Noname manuscript No.
(will be inserted by the editor)

Exploration of energy efficient memory organisations for dynamic multimedia applications using system scenarios

Iason Filippopoulos · Francky Catthoor · Per Gunnar Kjeldsberg

Received: date / Accepted: date

Abstract We propose a memory-aware system scenario approach that exploits variations in memory needs during the lifetime of an application in order to optimize energy usage. Different system scenarios capture the application's different resource requirements that change dynamically at run-time. In addition to computational resources, the many possible memory platform configurations and data-to-memory assignments are important system scenario parameters. In this work we focus on clustering of different memory requirements into groups and the system scenario generation is presented in detail. The clustering is a non-trivial problem due to the high volume of different memory requirements, which leads to a very large exploration space. An extended memory model is used as a practical enabler, in order to evaluate the methodology. The memory models includes existing state-of-the-art memories, available from industry and academia, and we show how they are employed during the system design exploration phase. Both commercial SRAM and standard cell based memory models are explored in this study. The effectiveness of the proposed methodology is demonstrated and tested using a large set of multimedia benchmarks published in the Polybench, Mibench and Mediabench suites. The broad set of multimedia benchmarks is representative for the entire domain of multimedia applications. Reduction in energy consumption in the memory subsystem ranges from 35% to 55% for the chosen set of benchmarks.

Keywords System Scenarios · Design Space Exploration · Reconfigurable Design · Memory Reconfiguration · Dynamic Multimedia Applications

Iason Filippopoulos, Per Gunnar Kjeldsberg
Department of Electronics and Telecommunications
Norwegian University of Science and Technology (NTNU), Norway
E-mail: iason.filippopoulos, pgk @iet.ntnu.no

Francky Catthoor
IMEC, Belgium

1 Introduction

Modern embedded systems are becoming more and more powerful as the semiconductor processing techniques keep increasing the number of transistors on a single chip. Consequently, demanding applications, e.g., in the signal processing and multimedia domains, can be executed on these devices [23]. On the other hand, the desired performance has to be delivered with minimum power consumption due to the limited energy available in mobile devices [18]. System scenario methodologies propose the use of different platform configurations in order to exploit run-time variations in computational and memory needs often seen in such applications [18].

Platform reconfiguration is performed through tuning of different system parameters, also called system knobs. For the memory-aware system scenario methodology, a platform can be reconfigured through a number of potential knobs, each resulting in different performance and power consumption in the memory subsystem. Foremost, modern memories support different energy states, e.g., through power gating techniques and by switching to lower power modes when not accessed. The second platform knob is the assignment of data to the available memory banks. The data assignment decisions affect both the energy per access for the mapped data, the data conflicts as a result of suboptimal assignment, and the number of active banks. In this work a reconfigurable memory platform is constructed using detailed memory models. This is followed by experiments with dynamic multimedia applications in order to study the effectiveness of the methodology.

The main contribution of the current work is the development of data variable [12] based system scenarios. Previous control variable based system scenarios [8] are unable to handle the fine-grain behaviour of the studied multimedia applications due to their significant variation under different execution situations. Furthermore, compared with use case scenario approaches in which scenarios are generated based on a user's behaviour [41], the system scenario methodology focuses on the behaviour of the system to generate scenarios and can, therefore, fully exploit the detailed platform mapping information. Rather than focusing on the processing cores, this work analyses the application of system scenarios on the memory organisation. More specifically this work focuses on the system scenario identification phase of the methodology. The wide range of memory requirements, the amount of different cases and the different frequency in which each case occurs, results in a very large exploration space. Therefore, there is a need for developing an algorithmic approach that can efficiently tackle this problem.

Another significant contribution is the extensive number of benchmark applications on which the methodology is applied. The chosen set is representative for the entire domain of multimedia applications. Furthermore, we present a categorisation of applications based on their dynamic characteristics, which is also applicable to the entire multimedia domain. For the experimental needs of this work we present for the purpose sufficiently detailed and accurate memory models, which are used for the system design exploration. For the multimedia domain, the current work presents a comprehensive methodology for optimising energy consumption in the memory subsystem.

This article is organized as follows. Section 2 motivates the study of optimization of the memory organisation. Section 3 surveys related work on system level memory exploration and on system scenario methodologies and compares it with the current work. Section 4 presents the chosen methodology with main focus on the memory organisation study. In Section 5 the target platform is described accompanied by a detailed description of the employed memory models, while the multimedia benchmarks and their characteristics are analysed in Section 6. Results of applying the described methodology to the targeted applications are shown in Section 7, while conclusions are drawn in Section 8.

2 Motivational Example

A large number of papers have demonstrated the importance of the memory organization to the overall system energy consumption. As shown in [9] memory contributes around 40% to the overall power consumption in general purpose systems. Especially for embedded systems, the memory subsystem accounts for up to 50% of the overall energy consumption [4] and the cycle-accurate simulator presented in [29] estimates that the energy expenditures in the memory subsystem range from 35% up to 65% for different architectures. The breakdown of power consumption for a recently implemented embedded system presented in [14] shows that the memory subsystem consumes more than 40% of the leakage power on the platform. According to [18], conventional allocation and assignment of data done by regular compilers is suboptimal. Performance loss is caused by stalls for fetching data and data conflicts for different tasks, due to the limited size of memory and the competition between tasks.

In addition, modern applications exhibit more and more dynamic behaviour, which is reflected also in fluctuating memory requirements [18]. Techniques have been developed in order to estimate the memory size requirements of applications in a systematic way [16]. The significant contribution that the memory subsystem has to the overall energy consumption of a system and the dynamic nature of many applications offer a strong motivation for the study and optimization of the memory organisation in modern embedded devices.

To illustrate the sub-optimal conventional allocation and assignment of data, the simple example of Alg. 1 is used. The kernel code of an image processing application continuously reads a sequence of images, saves each image in memory and performs function *func1* on each pixel of the image. Typically arrays are used for storing the intermediate calculations in image processing applications, i.e., the *array* variable in the motivation example. The memory size used for the storage of each initial *image* and the computed *array* are determined by the dimensions of the input image and can be potentially different for a series of input images. In a conventional assignment the highest values of *height* and *width* are identified and a static compiling results in allocation of the worst-case area for the *array* variable. However, only a part of the allocated space is accessed during processing of smaller images.

Assume for instance that we have two different image sizes, ImgA with L=H=1 and ImgB with L=H=2. That is, the size of ImgB is 4 × ImgA. Each pixel in each image is accessed once giving rise to N accesses to each ImgA and 4 × N accesses to each ImgB. Furthermore, in the input stream of images, there are four times as many ImgA as ImgB. In our pool of alternative memories we have three memories with Size1 = ImgA, Size2 = 3 × ImgA, and Size 3 = 4 × ImgA (i.e., the size of ImgB). The energy cost of accessing a Size1 memory is 1E, while the average leakage energy in the time between the start of two accesses is 0.3E. The

Algorithm 1 Motivation example of dynamic memory usage

```
1: while $image \neq EndOfDatabase$ do
2:     $height \leftarrow height(image)$
3:     $width \leftarrow width(image)$
4:     $store(image[height][width])$
5:     for $i = 0 \rightarrow height$ do
6:         for $j = 0 \rightarrow width$ do
7:             $array[i][j] \leftarrow func1(image[i][j])$
8:         end for
9:     end for
10:    $image \leftarrow new.image$
11: end while
```

corresponding access/leakage numbers for Size2 and Size3 memories are 1.3E/0.9E and 1.5E/1.2E, respectively. The numbers reflects the fact that as a first order approximation access energy increases sub linearly with increased memory size, while leakage increases linearly with memory size. The total energy (access + leakage) during computations on four ImgA and one ImB using only one memory of Size3 is:

$$4 \times N \times 1.5E + 4 \times N \times 1.5E + 8 \times N \times 1.2E = 21.6NE$$

The same calculation using one memory of Size1 and one of Size2, in total the size of ImgB, is:

$$\begin{aligned} &4 \times N \times 1.0E + 1 \times N \times 1.0E + 3 \times N \times 1.3E \\ &+ 4 \times N \times 0.3E + 4 \times N \times (0.3E + 0.9E) = 14.9NE \end{aligned}$$

giving a reduction in energy consumption of 31%. These calculations are done with simplified assumptions regarding input data and memory models. The results in Section 7 show even larger gain with realistic dynamic applications, memory models and data.

3 Related Work and Contribution Discussion

Many papers have focused on memory related optimisations, also in the presence of a partitioned and distributed memory organisation with memory blocks of different sizes. In [1] authors present a methodology for automatic memory hierarchy generation that exploits memory access locality, while in [2] they propose an algorithm for the automatic partitioning of on-chip SRAM in multiple banks. Several design techniques for designing energy efficient memory architectures for embedded systems are presented in [19]. The current work differentiates by employing a platform that is reconfigurable during run-time. In [27] a large number of data and memory optimisation techniques, that could be dependent or independent of a target platform, are discussed. Again, reconfigurable platforms are not considered.

Energy-aware assignment of data to memory banks for several task-sets based on the MediaBench suit of benchmarks is presented in [20]. Low energy multimedia applications are discussed also in [5] with focus on processing rather than the memory platform. Furthermore, both [20] and [5] base their analysis on use case situations and do not incorporate sufficient support for very dynamically behaving application codes. System scenarios alleviate this bottleneck and enable handling of such dynamic behaviour. In addition, the current work explores the assignment of data to the memory and the effect of different assignment decisions on the overall energy consumption.

The authors in [30], [34] and [37] present methodologies for designing memory hierarchies. Design methods with main focus on the traffic and latencies in the memory architecture are presented in [31], [33], [35] and [39]. Improving memory energy efficiency based on a study of access patterns is discussed in [36]. Application specific memory design is a research topic in [40], while memory design for multimedia applications is presented in [38]. The current work differentiates by introducing the concept of system scenarios that supports the dynamic handling of application's requirements, although the data mapping is static inside each scenario.

An overview of work on system scenario methodologies and their application are presented in [8]. In [6] extensions towards a memory-aware system scenario methodology are presented and demonstrated using theoretical memory models and two target applications. This work is an extension both in complexity and accuracy of the considered memory library and on the number of target applications.

Furthermore, the majority of the published work focus on control variables for system scenario prediction and selection. Control variables can take a relatively small set of different values and thus can be fully explored. However, the use of data variables [11] is required by many dynamic systems including the majority of multimedia applications. The range of possible values for data variables is wider and makes full exploration impossible.

Authors in [24] present a technique to optimise memory accesses for input data dependent applications by duplicating and optimising the code for different execution paths of a control flow graph (CFG). One path or a group of paths in a CFG form a scenario and its memory accesses are optimised using global loop transformations (GLT). Apart from if-statement evaluations that define different execution paths, they extend their technique to include while loops with variable trip count in [26]. A heuristic to perform efficient grouping of execution paths for scenario creation is analysed in [25]. Our work extends the existing solutions towards exploiting the presence of a distributed memory organisation with reconfiguration possibilities.

Reconfigurable hardware for embedded systems, including the memory architecture, is a topic of active research. An extensive overview of current approaches is found in [7]. The approach presented in this paper differentiates by focusing on the data-to-memory assignment aspects in the presence of a platform with dynamically configurable memory blocks. Moreover, many methods for source code transformations, and especially loop transformations, have been proposed in the memory management context. These methods are fully complementary to our focus on data-to-memory assignment and should be performed prior to our step.

4 Data Variable Based Memory-Aware System Scenario Methodology

The memory-aware system scenario methodology is based on the observation that the memory subsystem requirements at run-time vary significantly due to dynamic variations of memory needs in the application code. Most existing design methodologies define the memory requirements as that of the most demanding task and tune the system in order to meet its needs [18]. Obviously, this approach leads to unused memory area for tasks with lower memory requirements, since those tasks could meet their needs using fewer resources and consequently consuming less energy. Another source of unnecessary waste of energy in the memory is caused by data conflicts due to misplaced data. Replacement of old data and fetching of new data is both time and energy consuming and should therefore be avoided. Handling of data conflicts is also part of a memory-aware system scenario methodology.

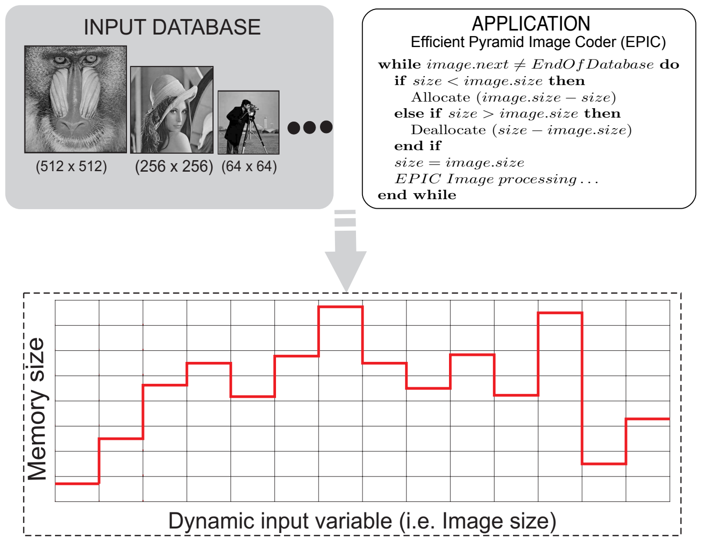
Designing with system scenarios is workload adaptive and offers different configurations of the platform and the freedom of switching to the most efficient scenario at run-time. A system scenario is a configuration of the system that combines similar run-time situations (RTSs). An RTS consists of a running instance of a task and its corresponding cost (e.g., energy consumption) and one complete run of the application on the target platform represents a sequence of RTSs [11]. The system is configured to meet the cost requirements of an RTS by choosing the appropriate system scenario, which is the one that satisfies the requirements using minimal power. In the following subsections, the different steps of the memory-aware system scenario methodology are outlined.

The system scenario methodology follows a two stage exploration, namely design-time and run-time stages, as described in [8]. This splitting is also employed in the memory-aware extension of the methodology. The two stage exploration is chosen because it reduces run-time overhead while preserving an important degree of freedom for run-time configuration [18]. The application is analysed at design-time and different execution paths causing variations in memory demands are identified. This procedure, which is time consuming and as a result can be performed only during the design phase, will result in a grey-box model representation of the application. The grey-box model hides all static and deterministic parts of the application, by providing only related memory costs for those, and keeps parts of the application code that are non-deterministic in terms of memory usage available to the system designer [13].

4.1 Design-time Profiling Based on Data Variables

Application profiling is performed at design-time for a wide range of inputs. The analysis focuses on the allocated memory size during execution and on access pattern variations. Techniques described in [15] are, e.g., used in order to extract the access scheme through analysis of array iteration spaces.

The profiling stage is depicted in Fig. 1 and consists of running the application code with suitable input data often found in a database, in order to produce profiling results. The results shown here are limited for demonstrational purposes. A real application would have thousands or millions of profiling samples. The profiling reveals parts of the application code with high memory activity and with varying memory access intensity, which possibly depends on input data variables. Because of this behaviour, a static study of the application code alone is insufficient since the target applications for this methodology have non-deterministic behaviour that is driven by input.


INPUT DATABASE
(512 x 512)
(256 x 256)
(64 x 64)
APPLICATION
Efficient Pyramid Image Coder (EPIC)
Memory size
Dynamic input variable (i.e. Image size)

```
while $image.next \neq EndOfDatabase$ do
  if $size < image.size$ then
    Allocate ($image.size - size$)
  else if $size > image.size$ then
    Deallocate ($size - image.size$)
  end if
  $size = image.size$
  $EPIC\ Image\ processing \ldots$
end while
```

Fig. 1 Profiling results based on application code and input data

Given code and database with data inputs, profiling will show memory usage during run-time. Results provided to the designer include complete information about allocated memory size values together with the number of occurrences and duration for each of these memory size values. Moreover, correlation between input data variable values and the resulting memory behaviour can possibly be observed. This information is useful for the clustering step that follows.

In Fig. 1 the profiled applications are two image related multimedia benchmarks and the input database should consist of a variety of images. The memory requirements in each case are driven by the current input image size, which is classified as a data variable due to the wide range of its possible values. Depending on the application the whole image or a region of interest is processed. Other applications have other input variables deciding the memory requirement dynamism, e.g., the SNR level on the channel in the case of an encoding/decoding application.

4.2 Design-time System Scenario Identification Based on Data Variables

The next step is the clustering of the profiled memory sizes into groups with similar characteristics. This is referred to as system scenario identification. Clustering is necessary, because it will be extremely costly to have a different scenario for every possible size, due to the number of memories needed. Clustering neighbouring RTSs is a rational choice, because two instances with similar memory needs have similar energy consumption.

In Fig. 2 the clustering of the previously profiled information is presented. The clustering of RTSs is based both on their distance on the memory size axis and the frequency of their occurrence. Consequently, the memory size is split unevenly with more frequent RTSs having a shorter memory size range. In the case of a clustering to three system scenarios the space is divided in the three differently

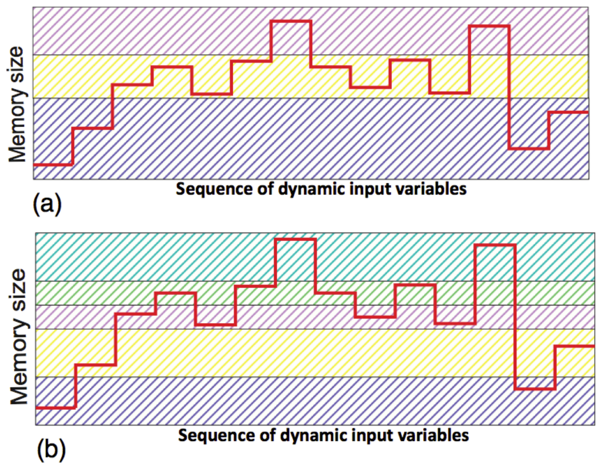


Fig. 2 Clustering of profiling results into three (a) or five (b) system scenarios

coloured hashed areas depicted in Fig. 2(a). Due to the higher frequency of RTSs in the yellow hashed area, that system scenario has a shorter range compared with its neighbouring scenarios. Such clustering is better than an even splitting because the energy cost of each system scenario is defined by the upper size limit, as each scenario should support all RTSs within its range. Consequently the overhead for the RTSs in the yellow area is lower compared to the overhead in the two other areas.

The same principle applies also when the number of system scenarios is increased to five, as depicted in Fig. 2(b). The frequency sensitive clustering results in two short system scenarios that contain four RTSs each and three wider system scenarios with lower numbers of RTSs. The number of system scenarios should be kept limited mainly due to two factors. First, implementation of a high number of system scenarios in a memory platform is more difficult and complex. Second, the switching between the different scenarios involves an energy penalty that could become significant when the switching takes place frequently.

The memory size and the frequency of each RTS are not the only two parameters that should be taken into consideration during the system scenario identification. The memory size of each RTS results in a different energy cost depending on the way it is mapped into memory. The impact of the different assignment possibilities is included into clustering by introduction of energy as a cost metric. The energy cost for each RTS is calculated using a reference platform with one to N memory banks. Increasing the number of memory banks results in lower energy per access since the most accessed elements can be assigned to smaller and more energy efficient banks. Unused banks can be switched off.

In the system scenario methodology, Pareto curves are used to capture alternative system configurations within a scenario [18]. In our work, a Pareto space is used for clustering that also includes the energy cost metric. For each RTS all different assignment options on all alternative platform configurations are studied. A Pareto curve is constructed for each RTS that contains the optimal assignment

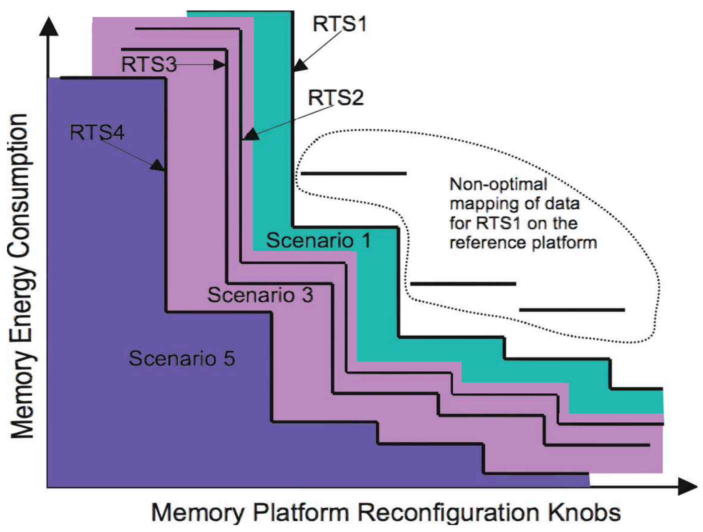

RTS1
RTS2
RTS3
RTS4
Non-optimal mapping of data for RTS1 on the reference platform
Scenario 1
Scenario 3
Scenario 5
Memory Energy Consumption
Memory Platform Reconfiguration Knobs

Fig. 3 Clustering of Pareto curves

for each platform configuration. Hence, suboptimal assignments and assignments that result in conflicts are not included in the Pareto curve. In Fig. 3 four Pareto curves, each corresponding to a different RTS, are shown together with energy cost levels corresponding to different platform configuration and data-to-memory assignment decisions. Three non-optimal mappings are also shown in Fig. 3 for illustration. They are not part of the Pareto curve and consequently not included in the generation of scenarios. Pareto curves are clustered into three different system scenarios based again both on their memory size differences and frequency of occurrence. Clustering of RTSs using Pareto curves is more accurate compared to the clustering depicted in Fig. 2, as it includes data-to-memory assignment options in the exploration.

The system scenario identification step includes the selection of the data variables that determine the active system scenario. This can be achieved by careful study of the application code, combined with the application's data input. The variable selection is done before clustering of RTSs into scenarios. Based on the choice of the identification variable, there is a trade-off between the complexity and the accuracy of the scenario detection step. On the one hand, if the identification is done using a group of complex variables and their correlation, there is a number of calculations needed in order to predict the active scenario. On the other hand, if the value of a single variable is monitored for scenario identification, the scenario detection is straightforward. Obviously, the accuracy of the scenario detection is higher on the first case, while the computational needs for scenario detection are lower on the second case. In other words, the more accurate scenario detection is, the more resources are used by the run-time manager for detection. In our case the grey-box model reveals only the code parts that will influence memory usage, so that data variables deciding memory space changes can be identified. An example of this is a non static variable that influences the number of iterations for a loop that performs one memory allocation at each iteration. In the depicted example the system scenario detection data variable is the input image height and width

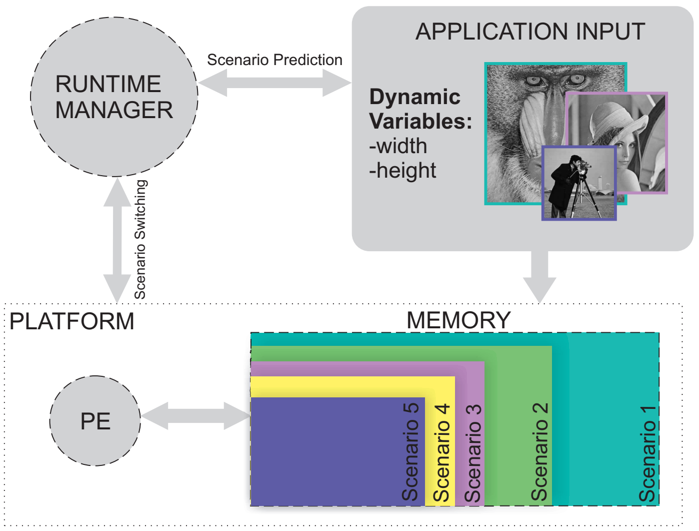


Fig. 4 Run-time system scenario detection and switching based on the current input

values. Moreover, the designer should look for a correlation between input values and the corresponding cost. This information will be useful in the following steps of the methodology [18].

4.3 Run-time System Scenario Detection and Switching Based on Data Variables

Switching decisions are taken at run-time by the run-time manager. In this work, we use a simple and straightforward switching approach. The switching step consists of all platform configuration decisions that can be made at run-time, e.g., frequency/voltage scaling, changing the power mode of memory units, including turning them off, and reassignment of data on memory units. Switching takes place when the switching cost is lower than the energy gains achieved by switching. In more detail, the run-time manager compares the memory energy consumption of executing the next task in the current active system scenario with the energy consumption of execution with the optimal system scenario. If the difference is greater than the switching cost, then scenario switching is performed [18]. Switching costs are defined by the platform and include all memory energy penalties for run-time reconfigurations of the platform, e.g., extra energy needed to change state of a memory unit.

In Fig. 4 an example of the run-time phase of the methodology is depicted. The run-time manager identifies the size of the image that will be processed and reconfigures the memory subsystem on the platform, if needed, by increasing or decreasing the available memory size. The reconfiguration options are effected by platform hardware limitations. The image size is the data variable monitored in order to detect the system scenario and the need for switching.

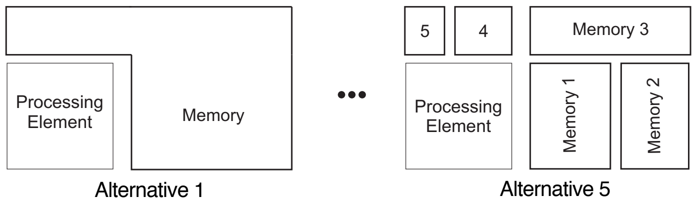


Fig. 5 Alternative memory platforms with varying number of banks

5 Target Platform and Energy Models

Selection of target platform is an important aspect of the memory-aware system scenario methodology. The key feature needed in the platform architecture is the ability to efficiently support different memory sizes that correspond to the system scenarios generated by the methodology. Execution of different system scenarios then leads to different energy costs, as each configuration of the platform results in a specific memory energy consumption. The dynamic memory platform is achieved by organising the memory area in a varying number of banks that can be switched between different energy states.

5.1 Target Memory Platform Architecture

In this work, a clustered memory organisation with up to five memory banks of varying sizes is explored. The limitation in the number of memory banks is necessary in order to keep the interconnection cost between the processing element (PE) and the memories constant through exploration of different architectures.

For more complex architectures the interconnection cost should be considered and analysed separately for accurate results. Although power gating can be applied to the bus when only a part of a longer bus is needed, an accurate model of the memory wrapper and interconnection must developed, which is beyond the scope of the current work.

Some examples of alternative memory platforms that can be used for exploration is shown in Fig. 5. Point-to-point connections with negligible interconnect costs between elements are assumed for up to five memory banks.

5.2 Models of Different Memory Types

The dynamic memory organisation is constructed using commercially available SRAM memory models (MM). For those models delay and energy numbers are derived from a commercial memory compiler. In addition, experimental standard cell-based memories (SCMEM) [22] are considered for smaller memories due to their energy and area efficiency for reasonably small storage capacities, as argued in [21]. The standard cell-based memories are synthesized using Cadence RTL compiler for TSMC 40nm standard library. Afterwords, power simulations on the synthesized design are carried out using Synopsys PrimeTime, in order to obtain energy numbers. Both MMs and SCMEMs can operate under a wide range of supply voltages, thus support different operating modes that provide an important exploration space.

- Active mode: The normal operation mode, in which the memory can be accessed at the maximum supported speed. The supply voltage is 1.1V. The dynamic and leakage power are higher compared to the other modes. Only on active mode the data are accessible without time penalties, in contrast to light and deep sleep modes. In this work all the memory accesses are performed on the active mode.
- Light sleep mode: The supply voltage in this mode is lower than active with values around 0.7V. The access time of the memory is significantly higher than the access time in active mode. Switching to active mode can be performed with a negligible energy penalty and a small time penalty of a few clock cycles (less than 10). Data is retained.
- Deep sleep mode: The supply voltage is set to the lowest possible value that can be used without loss of data. This voltage threshold is expected to be lower for SCMEMs than MM models and can be as low as 0.3V. The number of clock cycles needed for switching to active mode is higher compared to sleep mode, typically in the range of 20 to 50 clock cycles depending on the clock speed. Consequently, the speed of the PE and the real-time constrains of the applications has to be taken into consideration when choosing light or deep sleep mode at a specific time.
- Shut down mode: Power-gating techniques are used to achieve near zero leakage power. Stored data is lost. The switch to active mode requires substantially more energy and time. However, switching unused memories to this mode, providing that their data are not needed in the future, results in substantial energy savings.

The necessary energy/power information is available to the system designer and relative values for a subset of the used sizes in the current work are presented in Tab. 1 and in Tab. 2. It shows that the choice of memory units has an important impact on the energy consumption. Moreover, different decisions have to be made based on the dominance of dynamic or leakage energy in a specific application. In the current work memory architectures with 1 to 5 memory units of different sizes are explored and the optimal configuration is chosen. The methodology is in general not restricted to specific memory types or benchmarks and can handle more complex hierarchical memory architectures and applications. However, in this study the chosen applications have a relatively small memory space requirement limited to around 100KB, which is the case for many applications run on modern embedded systems.

5.3 Total Energy Consumption Calculation

Both the dynamic and the static energy consumed in the memory subsystem is included in the calculations. The overall energy consumption for each configuration

Table 1 Relative dynamic energy for a range of memories with varying capacity and type

Type	**Lines x wordlength**	**Dynamic Energy [J]**		Switching to Active from	
		Read	Write	Deep[uJ]	Light[uJ]
MM	32 x 8	4.18×10^{-8}	3.24×10^{-8}	0.223	0.031
MM	32 x 16	6.79×10^{-8}	5.89×10^{-8}	0.223	0.031
MM	32 x 128	4.33×10^{-7}	4.31×10^{-7}	1.42	0.168
MM	256 x 128	4.48×10^{-7}	4.60×10^{-7}	1.70	0.171
MM	1024 x 128	5.11×10^{-7}	5.75×10^{-7}	2.81	0.179
MM	4096 x 128	9.60×10^{-7}	4.57×10^{-7}	9.01	0.457
SCMEM	128 x 128	2.5×10^{-7}	0.8×10^{-8}	1.51	0.045
SCMEM	1024 x 8	1.7×10^{-8}	0.6×10^{-8}	0.325	0.021

Table 2 Relative static power for a range of memories with varying capacity and type

Type	**Lines x wordlength**	**Static Leakage Power per Mode[W]**			
		Active	Light-sleep	Deep-sleep	Shut-down
MM	32 x 8	0.132	0.125	0.063	0.0016
MM	32 x 16	0.134	0.127	0.064	0.0022
MM	32 x 128	0.171	0.160	0.083	0.0112
MM	256 x 128	0.207	0.184	0.104	0.0293
MM	1024 x 128	0.349	0.283	0.189	0.102
MM	4096 x 128	0.95	0.708	0.544	0.396
SCMEM	128 x 128	0.083	0.057	0.027	0.0022
SCMEM	1024 x 8	0.042	0.028	0.014	0.0011

is calculated using a detailed formula, as can be seen in Eq.1.

$$\begin{aligned}
E = \sum_{memories}^{all} &(N_{rd} \times E_{Read} \\
&+N_{wr} \times E_{Write} \\
&+(T - T_{LightSleep} - T_{DeepSleep} - T_{ShutDown}) \times P_{leak_{Active}} \\
&+T_{LightSleep} \times P_{leak_{LightSleep}} \\
&+T_{DeepSleep} \times P_{leak_{DeepSleep}} \\
&+T_{ShutDown} \times P_{leak_{ShutDown}} \\
&+N_{SW\,Light} \times E_{LightSleep\,to\,Active} \\
&+N_{SW\,Deep} \times E_{DeepSleep\,to\,Active} \\
&+N_{SW\,ShutDown} \times E_{ShutDown\,to\,Active})
\end{aligned} \tag{1}$$

All the important transactions on the platform that contribute to the overall energy are included, in order to achieve as accurate results as possible. In particular:

- N_{rd} is the number of read accesses
- E_{Read} is the energy per read
- N_{wr} is the number of write accesses
- E_{Write} is the energy per write
- T is the execution time of the application
- $T_{LightSleep}$, $T_{DeepSleep}$ and $T_{ShutDown}$ are the times spent in light sleep, deep sleep and shut down states respectively

- $P_{leak_{Active}}$ is the leakage power in active mode
- $P_{leak_{LightSleep}}$, $P_{leak_{DeepSleep}}$ and $P_{leak_{Shutdown}}$ are the leakage power values in light sleep, deep sleep and shut down modes with different values corresponding to each mode
- $N_{SWLight}$, N_{SWDeep} and $N_{SWShutDown}$ are the number of transitions from each retention state to active state
- $E_{LightSleep\,to\,Active}$, $E_{DeepSleep\,to\,Active}$ and $E_{ShutDown\,to\,Active}$ are the energy penalties for each transition respectively.

The overall energy consumption is given after calculating the energy for each memory bank. The execution time of the application is needed to calculate the leakage time. It can be found by executing the application on a reference embedded processor. The simulator described in [3] is chosen to calculate execution time for the chosen applications in this work. The processor is assumed to be running continuously, accepting new input data as soon as computations on the previous data set has been finished. Memory sleep times are hence only caused by data dependent dynamic behaviour.

5.4 Memory Architecture Exploration

The exploration of alternative memory platforms is performed using the steps described in Alg. 2. All potentially energy efficient configurations are tested for a given number of scenarios and the sequence of RTSs of the application. First, all possible configurations for a given number of memory banks are constructed. The only requirement in order to keep a configuration for further investigation is that the combined size of all banks should satisfy the storage requirements of the most demanding RTS. Then, each configuration is tested for the sequence of RTSs and the one that minimizes Eq.1 is chosen as the most energy efficient for this number of scenarios (i.e., number of banks).

Algorithm 2 Memory organisation exploration steps

```
1: $RTSset \leftarrow$ storage requirement for each RTS
2: $Database \leftarrow$ memory database
3: $N \leftarrow$ number of scenarios (up to 5 in this work)
4: for $i = 1 \rightarrow N$ do
5:     for all combinations of i banks in database do
6:         if $\sum_1^i size(bank) \geq size(max(RTS))$ then
7:             Keep configuration
8:         end if
9:         Select configuration that minimizes Eq.1 for $RTSset$
10:    end for
11: end for
```

6 Application Benchmarks

The applications that benefit most from the memory-aware system scenario methodology are characterised by having dynamic utilization of the memory organisation

Table 3 Benchmark applications overview

Index	Name	Source	Scenario detection variable
1	Epic image compression	MediaBench	Image size
2	Motion Estimation	MediaBench	Image size
3	Blowfish decoder	MiBench	Input file size
4	Jacobi 1D Decomposition	Polybench	Number of steps
5	Mesa 3D	MediaBench	Loop bound
6	JPEG DCT	MediaBench	Block size
7	PGP encryption	MediaBench	Encryption length
8	Viterbi encoder	Open	Constraint length

during their execution. Multimedia applications often exhibit such a dynamic variation in memory requirements during their lifetime and consequently are suitable candidates for the presented methodology. The effectiveness is demonstrated and tested using a variety of open multimedia benchmarks, which can be found in the Polybench [28], Mibench [10] and Mediabench [17] benchmark suites. The broad set of multimedia benchmarks under exploration is representative for the entire domain of multimedia applications.

6.1 Presentation of Multimedia Benchmark Applications and Corresponding Input Databases

An overview of the benchmark applications that were tested is presented in Tab. 3.

Two key parameters under consideration are the dynamic data variable of each application and the variation in the memory requirement it causes. The dynamic data variable is the variable that results in different system scenarios due to its range of values. Examples of such a variable are an input image of varying size or data dependent loop bound values. For each application an appropriate set of realistic RTS cases is constructed. The memory size limits are defined as the minimum and maximum storage requirement occurred during testing of an application.

EPIC (Efficient Pyramid Image Coder) image compression can compress all possible sizes of images. The size of the input image has an effect on memory requirements during compression and several images were given as inputs. *Motion estimation* is another media application in which image size is the dynamic data variable. In this case the image defines the area that has to be explored to determine the motion vectors and different images are tested. The set of input data is constructed using publicly available images commonly used for testing this algorithm.

The dynamism in the *blowfish decoder* benchmark is a result of variations in the input file that is decoded. Again, the methodology explores the behaviour for several input files in order to identify system scenarios. The *Jacobi 1D decomposition* algorithm can be executed using a varying number of steps with a direct effect on memory usage and is hence another suitable benchmark for the system scenario methodology. The set of input data is selected in a way that the number of steps is increased on every next iteration. *Mesa 3D* is an open graphics library with a dynamic loop bound in its kernel that provides the desired dynamic behaviour.

The discrete cosine transformation (DCT) block used in the *JPEG compression* algorithm has a memory footprint that is heavily influenced by the block size. The

Table 4 Characterization of benchmark applications (See Tab. 3 for index

Index	**Dynamic Characteristics**		
	Memory Variation(B)	Width of histogram	Shape of histogram
1	4257 - 34609	Average	Right skewed
2	4800 - 52800	High	Gaussian-like
3	256 - 5120	Low	Left skewed
4	502 - 32002	Low	Right skewed
5	5 - 50000	High	Gaussian-like
6	10239 - 61439	High	Gaussian-like
7	3073 - 49153	High	Gaussian-like
8	5121 - 14337	Low	Right skewed

input database consists of an ascending size sequence of blocks. For the *PGP encryption* algorithm the encryption length parameter has an important impact on memory size, which can be exploited using system scenarios. Thus, we create a database starting from the lowest encryption length value of 384 and gradually increasing it up to 2048. The effect of the channel SNR level on the constraint length value of the *Viterbi encoder* algorithm is discussed in [6]. Increasing noise on the channel demands a more complex encoding in order to maintain a constant bit error rate (BER), which consequently increases the memory requirements during execution. The memory size variation is given for execution under different SNR levels.

6.2 Classification of Applications Based on Dynamic Characteristics

The required dynamism for applying the memory-aware system scenario methodology can be produced by several code characteristics, covering a wide range of potential applications, as discussed in the previous subsection. In this subsection dynamic characteristics are outlined that can assist the system designer in the employment of the methodology and reveal the expected behaviour prior to experimentation with an application. The dynamic characteristics that are used to categorize the applications are the dynamism in the memory size bounds and the variance of cases within the memory size limits. The characterization of the benchmark applications based on those key parameters is presented in Tab. 4.

The profiling information can be organised into a histogram in order to be easily comprehensive. The horizontal axis depicts the memory requirements for the RTSs and the vertical axis the number of occurrences for its RTS. In this way the system designer can quickly identify the expected gains of the system scenario methodology by identifying the width and the shape of the histogram. The width of the histogram gives an overview of the memory size bounds, while the shape reveals the variation of the RTSs for the studied application.

The memory size bounds correspond to the minimum and maximum memory size values profiled over all possible cases. In general, larger distances between upper and lower bounds increase the possibilities for energy gains. This is a result of using larger and more energy hungry memories in order to support the memory requirements for the worst case even when only small memories are required. Large energy gain is expected when large parts of the memory subsystem can be switched into retention for a long time. For several of the benchmarks the difference between maximum and minimum memory size is close to 50KB. This includes JPEG, motion estimator, mesa 3D, and PGP, where large gains can be expected. On the other hand, the system designer should expect lower energy gains for applications that show a relatively less dynamic behaviour with regard to their memory size limits. Examples here are the blowfish and viterbi algorithms.

The variation takes into consideration both the number of different cases that are present within the memory requirement limits and the distribution of those cases between minimum and maximum memory size. This variation corresponds to the shape of the histogram of the application. Applications with a limited number of different cases are expected to have most of its possible gain obtained with a few platform supported system scenarios and much smaller energy gains from additional system scenarios. After this point most of the cases are already fitting one of the platform configurations and adding new configurations have a minimal impact. The opposite is seen for applications that feature a wide range of well distributed cases.

Based on the analysis above, applications can be classified into four main categories. The first category has a histogram with a wide range of RTS memory sizes and most RTSs placed to the left. This category is defined as a right skewed distribution, according to the direction of the tail. Inb this category the RTSs corresponding to large memories are rarely used, so high gains are expected by applying the methodology. The second case is when there is a close to Gaussian RTS distribution in the histogram. The expected gains are here lower than for the first category. The opposite of the first category is when the histogram is left skewed, meaning that the RTSs with the higher memory requirements are dominant. In this category, system designer should expect the smallest gains. The forth case is when all the RTSs have the same memory requirements and there is no distribution on the histogram, which results in no energy gains for the methodology.

7 Results

The memory aware system scenario methodology is applied to all the presented benchmark applications to study its effectiveness. The profiling phase is based on different input for the data variables shown in Tab. 3 and is followed by the clustering phase. The execution and sleep times needed in Eq.1 are found through the profiling but are also reflected by the dynamic characteristics in Tab. 4. Data variables are the variables used by the run-time manager in order to predict the next active scenario. The clustering is performed with one to five system scenarios. All potentially energy efficient configurations are tested for a given number of scenarios using the steps described in Alg. 2. For example, in the case of 2 scenarios all possible memory platforms with 2 memory banks that fulfil the memory size requirement of the worst case are generated and tested. The same procedure is performed for 3, 4 and 5 scenarios. The exploration includes memories of different sizes, technologies and varying word lengths.

The energy gain percentages are presented in Fig. 6. Energy gains are compared to the case of a fixed non-re-configurable platform, i.e., a static platform configuration with only 1 scenario. This corresponds to zero percentage gain in Fig. 6.

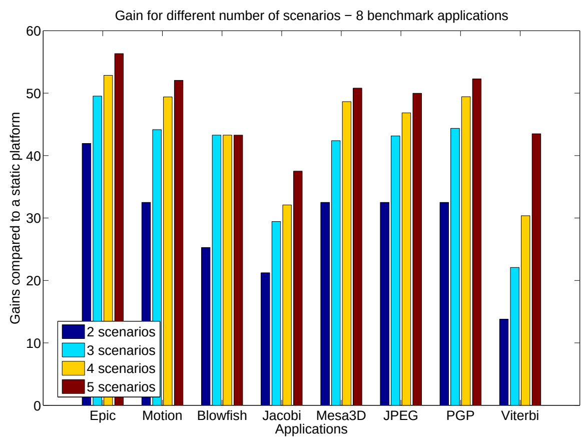


Fig. 6 Energy gain for increasing number of system scenarios - Static platform corresponds to 0%

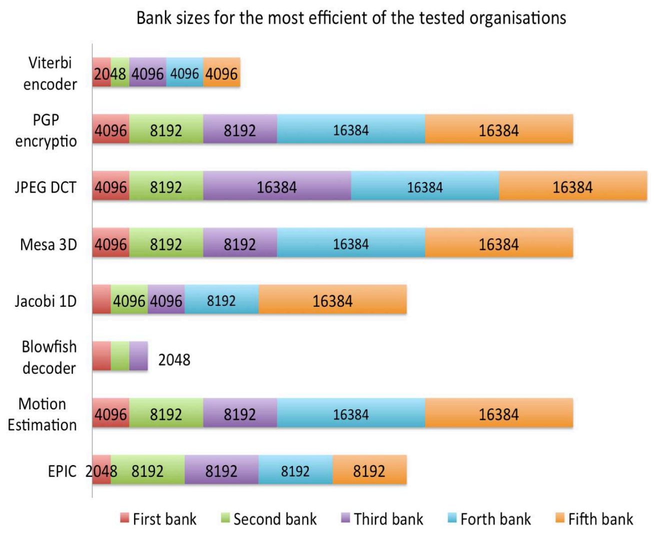


Fig. 7 Bank sizes for the most efficient of the tested organisations for each benchmark

The most efficient of the tested organisations for each benchmark are presented in Fig. 7, where each memory bank is depicted with a different colour and each length is proportional to the memory bank size. The blowfish decoder is the only benchmark that has only 3 banks in its most efficient memory organisation. In Tab. 5 the minimum and maximum energy gains for each benchmark application are shown.

Table 5 Range of energy gains on the memory subsystem

EPIC		**Motion**		**Blowfish**		**Jacobi**	
Min	Max	Min	Max	Min	Max	Min	Max
41.9%	56.3%	32.4%	52.1%	25.3%	43.3%	21.2%	37.5%
Mesa3D		**JPEG**		**PGP**		**Viterbi**	
Min	Max	Min	Max	Min	Max	Min	Max
32.5%	50.8%	33.0%	49.9%	32.2%	52.3%	13.8%	43.5%

7.1 Classification of the Applications

The introduction of a second system scenario results in energy gains between 15% and 40% for the tested applications. Depending on the application's dynamism the maximum reported energy gains range from around 35% to 55%. As expected according to the categorisation presented in subsection 6.2, higher energy gains are achieved for applications with more dynamic memory requirements, i.e., bigger difference between the minimum and maximum allocated size. The maximum gains for JPEG, motion estimator, mesa 3D and PGP are around 50% while blowfish, jacobi, and Viterbi decoders are around 40%.

As the number of system scenarios that are implemented on the memory subsystem increases, the energy gains improve since variations in memory requirements can be better exploited with more configurations. However, the improvement with increasing numbers of system scenarios differ depending on the kind of dynamism present in each application. The application with the highest variation in distribution of memory requirements is the Viterbi encoder/decoder and gains around 10% is seen for every new memory bank added, even for a platform growing from four to five banks. In contrast, the application with the lowest number of different cases, blowfish, cannot further exploit a platform with more than three banks. Another case in which smaller energy gains are achieved, after a certain number of platform supported system scenarios have been reached, is the PGP encryption algorithm. In this benchmark the introduction of more scenarios has an energy impact of less than 5% after the limit of three system scenarios has been reached.

7.2 Switching Overhead

The switching cost increases for an increasing number of system scenarios due to the increasing frequency of platform reconfiguration. This overhead reduces the achieved gain, but for up to 5 scenarios we still see improvements for all but one of our benchmarks. The switching cost is below 2% even for a platform with 5 memory banks in all cases. Apart from the number of scenarios, the switching cost depends on the sensitivity of the variable used for scenario identification. A change of value on the identification variable indicates potentially a new scenario. For an increasing frequency of changes, the switching cost increases.

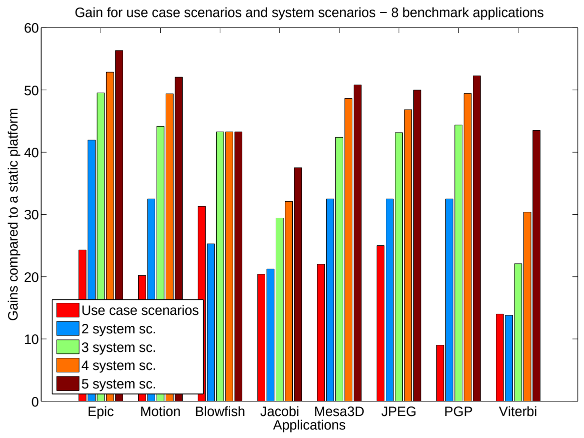


Fig. 8 Energy gain for use case scenarios and system scenarios

7.3 Comparison with Use Case Scenario

Comparative results from applying a use case scenario approach as a reference are presented in Fig. 8. Reported energy gains for both use case scenarios and system scenarios are given assuming a static platform as a base (0%). Use case scenarios are generated based on a higher abstraction level that is visible as a user's behaviour. For example, use case scenarios for image processing applications generate three scenarios, if large, medium and small are the image sizes identified by the user. Similarly, use case scenarios for JPEG compression identify only low and high compression as options and motion estimation is performed on I, P and B video frames, without exploring fine grain differences inside a frame. In general, use case scenario identification can be seen as more coarse compared to identification on the detailed system implementation level. As seen in Fig. 8 the use case gains are superior only to a static platform and for two benchmarks to a platform with only two scenarios.

7.4 Run-Time Overhead

The reported energy gains are for the memory subsystem. As motivated in Section 2 this has previously been shown to be a major contributor to the total energy consumption. An additional energy overhead from the system scenario approach can be found in the processor performing the run-time system scenario detection and switching. This overhead is partly incorporated in $E_{SleepActive}$, in particular if traditional system scenarios are already implemented so that the only overhead is the addition of memory-awareness. The run-time overhead is kept low, because the run-time manager is active for less than 1% of the time needed for the execution of the application.

8 Conclusions

The scope of this work is to apply the memory-aware system scenario methodology to a wide range of multimedia application and test its effectiveness based on an extensive memory energy model. A wide range of applications is studied that allow us to draw conclusions about different kinds of dynamic behaviour and their effect on the energy gains achieved using the methodology. The results demonstrate the effectiveness of the methodology reducing the memory energy consumption with between 35% and 55%. Since memory size requirements are still met in all situations, performance is not reduced. The memory-aware system scenario methodology is suited for applications that experience dynamic behaviour with respect to memory organisation utilization during their execution.

References

1. Benini L, Macii A, Poncino M (2000a) A recursive algorithm for low-power memory partitioning. In: Proceedings of the 2000 intr symp on Low Power Electronics and Design, pp 78 – 83
2. Benini L, et al (2000b) Increasing energy efficiency of embedded systems by application-specific memory hierarchy generation. Design Test of Computers, IEEE 17(2):74 –85, DOI 10.1109/54.844336
3. Binkert N, et al (2011) The gem5 simulator. SIGARCH Comput Archit News 39(2):1–7
4. Cheung E, et al (2009) Memory subsystem simulation in software tlm/t models. In: Proceedings of Asia and South Pacific Design Automation Conference, 2009., pp 811 –816, DOI 10.1109/ASPDAC.2009.4796580
5. Chung EY, et al (2002) Contents provider-assisted dynamic voltage scaling for low energy multimedia applications. In: Proceedings of the 2002 intr symp on Low Power Electronics and Design, ISLPED '02, pp 42–47
6. Filippopoulos I, et al (2012) Memory-aware system scenario approach energy impact. In: NORCHIP, 2012, pp 1 –6, DOI 10.1109/NORCHP.2012.6403111
7. Garcia P, et al (2006) An overview of reconfigurable hardware in embedded systems. EURASIP J Embedded Syst 2006(1):13–13
8. Gheorghita SV, et al (2009) System-scenario-based design of dynamic embedded systems. ACM Trans Des Autom Electron Syst 14(1):3:1–3:45
9. Gonzalez R, Horowitz M (1996) Energy dissipation in general purpose microprocessors. Solid-State Circuits, IEEE Journal of 31(9):1277 –1284, DOI 10.1109/4.535411
10. Guthaus M, Ringenberg J, Ernst D, Austin T, Mudge T, Brown R (2001) Mibench: A free, commercially representative embedded benchmark suite. In: Workload Characterization, 2001. WWC-4. 2001 IEEE Int. Workshop on, IEEE, pp 3–14
11. Hammari E, et al (2010) Application of medium-grain multiprocessor mapping methodology to epileptic seizure predictor. In: NORCHIP, 2010, pp 1 –6, DOI 10.1109/NORCHIP.2010.5669489
12. Hammari E, and Catthoor F, and Kjeldsberg PG, and Huisken J, and Tsakalis K, and Iasemidis L, (2012) The International Conference on Engineering of Reconfigurable Systems and Algorithms, 2012
13. Himpe S, et al (2002) MTG* and grey-box: modeling dynamic multimedia applications with concurrency and non-determinism. In: System Specification and Design Languages: Best of FDL02
14. Hulzink J, et al (2011) An ultra low energy biomedical signal processing system operating at near-threshold. IEEE Trans on Biomedical Circuits and Systems 5(6):546–554
15. Kritikakou A, et al (2013a) Near-optimal and scalable intra-signal in-place for non-overlapping and irregular access scheme. ACM Trans Design Automation of Electronic Systems (TODAES) conditionally accepted
16. Kritikakou A, et al (2013b) A scalable and near-optimal representation for storage size management. ACM Trans Architecture and Code Optimization accepted

17. Lee C, et al (1997) Mediabench: a tool for evaluating and synthesizing multimedia and communicatons systems. In: Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture, IEEE Computer Society, pp 330–335
18. Ma Z, et al (2007) Systematic Methodology for Real-Time Cost-Effective Mapping of Dynamic Concurrent Task-Based Systems on Heterogenous Platforms, 1st edn. Springer Publishing Company, Incorporated
19. Macii A, Benini L, Poncino M (2002) Memory Design Techniques for Low-Energy Embedded Systems. Kluwer Academic Publishers
20. Marchal P, et al (2003) SDRAM energy-aware memory allocation for dynamic multi-media applications on multi-processor platforms. In: Design, Automation and Test in Europe, pp 516–521
21. Meinerzhagen P, Roth C, Burg A (2010) Towards generic low-power area-efficient standard cell based memory architectures. In: Circuits and Systems (MWSCAS), 2010 53rd IEEE Int. Midwest Symposium on, IEEE, pp 129–132
22. Meinerzhagen P, et al (2011) Benchmarking of standard-cell based memories in the sub-vt domain in 65-nm cmos technology. IEEE Transactions on Emerging and Selected Topics in Circuits and Systems 1(2)
23. Miniskar NR (2012) System scenario based resource management of processing elements on mpsoc. PhD thesis, Katholieke Universiteit Leuven
24. Palkovic M, Catthoor F, Corporaal H (2006a) Dealing with variable trip count loops in system level exploration. In: Proc. of the 4th Workshop on Optimizations for DSP and Embedded Systems, IEEE and ACM SIGMICRO, pp 21–30
25. Palkovic M, Corporaal H, Catthoor F (2007) Heuristics for scenario creation to enable general loop transformations. In: System-on-Chip, 2007 Int. Symposium on, pp 1 –4, DOI 10.1109/ISSOC.2007.4427430
26. Palkovic M, et al (2006b) Systematic preprocessing of data dependent constructs for embedded systems. Journal of Low Power Electronics, Volume 2, Number 1
27. Panda PR, et al (2001) Data and memory optimization techniques for embedded systems. ACM Trans Des Autom Electron Syst 6(2):149–206
28. Pouchet L (2012) Polybench: The polyhedral benchmark suite
29. Simunic T, et al (1999) Cycle-accurate simulation of energy consumption in embedded systems. In: Proceedings of the 36th Design Automation Conference, 1999., pp 867 –872, DOI 10.1109/DAC.1999.782199
30. Abraham SG, Mahlke SA (1999) Automatic and efficient evaluation of memory hierarchies for embedded systems. In: Microarchitecture, 1999. MICRO-32. Proceedings. 32nd Annual International Symposium on, IEEE, pp 114–125
31. Chen F, Sha EHM (1999) Loop scheduling and partitions for hiding memory latencies. In: Proceedings of the 12th international symposium on System synthesis, IEEE Computer Society, p 64
32. Chen S, Postula A (2000) Synthesis of custom interleaved memory systems. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on 8(1):74–83
33. Grun P, Dutt N, Nicolau A (2000) Mist: An algorithm for memory miss traffic management. In: Proceedings of the 2000 IEEE/ACM international conference on Computer-aided design, IEEE Press, pp 431–438
34. Jacob BL, Chen PM, Silverman SR, Mudge TN (1996) An analytical model for designing memory hierarchies. Computers, IEEE Transactions on 45(10):1180–1194
35. Jantsch A, Ellervee P, Hemani A, Öberg J, Tenhunen H (1994) Hardware/software partitioning and minimizing memory interface traffic. In: Proceedings of the conference on European design automation, IEEE Computer Society Press, pp 226–231
36. Kandemir M, Sezer U, Delaluz V (2001) Improving memory energy using access pattern classification. In: Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design, IEEE Press, pp 201–206
37. Li Y, Wolf WH (1999) Hardware/software co-synthesis with memory hierarchies. Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on 18(10):1405–1417
38. Oshima Y, Sheu BJ, Jen SH (1997) High-speed memory architectures for multimedia applications. Circuits and Devices Magazine, IEEE 13(1):8–13
39. Passes N, Sha EM, Chao LF (1995) Multi-dimensional interleaving for time-and-memory design optimization. In: Computer Design: VLSI in Computers and Processors, 1995. ICCD'95. Proceedings., 1995 IEEE International Conference on, IEEE, pp 440–445

40. Schmit H, Thomas DE (1997) Synthesis of application-specific memory designs. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on 5(1):101–111
41. Heumann, Jim (2001) Generating test cases from use cases. The rational edge, 6(01)