Performance Evaluation of Parallel Applications on Next Generation Memory Architecture with Power-Aware Paging Method

Yuto Hosogaya[†,*], Toshio Endo[†,*], and Satoshi Matsuoka[†,‡,*]
[†] Tokyo Institute of Technology
[‡] National Institute of Informatics
[*] JST, CREST
{hosogaya, endo}@matsulab.is.titech.ac.jp, matsu@is.titech.ac.jp

Abstract

With increasing demand for low power high performance computing,reducing power of not only CPUs but also memory is becoming important. In typical general-purpose HPC environments, DRAM is installed in an over-provisioned fashion to avoid swapping, although in most cases not all such memory is used, leading to unnecessary and excessive power consumption, even in a standby state. We propose a next generation low power memory system that reduces required DRAM capacity while minimizing application performance degradation. In this system, both DRAM and MRAM, fast non-volatile memory, are used as main memory, while flash memory is used as a swap device. Our profile-based paging algorithm optimizes memory accesses by using faster memory as much as possible, reducing accesses to slower memory. Simulated results of our architecture show that the overall energy consumption of the memory system can be reduced to 25% by in the best case by reducing DRAM capacity, with only 17% performance loss in application benchmarks.

1 Introduction

In recent years, power consumption of HPC systems have been on the rise, and its reduction has become one of the major concerns in its design and management. Most previous work had focused on reducing energy for processors, biggest power consuming part of system in many cases, with dynamic voltage scaling (DVS)[8]. However, due to higher capacity of memory chips and corresponding decrease in price, recent HPC systems are equipped with substantially larger memory, which is quickly becoming the major source of power consumption. Recent energy breakdown study measured on a real server shows that main memory consumes 41% of the total energy and it is 50% larger than processors[7].

One may argue that such large memory footprint is necessary for HPC, but for many applications such capacity is overkill. Principally, there is a strong belief that parallel HPC applications should not swap that would be fatal to performance, and therefore memory is typically overprovisioned up to any foreseeable application with maximum memory usage, and as a result, in real systems memory usage over time is usually low[2].

Now DRAM, being volatile, usually incurs large standby power. Since main memory size of some HPC applications are on the increase as performance increase of CPUs, such overprovisioning of memory results in significant waste in power and energy. We propose to challenge to reduce such DRAM over capacity with the following approach.

We replace a part of DRAM with next generation memory MRAM, which has high speed and low power, albeit at a higher system cost compared to DRAM. By placing frequently accessed data onto MRAM and reducing DRAM capacity at the same time, we can reduce power consumption while speeding up memory accesses. Moreover, for application that requires higher memory usage, we first use the DRAM portion, and then selectively swap to FLASH memory, which has significantly lower latency and power than HDDs. In order to best utilize this revised hybrid memory hierarchy we devise a *low power paging algorithm* that accounts for such usage frequency and power characteristics. In order to evaluate above system, we create a model to predict execution time of applications and energy consumption of memory system, with which we parameterize our simulator. Results from several applications show that the energy consumption of memory chips can be reduced up to 25% or only 1/4 of the original, with only marginal 17% performance loss. Although not applicable to all application cases, the most interesting aspect of our finding is that, the lowest overall energy consumption is actually observed when swaps occur, for applications such as NAS CG and HPL (High Performance Linpack) that exhibit some amount of locality, with only mild speed downs. Such a result is important for future systems where reducing the amount of DRAMs without substantial performance sacrifices would be tantamount to large system scaling.

2 Background: Non-Volatile Memory—FLASH and MRAM

Non-volatile memory is in high demand for low power embedded computing. FLASH memory is in widespread use in many embedded and consumer devices, and now for mainstream PCs and servers such as Solid State Disks(SSDs) and HHDDs(Hybrid Hard Disc Drives) as alternative to classic HDDs. Fusion-io is a FLASH storage device connected by PCI Expresses and can read at 800MB/s with 8KB sequential access speed, far outpacing HDD I/O performances. Windows Ready-Boost in Windows Vista uses USB FLASH memory as I/O cache devices. Future capacity increase looks promising with continued expansion of application areas, with implementations of MLC (Multi Level Cell) and 3-D structures. However, use of FLASH devices has received little attention in the HPC space, especially in relation to lowering power consumption as we propose here, due to its substantially slower write speed and slower access latency compared to DRAM, as well as significantly limited life in the number (approximately 100,000) of writes making it infeasible as direct main memory substitutes.

There are several non-volatile memory technologies that can serve as main memory developed in the recent years. They are, for example, Magnetoresistive RAM (MRAM) [15], Phase-change RAM (PRAM), and Ferroelectric RAM (FeRAM). Out of these, MRAM is ideal in that it exhibits higher access speed and lower power than DRAM; there are currently (circa 2007) small capacity (a few Megabits) chips already available from companies such as FreeScale. Compared to DRAM that holds volatile states as electrical charges in a capacitor, MRAM maintains its non-volatile state by the polarity of two ferromagnetic plates. Although MRAM may incur higher write energy, overall it is lower power than DRAM since the state is non-volatile.

However, the biggest problem with MRAM is its extremely high cost-per-bit due to limited production. Until massive production is seen in a not-so-near future, the price of MRAMs will be at least an order of magnitude more expensive than DRAMs at the same capacity. As such, most expectation is that MRAM will be used in a sparing, small-scale fashion primarily in the embedded space, where small memory is feasible, and non-volatility and lower power consumption could be considered as premium, whereas for HPC, memory capacity/price metric precludes its use.

3 Our Proposal—Hybrid Memory Architecture and Power Aware Swapping

In order to achieve substantially lower power consumption in memory without significant loss in performance, we propose hybrid usage of next-generation memory technologies so as to compose a much more sophisticated memory hierarchy than previous HPC systems, so as to reduce the overall amount of DRAM in the system. Later on, we show that we can achieve significant reduction in DRAM capacity and thus power savings with moderate speed penalties for certain applications.

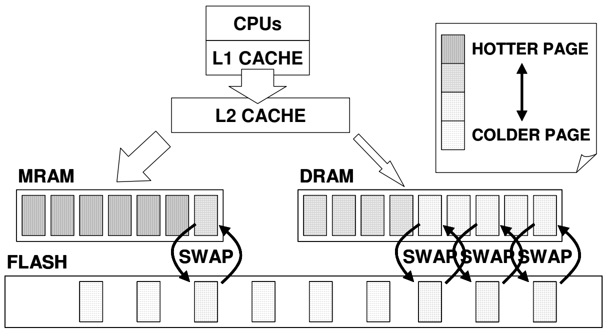


Figure 1. Overview of Our Proposed Low Power Memory Architecture

More specifically, our next generation low power memory architecture makes parallel use of both MRAM and DRAM as main memory, and FLASH as fast random-access swap device, to which both MRAM and DRAM pages are swapped in/out directly. An overview of our architecture is shown in Figure1; here, notice that faster-but-lower-capacity MRAM is not used as a cache to DRAM as one would expect, but rather in parallel with DRAM comprising main memory. This is because 1) differences between MRAM and DRAM speeds are much less significant compared to SRAM/DRAM combinations, 2) despite the higher cost, MRAM can still be much larger than CPU caches economically, and 3) larger transfer granularity precludes the use of MRAM as exclusive cache, and thus does not help to reduce DRAM capacity, which is our original objective.

Merely replacing DRAM with MRAM will not provision for datasets that overflows main memory. In fact, we would like to investigate the possibility of whether swaps can be done efficiently for such situations, counter to the traditional wisdom that "HPC apps shall never swap"; in fact we would want to investigate the possibility of whether swapping actually entails lower energy cost. For this purpose, we resort to FLASH memory as a swap device with significantly faster random access read speed, and observe the energy consumption differences.

3.1 Low-Power Paging Algorithm

The proposed architecture cannot work without *low-power paging* algorithm, which we also propose and implement here. As a general observation, MRAMs should have prioritized usage since it has higher performance and lower power, while swaps should be done in a much more strategic fashion than standard LRU paging. For this purpose, we profile the application memory access, and come up with optimal strategies for MRAM/DRAM/FLASH usage.

More specifically, we first claim that LRU or its approximation would not serve well, as the information conveyed in the LRU scheme does not necessarily select the "hottest", i.e., the set of pages that are accessed the most frequently throughout the course of the execution, not just within a given timeframe. Since our energy minimization is global rather than local, we need much more finer-grained control of page placement.

For this purpose, we assume that the algorithm can obtain, possibly through profiling, the per-page memory access frequency of a given application throughout its execution. Pages that experience large number of accesses are called *hot pages*, and others called *cold pages*. Moreover we also assume that the algorithm can know access count on DRAM and MRAM at any given point in time from start of an application. Both can be obtained in various ways via pre-execution trial runs, or through selective sampling with hardware assists.

Given such information, our first simple algorithm works as follows: we pin down the hottest pages so that they are never swapped out and allocated onto MRAM (called MRAM fixed pages). For this, we allocate the pages with the most frequent accesses in a descending fashion onto the (non-swapped) MRAM region until we run out of MRAM pages. The remaining pages are allocated onto DRAM and are subject to LRU-based swapping with flash memory.

However, we were surprised to find that, this simple algorithm in some cases increased application execution time by nearly a factor of two compared to simple LRU. Upon profiling the situation, we found the reason: the phenomenon was especially observed for applications that have relatively lower memory access locality, i.e., the difference between the high and low page access counts were small. In such a case, pages with only minor access differences were pinned down while effectively we were faced with great reductions in the total amount of DRAM memory, causing excessive swaps, slowing down the application considerably.

To resolve this situation, we extend our algorithm by introduce a metric called *MRAM hit rate* and its threshold, so that applications exhibiting lower locality may use both MRAM and DRAM as swappable main memory. MRAM hit rate is a dynamic value that indicates the ratio of the access counts onto MRAM versus memory accesses to all the memory at each point in execution time. If the ratio is large, then we can decide that accesses to MRAM has sufficient locality such that pages should be pinned down. On the other hand, if the ratio is small, the application lacks locality and thus the entire main memory should be seen as swappable. More concretely, we define the following metric:

$$Thr = \alpha \times MRAM\_SIZE/TOTAL\_SIZE$$

In the above equation, MRAM_SIZE and TOTAL_SIZE indicate the total MRAM capacity and sum of MRAM and DRAM capacities, respectively; α ($\approx$1) is a configurable parameter to be used to determine the threshold. Several preliminary experiments have shown that a threshold value of 0.9 seems to work for the NAS and other HPC applications we have tried, although further performance analysis will be required to precisely determine this value to be optimal.

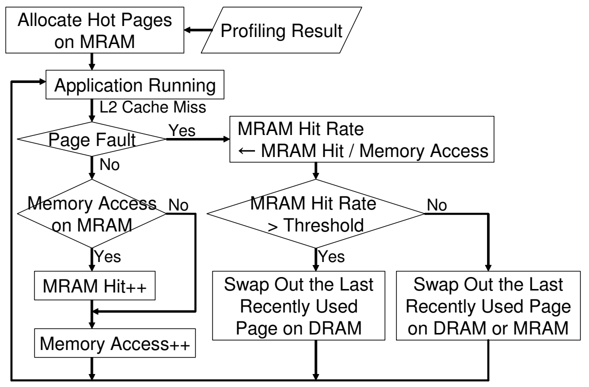

Allocate Hot Pages on MRAM
Profiling Result
Application Running
L2 Cache Miss
Page Fault
Yes
No
MRAM Hit Rate ← MRAM Hit / Memory Access
Memory Access on MRAM
No
Yes
MRAM Hit Rate > Threshold
No
Yes
MRAM Hit++
Memory Access++
Swap Out the Last Recently Used Page on DRAM
Swap Out the Last Recently Used Page on DRAM or MRAM

Figure 2. Algorithmic Flow of Our Proposed Paging Algorithm

There are further optimizations specific to FLASH memory: for example, if there is a replica of a page in main memory within the FLASH swap device, we need not write back the clean page. Although such optimization is well known[11], it has major advantages for FLASH-based swap device because writes to FLASH are slower than HDDs, and that the number of writes to FLASH have limited lifetime as mentioned.

We show the overall algorithmic flow of our proposed method in Figure 2.

3.2 Performance Modeling of Time and Energy Consumption

We estimate application executing time and energy consumption of our proposed memory by using run-time profiling of actual execution of applications, instantiated with the performance models of time and energy consumptions.

We first run our application execution and obtain a complete trace of memory accesses. Then, total memory access delays are determined as a product of memory access delay by the total memory access count, and energy consumption in a similar fashion. The delay and energy per access is different depending on type of memory (MRAM or DRAM) and access granularity. When page swapping occurs, both reads and writes occur at page granularity with associated time and energy consumptions, but there are a few exceptional cases. In one case, if the swapped out page is a clean page, the write operation is nullified. In another case, the time delay for reading and writing main memory is hidden by access delay to FLASH and excluded from total memory access delays. Contrastingly, energy consumption cannot naturally be hidden away in the latter case, and thus properly accumulated as dynamic energy consumption.

Overall, the application execution time is determined to be the sum of these memory access delays and computation time independent from memory accesses, and the total energy consumption of memory chips are determined to be the sum of dynamic energy consumption and the static energy consumption as defined by each memory technology.

4 Evaluation

4.1 Architecture Parameters

We evaluate our architecture by simulation, since our architecture includes next generation memory architecture and fast MRAM chips are not available yet. Table 1 shows parameters of cache and memory pages in the simulated environment. Table 2 shows those of memory modules: DRAM, MRAM and FLASH. It shows the delay and energy per access unit, which is an L2 cache line of 64 bytes in DRAM and MRAM, and a page of 4096 bytes in FLASH.

The parameters of DRAM are derived from performance of typical 64MB DDR3 chips. For parameters of FLASH memory, we take the performance of existing SSD products from Samsung as a baseline, which have 58MB/sec read speed, 32MB/sec write speed, and 0.4W active power. With an expectation of improvement on cell integration, we have determined FLASH parameters by doubling the speed and reduce the power to 50%. To make model simpler, we assume sufficiently large capacity for FLASH and standby power per bit is negligible.

It is more difficult to set parameters for MRAM modules, because it is emerging and currently available ones are slower than DRAM. Recently, NEC has developed MRAM chips whose speed is similar to SRAM. Thus we let access delay of MRAM be 1.5 times faster than DRAM. To determine read/write energies, we take MRAM chips from FreeScale, whose read power is 182mW and write power is 346mW (typical cases). And we expect they will be improved in a few years, and reduced to 50%. By multiplying the reduced powers and access delays above, we have determined read/write energies in the table. We estimate standby power by using the following assumptions. First, standby power per FreeScale MRAM chip (currently 60mW) will be reduced to 50% again. Next, capacity of MRAM chips will become equal to that of DRAM chips, although there is a large gap currently (0.5MB versus 64MB). It is difficult to say whether these assumptions will be realistic in a few years, and we will reconsider parameters as non-volatile memory technology proceeds.

4.2 Evaluation Method

With the above architecture parameters, we conduct simulation to evaluate the performance and energy consumption of several application benchmarks, for various MRAM size, DRAM size, and paging method. The benchmarks are CG (class B), MG (class A) and SP (class B) from NAS Parallel Benchmark 3.2[1] and HPL[13], all of which have been executed on a single node. To reduce the size of trace files and simulation time, we have adjusted the problem sizes; the number of iterations is five in NPB benchmarks, and matrix size in HPL is 8192. With these configurations, we have observed that the memory sizes used by CG, MG, SP and HPL are 412MB, 448MB, 344MB and 544MB, respectively.

As the first step of simulation, we make a trace file of memory accesses in each application benchmark by using the Valgrind profiling tool[3]. For each memory access that incurs L2 cache miss, we collect memory address and operation types (read or write). A trace file also includes profiling results, which are access counts on all the memory pages.

With the trace files, we replay behavior of application with our event-driven simulator. It can simulate our paging methods, LRU, MRAM-fixed and our low-power algorithm on various sizes of DRAM and MRAM. It maintains the number of memory accesses and page swapping to calculate the total access energy and the total access delay, which are used in the model described in Section 3.2.

Table 1. Simulated Memory System

L2 cache size	1MB
block size	64B
associative size	1
page size	4KB

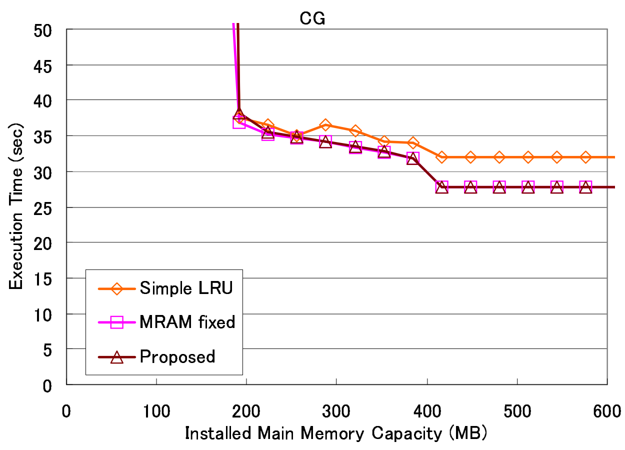
Table 2. Parameters of Memory Modules

	DRAM	MRAM	FLASH
access size(Byte)	64	64	4096
READ delay(ns)	22.5	15	35000
WRITE delay(ns)	22.5	15	64000
READ energy(nJ)	6.24	1.36	7000
WRITE energy(nJ)	6.24	2.60	12800
standby power (μW/MB)	867	469	0

4.3 Evaluation of Paging Method

We evaluate effects of paging methods to application performance. Figure 3 illustrates execution times of CG and MG for various DRAM capacities. Here, MRAM capacity is fixed to 128MB. The lines in the graphs correspond to paging methods: Simple-LRU, MRAM-fixed and our low-power algorithm (Proposed in the graph), respectively.

In both benchmarks, we see our proposed method achieves the best performance in almost all cases. It is sometimes inferior to MRAM-fixed in MG, but the differ-



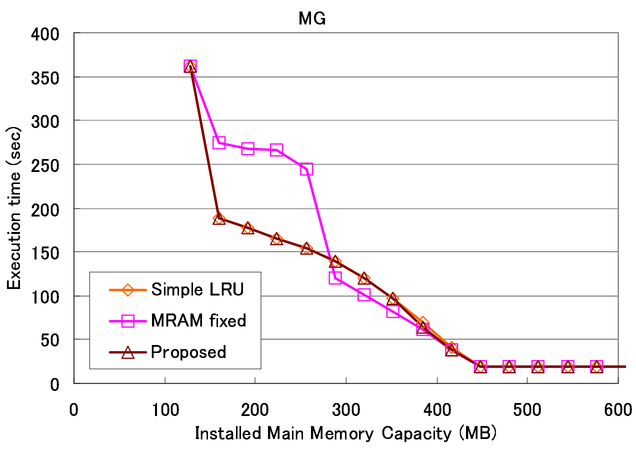


Figure 3. Evaluation of Paging Methods with 128MB MRAM

ence is small. In CG, MRAM-fixed and Proposed are faster than Simple LRU, which means that they successfully allocate hot pages on MRAM. The gain of the methods is up to 15% in this application.

On the other hand, we see a large difference between MRAM-fixed and Proposed; the former fixed can be 60% slower than Simple-LRU. To discuss this difference, we have examined locality of memory accesses in each application. Figure 4 shows how accesses concentrate to hot pages. For example, we see that about 98% of accesses concentrate to 40% of used memory pages in CG, thus we can say it has high locality. Also, HPL shows similar tendency. On the other hand, such tendency is much weaker in MG and SP, and all the pages are accessed at similar frequency. We consider this difference is the reason why MRAM-fixed does not work well in MG. On the other hand, our low-power algorithm, which adapts to locality of application dynamically, is efficient in both types of applications.

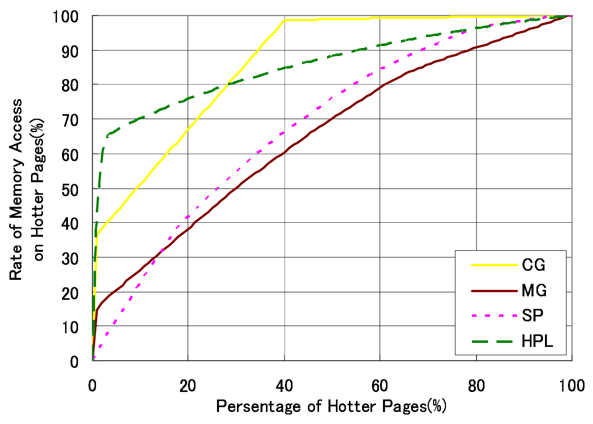


Figure 4. Memory Access Locality of Application Benchmarks

4.4 Evaluation of Application Performance

Graphs in Figure 5 show execution time of applications for various DRAM and MRAM capacities. The X-axis corresponds to capacity of the total main memory of DRAM and MRAM, and each line corresponds to that of MRAM: no MRAM, 64MB, 128MB and 256MB. All the cases use our low-power paging method.

In all applications, the execution time gets longer as main memory decreases, because of swapping cost. However, effects of swapping heavily depend on applications; while the performance of MG and SP notably suffers from swapping, the effects are much milder in CG and HPL. For example, in CG with 256MB main memory (DRAM only), the increase of execution time caused by swapping is limited to 20%. However, if memory capacity gets smaller than a certain point (192MB in CG and 64MB in HPL), the performance gets heavily worse even in CG and HPL.

We consider that this difference comes from access locality of applications, as discussed above. According to Figure 4, 98% of memory accesses in CG are targeted for 40% of used memory space, whose size is about 165MB in this case. Similarly, 65% accesses in HPL are for about 3% of memory (16MB). This property explains the above behavior fairly well. On the other hand, in MG and SP, since all the pages are accessed almost equally, even a small lack of main memory incurs large swapping costs.

4.5 Evaluation of Energy consumption

Figure 6 shows energy consumption by our memory architecture during execution of each application. First, we

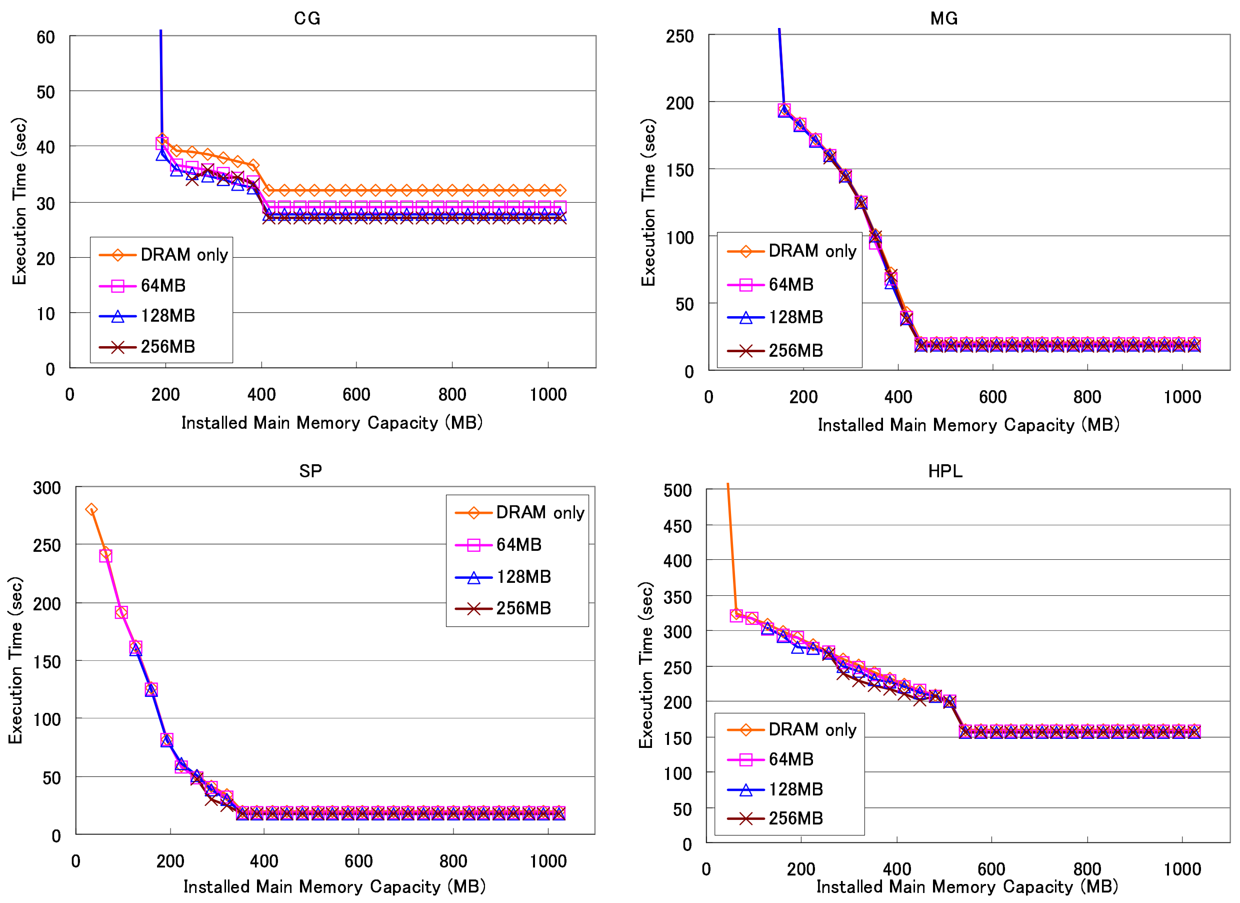


Figure 5. Transit of Execution Time of Applications

see that too large capacity and too small capacity heavily increases energy consumption. Too large capacity increases static energy consumption by standby power, which is proportional to memory capacity. On the other hand, when memory capacity is smaller, we suffer from two factors, both of which are due to swapping. First, since the execution time gets longer by swapping, static energy consumption increases. Next, accesses to FLASH memory for swapping also consume energy.

In addition to this general tendency, we observe an interesting property in three graphs except in SP; each graph has two notches, where energy is at local minimum. Obviously the right notch is due to the total memory size required by each application. On the other hand, the left one corresponds to capacity of hot pages, which is discussed above. We see that not only CG and HPL, but MG has the second notch. However, relationship between notches is different; in CG and HPL, the left notch is more energy-effective even with swapping, while the right is better in MG.

These results provide us a new observation on memory capacity; in HPC applications with sufficient locality, reducing DRAM capacity aggressively can reduce energy consumption, even with swapping. Then the next question is, how can we distinguish applications with good locality from others? And how can we find the best capacity (where are the notches)? We will challenge these issues in the future.

Next, we discuss the capacity of MRAM. In all applications, increasing MRAM instead of DRAM reduces energy, while its effect depends on applications. In CG, we see difference between ”DRAM only” and ”64MB” is larger than other cases. Especially in this case, using MRAM is energy-effective even if its capacity is small. We are currently investigating reasons for the difference between applications.

Finally, we compare performance of proposed architecture with ”typical” design of HPC systems. In typical systems, memory consumption tends to be 30–50% [2]. Since our applications require 344 to 544MB memory, it is rather natural to execute them on a machine with 1GB DRAM. On such a typical architecture, CG consumes 33J (Figure 6). Instead, when we install 128MB MRAM and 64MB DRAM (we assume that the left notch can be found), the

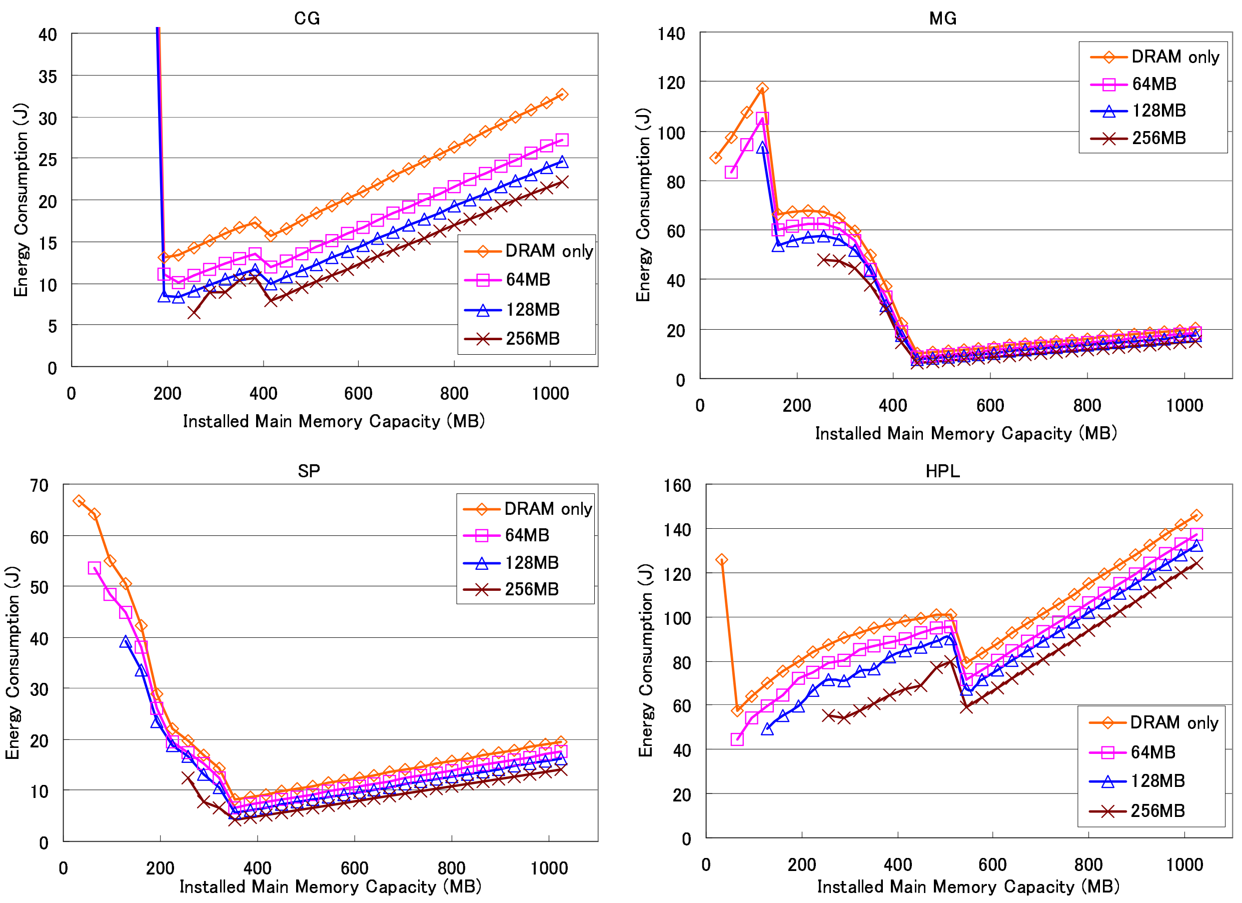


Figure 6. Transit of Energy Consumption by Memory System

energy consumption is reduced to about 8.5J, which is 25% of that on the typical architecture. In this case, the increase on execution time, compared to the typical case, is limited to 17%.

5 Related Work

We can find several studies of energy consumption of memory modules with various approaches. Improving cache efficiency enables to save energy consumption, for it reduces accesses to lower level cache and memory, and simultaneously shortens execution time of applications to save static energy. Kondo et al. have proposed a memory architecture that improves such efficiency with software controllable on-chip memory[12]. Hung et al. and Lebeck et al. control memory states dynamically, on an assumption that DRAM chips have standby and/or low power mode[9][4]. Since memory state can be switched per chip rather than per page, they propose paging methods to collect active pages in a smaller number of chips. Cai et al. also tune active memory size, in order to minimize the total energy consumption of memory chips and HDDs by tuning timeout interval of HDDs[6]. Tolentino et al. proposed several page shaping techniques for minimizing the number of active memory chips[10]. Their architectures, except that of Kondo et al., have been evaluated with desktop or server applications, rather than HPC applications. Also, their approaches in themselves do not reduce introduction costs of large main memory.

Remote swapping, which uses memories on remote machines as swap device, has been proposed[16] and evaluated with modern high speed networks, for example with InfiniBand[14] or 10GbE[11], achieving lower swapping costs than using HDDs. As far as we know, energy consumption of systems using remote swapping has not been reported. While it can reduce local memory capacity, energy consumption in remote has to be considered. However, in large scale systems with multi-applications, it is expected to smooth the memory usage among machines to reduce the total capacity of main memory. Thus it would be one of promising approaches to combine remote swapping and our architecture.

While the above architectures assume main memory to be homogeneous, embedded systems often embody hybrid main memories such as SRAM and DRAM, mainly for the limitation of power consumption and mounting space. In such systems, allocation strategy of data has a large impact on performance; thus Avissar et al. have proposed a compiler technique to allocate static data so that it minimizes memory access delay[5]. However their method is difficult to accommodate large arrays or dynamically allocated regions, which are frequently used in HPC. On the other hand, we have proposed a paging method to use MRAM and DRAM efficiently and dynamically, and evaluated it with HPC applications.

6 Conclusion

We have proposed low power memory architecture including next generation non-volatile memories. In our architecture, we replace a part of DRAM from MRAM being non-volatile memory for reducing capacity of DRAM costing high power at main memory level, and we allocate FLASH that expected to become large capacity storage as swap device. And we have also proposed the low power paging method utilizing heterogeneous memories efficiently. For evaluating our proposed architecture, we construct the performance model estimating application execution time and energy consumption of memory chips. We evaluate our architecture with simulating several HPC application benchmarks. Simulated results show that the energy consumption can be reduced to 25% by reducing DRAM capacity, with 17% performance loss of application benchmarks. Moreover we obtained an interesting observation that is counter to the traditional wisdom; reducing DRAM capacity even if swapping occur reduces energy consumption in execution of the applications having high locality memory accesses.

Our short term future work are that we extend our model confined to only memory chips to the one include CPU and the others components and considering paging method based on other than LRU. And in this paper we assume that MRAM excel than DRAM at speed and power, but we will have to consider memory hierarchy and introduction of the other non-volatile memories according to the technological direction in the future.

We aim to utilize accomplishment of our work for designing and management power-aware large scale HPC systems in future, there are some challenges below. First the method that dynamically detects optimum main memory capacity that minimizes application's energy consumption and is indicated in our simulation results. And we consider construction of the performance model for reducing energy consumption of multi applications and coordination with job schedulers, remote swapping and the technique of dynamical memory states switching.

Acknowledgments

This work was partly supported by JST, CREST and Microsoft Technical Computing Initiative.

References

[1] Nas Parallel Benchmarks. `http://www.nas.gov/software/NPB`.

[2] TSUBAME Grid Cluster, Tokyo Institute of Technology. `http://www.gsic.titech.ac.jp/~ccwww/`.

[3] Valgrind. `http://valgrind.org/`.

[4] A.R.Lebeck et al. Power aware page allocation. In *ASPLOS-IX: Proceedings of the ninth international conference on Architectureal support for programming languages and operating systems*, 2000.

[5] O. Avissar, R. Barua, and D. Strwart. Heterogeneous memory management for embedded system. In *International Conference on Compilers, Architecture and Synthesis for Embedded Systems(CASES)*, 2001.

[6] L. Cai et al. Joint power management of memory and disk. In *Proceedings of the conference on Design, Automation and Test in Europe (DATE'05)*, volume 1, pages 86–91, 2005.

[7] C.Lefurgy et al. Energy management for commercial servers. *Computer*, 2003.

[8] C.Liu et al. Exploiting barriers to optimize power consumption of cmps. In *19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)*, 2005.

[9] H.Hung et al. Design and implementation of power-aware virtual memory. In *USENIX 2003 Annual Technical Conference*, 2003.

[10] M.E.Tolentino et al. an implementation of page allocation shaping for energy efficiency. In *International Parallel and Distributed Processing Symposium (IPDPS'06) HP-PAC*, 2006.

[11] M.Goto et al. implementing remote swap memory using rdma over 10gb ethernet (in japanese). *The Institute of Electronics, Information and Communication Engineers*, 2006.

[12] M.Kondo et al. Reducing memory system energy in data intensive computations by software-controlled on-chip memory. In *Int'l Workshop on Compilers and Operating Systems for Low Power*, 2002.

[13] A. Petitet, R. C. Whaley, J. Dongarra, and A. Clearyb. HPL - a Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers. `http://www.netlib.org/benchmark/hpl`.

[14] S.Liang et al. Swapping to remote memory over infiniband: An approach using a high performance network block device. In *Proceedings of the IEEE Cluster Computing (Cluster2005)*, 2005.

[15] S.Tehrani et al. Magnetoresistive random access memory using magnetic tunnel junctions. In *Proceedings of IEEE*, 2003.

[16] T.Newhall et al. Nswap: A network swapping module for linux cluster. In *Proceedings of Euro-Par '03 International Conference on Parallel and Distributed Computing*, 2003.