Architectural Leakage-Aware Management of Partitioned Scratchpad Memories

Olga Golubeva Mirko Loghi Massimo Poncino Enrico Macii

Dipartimento di Automatica e Informatica
Politecnico di Torino
{olga.golubeva, mirko.loghi, massimo.poncino, enrico.macii}@polito.it

ABSTRACT

Partitioning a memory into multiple blocks that can be independently accessed is a widely used technique to reduce its dynamic power. For embedded systems, its benefits can be even pushed further by properly matching the partition to the memory access patterns. When leakage energy comes into play, however, idle memory blocks must be put into a proper low-leakage sleep state to actually save energy when not accessed. In this case, the matching becomes an instance of power management problem, because moving to and from this sleep state requires additional energy.

In this work, we propose an explorative solution to the problem of leakage-aware partitioning of a memory into disjoint sub-blocks. In particular, we target scratchpad memories, which are commonly used in some embedded systems as a replacement of caches.

We show that the total energy (dynamic and static) cost function yields a non-convex partitioning space, making smart exploration the only viable option; we propose an effective randomized search in the solution space which has very good match with the results of exhaustive exploration, when this is feasible.

Experiments on a different sets of embedded applications has shown that total energy savings larger than 60% on average can be obtained, with a marginal overhead in execution time, thanks to an effective implementation of the low-leakage sleep state.

1. INTRODUCTION

As a major contributor to a system's overall power budget, memories have always been one of the main targets of power optimization techniques. This interest has generated a wide range of solutions, which historically have focused on the reduction of *dynamic* power [1, 2]. With the scaling of technology to feature sizes below 100nm, however, *static* power due to leakage currents has become increasingly important. While leakage is a problem for any transistor, it is even more critical for memories: their high density of integration translates into a high power density that increases temperature, which in turn affects leakage current exponentially. For this reasons, several leakage-aware memories structures, in particular for caches, have been devised in the recent past ([11]–[16]).

The central idea behind most of these techniques is to put infrequently or unused portions of a memory (e.g., cache lines) into a low-leakage state to reduce power. Since the transition from and to the low-leakage state has some penalty, these techniques become a variant of a power management problem, yielding a tradeoff between power and performance. This "selective shutdown" paradigm has been mostly applied to caches because they are the most critical element in the power-performance tradeoff for processor-based systems; its rationale, however, do apply to other types of memories, such as scratch-pad memories (SPM).

SPM are widely used in embedded systems, in which the flexibility of caches in terms of workload adaptability is often unneeded, and power consumption and cost play a much more critical role. In SPMs, it is thus the designer that decides the mapping of addresses to locations of the scratchpad.

The selective shutdown paradigm for SPMs has been addressed in [9] and [10], in which only dynamic power was considered. Neglecting static power simplifies the problems in several ways. First, the sleep mechanism is automatically achieved, at no penalty, by not accessing blocks. Second, dynamic energy is an average quantity that does allow to abstract away the temporal dimension. As a consequence, the partitioning algorithms can search along "space" dimension only (the memory addresses), thus reducing the size of the search space [10].

In this work, we improve previous SPM partitioning approaches by including static power in the cost function. This complicates the partitioning problem since it removes the two above mentioned simplifications. Our contribution is twofold; first, we characterize the search space of the SPM partitioning problem in a static power regime; second, we propose a SPM partitioning algorithm based on an implicit enumeration of the partitioning solutions.

Our approach has two characteristic features that differentiate it from existing memory leakage optimizations solutions. First, it is purely architectural; no special memory *internal* design is required (as in most existing approaches – [11]-[16]), and standard SRAM arrays can be used. Moreover, the extra hardware used to implement the scheme implements a very simple decoding function that allows it to be synthesized automatically, lending itself to being used in a standard design flow. Second, our approach is trace-based, i.e., a given scratchpad partition is computed based on an execution trace (as in [9],[10]). The advantage of this approach is that our technique can be used even in cases where application binaries can not be modified. This also implies complete transparency to the embedded SW developer, who will use a completely standard programming tool-chain. Clearly, the mapping is application specific and, as such, will be different for each different application.

Results show that it is possible to save up to 89% of the energy (about 60% on average).

2. BACKGROUND AND MOTIVATION

Partitioning a scratchpad memory into multiple blocks is a commonly used technique to reduce its average dynamic power. The idea relies on the fact that, for a memory block, (1) memory accesses are not uniformly distributed, (2) energy is consumed only when accessing it, and (3) its cost is proportional to the size of the block times the total number of access. Using these properties, it is intuitive to split the address space (a single memory block) into multiple, independently accessed memory blocks in such a way that

978-3-9810801-2-4/DATE07 © 2007 EDAA

most of the accesses will occur into the smaller blocks and only few ones into the larger ones. Figure 1 ([4]) pictorially summarizes this principle for a two-block partition. The SPM (left) consists of W

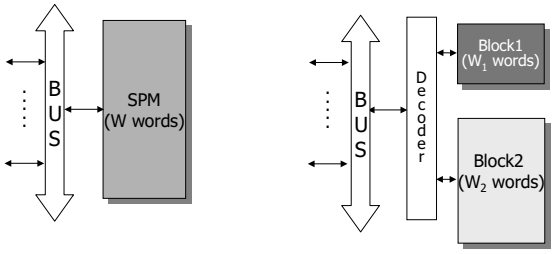

Figure 1: Common-Case Optimization Applied to Memory.

words, which are split (right) into two different memory blocks of sizes N_1 and N_2, with $N_1 < N_2$, and $N_1 + N_2 = N$ (we assume a partition into disjoint set of addresses). Tones of grey denote frequency of access. The common case, that is, the most frequently accessed addresses are placed into the smaller (darker) memory.

Assuming a simplified power model in which memory access cost is proportional to the number of words, i.e., $C_{mem} = n$, and that N is the total number of memory accesses, N_1 (N_2) of which will fall into block *Mem*$_1$ (*Mem*$_2$). The average power consumption in the original case is $P_{mem,1} = \frac{N \cdot W}{N} = W$; in the second case, the total power consumption is $P_{mem,2} = \frac{N_1 \cdot W_1 + N_2 \cdot W_2}{N}$. The second scheme is more convenient as N_1 gets larger with respect to N_2.

As technology scales, however, some of the above properties do not hold anymore. In fact, the importance of leakage power increases, and energy is consumed even when a memory is not accessed (the idle state). To reduce the energy consumed in the idle state, proper schemes to put a memory block into a *sleep* state with negligible energy consumption are required. These schemes, however, normally imply a timing overhead: transitioning into and especially out of the sleep state consumes energy and time, and putting a memory block into a sleep state should be done only if this cost can be amortized.

Introducing of the time dimension makes the problem much more complex than the case of Figure 1. As a matter of fact, for dynamic energy we are interested only in the total number of accesses and not of their distribution over *time*. Conversely, deciding about putting a memory block into sleep requires extraction of the *idleness* of a memory block.

The example of Figure 2 shows how the relative importance of leakage power affects the quality of a partitioning architecture. The plot refers to a trace of memory accesses issued by an embedded application, and shows two curves. The solid one reports the energy saving provided by three-bank partitioned architecture changes as a function of the ratio γ of leakage and dynamic energy ($\gamma = 0$ = dynamic energy only, $\gamma \to \infty$ = static energy only). The curve is obtained by using the solution achieved by considering dynamic energy only ($\gamma = 0$), and accounting for leakage energy as well when computing the savings. In other words, all points on this curve refer to the same partition.

We can notice how the efficiency of the partitioning degrades for increasing values of γ, ending up in an increase of the total energy for when leakage dominates. The dashed curve reports instead the saving that can be achieved when a leakage-aware partitioning (such as the one presented in this work) is used.

3. PREVIOUS WORK

The problem of the energy-efficient partitioning an on-chip memory in multiple banks have been studied by several authors, as well

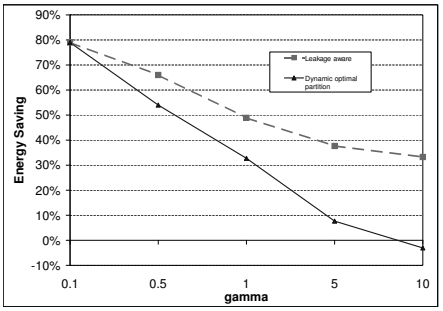

Figure 2: Importance of Leakage-Aware Partitioning.

as at different levels of abstractions. In these section we will only review hardware techniques that do not modify the access patterns. Traditional techniques ([6, 7, 8]) are of explorative nature: all possible partitions are analyzed and matched against the access patterns of the application and the best solution is recorded. More sophisticated schemes exploit the properties of the memory access energy cost and the resulting structure of the partitioning space ([9, 10, 11]) to come up with polynomial-time algorithms that are able to derive the optimal partition for a given access pattern. All these techniques only consider dynamic power, and the issue of the energy consumption of a non-accessed memory is not considered.

Leakage-aware partitioning of memory structures is addressed at the circuit-level by many authors, in particular for cache-like structures. A well-known approach is the *cache-decay* architecture[12], which turns off the cache lines during their "dead time" in which they are not used before being evicted. Another popular architecture is the *drowsy cache*[13], in which cache lines are put into state-preserving low-power modes (drowsy modes) based on usage statistics that can be collected at run time and implemented in hardware. Dynamic run-time resizing of a cache (based again on usage statistics) through deactivation of a subset of lines, columns or even entire ways is another viable approach [14, 16]. These techniques, however, require the modification of the internal structure of caches, which are normally very highly-optimized designs.

The only approach that deals with sub-banking at a higher level of abstraction is the work of Kandemir et al. [15]: exploiting bank locality (i.e., consecutive memory accesses use the same memory bank as much as possible) it is possible to maximize the idleness thus ensuring maximal amortization of the re-activation overhead.

Our work differs from [15] in three main aspects. First, our method is purely architectural in the sense that it is completely transparent to the processor and can be applied to any trace. Second, it is trace-based and does not require any "analytical" description of the application in terms of loop indices as in [15]. Third, it concurrently accounts for dynamic and leakage power in the computation of the best solution.

4. MEMORY ENERGY CHARACTERIZATION

4.1 Memory Energy Model

Key to our method is the availability of a low-leakage sleep state for a given memory block. Two aspects must be considered in defining such a sleep state. The first one concerns whether the sleep state should preserve memory content or not. In principle, both options are viable. In practice, since we target embedded systems, for which the energy overhead of refilling of the entire SPM would consume excessive power, we resort to a state-preserving mechanism.

The second issue is related to the actual implementation of the sleep state. Our constraint of not changing the internal memory structure (cells and/or architecture) implies that modifications must be "external" to the memory. In order to set a standard memory block in a low energy state, two solutions are possible. The first one is to increase the threshold voltage (V_{th}) of the transistors by varying their substrate voltage (V_{bs}). This option requires access to bulk contacts of the components, which is not always possible (as in the case of the memories used in this work). The second way, which is always feasible, is to vary the supply voltage (V_{dd}) of the SRAMs. This also reduces leakage because of the drain induced barrier lowering (DIBL) effect, and, moreover, because it affects the drain-source voltage.[1] Due to its general applicability, we implement a sleep state as a low-V_{dd} state.

To give a quantitative idea of how leakage can be saved by modulating V_{dd}, Figure 3 shows the dependency of SRAM leakage power on memory size and on power supply. Data are obtained from experiments on a 65nm technology from STMicroelectronics, as discussed later. The projections on the two planes (fixed V_{dd} and fixed size) show a linear and exponential trend, respectively.

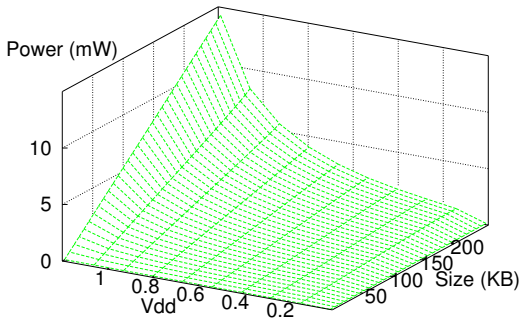

Figure 3: Leakage Power as Function of Size and V_{dd}.

The state preserving constraint is achieved by imposing that V_{dd} be larger than V_{th}. Notice however that memory cannot be reliably read or written in this low-V_{dd} operating condition. Therefore, reading and writing will require going back to the normal, active state. Memory will thus evolve between a *Sleep* and an *Active* state; The *Sleep* state is characterized by a low voltage supply (V_{dd_L}), and, hence, a low leakage energy consumption. Transitions between the two states have an energy and a time cost that, in general, will depend (i) on the size W of the memory block considered, and (ii) on the voltage level V_{dd_L} of the *Sleep* state (Figure 4).

Such an overhead must be taken into account when computing an effective partition, because the cost of the transition to *Sleep* must be compensated by the benefit provided by the exploited idleness.

Energy is actually spent only during the *Sleep-to-Active* transition, which in fact causes the loading of internal capacitances from a V_{ddL} to V_{dd}. The opposite transition, on the contrary, just discharges the corresponding capacitances, and does not draw current from the power supply. Thus, $E_{AS} = 0$.

Timing overheads can be derived based on the same reasoning: the only transition we are interested in is the one from *Sleep* to *Active*, since the other one can be overlapped with accesses to other memory blocks (because a block switched to *Sleep* will not be accessed in the immediate future).

[1] V_{th} can be expressed as:

$$V_{th} = V_{th0} - \mathrm{DIBL}(V_{ds})$$

where $\mathrm{DIBL}(V_{ds})$ is proportional to V_{ds}.

In the sub-threshold region, the drain current is:

$$I_{ds} = I_0 \cdot (1 - e^{-\frac{V_{ds}}{v_t}}) \cdot e^{-\frac{V_{gs} - V_{th}}{n v_t}}$$

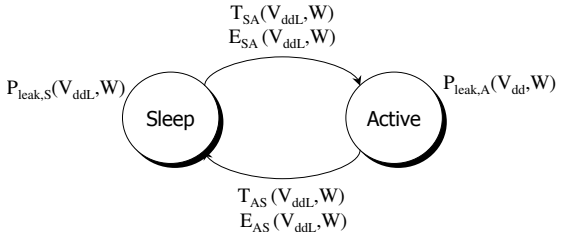

Figure 4: Memory Power States and Transition Costs.

Since upon a regular access a memory is charged from 0 to V_{dd} in a fraction of its cycle time, and since we assume that a memory block can be accessed in a single cycle, we can safely estimate the time overhead as one cycle. This consideration is in accordance with simulated data on caches reported in [13], in which restoring V_{dd} from a 300mV drowsy state was reported to take less than one cycle.

This point is interesting because it differentiate this instance of power management from traditional ones, for which the timing overhead is normally sizable. The most important consequence is that we can derive a *breakeven* value, defined as the minimum time interval for which a memory must remain in the *Sleep* state to overcome the transition cost

Based on the above considerations, total energy consumption of a memory block can be defined as:

$$E = N_{acc} \cdot E_d + (T - T_S) \cdot P_{leak_A} + T_S \cdot P_{leak_S} + N_{sw} \cdot E_{AS} \quad (1)$$

where N_{acc} is the number of accesses to the block, E_d is the dynamic energy spent for each access, T is the total execution time, T_S is the sum of the cycles in which the memory is kept in sleep state, P_{leak_A} and P_{leak_S} are the static power spent in *Active* and in *Sleep* state respectively, and N_{sw} is the number of times the block switches from *Sleep* to *Active* state.

Based on this energy model, the breakeven value B can be obtained by imposing $E(T_S = B, N_{sw} = 1) = E(T_S = 0, N_{sw} = 0)$, thus resulting in the formula:

$$B = \frac{E_{SA}}{P_{leak_A} - P_{leak_S}}$$

which depends on V_{ddL} and W. Typical values are relatively small and are in the order of the hundred of cycles.

4.2 Characterization

The characterization of the quantities described in the previous section were carried out on a family of 32-bits memories, developed by STMicroelectronics for a 65nm technology. Foundry data-sheets provide information about the behavior (static and dynamic energy consumption, and access/cycle times) for the normal functioning (*Active* state).

Static power consumption in the *Sleep* state P_{leak_S} was derived by scaling the data related to the *Active* state (provided by the data-sheet), based on the architecture of the SRAM and using the MOS analytical formula; more precisely, we first calculate the ratio between the leakage of a transistor that operates at V_{dd} and the leakage of the same transistor that operates at V_{dd_L}. Then, by inspecting the SRAM architecture, we can determine which transistors are leaking. This allows to compute a scale factor to apply to the foundry data.

Energy transition overheads have been estimated by evaluating the capacitance seen from the supply node. A rough quantitative evaluation of the *Sleep* to *Active* time can be done by using the energy

cost of a regular memory access (which charges and the discharges the bitlines) to first estimate the sum of bitline capacitances. From this value, we can calculate the energy needed to charge the capacitance from V_{dd_L} to V_{dd}. To increase the accuracy of this procedure, these data were integrated by accurate simulation data from some SPICE simulations of small SRAM arrays, properly tuned to match the energy figures reported in foundry data-sheets.
In this work, we chose 0.5 V for V_{dd_L}, (V_{dd} is 1.2 V). Since V_{th} is $0.42V$, this values is large enough to guarantee preservation of memory state, yet with a reduction of leakage of a factor 10.
Energy data are related to a temperature of $50\,^{\circ}$C. Such a temperature, for the used technology, implies $\gamma = 0.5$, therefore the average dynamic power is two times the average power due to leakage. Since in our exploration framework what matters is the dependency of these quantities on memory size, Figure 5 shows the dependency on memory size of P_{leak_A} and P_{leak_S} (left), and of E_d and E_{SA} (right).

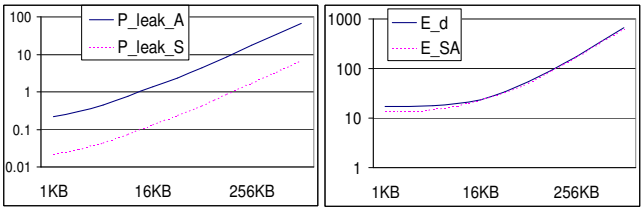

Figure 5: P_{leak_A}, P_{leak_S} (μW) **(left),** E_d, E_{SA} (pJ) **(right).**

The evaluation time transition cost is much simpler because, as already mentioned, we can conservatively assume an overhead of one cycle for the *Sleep-to-Active* transition. The actual overhead is in fact smaller than a cycle; an entire cycle is in fact needed to restore the active state from a 0V state, while in our case we have to restore from a larger voltage level (V_{ddL}). Estimating of this overhead would require an evaluation of the internal capacitance of the memory array similar to what has been done for energy overhead estimation. However, the analysis is in this case more difficult, because not all the internal nodes are charged simultaneously, and internal propagation delays must be taken into account.

4.3 Partitioning Overhead

Partitioning a monolithic SPM into disjoint sub-blocks implies an overhead due to an additional decoder (to convert global addresses into sub-block addresses), and to the wiring to connect the decoder to the sub-blocks [9]. As the number of sub-blocks increases, the complexity of the decoder stay almost constant, but the wiring overhead increases, thus preventing arbitrarily fine grain partitioning of the SPM. Following the approach described in [9], we characterized the partitioning overhead by adapting published data to the 65nm technology used in this work (e.g., doubling the relative importance of wires with respect to that of cells on energy consumption). In spite of that, the overhead stays relatively small. The overhead partitioning for a 2-block partition is only 4% of the monolithic SPM, about 6% for a 3-block partition, and a 8.5% for a 4-block partition.
Results in [9] have shown that the overhead for larger number of blocks is in most cases not amortized.

5. ENERGY-EFFICIENT SCRATCHPAD PARTITIONING

5.1 Problem Formulation

We assume that the memory accesses of the application running on the system is described by a trace $T = \{a_1, ..., a_L\}$, where a_i denotes the generic address accessed at cycle i. L is the length of the trace, i.e., the number of execution cycles of the application. Without loss of generality, we assume that addresses are all 32-bit wide, and are aligned to 4-bytes boundary.
We consider a memory consisting of M words, to be partitioned into a set of N non-overlapping blocks. A partition is defined by the $N-1$ boundaries (addresses) of the partition $\Pi = [p_1, p_2, \ldots, p_{N-1}]$, where p_i is the address boundary between the i-th and the i+1-th memory block.
The problem we are solving can be formulated as follows: *Given a trace T of length L, and the maximum number N of blocks into which to split the memory, find a partition $\Pi = [p_1, p_2, \ldots, p_{N-1}]$ for which total energy consumption:*

$$\mathcal{E}(p_1, p_2, \ldots, p_{N-1}) = \sum_{i=1}^{N} E_i + OV(N) \qquad (2)$$

is minimized.
In the above formula, E_i is the energy spent by the i-th block (computed with formula 1), while $\mathrm{OV}(N)$ is the partitioning overhead that depends only on N.

5.2 Searching the Solution Space

Scanning the trace, we can obtain information about which memory cell is accessed at which time, hence we can build, in principle, a bidimensional matrix MAP with addresses on the horizontal dimension and times on the vertical one. Each cell $\mathrm{MAP}[a, t]$ is thus identified by the address a and the time t, and it contains a 1 if the address a was been accessed at time t, and 0 otherwise. Such a matrix, however, is unpractical because of its huge size. Moreover, storing data for each distinct cycle and address is useless, because data locality makes sense only on larger time and address intervals. Furthermore, exploring a possible partitioning with a step of one word is unrealistic, because memory cuts cannot have arbitrary small dimension.
For these reasons, we discretized both addresses and times, denoting with ΔS and ΔT the space and time granularity, respectively. As a result of discretization, a cell $\mathrm{MAP}[a, t]$ of the matrix contains 1 if at least an access to some address between a and $a + \Delta S$ happened in the time interval $[t, t + \Delta T]$; otherwise it contains a 0. The matrix has now size $\mathcal{L} \times \mathcal{M}$, where $\mathcal{L} = \frac{L}{\Delta T}$ and $\mathcal{M} = \frac{M}{\Delta S}$. Once the map has been generated, we can use it to evaluate formula 2 for an arbitrary partition $\Pi = \{p_1, p_2, \ldots, p_{N-1}\}$. In this work we adopted a very conservative choice for ΔS (64 bytes) and ΔT (1000 cycles). However, as we will show later, searches can in most cases be sped up by using much larger values of ΔS, without compromising the effectiveness of the search.
The computation of an optimal partition can be carried out by searching the solution space, identified by all possible N-block partition. An exhaustive exploration of all the partitions has complexity $O(\mathcal{M}^{N-1})$[2]. Given the large values of M, an exhaustive search is prohibitive. Search-based algorithms that deal with dynamic energy only ([9, 10]) completely abstract away the time dimension, transforming the problem into a uni-dimensional search. Furthermore, the dynamic energy cost function allows to avoid useless computation. A more formal analysis ([10]) shows that the search space exhibits the properties of the optimality of sub-problems as well as the fact that the optimal solution is a set of sub-optimal solutions, allowing to fit it to a dynamic programming paradigm.
Considering (i) the time dimension and (ii) the leakage energy cost function makes the problem totally different, and, more problematic, the search space very much less regular.

[2]The actual size of the solution space is $\left(\begin{array}{c} \mathcal{M} \\ N-1 \end{array} \right)$

Figure 6 shows the total energy as a function of the bi-partition boundary, for an example trace; we can notice how the cost function is quite irregular, with several local minima (notice, also, that the rising portion, on the left side of the plot, is actually composed of many little oscillations, thus of many local minima that could trap a local search). The shape of the energy cost is even more irregular when the partition cardinality N increases, thus preventing alternatives to exhaustive search. In general the cost function is not convex, thus a search algorithm cannot leverage such a simplification (albeit the function cost is application dependent, thus it is not impossible, only very unlikely that some realization turns out to be convex). Searching a global optimum in a non-convex space

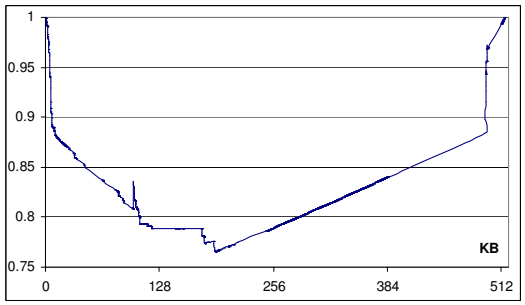

Figure 6: Normalized energy consumption as function of the partition boundary (KB) in a bi-partition.

implies resorting to approximate solutions, which do not guarantee optimality in general, but can yield a provably good solution in reasonable time. Many methods do exist to solve the problem of searching into a very large (non-convex) space, such as simulated annealing, tabu search, genetic algorithms, particle swarm optimization. In this work, we have used a variant of a *random-restart gradient descent* approach [21], which essentially consists of a running an outer loop over a conventional gradient descent search; each step of the outer loop chooses a random initial condition to start the descent.

Our implementation starts by picking a set of random partitions, among which the best one is selected (the *candidate solution*).

Then, we generate and evaluate a new random partition. If this partition does not improve the candidate solution, we discard it. Otherwise, we perform a local search around it: partition boundaries are perturbed by a vectorial quantity $\{\Delta p_1, \Delta p_2, \ldots, \Delta p_{N-1}\}$ in direction of the opposite of the energy gradient $-\nabla\mathcal{E}(p_1, \ldots, p_{N-1})$. In order to avoid computing all the possible incremental ratios around the current position, we estimate $\nabla\mathcal{E}$ with a *Montecarlo* search. In this way we quickly reach a local minimum that outperforms and, therefore, replaces the candidate solution.

When the candidate solution survives R (namely, 1000) random jumps, the algorithm terminates and returns the best solution found. This search algorithm has no optimality claim, but it aims at achieving a solution with good quality in acceptable time. As results will show, the approximate solution is very close, in most cases, to the optimal partition (when the latter can be computed).

6. EXPERIMENTAL RESULTS

6.1 Experimental Setup

To assess the effectiveness of our algorithm, we used two sets of traces. The first one was generated by running the Powerstone benchmarks [17] on top of Platune [18] (a MIPS simulator). Applications are fed with small inputs, in order to keep their memory footprint (and thus M) small enough to allow exhaustive explorations.

The second set of traces are taken from MIBENCH [19] application suit on the ARM version of Simplescalar [20]. Since we used large input data for these simulations, the resulting traces and the required memory are quite large, thus exhaustive explorations cannot be performed within practical times.

6.2 Experimental Data

Figure 7 shows the results of the exploration performed on Powerstone benchmarks. For each trace and γ value, we plot three energy savings: (i) the one provided by a partitioning scheme that considers dynamic energy only, (ii) the one achieved with the optimum partition (found with the exhaustive search), and (iii) the saving obtained with our algorithm. All values are computed against the energy spent by a monolithic SPM.

We can notice that, for almost all the traces, our search heuristic provides almost identical results to exhaustive search, and, in the few cases where it does not, its sub-optimality is very limited. Notice that, in some cases, the savings provided by a partitioning scheme driven only by dynamic energy increases for larger values of γ; this is because these partitioning schemes tend to cluster accesses in small memory areas, leaving large memory portions which are barely used. Because spatial correlation is often associated to temporal correlation of accesses, such memory blocks can very often be turned into *Sleep* state, also saving leakage.

However, the actual effectiveness of a partitioning depends on the distribution over time of the idleness, that is not taken into account by dynamic-energy-based strategies (e.g., if a block remains idle one time for 1000 cycles, the saving is higher than if it is idle 10 times for 100 cycles each). Therefore, their saving potential is always smaller than our method, and decreases as γ increases.

	$\gamma = 0.5$	$\gamma = 0.75$	$\gamma = 1$	$\gamma = 2$	$\gamma = 5$
CRC32	64.8	60.4	57.2	49.8	42.5
adpcm.dec	89.1	84.1	87.5	85.0	84.3
adpcm.enc	86.8	84.3	87.9	86.4	83.4
search	58.1	51.7	48.7	41.1	33.8
sha	82.9	86.6	85.8	83.9	82.1
rijndael_o	59.8	55.5	52.3	44.4	36.9
rijndael_i	58.7	54.1	51.1	43.2	35.7
gsmd	73.1	69.6	68.0	62.1	57.1
gsme	73.5	69.0	66.3	61.9	57.0
tiff2bw	77.1	74.1	73.3	69.1	65.1
dijkstra	69.7	66.4	62.4	58.4	52.8
djpeg	70.1	65.8	64.1	58.2	52.7
fft_1	60.3	52.9	50.2	42.8	36.2
fft_2	59.2	53.4	49.1	42.8	31.9
say	68.1	63.7	60.7	56.3	50.1
mad	64.4	60.9	55.3	51.4	42.6
cjpeg	64.2	59.4	55.2	48.0	39.3
ispell	58.3	50.3	44.3	34.3	24.6

Table 1: Savings for MIBENCH Trace (4-Block Partition).

Table 1 shows the energy savings, with regard to the unpartitioned case, that result for the MIBENCH applications. For each benchmark we reported results concerning four different values of γ. We can see as our algorithm provides savings between 24% and 89% (60% on average). The decreasing savings when γ increases is due to the characteristics of the applications: when $\gamma \to 0$ the resulting saving is S_d (the one related to dynamic energy only), while when $\gamma \to \infty$ the saving is S_L (saving that accounts only leakage energy). The relation between this two quantities depends on the application memory usage: if the least used memory areas are almost uniformly accessed (as it happens for applications we examined),

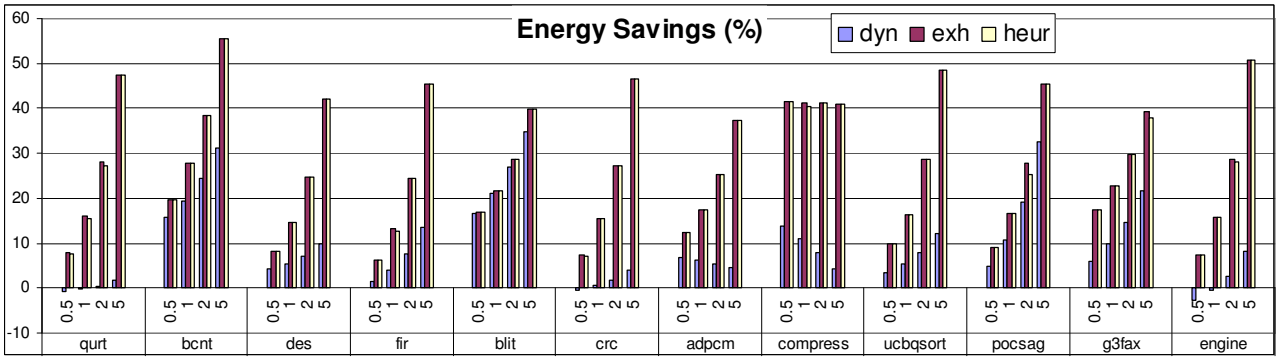

Figure 7: Energy savings for Powerstone benchmarks and a 4-blocks partition.

than there is fewer potential for leakage saving, because they must be frequently turned on. Therefore, for MIBENCH applications, when leakage impact increases, the energy saving decreases.

6.3 Sensitivity Analysis

We also analyzed the impact of the discretization step (ΔS) on the results of our algorithm. In general, increasing ΔS reduces the solution space, thus decreasing the search time. Intuitively, we expect a trade-off between the algorithm speed and its efficiency, depending on the application.

Figure 8 shows the behavior of the energy savings when ΔS varies, for some of the MIBENCH benchmarks. We reported the energy savings obtained for increasing steps normalized to the saving obtained with a very small ΔS. Notice that step values are normalized with respect to the actual memory usage of the application (M).

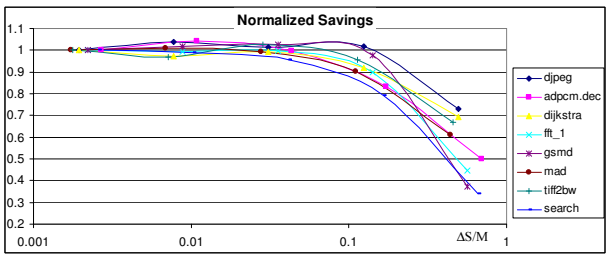

Figure 8: Normalized energy savings when varying ΔS.

Interestingly, for values of ΔS below 5% of the application address space, the savings are not impacted (there are a few oscillations, due to the randomness of the algorithm). Moreover, even increasing ΔS up to 10% of M still yields quite good energy savings. In quantitative terms, since the typical values of M of these benchmarks is between 256KB and 1MB, it means that we can achieve good results with exploration steps in the range 2–16KB.

Notice that for large step values, the 4-partition solution reduces first to a 3-partition (when $\Delta S \sim \frac{M}{3}$), and then to a bi-partition (when ΔS approaches $\frac{M}{2}$), thus further reducing the effectiveness of the algorithm.

7. CONCLUSIONS

In this work we have used the selective shutdown paradigm to reduce the energy consumption of scratchpad memories in a scenario where leakage energy takes a relevant fraction of total energy.

We propose a search-based partitioning algorithm that accounts for both spatial and temporal correlation, and leverages (i) a careful characterization of the solution space, (ii) an accurate state-based model of memory energy. Results show an average energy saving of about 60% (up to 89% for the most profitable cases), while just for 8% of cases the saving is below 40%.

8. REFERENCES

[1] P. Panda, N. Dutt, *Memory Issues in Embedded Systems-on-Chip Optimization and Exploration*, Kluwer, 1999.

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

[3] "International Technology Roadmap for Semiconductors 2002 Edition," Semiconductor Industry Association, http://public.itrs.net.

[4] E. Macii, R. Mehra, M. Poncino, "Micro-Architectural Power Estimation and Optimization," in *Handbook of EDA for IC Design*, G. Martin, L. Lavagno, and L. Scheffer Editors, CRC gg, Boca Raton, Florida, 2006.

[5] A. Bogliolo, L. Benini, E. Lattanzi, G. De Micheli, "Specification and Analysis of Power-Managed Systems", *Proceedings of the IEEE*, Vol. 92, No. 8, pp. 1308-1346, Aug. 2004.

[6] C. Su, A. Despain, "Cache Design Tradeoffs for Power and Performance Optimization: A Case Study," *ISLPD-95*, pp. 63-68, April 1995.

[7] W. Shiue, C. Chakrabarti, "Memory Exploration for Low-Power Embedded Systems," *DAC-35*, pp. 140-145, June 1998.

[8] S. Coumeri, D. E. Thomas, "Memory Modeling for System Synthesis," *IEEE Transactions on VLSI Systems*, Vol. 8, No. 3, pp. 327-334, June 2000.

[9] L. Benini, L. Macchiarulo, A. Macii, E. Macii, M. Poncino, "Layout-Driven Memory Synthesis for Embedded Systems-on-Chip," *IEEE Transactions on VLSI Systems*, Vol. 10, No. 2, pp. 96-105, April 2002.

[10] F. Angiolini, L. Benini, A. Caprara, "An Efficient Profile-Based Algorithm for Scratchpad Memory Partitioning", *IEEE Transactions on Computer-Aided Design*, Nov 2005, Vol. 24, No. 11, pp. 1660-1676.

[11] O. Ozturk, M. Kandemir, "Nonuniform Banking for Reducing Memory Energy Consumption," *DATE'05: Design, Automation and Test in Europe*, Mar. 2005, pp. 814–819.

[12] S. Kaxiras, Z. Hu, M. Martonosi, "Cache decay: Exploiting generational behavior to reduce cache leakage power," *ISCA'01: Int. Symp. Computer Architecture*, Jun. 2001, pp. 240–251.

[13] K. Flautner, N. Kim, S. Martin, D. Blaauw, T. Mudge, "Drowsy caches: Simple techniques for reducing leakage power," *ISCA'02: Int. Symp. on Computer Architecture*, May 2002, pp. 148–157.

[14] M. Powell, S.H. Yang, B. Falsafi, K Roy, T.N. Vijaykumar, "Gated-Vdd: A Circuit Technique to Reduce Leakage in Deep-Submicron Cache Memories", *ISLPED'00: International Symposium on Low Power Electronics and Design*, July 2000, pp. 90 - 95.

[15] M. Kandemir, M. J. Irwin, G. Chen, I. Kolcu, "Compiler-guided Leakage Optimization for Banked Scratch-Pad Memories," *IEEE Transactionns on VLSI Systems*, Vol. 13, No. 10, Oct. 2005, pp. 1136–1146.

[16] X. Lu, Y. Fu, "Reducing Leakage Power in Instruction Cache using WDC for Embedded Processors,", *ASPDAC'05: Asia South Pacific Design Automation Conference*, Jan. 2005, pp. 1292 - 1295.

[17] J. Scott, L. Lee, J. Arends, B. Moyer, "Designing the Low-Power M·CORE Architecture", *Int'l. Symp. on Computer Architecture Power Driven Microarchitecture Workshop*, pp. 145–150, Jul. 1998.

[18] T. Givargis, F. Vahid, F., "Platune: a tuning framework for system-on-a-chip platforms", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 21, No. 11, Nov. 2002, pp. 1317–1327.

[19] M. R. Guthaus et al., "MiBench: A free, commercially representative embedded benchmark suite", *IEEE 4th Annual Workshop on Workload Characterization*, pp. 3–14, Dec. 2001.

[20] Simplescalar toolset, *http://www.simplescalar.com*

[21] M. Avriel, *Nonlinear Programming: Analysis and Methods*, Dover Publishing, 2003.