embedded systems, featuring heterogeneous multi-processors, multiple power modes, complex memory hierarchies and advanced interconnects, is a daunting task. State-of-the-art tools that schedule software tasks to hardware resources face limitations: (1) either they do not take into account the interdependancies among processing, memory and communication constraints (2) or they decouple the problem of spatial assignment from temporal scheduling. As a result existing tools make sub-optimal spatio-temporal scheduling decisions. This paper presents a technique to find globally optimized solutions by co-exploring spatio-temporal schedules for computation, data storage and communication simultaneously, considering the inter-dependencies between them. Experiments on mapping exploration of an image processing application on a heterogeneous MPSoC platform show that this co-exploration methodology finds schedules that are more energy efficient, when compared to decoupled exploration techniques for the particular application and target platform.**

I. Introduction

Graphics, multimedia and wireless applications on nomadic, battery-operated consumer devices need to process large amounts of data and deliver high throughput but within very limited energy budgets and timing constraints. Developing these applications is difficult because of two primary reasons. On the one hand, the platforms feature heterogeneous cores, each with different power modes, and complex memory hierarchies including software managed scratchpad memories. This increasingly complex hardware has become very difficult to exploit efficiently, for the developers. On the other hand, the applications have become very dynamic, making it impossible for compilers to statically make all the scheduling and resource management decisions. Moreover, embedded and real-time operating systems cannot make these decisions in a fine grained manner because of the runtime overhead this would incur.

To reduce this overhead researchers have studied two-phased mapping techniques [1], [2], [3] for configuring the hardware and scheduling software. These techniques split the mapping into design-time and runtime phases. At design-time, static schedules are made off-line for several possible runtime scenarios [3]. At runtime, these precomputed schedules are dynamically activated depending on the runtime situations. These approaches have shown to be capable of handling different types of dynamism with low overhead [1], [2], [3].

The design-time phase of mapping software applications onto an MPSoC platform includes (implicitly or explicitly) scheduling of three different types of *activities* onto three different types of *resources*; (1) computational tasks on the different cores, (2) data storage in the memories and (3) the data transfers on the interconnect. Several memory and communication aware mapping techniques have been proposed [4], [5], [6], [7]. The techniques presented in [4], [7] split the mapping problem into two sub-problems of spatial assignment and temporal scheduling and solve them separately. This reduces the search space but at the cost of quality of the solution. Similarly, [5] uses the ant colony technique in a multi-stage implementation but only optimizes the execution time of the application ignoring the energy consumption, whereas, [6] optimizes the energy consumption along with the execution time of the application but defines mapping only as a spatial assignment problem, without the temporal aspect of scheduling.

This paper provides a novel formulation of the memory and communication aware mapping problem that allows a combined exploration of spatio-temporal schedules for computation, data storage and communication. Moreover, the cross-cutting aspects of performance, energy and time are considered. The proposed technique is used to find the energy-speed trade-offs for a software pipeline version of the cavity detector application [8] mapped onto an MPSoC platform. The target MPSoC consists of four Strong ARM 1100x processors and two TI-C64X+ processors connected via a shared bus (see Fig. 3). We observe that the mapping solutions found by co-exploration of the three schedules are significantly better when compared to decoupled exploration. The Pareto-optimal mapping solutions found by our exploration technique may be used in the run-time phase of the two-phased mapping techniques [1], [2], [3].

The rest of the paper is structured as follows. Section II gives the motivation behind co-exploration. Section III provides an abstraction for the application. Section IV presents the formulation of the scheduling co-exploration as a constraint based optimization problem. Section V describes the experimental setup and presents the results. The paper ends

with the conclusion and future work in Section VI.

II. Motivation

Our goal is to map applications onto MPSoCs with multi-level memory hierarchies. On such hardware the execution time of a computational task depends not only on the processor it is mapped onto and the corresponding power mode of the processor but also on the location of data in the memory hierarchy (e.g. in L1 or main memories). When mapping software tasks onto hardware in an energy efficient way, both the timing deadlines of the application as well as the resource constraints of the hardware have to be respected. If the mapping is split into different sub-problems solved independently, it leads to sub-optimal solutions. To illustrate the problem, suppose that task mapping is done before memory exploration. Since the location of data is not known yet but tasks need to meet their timing deadlines, the task-mapping exploration has to take one of the two extreme assumptions:

1) All data read/writes are from/to the (slowest) main memory.
2) All input data is first brought into the (fastest) L1 memory before starting the computation and all output data is first produced in the L1 memory before being written off to the main.

The first assumption is too conservative and would unnecessarily force the tasks to run on faster and energy hungry processors or a higher power mode in order to guarantee meeting the deadlines, whereas the deadline could also be met if the data was put in the faster L1 memory while still running the task on the slower (energy efficient) processor or a lower power mode. The second assumption is also undesirable because it creates unnecessary traffic due to copy operations between different memories. Even the data that is used only once is first copied to the L1 memory before being used. Moreover, in order to efficiently utilize the small L1 memories, data objects that are not alive simultaneously may need to use the same memory space. However this is only possible with a combined spatio-temporal scheduling exploration of computations and data storage, like we propose in this paper. Techniques that decouple spatial-binding from temporal-scheduling can not handle this. The alternative solution of first exploring the mapping of data to memory before considering task mapping leads to similar problems. This illustrates the fundamental problem that the decisions of whether or not to copy data into local memories or to run tasks on different processors in order to save time and energy depend not only on each other but also on the availability of communication resources.

To address this problem our co-exploration technique explores spatio-temporal schedules for computation, data storage and communication simultaneously considering the inter-dependencies between them, in order to find globally optimized solutions.

III. Application model

Even though many real life applications are written as sequences of statements (e.g. in C or JAVA). It is extremely complex to reason about scheduling and resource management issues at the level of these statements. The execution semantics of statements are too complex and the granularity is so fine, it makes a reasonable exploration for a real-life application almost impossible due to the sheer size of the problem. In the rest of this paper we therefore use annotated task-graphs as an abstraction for the application.

A. Annotated task-graphs

An annotated task-graph is a directed acyclic graph (DAG) that defines the execution semantics of a parallel application, annotated with profiling information. The nodes in this graph are tasks that represent the computations to be performed. The arcs in the graph are edges that represent dataflow dependencies between the tasks. Tasks in the task-graph execute in parallel when their required data is available on their incoming edges.

1) Tasks: Tasks are modular units of functionality that are composed together to form an application. Tasks of the same application can depend on each other. When executed they consume data from each incoming edge and produce data on the outgoing edges. The execution time and energy consumption of the task depends on three different types of parameters; (1) type of the processor it is executed on, (2) power mode of the processor, and (3) the memory mappings of the incoming and outgoing edges (e.g. L1 memory or main memory). These three parameters together form a *task-mapping*. The energy consumption, execution time along with the bus bandwidth required for each of the task-mappings are profiled and annotated to the tasks. Worst case estimates are taken in case of dynamism.

a) Degree of freedom in task-mappings:: Assume that a task can be executed on P different types of processors, with M power modes each, the task has a total of E incoming and outgoing edges, each of which can be mapped onto L different types of memories. It then has $P \times M \times (L)^E$ possible task-mappings. Although the number of task-mappings grows exponentially with the number of edges connected, it usually does not cause a explosion of combinations, as the number of edges connected to a task is fairly small for many applications [9].

2) Edges: Edges represent dataflow dependencies between tasks. They are annotated with the amount of data being transferred along the edge. Worst case estimates are again used in case of dynamism.

a) Degree of freedom in edge-mappings:: An edge can be mapped to the memory and communication resources in five different ways:

1) *Local memory:* An edge can only be mapped onto the local memory of a processor if both the source and destination tasks of the edge are mapped onto the same processor. In this case data is produced and consumed

in local memory; hence, no data transfers on the bus are required.

2) *Main memory:* If an edge is mapped onto the shared main memory, tasks will read/write data from/to main memory as they execute. Therefore tasks require bus bandwidth to be allocated to them as they execute.
3) *Main-local:* The source task produces data in main memory, whereas the destination task consumes it from local memory. Bandwidth allocation for two different types of data transfers on the bus are required for this mapping, (1) bandwidth allocation for the source task, and (2) bandwidth allocation for copying data from the shared main memory to local memory.
4) *Local-main:* The source task produces data in local memory, whereas the destination task consumes it from main memory. Bandwidth allocation for two different types of data transfers on the bus are required for this mapping, (1) bandwidth allocation for the destination task, and (2) bandwidth allocation for copying data from local memory to main memory.
5) *Local-main-local:* Both source and destination tasks produce and consume data in their local memories, however, they are mapped onto different processors. Therefore data needs to be copied from one local memory to main memory and then from main memory to the other local memory. Bandwidth is allocated for the two copy operations; however, no bandwidth is required for the edge during the execution of the tasks it connects.

An optimal choice of task mapping requires knowledge of edge mappings and vice versa. Moreover, the relations of these mappings to the cross-cutting time and energy cost are non-linear. For example, memory intensive tasks with higher data locality usually cost more if the required data is put into main memory compared to their compute intensive counterparts that can interleave data access with computations. Therefore all these edge-mapping options are explored taking into account the task-mappings, and their relative costs.

IV. Co-Exploration

In [10], the authors show that scheduling a DAG onto multiple processors is an NP-Complete problem. In order to find a schedule for tasks on the processors and schedule bus bandwidth for these tasks (if necessary) at the same time schedule memory space and bus bandwidth for the edges, we formulate the problem as a constraint based scheduling problem and solve it with a constraint solver tool. In this section we describe the search space, constraints, minimization objective of the problem and the exploration procedure for the energy-speed trade-off. Fig. 1 gives an overview of the different types of constraints that are considered.

A. Search Space

We have two types of activities in our problem: tasks and edges. These activities use five types of resources: processors, memory space, bus bandwidth, time and energy. The usage of these resources by the activities form the schedules that define the solution.

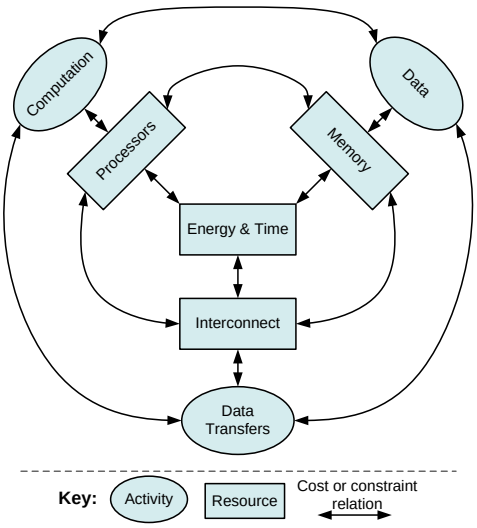

Fig. 1. Relations between activities and resources

The resources in our system are classified into two types; *Renewable* and *Nonrenewable*. Renewable resources are resources that can be returned to the system once a task is finished, such as processors and memories. The nonrenewable resources are permanently consumed, such as energy and time. Activities are modeled with the variables of type *Intervals*, whereas the usage of nonrenewable resources as *Sequences* of *Intervals* and renewable resources are represented as *Cumulative functions* of intervals over time.

1) Intervals: Interval variables are characterized by their start and end points. We use them to model the consumption of non-renewable resources (energy and time) for a possible *mapping* of an activity.

Each possible *task-mapping* is represented by two intervals, a time-interval and an energy-interval. The size of the *time-interval* equals the execution time of the task, including the time spent in memory accesses, for a particular possible mapping of the task. Similarly the size of the *energy-interval* represents the total energy consumed by the task.

Edges are mapped according to one of the five degrees of freedom we differentiated earlier. For example, the edge $E1$ in Fig. 2 mapped as *local-main-local* comprises the following five steps (sub-activities) for more aggressive optimization:

1) Initial storage in a $L1_1$
2) A copy operation to the $L2$
3) Intermediate storage in $L2$
4) A copy operation to the $L1_2$
5) Final storage in the $L1_2$

In order to be able to free resources, each of the sub-activities is modeled with a separate time-interval. However, the energy consumption for steps 1,3 and 5 is either included in the energy consumption of the source/destination tasks or in the copy operations. Therefore, we need at most five time-intervals and two energy intervals to model an edge, i.e. when it is mapped

as a local-main-local edge.

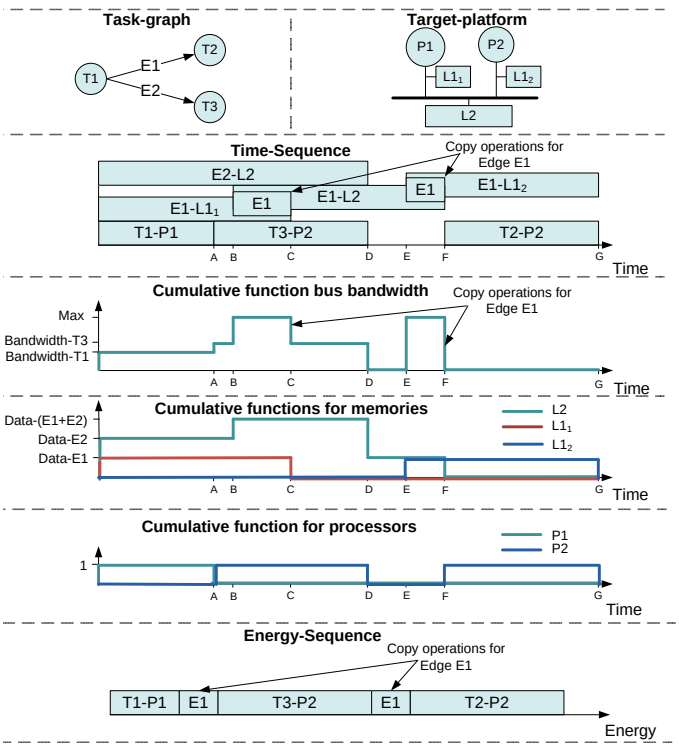

Fig. 2. An example task-graph, a possible schedules for its resource usage and the corresponding cumulative memory usage on the target platform. Edge E1 is mapped as *local-main-local* and edge E2 as *main-memory*. The interval label T1-P1 indicated that the task T1 is executed on the processor P1.

2) Sequences: Sequences are total ordered sets of intervals that belong to the chosen task and edge mappings. There are two sequences, one for time and an other one for energy. Activities such as parallel processing of tasks onto different processors or interleaved copy operations between the local and main memories can happen simultaneously. Therefore, time-intervals of activities are allowed to overlap with each other as long as they respect the capacity constraints of the platform and execution semantics of the model of computation. Note that the energy sequence does not allow overlapping because two activities can never share the same energy.

3) Cumulative functions: A cumulative function for a non-renewable resource is the sum of pulse functions over time-intervals of all the activities using that resource i.e there is a separate cumulative function representing the usage of each processor, memory and interconnect. $f(R)$ is the cumulative function for the resource R.

$$f(R) = \sum_{\forall A | R_A \neq 0} R_A \times \Pi(I_A)$$

R_A is the resource usage of resource R by the activity A (the amount of data in the case of memories, the bandwidth for interconnect and a binary 0/1 for processors). $\{A|R_A \neq 0\}$ is the set of all activities or sub-activities that use resource R. $\Pi(I_A)$ is a pulse function on the time-interval of the (sub-)activity A.

B. Constraints

A valid mapping solution needs to satisfy several constraints. These constraints impose boundaries within the search space, thus limiting the search space. Fig. 1 gives a general overview of the constraints.

1) Constraints between activities and resources: Constraints between activities and resources are of two types. Firstly activities or sub-activities have to respect the capacity of the resources they occupy:

- The cumulative functions for all the memories are always less then, or equal to, the size of the memories.
- The cumulative function for the interconnect is always less, or equal to, then the arbitrated net bandwidth. (The arbitrated net bandwidth is usually the physical bandwidth, depending on the arbitration policy and it's latency and fairness guarantees.)
- The cumulative functions for all the processors are always less than, or equal to, one.

Secondly the resource to which an activity is assigned to affects the execution time and energy consumption of the activity.

- The sizes of the (energy and time)-intervals of a task are equal to the profiled values energy and time of the chosen *task-mapping*i.e. depend on the processor it executes on, the power mode of the processor and on the memories the task reads from and writes to.
- Tasks require a guaranteed bandwidth in order to finish within a certain amount of time. Even though it is possible to explore task execution with reduced bandwidth the relations to timing properties are complex. Therefore we only schedule tasks if the full required bus bandwidth is available to them.
- Copy operations have linear timing properties under reduced bandwidth execution. This allows a more aggressive exploration for a better bus utilization. The size of the time-interval is calculated by dividing the amount of data by the *allocated* bandwidth, whereas, the size of the energy-interval equals the sum of the energy per byte for reading, writing and communication multiplied by the amount of data.

2) Constraints between activities and activities: The execution semantics of the model of computation form the constraints between the different activities:

- Each task executes once and only once.
- A mapping is selected for each activity, the time and energy intervals for all selected mappings are added to the time and energy sequences.
- According to the execution semantics of a task-graph, A task can only execute when the required data is available at its input port. This also enforces precedence constraints between different tasks.
- The time interval of memory storage sub-activities start with the starts of its source task or copy operation and ends with its destination task or copy operation.

3) Constraints between resources and resources: The platform model constitutes the constraints between resources. These constrains express the architecture of the platform (as shown in Fig. 3 i.e which processors can use which memories and which communication requires the shared bus.

C. Objective function and exploration

In order to explore the trade-off between speed and energy consumption the solver is given a time deadline and the objective of minimizing the energy within the time deadline.

- $end\ of(Time\text{-}Sequence) \leq deadline$
- $Minimize(end\ of(Energy\text{-}Sequence))$

The deadline is iteratively increased giving a minimum energy schedule each time and the solutions that provide a trade-off on either the energy or time axis form the set of Pareto-optimal solutions. We are aware that more sophisticated multi-objective exploration techniques exist [5], [6], but the main focus of this paper is on the quality of the solutions.

V. Experimental Evaluation

To evaluate our co-exploration technique we study two aspects: the quality of the mapping solution and the scalability of the approach. To evaluate the quality of the mapping solution we mapped the pipelined version of the cavity detector application [8] on a heterogeneous MPSoC (see Fig. 3). In this task graph $T1_1$ and $T1_2$ are instances of the same function Horizontal Blur. The platform consists of four StrongARM 1100 and two TI-C64X+ processors, each with a local L1 memory of 4KB. The platform also features a shared L2 memory of 128MB. A shared bus connects all memories and processors. To profile the execution times of application tasks, SimItARM and TI-CCStudio v3.3 simulators were used for the StrongARM and TI-C64X+ processors respectively. For the energy consumption, the energy models JouleTrack [11] and functional level power analysis model of TI-C64X+ [12] were used for the StrongARM and TI-C64X+ processors. The amount of communication between different tasks was measured with the architecture independent communication profiler PinComm [13]. The constraints and the objective function were modeled and solved using the open source constraint solver Gecode [14].

For evaluating the quality of the mapping solutions, the execution time and energy consumption of the schedules found by the co-exploration technique are compared with its decoupled counterparts. The two different types of decoupled approaches considered for validation are: (1) Exploring tasks to cores mapping with worst case assumptions for memories and communication. (2) Exploring tasks to cores and edges to Memories mapping with worst case assumptions on communication. We have implemented these two approaches with similar models as of the co-exploration.

Fig. 4 shows the results of the experiments. Each point in the trade-off space of Fig. 4, represents a unique mapping solution. The energy consumption and execution time are synthesized using the three schedules that constitute the mapping solution (computation, memory and communication). From Fig. 4 we observe that higher degrees of coupling and co-exploration can find significantly better results. For these experiments it took an average of 54 minutes to find a solution on an Intel Core2Duo 2GHz machine.

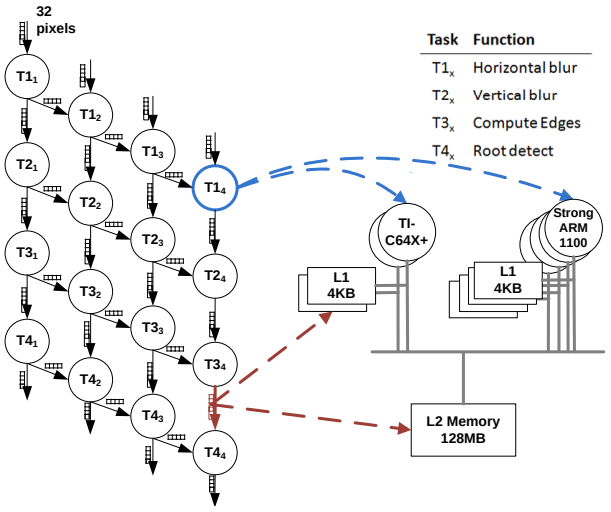

Fig. 3. Mapping pipelined cavity detector on MPSoC

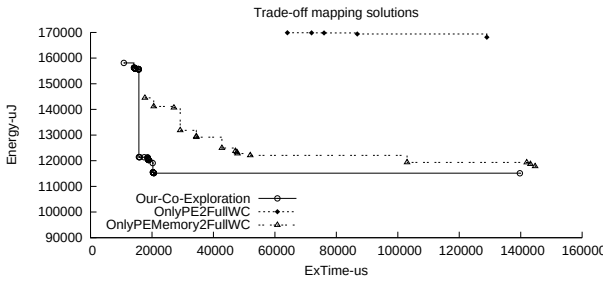

Fig. 4. Exploration results for the different techniques

VI. Conclusion and Future Work

This paper presents a design-time co-exploration technique that schedules tasks on processors, data objects on the memories and data transfers on the interconnect. The inter-dependencies between the three different types of schedules are modeled and accounted for. The effects of decoupling the three explorations are studied by comparing the co-exploration technique with decoupled ones. Validation on a medical image processing application shows that the co-exploration technique is able to find mapping solutions that are significantly more energy efficient and/or faster when compared to decoupled exploration techniques. In future we are looking to evaluate our methodology for other industrial use cases and validate them on the state-of-the-art platforms. In addition we want to extend out methodology for network-on-chips, taking into account multi-hop communication. We would also like to implement bio-inspired search algorithms in order to improve the time taken by the design time exploration.

VII. Acknowledgements

This research is partially funded by the Flemish Agency for Innovation by Science and Technology (IWT) through a Strategic Basic Research project OptiMMA (Contract No IWT-060831) and by the research fund KU Leuven.

References

[1] S. Stuijk, M. Geilen, B. Theelen, and T. Basten, "Scenario-aware dataflow: Modeling, analysis and implementation of dynamic applications," in *Embedded Computer Systems (SAMOS), 2011 International Conference on*, july 2011, pp. 404 –411.

[2] P. van Stralen and A. Pimentel, "Scenario-based design space exploration of mpsocs," in *Computer Design (ICCD), 2010 IEEE International Conference on*, oct. 2010, pp. 305 –312.

[3] S. V. Gheorghita and F. Catthoor, "System-scenario-based design of dynamic embedded systems," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 14, no. 1, 2009.

[4] M. Lukasiewycz, M. Streubühr, M. Glaß, C. Haubelt, and J. Teich, "Combined system synthesis and communication architecture exploration for mpsocs," in *Proceedings of the Conference on Design, Automation and Test in Europe*, ser. DATE '09. 3001 Leuven, Belgium, Belgium: European Design and Automation Association, 2009, pp. 472–477. [Online]. Available: http://dl.acm.org/citation.cfm?id=1874620.1874737

[5] F. Ferrandi, P. L. Lanzi, C. Pilato, D. Sciuto, and A. Tumeo, "Ant colony heuristic for mapping and scheduling tasks and communications on heterogeneous embedded systems," *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, vol. 29, pp. 911–924, June 2010. [Online]. Available: http://dx.doi.org/10.1109/TCAD.2010.2048354

[6] C. Erbas, S. Cerav-Erbas, and A. Pimentel, "Multiobjective optimization and evolutionary algorithms for the application mapping problem in multiprocessor system-on-chip design," *Evolutionary Computation, IEEE Transactions on*, vol. 10, no. 3, pp. 358 – 374, june 2006.

[7] J. Gladigau, A. Gerstlauer, C. Haubelt, M. Streubu andhr, and J. Teich, "A system-level synthesis approach from formal application models to generic bus-based mpsocs," in *Embedded Computer Systems (SAMOS), 2010 International Conference on*, july 2010, pp. 118 –125.

[8] M. Bister, Y. Taeymans, and J. Cornelis, "Automatic segmentation of cardiac MR images," *Computers in cardiology*, pp. 215–218, 1989.

[9] W. Thies and S. Amarasinghe, "An empirical characterization of stream programs and its implications for language and compiler design," in *Proceedings of the 19th international conference on Parallel architectures and compilation techniques*, ser. PACT '10. New York, NY, USA: ACM, 2010, pp. 365–376. [Online]. Available: http://doi.acm.org/10.1145/1854273.1854319

[10] A. Darte, Yves, and F. Vivien, *Scheduling and Automatic Parallelization*, 1st ed. Birkhäuser Boston, Mar. 2000.

[11] A. Sinha and A. Chandrakasan, "Jouletrack-a web based tool for software energy profiling," in *Design Automation Conference, 2001. Proceedings*, 2001, pp. 220 – 225.

[12] J. Laurent, N. Julien, E. Senn, and E. Martin, "Functional level power analysis: An efficient approach for modeling the power consumption of complex processors," *Design, Automation and Test in Europe Conference and Exhibition*, vol. 1, p. 10666, 2004.

[13] W. Heirman, D. Stroobandt, N. Miniskar, R. Wuyts, and F. Catthoor, "Pincomm: Characterizing intra-application communication for the many-core era," in *Parallel and Distributed Systems (ICPADS), 2010 IEEE 16th International Conference on*, dec. 2010, pp. 500 –507.

[14] "Gecode: Generic constraint development environment, http://www.gecode.org."