SAMOSA: Scratchpad Aware Mapping Of Streaming Applications

Zubair Wadood Bhatti,
Davy Preuveneers and Yolande Berbers
DistriNet, K.U.Leuven, Belgium
{zubairwadood.bhatti, davy.preuveneers,
yolande.berbers}@cs.kuleuven.be

Narasinga Rao Miniskar and Roel Wuyts
IMEC, Belgium
{miniskar, wuytsr}@imec.be

***Abstract*—Scratchpad memories have now emerged as an alternative to caches for energy constrained embedded systems. However, effectively mapping data on them while considering energy/timing trade-offs remains a challenge. We present SAMOSA as a technique for mapping streaming applications to scratchpad based MPSoCs. The contribution of this approach is a representation and transformation of the mapping problems – buffer dimensioning and allocation – to a constraint-based optimization problem. SAMOSA was used to explore energy-execution time trade-offs for mapping the H.264 decoder to a scratchpad-based MPSoC. Results show that scratchpad awareness has significant impacts on the energy-execution time trade-offs.**

I. Introduction

With the recent widespread availability of mobile broadband internet and multi-core computing resources on hand held computers, the use of such devices for data intensive applications, such as multimedia is increasing. However, battery time remains a great concern for such devices. Viredaz et al. show in [26] that multimedia applications are one of the most power hungry activities on a mobile device and that the energy consumption of the memory sub-system for multimedia applications is significant. In this paper we focus on reducing the energy consumption of multimedia applications on such devices through better memory management.

In [2], the authors show that scratchpad memories (SPM) consume on average 40% less energy and 46% less area-time when compared to caches of the same capacity. Moreover, they provide much better timing predictability than caches. Therefore, SPMs are now included in many modern day embedded platforms such as ARM 10E, IBM Cell BE, GeForce GTX and Texas Instruments TMS370CX7X. Unlike caches that are managed by hardware and that select their contents on the principle of spatio-temporal locality, in SPM based systems the software is responsible for the allocation to scratchpads.

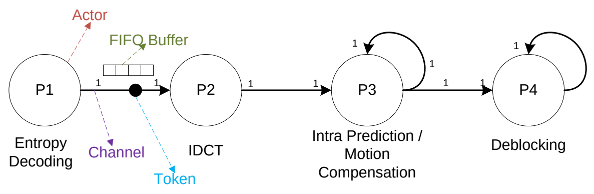

Fig. 1. Synchronous Dataflow Graph for H.264 decoder

One particular application domain that can benefit from scratchpad memories are data intensive streaming applications, such as multimedia players. These applications are often modeled with Synchronous Dataflow Graphs (SDFG) [17] as shown in Fig. 1. In an SDFG, *actors* communicate with each other by passing *tokens* over the logical links called *channels*. These logical channels need to mapped in the memories as *FIFO buffers*. Being a simple model of computation, SDFGs are easy to parallelize, schedule and analyze for timing and resource requirements. OpenDF [5] and StreamIt [24] are examples of stream programming languages whose programming models resemble SDFG.

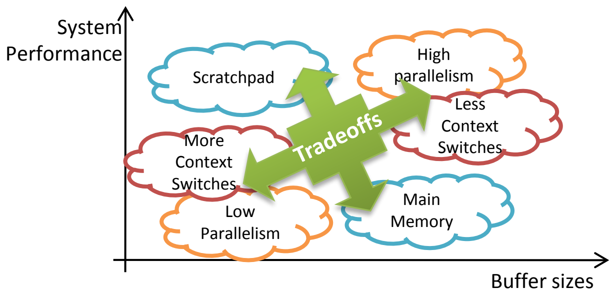

Fig. 2. Trade-offs in buffer dimensioning

Mapping applications described as synchronous dataflow graphs to scratchpad-based MPSoCs raises two important challenges, i.e. *buffer dimensioning* and *scratchpad allocation*:

- *Buffer dimensioning* is defined as, deciding the sizes of the different buffers between actors over intervals of time. Fig. 2 highlights some trade-offs in buffer dimensioning for a scratchpad based system.
  - From a scheduling perspective, a larger buffer provides better decoupling between a pair of actors. Better decoupling provides the freedom to construct schedules with fewer context switches and/or with more parallelism as shown in Fig. 3.
  - For a scratchpad based MPSoC, the execution time and energy consumption of an actor depends on the memory its reads and writes are addressed to. Therefore it is sometimes desirable to have smaller

buffers, so that they could fit in the scratchpad.

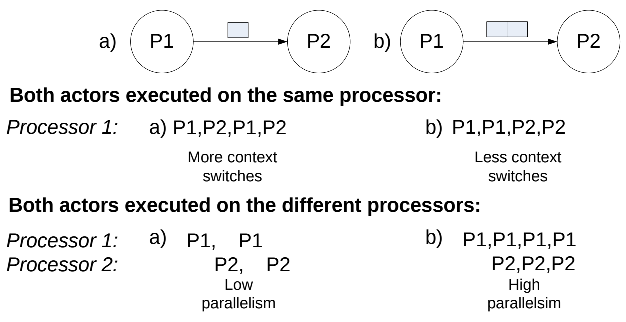

Fig. 3. Context switches and parallelism

- *Scratchpad allocation* is defined as, deciding which buffers are mapped onto the scratchpad memory and over what period of time.
  - Data access density of a buffer is the number of accesses made to the data in the buffer during a firing of an actor divided by the amount of data moved to/from the buffer. In uniprocessor system with only one execution path, allocating buffers with the highest access density to the scratchpad improves both the energy efficiency and the execution time of the application.
  - In a multiprocessor system, however, there are several execution paths of different lengths due to the way the application is scheduled on the different processors. In order to meet a given real-time deadline, it is sometimes necessary to allocate the buffers with lesser data access densities in the longest execution path to the faster scratchpad (as shown in Fig. 4) to reduce the length of the longest execution path.

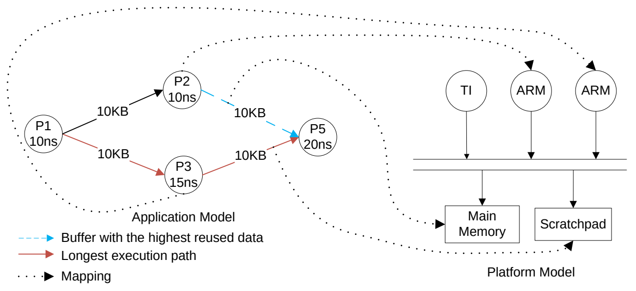

Fig. 4. Trade-offs with scratchpad allocations

The two decisions are interdependent. Moreover, our results show that both these decisions can have significant impact on the timing and energy aspects of system performance. SAMOSA is an technique for mapping SDFGs to scratchpad based MPSoCs. Actors are scheduled over processors and tokens are mapped onto memories. The trade-off between execution time (or throughput) and energy is explored. The buffer dimensioning in SAMOSA is scratchpad aware and the buffer sizes are allowed to vary over time. The scratchpad allocation in SAMOSA implicitly takes into account access densities to buffers and the lengths of the different execution paths on a multiprocessor system.

The proposed technique was used to find the energy-throughput trade-offs for the H.264 decoder (shown in Fig. 1) to a scratchpad-based MPSoC. The platform is similar to TI OMAP, with two StrongARM processors, a TI C64X+ processor, a scratchpad memory and a main memory. We observed that scratchpad sizes strongly influence the energy-throughput trade-off and that letting buffer sizes be functions of time produced mappings that were significantly more energy efficient when compared to fixed buffer sizes.

The paper is organized as follows. Section II gives an overview of the SAMOSA methodology. Section III explains the formats of the application model and platform model required as inputs for SAMOSA. Section IV explains how the application model is transformed to a model that is easier to schedule. Section V presents the constraint programming model for exploration and discusses the variables, constraints and the objective function. Section VI presents the results of our SAMOSA technique on the H.264 decoder. Section VII discusses related work in the domains of buffer dimensioning and scratchpad allocation. Finally, section VIII discusses the conclusion and future work.

II. Overview of SAMOSA

SAMOSA explores schedules of tasks over multiple processors and schedules of data buffers in different types of memories. Buffer sizes are explored as a property of the task schedules, whereas scratchpad allocations are explored as a property of the schedules of data buffers in the memories. The objective of SAMOSA is to find schedules and scratchpad allocations such that the energy consumption is minimized while meeting the quality of service (timing/througput) requirements. All aspects of the problem, including context switches, parallelism and data access densities, are transformed into the domain of energy consumption and execution time of task schedules and memory allocations. SAMOSA combines off-line compiled structural information about the application with dynamically profiled information of non-functional properties such as execution time and energy consumption to carry out this multi-objective optimization.

Fig. 5 presents an overview of SAMOSA. We start with an SDFG model of the application. Each actor of the SDFG is profiled for energy consumption and execution time on the different processors in the platform, with all possible memory mappings of its inputs and outputs. This profiling

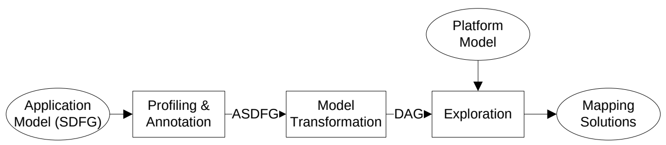

Fig. 5. Overview of SAMOSA

information is annotated on the SDFG to get an Annotated Synchronous Dataflow Graphs (ASDFG). In order to simplify the scheduling, we limit the concurrency of the ASDFG by unfolding it into a Directed Acyclic Taskgraph (DAG), as shown in Fig. 6. This DAG is then used for exploring the required schedules.

The scheduling exploration is realized with IBM ILOG where the mapping is modeled as a constraint based scheduling problem in Optimization Programming Language (OPL) [15]. The execution schematics of the DAG, along with the platform resources (processing, memory) are modeled as constraints. The solver is then asked to find a minimum energy schedule for a given deadline. A set of schedules that give a trade-off between energy and execution time is found by repeating the procedure for the different deadlines.

III. Platform & Application Models

This section describes the models for the application and the platform that are used as an input for SAMOSA.

A. Platform Model

Most of the information about the platform required is extracted through detailed profiling of the application on the platform. A platform is described as a tuple *(Processors, Memories)*, where *Processors* and *Memories* represent the sets of processors and memories in the platform. A processor is defined as a tupple *(ProcessorID, CtxSwthTime, CtxSwthEnergy)*, where *ProcessorID* is a unique number for every processor, *CtxSwthTime* and *CtxSwthEnergy* are the time and energy consumed during a context switch. A context switch includes the loading and the initialization of the actor code before it can start processing data. A memory is defined as a tuple *(MemID, Size)*, where *MemID* is a unique identifier for every memory and *Size* is the size of the memory. The energy values and execution time of the context switch include those spent in the memories and interconnect along with those of the processor.

B. Application Model

An Annotated Synchronous Dataflow Graph (ASDFG) is a Synchronous Dataflow Graph annotated with profiling information. The ASDFG is described as a tuple (A, C), where A is the set of *Actors* and C is the set of *Channels*.

1) Actors: An Actor is assumed to be a deterministic piece of code. In each execution instance it consumes a fixed amount of data from each of its input channels and produces a fixed amount of data on its outputs. The execution time and energy consumption of the actor depends only on the type of processor it is executed on and the memories where its input/output is mapped onto. In case when these properties also depend on the input data, worst case assumptions are taken. The actors themselves are stateless. Any required state is explicitly represented as a self loop channel. Actors are described as a tuple *(ActorID, Ports, Modes)*, where *ActorID* is a unique number for every actor. *Ports* is the set of all input and output ports of the actor. *Modes* is the set of all possible execution configurations for the actor, i.e. one configuration for each possible combination of processors and memories.

2) Modes: A mode is a configuration in which an actor can execute. Consider the application and the platform shown in Fig. 4; the actor $P2$ can be executed on either of the two processors. It has two ports each of which can be mapped to read/write the data either from/to the main memory or the scratchpad. Therefore, the actor $P2$ has eight modes. A mode describes the execution configuration as a tuple *(ModeID, ProcessorID, PortMemIDs, ExecTime, Energy)*, where *ModeID* is a unique number for every mode. *ProcessorID* identifies the processor used in this configuration. *PortMemIDs* is the set of *MemID* that identifies the memories used for each port, i.e. one *MemID* for each port. *ExecTime* and *Energy* are the execution time and energy consumed if the actor is executed in the given mode. These energy and execution time values are obtained through profiling and include the time and energy spent in the processors, memories and interconnect.

3) Channels: A channel is defined as *(ChID, Source, Sink, InRate, OutRate)*, where *ChID* is a unique identifier for each channel. *Source* and *Sinks* are *ActorID*s of the source and sink actors connected to the channel. *InRate* and *OutRate* are the amounts of data produced or consumed by the source and sink actors respectively, each time they execute.

IV. Model transformation

A Synchronous Data Flow Graph is an auto-concurrent model of computation where parallelism is implicit. In order to derive an optimal schedule on a multiprocessor, all possible combinations of the different instances of actors need to be considered which is often not possible. A common approach is to first construct a Directed Acyclic Graph (DAG) where parallelism is explicit [21]. Fig. 6 illustrates the model transformation of a synthetic SDFG. In order to construct a DAG from an SDFG, a periodically admissible sequential schedule (PASS) needs to be computed first. To compute a PASS balance equations for each channel are formulated and an integral vector is found that solves the system of equations. The PASS is then unfolded by an 'unroll factor' and dependencies are added between all the different instances of the actors [14]. The 'unroll factor' depends on the target platform and can be specified by the developer. A DAG is defined as (T, E) where, T is a set of *Tasks* and E is a set of *Edges*.

a) Tasks: Tasks are instances of SDFG actors. Therefore they have all of the same properties and modes as the SDFG actor they belong to. It should be noted that if the tasks belonging to the same actor are executed one after another on the same processor, there is no context switch because the same actor is fired multiple times. Tasks are defined as *(TaskID, ActorID, Ports, Modes)*.

b) Edges: Edges are data transfers between the tasks. An edge is defined as *(EdgeID, Source, Sink, Size)*. It is possible that several edges in the DAG belong to the same channel in the SDFG. The *Size* of an edge is the greatest common divisor of *InRate* and *OutRate* of the SDFG channel it belongs to. The question of time dependant buffer sizes is now transformed

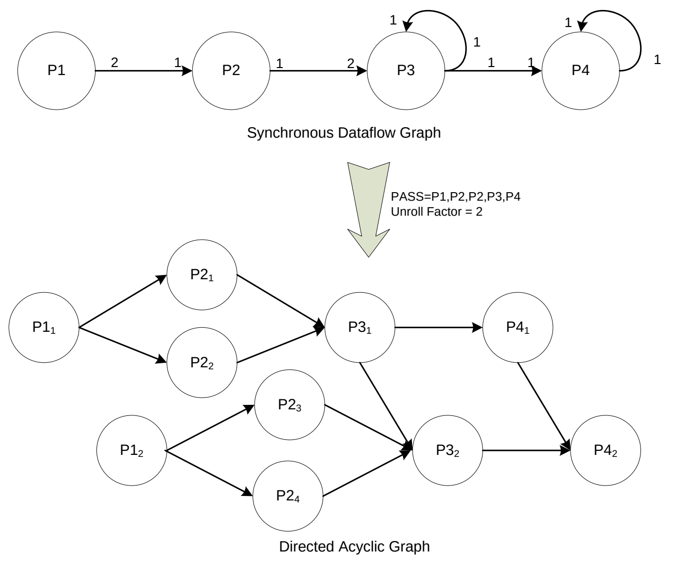

Fig. 6. Model Transformation

into the question of the number of live edges in the DAG and their fixed sizes.

V. Scheduling Exploration

This section describes the scheduling exploration in SAMOSA. In [7], 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 a schedule for the edges on the memories, we formulate the problem as a constraint based scheduling problem and solve it with a constraint solver [15]. In this section we describe the solution space, the constraints and the minimization objective of the problem.

A. Solution Space

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

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. We model usage of nonrenewable resources with the variables of types *Interval* and *Sequence*. The usage of renewable resources are represented as *Cumulative functions*.

1) Intervals: An Interval variable $\underline{a}$ has a start $s(\underline{a})$ and an end $e(\underline{a})$ when it is present; these variables can also be declared as 'optional' in which case they can also be absent $\perp$ i.e. they don't have a start or end. The domain of $\underline{a}$ is $dom(\underline{a})$: The size of interval $\underline{a}$ is $IntervalSize(\underline{a})$:

$$IntervalSize(\underline{a}) = e(\underline{a}) - s(\underline{a})$$

.

a) TaskTime: is an array of intervals (one for each task) that represent the time slot occupied by each task. These are not optional intervals. Therefore each task must be allocated at least one time slot. The size of this interval is not fixed and depends on the selected mode. $TaskTime[T]$ represents the interval that belongs to the task T.

b) TaskModeTime: is a array of optional intervals in time (one for each mode of every task). $TaskModeTime[M]$ represents the time interval that belongs to the mode M. If the mode M is selected:

$$Size(TaskModeTime[M]) = M.ExecTime$$

The dot operator '.' is used to represent tuple components. *M.ExecTime* is the task execution time under mode M. If the mode M is not selected:

$$TaskModeTime[M] = \{\perp\}$$

c) TaskModeEnergy: is an array of optional intervals in energy (one for each mode of every task). $TaskModeEnergy[M]$ represents the energy interval that belongs to the mode M. If the mode M is selected:

$$Size(TaskModeEnergy[M]) = M.Energy$$

where *M.Energy* is the energy consumption under mode M. If the mode M is not selected:

$$TaskModeEnergy[M] = \{\perp\}$$

d) EdgeTime: is an array of intervals that represents the lifetimes of the edges. $EdgeTime[E]$ represents the lifetime of the edge E.

e) EdgeMode: is a two dimensional matrix of optional intervals. The interval $EdgeMode[E, M]$ is present if the edge E is connected to the task of mode M and the mode M is selected see Section V-B3. This variable is responsible for the allocation of buffers to either the main memory or the scratchpad.

2) Sequences: A sequence is a total ordered set of interval variables. These interval variables can also have types assigned to them. An additional constraint of *noOverlap* on sequences can force all the intervals in a sequence to be non-overlapping. Optionally the intervals in a sequence can have *Types*. These types can be used to impose a minimum transition distance between the intervals of a sequence in the presence of a *noOverlap* constraint. This translates into

$$noOverlap(\pi, M) \Leftrightarrow \forall \underline{a}, \underline{b} \in \underline{A},$$
$$((\pi(\underline{b}) = \pi(\underline{a}) + 1) \Rightarrow (e(\underline{a}) + M(T(\pi, \underline{a}), T(\pi, \underline{b})) \leq s(\underline{b})))$$

where $\underline{A}$ is a set of intervals, π is a sequence in $\underline{A}$ with types T and M is a matrix with the minimum transition distances between the different types of intervals.

a) ProcessorTime: is an array of non-overlapping sequences, one for every processor. The domain of these sequences is the array of time intervals of task modes *TaskModeTime*, for all modes that use the particular processor. In the sequence every *TaskModeTime* interval has a *Type* that represents the SDFG actor to which the task belongs *ActorID*. A minimum transition distance that equals *CtxSwitchTime* of the processor is imposed whenever tasks belonging to different actors are executed consecutively on the same processor, as shown in Fig. 7.

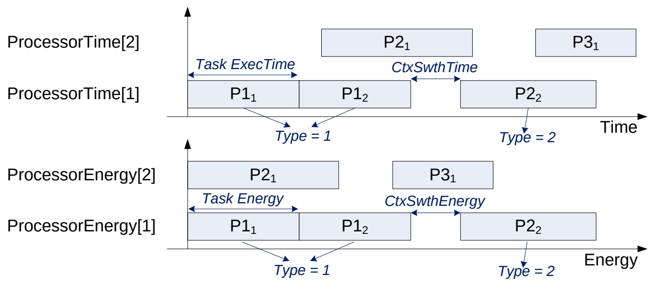

Fig. 7. ProcessorTime & ProcessorEnergy sequences

b) ProcessorEnergy: is an array of non-overlapping sequences, one for every processor. The domain of these sequences is the array of energy intervals of task modes *TaskModeEnergy*, for all modes that use the particular processor. In the sequence every *TaskModeTime* interval has a *Type* that represents the SDFG actor to which the task belongs *ActorID*. A minimum transition distance that equals *CtxSwitchEnergy* of the processor is imposed whenever tasks belonging to different actors are executed consecutively on the same processor, as shown in Fig. 7.

3) Cumulative functions: Cumulative functions are used to represent the usage of renewable resources, such as the usage of different memories. These functions can be composed of elementary functions such as *Pulse(height,interval)* and *StepAtStart(height,interval)* shown in Fig. 8.

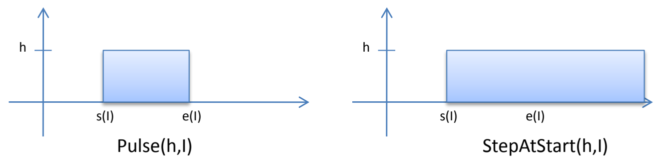

Fig. 8. Elementary Functions

a) MemUsage: is a set cumulative functions that represents the memory usage of different memories over time. $MemUsage[Mem]$ is a model of the usage of the memory *Mem*.

$$MemUsage[Mem] = \sum_{\{E \in Edges | EdgeMode[E,Mem] \neq \perp\}} Pulse(E.Size, EdgeTime(E))$$

B. Constraints

A valid mapping solution needs to satisfy several constraints. These constraints impose additional boundaries within the solution space, thus reducing the search space.

1) Tasks:

- All tasks need to finish before the deadline.

$$\forall_{T \in Tasks} e(TaskTime[T]) \leq Deadline$$

- One and only one *TaskModeTime* interval is present for every task in a valid schedule, and the *TaskTime* interval starts and ends with the selected *TaskModeTime* interval. This behavior is captured by the *alternative* constraint in OPL. An $alternative(\underline{a}, \{b1, .., bn\})$ constraint implies that, if the interval $\underline{a}$ is present then exactly one of $\{b1, .., bn\}$ is present. And that, $\underline{a}$ starts and ends with the chosen interval.
- If a task mode M is chosen the interval $TaskModeTime[M]$ is present, the corresponding interval $TaskModeEnergy[M]$ also has to be present and the sequences *ProcessorTime* and *ProcessorEnergy* should have the same order:

$$\forall_{M \in Modes}, presenceOf(TaskModeTime[M]) \Rightarrow presenceOf(TaskModeEnergy[M]).$$

$$\forall_{M \in Modes, K \in Modes | M.ProcessorID = K.ProcessorID}, s(TaskModeTime[M]) \leq s(TaskModeTime[K]) \Rightarrow s(TaskModeEnergy[M]) \leq s(TaskModeEnergy[K]).$$

2) Memory Size: At all times the maximum data allocated to a memory is less then the size of the memory:

$$\forall_{Mem \in Memories}, MemUsage[Mem] \leq Mem.Size$$

3) Edges:

- The edges enforce precedence constraints between the tasks and the life time of an edge starts at the end of its source task and ends at the end of its sink task:

$$\forall_{E \in Edges}, s(TaskTime[E.Sink]) \geq e(TaskTime[E.Source]), s(EdgeTime[E]) = e(TaskTime[E.Source]), e(EdgeTime[E]) = e(TaskTime[E.Sink]).$$

- The interval $EdgeMode[E, M]$ is present if the edge E is connected to the task of mode M and the mode M is selected:

$$\forall_{E \in Edges, M \in Modes}, presenceOf(TaskModeTime[M]) \Rightarrow s(EdgeMode[E,M]) = s(EdgeTime[E]), e(EdgeMode[E,M]) = e(EdgeTime[E]), \neg presenceOf(TaskModeTime[M]) \Rightarrow \neg presenceOf(EdgeMode[E,M]).$$

C. Objective function

The objective of the optimization is to minimize energy consumption while meeting the timing requirements and resource constraints. For our model we assume all energy spent in the processors and memories is used by tasks, in the form of *TaskModeEnergy* intervals. These intervals are aligned into *ProcessorEnergy* sequences, minimizing the total sum of the ends of these energy sequences will minimize the total energy consumption.

$$\textbf{minimize} \sum_{P \in Processors} ProcessorEnergy[P]$$

In order to explore the trade-off between energy consumption and execution time, a number of mapping solutions are calculated by incrementally varying the timing deadline.

VI. H.264 Decoder use case

In this section we present the results of our technique for mapping an implementation of the H.264 decoder (Fig. 1) to the platform shown in Fig. 5. The H.264 decoder implementation takes a stream of H.264 and processes it frame by frame. Our TI OMAP 35xx like multiprocessor platform consists of two RISC processors (Strong ARM 1100x) operating at 200MHz, one VLIW processor (TI-C64X+) operating at 500MHz a scratchpad memory (SRAM) of 64KB and a main memory (SDRAM) of 128MB. We profile the application for execution times using SimItARM and TI-CCStudio v3.3 simulators for the StrongARM and TI-C64X+ processors respectively. For the energy consumption, we use JouleTrack [22] and functional level power analysis model of TI-C6X [16] for the StrongARM and TIC64X processors. Whereas the dataflow between the actors was measured using PinComm [12] and the memory accesses by Cachegrind. For this experiment we profiled 15 different sample videos of resolution 352x480 for every frame, and took the worst case values for each actor. An unroll factor of two was used for the ASDFG. A higher unroll factor might improve the quality of our results, but the amount of time required for the exploration would significantly increase.

A. Effects of varying Scratchpad sizes

Fig. 9 shows that varying the amount of scratchpad memory for the buffers significantly effects the energy-execution time trade-off. We can observe that in order to meet the same timing deadline with a smaller scratchpad memory available, a higher energy schedule is often required. The scratchpad sizes affect several aspects of scheduling. Smaller scratchpads can cause the buffer sizes to shrink resulting in more context switches and less parallelism. Also, they can cause more DAG edges to be mapped onto the main memory thus increasing the execution time and energy consumption of the tasks connected to those edges. Furthermore, in order to meet the deadlines some tasks may have to be moved onto a faster but more energy hungry processor (in this case the TI-C64X+) thus causing higher energy schedules for the same timing performance.

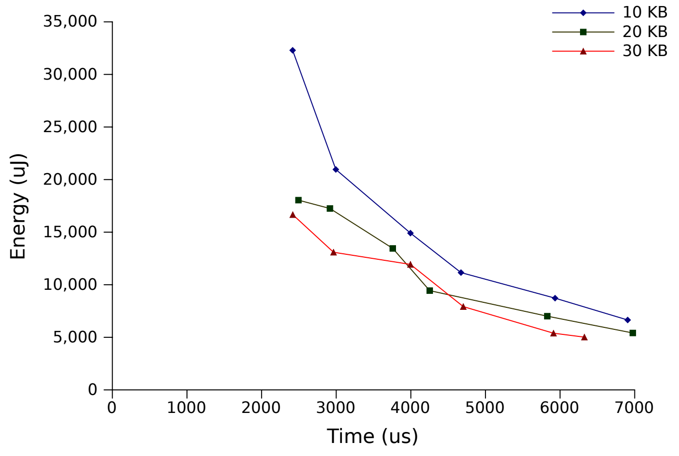

Fig. 9. Energy throughput trade-off under varying scratchpad sizes

B. Memory sharing buffers vs baseline allocation

Efficient utilization of memory space is very important for a scratchpad based system. We study the effects of *memory sharing* optimization [8], [20] on the trade-off between throughput and execution time. In the memory sharing approach buffer sizes are allowed to varry over time. Therefore, the same memory space can be used by different buffers. In the baseline allocation approach buffer sizes remain static. We simulated the baseline appraoch by using a *StepAtStart* function instead of a *Pulse* function in the *MemUsage* function in Section V-A3. Fig. 10 shows that significantly better schedules on the energy-execution time trade-off can be constructed for a memory sharing buffer allocation when compared to a baseline allocation. It is important to note that a memory sharing implementation is expected to require more runtime defragmentation overhead compared to a baseline allocation. However, defragmentation effects are not modeled in this simulation.

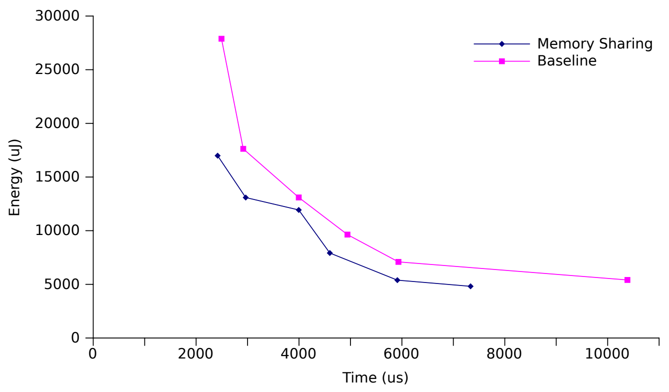

Fig. 10. Memory sharing vs baseline allocation

C. Illustrating the exploration phase

During the exploration phase, the solver attempts to find better alternatives given its best known solution thus far. Fig.

11 shows a screenshot from the ILOG profiler, illustrating how the solution converges in a feasible amount of time.

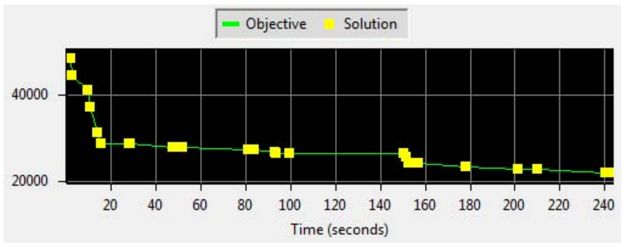

Fig. 11. IBM ILOG screenshot showing convergence of the solution.

VII. Related Work

The problem addressed in this paper crosscuts two concerns in the domain of memory management for embedded systems, and we therefore subdivide the related work in two sections accordingly.

A. Buffer dimensioning

Several techniques are available for calculating the minimum buffer requirements such that there are no deadlocks [9], [1], while other calculate buffer requirements for *rate-optimal* schedules [11], [19]. Between these two extremes, other techniques explore trading of throughput and buffer size [23], [27], [28]. However, none of these techniques takes platforms with complex memory hierarchies into account, such as today's embedded devices. For such platforms the actual execution times of actors might depend on whether data is mapped onto a fast scratchpad memory or a slower flash memory. Therefore, the buffer size throughput trade-off also depends on the sizes of scratchpads.

The concept of letting buffers share the same memory space has been studied in [20] on a coarse grain level, where buffers with non-overlaying lifetimes can share the same memory space. In [8] on a fine grain level where even buffers with overlapping lifetimes are allowed to share the same memory space. These techniques have been shown to significantly reduce the overall memory requirements of streaming applications. However, our goal is to provide users with better energy efficiency and quality of service. Therefore we study the impact of such an optimization in the trade-off between execution time and energy.

B. Scratchpad allocation techniques

The problem of content selection for scratchpads has been extensively studied for C-like application models. [13] presents a survey of scratchpad allocation techniques. Most of these techniques suffer from the lack of information about the program structure [3] and are sometimes not applicable to applications with non-affine accesses, pointers, passing by reference, dynamic assignments etc. Therefore the source code usually needs to be 'cleaned' before these techniques could be applied [18]. In contrast to all these approaches we propose a scratchpad allocation technique at the level of dataflow graphs.

The literature on scratchpad allocation for dataflow graphs is quite limited [6], [4], [3]. In [6] a scratchpad aware scheduling technique is presented that maps both code and data segments of an application described as a dataflow diagram. The trade-off between context switches and buffer sizes is modeled but parallelism and energy costs are not considered. [4] and [3] present methodologies to dynamically allocate code and data of a Hetrochronous Dataflow Graph to scratchpads in a Harvard Architecture, but buffer sizes are not explored.

VIII. Conclusion & Future Work

In this paper, our energy aware technique for mapping SDFGs to scratchpad based MPSoCs called SAMOSA is presented. Tasks are mapped onto different processors whereas data buffers are mapped onto various types of memories. The buffer dimensioning is scratchpad aware and tradeoffs with context switches and parallelism are explored. The content selection criteria for scratchpads in a multiprocessor system implicitly considers data access densities of buffers as well as the lengths of different execution paths.

We used SAMOSA to explore the trade-off between energy and execution time for mapping the H.264 decoder to a scratchpad based MPSoC. Our results show that scratchpad sizes significantly affect the trade-off between energy and execution time Fig. 9. Therefore, we conclude that scratchpad awareness in energy-aware mapping of streaming applications to MPSoCs is important. Our results also show that letting buffers share the same memory space allows significantly better mappings on the energy-time trade-off for a scratchpad based system Fig. 10.

In the future we would like to use bio-inspired multi-objective evolutionary techniques and heuristics for a faster exploration phase and apply SAMOSA to more complex applications and platforms. We aim to extend this technique for platforms with multi hop communication, such as Network-on-Chip based platforms. We also intend to apply the concept of system-scenarios based design [10] to deal with dynamism in the application and use Polyhedral Process Networks [25] as application model for more accurate information data access patterns.

IX. 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 K.U.Leuven.

References

[1] M. Ade, R. Lauwereins, and J. Peperstraete, "Data Memory Minimisation For Synchronous Data Flow Graphs Emulated On DSP-FPGA Targets," in *Design Automation Conference, 1997. Proceedings of the 34th*, jun 1997, pp. 64 –69.

[2] R. Banakar, S. Steinke, B.-S. Lee, M. Balakrishnan, and P. Marwedel, "Scratchpad memory: design alternative for cache on-chip memory in embedded systems," in *Proceedings of the tenth international symposium on Hardware/software codesign*, ser. CODES '02. New York, NY, USA: ACM, 2002, pp. 73–78. [Online]. Available: http://doi.acm.org/10.1145/774789.774805

[3] S. Bandyopadhyay, "Automated memory allocation of actor code and data buffer in heterochronous dataflow models to scratchpad memory," Master's thesis, EECS Department, University of California, Berkeley, Aug 2006. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-105.html

[4] S. Bandyopadhyay, T. H. Feng, H. D. Patel, and E. A. Lee, "A scratchpad memory allocation scheme for dataflow models," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2008-104, Aug 2008. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-104.html

[5] S. S. Bhattacharyya, G. Brebner, J. W. Janneck, J. Eker, C. von Platen, M. Mattavelli, and M. Raulet, "OpenDF: a dataflow toolset for reconfigurable hardware and multicore systems," *SIGARCH Comput. Archit. News*, vol. 36, pp. 29–35, June 2009. [Online]. Available: http://doi.acm.org/10.1145/1556444.1556449

[6] W. Che and K. Chatha, "Scheduling of synchronous data flow models on scratchpad memory based embedded processors," in *Computer-Aided Design (ICCAD), 2010 IEEE/ACM International Conference on*, nov. 2010, pp. 205 –212.

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

[8] M. H. Foroozannejad, M. Hashemi, T. L. Hodges, and S. Ghiasi, "Look into details: the benefits of fine-grain streaming buffer analysis," in *Proceedings of the ACM SIGPLAN/SIGBED 2010 conference on Languages, compilers, and tools for embedded systems*, ser. LCTES '10. New York, NY, USA: ACM, 2010, pp. 27–36. [Online]. Available: http://doi.acm.org/10.1145/1755888.1755894

[9] M. Geilen, T. Basten, and S. Stuijk, "Minimising buffer requirements of synchronous dataflow graphs with model checking," in *Proceedings of the 42nd annual Design Automation Conference*, ser. DAC '05. New York, NY, USA: ACM, 2005, pp. 819–824. [Online]. Available: http://doi.acm.org/10.1145/1065579.1065796

[10] S. V. Gheorghita, M. Palkovic, J. Hamers, A. Vandecappelle, S. Mamagkakis, T. Basten, L. Eeckhout, H. Corporaal, F. Catthoor, F. Vandeputte, and K. D. Bosschere, "System-scenario-based design of dynamic embedded systems," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 14, pp. 3:1–3:45, January 2009. [Online]. Available: http://doi.acm.org/10.1145/1455229.1455232

[11] R. Govindarajan, G. R. Gao, and P. Desai, "Minimizing buffer requirements under rate-optimal schedule in regular dataflow networks," *J. VLSI Signal Process. Syst.*, vol. 31, pp. 207–229, July 2002. [Online]. Available: http://portal.acm.org/citation.cfm?id=598549.598651

[12] W. Heirman, D. Stroobandt, N. R. Miniskar, R. Wuyts, and F. Catthoor, "Pincomm: Characterizing intra-application communication for the many-core era," *Parallel and Distributed Systems, International Conference on*, vol. 0, pp. 500–507, 2010.

[13] M. Idrissi Aouad and O. Zendra, "A Survey of Scratch-Pad Memory Management Techniques for low-power and -energy," in *2nd ECOOP Workshop on Implementation, Compilation, Optimization of Object-Oriented Languages, Programs and Systems (ICOOOLPS'2007)*. Berlin Allemagne: ECOOP, 2007, pp. 31–38.

[14] A. Jantsch, *Modeling Embedded Systems and SoC's: Concurrency and Time in Models of Computation (Systems on Silicon)*, 1st ed. Morgan Kaufmann, Jun. 2003.

[15] P. Laborie, "IBM ILOG CP Optimizer for Detailed Scheduling Illustrated on Three Problems," in *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, ser. Lecture Notes in Computer Science, W.-J. van Hoeve and J. Hooker, Eds. Springer Berlin / Heidelberg, 2009, vol. 5547, pp. 148–162. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-01929

[16] J. Laurent, N. Julien, E. Senn, and E. Martin, "Functional level power analysis: An efficient approach for modeling the power consumption of complex processors," in *Proceedings of the conference on Design, automation and test in Europe - Volume 1*, ser. DATE '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 10 666–. [Online]. Available: http://portal.acm.org/citation.cfm?id=968878.968987

[17] E. A. Lee and D. G. Messerschmitt, "Synchronous data flow," *Proceedings of the IEEE*, vol. 75, no. 9, pp. 1235 – 1245, sept. 1987.

[18] J.-Y. Mignolet and R. Wuyts, "Embedded multiprocessor systems-on-chip programming," *Software, IEEE*, vol. 26, no. 3, pp. 34 –41, may-june 2009.

[19] O. Moreira, T. Basten, M. Geilen, and S. Stuijk, "Buffer sizing for rate-optimal single-rate data-flow scheduling revisited," *Computers, IEEE Transactions on*, vol. 59, no. 2, pp. 188 –201, feb. 2010.

[20] P. Murthy and S. Bhattacharyya, "Shared buffer implementations of signal processing systems using lifetime analysis techniques," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 20, no. 2, pp. 177 –198, feb 2001.

[21] J. Pino and E. Lee, "Hierarchical static scheduling of dataflow graphs onto multiple processors," *Acoustics, Speech, and Signal Processing, IEEE International Conference on*, vol. 4, pp. 2643–2646, 1995.

[22] A. Sinha and A. P. Chandrakasan, "Jouletrack: a web based tool for software energy profiling," in *Proceedings of the 38th annual Design Automation Conference*, ser. DAC '01. New York, NY, USA: ACM, 2001, pp. 220–225. [Online]. Available: http://doi.acm.org/10.1145/378239.378467

[23] S. Stuijk, M. Geilen, and T. Basten, "Throughput-buffering trade-off exploration for cyclo-static and synchronous dataflow graphs," *Computers, IEEE Transactions on*, vol. 57, no. 10, pp. 1331 –1345, oct. 2008.

[24] W. Thies, M. Karczmarek, and S. Amarasinghe, "Streamit: A language for streaming applications," in *International Conference on Compiler Construction*, Grenoble, France, Apr 2002. [Online]. Available: http://groups.csail.mit.edu/commit/papers/02/streamit-cc.pdf

[25] S. Verdoolaege, "Polyhedral process networks," in *Handbook of Signal Processing Systems*, S. S. Bhattacharyya, E. F. Deprettere, R. Leupers, and J. Takala, Eds. Springer US, 2010, pp. 931–965.

[26] M. A. Viredaz and D. A. Wallach, "Power evaluation of a handheld computer," *Micro, IEEE*, vol. 23, no. 1, pp. 66 – 74, jan/feb 2003.

[27] Y. Yang, M. Geilen, T. Basten, S. Stuijk, and H. Corporaal, "Exploring trade-offs between performance and resource requirements for synchronous dataflow graphs," in *Embedded Systems for Real-Time Multimedia, 2009. ESTIMedia 2009. IEEE/ACM/IFIP 7th Workshop on*, oct. 2009, pp. 96 –105.

[28] E. Zitzler, J. Teich, and S. S. Bhattacharyya, "Evolutionary algorithms for the synthesis of embedded software," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 8, pp. 452–456, August 2000. [Online]. Available: http://portal.acm.org/citation.cfm?id=349683.358382