Scratchpad Sharing Strategies for Multiprocess Embedded Systems: A First Approach

Manish Verma, Klaus Petzold, Lars Wehmeyer, Heiko Falk, Peter Marwedel
Department of Computer Science XII
University of Dortmund, 44225 Dortmund, Germany
{Manish.Verma,Klaus.Petzold,Lars.Wehmeyer,Heiko.Falk,Peter.Marwedel}@udo.edu

Abstract

Portable embedded systems require diligence in managing their energy consumption. Thus, power efficient processors coupled with onchip memories (e.g. caches, scratchpads) are the base of today's portable devices. Scratchpads are more energy efficient than caches but require software support for their utilization. Portable devices' applications consist of multiple processes for different tasks. However, all the previous scratchpad allocation approaches only consider single process applications. In this paper, we propose a set of optimal strategies to reduce the energy consumption of applications by sharing the scratchpad among multiple processes. The strategies assign both code and data elements to the scratchpad and result in average total energy reductions of 9%-20% against a published single process approach. Furthermore, the strategies generate Pareto-optimal curves for the applications allowing design time exploration of energy/scratchpad size tradeoffs.

1. Introduction

Today's portable devices undergo an increasing convergence and feature enhancement, e.g. a mobile phone featuring Bluetooth and a color display. It can take digital pictures and videos, play MP3 files and act as a PDA. Besides faster processors and larger memories, energy optimization plays a major role in making these devices possible, since beside the aforementioned features, consumers expect their mobile phones to have an extended battery life. A lot of research effort is thus being directed towards energy optimizations.

The memory subsystem has been identified as an energy bottleneck of entire systems [5, 6]. Memory hierarchies are used to reduce the memory subsystem's energy dissipation. Caches and scratchpads represent two contrasting memory architectures. Caches improve performance by exploiting the temporal and spatial locality in a program. However, for embedded systems, the overheads associated with caches often negate their benefits. Moreover, caches are notorious for their unpredictable behavior.

On the other hand, a scratchpad memory (SPM) consists of just a memory array and address decoders. Therefore, scratchpads are both area and power efficient than caches [3]. However, they require complex program analysis and explicit support from the compiler for managing their contents. This in-turn makes scratchpads predictable, as their contents are 100% known during the execution time.

The convergence phenomenon for portable devices has resulted in complex multiprocess applications with each process being responsible for a particular task. Nevertheless, all the current scratchpad assignment techniques focus on single process applications. Applying the current techniques to a multiprocess application will result in non-optimal energy savings. This paper presents our first attempt at minimizing the energy consumption of multiprocess applications by utilizing a set of compiler based optimal scratchpad sharing techniques which are predictable and analyzable at design time. In that respect, this is the first work on fully-analyzable scratchpad sharing techniques for multiprocess systems.

The rest of the paper is structured as follows: After the presentation of related work, the multiprocess scratchpad allocation techniques are presented. This is followed by Section 4 presents the experimental workflow and Section 5 discussing the experimental results. The paper ends with a conclusion and future work.

2. Related Work

The research on scratchpad utilization for single process applications can be classified into two broad categories *viz.* non-overlay and overlay based allocation techniques. In the former, the scratchpad is loaded once at the start and its contents remain invariant during the entire execution of the application. In contrast, overlay based allocation techniques partition the application into overlays which are copied on and off the scratchpad during the execution to reflect the dynamic behavior of the application.

Non-overlay based allocation techniques [6, 8, 9] can be classified into techniques which allocate only data or only instructions or both instructions and data. The allocation approach proposed in [6] allocates only data, while [9] allocates only instructions onto the scratchpad. Authors in [8] assigned both aggregate variables and instruction segments onto the scratchpad.

Similarly, overlay based allocation techniques [1, 5, 10] can be classified into techniques which cover only data elements or only instructions or a combination of both. Authors in [5] proposed techniques to dynamically copy overlays of data elements onto the scratchpad. Authors in [1]

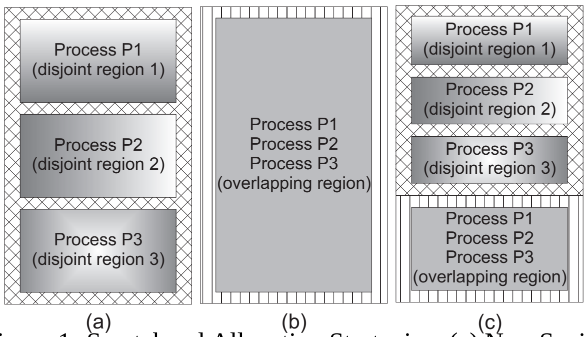


Figure 1: Scratchpad Allocation Strategies: (a) Non-Saving (b) Saving and (c) Hybrid

proposed an approach to assign instruction overlays onto the scratchpad. A recent approach [10] considers both instructions and data.

Scratchpad allocation for multiprocess systems is still in an early research stage. Only one approach [4] to share the scratchpad among processes is known. However, it is not automated and the programmer needs to insert API calls at appropriate locations to utilize the scratchpad. Our approaches are complementary to [4] as they automate the sharing of the scratchpad among the processes.

3. Scratchpad Sharing Strategies

The multiprocess scratchpad sharing approaches proposed in this paper enable sharing of the scratchpad memory by the processes of an application, under the objective to minimize the energy consumption of the entire application. We propose 3 scratchpad sharing approaches *viz.* Scratchpad Non-Saving/Restoring Context Switch (Non-Saving), Scratchpad Saving/Restoring Context Switch (Saving) and Hybrid Scratchpad Saving/Restoring Context Switch (Hybrid). The proposed approaches use a non-overlayed scratchpad allocation strategy [8] to select and map the energy optimal set of memory objects (instruction segments and variables) for each process onto the scratchpad. In addition, the Saving and the Hybrid approaches require runtime support from the dispatcher for saving and restoring the scratchpad contents. The Non-Saving approach partitions the scratchpad into disjoint regions and each process is exclusively allocated at most one region. As a consequence, the contents of the disjoint regions never need to be updated at context switch time. Figure 1(a) shows the distribution of scratchpad memory into 3 disjoint regions. This approach is beneficial for large scratchpads, as they can be adequately distributed among the processes.

In contrast to the previous approach, the Saving approach shares the scratchpad (cf. Figure 1(b)) as a common overlapping region among all the processes. Memory objects belonging to a process are copied to the scratchpad when the process is scheduled to execute on the processor. They are copied back to the main memory when the process is scheduled off the processor. This leads to reduced energy consumption when the copy overhead is less than the energy reduction achieved by the increased scratchpad utilization. For small scratchpads, the Saving approach is better

Variables	Definition
n	Number of processes.
M	Maximum scratchpad size.
$P_1 \ldots P_n$	Set of processes.
s_k	Schedule count for process P_k.
$f_k^N(x)$	Energy function for process P_k with x bytes disjoint SPM region $f_k^N\colon[0,M] \to \mathbb{R}$.
$f_k^S(x)$	Energy function for process P_k with x bytes overlapping SPM region $f_k^S\colon[0,M] \to \mathbb{R}$.
$f_k^H(x,y)$	Energy function for process P_k with x and y bytes disjoint and overlapping SPM regions $f_k^S\colon[0,M] \times [0,M] \to \mathbb{R}$.
$CE_{SP}(x)$	Copy Energy function, copying x bytes from main memory to SPM $CE_{SP}\colon[0,M] \to \mathbb{R}$.
$CE_{MM}(x)$	Copy Energy function, copying x bytes from SPM to main memory $CE_{MM}\colon[0,M] \to \mathbb{R}$.

Table 1: Definition of variables

than the Non-Saving approach while the opposite is true for large scratchpads. Consequently, the Hybrid approach is proposed as a combination of the two approaches. As shown in Figure 1(c), a portion of the scratchpad is distributed as disjoint regions while the remaining portion is commonly utilized by the processes. Only this common region needs to be updated at every context switch. The hybrid approach minimizes the energy consumption of the application for all sizes, as it can partition the scratchpad into both disjoint and overlapping regions.

For this work, we assumed a statically scheduled system with n processes such that the execution profile (*i.e.* execution time and energy consumption) of each process is known or can be estimated a priori. Let us define a few variables (cf. Table 1) prior to presenting the allocation algorithms. Assume that the multiprocess application consists of n equal priority processes $P_1 \ldots P_n$ running on a single processor system with an M byte scratchpad memory and that the non-saving $f_k^N(x)$, the saving $f_k^S(x)$ and the hybrid $f_k^H(x,y)$ energy functions are known for each process P_k. The non-saving (saving) energy function $f_k^N(x)$ ($f_k^S(x)$) computes the energy consumed by process P_k when it is assigned a disjoint (overlapping) scratchpad region of x bytes. Similarly, $f_k^H(x,y)$ is the hybrid energy function of a process P_k utilizing both disjoint and overlapping scratchpad regions of x and y bytes, respectively. Furthermore, schedule count s_k is the number of times a process P_k is scheduled to execute on the processor.

We now introduce a motivating example of a mobile application consisting of three simultaneously running processes (*viz. mpegdec, receive, ui*). The process *receive* receives packets of MPEG video over the network. They are decoded by the process *mpegdec* to produce video frames which are then displayed on the screen by process *ui*. Table 2 presents the non-saving energy functions of the processes. All energy values related to the example application are in abstract units. Energy function $f_{mpegdec}^N$ e.g. denotes that process *mpegdec* consumes 84 units of energy when it utilizes a 1kB disjoint scratchpad region. Assume that a 4kB scratchpad is present in the system that can be assigned to a

Process	Energy fn.	0kB	1kB	2kB	3kB	4kB
mpegdec	$f^N_{mpegdec}$	100	84	60	60	57
receive	$f^N_{recieve}$	40	25	20	18	18
ui	f^N_{ui}	20	9	8	8	8

Table 2: Non-Saving Energy functions (abstract units) for Mobile application

Algorithm 1 BinMin

```
Require: Energy functions $f(x)$ and $g(x)$ st. $f, g : [0,M] \to \mathbb{R}$
Ensure: $h : [0,M] \to \mathbb{R}$, where $h(x_i) = min\{f(l_j) + g(m_k) | l_j + m_k \le x_i\}$
1: $min = \infty$
2: for $x_i = 0$ to $M$ do
3:     for $tmp = 0$ to $x_i$ do
4:         $l_j = tmp$
5:         $m_k = x_i - tmp$
6:         if $f(l_j) + g(m_k) < min$ then
7:             $min = f(l_j) + g(m_k)$
8:     $h(x_i) = min$
9: return $h(x)$
```

single process. Under these assumptions, process *mpegdec* is assigned the entire 4kB scratchpad, as it results in the maximum energy reduction of the application: It is computed to be $57+40+20 = 117$ units when *mpegdec* is assigned a 4kB scratchpad region. The approaches presented in this paper will reduce the energy consumption further by sharing the scratchpad.

3.1. Notation

Several notations to represent functions are commonly used. The rigorous notation [11] $f\colon x \to f(x)$ specifies that f is a function acting upon a single number x and returns a value $f(x)$. In addition, the notation $f(x)$ is used to refer to the function f. In this text, unless indicated otherwise, the notation $f(x)$ refers to the rigorous notation $f\colon x \to f(x)$ and the notation $f(x_i)$ with a subscripted argument x_i refers to the value of the function f at number x_i.

3.2. Non-Saving Approach

The Non-Saving approach partitions the scratchpad into disjoint regions such that each process is allocated a maximum of one scratchpad region, the objective being to minimize the energy consumption of the application by assigning appropriate scratchpad regions to each of the processes. Given the non-saving energy function $f^N_1(x) \dots f^N_n(x)$ of each process, the non-saving energy function $h^N_n(x)$ of the application is computed as the following:

$$h^N_n(x) = min\{f^N_1(x_1) + \ldots + f^N_n(x_n) | x_1 + \ldots + x_n \le x\} \quad (1)$$

The value of the function $h^N_n(x)$ at $x = M$ denotes the energy consumption of the application utilizing an M bytes scratchpad. Computing the energy function $h^N_n(x)$ using the above equation requires $O((x+1)^n)$ summation operations. Thus, a full-exhaustive search algorithm would require an exponential $O(M^n)$ running time. An efficient algorithm can utilize the following distributive property of the *min* operator:

$$\begin{aligned} h^N_3(x) &= min\{f^N_1(x_1) + f^N_2(x_2) + f^N_3(x_3) | x_1 + x_2 + x_3 \le x\} \\ &= binmin(\ binmin(f^N_1, f^N_2),\ f^N_3\ ) \end{aligned} \quad (2)$$

Algorithm 2 NonSaving

```
Require: Process Energy functions $f^N_1(x), \ldots, f^N_n(x)$
Ensure: $h^N_n : [0,M] \to \mathbb{R}$, where
    $h^N_n(x) = min\{f^N_1(x_1) + \ldots + f^N_n(x_n) \mid x_1 + \ldots + x_n \le x\}$
1: if $n > 1$ then
2:     $h^N_{n-1}(x) = NonSaving(f^N_1(x), \ldots, f^N_{n-1}(x))$
3:     $h^N_n(x) = BinMin(h^N_{n-1}(x), f^N_n(x))$
4: else
5:     $h^N_n(x) = BinMin(f^N_1(x), Z(x))$ \* $Z(x) = 0\ \forall x$ *\
6: return $h^N_n(x)$
```

Energy fn. (Soln. Set)	0kB	1kB	2kB	3kB	4kB
$h^N_2(x) =$ $binmin(f^N_{mpegdec}, f^N_{receive})$	140 (0,0)	124 (1,0)	100 (2,0)	85 (2,1)	80 (2,2)
$h^N_3(x) =$ $binmin(h^N_2, f^N_{ui})$	160 (0,0)	144 (1,0)	120 (2,0)	105 (3,0)	94 (3,1)

Table 3: Non-Saving Approach for Mobile Application

The binary minimum function *binmin* is defined below:

$$\begin{aligned} h(x) &= binmin(f, g) \\ &= min\{f(l_j) + g(m_k) | l_j + m_k \le x\} \end{aligned} \quad (3)$$

The binary minimum function takes two functions as input and returns another function. The returned function $h(x)$ represents the minimum sum of the input functions f and g such that the sum of the arguments l_j and m_k is less than x. Equations 2 are 3 used to convert the energy function (cf. equation 1) into the following recurrence equation.

$$\begin{aligned} h^N_1(x) &= binmin(f^N_1(x), Z(x)) \\ h^N_n(x) &= binmin(h^N_{n-1}(x), f^N_n(x)) \end{aligned} \quad (4)$$

where $Z(x) = 0$ is a zero function which returns zero for all values of x. The correctness proof of the recurrence equation is presented in [2]. Algorithm 1 presents the pseudo-code for computing the binary minimum function *binmin* for two energy functions, requiring $O(M^2)$ runtime. The recurrence equation (cf. equation 4) is implemented using algorithm 2. The recursive algorithm takes energy functions $f^N_1(x), \ldots, f^N_n(x)$ each corresponding to one of the n processes and returns the energy function $h^N_n(x)$ of the application which describes the optimal energy values for the Non-Saving approach. The application of the algorithm to the energy functions (cf. Table 2) of the mobile application is shown in Table 3. The table also presents the scratchpad regions assigned to each process and the saving energy function $h^N_3(x)$ of the application. The overall runtime of algorithm 2 is $O(nM^2)$. The solution set S contains the size of the scratchpad region allocated to each process. Only minimal extensions are required in algorithms 1 and 2 to also determine the solution set.

3.3. Saving Approach

The Saving approach assumes that the entire scratchpad is a single overlapping region shared by all processes constituting the application. The dispatcher copies a process' memory objects from the main memory to the scratchpad every time it is scheduled to run on the processor and back to the main memory when the process is taken off the processor. The saving energy function $f^S_k(x)$ of process P_k uti-

Process	Energy fn.	0kB	1kB	2kB	3kB	4kB
mpegdec	$f^S_{mpegdec}$	100	89	70	75	77
receive	$f^S_{receive}$	40	30	30	33	38
ui	f^S_{ui}	20	14	18	23	28
Mobile Application	$h^S_3(x) = Saving(f^S_{ui}, f^S_{mpegdec}, f^S_{receive})$	160	133	114	114	114

Table 4: Saving Energy functions (abstract units) for Mobile Application

Algorithm 3 Saving

```
Require: Process Energy functions $f^N_1(x), \dots, f^N_n(x)$
Require: Copy Energy functions $CE_{SP}(x)$ and $CE_{MM}(x)$
Require: Schedule count $s_1, \dots, s_n$
Ensure: $h^S_n : [0,M] \to \mathbb{R}$, where $h^S_n(x_i) = \sum_{k=1}^n \{min[f^S_k(l_j)|l_j \le x_i]\}$
 1: for $k = 1$ to $n$ do
 2:     $prev\_min[k] = \infty$
 3: for $x_i = 0$ to $M$ do
 4:     for $k = 1$ to $n$ do
 5:         $l_j = x_i$
 6:         $f^S_k(l_j) = f^N_k(l_j) + s_k * (CE_{SP}(l_j) + CE_{MM}(l_j))$
 7:         if $f^S_k(l_j) < prev\_min[k]$ then
 8:             $min[k] = f^S_k(l_j)$
 9:         else
10:             $min[k] = prev\_min[k]$
11:     $E_{min} = 0$
12:     for $k = 1$ to $n$ do
13:         $E_{min} = E_{min} + min[k]$, $prev\_min[k] = min[k]$
14:     $h^S_n(x_i) = E_{min}$
15: return $h^S_n(x)$
```

lizing x bytes overlapping scratchpad region is derived as:

$$f^S_k(x) = f^N_k(x) + s_k * [CE_{SP}(x) + CE_{MM}(x)] \quad (5)$$

where $f^N_k(x)$ and s_k represent the non-saving energy function and the schedule count of process P_k, respectively. The copy energy functions $CE_{SP}(x)$ and $CE_{MM}(x)$ return the energy consumed in copying x bytes to/from the scratchpad from/to the main memory, respectively. Table 4 presents the saving energy functions for the processes of the mobile application computed using equation 5 as well as the saving energy function of the entire application. The non-saving energy function values were taken from Table 2, while the aggregate copy energy overhead was assumed to contribute 5 abstract units of energy for every 1kB of scratchpad in this simplified example.

The minimum energy consumption $h^S_n(x)$ of the application for x bytes of scratchpad is the sum of the minimum value of the energy function $f^S_k(x)$ over the range of $[0,x]$ bytes for each process P_k. The saving energy function of the application allocated using the Saving approach is written as:

$$h^S_n(x) = \sum_{k=1}^n min[f^S_k(l_j)|l_j \le x] \quad (6)$$

Algorithm 3 presents the algorithm for the computation of $h^S_n(x)$ for the application. The algorithm computes the saving energy function $f^S_k(x)$ for each process P_k using equation 5, requiring $O(nM)$ time to compute the optimal energy function $h^S_n(x)$ for the multiprocess application. The energy function $h^S_3(x)$ for the mobile application is presented in the last row of Table 4. The following subsection proposes a hybrid approach to share the scratchpad.

Algorithm 4 HybBinMin

```
Require: Energy functions $f$ and $g$ st. $f,g : [0,M] \times [0,M] \to \mathbb{R}$
Ensure: $h : [0,M] \times [0,M] \to \mathbb{R}$, where $h(x_i,y_i) =$
    $min\{f(l_j,m_j) + g(n_k,o_k) \mid l_j + n_k \le x_i \wedge m_j \le y_i \wedge o_k \le y_i\}$
 1: for $y_i = 0$ to $M$ do
 2:     $min = \infty$
 3:     for $x_i = 0$ to $M - y_i$ do
 4:         for $tmp = 0$ to $x_i$ do
 5:             $l_j = tmp$, $m_j = y_i$
 6:             $n_k = x_i - tmp$, $o_k = y_i$
 7:             if $f(l_j,m_j) + g(n_k,o_k) < min$ then
 8:                 $min = f(l_j,m_j) + g(n_k,o_k)$
 9:         $h(x_i,y_i) = min$
10: return $h(x,y)$
```

Algorithm 5 Hybrid

```
Require: Energy functions $f^H_1(x,y), \dots, f^H_n(x,y)$
Ensure: $h^H_n : [0,M] \times [0,M] \to \mathbb{R}$, where
    $h^H_n(x,y) = min\{\sum_{k=1}^n f^H_k(x_k,y_k) \mid \sum_{k=1}^n [x_k] \le x \wedge \forall k : y_k \le y\}$
 1: if $n > 1$ then
 2:     $h^H_{n-1}(x,y) = Hybrid(f^H_1(x,y), \dots, f^H_{n-1}(x,y))$
 3:     $h^H_n(x,y) = HybBinMin(h^H_{n-1}(x,y), f^H_n(x,y))$
 4: else
 5:     $h^H_n(x,y) = HybBinMin(f^H_1(x,y), Z(x,y))$ \*$Z(x,y) = 0\ \forall x,y$ *\
 6: return $h^H_n(x,y)$
```

3.4. Hybrid Approach

The Hybrid approach combines the presented scratchpad allocation approaches. It partitions the scratchpad into disjoint regions each allocated to one process, and one overlapping region which is commonly used by all processes. The most frequently accessed memory objects of a process assigned to the disjoint region cause no copy overhead, while the other important memory objects assigned to the overlapping region cause a tolerable copy overhead.

The Hybrid approach assumes that the hybrid energy function $f^H_k(x,y)$ for each process P_k is known. These energy functions are used to compute the hybrid energy consumption of the application $h^H_n(x,y)$:

$$h^H_n(x,y) = min\{\sum_{k=1}^n [f^H_k(x_k,y_k) | \sum_{k=1}^n x_k \le x \wedge \forall k : y_k \le y]\} \quad (7)$$

$h^H_n(x,y)$ is the minimum sum of the hybrid energy function $f^H_k(x_k,y_k)$ for all processes P_k, respecting the disjoint and the overlapping region constraints. Similar to the Non-Saving approach, the hybrid binary minimum function *hybbinmin* is defined as:

$$\begin{aligned} h(x,y) &= hybbinmin(f,g) = \\ & min\{f(l_j,m_j) + g(n_k,o_k)|l_j + n_k \le x \wedge m_j \le y \wedge n_k \le y\} \end{aligned} \quad (8)$$

hybbinmin is used to convert the energy function $h^H_n(x,y)$ to the following recurrence equation:

$$\begin{aligned} h^H_1(x,y) &= hybbinmin(f^H_1(x,y), Z(x,y)) \\ h^H_n(x,y) &= hybbinmin(h^H_{n-1}(x,y), f^H_n(x,y)) \end{aligned} \quad (9)$$

where $Z(x,y)$ is the zero function. Algorithms for the Hybrid approach are similar to the respective algorithms for the Non-Saving approach. Algorithm 4 presents the pseudo-code for implementing the hybrid binary minimum func-

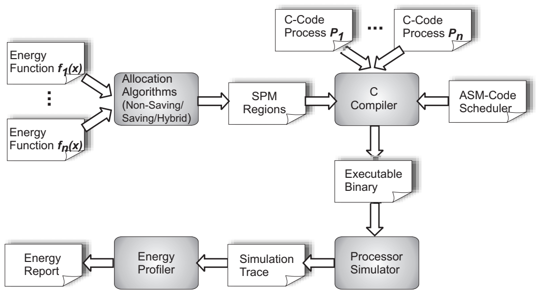


Figure 2: Experimental Workflow

tion. Algorithm 5 implements the above recurrence equation, requiring $O(nM^3)$ time to compute the hybrid energy function $h_n^H(x,y)$ of the application.

4. Experimental Setup

Our experimental setup consists of an ARM7 processor core with a 4kB onchip scratchpad and an offchip main memory. A custom operating system was created to manage the application on the processor. It provides system calls for managing the contents of the scratchpad. The energy function $f_k^N(x)$ for a process P_k is determined by computing the energy consumption values of the process for each scratchpad size. An energy optimal non-overlay based allocation algorithm [8] is used to allocate memory objects (*i.e.* instruction segments and variables) onto the scratchpad. Accurate energy models [7, 3] are used to compute the energy consumption of the application. The Hybrid energy function $f_k^H(x,y)$ of the application is computed in a similar manner. For our benchmarks, computing the energy functions over the range of [0, 4096] bytes at a granularity of 16 bytes required a maximum of 130 CPU seconds and 1000 CPU seconds on a Sun Sparc 1300MHz machine for $f_k^N(x)$ and $f_k^H(x,y)$, respectively. The computation of energy functions takes ample computation time, however, the database of the process energy functions can later be reused for many applications. We evaluated the proposed allocation techniques for statically scheduled multiprocess systems. All processes have equal priority and are scheduled in a round-robin manner. Additionally, we assumed that the execution time and energy consumption of each process can be computed. A time slice of 33000 CPU cycles (or 1 ms on the 33Mhz ARM7 processor) is provided to each process for execution on the processor.

The main workflow shown in figure 2 assumes that the energy functions for all processes are known. The energy functions are passed to the presented allocation algorithms. The output consists of the assignment of scratchpad region(s) to each process. The source code of each process is then compiled and the optimal set of memory objects are marked for allocation onto the assigned scratchpad regions. The assembly codes of the process and that of the operating system are assembled and linked to create a single executable binary. The executable is executed on ARMulator (the simulator from ARM Ltd.) to generate the instruction trace which is then passed as input to the energy profiler.

Application Name	Size (kB)	Processes (benchmarks)
Media	77	adpcm, g721, mpeg4, edge-detection
Mobile	80	gsm, mpeg4
DSP	26	fast-idct, lattice-init, lattice-small, fft, fir,

Table 5: Multiprocess applications

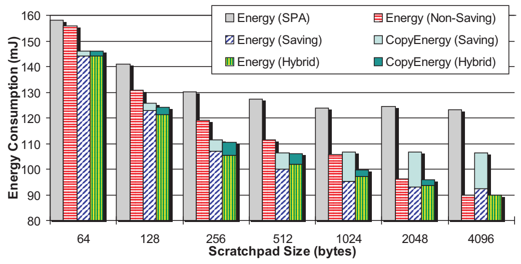


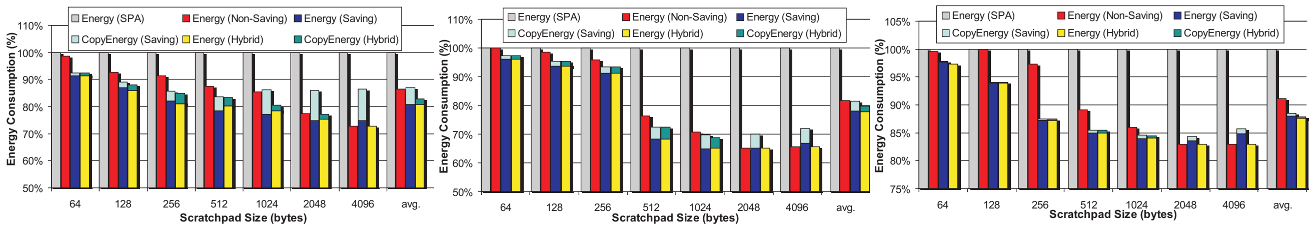
Figure 3: Energy consumption of Media application for SPA, Non-Saving, Saving and Hybrid Approach

The energy profiler utilizes accurate energy models [7, 3] to compute the total energy consumed by the application.

5. Results

The proposed allocation approaches are evaluated for an assorted set of multiprocess applications presented in Table 5. Lacking a benchmark suite consisting of multiprocess applications, we assembled well-known benchmarks as processes to construct representative multiprocess applications. Applications *Media* and *Mobile* comprise benchmarks from Mediabench, while *DSP* comprises benchmarks from the UTDSP benchmark suite. The benefits of sharing the scratchpad among multiple processes are evaluated by comparing them against a single process allocation (SPA) approach. The SPA approach utilizes the Non-Saving approach for that process which leads to the maximum reduction in the energy consumption of the application.

Comparing the allocation approaches (Non-Saving, Saving and Hybrid) and the SPA approach against each other, the energy overhead due to the copy routines for Saving and Hybrid approaches is demarcated from the corresponding aggregate energy values of the approaches in Figure 3. The energy consumption values for the proposed approaches are always better than those for the SPA approach. This justifies the use of the multiprocess allocation approaches for minimizing the energy consumption of the multiprocess applications. Also, the energy consumption of the system for Non-Saving and Hybrid approaches decreases monotonically with the increase in the scratchpad size. However, the energy values for the Saving approach first decrease and then remain constant for all scratchpads larger than 512 bytes since it does not always utilize the entire scratchpad, as high energy overhead is incurred due to the copy routines for large scratchpads. Finally, for small scratchpad sizes (64-512 bytes) the energy values for the Saving approach are smaller than those for the Non-Saving approach, while the opposite is true for larger scratchpads. For small scratchpads, the improved utilization of the scratchpad due to the Saving approach and the small copy energy overhead



(a) Media application (b) DSP application (c) Mobile application

Figure 4: Relative energy consumption of applications for Non-Saving, Saving and Hybrid Approach vs. SPA

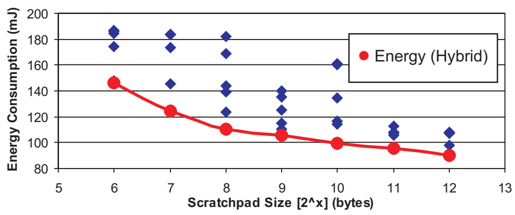


Figure 5: Pareto curve for Media application

minimizes the energy consumption of the system. The Hybrid approach distributes the scratchpad into disjoint and overlapping regions and therefore achieves the minimum energy consumption values for all scratchpad sizes.

Next, we compare the multiprocess allocation approaches against the SPA approach for Media, DSP and Mobile applications. Figure 4 presents the energy values for the proposed approaches relative to those of the SPA approach (presented as the 100% bars). From Figure 4(a) and 4(b), we observe that the Hybrid approach is able to reduce energy consumption up to 27% and 35% against the SPA approach, respectively. For the Media application, we report average energy reductions of 14%, 13% and 17% due to Non-Saving, Saving and Hybrid approaches, respectively. Even higher average energy reductions of 18%, 19% and 20% due to Non-Saving, Saving and Hybrid approaches, respectively, are observed for the DSP application. For the Mobile application, average energy reductions (cf. Figure 4(c)) of 9%, 11% and 12% are reported.

Finally, we would like to discuss the generation of Pareto-optimal curves for the multiprocess application by the proposed approaches. In addition to the optimal allocation of the application onto an M bytes scratchpad, the energy functions ($h_n^N(x)$, $h_n^S(x)$ and $h_n^H(x,y)$) defined over the range $[0,M]$ bytes of scratchpad sizes are also determined. These energy functions represent the Pareto-optimal curves for the application. Figure 5 presents the curve of the hybrid energy function $h_n^H(x)$ for the Media application. The remaining points represent the energy consumption values for promising allocations of scratchpad to the processes. However, they represent non-optimal allocations, as they consume more energy than those using the Hybrid approach. The Pareto-optimal curve aids the system designer in performing design-time scratchpad size vs. energy consumption tradeoffs. For example, a system with a 4kB scratchpad consumes just 10% less energy than a system with 2kB scratchpad. A system designer can thus tradeoff 10% energy consumption for half of the onchip scratchpad size.

6. Conclusion and Future Work

In this paper, a set of strategies (*viz.* Non-Saving, Saving and Hybrid) were proposed for sharing the scratchpad by the processes of the multiprocess applications. In addition, the Pareto-optimal curves allowing design-time exploration were generated. The Hybrid approach provided the maximum energy reductions for all scratchpad sizes, though it required the longest computational time for preprocessign step. The Saving approach was demonstrated to be better than the Non-Saving approach for small scratchpads and vice-versa for large scratchpads. The proposed approaches report average energy reductions of 9%-20% against a single process approach. In future, we intend to extend the approaches for processes with priorities and deadlines.

References

[1] F. Angiolini, M. Francesco, F. Alberto, L. Benini, and M. Olivieri. A Post-Compiler Approach to Scratchpad Mapping of Code. In *Proc. of CASES'04*, Sept. 2004.

[2] Appendix. . `http://ls12.cs.uni-dortmund.de/~verma/ESTIMedia2005/appendix`.

[3] R. Banakar, S. Steinke, B.-S. Lee, M. Balakrishnan, and P. Marwedel. Scratchpad Memory: A Design Alternative for Cache On-chip Memory in Embedded Systems. In *Proc. of Intl. Sym. on CODES*, Col., USA, May 2002.

[4] P. Francesco, P. Marchal, D. Atienza, L. Benini, F. Catthoor, and M. Mendias. An Integrated Hardware/Software Approach for Run-Time Scratchpad Management. In *Proc. of DAC*, San Deigo, CA, USA, June 2004. DAC.

[5] M. Kandemir and A. Choudhary. Compiler-Directed Scratch Pad Memory Hierarchy Design and Management. In *Proc. of DAC*, New Orleans, USA, June 2002.

[6] P. R. Panda, N. Dutt, and A. Nicolau. *Memory Issues in Embedded Systems-On-Chip*. Kluwer Academic Publishers, MA, 1999.

[7] S. Steinke, M. Knauer, L. Wehmeyer, and P. Marwedel. An Accurate and Fine Grain Instruction-Level Energy Model Supporting Software Optimizations. In *Proc. of PATMOS*, Switzerland, Sep. 2001.

[8] S. Steinke, L. Wehmeyer, B. S. Lee, and P. Marwedel. Assigning program and data objects to scratchpad for energy reduction. In *Proc. of DATE*, Paris, France, Mar. 2002.

[9] M. Verma, L. Wehmeyer, and P. Marwedel. Cache-aware Scratchpad Allocation Algorihm. In *Proc. of DATE*, Paris, France, Feb, 2004.

[10] M. Verma, L. Wehmeyer, and P. Marwedel. Dynamic Overlay of Scratchpad Memory for Energy Minimization. In *Proc. of CODES+ISSS*, Stockholm, Sweden, Sep. 2004.

[11] E. W. Weisstein. *"Function": MathWorld-A Wolfram Web Resource*. http://mathworld.wolfram.com/Function.html.