Static Energy Saving Through Multi-Bank Memory Architecture

Sébastien Lafond
Turku Centre for Computer Science
Embedded Systems Laboratory
Lemminkäisenkatu 14A, FIN-20520 Turku, Finland
Email: sebastien.lafond@abo.fi

Johan Lilius
Åbo Akademi University
Department of Information Technologies
Lemminkäinengatan 14A, FIN-20520 Åbo, Finland
Email: johan.lilius@abo.fi

***Abstract*— Managing the energy consumption of embedded systems has become a major problem with the increasing demand for portable electronic devices. This paper propose a multi-bank memory architecture as a solution to decrease the static energy cost in memory. We set up the equations ruling the optimization problem for decreasing the memory static energy cost, analyze the impact of different parameters on the energy cost and finally present some case study results.**

I. Introduction

In recent years we have seen an explosion of the market for portable electronic devices such as PDAs, personal communicators and mobile phones. They have in common strong constraints on energy consumption, and thus maximizing battery life for such devices is crucial.

Several studies [1] show that memory is becoming a predominant energy consumption component in handheld devices. As the static energy due to leakage currents is becoming the major element of memory energy consumption [2], a reduction of the static energy cost will have a significant impact on the overall system energy consumption. In traditional systems one continuous memory region is generally used to store dynamic memory allocation and its size must to be sufficiently large to hold in any case all allocations. This required size is most of the time oversized for the average allocation behavior of the application(s), leading to a waste of static energy in the memory area used only during the worst cases. In order to cut down the cost associated to this 'most of the time' unused memory area we propose a multi-bank memory architecture (MBMA) as a solution to decrease the static energy cost of the memory. Such architecture would have the ability to follow the application(s) memory need by adjusting the number of memory bank switched on. Fig. 1 simply illustrates the general behavior of such MBMA. In Fig. 1, which doesn't take into account the possible fragmentation in a bank, the maximal energy savings would be proportional to the size of the *switched off* memory area. The remain wasted static energy would be proportional to the size of *Free Memory* area.

The major contributions of this paper are: 1) the introduction of a complete static energy costs model for a multi-bank memory architecture, 2) the establishment of the equations governing optimization problem for decreasing the static energy consumed by the memory and an analysis of the impact of the different parameters on energy consumption and 3) the performance analysis of such architecture in terms of static energy consumption and execution speed.

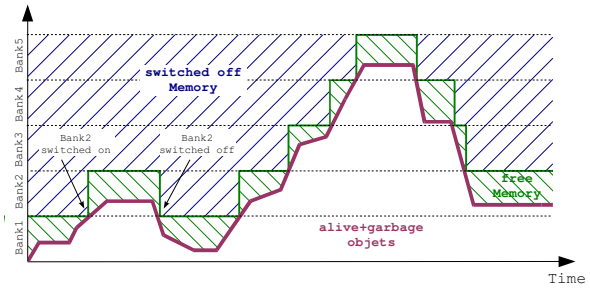

Fig. 1. MBMA behavior

The paper is organized as follow: we briefly describe software allocation behavior and the general memory model for static energy consumption. We then present the static energy consumption model for a MBMA and two reference architectures. Then we setup the optimization problem for the MBMA static energy cost and show some simulation results. We conclude with a discussion of related and future work.

II. Allocation behavior

Dynamic memory allocation is the assignment of memory block(s) to store specific data used during the runtime of an application. A dynamically allocated memory block remains in $allocated\ state$ until it is explicitly deallocated or implicitly deallocated by the means of a garbage collector (GC). Before a block is deallocated it can be in an intermediate state, called garbage, where the block content is no longer needed by the application.

The software applications(s) is driving the allocation t_a, garbage t_g and deallocation t_d events occurring in the memory. We define the variables $free$, $alive$ and $garbage$ as the total memory size of blocks in respectively free, alive and garbage state. For each new event concerning b blocks of size S Table I presents the variables updates triggered by the event.

When different size objects are dynamically allocated, a deallocated block might let an free and unuseable space for the following allocations. In that case this space contributes to memory fragmentation which denotes a waste of the memory.

TABLE I

EVENTS TRIGGERING VARIABLES UPDATES

Event	Variables updates *(occur simultaneously)*
t_a	$free = free - b \cdot S$ $alive = alive + b \cdot S$
t_g	$alive = alive - b \cdot S$ $garbage = garbage + b \cdot S$
t_d	$garbage = garbage - b \cdot S$ $free = free + b \cdot S$

There is commonly two types of fragmentation : internal and external. Internal fragmentation refers to waste in the memory due to alignment and storage of additional information needed by the allocator such as bookkeeping. External fragmentation describes on the other hand a waste due to holes of free memory interspersed with live objects. As this study aims to analyze the external fragmentation in the context of MBMA, in the remain of this paper the term fragmentation means external fragmentation.

There is several ways to evaluate memory external fragmentation. One can measure it as a percentage of the actual memory usage or as a percentage of the amount of live objects. This late approach has been used by Johnstone and Wilson in [3]. However their four proposed ways to measure fragmentation require knowledge about the past allocation behavior which demand extra storage place if the system has to manage by itself on the pertinence to launch a memory compaction. In the context of MBMA we propose to compute the instantaneous fragmentation for each bank as:

$$fragmentation \quad = \quad 1 - \frac{largest\ free\ area}{total\ free\ area} \qquad (1)$$

With this evaluation a bank containing all its free memory in one continuous area has a fragmentation of 0. If the largest free area tends to be relatively small compare to the total free area the bank fragmentation will approach 1. Additionaly a fully allocated bank is considered to have fragmentation of 0. This fragmentation evaluation method has several advantages: (a) it returns a value relative to the largest free block available, (b) the returned value is bounded between 0 and 1 and (c) it compute the total free memory available in the bank. These advantages can be used to precisely evaluate the state of each bank in term of fragmentation and potential memory availability for future allocations.

III. MEMORY MODEL

There are two main families of RAM technology: Static Ram (SRAM) and Dynamic RAM (DRAM). SRAM's store each bit in a memory cell that are basically flip flops build from six CMOS transistors. The static dissipation of SRAM's is due to the leakage current of each memory cell. During the idle phase the SRAM cell leakage current for a given threshold voltage and temperature will be constant. Thus the static energy consumed by an SRAM during idle time is proportional to its number of cells [4]. DRAM's stores each bit in a memory cell consisting of one capacitor and one transistor. In addition to the transistor leakage currents the DRAM static energy consumption should also include the energy dissipated in order to refresh the cells. Assuming that there is a $\frac{1}{2}$ probability for a cell to hold 1, the energy needed in order to refresh the memory cells will be proportional to half memory size.

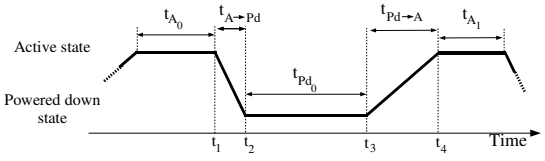

Fig. 2. Memory state transitions

A the static energy cost for a RAM or DRAM memory is proportional to its size, for a determined technology the average static power dissipated by a memory can be modeled by the following equation :

$$\overline{P}_S = k \cdot Size \qquad (2)$$

where $\overline{P}$ represents the average static power dissipated by the memory, $Size$ the size of the memory and k a constant factor depending on the memory technology and hardware implementation.

We adopt a two-state memory model consisting of one active and one powered down state. In active state the memory can be accessed for read and write operation and dissipates an average static power of $\overline{P} = k \cdot Size$. In powered down state data retention is not required and the memory does not consume any energy.

Fig. 2 represents the memory state transitions and define $t_{A \to Pd}$ and $t_{Pd \to A}$ respectively the time needed to switch from state active into powered down and conversely. From Fig. 2 we assume that the initial and last state of the memory is powered down and the memory has been in active state N times. We can then define T_A, T_{Pd}, $T_{A \to Pd}$ and $T_{Pd \to A}$ as:

$$T_A = \sum_{i=0}^{N} t_{A_i} \qquad T_{Pd} = \sum_{i=0}^{N} t_{Pd_i}$$
$$T_{A \to Pd} = N \cdot t_{A \to Pd} \qquad T_{Pd \to A} = N \cdot t_{Pd \to A}$$

where T_A represents the total time during which the memory was in active state, T_{Pd} the total time during which the memory was in powered down state , $T_{A \to Pd}$ the total time during which the memory was in transition phase from active state into powered down state, and $T_{Pd \to A}$ the total time during which the memory was in transition phase from powered down state into active state.

IV. MBMA MODEL FOR STATIC ENERGY COST

The energy cost model for a MBMA consisting of B banks has the following parameters: the average static power dissipated by the i^{th} bank $\overline{P}_{S_i}$, the total time during which the i^{th} bank was powered on in active state T_{A_i}, the average static power dissipated by the i^{th} bank in transition phase

from active state into powered down state $\overline{P}_{(A\rightarrow Pd)_i}$ and from powered down state into active state $\overline{P}_{(Pd\rightarrow A)_i}$, the total time during which the i^{th} bank was respectively in transition phase from active into powered down state $T_{(A\rightarrow Pd)_i}$ and vice versa $T_{(A\rightarrow Pd)_i}$. The static energy cost model for a MBMA composed of B banks is then defined by:

$$\begin{aligned} Es_{total} &= \sum_{i=1}^{B}[(\overline{P}_{S_i}\cdot T_{A_i}) + (\overline{P}_{(A\rightarrow Pd)_i}\cdot T_{(A\rightarrow Pd)_i}) \\ &\quad +(\overline{P}_{(Pd\rightarrow A)_i}\cdot T_{(Pd\rightarrow A)_i})] \end{aligned} \quad (3)$$

During transition phases the instantaneous dissipated power will most likely not be constant. In order to simplify the expression we use the average power $\overline{P}_{(A\rightarrow Pd)_i}$ and $\overline{P}_{(Pd\rightarrow A)_i}$instead of the respective literal expressions $\int_{t_1}^{t_2} P_{A\rightarrow Pd}(t)\cdot dt$ and $\int_{t_3}^{t_4} P_{Pd\rightarrow A}(t)\cdot dt$. This simplification doesn't change the correctness of Es_{total} definition if we consider that $t_{A\rightarrow Pd}$, $t_{Pd\rightarrow A}$, $P_{A\rightarrow Pd}(t)$ and $P_{Pd\rightarrow A}(t)$ are equal for each active to powered down and powered down to active transitions. With $BankSize_i$ standing for the i^{th}bank size we have then :

$$\begin{aligned} Es_{total} &= \sum_{i=1}^{B}[(k\cdot BankSize_i\cdot T_{A_i}) \\ &\quad +(\overline{P}_{(A\rightarrow Pd)_i}\cdot T_{(A\rightarrow Pd)_i}) \\ &\quad +(\overline{P}_{(Pd\rightarrow A)_i}\cdot T_{(Pd\rightarrow A)_i})] \end{aligned} \quad (4)$$

For each instant t, $BankSize_i$ can be further refined as :

$$BankSize_i = alive(t)_i + [(free(t)_i + garbage(t)_i)] \quad (5)$$

During power down states $alive(t)$, $free(t)$ and $garbage(t)$ are considered to be null. During the transition $t_{(Pd\rightarrow A)_i}$, $free(t)_i = BankSize_i$ and $alive(t)_i = garbage(t)_i = 0$. During transition $t_{(A\rightarrow Pd)_i}$ values for $live(t)_i$, $dead(t)_i$ and $free(t)_i$ are considered constant and equal to their respective last active state value. Thus $BankSize_i\cdot T_{A_i}$ in (4) can also be expressed by :

$$\begin{aligned} BankSize_i\cdot T_{A_i} &= \int_0^{T_{A_i}} alive(t)_i\cdot dt \\ &\quad +\int_0^{T_{A_i}} free(t)_i\cdot dt \\ &\quad +\int_0^{T_{A_i}} garbage(t)_i\cdot dt \\ &= Alive_i + Free_i + Garbage_i \end{aligned} \quad (6)$$

And Es_{total} from (4) can be re-expressed as :

$$\begin{aligned} Es_{total} &= \sum_{i=1}^{B}[(k\cdot(Alive_i + Free_i + Garbage_i)) \\ &\quad +(\overline{P}_{(A\rightarrow Pd)_i}\cdot T_{(A\rightarrow Pd)_i}) \\ &\quad +(\overline{P}_{(Pd\rightarrow A)_i}\cdot T_{(Pd\rightarrow A)_i})] \end{aligned} \quad (7)$$

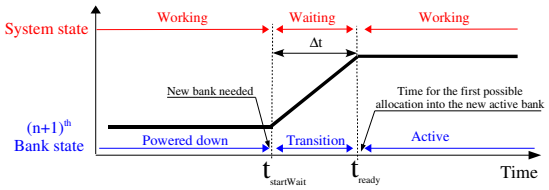

Fig. 3. Waiting state

During the system life the MBMA can be in two different states : *waiting state* and *working state*. A waiting state, as illustrated on Fig. 3, occurs when the system needs to allocate a new object on the memory but all switched on banks are full. In that case, if the system was not able to anticipate this allocation need beforehand, the system will have to wait for a time $\triangle t$ until a new bank has been switched on. The waiting cost is expressed by :

$$E_{wait} = \sum_{i=1}^{n}[BankSize_i\cdot k\cdot\triangle t] + \overline{P}_{wait}\cdot\triangle t \quad (8)$$

where E_{wait} represents the energy cost wasted during the waiting state and $\overline{P}_{wait}$ the average power dissipated during the waiting time by the $(n+1)^{th}$ bank which is in transition phase. $\triangle t$ is bounded by value 0 and $(t_{A\rightarrow Pd} + t_{Pd\rightarrow A})$ if we assume that the transition phase from active into powered down state can't be interrupted before it ends.

During the working state the MBMA has enough free available memory space for new allocation, and the need for new bank doesn't exist. With l and m representing respectively the numbers of active to powered down and powered down to active transitions occurred during T_{A_i}, $P_{Trans.}(t)$ the function describing the instantaneous power during the period from last transition request to the new transition request time , each bank is consuming during active states:

$$\begin{aligned} E_{working_i} &= k\cdot BankSize\cdot T_{A_i} \\ &\quad +l\cdot(\overline{P}_{(A\rightarrow Pd)}\cdot t_{(A\rightarrow Pd)}) \\ &\quad +m\cdot(\overline{P}_{(Pd\rightarrow A)}\cdot t_{(Pd\rightarrow A)}) \\ &\quad +\int_{t.Last}^{t_{Rpd}} P_{Trans.}(t)\cdot dt \end{aligned} \quad (9)$$

For a B banks system where W waiting states occur we can express Es_{total} as :

$$Es_{total} = \sum_{k=1}^{W} E_{wait_k} + \sum_{i=1}^{B}[E_{working_i}] \quad (10)$$

In order to simplify the model one could assume that the average power dissipated by banks during transition phases from active into powered down state, and analogously for powered-down into active state, is the same for all such transitions. In addition a conservative simplification would assign to $\overline{P}_{(A\rightarrow Pd)}$, $\overline{P}_{(Pd\rightarrow A)}$ and $\overline{P}_{wait}$ the $\overline{P}_S$ value representing the upper bound for the functions $P_{(A\rightarrow Pd)(t)}$, $P_{(Pd\rightarrow A)}(t)$ and $P_{wait}(t)$. In the same way $\triangle t$ can be simplified by its

maximal value $(t_{A\rightarrow Pd}+t_{Pd\rightarrow A})$. From (4) Es_{total} can then be re-expressed as:

$$Es_{total}=\sum_{i=1}^{B}[(k\cdot BankSize_i\cdot(T_{A_i}+T_{(A\rightarrow Pd)_i}+T_{(Pd\rightarrow A)_i})] = k\cdot\sum_{i=1}^{B}[\int_0^{(T_{A_i}+T_{(A\rightarrow Pd)_i}+T_{(Pd\rightarrow A)_i})} BankSize_i\cdot dt] \quad (11)$$

Based on these assumption we can simplify (8) and (9):

$$E_{wait}=k\cdot\sum_{i=1}^{n+1}[\int_{t_{LastTrans}}^{t_{Active}}(alive(t)_i+(free(t)_i+garbage(t)_i)\cdot dt]$$

$$E_{working_i}=k\cdot\int_0^{T_{A_i}}(alive(t)_i+free(t)_i+garbage(t)_i)\cdot dt +k\cdot l\cdot\int_0^{t_{A\rightarrow Pd}}(alive(t)_i+free(t)_i+garbage(t)_i)\cdot dt +k\cdot m\cdot\int_0^{t_{Pd\rightarrow A}}(alive(t)_i+free(t)_i+garbage(t)_i)\cdot dt$$

From now on, if not explicitly mentioned we will always refer to this simplified model.

V. Reference architectures

The ideal MBMA would be composed of an infinity of 1 bit size banks with the ability to be instantaneously switch on and off. Such ideal MBMA would permanently be able to adjust its memory size (i.e. the total size of the all banks that are powered on) to the exact system needs and thus reaches the obtainable minimum static energy consumption due to leakage current without any additional time penalty. This ideal system is reducing the functions $free(t)_i$ and $garbage(t)_i$ as well as the value of $t_{(A\rightarrow Pd)_i}$ and $t_{(Pd\rightarrow A)_i}$ to the constant zero. The ideal model can be modeled by the following equations :

$$free(t)=garbage(t)=O \quad (12)$$

$$Es_{total}=k\cdot\sum_{i=1}^{\infty}[\int_0^{T_{A_i}} alive(t)_i\cdot dt] \quad (13)$$

$$E_{wait}=0 \quad (14)$$

$$E_{working_i}=k\cdot\int_0^{T_{A_i}} alive(t)_i\cdot dt \quad (15)$$

This theoretically best solution to reduce static energy cost can't be obtained for evident physical constraints, but we will use it as a reference.

Compare to this idealistic architecture, a 'real life' MBMA has three additional costs: the sum of E_{wait} for all waiting states, E_{wasted_i} representing the static energy consumed by $Free$ and $Garbage$ memory area during all system working states for the i^{th} bank and E_{trans_i} representing the static energy consumed by state transition during working states for the i^{th} bank. For each active state E_{wasted_i} and E_{trans_i} can be expressed using the simplified model by :

$$E_{wasted_i}=k\cdot\int_0^{T_{A_i}}(free(t)_i+garbage(t)_i)\cdot dt \quad (16)$$

$$E_{trans_i}=k\cdot l\cdot\int_0^{t_{A\rightarrow Pd}}(alive(t)+free(t)+garbage(t))\cdot dt +k\cdot m\cdot\int_0^{t_{Pd\rightarrow A}}(alive(t)+free(t)+garbage(t))\cdot dt \quad (17)$$

If during the system life time the MBMA will be J times in waiting state the total extra costs E_{extra} compare to the idealistic architecture reference can be expressed by :

$$E_{extra}=\sum_{m=1}^{J}[E^m_{wait}]+\sum_{i=1}^{B}[E_{wasted_i}+E_{trans_i}] \quad (18)$$

where E^k_{wait} represents the costs due to the k^{th}waiting state, E_{wasted_i} and E_{trans_i} respectively the wasted energy and transition energy consumed in i^{th} bank during all active states. Thus :

$$Es_{total}=E_{extra}+k\cdot\sum_{i=1}^{B}[\int_0^{T_{A_i}} alive(t)_i\cdot dt] \quad (19)$$

The second interesting architecture reference to compare with is the architecture consisting of only one region memory to hold all dynamic allocations. This traditional architecture, which can also be seen has an one bank architecture, has the advantage to completely eliminate the waiting cost E_{wait} and E_{trans} but to the detriment of E_{wasted}. Indeed in that case the memory size needs also to match with the worst case allocation scenario and thus most likely drives a much greater $free(t)$ function compare to the one achievable with a MBMA.

VI. Optimization problem

In order to minimize Es_{total} we need to determine the MBMA configuration parameters influencing it. By MBMA configuration parameters we mean the size of the banks, a possible implementation of allocations prediction or bank need prediction feature(s), and allocation policies. The optimization goal is to reduce to the maximum the total static energy Es_{total} consumed by a MBMA for a specific application or a specific set of applications. From (19) we can derive E_{Active} which represents the $active\ energy$ dissipated by the MBMA, in other words the static energy that is spent only on memory blocks holding alive objects during active states. Thus E_{Active} corresponds to the minimum energy that any memory architecture will have to dissipate.

$$E_{Active}=k\cdot\sum_{i=1}^{B}[\int_0^{T_{A_i}} alive(t)_i\cdot dt] \quad (20)$$

Therefore optimizing Es_{total} comes to the problem of minimizing E_{extra} value. E_{extra} can be decomposed into the sum of three terms : E_{extra_A}, E_{extra_B} and E_{extra_C}. E_{extra_A} expresses the energy wasted during waiting states, E_{extra_B} represents the energy wasted in holding garbage objects and free memory space switched on and E_{extra_C} denotes the energy wasted during transition phases (from bank state powered-on into powered-off and vice versa) while the

system was in a working state. Fig. 4 illustrates the origin of E_{extra_C} cost components.

$$E_{extra_A} = \sum_{m=1}^{J}[E^m_{wait}]$$
$$= \sum_{m=1}^{J}[k \cdot \sum_{i=1}^{(n+1)_m}[\int_{t_{LastTrans_m}}^{t_{Active_m}} BankSize_i \cdot dt]] \quad (21)$$

$$E_{extra_B} = \sum_{i=1}^{B}[E_{wasted_i}]$$
$$= \sum_{i=1}^{B}[k \cdot \int_0^{T_{A_i}}(free(t)_i + garbage(t)_i) \cdot dt]$$

$$E_{extra_C} = \sum_{i=1}^{B}[E_{trans_i}]$$
$$= \sum_{i=1}^{B}[k \cdot l \cdot \int_0^{t_{A\to Pd}}(alive(t)+free(t)+garbage(t)) \cdot dt$$
$$+k \cdot m \cdot \int_0^{t_{Pd\to A}}(alive(t)+free(t)+garbage(t)) \cdot dt]$$

In (21) all variables labeled m refer to their respective value in m^{th} waiting state. E_{extra_A} is thus dependent on the number of waiting states that occurred during that execution. Determining the number of waiting states is not a trivial problem as it will depend on the configuration of the MBMA and the distribution of the allocation, garbage and deallocation events. The weight of E_{extra_A} inside Es_{total} is also dependent on the time needed for a bank to be switched from powered off to powered on state.

As for E_{extra_A}, E_{extra_C} weight inside Es_{total} is driven by the memory technology and more particularly by the time needed for a bank to be switched between powered off and powered on states. The more time is needed for the bank to be switched, the more predominant E_{extra_C} will be inside Es_{total}.

VII. Optimization parameters

In this section we go through parameters influencing the optimization problem introduced in above section.

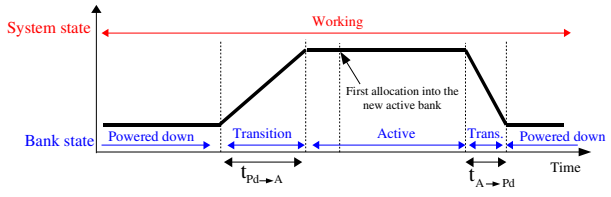

Fig. 4. E_{extra_C} components

a) Bank size: The bank size used for a MBMA has an impact on all three optimization subproblems. Increasing the banks size will increase the function $free(t)$ and thus obviously will increase E_{extra_B}. For E_{extra_A} and E_{extra_C} the increase of $free(t)$ has to be balanced by the fact that bigger banks will most likely reduce the number of waiting and transition states thus will reduce n in E_{extra_A}, l and m in E_{extra_C}. The ratio between the energy increase due to $free(t)$ and the energy decrease due to n, l and m will depend on the distribution of the allocation, garbage and deallocation events.

b) Deallocation scheme or garbage collector: Frequent garbage collections (GC) or explicit deallocation will decrease the function $garbage(t)$. Moreover if an optimum deallocation scheme is able to deallocate objects right after their last use, it would be theoretically possible to reduce function $garbage(t)$ to the constant null. But a possible decrease in $garbage(t)$ generates an identical increase of $free(t)$. In this way a better deallocation scheme reduces functions E_{extra_A} and E_{extra_C} as it reduces the number of waiting and transition states. As the possible $garbage(t)$ decrease will be identical to the $free(t)$ increase it will not affect E_{extra_B} value. In addition we also have to remember that frequent GC increases the application running time and thus tends to increase E_{extra_B} with the increase of T_{A_i}.

c) Allocation or bank need prediction: Allocation prediction or bank need prediction feature(s) intents to switch on banks beforehand in order to avoid waiting states. As a result it decreases the number of waiting states and intents to decrease E_{extra_A}. But at the same time it also drives an increase of $free(t)$ and thus an increase of E_{extra_B} and also possibly E_{extra_C}. The ratio between the energy increase due to $free(t)$ evolution and the energy decrease due to the number of waiting states decrease depends on how long beforehand banks are switched on. The extreme case would be to switch on all banks beforehand, eliminating thus E_{extra_A}, but then maximizing $free(t)$.

d) Allocation policies: Our strong feeling concerning the allocation policies is that if the policies group in a same bank objects with a similar life time, it decreases the number of waiting and transition states. Regrouping similar life time objects into particular banks is expected to increase the overall number of banks switched on and thus limiting the need for new banks. As a consequence such policy will reduce the number of waiting and transition states while $free(t)$ will increase. E_{extra_B} will then surely increase as E_{extra_A} and E_{extra_C} evolution will depend on the distribution of the allocation, garbage and deallocation events.

e) Memory fragmentation: represents free memory areas that might be unuseable for future dynamic allocations due to their relative small sizes. If these free memory areas distributed over each bank turn out to be unuseable for future allocations, they will waste during all the system runtime a static energy proportionally to their sizes. Hence fragmentation plays a role in the MBMA static energy cost as higher fragmentation lead to a potential increase of function $free(t)_i$ in E_{extra_b} and J in E_{extra_a}.

VIII. MBMA BEHAVIOR SIMULATION

The MBMA behavior was simulate on 2 different applications: a) Tobi-Tris a tetris like game written in Java, b) $Cfrac$ [5] written in C. $Cfrac$ is a allocation intensive application factoring large integers using the continued fraction method. The application input was 6 successive integers, from 21 to 37 digits, fed to the application with a 10 to 35 seconds interval.

For the Tobi-Tris Java application we used the SUN J2ME Wireless Toolkit [6] and its MIDP device emulator to capture the $allocation$, $deallocations$ and $garbage$ events. The Java application is run twice, a first run is done on the emulator compiled with default options and used to retrieve the $allocation$ and $deallocation$ events. A second run is done with the emulator compiled with options tuned to launch a garbadge collection (GC) at each bytecode execution. From this second run we are able to retrieve the $garbage$ events. The captured events $deallocation$ reflect the GC actions of the emulator's Java Virtual Machine (JVM) in the context of one memory region. We acknowledge that appropriate policies ruling the GC launch might be different for a MBMA than for a one memory region. However we want to constrain the optimization problem and mainly look first at bank size influence.

To analyze the energy behavior of an MBMA for Cfrac application we implemented a customized memory allocator. It features a first fit algorithm with one address-ordered free list per bank. When an object is allocated it scans all free lists until it finds the first free space that can hold the new object. If no free space is available, a new bank is switched on. When a object is deallocated its corresponding memory block is inserted into the respective bank free list. If there is adjacent free blocks they are coalesced in one free block. If after a $deallocation$ event one bank is left over empty the bank is switched off. The allocator is able to trace all allocations, deallocations and bank state transitions in order to compute afterward the energy cost of the MBMA.

For all MBMA behavior simulations, no allocation nor bank need prediction is used. The biggest allocated object fixes the smallest possible bank size, and the maximum number of powered on banks is used as reference for computing $E_{oneBank}$, the static energy cost if the MBMA would had only one bank. Numerical value for constant k is deducted from the characteristics of the low power $\mu PD431000A$ SRAM [7]. $t_{Pd\rightarrow A}$ and $t_{A\rightarrow Pd}$ are over evaluated at 10 ms, twice the $\mu PD431000A$ operation recovery time from standby mode.

TABLE II

CFRAC - BANK SIZE AND ENERGY COSTS IN JOULE

Bank Size (Kb)	E_{Active}	E_{extra_A}	E_{extra_B}	E_{extra_C}
2	36.03	1.61e-4	32.08	1.35e-6 %
4	36.03	7.37e-5	38.35	1.03e-6 %
8	36.03	3.40e-5	42.38	6.87e-7 %

TABLE III

CFRAC - BANK SIZE AND ENERGY COSTS COMPARISON IN JOULE

Bank Size (Kb)	**Total**	$E_{oneBank}$	**Saving**
2	68,11	85,88	20 %
4	74.38	85,88	13 %
8	78.41	85,88	8.5 %

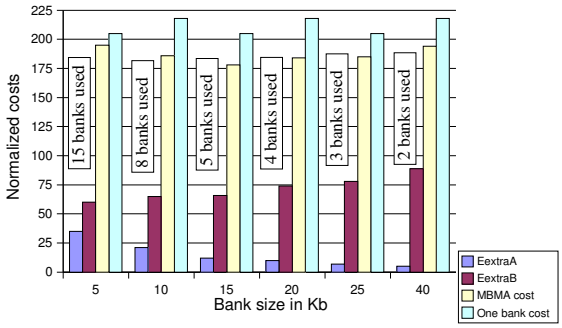

Fig. 5. Tobi-Tris - Bank size and and energy costs

Fig. 5 shows the results for the Tobi-Tris game where all values are normalized to E_{Active}, 100 representing E_{Active} value. On Fig. 5 clearly appears the tradeoff between E_{extra_A} and E_{extra_B} to get the optimum bank size. For this application small bank size is cost unefficient due to E_{extra_A} and big bank size is cost unefficient due to E_{extra_B}. A tradeoff has to be found in between and Fig. 5 shows that the most cost efficient bank size is 15Kb. For Tobi-Tris application, with a 15Kb bank size the MBMA is using 5 banks and consumes about 175% of E_{Active}, while a 75Kb single bank architecture will consume about twice the E_{Active} value.

Table IV shows statistics on Cfrac execution for 2Kb banks. Tables II and III present the saving on static energy consumption for 3 bank sizes. With this application we note that E_{extra_A} and E_{extra_C} are relatively small compare to E_{extra_B}. This is due to the relatively small period spend in switching on and off time compare to the application run time. Each time a bank was switched on the fragmentation was compute for all already powered on banks. This gave us an average fragmentation of 0,27 with a standard deviation of 0,31. But it is important to also say that each time a new banks is switched on, on average the already switched on banks occupation rate is 99%. This clearly indicates that for this example fragmentation is causing insignificant degradation on the MBMA costs. The average occupation rate denotes the ratio over the time between the bank size and the allocated objects size in the bank. Figures from Table V indicates that banks occupation rate have a bigger impact than fragmentation. On average with a 2kb bank size only 52,91% of available memory is allocated. This is mainly due to left alone objects spread over several banks, preventing banks to be switched off.

Those results show that a substantial saving can be achieve on static memory energy consumption through MBMA with-

out any application optimization nor customized allocation policies. However it also indicates that further savings could be obtained, mainly on E_{extra_B}.

IX. Related work

Several researches have been done on data transformation or migration and memory access scheduling to exploit MBMA [8][9][10]. In contrast to these, this paper doesn't explore the possibilities to adapted the running application(s) on the system but on the contrary how to set up an optimum MBMA for a specific application. Nevertheless we believe that after the optimal memory architecture has been set further cost reductions can be achieve through application optimization. L. Benini et al. propose an algorithm for automatic scratch-pad RAMs partitioning from several application execution profiles in [11]. In this work the scratch-pad RAMs doesn't have the possibility to be switched off and on. K. Flautner et al. present in [12] a method using dynamic voltage scaling (DVS) for putting cache lines in a low-power mode, called drowsy, where data are preserved. In [12] only cache memories are addressed, while our work addreses only static energy saving on the main memory.

In [13] G. Chen et al. describe the impact of GC, compaction and bank size on an embedded Java environments with MBMA. Our work differs from [13] in that we express the optimization problem and explicitly describe the optimization parameters influencing the system, providing in this manner a total understanding on the relations between energy cost evolution and optimization parameters.

X. Conclusion and Future work

In this paper we proposed a MBMA as solution to decrease the static energy cost in memory and set up the equations ruling the optimization problem. We showed that implementing a MBMA and choosing appropriate bank size can lead to a 20% static energy saving without any software optimization, nor bank need prediction, nor dedicated allocation policies. We also observed that the banks occupation rate plays a predominant role in the MBMA static energy cost.

Future work for this study includes more simulations with deeper optimization parameter analyzes, especially on possible bank need prediction algorithms and allocation policies. We could also imagine to implement the MBMA management process within the memory as it is done for intelligent memory manager [14], leading to a probable performance improvement for cache memory architecture. Moreover, we could also investigate the possibility to compact the allocated object within all banks in order to increase the banks average occupation rate. This might be impossible to implement for conventional programming languages, such as C, Pascal, Ada, etc., but would probably better fit with object oriented language like Java. An other approach would be the use of region allocation mechanism based on objects life time. Furthermore, in the case of a multi applications platform, it is also worth exploring the solution of having several sets of different bank sizes.

TABLE IV

Cfrac - Statistics on Cfrac execution for 2Kb bank size

Total Number of Allocations: 59165
Total Number of Deallocations: 58596
Biggest allocated object in bytes: 1244
Maximum live objects size: 190041 bytes
162 times a bank has been switched on
71 times a bank has been switched off
Maximum numbers of powered on bank: 94
$\sum t_{A \rightarrow Pd}$ in mSec: 1620

TABLE V

Cfrac - Statistics on the banks occupation rate for 2Kb bank size

Average occupation rate:	52,91 %
Maximal average occupation rate:	81,69 % $(third\ bank)$
Minimal average occupation rate:	0,15 % $(94^{th}\ bank)$
Standard deviation:	23,73

References

[1] F. Catthoor, E. de Greef, and S. Suytack, *Custom Memory Management Methodology: Exploration of Memory Organisation for Embedded Multimedia System Design*. Kluwer Academic Publishers, 1998.

[2] J. A. Butts and G. S. Sohi, "A static power model for architects," in *MICRO 33: Proceedings of the 33rd annual ACM/IEEE international symposium on Microarchitecture*, 2000.

[3] M. S. Johnstone and P. R. Wilson, "The memory fragmentation problem: solved?" *ACM SIGPLAN Notices*, vol. 34, no. 3, pp. 26–36, 1999. [Online]. Available: citeseer.ist.psu.edu/johnstone97memory.html

[4] M. et al., "Leakage power estimation in srams," UC Irvine, CECS Technical report TR 03-32, Oct. 2003.

[5] D. Detlefs, A. Dosser, and B. Zorn, "Memory allocation costs in large c and c++ programs," *Software-Pratcice and Experience*, vol. 24(6), pp. 527–542, 1994.

[6] T. S. J. W. Toolkit, "http://java.sun.com/products/sjwtoolkit."

[7] P. S. data sheet, "http://www.necel.com/memory/."

[8] M. Kandemir, "Impact of data transformation on memory bank locality," in *DATE'04*.

[9] C.-G. Lyuh and T. Kim, "Memory access scheduling and binding considering energy minimization in multi-bank memory systems," in *DAC 2004*.

[10] V. D. L. Luz, M. Kandemir, and I. Kolcu, "Automatic data migration for reducing energy consumption in multi-bank memory systems," in *DAC*, 2002.

[11] L. Benini, A. Macii, and M. Poncino, "A recursive algorithm for low-power memory partitioning," in *ISLPED*, 2000.

[12] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. N. Mudge, "Drowsy caches: Simple techniques for reducing leakage power." in *ISCA*, 2002, pp. 148–157.

[13] G. Chen, R. Shetty, M. T. Kandemir, N. Vijaykrishnan, M. J. Irwin, and M. Wolczko, "Tuning garbage collection for reducing memory system energy in an embedded java environment." *ACM Trans. Embedded Comput. Syst.*, vol. 1, no. 1, pp. 27–55, 2002.

[14] M. Rezaei and K. M. Kavi, "Intelligent memory manager: Reducing cache pollution due to memory management functions," *Journal of Systems Architecture*, vol. Volume 52, January 2006, 41-55.