Journal of
Low Power Electronics
Vol. 4, 1–9, 2008

Low Power Main Memory Configuration and Tasks Allocation

Hanene Ben Fradj[1], Cécile Belleudy[1,*], Michel Auguin[1], and Alain Pegatoquet[2]

[1] *LEAT, University of Nice Sophia-Antipolis, France*
[2] *Texas Instruments, Villeneuve Loubet, France*

(Received: 14 February 2008. Accepted: 9 June 2008)

Modern Rambus DRAM technologies offer power management features for energy consumption optimization. It consists in multi-banking the memory addressing space instead of considering a monolithic module. The main advantage of this approach is the capability to set banks in low power modes when they are not accessed, such that only the accessed bank is maintained in active mode. In this paper we investigate how this power management capability can be handled for real-time and multi-tasking applications. Based on the application scheduler, the objective is to find an efficient allocation of application's tasks to memory banks as well as the corresponding memory configuration that lessen the energy consumption. Experiments on signal processing and multimedia applications show the effectiveness of this approach and the large energy savings.

Keywords: Low Power, Multi-Bank Main Memory, Memory Configuration, Tasks, Scheduling.

1. INTRODUCTION AND RELATED WORK

Memories in SoCs become increasingly broad especially for multimedia applications which handle a large quantity of data. As a consequence the power consumption of memories increases tremendously and can reach up to 80% of the global system's consumption.[1] The main memory is consuming an increasing part of the power budget and thus motivates efforts to improve energy efficiency. Recently, memories with multiple banks instead of a monolithic module appeared in several architectures such as RAMBUS-DRAM technology (RDRAM)[2] and Infineon Mobile-RAM (SDRAM).[3] This kind of memory architecture is exploited in order to reduce the energy dissipation by operating banks at different modes (Active, Standby, Nap, Power-Down . . .). Indeed, bank consumes most of the power when a read or write memory access is requested. In the other hand, an inactive bank can be put in any low power mode (Standby, Nap, Power-Down). Each mode is characterized by its power consumption and the time required to transit back to the active mode (resynchronization time). The lower the energy consumption of the low power mode, the higher the resynchronization time. Several techniques exploiting the low power modes of memories were recently published. Based on access data pattern analysis, these methods determine when to power down and into which low power mode it is possible to transit the memory banks. These memory controller policies are compiler-based,[4,5] hardware-assisted,[6] or operating system oriented.[7,8]

At compiler level, Kandemir et al.[4] studied the impact of loop transformations on banked memory architecture. Ozturk et al.[5] proposed an integer linear programming (ILP) based approach that returns the optimal non uniform sizes of banks and the data mapping to banks. However, these methods do not take into account resynchronization cost overhead as the memory bank usage is predicted at compile time for the target application. Moreover not all the information are available at compiler level and only mono-programming applications are considered.

For the hardware assisted techniques,[6] the self-monitored hardware transits banks automatically to low power modes based on information captured by the supporting hardware. These techniques showed better performance than the compiler based approach but need extra hardware which consumes itself energy.

The operating system-based approach has the advantage of a global view of the system, without introducing a performance or energy overhead. Lebeck et al.[8] proposed a scheme for reducing DRAM energy by power aware page allocation algorithm. Delaluz et al.[7] proposed a scheduler-based approach where a bank usage table is managed by the operating system, and all banks' power modes are reset at context switch.

*Author to whom correspondence should be addressed.
Email: belleudy@unice.fr

In this paper, we address the energy optimization problem in a multi-bank main memory, based on the operating system scheduler. At this level of the co-design, larger energy savings can be achieved without any performance or energy overhead. Our approach is considering data granularity is the combined task's data and code. We noted that results in previous compiler-based researches like[5,4] can be added to our approach by optimizing the data and code allocation of each task. Our objective is to find the optimal allocation of tasks to banks based on several parameters (task size, number of times the task (NexeTi) is executed during the hyperperiod, number of main memory accesses, number of preemptions between tasks…) and the corresponding memory configuration that reduces the overall memory energy consumption (optimal number of banks and optimal size of each bank). This paper is structured as follow: the Section 2 presents the memory architecture and the system model used in our approach. In the Section 3, parameters influencing memory consumption are studied and energy models for multi-task applications are developed at system level. In the Section 4 we focus on searching the low power tasks allocation to banks and the associated memory configuration (number of banks and banks size). Section 5 shows experiments and results obtained with our approach. Remarks and future works conclude this paper in Section 6.

2. MEMORY ARCHITECTURE AND SYSTEM MODEL

In this paper, multi-bank main memory architectures are considered: each bank can be controlled independently and placed into one of the available low power modes. Each low power mode is characterized by the number of components being disabled to save energy. As shown on the Figure 1, this multi-bank main memory communicates with an embedded processor through a L1 SRAM cache.

In order to address a wide range of applications, real-time and multi-tasking embedded applications are considered in this approach. This kind of application is described by a set of N periodic tasks where each task is characterized by:

- temporal parameters namely (Pi: period, ci: execution time)
- the number of memory accesses (Mi) and
- the task size of instructions and data (STi).

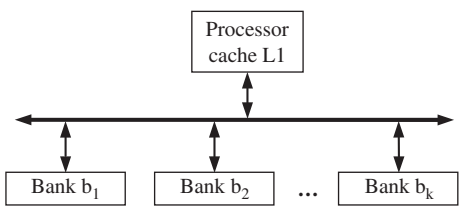

Fig. 1. Multi-bank main memory architecture.

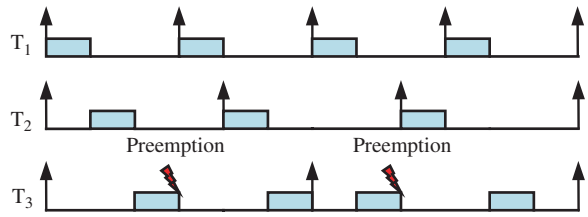

Fig. 2. RM schedule of the 3tasks set during the hyperperiod H of 12 time units.

Table I. Energy consumption (per cycle) and resynchronization times for different operating modes for a 8 MB RDRAM bank size.

Operating modes	Energy consumption (nJ)	Resynchronization cost (cycles)
Active	3.57	0
Standby	0.83	2
Nap	0.32	30
Power-down	0.005	9,000

Source: Reprinted with permission from [2], Rambus RDRAM Data Sheet 512/576 MBit, Rambus Inc. **(2003)**, http://www.rambus.com/us/products/rdram/documentation/ datasheets.html, © 2003.

These tasks are scheduled during an hyperperiod $H = LCM(P1, \ldots, PN)$ using for instance a fixed priority and preemptive Rate Monotonic (RM) algorithm. The corresponding scheduling is shown on the Figure 2 for the task set described in Table II.

3. ENERGY ESTIMATION AND MODELS

We define an allocation function noted φ that associates each task T_i from a set of N tasks to a bank b_j belonging to a set of k banks $(1 < k \leq N)$.

$$\varphi\colon \{T_1, T_2, \ldots T_N\} \rightarrow \{b_1, b_2, \ldots b_k\}$$

$$\varphi(T_i) = b_j$$

3.1. Parameters Influencing Memory Consumption

3.1.1. Bank Size

The energy consumption monotonically increases with the memory size. The analytical model given in (Ref. [9]) illustrates that the memory energy consumption increases with the number of lines and columns in the memory. For the multi-bank main memory, several papers consider that the energy values increase by $\tau_1 = 30\%$ when bank size is doubled.[5,6] In our approach, we consider that the size S_{b_j} of bank b_j is the sum of the size of all tasks T_i allocated

Table II. Example of 3 tasks set.

Task	c_i	P_i
T_1	1	3
T_2	1	4
T_3	2	6

to this bank:

$$S_{b_j} = \sum_{T_i/\varphi(T_i)=b_j} S_{T_i}$$

So, a given architecture can have banks of different sizes (non uniform banks sizes). A mathematic formula was developed to traduce this increase of 30% in the memory energies when the bank size is doubled for RDRAM technology. With Eq. (1), the energy values per memory cycle of a bank b_j with size S_{b_j} can be determined:

$$E_\alpha = E_{0\alpha}(1.3)^{Log_2(S_{b_j}/N)} \quad (1)$$

$\alpha = \{\text{active, lp-mode, resynchronization}\}$, $E_{0\alpha}$: The energy values for a bank size of N Mbytes.

3.1.2. Number of Banks

The multi-bank energy consumption also depends on the number of banks in the memory architecture. When a new bank is added, the size of some banks decreases (less tasks in these banks) as well as the energy values (active, low power mode and resynchronization). In (Ref. [10]) authors consider a 20% overhead for the first bank and then a decreasing overhead for each new added banks. However, we assume that the energy consumption for communication increases in worst case by $\tau_2 = 20\%$ when a new bank is added to the architecture. So for main memory architectures with k banks, the communication energy is described by Eq. (2).

$$E_{\text{bus}} = E_{0\text{bus}}(1.2)^{k-1} \quad (2)$$

$E_{0\text{bus}}$: The bus consumption for monolithic memory architecture.

In our approach, τ_1 and τ_2 can be easily adjusted for different technologies.

3.1.3. Successivity and Preemption Between Tasks

We call successivity between task T_i and task T_j noted σ_{ij} when T_j begins its execution just after the end of T_i or when a higher priority task (T_i or T_j) preempts the other one. The successivity parameters are deduced from the application's scheduling over the hyperperiod and are exploited to minimize the resynchronization numbers of banks and to make the idle periods longer. The resynchronization number of a bank b_j, i.e., the number of times a bank is activated is computed with the following equation.

$$N_{\text{resynchronization\_}b_j} = \sum_{T_i/\varphi(T_i)=b_j} N_{\text{exe}T_i} - \sum_{T_i,T_j/(\varphi(T_i),\varphi(T_j))=(b_j,b_j)} \sigma_{ij} \quad (3)$$

$N_{\text{exe}T_i}$ is the number of execution of task T_i during the hyperperiod H. From the RM schedule of Figure 2, $N_{\text{exe}T_1} = 4$, $N_{\text{exe}T_2} = 3$, and $N_{\text{exe}T_3} = 2$. The task T_3 is executed only two times over the hyperperiod but is preempted two times by tasks T_1 and T_2 respectively.

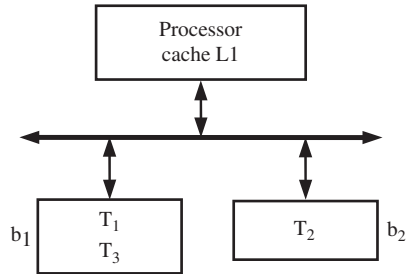

Fig. 3. Tasks allocation to 2 memory banks.

The successivity parameters between the three tasks are then: $\sigma_{12} = 3$, $\sigma_{13} = 4$, $\sigma_{23} = 3$. Considering the tasks allocation given in Figure 3, the resynchronization numbers of each bank are: $N_{\text{resynchronization\_}b_1} = N_{\text{exe}T_1} + N_{\text{exe}T_3} - \sigma_{13} = 4$, $N_{\text{resynchronization\_}b_2} = N_{\text{exe}T_2} = 3$.

By exploiting the successivity between tasks, the resynchronization numbers of banks and the energy associated can be minimized. However, reducing the energy of resynchronisation by grouping in the same bank the tasks having the maximum number of successivity may increase other energy contributions such as the size of the banks.

In summary, minimizing each memory energy contribution separately cannot usually minimize the overall memory consumption because of the strong interdependence between the memory parameters relevant to energy consumption. The problem can therefore be modeled as a problem of tasks allocation to banks with an objective of energy optimization.

3.2. Energy Models for a Multi-Bank Memory

The energy consumption of a memory composed of k banks and for a given allocation of N tasks to these banks is evaluated with Eq. (4).

$$E_{\text{memory}} = E_{\text{access}} + E_{\text{nonaccess}} + E_{\text{lpmode}} + E_{\text{resynchronization}} + E_{\text{preemption}} + E_{\text{bus}} \quad (4)$$

Unlike,[5] we consider an active mode separated into two different operating modes (as defined in Table I): the read/write mode (access) and active but idle mode (non-access). As it is very difficult to differentiate read and write access at system level, an average number of memory access is considered whatever read or write operation. All parameters used in the energy model are summarized in the Table III.

The energy E_{access} due to read or write accesses to the memory banks is defined as follows:

$$E_{\text{access}} = \sum_{b_j/j=1}^{k} \left(\sum_{T_i/\varphi(T_i)=b_j} N_{\text{cycles\_access\_}T_i} \times E_{0\text{access}}(1.3)^{Log_2(S_{b_j}/N)} \right)$$

Table III. Example of 3 task set.

Model parameters	Definition
Architectural parameters	
$E_{0\text{access}}$	The access energy value per memory cycle for an initial bank size
$E_{0\text{nonaccess}}$	The non access energy value per memory cycle for an initial bank size.
$E_{0\text{lpmode}}$	The resynchronization energy values for an initial bank size.
$E_{0\text{resynchronization}}$	The resynchronization energy values for an initial bank size.
$E_{0\text{bus}}$	The bus consumption for one bank main memory architecture (monolithic memory).
$E_{\text{context\_switch}}$	The preemption energy due to the context switching.
t_{accessM}	The main memory access time.
f_{memory}	The main memory frequency.
$f_{\text{processor}}$	The processor frequency.
Application parameters	
M_i	The number of task T_i main memory accesses
$N_{\text{cycles\_access\_}T_i}$	The number of memory access cycles of task T_i to the memory bank.
$N_{\text{cycles\_nonaccess\_}T_i}$	The number of memory cycles of Task T_i when the memory bank is active but idle.
$N_{\text{cycles\_lpmode\_}b_j}$	The number of memory cycles, the bank b_j spent in low power mode.
Scheduling parameters	
$N_{\text{resynchronization\_}b_j}$	The resynchronization number of bank b_j from low power mode to the active mode (equation 3)
S_{ij}	Successivity between tasks T_i and T_j
$N_{\text{exe}T_i}$	The number of execution of task T_i in the hyperperiod H
$N_{\text{preemptions}}$	The preemptions number of tasks in the hyperperiod

The number of cycles for memory access $N_{\text{cycles\_access\_}T_i}$ of a task T_i to a memory bank b_j is:

$$N_{\text{cycles\_access}T_i} = M_{\text{i}} \times t_{\text{accessM}} \times f_{\text{memory}}$$

When a memory bank is active but not servicing any read or write operation, the energy consumption $E_{\text{nonaccess}}$ is defined below. Note that this energy is essentially due to the co-activation of the memory bank with the task execution by the processor.

$$E_{\text{nonaccess}} = \sum_{b_j/j=1}^{k} \left(\sum_{T_i/\varphi(T_i)=b_j} N_{\text{cycles\_nonaccess\_}T_i} \times E_{0\text{nonaccess}} (1.3)^{Log_2(S_{b_j}/N)} \right)$$

When a memory bank is active but idle, the number of memory cycles for a Task T_i is:

$$N_{\text{cycles\_nonaccess}T_i} = \left(c_i \times \frac{1}{f_{\text{processor}}} - M_i \times t_{\text{accessM}} \right) \times f_{\text{memory}}$$

E_{lpmode} represents the energy consumed by a bank when in low power mode:

$$E_{\text{lpmode}} = \sum_{b_j/j=1}^{k} N_{\text{cycles\_lpmode\_}b_j} E_{0\text{lpmode}} (1.3)^{Log_2(S_{b_j}/N)}$$

When in low power mode, a bank b_j spends $N_{\text{cycles\_lpmode\_}b_j}$ memory cycles:

$$N_{\text{cycle\_lpmode\_}b_j} = \left(\text{H} - \sum_{T_i \varphi(T_i)=b_j} N_{\text{exe}T_i} \times c_i \right) \times \frac{f_{\text{memory}}}{f_{\text{processor}}}$$

The energy consumption $E_{\text{resynchronization}}$ represents the energy of transition for a memory bank from a low power mode to the active mode to service a memory request.

$$E_{\text{resynchronization}} = \sum_{b_j/j=1}^{k} N_{\text{resynchronization\_}b_j} \times E_{0\text{resynchronization}} (1.3)^{Log_2(S_{b_j}/N)}$$

The energy $E_{\text{preemption}}$ is induced by context switches due to the preemption between tasks on the processor:

$$E_{\text{preemption}} = N_{\text{preemption}} E_{\text{context\_switch}}$$

The energy consumption E_{bus} is used to represent the overhead banks interconnection.

$$E_{\text{bus}} = E_{0\text{bus}} (1.2)^{k-1}$$

4. SOLUTION EXPLORATION ALGORITHM

Our aim is to find both an allocation φ of tasks to a multi-bank memory and the number of banks with their respective sizes in order to minimize the overall energy consumption due to the main memory structure. In this study, only a single low power mode is considered.

4.1. Exhaustive Explorations

The strong interdependence of the different parameters influencing memory consumption makes the problem of memory allocation to a multi-bank memory NP-complete.[11] An exhaustive approach exploring all the configurations space was adopted first. This technique allows finding the optimal solution by comparing the energy of all the configurations and observing the memory consumption according to the variations of tasks allocation and system characteristics. The algorithm starts by finding all the banks configurations to arrange N tasks in a k memory bank structure with $k = 1$ to N. Each configuration is represented by a set of integers; the cardinal of the set (k) represents the number of banks in the memory architecture and the set's elements represent the number of tasks in a bank b_i.

Let us consider an application of 4 tasks, five bank configurations are then possible : {4}; {3, 1}; {2, 2}; {2, 1, 1}; {1, 1, 1, 1}. For a number of banks equals to two, there are two different configurations for arranging tasks in banks.

In the first one {3, 1}, three tasks are allocated to the first bank and one task is in the second while in the second configuration {2, 2}, two tasks are allocated to each bank. In a second step, for each configuration, the algorithm enumerates all possible permutations of tasks in banks. For the configuration {3,1}, allocating tasks T_1, T_2 and T_3 in the first bank b_1 and task T_4 in b_2, i.e., $\{(T_1, T_2, T_3)\ ;\ (T_4)\}$ does not consume the same energy if tasks T_2, T_3 and T_4 are allocated in the bank b_1 and task T_1 in bank b_2, i.e., $\{(T_2, T_3, T_4)\ ;\ (T_1)\}$. This is due for instance to the different number of banks resynchronization or bank sizes. In the example above, there are fifteen possibilities for arranging four tasks in a number of banks varying from one to four. As soon as all tasks permutations are exhibited for all bank configurations, the energy consumption of each solution is estimated based on energy models developed in Section 3.2. The optimal multi-banked memory configuration and the associated tasks' allocation are then obtained.

The total number of memory configurations is known as number of Bell (B_N), which is the sum of Stirling number of the second kind $S(N, k)$. $S(N, k)$ represents the number of memory configurations of N tasks in k banks.

$$S(N,k) = \frac{1}{k!}\sum_{j=1}^{k}(-1)^{k-j}\binom{k}{j}j^N \qquad B_N = \sum_{k=1}^{k} S(N,k)$$

Some numbers of Bell are presented to give an idea of the memory configurations increase with the number of tasks: $B_4 = 15$; $B_5 = 52$, $B_6 = 203$, $B_7 = 877$, $B_8 = 4140$, $B_9 = 21147$, $B_{10} = 115975\ldots$ $B_{20} = 51724158235372$. Although the exhaustive approach returns the optimal solution, the exploration space increases exponentially with the numbers of tasks. This approach is therefore impractical especially to take into account for example additional tasks that appear on-line (sporadic tasks for example).

4.2. Heuristic Approach Overview

In this section we propose a heuristic-based approach to efficiently resolve, in polynomial time, the power aware multi-bank main memory configuration and the corresponding tasks allocation. This two-step heuristic approach first generates an initial memory configuration that is then refined iteratively.

- *Step* 1: Initial solution generation

Initially all tasks are allocated to a same initial bank. Then the task that provides the greatest energy reduction when moved to an additional bank is isolated. This process is iteratively repeated on the remaining tasks. The choice of the task to isolate in an additional bank is based on a criteria computed for each task T_i and called *Isolation* (T_i).

This criterion is determined in order to detect the most consuming task in the first bank while expressing the energy benefit to isolate a task in a new bank. This criterion is based on the characteristics of the tasks, the scheduling and the memory architecture. The characteristics of the tasks use parameters already defined in the Section 3.2. The scheduling features of a task T_i are managed through the number of times this is executed over the hyperperiod (H/P_i), as well as its successivity parameter. $E_{0\text{access}}$, $E_{0\text{nonaccess}}$, $E_{0\text{resynchronization}}$, and $E_{0\text{lpmode}}$ energies reflects the memory energy consumption features.

$$Isolation(T_i) = \left[\begin{array}{l} \frac{H}{P_i} \times (N_{\text{cycles\_access\_}T_i} \times E_{0\text{access}} \\ \quad + N_{\text{cycles\_nonaccesss\_}T_i} \times E_{0\text{nonaccesss}} \\ \quad + c_i \times E_{0\text{lpmode}}) \\ \quad + \left(\frac{H}{P_i} - \sum_{j=1,\, j\neq i/\varphi(T_j)=b_1}^{N} S_{ij}\right) \\ \quad \times E_{0\text{resynchrnization}} \end{array}\right] \times \left(\frac{\sum_{j=1,\, j\neq i/\varphi(T_j)=1}^{N} S_{T_j} - S_{T_i}}{S_{T_i}}\right)$$

As shown above on the formula, this criteria supports the isolation of a task having many memory accesses and forcing the memory to remain a long time co-active with the processor execution. An isolated task allows putting for a long time the initial bank in low power mode. At last, a task that often activates the initial bank and at the same time has the lower size compared to the remaining tasks in the initial bank will be a good candidate for isolation. An energy evaluation is done for each new solution. If the memory consumption is reduced, this memory configuration is selected, the criteria are updated and a new iteration is performed to isolate another task from the initial bank. Otherwise, the current solution is rejected and the previous memory configuration solution is selected for

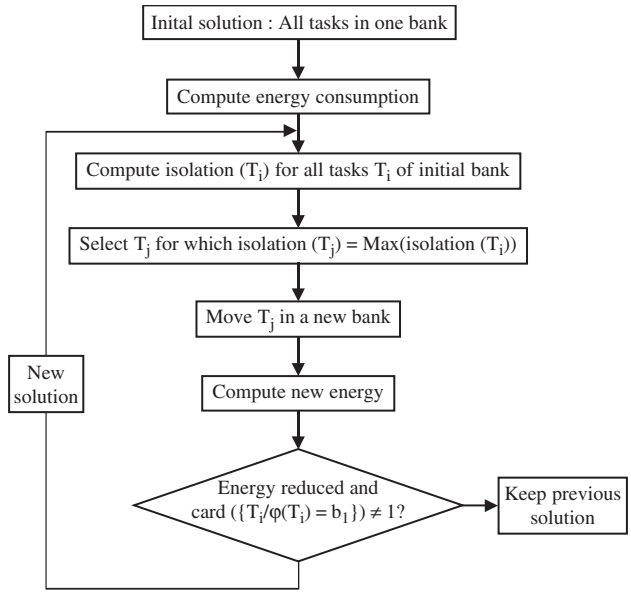

Fig. 4. Initial solution generation algorithm.

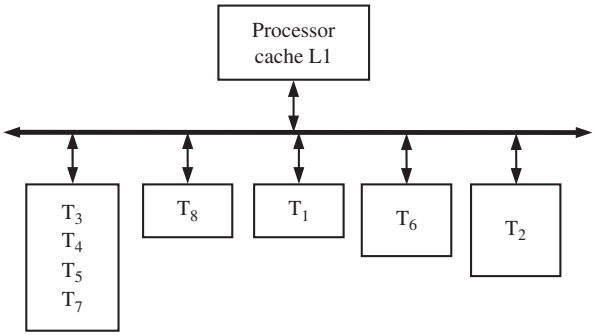

Fig. 5. Memory configuration example obtained with step 1.

an energy refinement (step 2). The Figure 4 graphically illustrates the heuristic used to generate an initial memory bank configuration.

For a set of eight tasks, Figure 5 shows an example of a memory configuration generated after step 1. Only the first bank contains more than one task, in the others banks one task is allocated.

- *Step* 2: Initial step refinements

A refinement of the memory configuration obtained in step 1 is performed in a second step. First, banks are sorted by an increasing order of their energy consumption. The task having the greatest *Isolation* (T_i) criteria is chosen to be allocated to the least consuming bank. An energy evaluation is performed for each new solution: if a reduction in the memory consumption is detected in the new solution, this memory configuration is kept and both the criteria of tasks in the initial bank and the banks consumption are updated. This process is iterated as long as tasks can be isolated from the initial bank and moved to the least consuming bank. Otherwise the previous memory configuration is kept. The Figure 6 below illustrates the heuristic used to refine the initial memory bank configuration.

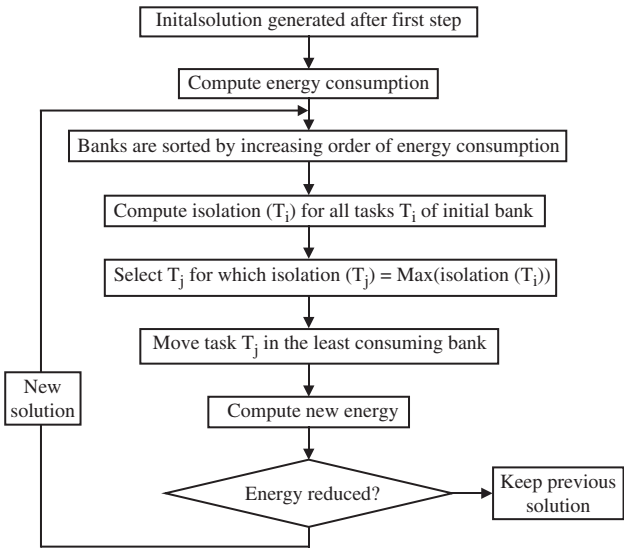

Fig. 6. Initial solution refinements algorithm.

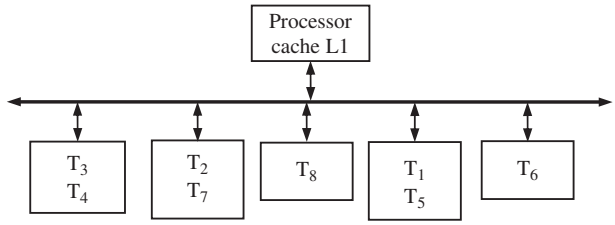

Fig. 7. Memory configuration example obtained with step 2.

For the same task set of Figures 5 and 7 depicts an example of a final memory configuration obtained with step 2. The complexity of the proposed heuristic is $O(N^3)$ while a complexity of $O(N^2)$ is observed for the energy evaluation.

5. EXPERIMENTS AND RESULTS

5.1. Signal Processing Application

For experiments, energy consumption and resynchronization costs for different operating modes of a 8 Mbytes RDRAM bank size is used (Table I). SNU real-time benchmarks are used from Seoul National University and described in Table IV. The SimpleScalar architectural simulation platform[12] has been used in order to estimate the tasks' parameters: the execution time c_i (average number), the size S_{T_i} (instructions and data) and the number of L1 cache misses representing also the number of main memory accesses (M_i). SimpleScalar has been modified in order to handle multi-tasks applications. A 32 kB 2-way set associative unified instruction/data cache with 2 cycle's latency access time and a RDRAM main memory with 20 cycle's latency are simulated. In (Ref. [6]) experiments show that the best energy savings is achieved with the *Nap* mode. This low power mode is therefore used in our experiments.

In this study, we take into account, in addition of the cold and intrinsic misses,[13] the extrinsic misses due to multi-task effect. Extrinsic misses occur when blocks of different tasks are in conflict for the same cache line. The DLite! debugger of SimpleScalar is used to insert breakpoints at the beginning and at the end of each task's instance and to collect the number of misses between two breakpoints. This way the number of misses for each task's

Table IV. Benchmark's description.

Benchmarks	Description
IDCT	Inverse discrete cosine transform.
ADPCM	CCITT G.722 adaptive differential pulse code modulation.
FIR	FIR filter with gaussian number generation.
Fibcall	Fibonacci series function.
Qsort	Non-recursive quick sort algorithm.
FFT	Fast fourier transform using cooley-tukey algorithm.

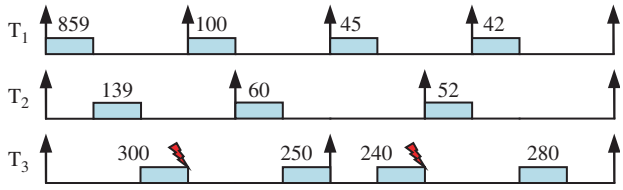

Fig. 8. Cache misses for a multi-task system.

instance in a multi-task execution system is obtained as depicted in Figure 8 (the bold numbers represent the number of L1 cache misses).

In our energy models, the average number of cache misses over the hyperperiod is used for each task. For example, the schedule in Figure 8 shows an average M_1 of 262 misses for the task T_1. The tasks characterizations obtained with SimpleScalar are presented in Table V.

Due to the important number of memory configurations (203), only the minimal energy consuming configuration for each bank is presented on Figure 9.

The Figure 9 shows that the main memory consumption decreases when a new bank is added to the architecture until the optimal number of bank. Compared to a monolithic architecture, the energy savings is about 24%. The minimum consumption is obtained with a memory architecture of four banks with the following memory configuration: {(FIR, Fibcall, Qsort); FFT; IDCT; ADPCM}. As shown on the Figure 10, exceeding four banks will increase the overall memory consumption. Indeed, adding a fifth bank to the architecture does not significantly reduce E_{access} and $E_{\text{nonaccess}}$ while it significantly increases E_{lpmode}, $E_{\text{resynchronization}}$ and E_{bus}, leading to more important overall energy consumption.

5.2. Multimedia Application

5.2.1. Applications and Platform Description

The multimedia application is composed of a GSM baseband modem and a MPEG-2 decoder. The GSM signal processing chain for both uplink and downlink path is well depicted in.[14] The MPEG-2 decoder divided images in macro blocks. Each macro block is composed of 6 blocks of 8×8 pixels and is decoded as shown on the Figure 11.

This application is executed on a Texas Instruments OMAP1510 dual-core platform.[15] Although each processor core has its own internal SRAM memories (caches, Scratchpads), an on-chip SRAM memory is used for communicating and sharing information between the ARM and the DSP processors. More storage capacity is available through synchronous (SDRAM, RDRAM, and DDR SDRAM) or asynchronous (Fast Page Mode, Extended Data Out) external memories.

Table V. Task set 1.

Tasks	P_i (cycles)	c_i (cycles)	S_{T_i} (kB)	M_i
IDCT	250,000	16,131	33	193
ADPCM	10,000,000	2,486,633	133	4,053
FIR	1,000,000	33,983	152	133
Fibcall	1,000,000	9,536	27	114
Qsort	1,000,000	13,309	31	97
FFT	5,000,000	515,771	97	38,404

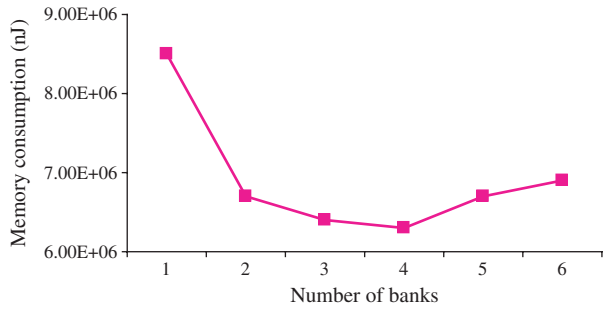

Fig. 9. Multi-bank main memory consumption.

5.2.2. Application Partitioning and Scheduling

The partitioning of tasks on the ARM or DSP processors has been based on the nature of the tasks. The DSP is in charge of signal processing functions while the ARM takes care of control type processing. Accordingly, Speech Coding (SC) and Decoding (SD), Channel Coding (CC) and Decoding (CD), the Inverse Discrete Cosine Transform (IDCT), Motion Compensation Prediction (MCP) and Addition (ADD) are executed on the DSP as shown in Figure 12.

Tasks scheduling is performed over a 20 ms speech period. During this time, four blocks of 8×8 pixels are decoded as well. From this scheduling, the successivities between tasks σ_{ij} as well as the number of executions for the different tasks are determined. The characteristics of each MPEG-2 decoder task are obtained using the SimpleScalar tool. For the GSM modem application, tasks

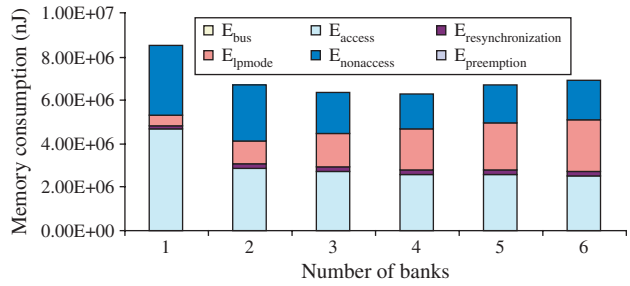

Fig. 10. Main memory energies contributions.

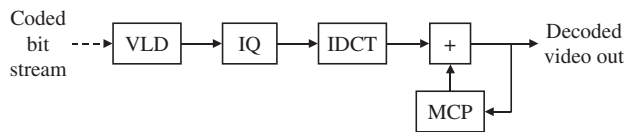

Fig. 11. MPEG-2 decoder block diagram.

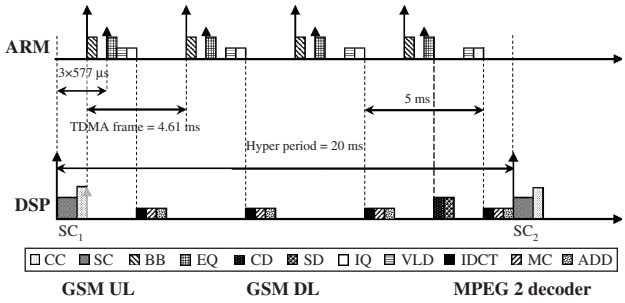

Fig. 12. GSM modem and MPEG-2 decoder application scheduling.

Table VI. Application description.

MPEG-2	P_i (ms)	c_i (cycles)	S_{T_i} (kB)	M_i
MC	5	87,836	213	888
VLD	5	58,783	281	1,465
IQ	5	12,922	29	355
IDCT	5	16,131	33	193
GSM task				
BB	4.615	2000	2.15	456
EQ	4.615	23,000	5.27	502
SC	20	36,000	1.43	8, 863
SD	20	10,000	1.64	2,435
CC	20	6,875	1.4	1,700
CD	20	15,140	1.45	3,626

features are taken from (Ref. [14]). Table VI summarizes all the features for the application tasks.

As shown on the Figure 13, using our approach an optimal configuration is obtained with 3 banks. An energy saving of 37% compared to the most consuming configuration (10 banks) is obtained.

5.3. Heuristic Approach Strength

In order to evaluate the efficiency of the proposed heuristics, we have considered three task sets described respectively in Tables V, VII, and VIII. Table IX presents both the optimal configuration and the solution obtained using our heuristic approach for each task set. Results clearly show that our methodology provides a solution very close, in terms of energy, to the optimal allocation.

Concerning the computation time, the exhaustive exploration takes less than 1 second for a 7 tasks set. For 8 tasks, only 11 seconds are necessary to explore all the memory configurations, 106 seconds for 9 tasks and about 25 minutes for 10 tasks. These durations were obtained using a PC running at 2.6 GHz with 1 Giga byte of memory. Concerning the heuristic algorithm, though the computation time depends on the number of iterations, the solution is generally found in less than one second.

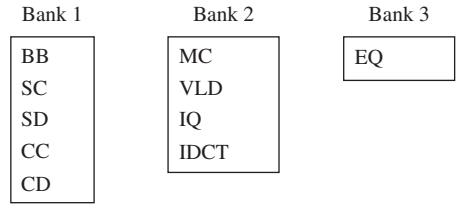

Fig. 13. The optimal main memory configuration.

Table VII. Task set 2.

Tasks	P_i (cycles)	c_i (cycles)	S_{T_i} (kB)	M_i
CRC	1,000,000	42,907	31	99
Fir	1,000,000	33,983	152	47
FFT	5,000,000	515,771	97	493
LMS	5,000,000	365,893	32	123
LUD	5,000,000	255,998	38	102
Matmul	1,000,000	13,985	29	26
ADPCM	10,000,000	2,486,633	139	3,387

Table VIII. Task set 3 from.

Tasks	P_i (cycles)	c_i (cycles)	S_{T_i} (kB)	M_i
FFT	320,000	60,234	96	280
LUD	1,120,000	255,998	38	364
LMS	1,920,000	365,893	32	474
FIR	6,000,000	557,589	152	405

Source: Reprinted with permission from [16], C. Lee et al. Bounding cache-related preemption delay for real-time systems. *IEEE Transactions on Software Engineering* 27, 805 **(2001)** © 2001.

6. CONCLUSION AND FUTURE WORKS

For the multi-bank memory consumption problem, two approaches operating at system level and for real-time applications were proposed. An exhaustive algorithm that returns the optimal allocation of tasks to banks was first developed. Using this approach the memory consumption can be reduced up to 24% compared to the most consuming memory configuration. The exponential

Table IX. Energy comparison of exhaustive algorithm and the heuristic approach solutions.

	Optimal allocation		Heuristic solution		
	Memory configuration	Energy (nJ)	Memory configuration	Energy (nJ)	(%) Gap
Set 1 Table V	{(FIR, Fibcall, Qsort); FFT; IDCT; ADPCM}	$6.39 \cdot 10^6$	{(FIR, Fibcall, Qsort); FFT; IDCT; ADPCM}	$6.39 \cdot 10^6$	0
Set 2 Table VII	{(CRC, FIR, FFT, LUD, Matmul); (ADPCM, LUD)}	$4.20 \cdot 10^6$	{(CRC, FIR, FFT, LUD, Matmul, LUD); ADPCM};	$4.23 \cdot 10^6$	0.8
Set 3 Table VIII	{FFT; LUD; LMS; FIR}	$1.4 \cdot 10^7$	{FFT; FIR; LUD; LMS}	$1.4 \cdot 10^7$	0
Multi-media Table IV	{(BB, SC, SD, CC, CD); (MC, VLD, IQ, IDCT); EQ}	$9.686 \cdot 10^6$	{MC, VLD, IQ, IDCT,); SC; SD; CD; BB; CC;EQ }	$9.688 \cdot 10^6$	0.02

complexity and the corresponding computational time of this exhaustive approach lead to the development of a 2-step heuristic based approach. Experiments presented in this article show the effectiveness of this approach to reduce the overall main memory (Rambus DRAM) energy consumption. However multi-bank memory architecture increases the area especially for embedded DRAM.[17] As future work, the energy saving versus area optimization trade-off will be taken into account. A more sophisticated model of the main memory accesses could also be investigated, as the number of access is considered as a constant in our methodology. An extended model of energy consumption based on both the main memory and the cache accesses will help designers to adapt the architecture of the cache to reduce the main memory consumption. Moreover, the proposed approach has been developed for static applications. It will be interesting to study the case of dynamic applications where the number of tasks and the memory requirements are variable during execution. We also plan to investigate the energy aware allocation of code and data of real-time operating systems, as multi-task and preemptive applications are aimed.

References

1. ITRS, System drivers, **(2005)**, http://www.itrs.net/Links/2005ITRS/SysDrivers2005.pdf.
2. Rambus RDRAM Data Sheet 512/576 MBit, Rambus Inc. **(2003)**, http://www.rambus.com/us/products/rdram/documentation/datasheets.html.
3. Mobile-RAM data sheet, Infineon Inc **(2004)**.
4. M. Kandemir, I. Kolcu, and I. Kadayif, Influence of loop optimizations on energy consumption of multi-bank memory systems. *Proc. Compiler Construction* **(2002)**.
5. O. Ozturk and M. Kandemir, Nonuniform banking for reducing memory energy consumption. *DATE'05 Munich*, Germany **(2005)**.
6. V. Delaluz, M. Kandemir, N. Vijaykrishnan, A. Sivasubramaniam, and M. J. Irwin, DRAM energy management using software and hardware directed power mode control, HPCA **(2001)**, pp. 159–169.
7. V. Delaluz, M. Kandemir, N. Vijaykrishnan, A. Sivasubramaniam, and M. J. Irwin, Scheduler-based DRAM energy management, design automation conference, *DAC* **(2002)**, pp. 697–702.
8. A. R. Lebeck, X. Fan, H. Zeng, and C. Ellis, Power aware page allocation. *ASPLOS* **(2000)**.
9. K. Itoh, K. Sasaki, and Y. Nakagome, Trends in low-power RAM circuit technologies. *Proc. IEEE* 83, 524 **(1995)**.
10. L. Benini, A. Macci, and M. Poncino, A recursive algorithm for low-power memory partitioning. ISLPED, Rapallo, Italy **(2000)**.
11. A. H. Farrahi, G. E. Tellez, and M. Sarrafzadeh, Exploiting sleep mode for memory partitioning and other applications. *VLSI Design Journal* 7, 271 **(1998)**.
12. D. Burger and T. M. Austin, The simplescalar tool set, version 2.0. *Univ. of Wisconsin-Madison Computer Sciences Dept. Technical Report # 1342* **(1997)**.
13. A. Agarwal, J. Hennessy, and M. Horowitz, An analytical cache model. *ACM Transactions on Computer Systems (TOCS)* 7, 184 **(1989)**.
14. E. Auslander and M. Couvrat, Take the lead in GSM. *Applications of Digital Signal Processing, Proc. DSP* **(1994)**.
15. Texas Instruments OMAP1510 **(2001)**. http://focus.ti.com/pdfs/vf/wireless/omap1510_bulltn.pdf.
16. C. Lee, K. Lee, J. Hahn, Y. Seo, S. L. Min, R. Ha, S. Hong, C. Y. Park, M. Lee, and C. S. Kim, Bounding cache-related preemption delay for real-time systems. *IEEE Transactions on Software Engineering* 27, 805 **(2001)**.
17. T. Yamauchi, L. Hammond, and K. Olukotun, The hierarchical multibank DRAM: A high-performance architecture for memory integrated with processors. Advanced Research in VLSI, Ann Arbor, MI, USA **(1997)**, pp. 303–319.
18. T. Pering, T. Burd, and R. Brodersen, The simulation and evaluation of …. *Proceedings of the IEEE International Symposium on Low Power Electronics and Design* **(1998)**, pp. 76–81.

Hanene Ben Fradj

Hanene Ben Fradj *received her Ph.D. from the University of Nice Sophia Antipolis in 2006. She is actually an assistant professor in the CIRTACOM team at SUPCOM school of tunis. His research interests include low power design techniques.*

Cécile Belleudy

Cécile Belleudy *joined the Electronics, Antennas, Telecommunications Laboratory (LEAT) at the University of Nice_Sophia Antipolis as an assistant professor. Her current work focuses on studying and developing low power strategies for embedded system. More especially, she addresses the problem of low power RTOS for monoprocessor and multiprocessor platforms and energy aware memory allocation. In the field of low power SoC design methodologies, she participates in national and European collaborative projects.*

Michel Auguin

Michel Auguin *has currently a position of Research Director at CNRS in the group "System level modelization and design of communicating objects" of the LEAT laboratory from University of Nice Sophia Antipolis in France and CNRS. In this group he is working on SoC system level design methodologies. Previously, he has been involved since 1980 and for nearly 15 years in the area of parallel processing and architecture. Since 1995 he has been a staff member of several national research programs focusing on parallel architecture and SoC. In the field of SoC design methodologies he currently participates to regional, national and European collaborative projects. Since October 2004 he heads the Ph.D. program on embedded systems at University of Nice Sophia Antipolis.*

Alain Pegatoquet

Alain Pegatoquet *is a system DSP engineer with 13 years of experience in the semiconductor industry for wireless terminals. Since 2003, he is working for Texas Instruments (France) within the Layer-1 team of the Wireless Terminal Software Development group. His research interests includes virtual prototyping and approaches for low power. Alain Pegatoquet received his M.S. and Ph.D. degrees in Computer Engineering from Nice University in 1995 and 1999.*