A Tabu Search Heuristic for Scratch-Pad Memory Management

Maha IDRISSI AOUAD, René SCHOTT, and Olivier ZENDRA

Abstract—Reducing energy consumption of embedded systems requires careful memory management. It has been shown that Scratch-Pad Memories (SPMs) are low size, low cost, efficient (i.e. energy saving) data structures directly managed at the software level. In this paper, we focus on heuristic methods for SPMs management. A method is efficient if the number of accesses to SPM is as large as possible and if all available space (i.e. bits) is used. We propose a Tabu Search (TS) approach for memory management which is, to the best of our knowledge, a new original alternative to the best known existing heuristic (BEH). In fact, experimentations performed on our benchmarks show that our Tabu Search method is as efficient as BEH (in terms of energy consumption) but BEH requires a sorting which can be computationally expensive for a large amount of data. TS is easy to implement and since no sorting is necessary, unlike BEH, we save the corresponding sorting time. In addition to that, in a dynamic perspective where the maximum capacity of the SPM is not known in advance, our TS heuristic will perform better than BEH.

Keywords—Energy consumption, memory allocation management, optimization, Tabu Search heuristic.

I. Introduction

EMBEDDED systems are everywhere. Due to technology evolution, these systems must integrate more complex functionalities (video, audio, Internet, videophone, etc.) which needs more and more battery and memory.

Depending on that, memory will become the major energy consumer in an embedded system. Numerous options to economize energy, hence increase autonomy, exist. These various approaches can be classified in two main categories: hardware optimizations and software optimizations. Hardware techniques fall beyond the scope of this paper, but a large amount of literature about them is available (see first parts of [1]). In this paper, we will focus on software optimizations. Some techniques and algorithms, synthesized in [2], try to optimally allocate application code and/or data to one memory kind called Scratch-Pad Memory in order to reduce the energy consumption of embedded systems.

Cache memory is random access memory (RAM) that a computer microprocessor can access more quickly than it can access regular RAM. As the microprocessor processes data, it looks first in the cache memory and if it finds the data there (from a previous reading of data), it does not have to do the more time-consuming reading of data from larger memory [3]. Cache memories, although they help a lot with program speed, do not always fit in embedded systems: they increase the system size and its energy cost (cache area plus managing logic).

Scratch-Pad Memory (SPM), also known as scatchpad RAM or local store in computer terminology, is a high-speed internal memory used for temporary storage of calculations, data, and other work in progress. In reference to a microprocessor, SPM refers to a special high-speed memory circuit used to hold small items of data for rapid retrieval. It can be considered as similar to an L1 cache in that it is the memory next closest to the ALU's after the internal registers, with explicit instructions to move data from and to main memory. Like caches so, SPMs consist of small, fast SRAM, but the main difference between them is that SPMs are directly and explicitly managed at the software level, either by the developer or by the compiler, whereas caches require extra dedicated circuits. SPM's software management makes it more predictable (by avoiding cache miss cases which is an important feature in real-time embedded systems). Compared to cache, SPM thus has several advantages [4]. SPM requires up to 40% less energy and 34% less area than cache [5]. Further, the run-time with an SPM using a simple static knapsack-based [5] allocation algorithm is 18% better as compared to a cache. Contrarily to [5], [6] distinguish between static and dynamic energy. They also show the effectiveness of using an SPM in a memory architecture where a saving about 35% in energy consumption is achieved when compared to a memory architecture without an SPM. [7] use statistical methods and the Independent Reference Model (*IRM*) to prove that SPMs, with an optimal mapping based on access probabilities, will always outperform the direct-mapped cache, irrespective of the layout influencing the cache behavior.

The rest of the paper is organized as follows. Section II describes some existing heuristics for managing memory data allocation. Section III presents our approach based on a Tabu Search heuristic to find the optimized memory data allocation in order to reduce energy consumption. Section IV gives the memory energy consumption model we propose in order to estimate the energy consumed by our different heuristics. Section V shows the experimental results obtained. Finally, Section VI concludes and gives some perspectives.

II. Existing heuristics

In general, authors try to answer the following question: which kind of application data should be allocated to which

M. IDRISSI AOUAD is with INRIA Nancy - Grand Est / LORIA. 615, Rue du Jardin Botanique, 54600 Villers-Lès-Nancy, France (e-mail: Maha.IdrissiAouad@inria.fr web: www.loria.fr/∼idrissma).

R. SCHOTT is with IECN - LORIA, Université Henri Poincaré-Nancy 1. 54506 Vandoeuvre-Lès-Nancy, France (e-mail: www.loria.fr/∼schott web: www.loria.fr/∼schott).

O. ZENDRA is with INRIA Nancy - Grand Est / LORIA. 615, Rue du Jardin Botanique, 54600 Villers-Lès-Nancy, France (e-mail: Olivier.Zendra@inria.fr web: www.loria.fr/∼zendra).

kind of memory? In order to solve this problem, data placement could be guided on one hand by the features of the considered memory (access speed, energy cost, large number of miss access cases, etc.), and on the other hand by the information collected either statistically by analyzing the benchmark's code or dynamically by profiling benchmarks (number of times that data is accessed, data size, access frequency, etc.).
Due to the reduced size of SPMs, one tries to optimally allocate data in it in order to realize energy savings. Thus, most of the authors use one of these three following strategies.

Allocate data into SPM by size: the smaller data are allocated into SPM as there is space available else they are allocated in main memory (DRAM). This method has the advantage of being simple to implement since it only considers the size of the data but has the disadvantage of allocating the largest data in the DRAM. These largest data could be often accessed, which will imply a very few energy economy.

Allocate data into SPM by number of accesses: the most frequently accessed/used data are allocated into SPM as there is space available else they are allocated in DRAM. This strategy is optimal than the previous one, since the most frequently accessed/used data will be allocated in a memory that consumes less energy and therefore will achieve more savings as explained and demonstrated in [8] and [9]. However, we can note granularity problems in some cases such as a structure with only one part is often accessed/used.

Allocate data memory into SPM by number of accesses and size (BEH): this is somehow a combination of the two previous strategies. The idea here is to combine their advantages. If we consider the example of a structure in which only a part is the most frequently accessed/used, we take into account the average number of access to this structure. This avoids granularity problems. Here, data are sorted according to their ratio (access number/size) in descending order. The data with the highest ratio is allocated first into SPM as there is space available. Otherwise it is allocated in DRAM. This heuristic uses a sorting method which can be computationally expensive for a large amount of data. In addition to that, this sorting method will not work very well in a dynamic perspective where the maximum capacity of the SPM is not known in advance. This is, so far, the best known existing heuristic (BEH).

In the rest of this paper, we will refer to the strategy BEH as a basis for our memory energy optimizations.

III. Our approach

The problem we try to solve is a combinatorial optimization problem like the famous knapsack problem [10]. Suppose memory is a big knapsack and data are items. We want to fill this knapsack that can hold a total weight of W with some combination of items from a list of N possible items each with weight w_i and value v_i so that the value of the items packed into the knapsack is maximized. This problem has a single linear constraint, a linear objective function which sums the values of the items in the knapsack, and the added restriction that each item will be in the knapsack or not. If N is the total number of items, then there are 2^N subsets of the item collection. So an exhaustive search for a solution to this problem generally takes exponential running time. Therefore, the obvious brute-force approach is infeasible. Here, we investigate the problem using the Tabu Search method.

```
Generate an initial solution.
loop
     Identify neighbourhood set.
     Identify tabu set.
     Identify aspirant set.
     Choose the best move.
     exit when goal is satisfied or
     the stopping condition is reached.
end loop
```

Fig. 1. A Generic Tabu Search Algorithm.

In this paper, we propose a heuristic based on the Tabu Search approach to solve the problem of optimizing the memory data allocation. This heuristic is an alternative to the best known existing heuristic (BEH) presented in Section II. Tabu Search (TS) is a local search metaheuristic introduced by Glover (1986). Details about Tabu Search can be found in [11]. TS explores the solution space by moving at each iteration from a solution s to the best solution in a subset of its neighborhood $N(s)$. Contrary to classical descent methods, the current solution may deteriorate from one iteration to the next. New, poorer solutions are accepted only to avoid paths already investigated. This insures new regions of a problem solution space will be investigated with the goal of avoiding local minima and ultimately finding the desired solution. To avoid cycling, solutions possessing some attributes of recently explored solutions are temporarily declared tabu or forbidden. The duration that an attribute remains tabu is called its tabu tenure, and it can vary over different intervals of time. The tabu status can be overridden if certain conditions are met; this is called the aspiration criterion and it happens, for example, when a tabu solution is better than any previously seen solution. Finally, various techniques are often employed to diversify or to intensify the search process. A generic Tabu Search algorithm is summarized in Figure 1.

We want to fill the memory (SPM) that can hold a maximum capacity of C with a combination of data from a list of N possible data each with size $size_i$ and access number an_i so that the access number of the data allocated into SPM is maximized. In our implemented TS algorithm, we first start by generating an initial solution randomly. If N is the total number of data, then a solution is just a finite sequence s of N terms such that $s[n]$ is either 0 or the size of the n_{th} data. $s[n] = 0$ if and only if the n_{th} data is not selected in

the solution. This solution must satisfy the constraint of not exceeding the maximum SPM capacity (i.e. $\sum_{i=1}^{N} s[i] \leq C$). We also set a maximum number of iterations and a lifespan on the tabu list. Initially, the optimal solution equals the initial solution, the optimal access number is the access number of the initial solution and the tabu list is empty. As long as the number of iterations is not exceeded, we repeat the following :

- We generate the e^{th} neighborhood of the current solution.
- We compute a new matrix containing the neighboring vectors.
- Based on the solutions contained in this matrix, we calculate a vector of corresponding current size values and a vector of corresponding current access number values.
- We keep best solution from neighborhood.
- We update the tabu list to make a transition back to the old solution impossible for a period.
- Finally, we update if this new access number is better than the existing optimal access number.

IV. Our Memory Energy Estimation Model

In order to compute the energy cost of the system for each configuration, we propose in this section an energy consumption estimation model for our considered memory architecture composed by an SPM, a DRAM and an instruction cache memory. In our model, we distinguish between the two cache write policies: write-through and write-back. In a Write-Through cache (WT), every write to the cache causes a synchronous write to the DRAM. Alternatively, in a Write-Back cache (WB), writes are not immediately mirrored to the main memory. Instead, the cache tracks which of its locations have been written over and marks these locations as dirty. The data in these locations is written back to the DRAM when those data are evicted from the cache [3]. Our proposed energy consumption estimation model is presented below:

$$
\begin{aligned}
E &= E_{tspm} + E_{tic} + E_{tdram} \\
E &= N_{spmr} * E_{spmr} && (1)\\
&+ N_{spmw} * E_{spmw} && (2)\\
&+ \sum_{k=1}^{N_{icr}} [h_{i_k} * E_{icr} + (1 - h_{i_k}) * [E_{dramr} + E_{icw} \\
&\quad +(1 - WP_i) * DB_{i_k} * (E_{icr} + E_{dramw})]] && (3)\\
&+ \sum_{k=1}^{N_{icw}} [WP_i * E_{dramw} + h_{i_k} * E_{icw} + (1 - WP_i) * \\
&\quad (1 - h_{i_k}) * [E_{icw} + DB_{i_k} * (E_{icr} + E_{dramw})]] && (4)\\
&+ N_{dramr} * E_{dramr} && (5)\\
&+ N_{dramw} * E_{dramw} && (6)
\end{aligned}
$$

Lines (1) and (2) represent respectively the total energy consumed during a reading and during a writing from/into SPM. Lines (3) and (4) represent respectively the total energy consumed during a reading and during a writing from/into instruction cache. When, lines (3) and (4) represent respectively the total energy consumed during a reading and during a writing from/into DRAM. The various terms used in our energy consumption estimation model are explained in Table I.

TABLE I
List of terms.

Term	Meaning
E_{tspm}	Total energy consumed in SPM.
E_{tic}	Total energy consumed in instruction cache.
E_{tdram}	Total energy consumed in DRAM.
E_{spmr}	Energy consumed during a reading from SPM.
E_{spmw}	Energy consumed during a writing into SPM.
N_{spmr}	Reading access number to SPM.
N_{spmw}	Writing access number to SPM.
E_{icr}	Energy consumed during a reading from instruction cache.
E_{icw}	Energy consumed during a writing into instruction cache.
N_{icr}	Reading access number to instruction cache.
N_{icw}	Writing access number to instruction cache.
E_{dramr}	Energy consumed during a reading from DRAM.
E_{dramw}	Energy consumed during a writing into DRAM
N_{dramr}	Reading access number to DRAM.
N_{dramw}	Writing access number to DRAM.
WP_i	The considered cache write policy: WT or WB. In case of WT, $WP_i = 1$ else in case of WB then $WP_i = 0$.
DB_{i_k}	Dirty Bit used in case of WB to indicate during the access k if the instruction cache line has been modified before ($DB_i = 1$) or not ($DB_i = 0$).
h_{i_k}	Type of the access k to the instruction cache. In case of cache hit, $h_{i_k} = 1$. In case of cache miss, $h_{i_k} = 0$.

V. Experimental results

For our experiments, we consider a memory architecture composed by a Scratch-Pad Memory, a main memory (DRAM) and an instruction cache memory. We make sure to take similar features for the cache memory and the SPM in order to compare their energy performance fairly. We performed experiments with eleven benchmarks from six different suites: MiBench [12], SNU-RT, Mälardalen, Mediabenchs, Spec 2000 and Wcet Benchs. Table II gives a description of these benchmarks.

In order to compute the energy cost of the system for each configuration, we used our developed energy consumption estimation model presented in Section IV. This model is based on the OTAWA framework [13] to collect information about number of accesses and on the energy consumption estimation tool CACTI [14] in order to collect information about energy per access to each kind of memory. OTAWA (Open Tool for Adaptative WCET Analysis) is a framework of C++ classes dedicated to static analyses of programs in machine code and to the computation of Worst Case Execution Time (WCET). OTAWA is freely available (under the LGPL licence) and is designed to support different architectures like PowerPC, ARM or M68HC. In our case, we focus on PowerPC architectures. In our model, we distinguish between the two cache write policies: Write-Through (WT) and Write-Back (WB) as explained before. Our presented Tabu Search algorithm and the BEH strategy have been implemented with the C language on a PC Intel Core 2 Duo, with a 2.66 Ghz processor and 3 Gbytes of memory running

TABLE II
LIST OF BENCHMARKS.

Benchmarks	Suite	Description
Sha	MiBench	The secure hash algorithm that produces a 160-bit message digest for a given input.
Bitcount	MiBench	Tests the bit manipulation abilities of a processor by counting the number of bits in an array of integers.
Fir	SNU-RT	Finite impulse response filter (signal processing algorithms) over a 700 items long sample.
Jfdctint	SNU-RT	Discrete-cosine transformation on 8x8 pixel block.
Adpcm	Mälardalen	Adaptive pulse code modulation algorithm.
Cnt	Mälardalen	Counts non-negative numbers in a matrix.
Compress	Mälardalen	Data compression using lzw.
Djpeg	Mediabenchs	JPEG decoding.
Gzip	Spec 2000	Compression.
Nsichneu	Wcet Benchs	Simulate an extended Petri net. Automatically generated code with more than 250 if-statements.
Statemate	Wcet Benchs	Automatically generated code.

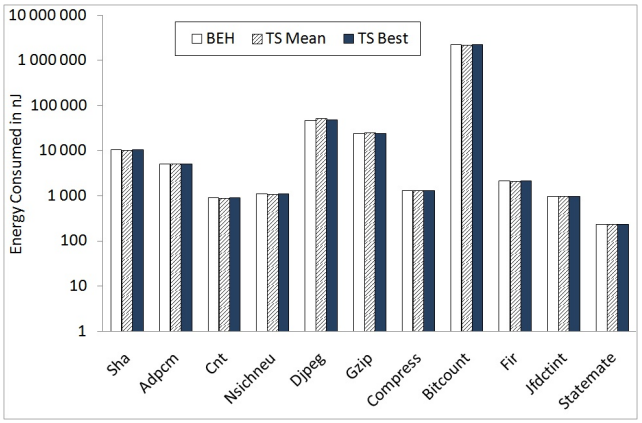

Fig. 2. Energy consumed in standard benchmarks with WB mode.

under Mandriva Linux 2008.

In our experiments, we generate 30 different executions for the Tabu Search heuristic as the solution given differs from an execution to another. *TS Mean* refers to the average results obtained on these 30 executions. In contrast, *TS Best* refers to the best solution obtained from the thirty executions performed. For BEH, the solution founded does not change from a running to another one.

Figure 2, presents the results obtained when comparing BEH and TS methods on the standard ANR benchmarks assuming the write-back cache policy. In the following, as the shape of curves obtained when comparing BEH and TS methods on our benchmarks assuming the Write-Through cache policy (WT) or the Write-Back cache policy (WB) are slightly the same, we just plot the results obtained with the write-back cache policy. Knowing that $E_{WTmode} \neq E_{WBmode}$.

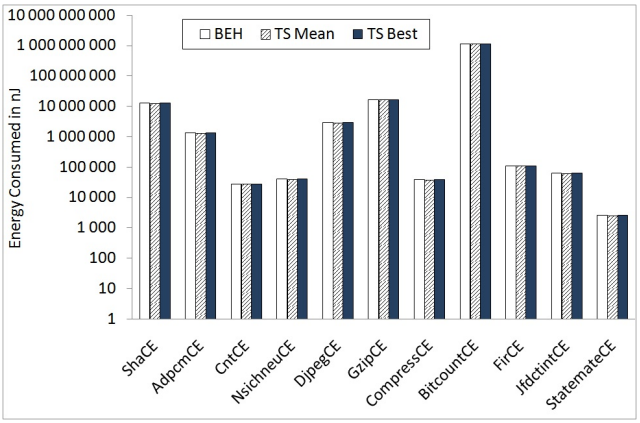

Fig. 3. Energy consumed in modified benchmarks with WB mode.

As we can see from this figure, *TS Best* achieves the same performance as BEH on energy savings on the standard ANR benchmarks. In addition to that, *TS Mean* produces nearly the same energy gains. We are not able to save more energy as we know that BEH already gives the optimal solution thanks to our developed backtracking algorithm. This is true for the standard ANR benchmarks we use as they contain uniform data leading to a big number of local minima. Thus, in order to put some trouble in the BEH strategy and see if it still gives the best solution, we decided to modify slightly our benchmarks. Concretely, our modification consists in adding only one variable to each benchmark. This variable performs an output and is big enough to provide relevant energy savings if it is selected for a Scratch-Pad Memory allocation. We referred to a modified benchmark as *benchmarkCE*.

Figure 3 presents the energy consumed in our modified benchmarks assuming the write-back cache policy. Reminding that $E_{WTmode} \neq E_{WBmode}$.

As we can notice from this figure, although we used the modified benchmarks, we still obtain the same energy savings as before. The BEH strategy didn't give the optimal solution anymore as one could expect as proven by our developed backtracking algorithm. This is normal due to the fact that BEH is a sort of access number/size of data as we explain in Section II. The variable we add in each benchmark has a given access number/size (this ratio depends on the data profiling of each benchmark) so that this variable is not a priority in the sorting made by the BEH method. This is done on purpose so that when it will be the turn of this variable to be treated by BEH, the remaining space in the SPM will not be enough to take this variable and hence it will be allocated in main memory. Whereas, an optimal solution would be to start by allocating this variable first into the SPM. We can see that our Tabu Search heuristic gives the same results as BEH on the modified ANR benchmarks. As our problem is a combinatorial optimization problem (NP-hard problem), an exhaustive search for a solution generally takes exponential running time. Therefore, the obvious brute-force approach is infeasible. That's why we are planning to investigate the

problem using evolutionary algorithms and, more specifically, genetic methods.

VI. Concluding remarks and further research aspects

In this paper, we have proposed a general energy consumption estimation model able to be adapted to different memory architecture configurations. We also have proposed a Tabu Search heuristic (TS) for memory allocation management which is, a new original alternative to the best known existing method (BEH). We show that our TS heuristic is as efficient as BEH in terms of energy consumption. TS is easy to implement and since no sorting is necessary, unlike BEH, we save the corresponding sorting time. In addition to that, in a dynamic perspective where the maximum capacity of the SPM is not known in advance, our TS heuristic will perform better than BEH. In future work, we will explore evolutionary heuristics (Genetic Algorithms, Markov Decision Processes, Simulated Annealing, ANT method, Particle Swarm technique, etc.) and hybrid heuristics for solving the problem of reducing memory energy consumption.

Acknowledgment

This work is financed by the french national reseach agency (ANR) in the Future Architectures program.

References

[1] R. Graybill and R. Melhem, *Power aware computing*, R. Graybill and R. Melhem, Eds. Norwell, MA, USA: Kluwer Academic Publishers, 2002.

[2] L. Benini and G. D. Micheli, "System-level power optimization: Techniques and tools." in *ISLPED-99:ACM/IEEE*, 1999, pp. 288–293.

[3] A. Tanenbaum, *Architecture de l'ordinateur 5e édition*, P. Education, Ed., November 2005.

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

[5] R. Banakar, S. Steinke, B. Lee, M. Balakrishnan, and P. Marwedel, "Scratchpad memory: design alternative for cache on-chip memory in embedded systems," in *CODES*. New York, NY, USA: ACM Press, 2002, pp. 73–78.

[6] H. Ben Fradj, A. E. Ouardighi, C. Belleudy, and M. Auguin, "Energy aware memory architecture configuration," vol. 33, no. 3, pp. 3–9, 2005.

[7] J. Absar and F. Catthoor, "Analysis of scratch-pad and data-cache performance using statistical methods," in *ASP-DAC*, 2006, pp. 820–825.

[8] J. Sjödin, B. Fröderberg, and T. Lindgren, "Allocation of global data objects in on-chip ram," in *Workshop on Compiler and Architectural Support for Embedded Computer Systems*. ACM, December 1998.

[9] S. Steinke, L. Wehmeyer, B. Lee, and P. Marwedel, "Assigning program and data objects to scratchpad for energy reduction." in *DATE*. IEEE Computer Society, 2002, p. 409.

[10] U. P. H. Kellerer and D. Pisinger, *Knapsack Problems*. Springer, Berlin, Germany, 2004.

[11] M. Gendreau, *An introduction to tabu search*, e. F. Glover, G. Kochenberger, Ed. Boston, MA: Kluwer Academic Publishers, 2003, vol. 57.

[12] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mibench: A free, commercially representative embedded benchmark suite," in *WWC '01: Proceedings of the Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop*. Washington, DC, USA: IEEE Computer Society, 2001, pp. 3–14.

[13] H. Cassé and C. Rochange, "OTAWA, Open Tool for Adaptative WCET Analysis," in *Design, Automation and Test in Europe (Poster session "University Booth") (DATE), Nice, 17/04/07-19/04/07*. http://www.date-conference.com/: DATE, avril 2007, p. (electronic medium), poster session. [Online]. Available: ftp://ftp.irit.fr/IRIT/TRACES/7722_date2007.pdf

[14] S. Wilton and N. Jouppi, "Cacti: An enhanced cache access and cycle time model." *IEEE Journal of Solid-State Circuits*, 1996.