Universität Dortmund
Lehrstuhl Informatik XII
44221 Dortmund

Comparison of Cache- and Scratch-Pad based Memory Systems with respect to Performance, Area and Energy Consumption

Rajeshwari Banakar, Stefan Steinke, Bo-Sik Lee,
M. Balakrishnan, Peter Marwedel

Technical Report # 762

28.9.2001

Comparison of Cache- and Scratch-Pad based Memory Systems with respect to Performance, Area and Energy Consumption

Rajeshwari Banakar, Stefan Steinke, Bo-Sik Lee, M. Balakrishnan, Peter Marwedel

{steinke | lee | marwedel}@ls12.cs.uni-dortmund.de

University of Dortmund, Dept. of Computer Science

Otto-Hahn-Strasse 16

44221 Dortmund, Germany

{banakar | mbala}@cse.iitd.ernet.in

Indian Institute of Technology, Delhi

Abstract

In this report we evaluate the options for low power on-chip memories during system design and configuration. Specifically, we compare the use of scratch pad memories with that of cache on the basis of performance, area and energy. The target architecture used in our experiments is the AT91M40400 microcontroller containing an ARM7TDMI core. A packing algorithm is used to map the memory objects of the benchmarks to the scratch pad. Area and energy for different scratch pad and cache sizes are computed using the CACTI tool while performance is derived using the trace results of the ARMulator. We observe area and performance improvements by using a scratch pad memory. For example, for bubble sort there is a performance improvement of 18% from a hardware which needs 34% less area. The scratch pad also needs less energy per access, due to the absence of tag comparison.

1 Introduction

Processor-based information processing can be used conveniently in many embedded systems. The detailed architecture of such systems is very much application driven to meet power and performance requirements. In particular, this applies to the memory architecture, since this architecture does have a major impact on the overall performance and the power consumption. In this report we analyze the effect of different options that exist for the design of memory architectures.

In order to work with realistic values, we are using an existing processor core, the ARM7TDMI, as an example. The ARM7TDMI core has found many applications in embedded systems. A typical example is a mobile telephone handset, where the core incorporates digital signal processing hardware and associated memories. In our work we use an AT91M40400 chip comprising an ARM7TDMI core, on-chip RAM-based scratch-pad memory (SPM) and peripherals. The scratch pad memory is a memory which is mapped into the processor's address space. The SPM is

involved whenever addresses within the SPM's address window are used. There are two key advantages of using the SPM: references to the SPM are faster and they consume less energy than references to larger or off-chip memories.
In a recent report [12], we have shown that a major amount of references to the main memory can be replaced by references to a faster, less energy hungry scratch pad memory if a compiler statically allocates instructions and data to the scratch pad. In that report, no comparison with caches was included. In the current report, we explore the benefits of the on-chip scratch pad memory vis-a-vis on-chip caches. On-chip caches using static RAM consume power in the range of 25% to 45% of the total chip power [3]. For example, in the StrongARM 110 the cache consumes 43% [14]. A number of researchers have concentrated on power reduction of caches both at the device level, and at the architectural level. Techniques like sub-banking and block buffering which reduce the cache power consumption at the organizational level, are described in [3]. In this report, we are proposing to add scratch pad memories to the memory system to reduce the power consumption.
Recently, interest has been focused on having on-chip scratch pad memory to reduce power consumption and improve performance. Scratch pad memories are considerably more energy and area efficient than caches. On the other hand, they can replace caches only if they are supported by an effective compiler. The mapping of memory elements of the application benchmarks to the scratch pad memory can be done only for data [5], only for instructions [17] or for both data and instructions [12]. Current embedded processors particularly in the area of multimedia applications and graphics controllers have on-chip scratch pad memories. Panda et.al [5] associate the use of scratch pad memory with data organization and demonstrate its effectiveness. Benini et.al [7] focus on memory synthesis for low power embedded systems. They map the frequently accessed locations onto a small application specific memory.
In our work we address the on-chip memory configuration and selection issue. Specifically, we make the following contribution.

- We present a comparison of on-chip scratch pad memories and on-chip caches on the basis of area, performance and energy consumption. Such a comparison helps the designer to make effective decisions at the architectural level.
  This comparison is carried out by evaluating performance through simulation and estimating area and energy using the scratch pad and cache memory models over a range of scratch pad and cache sizes. The performance and energy results for a specific application also evaluate the technique incorporated into the compiler for identifying and packing the most suitable instruction and data blocks into scratch pad memory. Results clearly show scratch pad memory to be a promising alternative to caches in many embedded applications.

The rest of the report is organized as follows: In section 2 we give an overview of the methodology that we have used. Section 3 describes the mapping of the memory elements to the scratch pad. In section 4, we present energy and area models of the caches and the scratch pad. Section 5 covers the method used for performance prediction. In section 6 we describe the experimental results followed by the conclusion and future work in section 7. In Appendix A we present some experimental data obtained using the CACTI tool.

2 Overview of the methodology

In this section we explain the methodology used in our work to explore the benefits of using on-chip scratch pad memory. We use the AT91M40400 as our target architecture. The AT91M40400 is a member of the ATMEL AT91 16/32 bit microcontroller family based on the ARM7TDMI embedded processor. This processor is a high performance

RISC with a very low power consumption. It has an on-chip scratch pad memory of 4 KBytes. The AT91M40400 can be used as a cost effective solution in many computationally intensive applications in the default configuration[10]. The ARM7TDMI comes with a 32 bit data path and two instruction sets. The ARM instruction set is a 32 bit instruction. The THUMB instruction set is a 16 bit instruction set that has a better code density than the ARM instruction set. Instructions can be simulated using the ARMulator [11] instruction set simulator.

The main goal of our work is to achieve a low power on-chip memory configuration. To study the benefits of using a scratch pad memory, we have to estimate the area, performance and energy consumption of the on-chip caches and scratch pad memory and compare them.

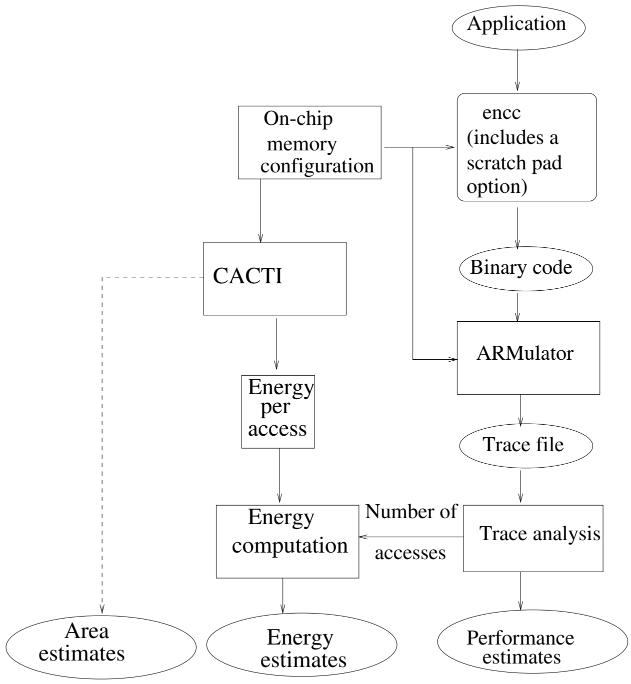


Figure 1: Memory configuration flow diagram

Fig. 1 shows the flow diagram of our evaluator. The encc compiler generates the code for the ARM7 core. It is a research compiler used for exploring the design and new optimization techniques. The input to this compiler is an application benchmark written in C. Constant propagation, copy propagation, dead code elimination, constant folding, jump optimization and common subexpression elimination are supported standard optimizations. The compiler is an energy aware compiler supporting, for example, the selection of instructions based on their energy consumption. This data can be built using the instruction level power model as referred by Steinke et al. [18]. As a post pass option, encc uses a special packing algorithm, known as the knapsack algorithm, for assigning code and data blocks to the scratch pad memory. The result is that blocks of instructions and data which are frequently accessed, and are likely to generate maximum energy savings, are assigned to the scratch pad. This algorithm is discussed in detail in section 3.

The output of the compiler is a binary ARM code which can be simulated by the ARMulator to produce a trace file. For on-chip cache configuration, the ARMulator accepts the cache size as parameter and generates the performance as the number of cycles. The on-chip memory configuration has an impact on the number of wait states for accessing a

location which are added during the post analysis of the trace file. This is explained in section 5.
The predicted area and energy for the cache is based on the CACTI [9] model for 0.5 μm technology. The same model has been used with the hardware pruned to predict area and energy of scratch pad memory. This is discussed in section 4.

3 Scratch pad packing algorithm

For the use of the scratch pad, an algorithm in the compiler is responsible for the selection of program parts and data to be moved to the limited on-chip memory. In this section we explain how this algorithm assigns a set of functions, basic blocks and/or variables to the scratch pad on a static basis. First we describe the identification and evaluation of program parts. Then, the identification and evaluation of data memory objects and finally the selection of the best set of memory objects is explained.

3.1 Program memory objects

Off-chip memory accesses account for high energy consumption and reduction of performance because of the necessary wait states and the number of gates which are activated to read or write the data.
In this report, we try to identify functions and basic blocks that should be assigned to the scratch pad memory. A basic block is a consecutive sequence of instructions without any jump instruction. The end of a basic block is always defined by a jump or by a following instruction which is a jump destination from another program instruction. All instructions of an individual basic block are executed the same number of times, so it is sufficient to analyze the number of executions for the whole basic block.
The identification of functions and basic blocks which can be assigned to the scratch pad memory is shown for the example code `biquad_N_sections` (see fig. 2) from the DSPStone benchmark suite [16].
The execution of each function is started at its beginning and is terminated by a return statement. There are no further jumps into a function. Thus, each function can be handled as one memory object which possibly can be moved and which does not require changing any of the included instructions.
The calling hierarchy of our program `biquad_N_sections` can be represented by a call graph (see fig. 3). This call graph has the body of the main program as its root. At the next level, we have all code blocks (basic blocks or functions) that are called from the main program. In our case, function `pin_down` is the only function at this level. One more level down, we have all blocks that can be called from function `pin_down`.
For each memory block i, we can compute the energy consumption of all executions by multiplying the energy consumption of a single execution $E_{instr\_fetch,i}$ with the number of executions n_i of this memory object:

$$E(BB_i) = n_i * E_{instr\_fetch,i}$$

This energy consumption can be reduced by a relocation of the function or basic block to on-chip scratch pad memory. The best set of functions and/or basic blocks are chosen by our algorithm described in section 3.3.
Moving functions does not require any change of its opcodes, but moving basic blocks requires the addition of jump instructions to jump from blocks mapped to regular memory to blocks stored in the scratch pad memory and back. The

```
int pin_down(int x, int coefficients[], int wi[]) {
  int f;

// pd_1:
  for (f = 0; f < 20; f++)
// pd_2:
    coefficients[f] = 7;
// pd_3:
  for (f = 0; f < 8; f++)
// pd_4:
    wi[f] = 0;
// pd_5:
  return (1);
}

int main() {
  int w, f;
  int *ptr_coeff, *ptr_wi1, *ptr_wi2;
  int wi[8];
  int coefficients[20];
  int x,y;

// main_1:
  ptr_coeff = &coefficients[0];
  ptr_wi1 = &wi[0];
  ptr_wi2 = &wi[1];
  x = pin_down(x, coefficients, wi);
// main_2:
  y = x;
  for (f = 0; f < 4; f++) {
// main_3:
      w  = y -  *ptr_coeff++ * *ptr_wi1 ;
      w -= *ptr_coeff++ * *ptr_wi2;
      y  = *ptr_coeff++ *  w;
      y += *ptr_coeff++ * *ptr_wi1;
      y += *ptr_coeff++ * *ptr_wi2;
      *ptr_wi2++ = *ptr_wi1;
      *ptr_wi1++ = w;
      ptr_wi2++;
      ptr_wi1++;
  }
// main_4:
  pin_down(y,coefficients,wi);
// main_5:
  return(y);
}
```

Figure 2: `biquad_N_sections` program code

jump instructions are an overhead especially if there are a number of small basic blocks. In order to minimize jump instructions, moving consecutive basic blocks is preferred.

Note that the program is statically allocated to the scratch pad memory. There is no dynamic reloading of memory blocks even though this could be useful for long programs having more hot spots than the SPM can accommodate. The extension to dynamic reloading (a kind of program-controlled paging) will be part of the future work.

3.2 Data memory objects

Apart from the program, variables can be allocated to the SPM, too. Each variable is viewed as one data memory object. The number of accesses of a variable $acc(v)$ is the sum of the number of accesses $acc_i(v)$ in each of the blocks i. $acc_i(v)$ is computed as the number of static references $stat_i(v)$ to variable v in block i times the number of executions of block i:

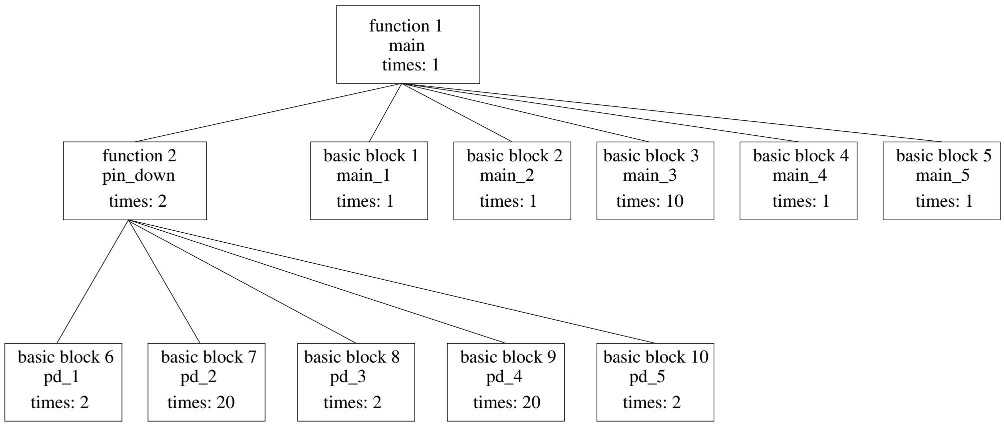


Figure 3: Call Graph of functions and basic blocks in `biquad_N_sections`

$$acc(v) \quad = \quad \sum_i acc_i(v) = \sum_i stat_i(v) * n_i$$

For example, in the benchmark `biquad_N_sections` the array `wi` is accessed 7 times in `main_3` and once in `pd_4`. The total number of accesses can thus be computed as follows:

$$acc(\mathtt{wi}) \quad = \quad 7 * n_{main\_3} + 1 * n_{pd\_4} = 7 * 10 + 1 * 20 = 90$$

For the energy $E(v)$ consumed by all the accesses to the variable, this number of accesses has to be multiplied by the energy cost $E_{dataword}$ of a single memory access with a load or store instruction:

$$E(v) \quad = \quad acc(v) * E_{dataword}$$

After the identification and evaluation of the energy costs for program and data memory objects, the best set of objects has to be chosen to fit in the limited space of the scratch pad.

3.3 Selection of memory objects for the scratch pad

The best set of memory objects which fit into the scratch pad and save the highest amount of energy now has to be identified. Moving a certain memory object to the scratch pad will result in a certain gain in terms of saved energy. Moving has to be such that the combined size of the memory objects does not exceed the size of the SPM. The size of each memory object is independent from the other objects. Maximizing the total gain is a problem called **knapsack problem** [19].

Our formulation of the problem uses the following definitions concerning moving basic blocks, functions and variables:

$$
\begin{aligned}
E(BB_i) &= \text{energy consumption for instruction fetches in basic block } BB_i \\
S(BB_i) &= \text{size of basic block } BB_i \\
m(BB_i) &= \text{1 if } BB_i \text{ is moved to the scratch pad, = 0 otherwise} \\
E(F_j) &= \text{energy consumption for instruction fetches in function } F_j \\
S(F_j) &= \text{size of function } F_j \\
m(F_j) &= 1, \text{if } F_j \text{ is moved to the scratch pad, = 0 otherwise} \\
E(var_k) &= \text{energy consumption for all loads or stores of variable } var_k \\
S(var_k) &= \text{size of variable } var_k \\
m(var_k) &= \text{1 if } var_k \text{ is moved to the scratch pad, = 0 otherwise}
\end{aligned}
$$

To optimize the energy saving, the following cost function needs to be maximized:

$$energy\_saving = \sum_{i \in I} m(BB_i) * E(BB_i) + \sum_{j \in J} m(F_j) * E(F_j) + \sum_{k \in K} m(var_k) * E(var_k) \tag{1}$$

Index sets I, J, and K correspond to index values for all basic blocks, all functions and all variables, respectively. The size constraint can be modeled as follows:

$$\sum_{i \in I} m(BB_i) * S(BB_i) + \sum_{j \in J} m(F_j) * S(F_j) + \sum_{k \in K} m(var_k) * S(var_k) \leq scratchpadsize \tag{2}$$

Taking jump instructions into account turns the problem into a multidimensional knapsack problem. This can be solved using integer linear programming [2] with a set of equations which are presented in this section.

To account for the saved jump instructions between consecutive basic blocks, additional memory objects have to be used for all possible combinations of consecutive basic blocks. For example, for `biquad_N_sections` the combinations for function `pin_down` are as follows:

```
BB_11 = (pd_1, pd_2)
BB_12 = (pd_1, pd_2, pd_3)
BB_13 = (pd_1, pd_2,pd_3, pd_4)
BB_14 = (pd_1, pd_2,pd_3, pd_4,pd_5)

BB_15 = (pd_2, pd_3)
BB_16 = (pd_2, pd_3, pd_4)
BB_17 = (pd_2, pd_3, pd_4, pd_5)

BB_18 = (pd_3, pd_4)
BB_19 = (pd_3, pd_4, pd_5)

BB_20 = (pd_4, pd_5)
```

For all these combinations of basic blocks, memory objects are constructed.

To prevent a basic block from being selected twice, e.g. as a single basic block and as part of a function or a multi basic block memory object, the equations of the following type (here for `pd_1`) have to be added:

$$m(BB_6) + m(F_2) + m(BB_{11}) + m(BB_{12}) + m(BB_{13}) + m(BB_{14}) \leq 1 \tag{3}$$

Based on the above equations, an IP solver [2] can find the best possible solution for the use of a scratch pad memory in an average runtime of 1.4 s. The chosen memory objects can then be placed in the scratch pad memory.
Fig. 4 shows the final mapping of memory components to the scratch pad and the main memory for the biquad benchmark example.

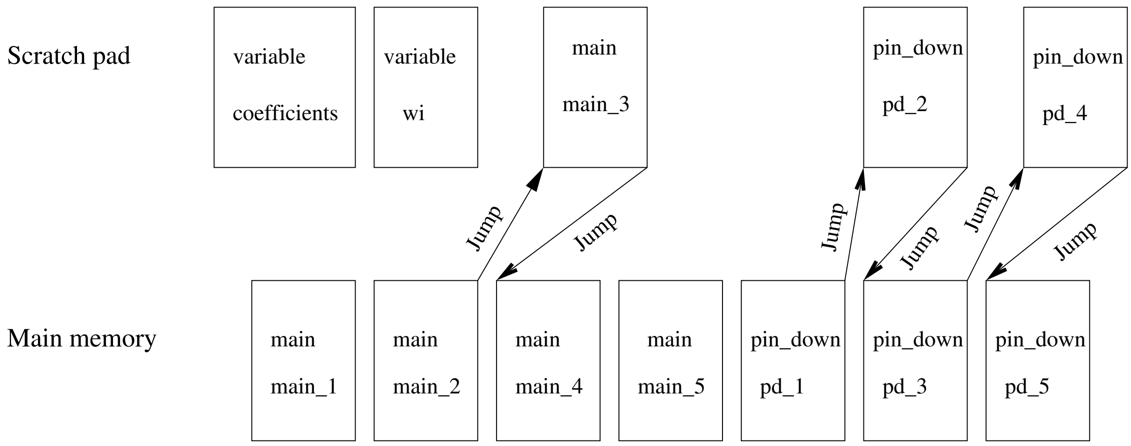


Figure 4: Allocation of basic blocks and variables to scratch pad and insertion of connecting jumps

In the next section we describe the energy models of the on-chip memories used in our approach.

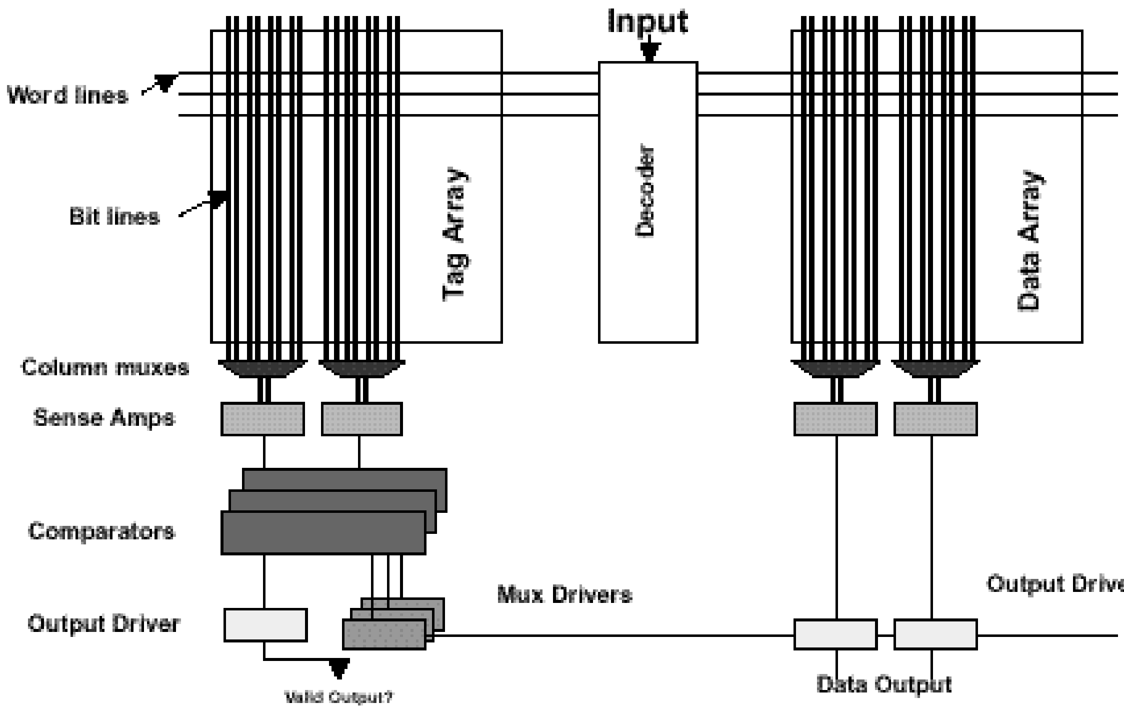
4 On-chip memory area and energy models

The on-chip energy model of the cache and the scratch pad are essential to compare the energy estimates of both. We will first describe the analytical cache energy model and then the scratch pad energy model.

4.1 Cache and Scratch pad organization

4.1.1 Cache power model

The basic organization of the cache taken from [8, 9] is shown in Fig. 5.
The main components are the decoder, the tag memory array, the tag column multiplexers, the tag sense amplifiers, the tag comparators, the tag output drivers, the data memory array, the data column multiplexers, the data sense amplifiers and the data output drivers.
The estimation of power can be done at different levels, from the transistor level to the architectural level [13]. In CACTI [9], transistor level power estimation is done. The total energy consumption in a cache is the sum of energy consumptions of all the components identified above and is obtained from CACTI.



,

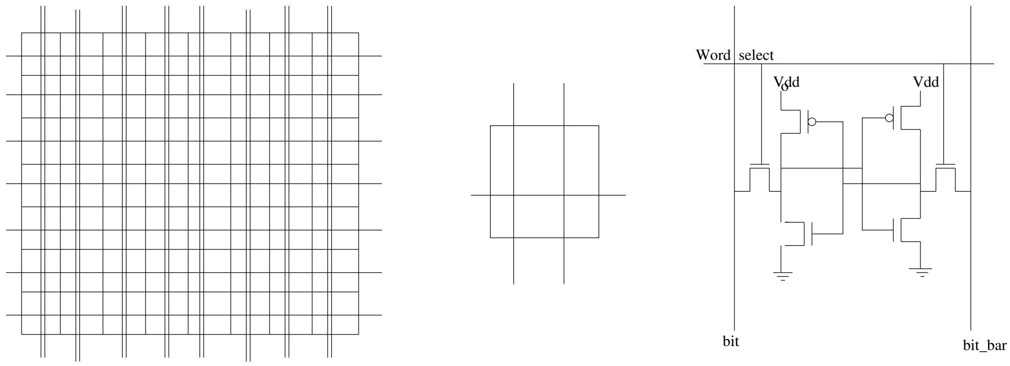
Figure 5: Cache Memory organization

4.1.2 Scratch pad power model

The scratch pad is a memory array with the decoding and the column circuitry logic. This model is designed keeping in view that the memory objects are mapped to the scratch pad in the last stage of the compiler. Thus, we need not check for the availability of the data/instruction in the scratch pad. It reduces the comparator and the signal miss/hit acknowledging circuitry. This contributes to the energy as well as area reduction.
The scratch pad memory array cell is shown in Fig. 6(a) and the memory cell in 6(b).
The 6 transistor static RAM cell is shown in Fig 6(c). The cell has one R/W port. Each cell has two bit-lines, bit and bit bar, and one word-line. The column circuitry consists of the sense amplifiers, column multiplexers, pre-charge unit and the output driver circuit.
The decoder circuit for a 1 kbyte SPM is as shown in Fig. 7. Fig. 7(a) shows the circuit with 2-to-4 decoder logic. Fig. 7(b) shows the decoder using 3-to-8 logic.
The complete scratch pad organization is as shown in Fig. 8.
The scratch pad memory energy consumption can be estimated from the energy consumption of its components i.e. decoder $E_{decoder}$ and memory columns E_{memcol}.

$$E_{scratchpad} = E_{decoder} + E_{memcol} \tag{4}$$

Energy in the memory array consists of the energy consumed in the sense amplifiers, column multiplexers, the output driver circuitry, and the memory cells due to the word-line, pre-charge circuit and the bit line circuitry. The energy



(a) Memory array

(b) Memory cell

(c) Six transistor Static RAM

Sense amplifiers, Column Mux
Pre-charge circuit, Output drivers

(d)
Column circuitry

Figure 6: Scratch memory array

consumption in the decoding logic is around 10% to 15% of the total power dissipation as observed in all our on-chip memory configurations. The major energy consumption is due to the memory array unit. The procedure followed in the CACTI tool to estimate the energy consumption is to first compute the capacitances for each unit. Then, energy is estimated. As an example we only describe the energy computation for the memory array. Similar analysis is performed for the decoder circuitry also, taking into account the various switching activity at the inputs of each stage. Let us consider the energy dissipation E_{memcol}. It consists of the energy dissipated in the memory cell. Thus

$$E_{memcol} = C_{memcol} * V_{dd}^2 * P_{0->1} \tag{5}$$

C_{memcol} in equation (5) is the capacitance of the memory array unit. $P_{0->1}$ is taken as 0.5

$$C_{memcol} = ncols * (C_{pre} + C_{readwrite}) \tag{6}$$

C_{memcol} is computed from equation (6). It is the sum of the capacitances due to pre-charge and read access to the scratch pad memory. C_{pre} is the effective load capacitance of the bit-lines during pre-charging and $C_{readwrite}$ is the effective load capacitance of the cell read/write. $ncols$ is the number of columns in the memory.

In the preparation for an access, bit-lines are pre-charged and during actual read/write, one side of the bit lines are pulled down. Energy is therefore dissipated in the bit-lines due to pre-charging and the read/write access. When the scratch pad memory is accessed, the address decoder first decodes the address bits to find the desired row. The transition in the address bits causes the charging and discharging of capacitances in the decoder path. This brings

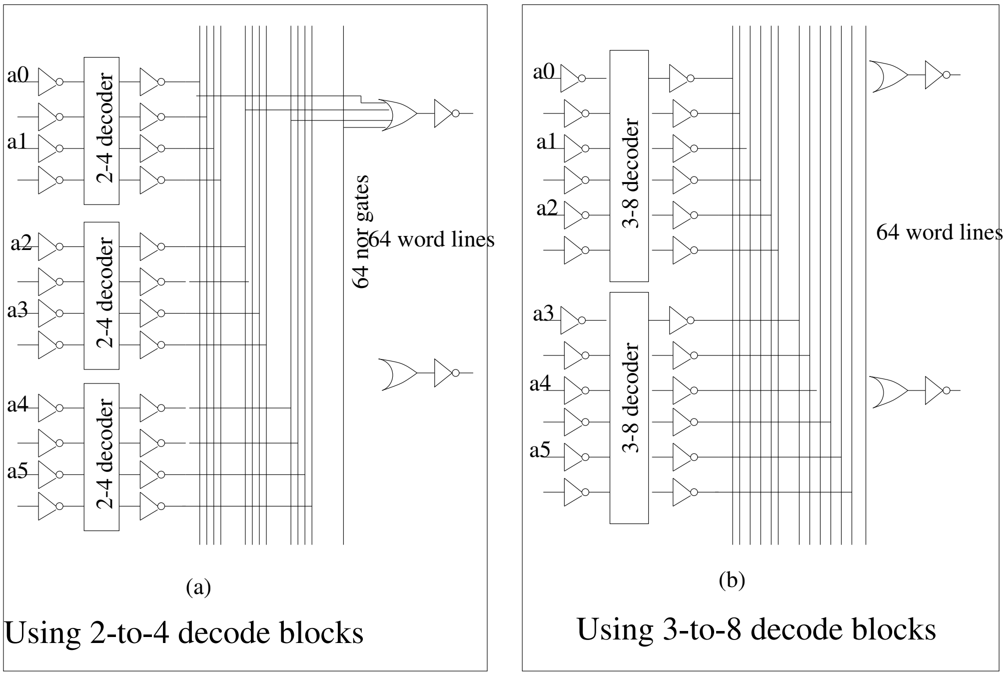


Figure 7: Decoder circuit for 1k scratch pad memory

about energy dissipation in the decoder path. The transition in the last stage, that is the word-line driver stage triggers the switching in the word-line. Regardless of how many address bits change, only two word-lines among all will be switched. One will be logic 0 and the other will be logic 1. The equations are derived based on [9] [3] [8].

$$E_{sptotal} = SP_{access} * E_{scratchpad} \quad (7)$$

where $E_{sptotal}$ is the total energy spent in the scratch pad memory. In case of a scratch pad as a contrast to cache we do not have events due to write miss and read miss. This is because of the mapping algorithm that is used in the encc compiler's last phase. The only possible case that holds good is the read hit. SP_{access} is the number of read hits which we obtain from the simulator for the scratch pad configuration for the AT91M40400 microcontroller. $E_{scratchpad}$ is the energy per access obtained from our analytical scratch pad model.

4.2 Area model

The area model that we use in our work is based on the transistor count in the circuitry. All transistor counts are computed from manual designs of the circuits. The area model of the cache is described followed by the explanation of the scratch pad memory area model.

4.2.1 Cache Area Model

From the organization shown in Fig. 5, the area of the cache (Ac) is the sum of the area occupied by the tag array (A_{tag}) and data array (A_{data}).

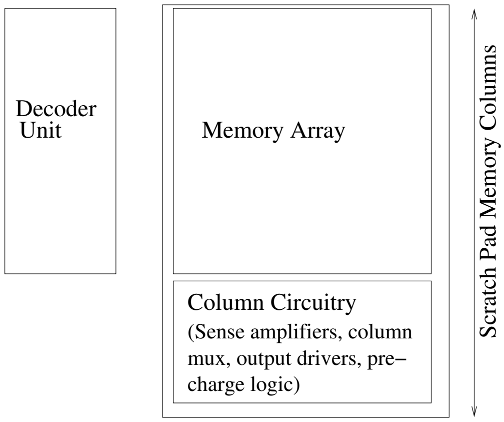


Figure 8: Scratch pad memory organization

$$A_c = A_{tag} + A_{data} \tag{8}$$

A_{tag} and A_{data} is computed using the area of its components.

$$A_{tag} = A_{deco-tag} + A_{tag-array} + A_{colmux-tag} + A_{pre-tag} + A_{sense-tag} + A_{comp} + A_{muxdrv-tag} \tag{9}$$

where $A_{deco-tag}$, $A_{tag-array}$, $A_{colmux-tag}$, $A_{pre-tag}$, $A_{sense-tag}$, A_{comp} and $A_{muxdrv-tag}$ is the area of the tag decoder unit, tag array, column multiplexer, the pre-charge, sense amplifiers, tag comparators and multiplexer driver units respectively.

$$A_{data} = A_{deco-data} + A_{data-array} + A_{colmux-data} + A_{pre-data} + A_{sense-data} + A_{outputdrv} \tag{10}$$

where $A_{deco-data}$, $A_{data-array}$, $A_{colmux-data}$, $A_{pre-data}$, $A_{sense-data}$, $A_{outputdrv}$ is the area of the data decoder unit, data array, column multiplexer, pre-charge, data sense amplifiers and the output driver units respectively.

4.2.2 Scratch pad area model

From the organization shown in Fig. 8, the area of the scratch pad is the sum of the area occupied by the decoder, data array and the column circuit. Let A_s be the area of the scratch pad memory.

$$A_s = A_{deco-data} + A_{data-array} + A_{colmux-data} + A_{pre-data} + A_{sense-data} + A_{outputdrv} \tag{11}$$

where $A_{deco-data}$, $A_{data-array}$, $A_{colmux-data}$, $A_{pre-data}$, $A_{sense-data}$ and $A_{outputdrv}$ is the area of the data decoder, data array area, column multiplexer, pre-charge, data sense amplifiers and the output driver units respectively.

5 Performance model

Clock cycle estimation is based on the ARMulator trace output for cache. This is assumed to directly reflect performance: the larger the number of clock cycles the lower the performance. This holds only when the change in the on-chip memory configuration (cache/scratch pad memory and its size) does not change the clock period because, for example, these units are not in the critical path for the set of configurations considered. From the trace file it is possible to obtain the number of cache read hits, read misses, write hits and write misses. Table 1 shows the number of cycles used in the trace analysis.

Access	Number of cycles
Cache	Using Table 2
Scratch pad	1 cycle
Main Memory 16 bit	1 cycle + 1 wait state
Main Memory 32 bit	1 cycle + 3 wait states

Table 1: Memory access cycles

From this data we compute the number of accesses to cache based on Table 2, where the number of cycles required for each type of access is listed in Table 1. The cache is a write through cache. There are four cases of cache access that we consider in our model. When the CPU requires some data, the tag array of the cache is accessed. If there is a cache read hit, then data is read from the cache. No write to the cache is done, and main memory is not accessed for a read or write. When there is a cache read miss, it implies that the data is not in the cache, and the line has to be brought from main memory to cache. In this case we have a cache read operation, followed by L words to be written in the cache, where L is the line size. Hence there will be a main memory read event of size L with no main memory write. If there is a cache write hit, we have a cache write, followed by a main memory write. In case of a cache write miss, a cache tag read (to establish the miss) is followed by the main memory write. There is no cache update in this case.

Access type	Ca_{read}	Ca_{write}	Mm_{read}	Mm_{write}
Read hit	1	0	0	0
Read miss	1	L	L	0
Write hit	0	1	0	1
Write miss	1	0	0	1

Table 2: Cache model

Using this model we derive the cache energy equation as

$$E_{cache} = (N_{c-read} + N_{c-write}) * E \tag{12}$$

Where E_{cache} is the energy spent in cache. N_{c-read} is the number of cache read accesses and $N_{c-write}$ is the number of cache write accesses. Energy E is computed like in equation (5), taking the appropriate load and the number of cycles into consideration. In the trace analyzer we model the cache as described above and use it in our energy estimations.

Regarding the scratch pad, the performance estimation is done as follows. The scratch pad is assumed to occupy part of the total memory address space. Based on the address values from the trace analyzer, the access is classified as going to scratch pad or memory and an appropriate latency is added to the overall program delay. One cycle is assumed if it is a scratch pad read access.

If it is a main memory 16 bit access then we take it as one cycle plus 1 wait state. If it is a main memory 32 bit access we consider it as one cycle plus 3 wait states. In case of a scratch pad there will not be any read miss, write miss. This is the reason for improved performance in case of scratch pad when compared to cache on-chip memory configuration.

6 Experimental Results

In this section we first give the experimental setup and then present the experimental results obtained for the cache and scratch pad memory. To demonstrate the merits of using on-chip scratch pad memory and on-chip caches, we conducted a series of experiments for both of these configurations. The trace analysis for the scratch pad and the cache is done in the design flow after the compilation phase. We used the cache power estimator tool and modified it to compute the energy estimates for varying address bit-widths. To compute the energy estimates of scratch pad memory we modified the tool according to our design requirement. The organization of the scratch pad is explained in the previous section. The two energy estimators that we use employ the same memory array unit organization. The feature size of target architecture is 0.5μm technology. We use a 2-way set associative cache configuration for comparison.

6.1 Performance

The benchmark suite we selected was compiled using the encc compiler. In Figs. 9 and 10 we present the graph of CPU cycles vs. the size of the scratch pad and the cache respectively. In both figures, y axes represent CPU cycles in 10s of thousands, whereas x-axes is the size of the on-chip memory. To observe the effect of on-chip memory configurations on CPU cycles we have varied the cache and the scratch pad size from 64 bytes to 8192 bytes. For the benchmark suite that we have chosen, after a certain knee point around 1024 bytes, both for cache and scratch pad, there will not be any performance improvement by increasing their size. This is a reflection of the overall memory requirements of these benchmarks.

Fig. 11 shows the performance variation for bubble sort, lattice and selection sort benchmarks for the two on-chip memory options in the same graph.

The performance of the scratch pad is mostly better than the cache. For example, the number of CPU cycles required for the lattice benchmark for a cache of size 1024 bytes is 169225, whereas for scratch pad size 1024 bytes is 141402 (a 16.4% improvement). Consider the bubble sort benchmark with a maximum of 240000 CPU cycles. The smallest cache size is 512 bytes and the scratch pad size is 128 bytes. Generally the graph shows the positive effect of using on-chip scratch pad memory. There are some points of exception. For example, the benchmark lattice for 180000 CPU cycles has a minimum cache size of 256 bytes and scratch pad size of 512 bytes.

6.2 Area

The area is represented in terms of number of transistors. These are obtained from the cache and scratch pad organization using [9]. Fig. 12 shows the area vs. performance for the cache and the scratch pad for biquad_N_sections,

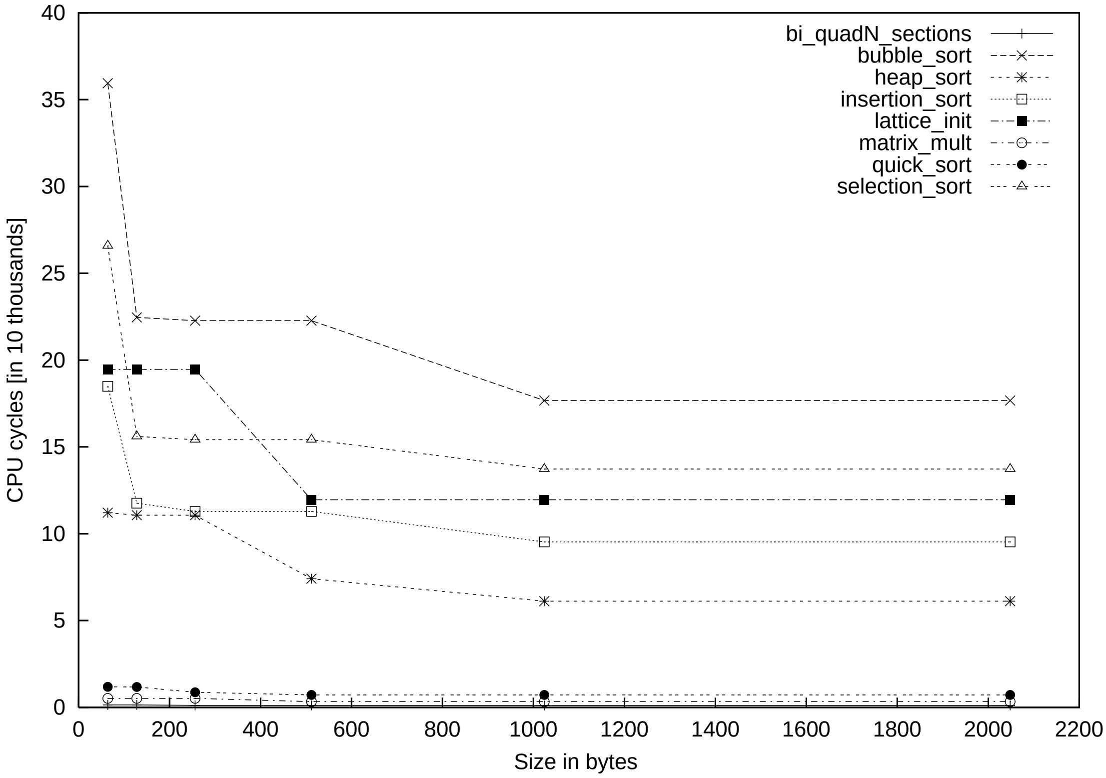


Figure 9: CPU cycles vs scratch pad size in bytes.

quick-sort and matrix mult.

Fig. 13 contains results for heap sort, lattice and selection sort.

Fig. 14 comprises results for the bubble sort and insertion sort benchmarks.

Table 3 gives the area/performance tradeoff. Column 1 is the size of scratch pad and cache in bytes. Columns 2 and 3 are the cache and scratch pad area in transistors. Columns 4 and 5 are the number of CPU cycles for cache- and scratch-pad based memory systems, respectively. As shown in table 2 there is an improvement in the area and performance using scratch pad.

Column 6 is the area improvement due to replacing a cache by an SPM, column 7 the corresponding improvement in the number of cycles and column 8 the improvement of the area-time product AT (assuming constant cycle times):

$$AT = (A_s * Ns)/(A_c * Nc) \tag{13}$$

The average area-, time-, and AT-product improvements are 34%, 18% and 48%, respectively.

6.3 Energy

Firstly, we describe the effect of varying address width on the energy for scratch pad and cache. Next, we give an example of the energy consumption required for a main memory access. Finally, we describe the total energy consumption for the various benchmarks that we have used in the experimental setup.

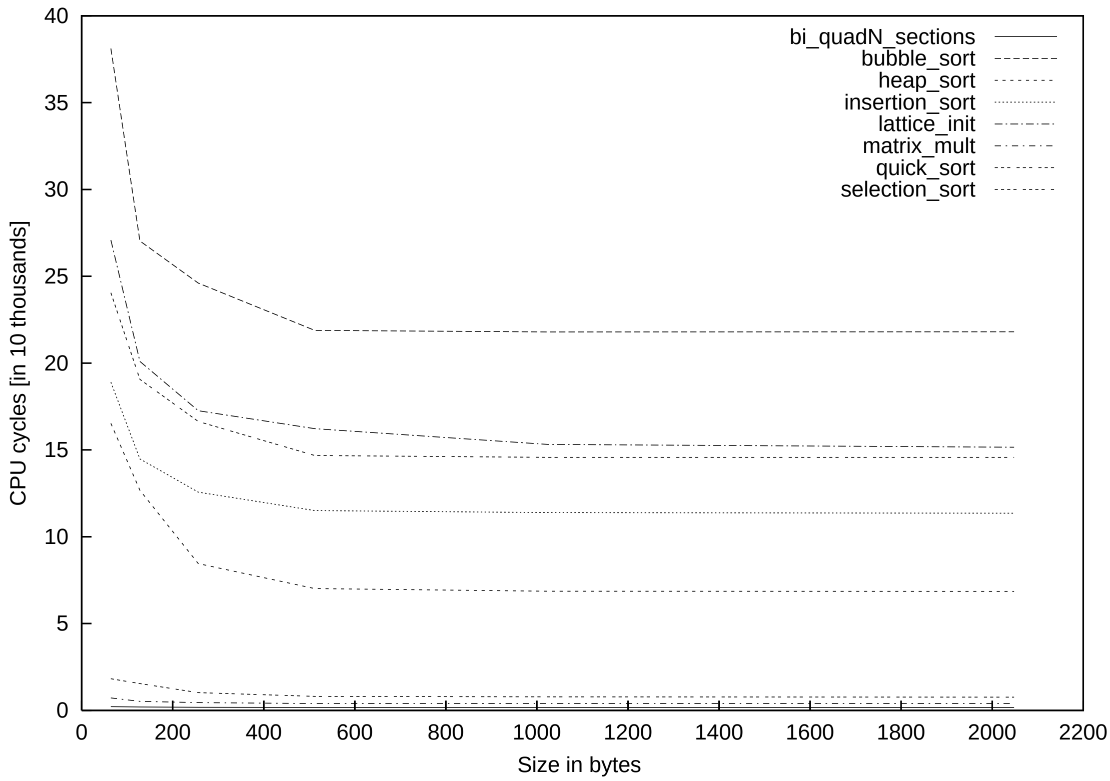


Figure 10: CPU cycles vs cache size in bytes

6.3.1 Address width variation

Fig. 15 shows the graph of cache size, scratch pad size in bytes vs the energy estimates per access obtained from estimation using CACTI. The x axis is the size in bytes and the y axis represents energy in nJ.

Firstly, we observe that by parameterizing the address width (which influences the cachable area) there is a notable change in the energy consumption of the 4 way set associative cache. The reason is that the number of tag-bits depends on the address width, which affects the circuit organization in the tag array side. Secondly the energy consumption of scratch pad memory is less than cache energy consumption, the reduction being in the range of 60% to 82%. We observe a very slight variation in the energy consumption per access for the scratch pad, when the address width is varied.

6.3.2 Main memory

Various technologies can be used for implementing the slowest and largest memory in the memory hierarchy, called “main memory". For PCs, dynamic memory (DRAM) is employed. Due to more predictable and smaller access times, supercomputers and DSP systems typically contain static main memory, implemented with SRAM technology. In general, SRAM memory consumes more energy per access than DRAM.

Our cycle count considerations in section 5 are based on the static RAM chips found on an ATMEL evaluation board. Accordingly, we are using the same chips for energy considerations. The energy required per access is listed in table

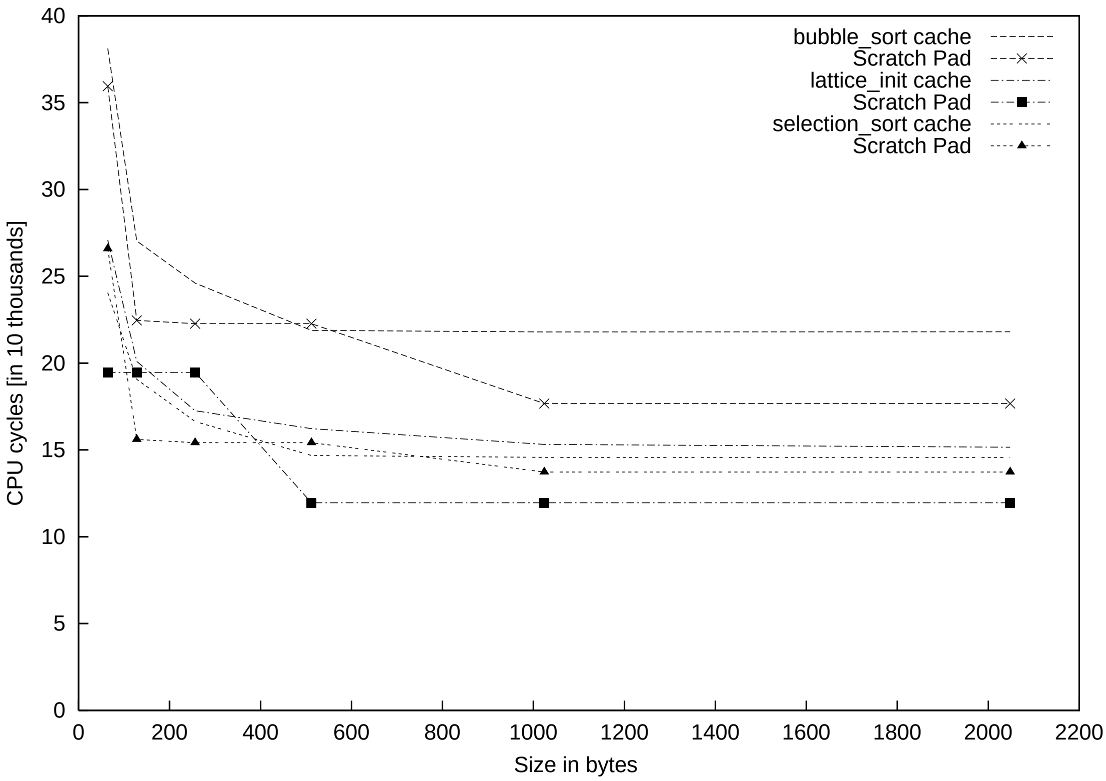


Figure 11: CPU cycles for cache and scratch pad memory

4.

Note that -for the technology which we are considering- there is a factor of up to about 50 between the access to a small scratch pad memory and the main SRAM memory. This is the key factor affecting the energy savings that are possible by using a scratch pad memory.

Future extensions of our current work will also consider the savings that are possible with DRAM-based main memories.

6.3.3 Application of the evaluator

Using our evaluator described in section 2, we compare the energy consumption of several benchmarks.

Figs. 16 to 20 show that most of our benchmarks have a similar behaviour:

- For very small memories (less than 600 bytes), none of the two types of memories is superior in all cases. Sometimes the cache-based architecture consumes less energy, in other cases it is the cache-based architecture. The reason for this is that SPMs have to be large enough to hold entire basic blocks; otherwise, only references to data can be mapped to the SPM. Caches, in contrast are useful even if only fractions of the important basic blocks and loops can be stored in the cache.
- For larger memories, the scratch-pad architecture requires less energy, due to its inherently lower energy consumption per access.

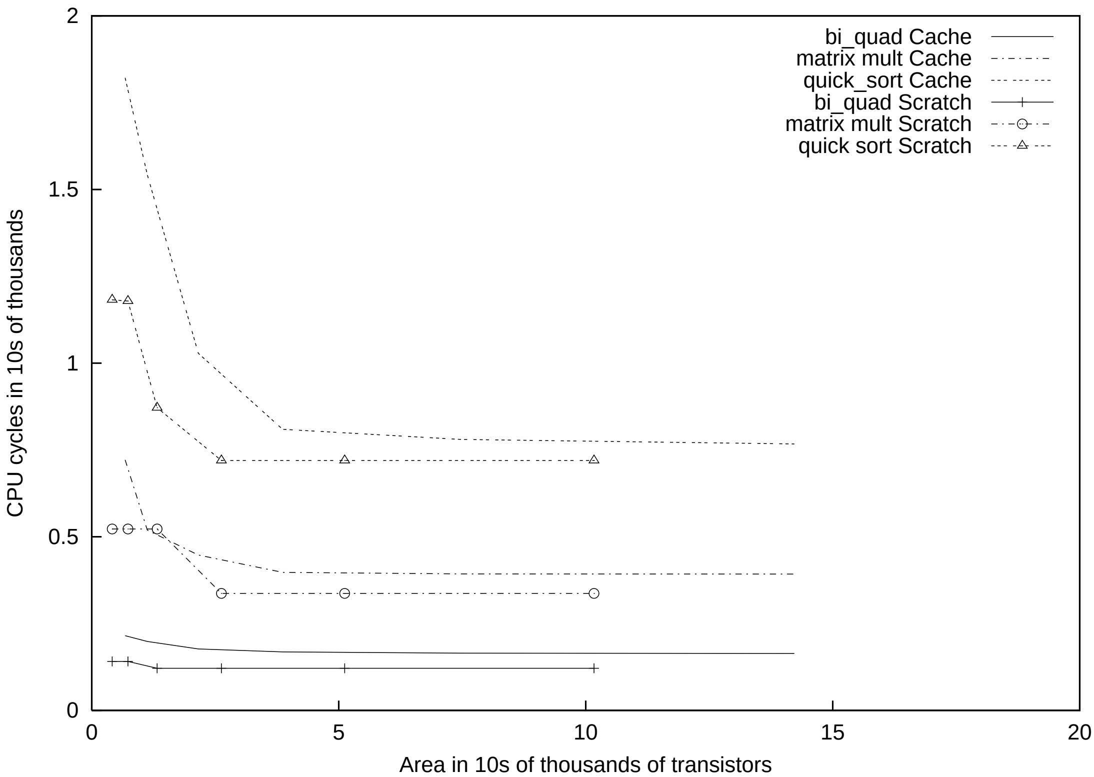


Figure 12: Area vs Performance biquad, matrix mult, quick sort

The cache-based architecture could be superiour for larger memories also if this architecture would have fewer references to the main memory. For our benchmarks, we have observed no such case, see fig 20.

Table 5 shows the energy/performance tradeoff for the bubble-sort benchmark. Column 1 is the size in bytes. Column 2 and 3 are the energy consumption for cache and scratch pad. Column 4 and 5 represent the CPU cycles for cache and scratch pad configurations respectively. Column 6 is the improvement in energy consumption of cache vs scratch pad. Column 7 is the performance improvement of cache-based architecture vs. the SPM-based architecture.
There is only a single case (size: 512 bytes) in which the cache-based architecture outperforms the SPM-based architecture in terms of the energy-consumption. For all the other cases, the latter requires less energy and is faster.

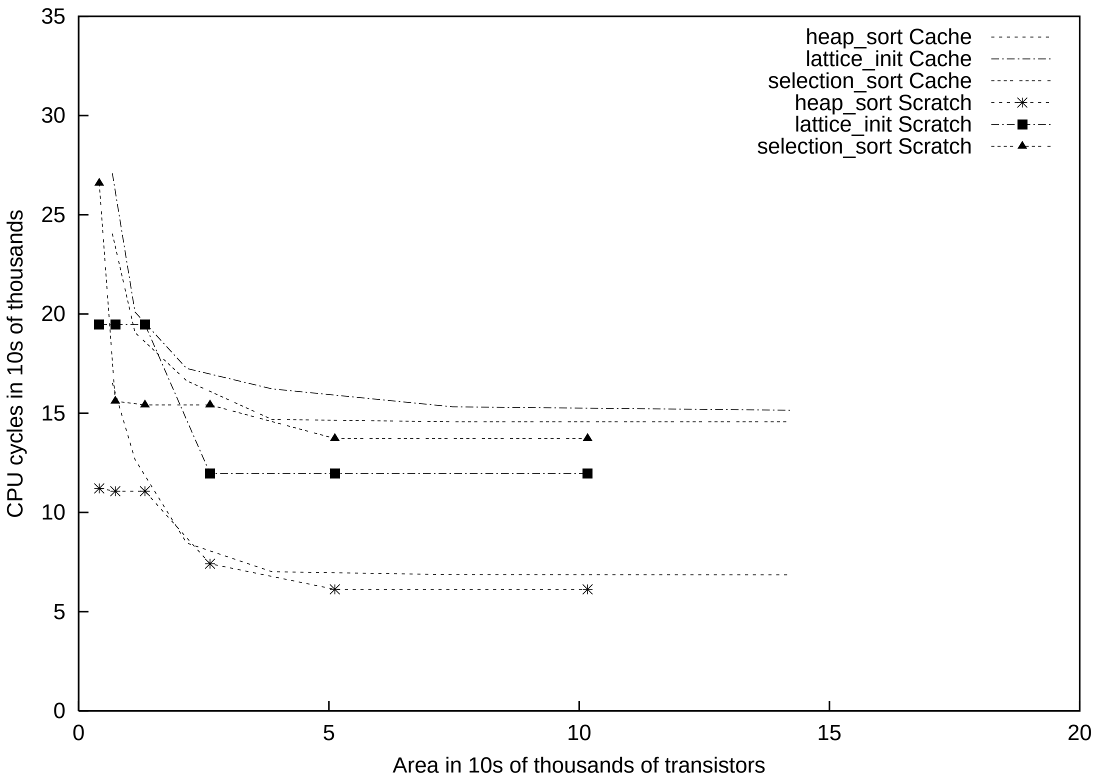


Figure 13: Area vs Performance heap, lattice and selection sort

Size	A_c [transistors]	A_s [transistors]	Nc [k]	Ns [k]	A_s/A_c	Ns/Nc	AT
64	6744	4032	481.9	347.5	0.61	0.72	0.44
128	11238	7104	302.4	239.9	0.65	0.79	0.51
256	21586	14306	264.0	237.9	0.61	0.90	0.55
512	38630	26722	242.6	237.9	0.68	0.90	0.61
1024	74680	53444	241.7	192.0	0.69	0.79	0.55
2048	142224	102852	241.5	192.0	0.71	0.80	0.57
Average ratio					0.66	0.82	0.54

Table 3: Area/Performance ratios for bubble-sort

Read access, 2 bytes	24 nJ
Read access, 4 bytes	49.3 nJ
Write access, 4 bytes	41.1 nJ

Table 4: Energy per access to SRAM chips on our evaluation board

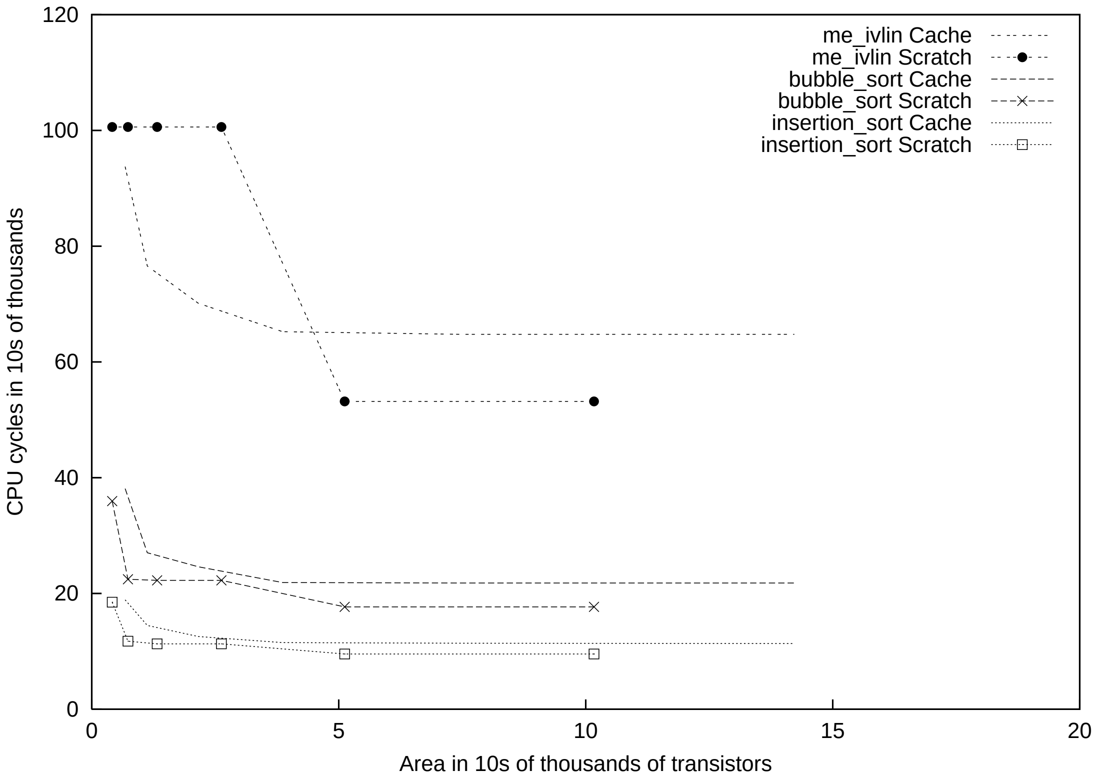


Figure 14: Area vs performance bubble sort and insertion sort

Size	energy, cache-based memory system [nJ]	energy, SPM-based memory system [nJ]	cycles Nc cache-based memory system [k]	cycles Ns SPM-based memory system [k]	Energy improve-ments [%]	Performance improve-ments [%]
64	3,520,380	2,808,619	481.9	347.5	-20.2	-27.9
128	1,431,713	982,567	302.4	239.9	-31.4	-20.7
256	989,092	950,135	264.0	237.9	-3.9	-9.9
512	749,824	962,413	242.6	237.9	28.4	-1.9
1024	773,549	281,949	241.7	192.0	-63.6	-20.6
2048	810,656	324,150	241.4	192.0	-60.0	-20.5

Table 5: Energy/performance results for bubble-sort

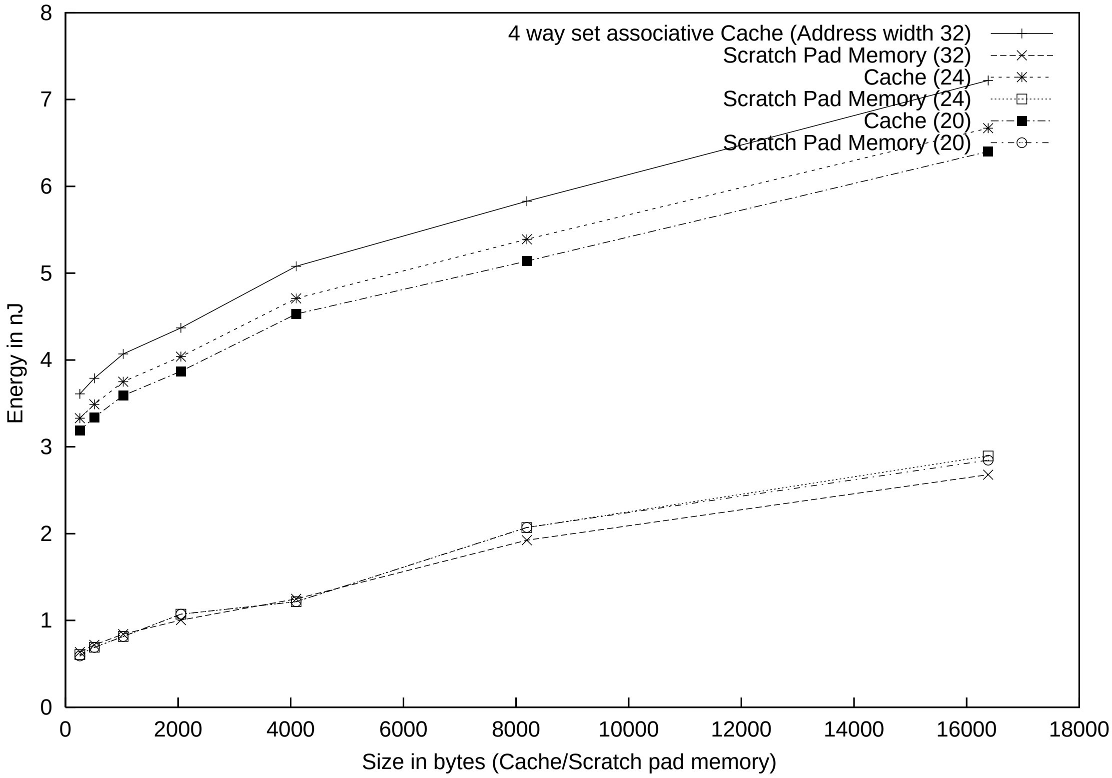


Figure 15: Scratch pad memory and cache energy comparison

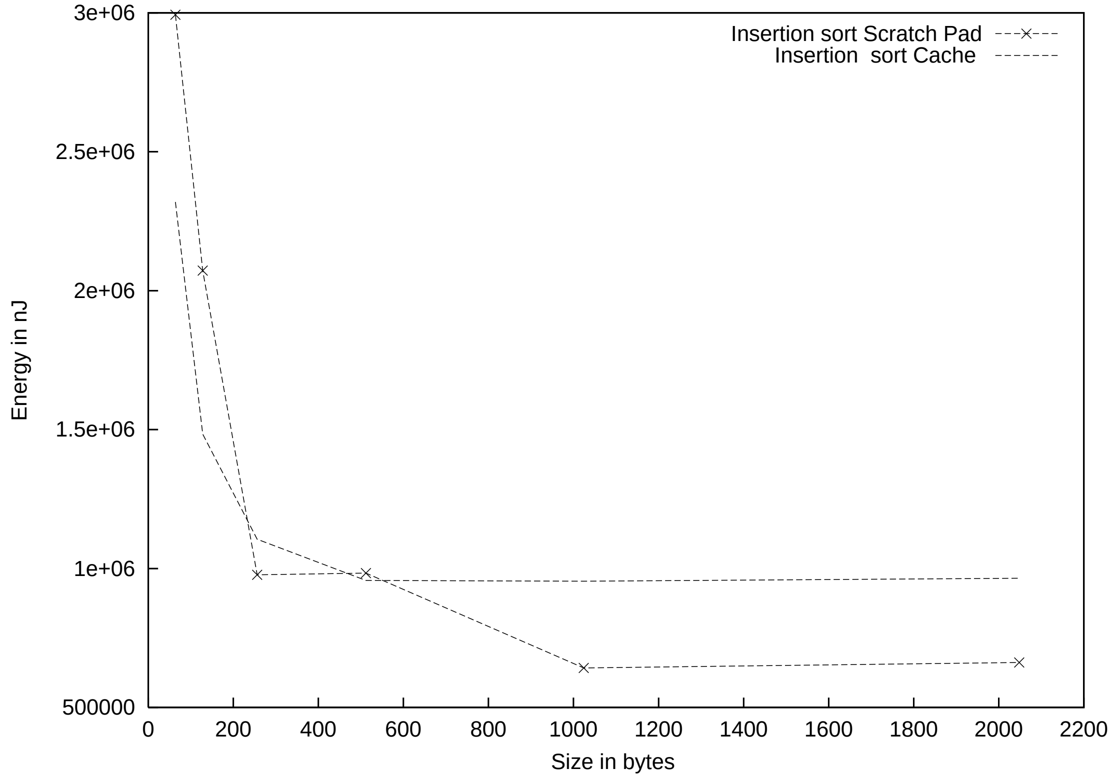


Figure 16: Energy consumed by the memory system for insertion-sort [nJ]

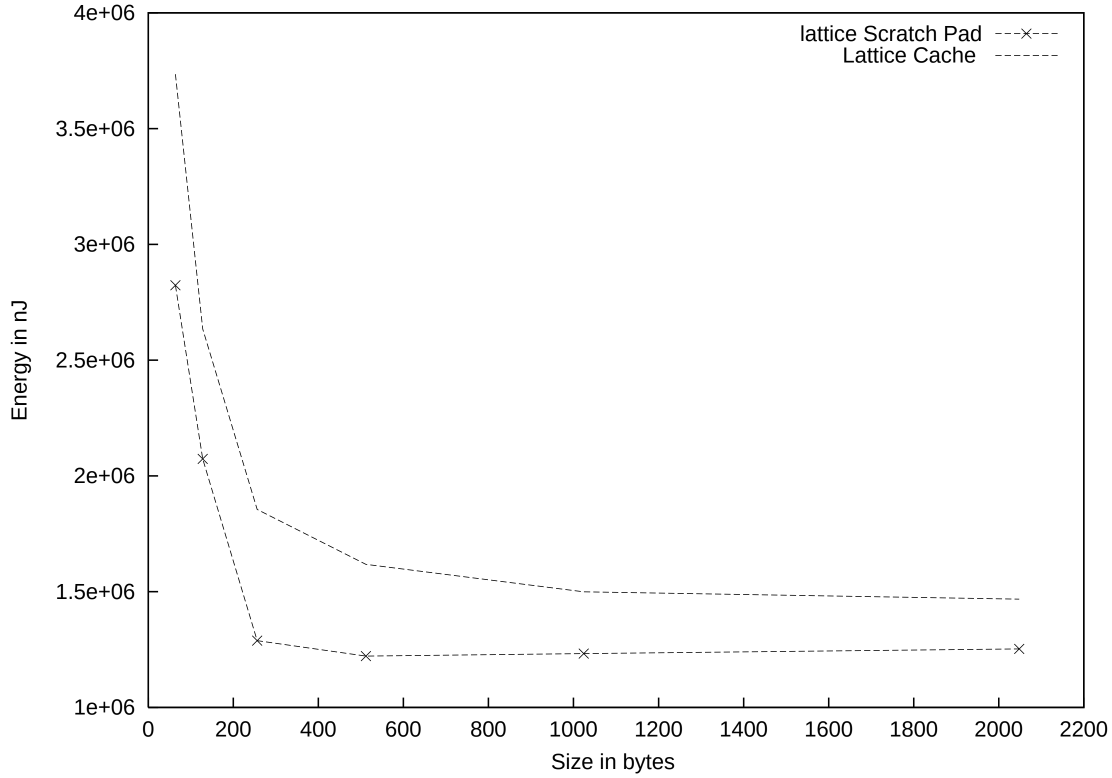


Figure 17: Energy consumed by the memory system for lattice filter

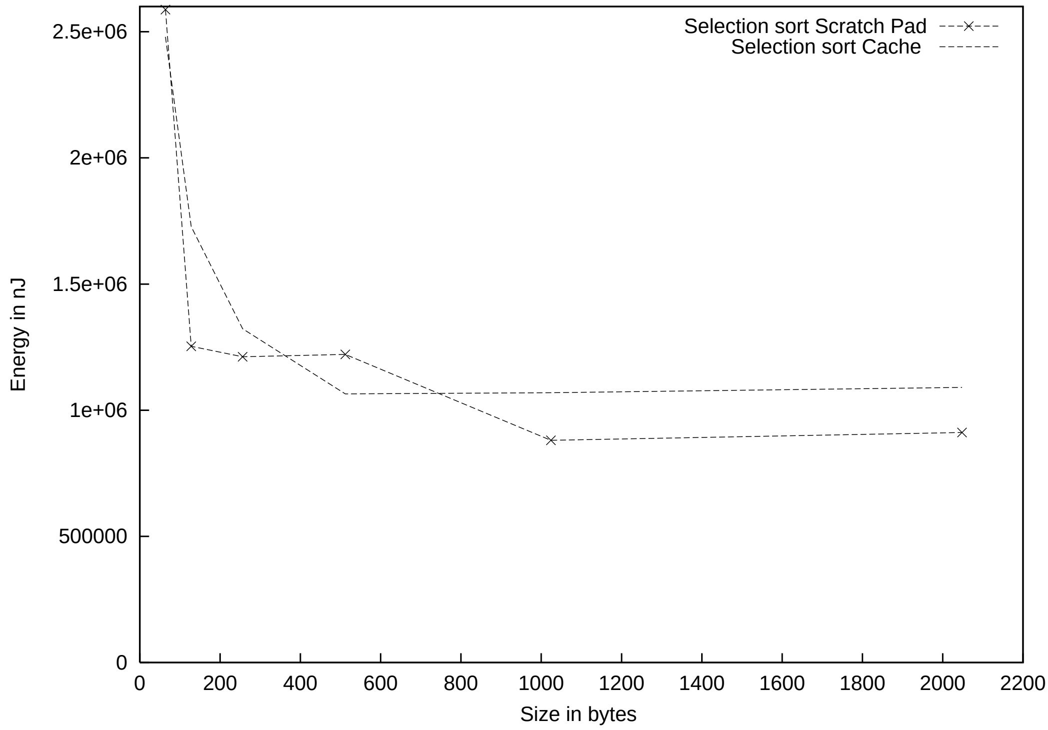


Figure 18: Energy consumed by the memory system for selection sort

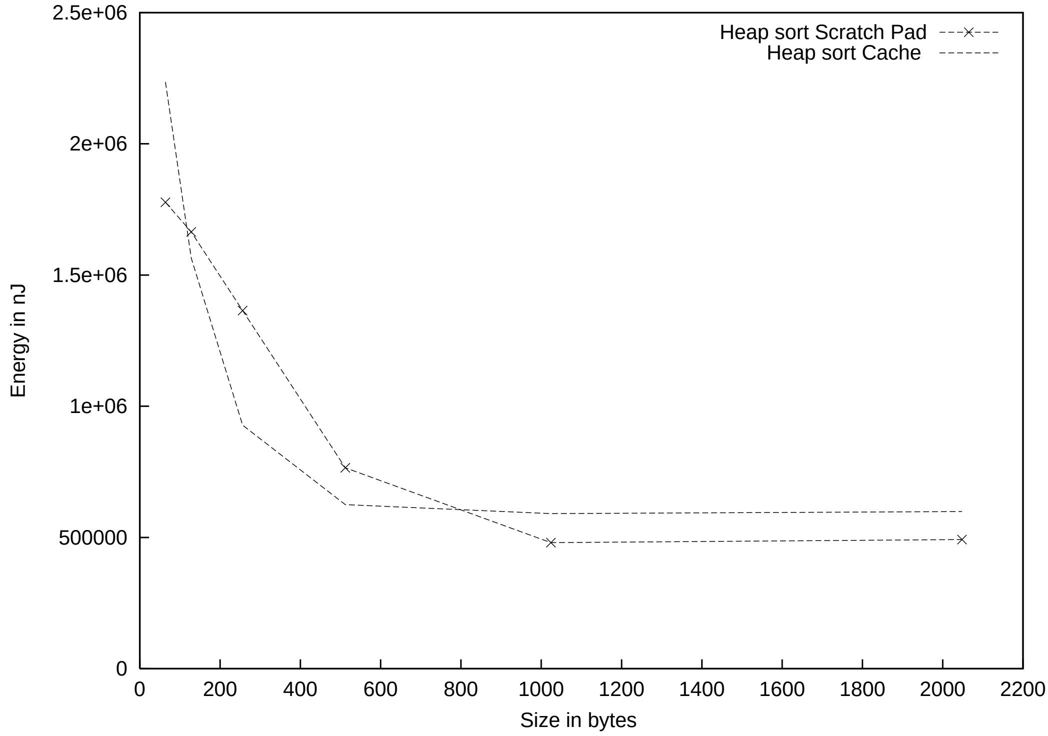


Figure 19: Energy consumed by the memory system for heap-sort

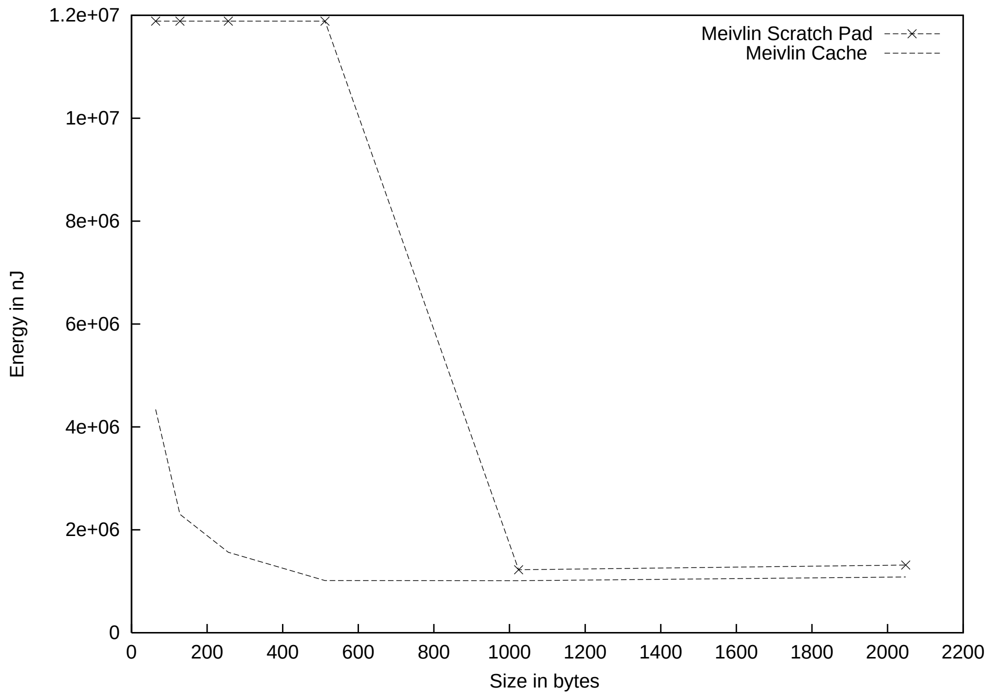


Figure 20: Energy consumed by the memory system for meivlin

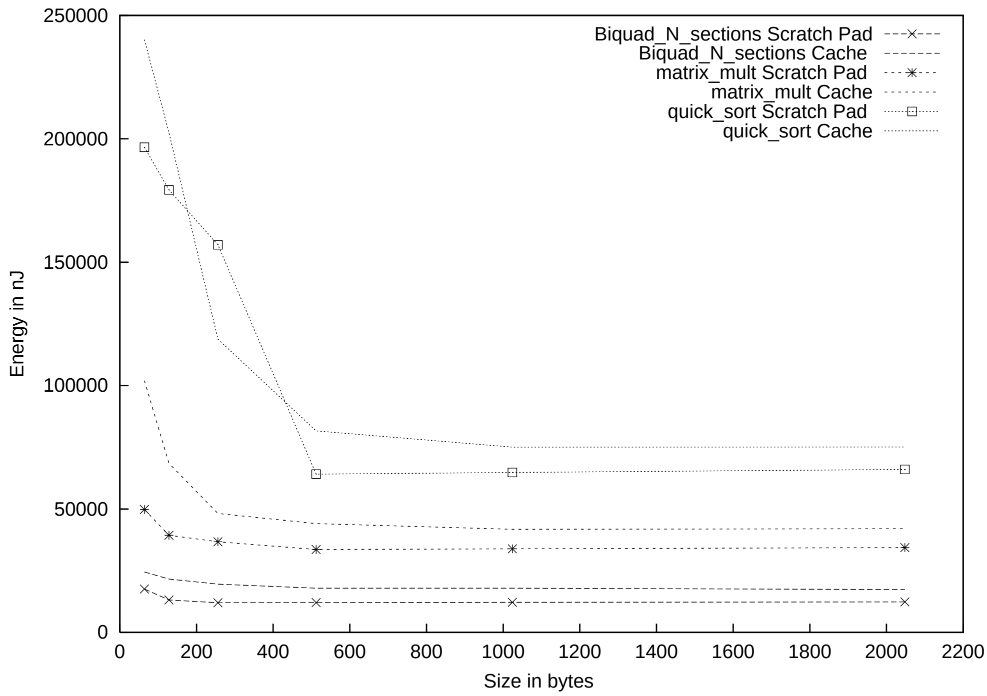


Figure 21: Energy consumed by the memory system

7 Conclusion and future work

In this report we present an approach for selection of memory configurations. The report presents a comprehensive methodology for computing area, energy and performance for various sizes of cache and scratch pad memories. Area and energy models are based on CACTI [9], whereas performance is based on trace analysis using a simulator. The technique implicitly evaluates a knapsack algorithm for identifying and judiciously assigning a given scratch pad memory area to contending code and data blocks. Experiments are performed on a set of benchmarks. The target used is the ARM7TDMI core with the ARMulator for trace analysis.

Results indicate that, for the programs considered, the performance of scratch-pad based compile-time memory allocation does usually exceed that of cache-based run-time memory allocation, except for very small scratch-pad memories. For a given performance, the SPM can be much smaller than the cache.

Similar results are obtained for the area. As long as the SPM is large enough to hold some of the important blocks, significantly less transistors are required in order to meet a certain maximum cycle count constraint.

Combining area and time, we can observe that the area/time product AT can be reduced by 46% on the average by introducing the SPM.

For the energy consumed per access, the SPM is about a factor of 3 more efficient than a 4-way set associative cache. This factor can be exploited unless the hit ratio of the caches is larger than that of the SPM. We found that, for many applications and memory configurations, the total energy consumption of SPM-based memory systems is less than that of cache-based systems. In other cases, the energy consumption is very much comparable. There is only one case in which there is a minor advantage for the cache-based architecture.

These results indicate that the approach proposed in a recent report [12] potentially more efficient than the use of caches and should be further analyzed.

Future work will aim at extending the current work in the following ways:

1. Larger benchmarks have to be considered. For such benchmarks, it may be necessary to consider dynamic reloading of the SPM.

2. An analysis of access times for the SPM and the cache is required. Currently, the performance is compared using the number of CPU cycles. In the future, changes of clock frequencies should also be considered. SPMs will have another advantage there, since no tags have to be compared.

3. For direct mapped caches, the overhead for fetching tags is reduced. Cost/performance tradeoffs for direct mapped caches should also be analyzed.

4. A more detailed analysis of DRAM main memory should be added.

5. Integration of SPM support in the system level has to be analyzed. For example, multiprocess systems require operating system support for SPMs. Fast context switching has to be maintained.

8 Appendix A

CACTI Results

In this appendix we present sample results of the CACTI tool which we have used to estimate the energy consumption of the caches and scratch pad memory.

CACTI tool is used to estimate the energy values for the scratch pad taking into consideration that tag memory and tag comparison is not done. In the following example one computation is shown for a scratch pad size of 512 bytes.
Size in bytes: 512, Vdd: 3.3V
decode data (nJ): 0.130763
word-line and bit-line data word-line power (nJ) : 0.00430446
bit-line power (nJ) : 0.113766
sense amp data (nJ) : 0.763764
valid signal driver (nJ): 0.00422912
data output driver (nJ) : 0.189504
====================================
Sum of these is 1.206 nJ
====================================

Sample result for the cache using CACTI is shown below.
Input : Cache size, Block size, k way associative, Feature size
OUTPUT :
Cache Parameters:
Size in bytes: 512
Number of sets: 32
Associativity: 2
Block Size (bytes): 8
Read/Write Ports: 1
Read Ports: 0
Write Ports: 0
Technology Size: 0.50um
Vdd: 3.3V
Access Time (ns): 2.86322
Cycle Time (wave pipelined) (ns): 0.954407
Power (nJ): 3.9582
Wire scale from mux drivers to data output: 0.10
Best Ndwl (L1): 1
Best Ndbl (L1): 1
Best Nspd (L1): 1
Best Ntwl (L1): 1
Best Ntbl (L1): 1
Best Ntspd (L1): 1
Nor inputs (data): 2
Nor inputs (tag): 2
Time Components:
data side (with Output driver) (ns): 1.82604
tag side (with Output driver) (ns): 2.86322

decode data (ns): 0.404611
decode data (nJ): 0.0702961
word-line and bit-line data (ns): 0.574366
word-line power (nJ): 0.00807173
bit-line power (nJ): 0.211472
sense amp data (ns): 0.3625
sense amp data (nJ): 2.29058
senseext driver (ns): 0.198405
senseext driver (nJ): 0.527851
decode tag (ns): 0.404611
decode tag (nJ): 0.0702961
word-line and bit-line tag (ns): 0.298164
word-line power (nJ): 0.0042859
bit-line power (nJ): 0.0802263
sense amp tag (ns): 0.1625
sense amp tag (nJ): 0.372219
compare (ns): 0.821622
compare (nJ): 0.0269032
mux driver (ns): 0.769588
mux driver (nJ): 0.0712788
sel inverter (ns): 0.120583
sel inverter (nJ): 0.000785287
data output driver (ns): 0.286153
data output driver (nJ): 0.223938
total data path (without output driver) (ns): 1.53988
total tag path is set assoc (ns): 2.57707

References

[1] Steve Furber : *ARM - System-on.chip architecture*, Second edition, Addison-Wesley publications, 2000.

[2] G. L. Nehmhauser, L. A. Wolsey : *Integer and Combinatorial Optimization*, John Wiley and Sons, 1988.

[3] Milind B Kamble and Kanad Ghose : *Analytical energy dissipation models for low power caches*, ISPLED-97, pp 143-148, Aug 1997.

[4] J Kin, M Gupta and WH Mangonie-Smith : *The filter cache: An energy efficient memory structure*, IEEE Micro-30 December 1997.

[5] Preeti Ranjan Panda, Nikhil Dutt, Alexandru Nicolau : *Memory issues in embedded systems on-chip - Optimisations and exploration*, Kluwer Academic Publishers, 1999.

[6] J L Hennessy and D A Patterson : *Computer Architecture - A Quantitative Approach*, Morgan Kaufman, San Francisco, CA, 1994.

[7] Luca Benini, Alberto Macii, Enrico Macii, Massino Poncino : *Synthesis of application specific memories for power optimisation in embedded systems*, DAC 2000 Los Angeles, California, pp 300-303.

[8] Glenn Reinman, Norm Jouppi : *An integrated cache timing and power model*, Report - COMPAQ Western Research Laboratory, Palo Alto,1999.

[9] S Wilton and Norm Jouppi : *Cacti : An enhanced access and cycle time model*, IEEE Journal of Solid State Circuits, May 1996.

[10] AT91M40400 processor : *www.atmel.com*, ATMEL Corporation.

[11] ARM Processors : *www.arm.com*, Advanced RISC Machines Ltd.

[12] Stefan Steinke, Christoph Zobiegala, Lars Wehmeyer, Peter Marwedel : *Moving program objects to scratch-pad memory for energy reduction*, Technical report number 756, Universitat Dortmund, Apr 2001.

[13] Rajeshwari M Banakar, Ranjan Bose, M Balakrishnan : *Low power design - Abstraction levels and RTL design techniques*, Technical Report, IIT Delhi, May 2001

[14] Vasily G Moshnyaga : *Reducing Cache Energy through Dual Voltage Supply*, ASP DAC Jan 2001.

[15] V. Tiwari, Sharad Malik and A. Wolfe : *Instruction Level Power Analysis and Optimization of Software*, Journal of VLSI Signal Processing Systems, August 1996.

[16] V. Zivojnovic, J. Velarde, and C. Schlager : *DSPStone : A DSP-oriented benchmarking methodology*, In Proceedings of the 5th International Conference on Signal Processing Applications and Technology, October 1994.

[17] T Ishihara and H Yasuura : *A power reduction technique with object code merging for application specific embedded processors*, Proceedings of Design Automation and Testing, Europe Conference (DATE 2000), March 2000.

[18] S. Steinke, M. Knauer, L. Wehmeyer, P. Marwedel : *An Accurate and Fine Grain Instruction-Level Energy Model Supporting Software Optimizations*, Proceedings of 11th International Workshop of Power And Timing Modeling, Optimization and Simulation (PATMOS), Sep 2001.

[19] R. Sedgewick: *Algorithms*, Addison Wesley, 1988.