International Journal of Applied Engineering Research ISSN 0973-4562 Volume 13, Number 6 (2018) pp. 3610-3618

A Data Management Technique of Hybrid Memory Systems for Energy Aware Flight Control Computing

Doosan Cho
Department of Electrical and Electronic Engineering,
Sunchon National University, Sunchon 540-742, South Korea.

Abstract:

Future flight control devices will be designed with more application-specific features than traditional designs. For an instance, devices mounted on shoes can provide various information by analyzing life pattern, and wrist watch devices are becoming not only information providing but also health care. These various drone/mobile devices are loaded with an optimum memory system for their ability to perform such computations. Battery capacity is determined by required capacity of memory system, which consumes a large portion of power. The size of a memory system and battery affects such device size, and then it directly links to success of the product in its market. Therefore, the optimization process mainly focuses on power consumption and performance at the design step.

The size of a memory system is usually occupied upto 60% on a chip space and its power consumption is proportional to the size. Especially, as the microfabrication process deepens, leakage current increases exponentially in system on a chip. This is called the power wall instead of the traditional memory wall. We are focused on a hybrid memory system using nonvolatile and scratchpad memory components to solve the power wall problem. Non-volatile memories have a small size (high density) and a relatively low power consumption, as well as consuming nearly zero leakage in a chip. Therefore, it is considered as a substitute for the traditional memory subsystem to the next generation high performance devices. In this paper, we propose a data management technique that can efficiently utilize the hybrid memory consisting of non-volatile memory and the scratch pad memory components. The proposed scheme overcomes the nonvolatile memory write endurance constraint and relatively slow write speed by using scratch pads and takes full advantage of nonvolatile low power and performance advantages. Using this study, it is possible to construct a memory subsystem optimized for a low power perspective for future high performance devices**.** Therefore, the proposed technique improves performance and energy consumption of the hybrid memory architecture.

Keywords: cache; memory subsystem; energy consumption; drone; flight control computer

INTRODUCTION

DRAM is the main memory device in which real processes perform tasks. In the past decades, the memory capacity has been expanded over 30 years at a rate of about 100 times per 10 years. Along with Moore's Law, which represents an increase in processor speed, it was the pillar that sustained IT technology development. However, 4 to 5 years ago, DRAM manufacturing technology was expected to be very difficult to develop in nano-level micro-processing under 30nm. When a capacitor of DRAM is going to store data, the capacitor needs to have 20 femto farads capacitance as the minimum. However, the capacitor cannot have upto 20 femto farads under 30nm technology, due to the capacitor's volume. There are several solutions to solve this capacitance problem by using high cost materials like white gold. If it chooses the white gold, the price competitiveness of DRAM is worser because it leads to the highest cost as a mobile devices' component. As a solution to this deepening microfabrication process, many companies have chosen nonvolatile memory technology, since nonvolatile memory provides higher density, lower power consumption, zero leakage power, and the same performance compared with DRAM.

Few years ago, researchers expected that the next-generation memory technologies such as non-volatile memory would replace traditional DRAMs to solve a limitation of deepening microfabrication process. However, the expectation of fabrication technology is not true, because it already commercialize at 14nm level product and commercialization of 10nm or less technology will be expected. Therefore, it seems that 'complementary' rather than 'replacement' will be commercialized as a type of hybrid memories. That is, they are developed in the form of complementary hybrid memories, but nonvolatile memories.

Few leading industries are already accelerating the development of the next generation memory technologies that can enhance the performance of traditional memories such as DRAM and NAND flash and compensate for their shortcomings. It is not an approach of developing a new product to create a market but a complementary approach to meet market demands. If it succeeds in commercialization, it will quickly become a new engine for growth market. The next-generation memory technology is focused on complementing performance rather than replacing DRAM, SRAM, and NAND flash, which are already well-positioned in the market. Major candidates of the next generation memory technologies are 3D X-Point, ReRAM (Resistive Memory), P-RAM (Phase Change Memory) and STT-MRAM (Spin Injection Magnetization Memory). In this work, we call them NVM (Non-Volatile Memories). Each memory technology has different advantages and disadvantages. Because STT-MRAM is faster than DRAM, it can be used as an on-chip memory or the last-level cache, but due to its large cell size, it is not an alternative solution of DRAM. PCM and ReRAM have a higher degree of density than DRAM, but

slow access speed. Thus, it is not easy to replace DRAM with the single technology. 3D X-Point memory technology was recently announced by Intel and Micron. It is known that the speed can be increased up to 1000 times as compared with the NAND flash. It is expected to replace SSD in the near future.

In this work, we focuses on a hybrid cache memory subsystem, which compensates for the drawbacks of NVM and DRAM, and helps them to share their advantages. As mentioned above, NVM is a memory technology that enables to fully utilize deepen microfabrication with zero leakage power, high density, low power consumption and non-volatility. However, current implementations of this promising alternative memory suffer from an important issue that its cells require relatively large energy and access latency when performing a write operation. It limits the ability to meet the requirements for the memory access latency in high performance systems. That is, SRAM cell is much more efficient for performing write operations with improved power consumption, performance and endurance compared to NVM cells, even though the cell size is four times lager. In order to compensate for the drawbacks of NVM with write operations, a hybrid memory system has been proposed with a scratchpad memory (SPM) component. By partitioning write intensive data onto SPM, it can provide a high performance memory system with keeping NVM advantages, since SPM performs a write operation at high speed and low energy consumption. In this paper, we explore a novel hybrid cache architecture consisting of SPM and traditional cache in NVM main memories, and its management method.

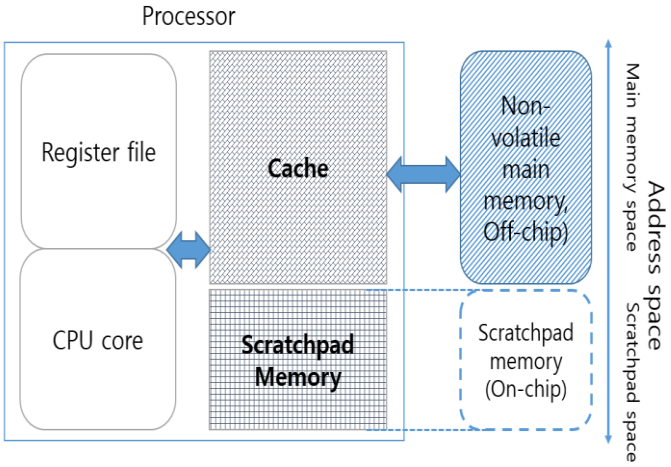

Figure 1. Hybrid cache architecture

Several previous works [1], [2], [3-12] confirm that NVM main memory can achieve significant energy savings with comparable performance to that of DRAM. Some research using NVMs to build cache hierarchies also show that NVM has advantages over SRAM when there is a certain management scheme. In order to obtain the advantage of these two types of RAMs, some works compose a set of caches that are configured with a small number of SRAM lines and many NVM (STT-RAM) lines in a hybrid manner. Usually, SPM is responsible to handle the frequently recorded data blocks that constitute a major part of the write operations in an application while NVM hides the read access latency by increasing the capacity of this hybrid cache. In this work, main memory consists of NVM and its address space controlled by traditional cache. A shown in Figure 1, the software controlled SPM is used as part of the main memory address space. And, it is a part of a software controlled data cache. The design of this hybrid cache memory subsystem is popular with low power mobile system such as IBM CELL architecture, NVIDIA GPUs, ARM Cortex-M and Cortex-R, etc. The detailed structure and its operation procedure is beyond the scope of this paper.

To the best of our knowledge, there is no existing work about a hybrid scratchpad memory cache with a NVM main memory (which called NVMM). With this hybrid scratchpad memory architecture, we can achieve many benefits, such as high density, non-volatility, and ultralow leakage power, promised by NVMs. To that end, we propose an efficient cache management technique for the hybrid memory subsystem that identifies write-intensive data blocks and rearranges the location of such data blocks for placing to SPM. Other data can be categorized as less-write or read-intensive data, they should be placed in NVM with the hardware cache.

A number of similar studies have been proposed to alleviate negative effects of the hybrid cache. On architecture design point of view, various Non-Volatile Memories (NVMs) have also been proposed to be used in hybrid cache. Mangalagiri [13] proposed PCM based a hybrid cache architecture. Dong [14] evaluated 3D MRAM architecture as an on chip cache. Joo [15] presented energy/endurance aware PCM based a cache design. In these works, they all considered NVM as a cache for adding current microarchitectures, and naturally they imply that it is technically feasible to integrate NVMs with SRAM into an onchip memory.

In system management point of view, there are a number of similar studies to minimize the drawback of NVM which caused by write operations. Reducing the number of write operations to NVM (like STT-RAM) can lower the negative effect. Thus, a data allocation technique is essential to reduce the number of write operations on such NVMs. Studies [16] show that hybrid caches with proper data rearrangement policies consumes less power than traditional cache. However, the cache rearrangement policies leads too much overhead since the policies did not make full use of many known characteristics of data access pattern in embedded applications. That is, their works focus on reducing power consumption with an objective function. As a result, it yields negative side effect in cache hit ratio. Our approach differs from the existing works in that ours improve energy consumption by reducing the write operations on NVM without increasing any cache misses. Because this work optimize placement of write intensive variables by partitioning them from NVM, there is no side effect. Specifically, data access information can be captured by profiling, and based on such information write-intensive data can be identified along with a function call graph. The negative effect of a hybrid cache can be reduced by placing such write-intensive data onto SPM. In experimental result, the proposed technique can achieve 6% performance improvement and 5% energy saving.

The rest of this paper is organized as follows. Section 2 introduces a background knowledge of the live variable analysis. It is the major component of the proposed data management technique for this hybrid cache memory subsystem. Section 3 presents a detailed optimization process to determine the variables placement. The experimental results are presented in Section 4, and finally, the conclusion is presented in Section 5.

BACKGROUND

Our technique is based on data flow analysis (DFA) which is one of compiler program analysis techniques. DFA is a technique for analyzing live range of each variable along with various execution paths. The definition of various terms related to LVA (Live Variable Analysis) are summarized as follows.

Definitions

- The variable v is said to **live** at some point p of the program, which means that the variable v is used at least once during the program's execution after the position of p.
- A variable v is dead at some point p in the program means that the variable v is not used at the point p.
- If a basic block is called B (where the basic block is a straight line code with no branches), the set of variables defined in the basic block is denoted as DEF (B). We denote DEF (B) as the set of variables in which the variable values are computed and stored in basic block B. If it is not necessary to mention basic blocks in this work, DEF will be used. This is the same for USE, IN, and OUT defined below.
- If a basic block is called B, then only those variables that are not defined and used in B are called USE (B).
- If a basic block is called B, the set of living variables at the entry of B is called IN (B).
- If a basic block is called B, the set of living variables at the end of B (exit) is called OUT (B). The relationship between IN and OUT is described in the next section.
- LVA finds OUT (B) for every basic block B of a target program. In conclusion, LVA aims to find out information about a set of variables that must exist for all basic blocks.

Live Variable Analysis

The algorithm that performs traditional LVA follows the DFA (Data Flow Analysis) framework. Here is how the components of the framework are defined in LVA.

- Flow values: I (B) is defined as IN (B) and O (B) is OUT (B) as defined in the previous subsection. That is a set of variables.
- Meet operator: A basic block in one of its predecessors in a basic block, that is, a variable that lives only in one node, must also live in its basic block. Since it means a union operation between sets, meet operator is a union operation. Let P be the successor basic block of basic block B. IN(P) and IN(Q) must be alive at the exit of B. Therefore, IN(P) U IN(Q) $\subset$ OUT(B). Conversely, variables contained in OUT(B), that is, the variables that must be alive at the exit of B. They must live at the entrance of the basic block after B is executed. It is because the variables have to live in every path from definition to use. Therefore, all variables of OUT(B) are included in either IN(P) or IN(Q), so OUT(B) $\subset$ IN(P) U IN(Q). That is, OUT(B) = IN(P) U IN(Q).

Flow functions: it is a function that returns IN(B) by insertion of OUT(B) because it is backward DFA format. f_B : OUT(B) – DEF(B) U USE(B) = IN(B).

- Flow equations: it shows the relationship between flow values.

$$OUT(B) = U_{each\_successors\_of\_B} IN(S)$$

OUT(B) – DEF(B) U USE(B) = IN(B)

The LVA algorithm using the DFA framework is shown in Algorithm 1.

Algorithm 1: live range analysis

Procedure Live_Range_Analysis

For each basic block BB do

IN(BB) = USE(BB)

End

While changes to any IN(BB) occur do

For each basic block BB in reverse Depth-First-Search order do

OUT(BB) = U_all_successors_of_BB_IN(S)

IN(BB) = USE(BB) U { OUT(BB) – DEF(BB) }

End

End

End of procedure

Let's consider the principle of operation by looking at Algorithm 1 and Figure 2. Figure 2 shows how the flow equation is applied. It shows the process of collecting the INs at the bottom and making the OUT, inserting the OUT into the flow function, and then computing the IN again. A bold arrow indicates the direction in which the information flows. If there is a loop in the input code, the procedure is as follows. From the bottom up, compute IN and OUT by visiting basic block in reverse order of Depth-first Search (DFS). It is a repetition

of the process of computing OUT by IN of successors and inserting OUT into flow function to make IN again. If there is no loop in the input program, the algorithm ends with a single scan (one complete run of 'for' loop shown in Algorithm 1). Without loops, all basic blocks are affected only by their own successors, so the computation in the reverse order of DFS will be in order. However, if there is a loop, the basic block that comes first in the DFS order affects OUT value of the basic block that comes later. In this case, you will need to visit the basic block again to apply the effect of the updated OUT. For this reason, a while loop is needed in Algorithm 1, and then it has been designed of an iterative algorithm.

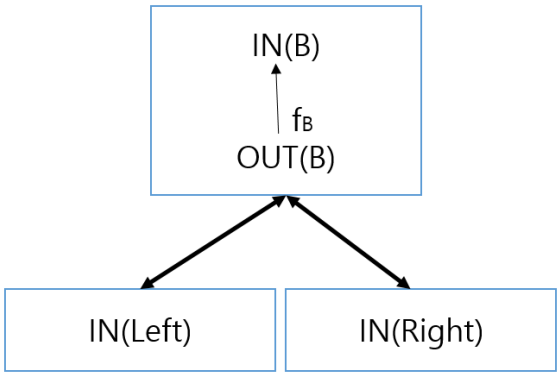

Figure 2. Flow equation

THE PROPOSED TECHNIQUE

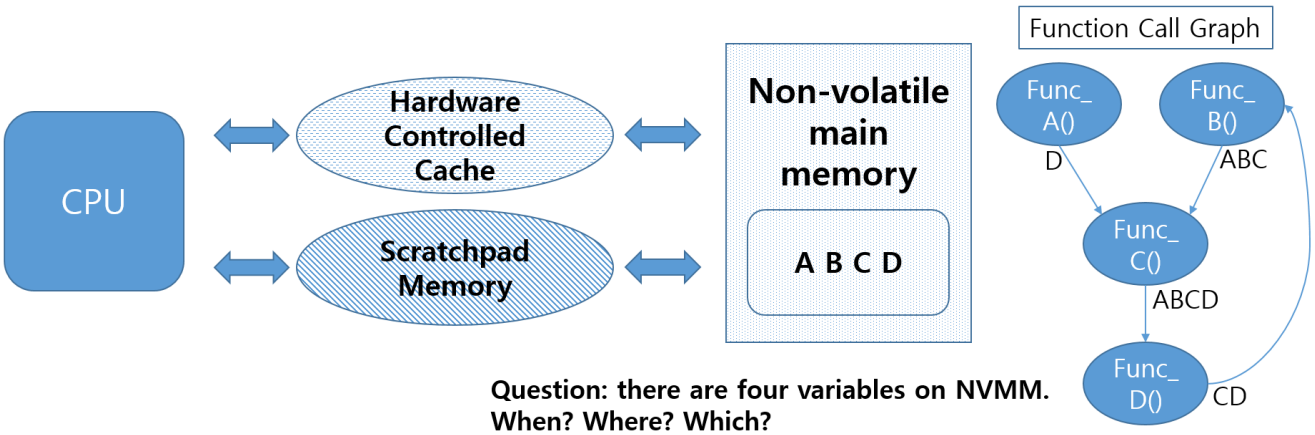

Figure 3. A motivation example

As shown in Figure 3, we use an example to describe a key idea in the data partitioning and assignment techniques proposed in this paper. It shows a requirement of the proposed technique that can take advantage of NVM and avoid the drawbacks of NVM in a hybrid cache architecture. Essentially, it calculates the minimum memory access cost and the corresponding optimal data allocation in the two caches are determined.

All data corresponding to the currently running process resides in the nonvolatile main memory, and now we need to determine whether a copy of this data will be created in the SPM or just copied from the HC (Hardware Controlled) cache memory to allow the read/write accesses. At making the decision, data that is frequently used and which is frequently updated should reside in the SPM. When such data placed on traditional cache, if a cache miss occurs in write operations, the write cost is too high. Let see an example based on the function call graph shown in Figure 3. In this graph, the functions Func_{A, B, C, D} carry out the calculation on the variables {A, B, C, D}. According to the program's function call flow, the optimal decision can be provided by those questions when/where/what. The question 'when' means the time of transfer of data, and 'where' means a placement either a SPM or a HC cache. The last question is what kind of data should be copied to SPM. They are key factors in determining the performance of a hybrid memory system. As shown in Figure 3, all data in the NVMM is basically managed by the traditional cache, through an efficient data selection mechanism, the selected data must be copied to the SPM. A dynamic method is used for SPM to obtain an optimal result, which described later in this section. When CPU executes

read/write operations, the execution cost is determined by the following parameters.

Table I. cost parameters

NVMM_write : the write cost of NVMM

NVMM_read : the read cost of NVMM

Cache_write : the write cost of traditional cache (with LRU policy)

Cache_read : the read cost of traditional cache (with LRU policy)

SPM_write : the write cost of SPM

SPM_read : the read cost of NVMM

MOVE_SPM_TO_NVMM : the cost of data transfer from SPM to NVMM (6.213ns)

MOVE_NVMM_TO_SPM : the cost of data transfer from NVMM to SPM (1.898ns)

MOVE_CACHE_TO_NVMM : the cost of data transfer from CACHE to NVMM (6.213ns)

MOVE_NVMM_TO_CACHE : the cost of data transfer from NVMM to CACHE (1.898ns)

Table II. Configuration of 45nm 16KB NVMM and SPM [17]

	Non-volatile RAM (based on STT-RAM)	Traditional cache	SPM
Area(mm)	0.018	0.1488	0.093
Read latency (ns)	0.813	(hit)1.085,(miss) 5.941	1.085
Write latency (ns)	5.128	(hit)1.085,(miss) 7.026	1.085
Read energy (pJ)	6.516	32.9424	20.589
Write energy (pJ)	22.65	23.2016	14.501
Leakage (mW)	0.606	12.6656	7.916

Our technique analyses a target program building a function call graph. The function call graph consists of functions attached with its live data generated from LVA analyzer. According to a function call graph, each node includes a certain set of live data, and they are candidates of selection procedures for HC cache and SPM placement. The proposed technique analyzes the candidates read/write access frequency according to their function call graph. Using such read/write information of the candidates, our data placing procedures are able to perform minimizing the cost function defined in Definition 1.

Definition 1: Minimize(Cost) = $\sum_{Call\_Graph(Functions)}$

$\sum_{i \in variables(S)}$ **NVMM_write(i) + NVMM_read(i) + Cache_write(i) + Cache_read(i) + SPM_write(i) + SPM_read(i) + MOVE_SPM_TO_NVMM(i) + MOVE_NVMM_TO_SPM(i) + MOVE_CACHE_TO_NVMM(i) + MOVE_NVMM_TO_CACHE(i)**

Table 1 shows the parameters needed to calculate the data access cost for the hybrid memory system described in Figure 3. The terms, MOVE_NVMM_TO_SPM, MOVE_CACHE_TO_NVMM, and MOVE_NVMM_TO_CACHE, and MOVE_ SPM_TO_NVMM respectively mean that the cost of data transfer from NVMM to SPM, the cost of data transfer from CACHE to NVMM, the cost of data transfer from NVMM to CACHE, the cost of data transfer from SPM to NVMM**.** Table 2 shows actual read/write latency and power consumption figures for the 16-KB NVMM and SPM memory systems in 45nm technology. In this paper, we propose a new method to improve performance and energy of the hybrid memory system by using SPM to place write intensive data, to separate the data from the NVMM, and to manage the other data by HC cache. The proposed technique is possible to optimize a key metric as either energy or performance. In our study, the choice is easy to perform by reflecting parameters in Table 2.

Our technique first constructs a function call graph of a target program to be analyzed, and then calculate the live range of each piece of data used along each node (function) of the graph. Live range is calculated by compiler static analysis described in Algorithm 1, and the number of read/write of each variable is summarized by profiling.

The objective function for optimizing the hybrid memory system is the same as minimizing the function shown in Definition 1. It determined an optimal data placement with minimizing read/write cost and its movement of each variable according to a function call graph. The final placement is calculated by using an exhaustive search to obtain the optimal result. For an example of a function call graph in Figure 3, it is necessary that the placement of the variables A, B, C, D be placed for each program execution step on the function call graph. The object of this study is optimizing the hybrid memory that uses NVMM as main memory and SPM as a part of cache, so it is necessary to calculate a solution that selects only data to be loaded on SPM in each execution step. In this case, the search algorithm that calculates the minimum cost based on the objective function is a depth-based exhaustive search algorithm shown in Algorithm 2 and illustrated in Figure 4.

Algorithm 2: Solution Search

Input : Data Set *base_S*, a graph K of all combinations with elements of *base_S*

Output : Data placement *opt_S*

```
Exaustive Search
    all elements K stored into a stack G
       K.push(v) // vertex v
    while K is not empty
       v = K.pop()
       if v is not labeled as visited
            label v as visited
            compute cost with v when it is stored in SPM
            for all edges from v to w in K.adjacentEdges(v)
do
                  K.push(w)
   Opt_S = MIN(cost)
End
```

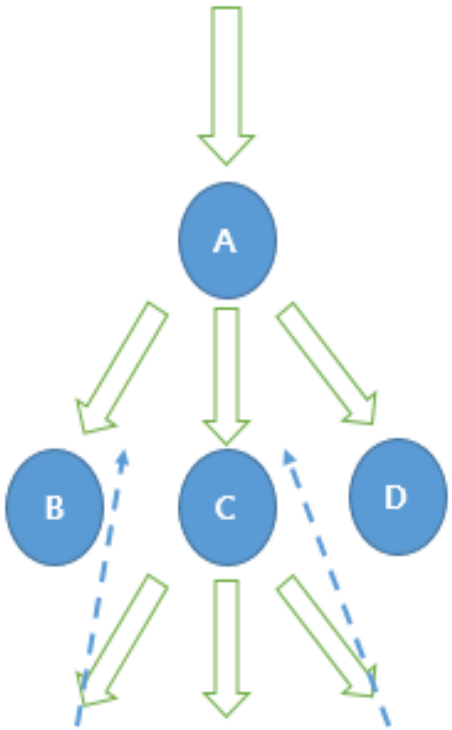

Figure 4. Exhaustive search with backtracking

For example, in the function call graph in Figure 3, let see the number of read/write of each ABCD variable, and the cost calculation as follows.

Table III. static/dynamic analysis results on the function call graph in Figure 3

Func_A() : D (read 1, write1)

Func_B() : A (read 1, write 1), B (read 1, write 1), C (read 1, write 1)

Func_C() : A (read 1, write 0), B (read 1, write 0), C (read 1, write 0), D (read 1, write 1)

Func_D() : C (read 1, write 0), D (read 1, write 1)

Live Range Analysis Result:

D : A() – D()

C : B() – D()

B : B() – C()

A : B() – C()

In the case of NVMM-CACHE, all data places on a traditional cache and NVMM.

Cost= NVMM_write(A,B,C,D) + NVMM_read(A,B,C,D) + Cache_write(A,B,C,D) + Cache_read(A,B,C,D) + MOVE_CACHE_TO_NVMM(A,B,C,D) + MOVE_NVMM_TO_CACHE(A,B,C,D)

The access patterns of variables in the function call graph are the same as A (): D, B (): A B C, C(): AB CD, D(): CD. Assuming a cache that can hold two variables, the data placement of the cache will be D A, B C, A B, C D, C D. It results to a cache hit once for the final CD access, and all previous accesses will be missed. Applying the actual parameters to the cost function in Definition 1, it is as follows: (with performance parameters with LRU policy).

Cost = Cost_A(0.813*2 + 5.128*1 + 2*1.085 + 2*1.085 + 2*6.213 + 2*1.898)

+ Cost_B(0.813*2 + 5.128*1 + 2*1.085 + 2*1.085 + 2*6.213 + 2*1.898)

+ Cost_C(0.813*2 + 5.128*1 + 3*1.085 + 2*1.085 + 2*6.213 + 2*1.898)

+ Cost_D(0.813*2 + 5.128*2 + 3*1.085 + 3*1.085 + 2*6.213 + 2*1.898)

The access cost of each variable is computed by the access type and access frequency of each variable analyzed along the function call graph. The access type and frequency for each variable are shown in Table III.

A: read 2 write 1, B: read 2 write 1, C: read 3 write 1, D: read 3 write 3 (only hit ‘CD’ last once).

The cost for each variable is calculated to A: 25.146, B: 25.146, C: 26.231, and D: 31.359. So in the example in Figure 3, the total cost of the memory system results to 107.882 ns.

Assuming that all variables are placed in the NVMM-SPM (performance optimization with SPM only), the total cost is calculated to 44.184 ns. The reason why the performance of HC cache only system provides more than twice is mainly due to high miss rate with data streaming application. However, simply increasing the size of the cache does not necessarily to improve performance result 107.882ns. Since the cache hit ratio is already over 95% in case of non-streaming application. Thus, there is a technique required to effectively utilize small SPM space to provide an optimum performance with data streaming application.

Cost= Cost_A (1.085 *2 + 1.085 *1 + 0.813 + 5.128)
+ Cost_B (1.085 *2 + 1.085 *1 + 0.813 + 5.128)
+ Cost_C (1.085 *3 + 1.085 *1 + 0.813 + 5.128)
+ Cost_D (1.085 *3 + 1.085 *3 + 0.813 + 5.128)

In this case, when a cache miss occurs, it is positive for the overall system performance to place such data as D which yields a large write overhead in the NVM, thus firstly placing into SPM. If variable 'C' has frequent read, even if a miss occurs with 'C', NVM provides low cost reload to the cache. Thus, it is preferable on the viewpoint of overall performance to place it into a hardware controlled cache. Using the exhaustive search technique, the optimal data placement is determined by the traditional cache {A C}, SPM {B D}. As a result, cache misses are eliminated, and an optimum performance (44.184 ns) is obtained.

The solution search algorithm finds the minimum cost by calculating the number of all cases to place each variable in SPM space and HC cache. First, it builds a search tree, and stores at each node the maximum performance (or energy) gain and the minimum gain on the objective function for this problem instance. The search scheme repeatedly (1) selects an unprocessed case of combinations with variables' placement, (2) processes the placement and then creates its other combinations as candidates of solution, and (3) propagates new max and min values through the tree and uses these values to select the next node. It performs this sequence of three stages until the search tree contains no more unprocessed placement combinations. The detailed execution of the three major steps is following.

The first step is to find the node to process next. The search algorithm selects a leaf placement by descending the search tree, starting at the root and taking the child at unobserved candidate solutions. Our implementation orders the child from left to right so that their values are non-decreasing with a priority queue.

The second step is to process and expand the node. For each of these unobserved nodes, maximum and minimum performance (or energy) gain on its objective function is obtained, and the best unobserved placement is chosen to branch on. The node is created and then processed and expanded in the same way. At each step, the set of nodes contributing to the maximum performance gain is stored to a solution set.

The third step is to propagate the new gain and prune the tree. Starting at the nodes just created and working up the tree to the root, the value of the maximum gain and the minimum gain are updated for each node.

As this stage assigns and reassigns gain, it checks to see if any node has one child whose maximum gain does not exceed the minimum gain of the other child. In such a case the solution of maximum can be no better than that of the minimum, so the node branch of maximum and all its descendants are removed from the tree. Finally, this search procedure generates a data placement to minimize the objective function as an optimal solution.

EXPERIMENTAL RESULTS

The proposed technique is implemented using LLVM [18], an open source compiler infrastructure. The LLVM-based compiler compiles some benchmark codes to generate an executable file and evaluates the optimized code using GEM5 simulator. In this experiment, the optimized code is called OPG (Optimized Group), and the original code will be referred to as ORG (Original Group). The OPG code is compiled by the LLVM compiler to improve the negative impact of write operations by providing data load code and write-driven data best fit in SPM / cache and optimizing variable placement. Benchmark programs and input files were mainly selected from the LLVM test suites used in some industry codes [19]. By using profiling, we were able to obtain the characteristics of a benchmark program, such as the frequency of accesses to a benchmark program. For the experiment evaluation, we used simulator setup with cache management strategy of [21]. The simulator targets one-level data cache on a single core processor. Cache parameters and memory parameters are taken from the modified CACTI [22].

The results of write-intensive variables' partitioning are illustrated in Figure 5. It shows the comparison of traditional data placement techniques using STT-RAM. The latency can be observed to be reduced or increased by a slight difference. Figure 6 illustrates energy consumption. The ORG default result is normalized to 1 and the resulting value of the OPG is shown as reduced from the baseline.

The proposed technique could reduce write accesses to NVMM. As a result, total latency can be improved by 6%. Therefore, it is promising that the proposed technique can improve the efficiency of the hybrid cache memory subsystems. Figure 6 shows that the overall energy saving result corresponding to the reduction of cache misses. As shown in the Figure, the proposed technique can achieve 5% energy saving by effective data rearrangement for the hybrid cache architecture.

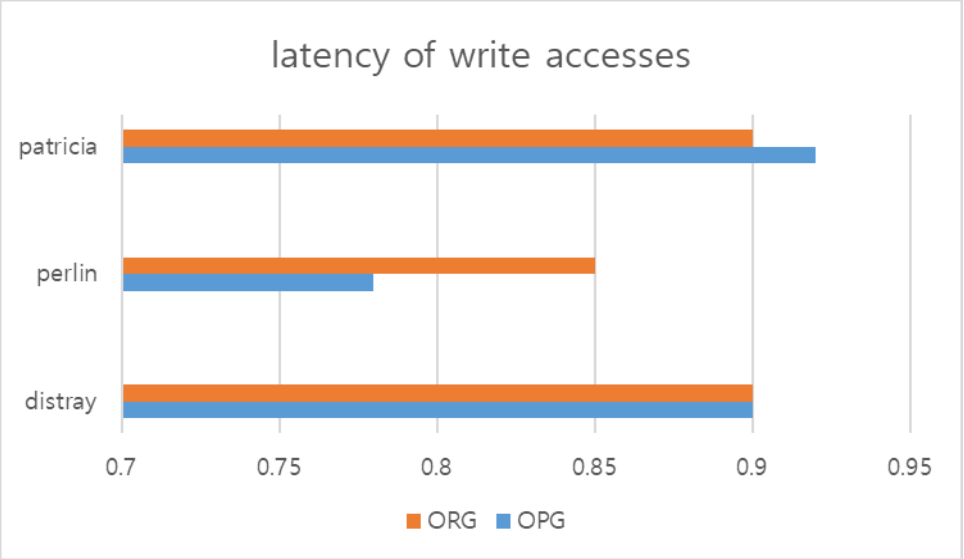

Figure 5. Result of total latency of write accesses

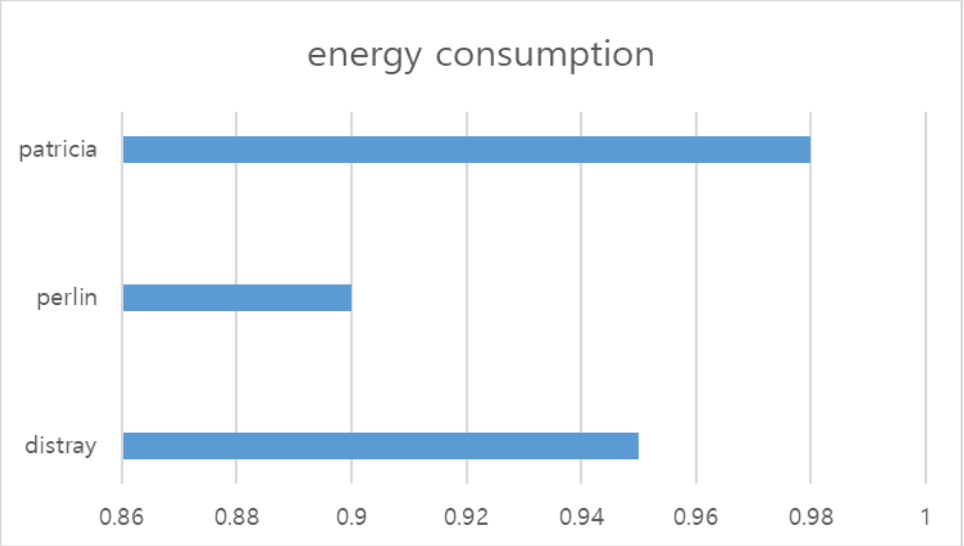

Figure 6. Result of the energy consumption

CONCLUSIONS

In this study, we introduce a technique that uses a compiler-based approach to improve energy consumption by rearranging variables in main memory space for SPM based the hybrid cache. To that end, we present a data flow based live range analysis technique, and search technique uses the static/dynamic analysis to minimize the cost to efficiently utilize a hybrid cache architecture. Specifically, traversing function call graph, rearrangement of local variables (stack data) is performed for each function, in order to maximize the benefits of the hybrid cache structure. The variables' placement are determined by the variables' live range and read/write frequency. In order to compensate for the drawbacks of NVM with write operations, a hybrid memory system has been proposed with a scratchpad memory (SPM) component. By partitioning write intensive data onto SPM, it can provide a high performance memory system with keeping NVM advantages, since SPM performs a write operation at high speed and low energy consumption. The experiment shows that the proposed technique is able to improve energy consumption 5% and execution time 6% in average.

ACKNOWLEDGMENTS

This research was supported by Unmanned Vehicles Advanced Core Technology Research and Development Program through the National Research Foundation of Korea(NRF), Unmanned Vehicle Advanced Research Center(UVARC) funded by the Ministry of Science, ICT and Future Planning, the Republic of Korea (2016M1B3A1A03937725).

REFERENCES AND NOTES

[1] P. Zhou, B. Zhao, J. Yang, and Y. Zhang, "A durable and energy efficient main memory using phase change memory technology," in Proc. 36th Annu. Int. Symp. Comput. Arch., 2009, pp. 14–23.

[2] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, "Architecting phase change memory as a scalable DRAM alternative," in Proc. 36th Annu. Int. Symp. Comput. Arch., 2009, pp. 2–13.

[3] L. Shi, C. J. Xue, J. Hu, W.-C. Tseng, and E. H.-M. Sha, "Write activity reduction on flash main memory via smart victim cache," in Proc. 20th Symp. Great Lakes Symp. VLSI, 2010, pp. 91–94.

[4] W.-C. Tseng, C. J. Xue, Q. Zhuge, J. Hu, and E. H.-M. Sha, "Optimal scheduling to minimize non-volatile memory access time with hardware cache," in Proc. 18th IEEE/IFIP VLSI Syst. Chip Conf., Sep. 2010, pp. 131–136.

[5] J. Hu, C. J. Xue, W.-C. Tseng, Q. Zhuge, and E. H.-M. Sha, "Minimizing write activities to non-volatile memory via scheduling and recomputation," in Proc. 8th IEEE Symp. Appl. Specific Process., Jun. 2010, pp. 7–12.

[6] J. Hu, C. J. Xue, W.-C. Tseng, Y. He, M. Qiu, and E. H.-M. Sha, "Reducing write activities on non-volatile memories in embedded cmps via data migration and recomputation," in Proc. Design Autom. Conf., 2010, pp. 350–355.

[7] J. Hu, C. J. Xue, Q. Zhuge, W.-C. Tseng, and E. H.-M. Sha, "Toward energy efficient hybrid on-chip scratch pad memory with nonvolatile memory," in Proc. Design Autom. Test Eur. Conf., 2011, pp. 136–141.

[8] J. Hu, W.-C. Tseng, C. Xue, Q. Zhuge, Y. Zhao, and E.-M. Sha, "Write activity minimization for nonvolatile main memory via scheduling and recomputation," IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 30, no. 4, pp. 584–592, Apr. 2011.

[9] C. J. Xue, Y. Zhang, Y. Chen, G. Sun, J. J. Yang, and H. Li, "Emerging non-volatile memories: Opportunities and challenges," in Proc. 9th Int. Conf. Hardw./Softw. Codesign Syst. Synth., Oct. 2011, pp. 325–334.

[10] T. Liu, Y. Zhao, C. J. Xue, and M. Li, "Power-aware variable partitioning for DSPs with hybrid PRAM and DRAM main memory," in Proc. Design Autom. Conf., 2011, pp. 405–410.

[11] J. Li, L. Shi, C. J. Xue, C. Yang, and Y. Xu, "Exploiting set-level write non-uniformity for energy-efficient NVM-based hybrid cache," in Proc. ESTImedia, 2011, pp. 19–28.

[12] Y. Huang, T. Liu, and C. J. Xue, "Register allocation for write activity minimization on non-volatile main memory," in Proc. 16th Asia South Pacific Design Autom. Conf., 2011, pp. 129–134.

[13] P. Mangalagiri, K. Sarpatwari, A. Yanamandra, V. Narayanan, Y. Xie, M. J. Irwin, and O. A. Karim, "A low-power phase change memory based hybrid cache architecture," in Proc. 20th Symp. Great Lakes Symp. VLSI, 2008, pp. 395–398.

[14] X. Dong, X. Wu, G. Sun, Y. Xie, H. Li, and Y. Chen, "Circuit and microarchitecture evaluation of 3D stacking magnetic RAM (MRAM) as a universal memory replacement," in Proc. 45th Annu. Design Autom. Conf., 2008, pp. 554–559.

[15] Y. Joo, D. Niu, X. Dong, G. Sun, N. Chang, and Y. Xie, "Energyand endurance-aware design of phase change memory caches," in Proc. Design Autom. Test Eur. Conf., 2010, pp. 136–141.

[16] X. Wu, J. Li, L. Zhang, E. Speight, R. Rajamony, and Y. Xie, "Hybrid cache architecture with disparate memory technologies," in Proc. 36th Annu. Int. Symp. Comput. Arch., 2009, pp. 34–45.

[17] Peng Wang, Guangyu Sun, Tao Wang, Yuan Xie, Jason Cong, "Designing scratchpad memory architecture with emerging sttram memory technologies", IEEE International Symposium on Circuits and Systems (ISCAS), 2013.

[18] Lattner, C., Adve, V., "LLVM: A compilation framework for lifelong program analysis & transformation", In Proceedings of the International Symposium on Code Generation and Optimization, 2004, 75-86.

[19] Guthaus, M., Ringenberg, J., Ernst, D., Austin, T., Mudge, T., Brown, R., "Mibench: A free, commercially representative embedded benchmark suite", IEEE International Workshop on Workload Characterization, 2001, 3-14

[20] Luk, C.-K., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., Wallace, S., Reddi, V.J., Hazelwood. K., "Pin: building customized program analysis tools with dynamic instrumentation", In Proceedings of the ACM SIGPLAN conference on Programming language design and implementation, 2005, 190-200.

[21] Li, J., Xue, C., Xu, Y., "Stt-ram based energy-efficiency hybrid cache for cmps", IEEE/IFIP 19th International Conference on VLSI and System-on-Chip, 2011, 31-36.

[22] Muralimanohar, N., Balasubramonian, R., Jouppi, N., "Optimizing nuca organizations and wiring alternatives for large caches with cacti 6.0", In Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 40, 2007, 3-14.