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

Iason Filippopoulos
Ph.D. student
NTNU
Trondheim, Norway
iason.filippopoulos@iet.ntnu.no

Francky Catthoor
IMEC
KU Leuven
Leuven, Belgium
catthoor@imec.be

Per Gunnar Kjeldsberg
Professor
NTNU
Trondheim, Norway
pgk@iet.ntnu.no

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 which change dynamically at runtime. In addition to computational resources, the many possible memory platform configurations and data-to-memory assignments are important system scenario parameters. Here we present an extended memory model that includes existing state-of-the-art memories, available in the industry and academia, and show how it is 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. Reduction in energy consumption in the memory subsystem ranges from 35% to 55 % for the chosen set of benchmarks.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques—*design, methodologies*; B.7.1 [**Integrated Circuits**]: Types and Design Styles—*memory technologies, design*; B.8.2 [**Performance and Reliability**]: Performance Analysis and Design Aids; C.3 [**Special-Purpose and Application-Based Systems**]: Real-time and embedded systems

General Terms

Design, Algorithms, Measurement

Keywords

system scenarios, design space exploration, reconfigurable design, memory reconfiguration, dynamic multimedia applications

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. Consequentially, demanding applications, e.g., in the signal processing and multimedia domains, can be executed on these devices [21]. On the other hand, the desired performance has to be delivered with minimum power consumption due to the limited amount of power offered in mobile devices [16]. 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 [16]. In contrast to use case scenario approaches in which scenarios are generated based on a user's behaviour, system scenarios are based on the actual behaviour of the system.

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 [5]. 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 more detailed and accurate memory models used for the system design exploration, the extended number of benchmark applications on which the methodology is applied, and the categorisation of applications based on their dynamic characteristics. For the multimedia domain, the current work presents a comprehensive methodology for optimising energy consumption in the memory subsystem.

This paper is organized as follows. Section 2 motivates the study of optimizations of the memory organisation. Section 3 surveys related work on system level memory exploration and on system scenario methodologies and compares

[0]Version 11.04.2013: Final comment by PGK

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. MOTIVATION

As shown in [8] 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 [3] and the cycle-accurate simulator presented in [27] 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 [12] shows that the memory subsystem consumes more than 40% of the leakage power on the platform. According to [16], 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. Techniques have been developed in order to estimate the memory size requirements of applications in a systematic way [14]. 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 memory organisation in modern embedded devices.

To illustrate the aforementioned sub-optimal conventional allocation and assignment of data, the simple example in Fig. 1 is used. The kernel code of an image processing application continuously reads images, saves image on the memory and performs a *foo* function on each pixel of the image. Typically arrays are used for storing the intermediate calculations in image processing applications, *array* in the motivation example. The memory size used for storage of initial *image* and computed *array* is defined by the dimensions of the *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 *array*. However, only a part of the allocated space is accessed during processing of smaller images.

Algorithm 1 Motivation example of dynamic memory usage

```
1: while image ≠ EndOfDatabase do
2:    height ← length(image)
3:    width ← width(image)
4:    save(image[height][width])
5:    for i = 0 → height do
6:       for j = 0 → width do
7:          array[i][j] ← foo(image[i][j])
8:       end for
9:    end for
10:   image ← new.image
11: end while
```

The effect of sub-optimal allocation on the energy consumption can be illustrated using a simplified memory model and a few simple calculations. Assuming that an image *ImgA* is double the size of another image *ImgB*, then the memory space allocated for *image* and *array* is four times higher for *ImgA* compared to *ImgB*. Assuming also a memory subsystem with two independent scratch pad memories with sizes equal to *ImgA* and $4 \times$ *ImgA* and energy per access E and $1.5 \times E$, respectively. The number of accesses is N and $4 \times N$ for the processing of *ImgA* and *ImgB*. The conventional approach always allocates the whole memory to fit the processing needs of both cases and always assigns data to the most energy hungry memory, while the proposed methodology allocates according to the memory requirements of each image. The simplified calculations for energy consumption for the first approach is: $N \times 1.5E + 4 \times N \times 1.5E = 7.5 \times NE$, while the second approach results in: $N \times E + (N \times E + 3N \times 1.5E) = 6.5 \times NE$. In addition, a crude assumption on the leakage, as a percentage of the access energy for the whole processing time of the two images, can be made. Assuming a leakage energy of 30% on always active mode of the first approach the final energy consumption is $1.30 \times (7.5 \times NE) = 9.75 \times NE$. In the proposed approach each of the scratch pad memories is set into sleep mode for half of the time and assuming no leakage during that time, the final energy consumption will be $1.15 \times (6.5 \times NE) = 7.475 \times NE$, which is around 25% lower. The proposed methodology optimises the memory usage for dynamic cases similar to the simplified motivation example.

3. RELATED WORK

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 [2] authors present a methodology for automatic memory hierarchy generation that exploits memory access locality, while in [1] they propose an algorithm for the automatic partitioning of on-chip SRAM in multiple banks that can be independently accessed. Several design techniques for designing energy efficient memory architectures for embedded systems are presented in [17]. In [25] data and memory optimisation techniques, that could be dependent or independent of a target platform, are discussed.

Energy-aware assignment of data to memory banks for several task-sets based on the MediaBench suit of benchmarks is presented in [18]. Low energy multimedia applications are

discussed also in [4] with focus on processing rather than the memory platform. Furthermore, both [18] and [4] 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.

An overview of work on system scenario methodologies and their application are presented in [7]. In [5] 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 [10] is required by many dynamic systems including the majority of multimedia applications. The wide range of possible values for data variables is higher and makes full exploration impossible. Most of the dynamic variables in the current work can be classified as data variables due to their significant variation under different execution situations.

Authors in [22] 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 [24]. A heuristic to perform efficient grouping of execution paths for scenario creation is analysed in [23]. 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 [6]. 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 [16]. 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 consequentially 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 the 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. In contrast to use case scenario approaches in which scenarios are generated based on a user's behaviour, the system scenario methodology focuses on behaviour of the system to generate scenarios. 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 [10]. 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 [7], which 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 [16]. In more detail, 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 [11].

4.1 Design-time Profiling Based on Data Variables

Application profiling is performed at design-time and consists of an analysis of the target application during its lifetime and for a wide range of inputs. The analysis focuses on the allocated memory size during execution and the variation in access pattern of the application. Techniques described in [13] are used among others, in order to extract the access scheme. Those memory size estimation techniques analyse the iteration space of array elements, identify any holes in it and provide the exact number of memory accesses. Several applications dynamically allocate and deallocate memory space during their execution, based both on the source code and the given input.

The profiling stage is depicted in Fig. 1 and consists of run-

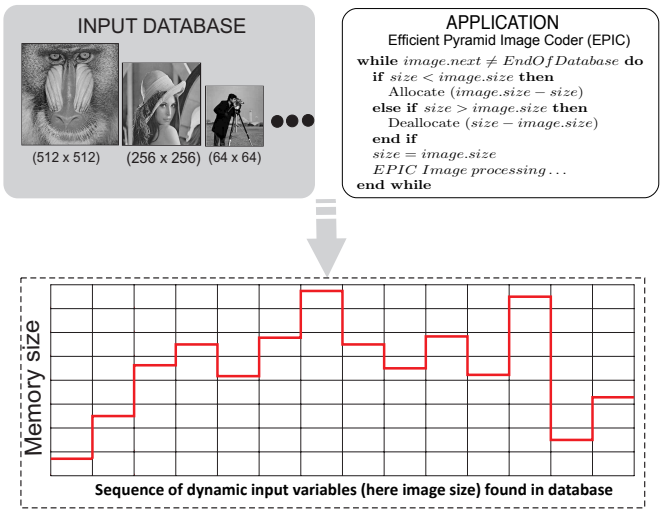

Figure 1: Profiling results based on application code and input data.

ning the application code with suitable input data often found in a database, in order to produce profiling results. This 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. Choosing an extensive and accurate database is vital and will heavily influence and steer the designer's decisions in later steps.

Given code and database as inputs, profiling will show memory usage during execution time by running the application using the whole database as an input. 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 to 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 an data variable due to the wide range of its possible values.

4.2 Design-time System Scenario Identification and Prediction Based on Data Variables

The next step is the clustering of the profiled memory sizes into groups with similar characteristics, which is referred 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 amount of memories needed. Clustering neighbouring RTSs is a rational choice, because two instances with similar memory needs have similar energy consumption.

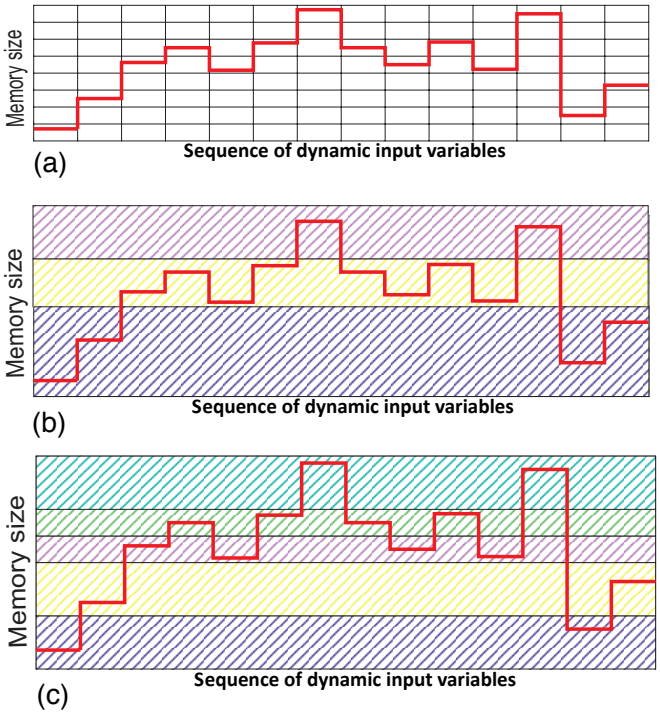

Figure 2: Clustering of profiling (a) results into three (b) or four (c) system scenarios

In Fig. 2 the clustering of the previously profiled information is presented. The clustering of RTSs is based both on their distance in 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 coloured hashed areas depicted in Fig. 2(b). Due to the higher frequency of RTSs in the yellow hashed area that system scenario is less wide compared to its neighbouring scenarios. That clustering is better compared to an even splitting of the area to three, 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 principal applies also when the number of system scenarios is increased to five, as depicted in Fig. 2(c). 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 facts. 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

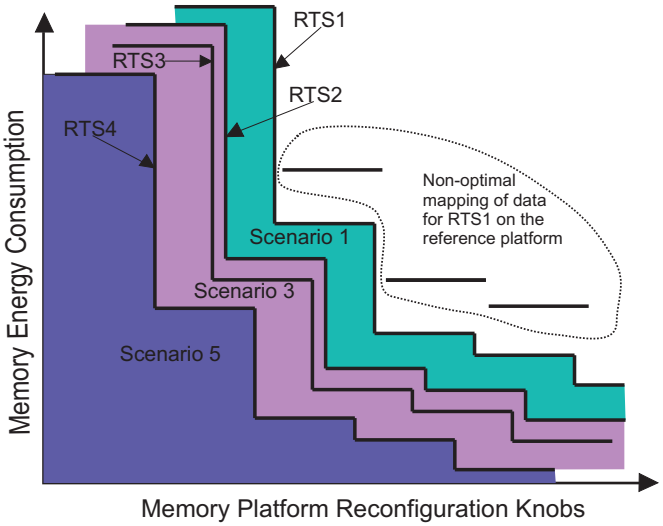

Figure 3: Clustering of Pareto curves

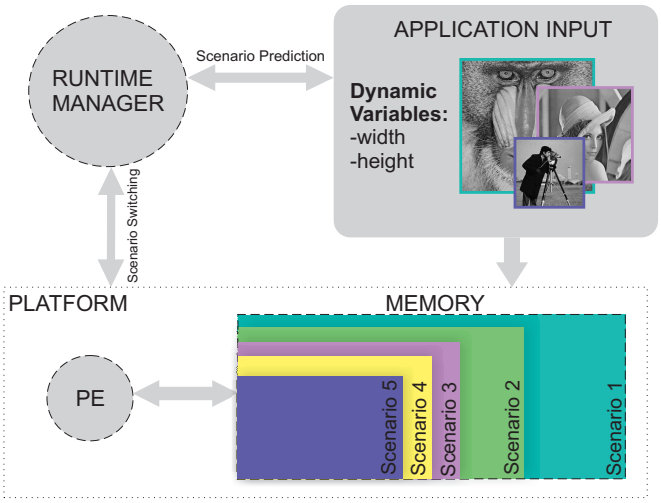

Figure 4: Runtime system scenario prediction and switching based on the current input

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 five 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.

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 for each platform configuration. 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 data-to-memory assignment decisions. Pareto curves are clustered into three different system scenarios based again both on their distance 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 into exploration.

The design-time system scenario prediction phase consists of determination of the data variables that define the active system scenario. This can be achieved by careful study of the application code, combined with the application's data input. 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 prediction data variable is the input image height and width 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 [16].

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. The switching phase consists of all platform configuration decisions that can be made at run-time, e.g., frequency/voltage scaling, turning on/off a memory unit, 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 [16]. 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. 1 an example of the run-time phase of the methodology is depicted. The runtime 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.

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.

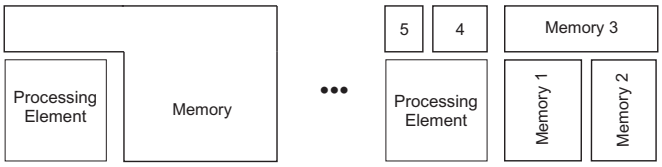

Figure 5: Exploration of memory clustering into varying number of banks

5.1 Architecture

In this work, a clustered memory organisations 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.

An example of the exploration of the memory platform is shown in Fig. 5. Point-to-point connections between elements are assumed and the interconnection cost is kept negligible 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). In addition, experimental standard cell-based memories (SCMEM) [20] are considered for smaller memories due to their energy and area efficiency for reasonably small storage capacities, as argued in [19]. 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 in the maximum supported speed. The supply voltage is 1.1V and it is expected that the dynamic and leakage power are higher compared to the other modes.
- Light sleep mode: The supply voltage in this mode is lower than active with values in the area of 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 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, approximately in the range of 20 to 50 clock cycles depending on the clock speed. Consequentially, 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 to substantial energy savings.

The necessary energy/power information is available to the system designer and relative values for some of the used sizes in the current work are presented in Tab. 1. It is clearly shown that the choice of the memory units has an important impact on the energy consumption. Moreover, different decisions have to be made based on the dominance of the dynamic or the 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 Energy consumption calculation

The overall energy consumption for each configuration is calculated using a detailed formula, as can be seen in (1).

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

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

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

Type	Lines x wordlength	Dynamic Energy		Static Leakage Power per Mode			
		Read	Write	Active	Light-sleep	Deep-sleep	Shut-down
MM	32 x 8	4.18×10^{-8}	3.24×10^{-8}	0.132	0.125	0.063	0.0016
MM	32 x 16	6.79×10^{-8}	5.89×10^{-8}	0.134	0.127	0.064	0.0022
MM	32 x 128	4.33×10^{-7}	4.31×10^{-7}	0.171	0.160	0.083	0.0112
MM	256 x 128	4.48×10^{-7}	4.60×10^{-7}	0.207	0.184	0.104	0.0293
MM	1024 x 128	5.11×10^{-7}	5.75×10^{-7}	0.349	0.283	0.189	0.102
MM	4096 x 128	9.60×10^{-7}	4.57×10^{-7}	0.95	0.708	0.544	0.396
SCMEM	128 x 128	2.5×10^{-7}	0.8×10^{-8}	0.083	0.057	0.027	0.0022
SCMEM	1024 x 8	1.7×10^{-8}	0.6×10^{-8}	0.042	0.028	0.014	0.0011

- 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 on the 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 leak time. It can be found by executing the application on a reference embedded processor.

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 during their execution. Multimedia applications often exhibit such a dynamic variation on memory requirements during their lifetime and consequentially 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 ([26]), Mibench ([9]) and Mediabench([15]) benchmark suites.

6.1 Presentation of Multimedia Benchmark Applications

An overview of the benchmark applications that were tested is presented in Tab. 2. Two key parameters under consideration are the dynamic data variable on each application and the variation in the memory requirements. 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 a loop bound with a non-fixed value. The memory size limits are defined as the minimum and maximum storage requirement occurred during testing of 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 for determining the motion vectors and different images are tested.

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. *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 *PGP encryption* algorithm is also included in the employed benchmark suites and its encryption length parameter has an important impact on memory size, which can be exploited using system scenarios. The effect of the SNR level on the channel on the constraint length value on the *Viterbi encoder* algorithm is discussed on [5]. The increment of noise in the channel demands a more complex encoding in order to maintain a constant bit error rate (BER), which consequentially 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 Basic 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 basic characteristics is outlined that can assist the system designer in the employment of the methodology and

Table 2: Benchmark applications overview

Application	Data variable	Memory variation(B)	Source	Characteristics
Epic image compression	Image size	4257 - 34609	MediaBench	Average dynamism, good distribution
Motion Estimation	Image size	4800 - 52800	MediaBench	High dynamism, average distribution
Blowfish decoder	Input file size	256 - 5120	MiBench	Low dynamism, poor distribution
Jacobi 1D Decomposition	Number of steps	502 - 32002	Polybench	Low dynamism, good distribution
Mesa 3D	Loop bound	5 - 50000	MediaBench	High dynamism, average distribution
JPEG DCT	Block size	10239 - 61439	MediaBench	High dynamism, average distribution
PGP encryption	Encryption length	3073 - 49153	MediaBench	High dynamism, average distribution
Viterbi encoder	Constraint length	5121 - 14337	Open	Low dynamism, good distribution

reveal the expected behaviour prior to experimentation with an application. The basic characteristics that are used to categorize the applications are the dynamism in the memory size bounds and the variance of cases within memory size limits.

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 a group of 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. Two good examples are the blowfish and viterbi algorithms.

The second metric used for identification of different kinds of dynamism is the memory requirement variation. 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. 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. much smaller energy gains after a number of platform supported system scenarios have reached. 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.

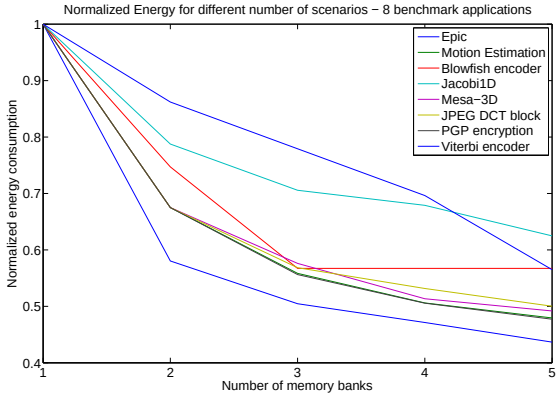

Figure 6: Energy consumption per number of memory banks - Energy is normalized per application

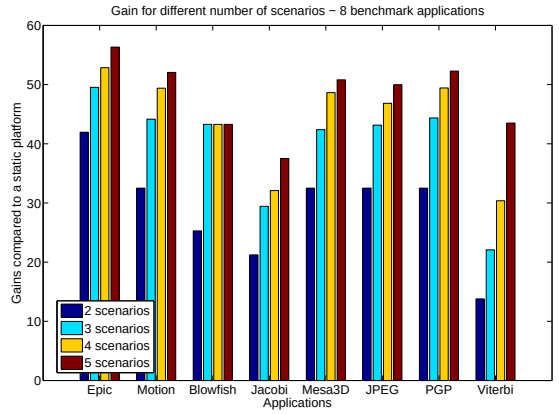

Figure 7: Energy gain for increasing number of system scenarios - Static platform corresponds to 0%

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. 2 and is followed by the clustering phase. The clustering is performed with one to five system scenarios. All potentially energy efficient configurations are tested for a given number of scenarios. The exploration includes memories of different sizes, technologies and varying word lengths.

The normalized energy consumption in the memory subsystem is shown in Fig. 6 while the energy gain percentages are presented in Fig. 7. Energy gains are compared to a static platform configuration, i.e., a platform with only 1 scenario, which corresponds to zero percentage gain in Fig. 7. Only the optimal configuration is presented for each number of system scenarios. The energy is normalized for each application separately. Gains are reported compared to the case of the fixed non-re-configurable platform.

Table 3: Range of energy gains on the memory subsystem

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

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 on 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. In Tab. 2 the minimum and maximum energy gains for each benchmark application are shown.

Comparative results using 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 crude 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 in for two benchmarks to a platform with only two scenarios.

The reported energy gains are for the memory subsystem. As reported 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.

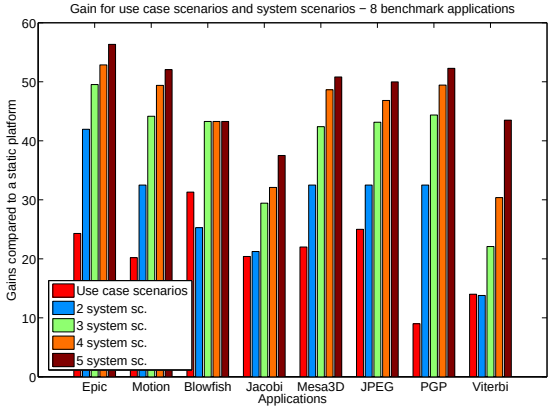

Figure 8: Energy gain for use case scenarios and system scenarios

8. CONCLUSION

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 of the methodology. The achieved 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.

9. REFERENCES

[1] L. Benini et al. Increasing energy efficiency of embedded systems by application-specific memory hierarchy generation. *Design Test of Computers, IEEE*, 17(2):74 –85, apr-jun 2000.

[2] L. Benini, A. Macii, and M. Poncino. A recursive algorithm for low-power memory partitioning. In *Low Power Electronics and Design, 2000. ISLPED '00. Proceedings of the 2000 International Symposium on*, pages 78 – 83, 2000.

[3] E. Cheung, H. Hsieh, and F. Balarin. Memory subsystem simulation in software tlm/t models. In *Design Automation Conference, 2009. ASP-DAC*

2009. Asia and South Pacific, pages 811 –816, jan. 2009.

[4] E.-Y. Chung, G. De Micheli, and L. Benini. Contents provider-assisted dynamic voltage scaling for low energy multimedia applications. In *Proceedings of the 2002 international symposium on Low power electronics and design*, ISLPED '02, pages 42–47, 2002.

[5] I. Filippopoulos, F. Catthoor, P. G. Kjeldsberg, E. Hammari, and J. Huisken. Memory-aware system scenario approach energy impact. In *NORCHIP, 2012*, pages 1 –6, nov. 2012.

[6] P. Garcia, K. Compton, M. Schulte, E. Blem, and W. Fu. An overview of reconfigurable hardware in embedded systems. *EURASIP J. Embedded Syst.*, 2006(1):13–13, Jan. 2006.

[7] S. V. Gheorghita, , et al. System-scenario-based design of dynamic embedded systems. *ACM Trans. Des. Autom. Electron. Syst.*, 14(1):3:1–3:45, Jan. 2009.

[8] R. Gonzalez and M. Horowitz. Energy dissipation in general purpose microprocessors. *Solid-State Circuits, IEEE Journal of*, 31(9):1277 –1284, sep 1996.

[9] M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R. Brown. Mibench: A free, commercially representative embedded benchmark suite. In *Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop on*, pages 3–14. IEEE, 2001.

[10] E. Hammari, F. Catthoor, J. Huisken, and P. G. Kjeldsberg. Application of medium-grain multiprocessor mapping methodology to epileptic seizure predictor. In *NORCHIP, 2010*, pages 1 –6, nov. 2010.

[11] S. Himpe, G. Deconinck, F. Catthoor, and J. van Meerbergen. Mtg* and grey-box: modelling dynamic multimedia applications with concurrency and non-determinism. In *System Specification and Design Languages: Best of FDLâĂŹ02*, 2002.

[12] J. Hulzink, M. Konijnenburg, M. Ashouei, A. Breeschoten, T. Berset, J. Huisken, J. Stuyt, H. de Groot, F. Barat, J. David, et al. An ultra low energy biomedical signal processing system operating at near-threshold. *Biomedical Circuits and Systems, IEEE Transactions on*, 5(6):546–554, 2011.

[13] A. Kritikakou, F. Catthoor, V. Kelefouras, and C. Goutis. 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, 2013.

[14] A. Kritikakou, F. Catthoor, V. Kelefouras, and C. Goutis. A scalable and near-optimal representation for storage size management. *ACM Trans. Architecture and Code Optimization*, conditionally accepted, 2013.

[15] C. Lee, M. Potkonjak, and W. Mangione-Smith. Mediabench: a tool for evaluating and synthesizing multimedia and communicatons systems. In *Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture*, pages 330–335. IEEE Computer Society, 1997.

[16] Z. Ma et al. *Systematic Methodology for Real-Time Cost-Effective Mapping of Dynamic Concurrent Task-Based Systems on Heterogenous Platforms.* Springer Publishing Company, Incorporated, 1st edition, 2007.

[17] A. Macii, L. Benini, and M. Poncino. *Memory Design Techniques for Low-Energy Embedded Systems.* Kluwer Academic Publishers, 2002.

[18] P. Marchal, D. Bruni, J. Gomez, L. Benini, L. Pinuel, F. Catthoor, and H. Corporaal. Sdram-energy-aware memory allocation for dynamic multi-media applications on multi-processor platforms. In *Design, Automation and Test in Europe Conference and Exhibition, 2003*, pages 516–521, 2003.

[19] P. Meinerzhagen, C. Roth, and A. Burg. Towards generic low-power area-efficient standard cell based memory architectures. In *Circuits and Systems (MWSCAS), 2010 53rd IEEE International Midwest Symposium on*, pages 129–132. IEEE, 2010.

[20] P. Meinerzhagen, S. Y. Sherazi, A. Burg, and J. N. Rodrigues. 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), 2011.

[21] N. R. Miniskar. *System Scenario Based Resource Management of Processing Elements on MPSoC.* PhD thesis, Katholieke Universiteit Leuven, 2012.

[22] M. Palkovic, F. Catthoor, and H. Corporaal. Dealing with variable trip count loops in system level exploration. In *ODES: 4th Workshop on Optimizations for DSP and Embedded Systems*, 2006.

[23] M. Palkovic, H. Corporaal, and F. Catthoor. Heuristics for scenario creation to enable general loop transformations. In *System-on-Chip, 2007 International Symposium on*, pages 1 –4, nov. 2007.

[24] M. Palkovic et al. Systematic preprocessing of data dependent constructs for embedded systems. *Journal of Low Power Electronics, Volume 2, Number 1*, 2006.

[25] P. R. Panda et al. Data and memory optimization techniques for embedded systems. *ACM Trans. Des. Autom. Electron. Syst.*, 6(2):149–206, Apr. 2001.

[26] L. Pouchet. Polybench: The polyhedral benchmark suite.

[27] T. Simunic, L. Benini, and G. De Micheli. Cycle-accurate simulation of energy consumption in embedded systems. In *Design Automation Conference, 1999. Proceedings. 36th*, pages 867 –872, 1999.