Handheld System Energy Reduction by OS-Driven Refresh

Vasily G. Moshnyaga[1], Hoa Vo[2], Glenn Reinman[2], and Miodrag Potkonjak[2]

[1] Department of Electronics Engineering and Computer Science, Fukuoka University
8-19-1, Nanakuma, Jonan-ku, Fukuoka 814-0180, Japan
[2] Computer Science Department, University of California, Los Angeles,
3532 Boelter Hall, Los Angeles, CA 90095-1596

Abstract. Emerging portable devices relay on DRAM/flash memory system to satisfy requirements on fast and large data storage and low-energy consumption. This paper presents a novel approach to reduce energy of memory system, which unlike others, lowers energy of refresh operation in DRAM. The approach is based on two key ideas: (1) DRAM-based flash cache that keeps dirty pages to reduce the number of accesses to flash memory; and (2) OS-controlled page allocation/aging to stop the refresh operations in banks, whose pages are clean and not accessed for a long time. Simulations show that by using this technique we can decrease the overall energy consumption of DRAM/flash memory on video applications by 8-26% while reducing the DRAM refresh energy by 59-74%.

1 Introduction

1.1 Motivation

Modern battery-operated handheld devices, such as mobile phones, incorporate 128Mb-512Mb SDRAM as main storage and 256MB-1GB solid-state flash memory as non-volatile secondary storage to satisfy performance and memory demands of data-intensive multimedia applications. In 2.5-3G cell phones [1], flash (mainly NAND) memory stores OS, application programs, and data. During boot time, the OS and application programs are copied from the flash memory to the main memory in a process referred to as "memory shadowing". To reduce both the program download delay and the DRAM size, recent systems employ "demand paging" which swaps pages of code/data between memories according to processor's requests. This memory organization leads to a smaller DRAM, less loading time, but requires memory management unit that ensures both high performance and low energy page swapping.

In cell-phones, energy is consumed during active operation as well as on idling, i.e. periods of inactivity. The memory system takes almost 30% of total device power during program execution [2], and more than half of total energy consumed by phone over a day [3] with almost 20% of the energy spent for data retention. Clearly, reducing of energy consumed by memory can significantly extend mobile phone battery life.

J. Vounckx, N. Azemard, and P. Maurine (Eds.): PATMOS 2006, LNCS 4148, pp. 24 – 35, 2006.

The main goal of this paper is to develop a memory management technique capable of lowering energy consumed not only by data accesses, but also by DRAM refresh and data retention.

1.2 Background

(a) Flash Memory: Flash memory is a non-volatile device. It has higher storage density than DRAM. However, its content is not randomly accessed. Usually, a NAND flash contains a fixed number of blocks each of which consists of 16-64 pages. A page normally includes 512B of main data and 16B of spare data. So, a typical 32MByte NAND flash has 4K blocks of 16 pages each [4].

Using flash memory has two main limitations. The first one is a potential durability problem. The flash memory cells have a limited number of write/erase accesses for which performance is guaranteed. After about 10,000 cycles, subsequent accesses begin to take longer. After 100,000 write/erase cycles, flash cells begin fail and become unusable. The second limitation of flash memory is the need to erase data before it can be overwritten. The flash memory manufacturer determines how much memory is erased in a single operation. Usually, erase is performed on a *block* basis, while read and write are conducted based on a *page* basis.

There are three important aspects to erasure: *flash cleaning*, *performance* and *power*. When the size of data block is larger than transfer unit, any block data that are still needed must be copied elsewhere. Cleaning flash memory is thus analogous to segment cleaning in a log-structured file system. The cost and frequency of segment cleaning is related in part to cost of erasure and in part to segment size. The larger the segment, the more data will likely to be moved before erasure can take place.

The second aspect to erasure is performance. In NAND flash memory, the time to erase and write a page is 8 times longer than the time of read (see Table 1) [4]. Because the erasure time is independent of the amount of data being erased, the cost of erasure is amortized over large erasure units. To avoid delaying writes for erasure it is important to keep a pool of erased memory available.

The third aspect to erasure is power. Erasing a page in NAND memory consumes as twice as more power than reading the page. Since a write with erase takes almost 10 times more power than a read, page swapping policy must minimize the number of flash memory writes, even though it might incur additional read operations. One such policy is Clean-First-LRU [5], which swaps clean (i.e. unmodified pages first) while keeping dirty pages in the DRAM as long as possible. If there are no clean pages in a predefined time window, a standard LRU is used. To further reduce the number of flush memory writes, the Clean-first-LRU can be combined with selective compression and caching [6]. Because of compression and decompression overhead (248us and 216us, respectively), the compression is applied only to those pages, whose compression ratio exceeds a predefined threshold. The other pages are stored in un-compressed form.

(b) DRAM: Dynamic Random Access Memory (DRAM) contains a number of components: cell array, decoders, sense amplifiers and controller. Each DRAM cell consists of one transistor and one capacitor. A write operation of a DRAM cell is performed by charging the capacitor via an on-state cell transistor, while the cell transistor is in an off-state during the charge retention period. DRAM cell retention time is limited by charge leakage from the capacitor through an off-state transistor channel and/or

Table 1. Energy and delay parameters of 512b NAND flash (per 4KB access) [4]

Operation	Energy(μJ)	Time(ns)
Read	9.44	1180
Write	76.08	7160
Erase	16.49	1950

a p-n junction. Also, the process of retrieving, or reading, data from the memory array tends to drain these charges, so the memory cells must be precharged before reading the data. Therefore, a periodical re-write operation, called a *refresh* operation, is necessary in DRAM.

Normally, the refresh operation in modern DDR SDRAMs is initiated by the system's memory controller, but some chips are designed for "self refresh." This means that the DRAM chip has its own refresh circuitry and does not require intervention from the CPU or external refresh circuitry. Self refresh dramatically reduces power consumption and is often used in portable computers. Conventional DRAMs can refresh either one bank at a time or all banks simultaneously within a fixed time interval. The refresh period usually is larger than the DRAM access time; so many read/write accesses are stalled while refreshing. Since the DRAM power is proportional to the number of accesses and the number of refreshes, optimizing the number of refreshes is necessary.

Furthermore, to reduce power consumption, modern DRAMs incorporate a number of power saving modes, such as active (or read/write), idle (or standby) and self-refresh [7]. In the *active mode,* normal operation is activated. The *idle mode* moves memory to a power down mode, reactivating it only when a refresh is required. Once a refresh is issued, the memory is returned to power-down mode again. In the *self-refresh mode* the controller does not actively issue any commands. In this mode, additional logic can be used to control partial array refresh for additional power savings; the clock to memory is gated off. Obviously, to service a request the DRAM must be in the active mode, because the low-power modes increase time to transition back to active (see Table 2).

1.3 Related Reseach

A number of architectural approaches have been proposed to increase energy efficiency of DRAM. Most studies use control algorithms that dynamically transition DRAM devices (or banks) to low power modes after they are idle for a certain threshold period of time. Lebeck, et al [8] studied DRAM power state transition policies in conjunction with software page placement policies. To improve transition decisions, they control the page allocation by working set locality. Fan, et al [9] further explored policies for manipulating DRAM power states in cache-based systems. They stated that an immediate transition of idling DRAM chip to a lower power state might work better than a more sophisticated policy that tries to predict idling time. Delauz, et al [10] suggested various threshold predictors to determine idling time after which the DRAM should transition to a low-power state. However, predicting the transitions correctly is not easy even with different thresholds because the time of DRAM idling varies with power states and applications. Due to lack of long idling, deep power-saving states are rarely explored; so the efficiency of power management is low. To

Table 2. Characteristics of SDRAM@66MHz (4 word burst access)[8]

Operation	Energy(nJ)	Time ns)
Read	70.2	120
Write	51.6	120
Refresh op	99.2nJper refresh	-
Active mode	100 mW	-
Idle mode	45.1 mW	-
Self-refresh mode	1.8 mW	-
Idle→ Act	15 mW	+6 ns
SelfRefresh→Act	160 mW	+6000 ns

prolong idling, Huang et al [11] advocated reshaping input traffic to DRAM by making memory accesses less random and thus more controllable. To save DRAM refresh energy, Ohsawa, et al. [12] used two schemes: a selective refresh with data allocation optimization and a variable period refreshing. The *selective* refresh scheme adds a valid bit to each memory row and only refreshes rows with valid bit set. The *variable* refresh scheme allocates a refresh counter to each row. When the number of cycles between the previous refresh exceeds the pre-defined threshold, the line is refreshed. As reported in [12], the selective refresh saves 5%-60% of energy while the variable refresh can save up to 75%. Hwang, et al. [13] proposed to apply array self-refresh operation partially, i.e. on a portion (e.g.1/2, 1/4, 1/8) of one or more selected memory banks. The operation is performed by (1) controlling the generation of row addresses during self-refresh operation, and (2) controlling a self-refresh cycle generating circuitry. Reduction in self-refresh current is achieved by blocking the activation of a non-used block in a memory bank. The partial array self-refresh is already supported by Mobile DRAM [14], Cellular RAM, etc. The main problem is how to distinguish the unused blocks in array? Since the currently unused blocks might be used in the future, the prediction policy is non-trivial.

1.4 Contribution

In this paper we propose an OS-based approach to reduce energy consumption of DRAM/flash memory system. Unlike related methods, the approach utilizes history of page accesses to improve energy efficiency of DRAM refresh.

This paper is organized as follows. Section 2 presents our approach. Section 3 shows the experimental results. Section 4 summarizes our findings and outlines future work.

2 The Proposed Approach

2.1 Main Idea

The approach we propose is based on observation that as DRAM memory size increases, more and more memory becomes unused at any given time. Because unused memory does not need to be refreshed, we can save energy by intelligently controlling which pages get refreshed. The system OS knows which pages are used and unused, so given the opportunity it could disable refresh on selected pages.

The main idea of our approach is simple and consists in disabling from refresh operations all individual banks which have not been referenced in given time-window *and* have no dirty (or modified) pages. If a non-referenced bank has dirty pages, we move the pages to the swap cache (see Fig.1) to keep them in DRAM as long as possible and thus minimize the number of writes to the flash storage. The swapping takes place either when the cache becomes full (in this case the LRU dirty page is moved from the cache to flash), or when the requested page is not in DRAM (in this case, the page is loaded from the flash memory to active DRAM bank).

In our approach, we exploit the fact that the OS not only has physical page allocation information for each executing process but also has information of pages that are actually being referenced (by sampling the reference bits in the page table and TLB). By compacting physical pages into minimum number of memory banks (using page coloring algorithm), we potentially eliminate refresh for entire DRAM banks in which there are no dirty pages. Modern memory systems swap out pages when the memory space is full. In our refresh-oriented page allocation, the OS starts swapping out pages when writing a page to the flash memory becomes less energy consuming than keeping the page refreshed in DRAM.

2.2 Assumptions

We take the following assumptions:

1. Each DRAM bank can be in two modes: refresh and non-refresh (i.e. power down). The refresh mode can be further partitioned into active, nap, and standby, as it done in conventional DRAM, however, we do not address this issue for the simplicity of explanation.
2. The banks are controlled separately, so each bank can be refreshed (if it in the refresh mode) or shut down (i.e. non-refreshed) independently of the others.
3. Each page can be either active (i.e. open) or closed. Any access to a close page makes it active. Refresh banks may have as open as close pages, but non-refresh banks have no active pages (all their pages are closed).
4. As in [5], the higher order banks in DRAM are allocated for flash cache to minimize the number of writes to the flash memory. Dirty pages are dropped into the cache banks unless these cache banks are full.

2.3 Algorithm

The proposed refresh-oriented page allocation scheme implements the following algorithm. After a given period of time, *t1*, it detects pages, which have not been accessed and closes them. Next, after a time, *t2*, the algorithm checks status of refreshed banks. If all pages in a bank are closed, the bank is put into a non-refresh mode, while dirty pages of this bank are moved to the flash cache. Finally, after a time, *t3*, the algorithm determines the least-recently used page in the swap-cache and moves it out onto the flash-memory. After dropping the content to flash memory, the cache page is considered empty, and hence can be used to store other dirty pages. If a requested page resides in the flash and DRAM is full, the algorithm applies “clean-page-first” [5] policy to allocate the DRAM page to be swapped with the requested page. If there are no clean pages in DRAM, the algorithm moves the LRU dirty page from DRAM to the swap-cache. The code below shows the algorithm in details.

Algorithm:

```
  Initialization: all banks are in refresh mode, the last bank is the
                  flash cache; all pages in DRAM are inactive;
  At t = t2,
     find minimum number of memory banks that allocate all active pages;
     move other banks to non-refresh mode;
for each access to a page AP
  if page is in DRAM, access AP directly;
  else
  { find an inactive page (RP) in (non-cache) DRAM refresh-banks;
    if no inactive pages
    then
     if there is a non-refresh bank;
     then move it to refresh mode;
     let RP be the first page in this bank;
     else
     { find CFLRU page (CFP),
       if CFP is dirty,
       then move CFP to cache
            load demand page to RP(or CFP) page
     }
   }
do page & bank aging
 end

  move page to cache:
    find an empty page (EP) in cache;
    if EP is not found
       find a non-refresh bank (NRB) in DRAM;
       if NRB exists, then move NRB to cache;
            let EP be first page in this bank;
       Else
         {find LRU cache page (LP) ;
           store content of LP to the flash memory;
           empty page LP;
         }
  Drop page to empty page EP (or LP)

  page & bank aging:
   for each cache bank do
    {for each cache page do
       { if page is not accessed in t3, drop page to flash, empty page;
         else if a cache bank has only empty pages
             then move the bank to the non-refresh mode
       }
    }
  for each refresh non-cache bank do
    { for each refresh page (RP) do
        { if RP is not accessed in t1
           then if RP is dirty, then move RP to cache;


                move RP to the inactive mode;
     if bank has no active pages, move it to the non-refresh mode
       }
     }
```

3 Experimental Evaluation

3.1 Energy Modeling

The energy consumed by memory system is modeled by the sum of energies consumed by DRAM and flash memory:

$$E_{total} = E_{DRAM} + E_{flash.} \quad (1)$$

The energy consumed by each DRAM bank is directly proportional to the number of reads (N_{read}) and writes (N_{write}) and the unit access energy per read (E_{read}) and write (E_{write}), respectively. Further, SDRAM consumes idle power (P_{idle}) and refresh power ($P_{refresh}$) power during program execution. If there is no memory access, the memory stays in the power down state consuming only retention power ($P_{retention}$). Thus, assuming that SDRAM consists of N banks, the energy consumed by DRAM can be calculated by,

$$E_{DRAM} = E_{read}*N_{read} + E_{write}*N_{write} + t_{active}*(P_{idle}+P_{refresh}) + t_{inactive}*P_{retention}\}. \quad (2)$$

Similarly, the energy consumption of flash memory is modeled as,

$$E_{flash} = E_{fread}*N_{fread} + (E_{fwrite} + E_{erase})*N_{fwrite}. \quad (3)$$

where, $\{E_{fread}$ and $E_{fwrite}\}$ are values of energy consumed by flash per read, write and erase operation, respectively, $\{N_{fread,}\ N_{fwrite}\}$are the number of flash reads and writes, respectively.

3.2 Experimental Setup

To collect data we augmented the SimpleScalar simulator [17] with our DRAM simulation program. The SimpleScalar simulated a 400MHz 32-bit RISC processor (similar to StrongARM-110 [18]), with 32 set-associative caches (16KB inst. and 16KB data), 32B cache block size, 1 clock cycle cache hit and 3 clock-cycle cache miss, 5 clock-cycle instruction miss-prediction penalty.

Four DRAM sizes (4, 8, 16, 32, 64, and 128) MB, respectively, have been tested. The energy parameters of DRAM and flash memory are given in Tables 1-2. The values of DRAM refresh power and retention power utilized in the experiments were 7mW and 1.8 mW, respectively [4]. We assumed that DRAM has 8 banks, page has 4KB, and refresh is performed every 15.6μsec per row.

Table 3. Benchmarks and descriptions

Program	Description	Dataset	Symbol	Inst.($x10^6$)
Mpeg_dec	A Mpeg2 video decoding	*tennis*	Mc	62
Mpeg_enc	A Mpeg2 video encoding	*tennis*	Md	667
Jpeg_com	JPEG image compression	*testimg*	Jc	577
Jpeg_dec	JPEG image decompression	*testimg*	Jd	48
gcc	C code compiler	*gcc*	gcc	1,497

We assumed that DRAM has 8 banks, page has 4KB, and refresh is performed every 12.6μsec. The data retention energy is 1.8mJ/sec.

Five benchmark programs (Table 3) have been used in the experiment: *gcc* from the SPEC2000 suite and the rest from MediaBench [19]. To model user interactions, we ran each video program 10 times with a 30 second-gap between the runs. The input video contained 100 frames and no delay between the frames. Each program was run to completion. The results have been measured in terms of the total energy consumed by the memory system, the DRAM refresh energy and the total execution time. The energy consumption of L1benchmark programs have been used in experiment: *gcc* from the SPEC2000 suite and *Mpeg2decode*, *Mpeg2encode* programs from MediaBench [19].

To model user interactions, we have run each video program 10 times with a 30 second-gap between the runs. The input video contained 100 frames and no delay between the frames. Each program was run to completion. The results have been measured in terms of the total energy consumed by the memory system, the DRAM refresh energy and the total execution time. The energy consumption of L1 (D- and I-) caches and the energy of MMU have not been considered. Also, it was assumed that OS consume 16MB and this amount of memory was not available to application programs. Therefore, the memory size the applications could freely use was limited to 16MB, unless otherwise explicitly stated.

3.3 Results

Figure 1 shows the breakdown in energy consumption and the execution time achieved by the proposed approach on *gcc* benchmark and normalized to conventional method [5]. Based on the results obtained at fixed *t2*, *t3* and variable *t1* (see Fig. 1,a), we conclude that a small *t1* increases both the energy and the delay due to very frequent page closing/opening and mode changing. Also, due to small size of DRAM, the amount of page swapping between DRAM and flash is large, so the flash access energy is high. As *t1* increases, both the number of page mode changes and page swapping decreasing; so the total energy also goes down. According to the results, the best value of t1 ranges between 1625ms and 3200ms. The rightmost point represents the case when *t1=t3*. As we fix *t1* and *t3* at 3250ms, and vary *t2*, we see that the smaller *t2*, the better (see Fig.1, b). At 0.0125ms, for example, the refresh energy can be as much as 10%. Finally, as we expected, small *t3* leads to fast page aging which extra page swapping and so increase of both DRAM access energy and flash energy (see Fig.1,c). For t3 larger than 375ms the figures do not change.

Figure 1(d) shows the impact of DRAM size on energy and execution time. During execution, the *gcc* program accesses 2196 different pages, which require little more than 8MB of DRAM. When the DRAM size is small, the refresh energy reduction achieved by our approach is diminished by energy consumed on page swapping. Therefore, the energy savings are small when memory is 4MB and 8MB. As memory grows, more energy can be saved. At 128MB DRAM, for example, the proposed technique can save up to 55% of the total energy without affecting the execution time.

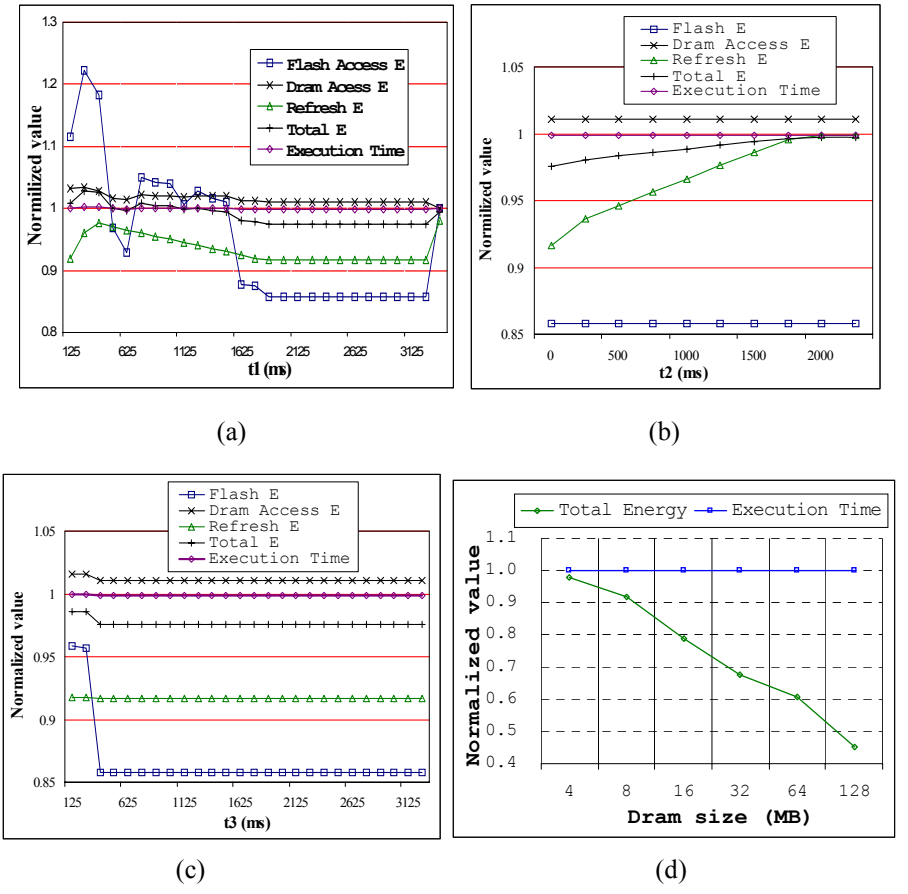

Fig. 1. (a-c) dependence of the results on t1, t2, and t3, respectively;(d) dependence of the results on the DRAM size

Figures 2, 3 show the performance of proposed technique in perspective to the related (conventional) technique [5] for 8- and 16-MB DRAM. In this figures, bars marked by *C* show results of [5]; bars marked by *P* depict results of the proposed technique. Symbols in parentheses denote the tested programs. We observe that the proposed technique lowers the refresh energy, while leaving the other energy components almost unchanged. Due to large variation in the number of pages accessed in DRAM (114 for *Mc*, 53 for *Md*, and 286 for *Jc*, 157 for *Jd*, and 2196 for *gcc*), number accesses (3535730 for *Mc*, 1374440 for *Md*, 16546151 for *Jc*, 1377033 for *Jd*, and 215594 for *gcc*), the results along the benchmarks as well memory size. We see that the proposed technique over performs the conventional method on all the benchmarks.

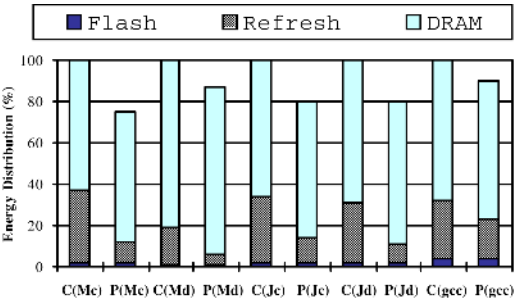

Fig. 2. Results obtained for 8MB DRAM

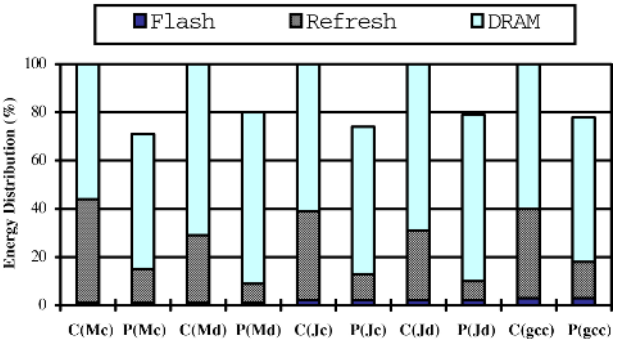

Fig. 3. Results obtained for 16MB DRAM

Table 4 summarizes the results in terms of energy reduction achieved by the proposed approach. The larger the memory size, the larger reduction ratio. For 16MB DRAM, for example, our approach reduces the DRAM refresh energy by 59-74% while lowering the total energy consumed by the tested applications by 8-26%.

Table 4. Energy reduction ratio observed for benchmarks

Benchmarks	Jc		Jd		Mc		Md		*gcc*	
DRAM size (MB)	8	16	8	16	8	16	8	16	8	16
Refresh energy (%)	62	71	64	72	68	70	72	74	32	59
Total energy (%)	20	26	21	23	24	28	14	20	10	21

4 Conclusion

In this paper we proposed a refresh-driven page allocation technique to lower the energy consumption of memory system in handheld devices. According to experiments,

the proposed technique can decrease the total energy consumption of memory system significantly (by 14-26%) on standard image and video processing applications without affecting the execution time. In this preliminary work, we have not considered the energy overhead of busses as well as the energy consumption of OS and the memory management unit. Also, the investigation has been restricted to a small set of benchmarks which only lightly represent real handheld applications. To evaluate the approach on tasks such as internet browsing, word processing, MS PowerPoint, Adobe Acrobat Reader, etc., we need to perform an extensive profiling of the applications. This work will be conducted in the near future.

References

[1] Weldon, T., Memory subsystems for 2.5G cellular handsets, Micron Techn. Inc., Jedex, San Jose, 2004 http://www.micron.com/products/dram/ddr2sdram/presentation.html
[2] Vargas, O., Minimum power consumption in mobile phone memory systems, Portable Design, 2006.
[3] Lee, H.G, and Chang, N., Low-energy heterogeneous non-volatile memory systems for mobile systems, J. of Low-Power Electronics, Vol.1, no.1, pp.52-62, 2005.
[4] Samsung Electronics. NAND flash memory & SmartMedia data book, 2002.
[5] Park, C., Kang,J.U., Park, S.,Y., Kim, J.S., Energy aware demand paging on NAND flash-based embedded systems, Proc. ACM/IEEE Int. Symp. Low-Power Electronics and Design, pp.338-343.
[6] Park, S., Lim, H., Chang, H., Sung, W., Compressed swapping or NAND flash memory based embedded systems, Proc. the IEEE Workshop on Signal Processing Systems (SiPS), 2003.
[7] Samsung Electronics, 128Mb DDR SDRAM Specification, Version 1.0, Rev.1.0, Nov.2, 2000.
[8] Lebeck, A.R., Fan, X., Zeng, H., Ellis, C., Power-aware page allocation, Proc. 9th Int. Conf. on Architectural Support for Programming Languages and Operation System (ASPLOS IX), Nov. 2000
[9] Fan, X., Zeng, H., Ellis, C., Lebeck, A.R., Memory controller policies for DRAM power management, Proc. ACM/IEEE Int. Symp. Low-Power Electronics and Design, 2001.
[10] Delauz, V., et al., Scheduler-based DRAM energy power management, 39th ACM/IEEE DAC, pp.697-702, 2002.
[11] Huang, H., Shin K., Lefurgy, C. Keller, T., Improving Energy efficiency by making DRAM less randomly accessed, Proc. ACM/IEEE Int. Symp. Low-Power Electronics and Design, 2005.
[12] Ohsawa, T., Kai, K., Murakami, K., Optimizing the DRAM refresh count for merged DRAM/logic LSIs, Proc. ACM/IEEE Int. Symp. Low-Power Electronics and Design, 1998, pp. 82-87.
[13] Hwang, H-R., Choi, J-H, Jang, H-S., Sysem and method for performing partial array self-refresh operation in a semi-conductor memory device, US Patent, no.20050041506, 02/24/2005.
[14] Takahashi M., et al., A 60-MHz 240-mW MPEG-4 videophone LSI with 16-Mb embedded DRAM. IEEE J. Solid-State Circuits, vol.35, no.11, pp.1713-1721, 2000
[15] Mobile DRAM: The secret to longer life, MicronTechn. Inc, http://download. micron. com/pdf/flyers/mobile_sdram_flyer.pdf

[16] Huang, H., Pillai, P., and Shin, K.G., Design and implementation of power-aware virtual memory, Proceedings of the 2003 USENIX Annual Technical Conf., June 2003.
[17] Burger D., Austin, T., Bennet, S., Evaluating future microprocessors- the superscalar tool set. Technical Report 1306, Univ. of Wisconsin-Madison, CSD, July 1996
[18] SA-110 Microprocessor, Technical Reference Manual, Intel Corporation, Dec.2000.
[19] Lee, C., Potkonjak, M., and Mangione-Smith, W-H., MediaBench: a tool for evaluating and synthesizing multimedia and communication systems, Proc. the IEEE Int. Symp. on Microarchitecture, 1997.