ISSN 1330-3651 (Print), ISSN 1848-6339 (Online) https://doi.org/10.17559/TV-20200826040830
Original scientific paper

Improvement Energy Efficiency for a Hybrid Multibank Memory in Energy Critical Applications

Jungseok CHO, Jonghee M. YOUN, Doosan CHO*

Abstract: High performance, low power multiprocessor/multibank memory system requires a compiler that provides efficient data partitioning and mapping procedures. This paper introduced two compiler techniques for the data mapping to multibank memory, since data mapping is still an open problem and needs a better solution. The multibank memory can be consisted of volatile and non-volatile memory components to support ultra-low powered wearable devices. This hybrid memory system including volatile and non-volatile memory components yields higher complexity to map data onto it. To efficiently solve this mapping problem, we formulate it to a simple decision problem. Based on the problem definition, we proposed two efficient algorithms to determine the placement of data to the multibank memory. The proposed techniques consider the characteristic of the non-volatile memory that its write operation consumes more energy than the same operation of a volatile memory even though it provides ultra-low operation power and nearly zero leakage current. The proposed technique solves this negative effect of non-volatile memory by using efficient data placement technique and hybrid memory architecture. In experimental section, the result shows that the proposed techniques improve energy saving up to 59.5% for the hybrid multibank memory architecture.

Keywords: low power; memory system; optimizing compiler; system software; wearable IoT devices

1 INTRODUCTION

As various portable electronic devices are being smaller in size with high performance, the importance of low power design is becoming more important in battery critical wearable devices. Design of wearable devices must consider cost, energy, performance, and size to meet market needs. The energy is the major component to determine cost, performance and device size, since it determines battery capacity. Thus, low power concern becomes an important factor in its design. Fig. 1 shows increasing of leakage power with a system chip fabrication process in nanoscale. In the deepening fabrication technology, the leakage power is being higher and higher than before with current transistors.

Most related works focus on memory components which consume about 40% of operation power in a system chip. Some of those works chose a non-volatile memory component, since it provides nearly zero leakage power. Such works presented techniques to avoid the leakage power problem. However, such nonvolatile memory has a critical disadvantage. It is that nonvolatile memory consumes more power to perform write operation than its read operation. This study proposes two techniques to mitigate the disadvantage of the nonvolatile memory component. In this work, low power design of a wearable system focuses on system software that controls function of hardware components. Similarly, many researches are being conducted on the control of the memory system that consumes about 40% of system on a chip. In the memory system researches, low power optimizations can be divided into two groups. The first group is focused on code restructuring like loop transformations [1-3]. The second group is data optimization for memory hierarchy [4, 5]. The code optimizations like loop transformations (such as loop tiling, fusion and unrolling) improve performance and power consumption for on/off memories. The data optimization techniques (such as data block prefetching) optimize data fitting to the shape of memory hierarchy. The proposed techniques in this study belong to the second group.

In this paper, the proposed techniques optimize software for a multibank memory system [6], which is adopted in many wearable systems that provide low cost/high performance. Since multibank memory uses several small memory cells connected to each other, the operation power is relatively smaller than that of the same size large cell memory. The multibank memory provides high performance by providing simultaneous accesses to data with multiple read/write ports. This is the efficient memory design approach that is usually adopted when low cost/high performance is required. In order to provide lesser operating power, multibank memories should be designed in a hybrid form. The hybrid memory can consist of a non-volatile memory and a traditional volatile memory. The disadvantage of the hybrid multibank memory is the needs to analyse data access patterns to efficiently determine data placement on the multibank. This data placement problem is determined by several data/control flow analyses of an optimizing compiler. The compiler can analyse a given program source code to determine the access patterns of data to efficiently determine their placement on the multiple memory banks.

Commercialized compilers for TI's TMS320x, ATL's ARC processor, and ADRES architecture [7-9], which adopt multibank memory architecture, are commercially available and confidential. Such existing technologies cannot be used for startup manufacturers. In the case of startup's product, the development cost increases exponentially because of technology license fee for the new wearable devices. This study helps the startups to choose the traditional hardware architecture and at the same time provide convenience to develop new wearable devices and their system software at low cost. This paper is organized as follows. The next chapter shows the background, and Chapter 3 explains the technical details proposed in this work. In Chapter 4, the experimental results are examined. In this study, we introduced a static data placement technique and a dynamic technique, and evaluated the two techniques proposed in the experiments. Chapter 5 illustrates related works and finally conclusion is described in Chapter 6.

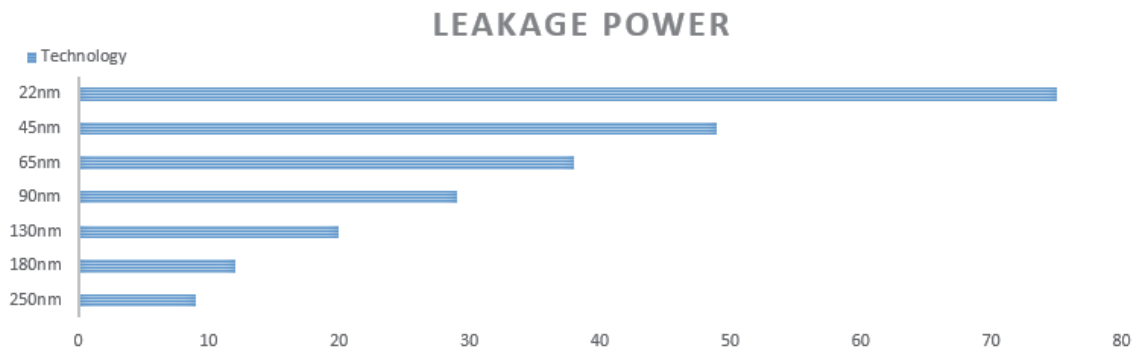

Figure 1 Leakage power with fabrication technologies in nanoscale

2 BACKGROUND OF A HYBRID MULTIBANK MEMORY AND ITS COMPILER

As mobile systems, cloud computing and wearable devices have becoming more prevalent; research and development on low-power embedded systems have rapidly increased along with battery technology. Memory system requires more than 40% power dissipation in an embedded system [10]. Thus, many studies have been conducted to reduce power consumption in memory systems [10-12]. The technique introduced in [11] is for placing blocks of code and provides several measurements of energy consumption. With various sizes of SRAM, experiment results of the work show reduction of energy by an average of 31.3% in benchmarks. The technique introduced in [12] is the only technique that considers heap data and applies SPM management at runtime. Its experimental results show that the technique can reduce the average power consumption by 39.9% for a fixed SPM of the same size with 5% of the total data size in the comparison between all heap variables placed in DRAM and global/stack data placed in SPM.

In the previous studies, researchers focused on energy saving by using data placement. Frequently used data can be placed onto low powered memory components. However, the use of data usage optimization is not sufficient to save energy consumption in a system chip. In addition to that, battery technology does not meet the market requirement to increase its capacity and size. Thus, it is essential to support new memory architecture to consume lower energy. This study presents compiler optimization techniques that support a hybrid multiple bank memory architecture, which consists of non-volatile and traditional volatile memories.

Nonvolatile memory bank has the advantage of high density and nearly zero leakage power, while it has a disadvantage for the write operation [13]. The disadvantage of nonvolatile memory is that it consumes more energy in execution of the write operation, and it also takes longer latency. Thus, data placement optimization is mandatory for the hybrid memory to make lower the negative effect of the write operation [14]. To overcome this negative effect, many studies proposed techniques to add hardware controller, operating system support and/or compiler support [15-17]. Additional hardware consumes more energy, thus it is not a viable option in mobile devices. To efficiently utilize a hybrid memory architecture, an operating system or a compiler must be able to determine data placement on volatile or nonvolatile memory banks. It is a well known problem called data partitioning and mapping [18]. Normally, a compiler determines data partition and its placement [19]. In this work, we proposed two compiler techniques that provide data partition and placement onto traditional volatile memory and nonvolatile memory banks.

In the hybrid memory, it is important to reduce disadvantage of non-volatile's write operation by allocation of the write-intensive data to traditional volatile memory and the read-intensive data to non-volatile memory. The proposed data placement techniques use profiling data for the whole variables in the given target program.

The first proposed technique is a static approach to determine data placement in the hybrid memory banks. Static approach means that the placement of data is determined at compile time. Therefore, there is no data movement during program execution. The second technique is a dynamic approach to determine data placement. The dynamic technique allows data transfers during program execution. It maximizes the advantage of the hybrid memory space. As a result, dynamic technique leads to a better result than the static one. In the experimental section, the comparison results with the applied proposed techniques are illustrated.

Fig. 2 shows a simple functional sequence of a traditional compiler backend. The compiler backend works in the following order: instruction selection, instruction scheduling, register allocation, instruction scheduling, and code emission. In multiprocessor systems, a global scheduler is added to the workflow behind of instruction scheduler. If the functions are implemented in a modular way, they can be used as a library. Thus, the design of a compiler framework maximizes its reusability. The proposed technique is designed for a traditional compiler. It can be used in the same way as a traditional compiler optimization, and the optimization order can be adjusted to its needs as shown in Fig. 2.

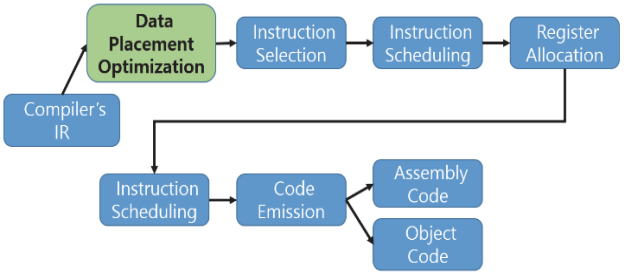

Figure 2 The proposed technique in the compiler's backend

There are some similar studies on a hybrid memory. Awasthi et al. introduced a technique that utilized DRAM and NVM as a main memory [20]. It saves energy by substituting flash memory for a main memory in specific applications. Park et al. introduced a technique to manage the operation power of a hybrid memory using DRAM and PRAM [21]. Hassan et al. introduced a technique to reduce total energy consumption using Scratch Pad Memory with DRAM [14]. One of the finest technologies for traditional multibank memory is presented in the work of Cho [22] on the asymmetric memory banks of TI architecture. The work is limited to a memory platform consisting of SRAM cells in an asymmetric memory bank, so it is difficult to apply to a hybrid form's multibank memory. Although the existing studies solved different problems in their own works, none of the papers addressed the problem of data placement in the hybrid multibank memory. To the best of our knowledge, this is the first work to solve the data placement problem in the hybrid multibank memory architecture.

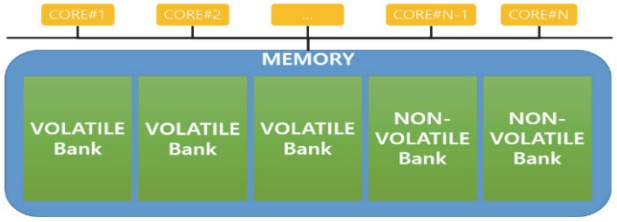

Figure 3 Multiprocessor and multibank memory architecture

3 THE PROPOSED TECHNIQUES

Fig. 3 shows the target memory architecture. It is the same as the traditional memory architecture, except that the memory space is divided into volatile memory banks (VM) and nonvolatile memory banks (NVM). As shown in the figure, read intensive variables should be placed to NVM to take all advantage of NVM. It maximizes the energy efficiency from the hybrid multibank memory system. The data partitioning problem to place data for the hybrid multibank memory can be determined by the equation shown in Fig. 4.

The function in Fig. 4 represents making decision to place data for selection VM and NVM in the hybrid memory. The metrics for selection VM and NVM in hybrid memory are defined to select the one that consumes lower energy based on the number of read and write operations profiled for each data block (variable). To take advantage of the hybrid memory, the amount of energy consumed when allocating to NVM and VM for each variable should be calculated. The amount of read energy of data_block (*i*) is calculated by Energy_READ function and the amount of write energy calculated by Energy_WRITE function. They calculate the total amount of NVM_Energy, and compare it with VM_Energy, and then make the final choice for variable's placement with the lower energy consumption.

The hybrid memory placement decision is determined by the hybrid decision function in Fig. 4. Specifically, the placement of the data variables must be determined taking into account the amount of total energy consumption, overhead and capacity of VM/NVM. The function NVM_Energy returns the amount of energy consumption if data_block (*i*) is placed onto a NVM bank. The function VM_Energy returns the amount of energy consumption if data_block (*i*) is placed onto a VM bank. The function OVERHEAD represents the amount of energy consumption of transfer data_block (*i*) from/to NVM/VM. The CHOOSE_MINIMUM function returns a placement result of data_block(i) that is calculated from the calculation of NVM_Energy and VM_Energy.

```
// There are profiled read/write operation counts for data block(i)
HYBRID_DECISION:
if ( read_instruction(data_block(i)) > write_instruction(data_block(i)) )
{
  if( IsAvailable_Space())
  {
    CHOOSE_MINIMUM(
    NVM_Energy (Energy_READ(data_block(i))
                + Energy_WRITE(data_block(i))
                  + OVERHEAD(Transfer_Energy (data_block(i)))),
    VM_Energy (Energy_READ(data_block(i))
                + Energy_WRITE(data_block(i))
                  + OVERHEAD(Transfer_Energy (data_block(i))))
    )
  }
} else
   return VM_PLACEMENT()
```

Figure 4 The hybrid placement decision

After determining the hybrid choice of NVM and VM, the whole variables are partitioned to NVM placement or VM placement. And then, the proposed techniques optimize data locality within the NVM/VM placement decision. The data placement result of our technique should provide higher data locality to improve system performance. To that end, we define a WORKING_WINDOW that provides a time slot to improve data locality. The WORKING_WINDOW supports data locality in temporal perspective. Otherwise, our technique obtains spatial locality to place such data used in WORKING_WINDOW to the same memory bank. The WORKING_WINDOW size can be determined by a cache page size, since it is designed with consideration of temporal and spatial data locality. On the other hand, our target system's programming model is based on a multitask model. It represents there is no shared variables, thus there is no concern about variable's consistency in this study.

To obtain variables information used in WORKING_WINDOW, it is needed to analyse a live range of variables at a certain point in program execution. Live range represents the time between definition of a variable and the end of its use. The variables should place the same memory bank when they have intersection of their live range. It provides better data locality in spatial and temporal. Algorithm 1 shows the live range analysis. The function Set_Live_Range returns live range of data_block (*i*). Begin_Live_Range represents the live range's beginning point of data_block (i). End_Live_Range represents the live range's end point of data_block (*i*).

```
Algorithm 1: Live Variable Analysis
Set_Live_Range(data_block(i))
{
   Begin_Live_Range = Find_Definition_of(data_block(i));
   End_Live_Range = Find_Last_Use_of(data_block(i));
   Return Live_Range(Begin_Live_Range, End_Live_Range);
}
```

Figure 5 The live variable analysis

This code is an example of the live range analysis. Live range of variable B is from line 2 to line 3. The definition of variable B is at line 2, and the last use is at line 3. The live range of variable A is from line 1 to line 4. Variable A and B should be placed on the same memory bank to make

higher data locality. Therefore, the size of WORKING_WINDOW is two in this example.

```
1          A = 1
2   L1 :   B = A + 1
3          A = B * 2
4          if A < 23  goto L1
```

```
MULTIBANK_DECISION:

WORKING_WINDOW_SIZE = Set_WorkingWindow()

IF(GET_DATA_BLOCK( WORKING_WINDOW_SIZE ) )
  IF(IsAvailable_Space())
     ALLOCATE_SAME_BANK ( DATA_BLOCK )
ELSE
     ALLOCATE_DIFFERENT_BANK ( DATA_BLOCK )
```

Figure 6 The multibank decision function

In Fig. 6, function GET_DATA_BLOCK brings data blocks having spatial and temporal locality to the same memory bank. The other data should be placed to different memory bank when there is no data locality.

The decision functions of hybrid memory selection and multibank selection are key components to the proposed static data placement algorithm in this work. The algorithm shows how to determine data placement in a static way to consider energy saving and memory bank space. It can be seen that the algorithm first determines the hybrid decision and then performs the multibank decision in the hybrid multibank memory consisting of VM/NVM.

Algorithm 2 shows the static data placement algorithm for a hybrid memory with VM and NVM multibank memory. It takes input data as the whole variables of a given target program, which is provided by profiler. The output is the result of data placement for the hybrid multibank memory.

NVM's read instruction has less energy consumption per access than the same size of VM. To improve energy consumption of a system, read intensive data should be placed to NVM as long as possible. Algorithm 2 determines the placement for maximizing energy saving using NVM memory bank. On the other hand, write intensive data should be placed on VM, since NVM's write instruction consumes more energy than the same size of VM. In the algorithm, each variable is defined as a data block.

```
ALGORITHM 2: STATIC DATA PLACEMENT APPROACH
INPUT : ALL DATA IN A GIVEN PROGRAM
OUTPUT : DATA PLACEMENT DECISION FOR A VM and NVM HYBRID MEMORY

HYBRID_DECISION( DATA_BLOCK )
{
  FOR_ALL( DATA_BLOCK(i) )
    IF( (NVM_SPACE > NVM_OCCUPIED_DATA_SIZE) && (VM_SPACE > VM_OCCUPIED_DATA_SIZE) )
       CHOOSE_MINIMUM( NVM_Energy(), VM_Energy())
}

MULTIBANK_DECISION( DATA_BLOCK )
{
  WORKING_WINDOW_SIZE = Set_WorkingWindow()
  FOR_ALL( DATA_BLOCK(i) )
    IF(GET_DATA_BLOCK( WORKING_WINDOW_SIZE ) )
      IF(IsAvailable_Space())
        ALLOCATE_SAME_BANK ( DATA_BLOCK )
    ELSE
        ALLOCATE_DIFFERENT_BANK ( DATA_BLOCK )
}

STATIC_DATA_PLACEMENT(DATA_BLOCK)
{
  FOR_ALL( DATA_BLOCK(i) )
     IF ( AVAILABLE_SPACE() )
        PUT( DATA_BLOCK(i), RESULT( HYBRID_DECISION, MULTIBANK_DECISION ) )
}
```

Figure 7 The static data placement algorithm

In Fig. 7, algorithm 2 makes the decisions that determine the final placement for data block *i*. In order to maximize energy saving using the advantage of NVM, read intensive data have higher priority placing on it. To that end, the placement procedure checks availability of NVM space. Write intensive data and other remaining data should be placed onto VM memory bank. Algorithm 2 is a static data placement technique, thus, the result of data placement is not changed during program execution. As shown in Fig. 7, the execution order of the overall workflow is as follows:

1. In HYBRID_DECISION (DATA_BLOCK) function, the read intensive data should be placed to NVM memory bank to maximize energy saving.
2. MULTIBANK_DECISION (DATA_BLOCK) function improves spatial and temporal data locality to exploit live range information. It determines whether the data blocks are placed to different memory bank or to the same bank. When a certain memory bank is full, then they should be placed to the other available bank in the same hybrid memory decision.
3. STATIC_DATA_PLACEMENT (DATA_BLOCK) function determines the physical memory address of the data block placed to a certain memory bank.

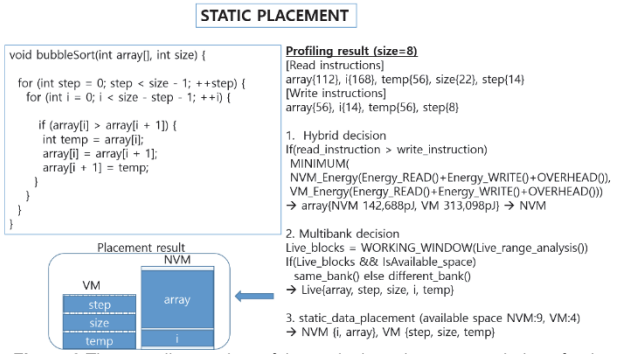

Figure 8 The overall procedure of the static data placement technique for the bubble sort code

Fig. 8 shows an example of the static data placement technique. The static technique does not change the determined data placement from the program's beginning of its execution to its completion, because the data placement is determined at compile time. In the dynamic technique, since placements of data blocks change dynamically during program execution, movement overhead is added due to the addition of move instructions according to the data movement. Because of the limited memory space, dynamic techniques usually provide a better result for the hybrid memory system despite the additional transfer overhead. The experiment describes this in Section 4.

The example in Fig. 8 shows overall procedures of the static placement technique applied to a bubble sort code. In the case of sort size 8, the number of read/write for data blocks (variables) was obtained by profiling and the profiled information is shown in the upper right of Fig. 8. A hybrid decision is first made according to the procedure of the static technique shown in Fig. 7. NVM energy and VM energy are calculated with the profiled information and energy consumption per a memory access. In this example, VM's read/write energy consumptions are 2396.16 pJ/798.72 pJ, NVM's read/write energy

consumptions are 798 pJ/952 pJ. Therefore, in the case of *array* variable, it executes 112 reads and 56 writes. By the hybrid decision function, NVM_Energy and VM_Energy are calculated to 142,688 pJ and 313,098 pJ respectively. NVM provides lower energy consumption for array, thus, it is placed into NVM. With the same procedures, *i* is placed on NVM, and *temp* is placed on VM because it has the same number of reads/writes. *step* and *size* are placed to VM because there is no available space in NVM with size 9, even though the number of reads is larger than writes.

Second, in the multibank decision, the live range of each data block should be analysed. As shown in the code at the top left of Fig. 8, it can be easily confirmed that the definition and use of all data blocks overlap.

Therefore, all data blocks should be placed to the same memory bank in their assigned memory. With a data block holding the largest energy saving, *i* is placed to the first bank of NVM. Next, array tries to be placed to the same memory bank of NVM. Since NVM consists of two memory banks of size 5 and 4, three elements of *array* are placed to the same bank of *i*, and the rest of *array* is placed to the other available memory bank. *Temp*, *step* and *size* are all placed to the same memory bank of VM.

In Fig. 9, algorithm 3 shows the proposed dynamic data placement algorithm. It has the same decision functions of HYBRID_DECISION, MULTIBANK_DECISION of Algorithm 2, and it has a unique function of DYNAMIC_DATA_PLACEMENT. Algorithm 3 uses timely ordered data blocks generated by profiling. HYBRID_DECISION function generates a result of data block placement for NVM or VM with all variables. For the data block placed to NVM/VM, MULTIBANK_DECISION function determines whether they should be allocated to the same memory bank or to a different bank. The final step is to execute DYNAMIC_DATA_PLACEMENT function. It determines physical memory addresses for all data blocks. Graph colouring algorithm leads to this physical memory placement.

```
ALGORITHM 3: DYNAMIC DATA PLACEMENT (BASED ON GRAPH COLORING APPROACH)
INPUT1 : INTERFERENCE GRAPH for ALL DATA BLOCKS in a GIVEN PROGRAM
INPUT2 : the NUMBER of COLOR in PARTITIONED NVM SPACE
OUTPUT : DATA PLACEMENT RESULT for a NVM and VM HYBRID MULTIBANK MEMORY

HYBRID_DECISION( DATA_BLOCK )
{
  FOR_ALL( DATA_BLOCK(i) )
    IF( (NVM_SPACE > NVM_OCCUPIED_DATA_SIZE) && (VM_SPACE > VM_OCCUPIED_DATA_SIZE) )
      CHOOSE_MINIMUM( NVM_Energy(), VM_Energy())
}

MULTIBANK_DECISION( DATA_BLOCK )
{
  WORKING_WINDOW_SIZE = Set_WorkingWindow()

  FOR_ALL( DATA_BLOCK(i) )
    IF(GET_DATA_BLOCK( WORKING_WINDOW_SIZE ) )
      IF(IsAvailable_Space())
        ALLOCATE_SAME_BANK ( DATA_BLOCK )
    ELSE
        ALLOCATE_DIFFERENT_BANK ( DATA_BLOCK )
}

DYNAMIC_DATA_PLACEMENT(DATA_BLOCK)
{
  WHILE ( !END == VISIT(INTERFERENCE_GRAPH) )
    IF( AVAILABLE_COLOR( ) == TRUE )
        CANDIDATE_NODE = VISIT( INTERFERENCE_GRAPH )
        IF( EDGE_CHECK(CANDIDATE_NODE) )
            ASSIGN_DIFFERENT_COLOR( CANDIDATE_NODE )
        ELSE
            ASSIGN_SAME_COLOR( CANDIDATE_NODE )
}
```

Figure 9 The dynamic data placement algorithm

The overall execution procedure of Algorithm 3 is as follows:

1. In HYBRID_DECISION (DATA_BLOCK) function, the read intensive data are placed to the limited NVM memory space with maximizing the energy saving.
2. MULTIBANK_DECISION (DATA_BLOCK) function improves temporal and spatial data locality. With live range analysis, data blocks should be placed onto the same memory bank or different memory bank. When a particular bank is full, data blocks are placed to the available memory bank in VM or NVM.
3. DYNAMIC_DTA_PLACEMENT (DATA_BLOCK) function determines the physical placement of the variables placed to each memory bank. At this time, each memory bank is divided into fixed size of virtual registers, and the physical placements of the variables are determined by a graph colouring algorithm. By using the graph colouring algorithm, a set of placements of data blocks can be changed during the program execution. As a result, the hybrid multibank memory can be used more efficiently.

As shown in Fig. 9, the proposed dynamic technique is designed based on a graph colouring technique. The goal of graph colouring technique is to efficiently use limited memory space. Traditionally it is used for register allocation algorithm [23]. It is designed to allocate a small number of registers to as many variables as possible. In the dynamic data placement function, the number of colours is the number of registers of the register allocation technique, and the number of variables is the target for the coloring problem of the register allocation technique.

For this colour allocation procedure, the live range of the variables should be calculated. Live variables are calculated with their used range in the program execution so that variables should not be assigned to the same colour. The live range of a variable can be represented by a node of a graph, which is called interference graph. The node can be colorized based on the interference graph that indicates live range overlap information between nodes [23]. Specifically, nodes in the graph represent variables, and variables with live range overlapping have edges between such nodes. The graph colouring procedure cannot allocate the same colour to the nodes having edges. At the end of this colouring procedure, all nodes having edges have a different colour if there are available colours. It means all variables have their own memory space (colour) on the hybrid multibank memory.

In order to use the graph colouring technique in the hybrid multibank memory, a certain partitioned space (colour) of the memory space to be allocated must be defined. This is defined by profiling the given program. For example, if the profiling result shows that the given program uses 80% of 32 bytes data blocks and 20% of 16 bytes data blocks, the memory space should be divided into 80% of 32 bytes partitions and 20% of 16 bytes data block partitions for the colouring procedure.

If there is a 1024 bytes of memory space, the 80% of space (820 bytes) is divided into twenty five (colours) 32-byte spaces to allocate (colouring) the variables. The remaining 204 bytes are divided into thirteen (colours) 16 bytes to allocate (colouring), and apply the graph colouring by constructing the interference graph for the variables used in the given program. As a result, the limited memory

space can be used more efficiently with the graph colouring.

By using the control flow analysis and data flow analysis of a compiler analyzer, the live range of all data variables can be obtained with the technique [24]. Since the analysis technique of the compiler is not the purpose of this study, it is described how to use the interference graph and graph colouring [25] in the proposed dynamic technique, except the data/control flow analysis [26].

The basic idea of live range analysis is to summarize when the variable was defined and when it was finally used. The operation of live range analysis is quite straight forward. To achieve optimal data placement or global optimization, it is necessary to do a live range analysis on the whole procedures, not a single basic block. To that end, data flow information is needed that collects by a compiler process called data flow analysis. The live range information for the whole data variables can be obtained by the technique [24]. Algorithm 3 calculates live variables at each basic block in the given program. Based on this live range information, an interference graph can be constructed to colorize memory space which partitioned profiled memory sizes.

Fig. 10 shows the live ranges of the four variables A, B, C, D and their interference graph. Variables A and D have no edge in the interference graph because their live ranges do not overlap. The live range of variable A overlaps with B and C, thus, they have edges in the interference graph. The live range of variable B overlaps with variables A, C, D. The live range of variable C overlaps with variables A, B, D. Thus, they have edges to all other nodes. The live range of variable D overlaps with variables B and C. Variables A and D have no overlap and no edge. They can have the same colour by the graph colouring approach. Thus, it needs three colours for the four variables. This represents that variables A and D can be placed to the same memory space in a different time. It provides efficient use of the hybrid memory space.

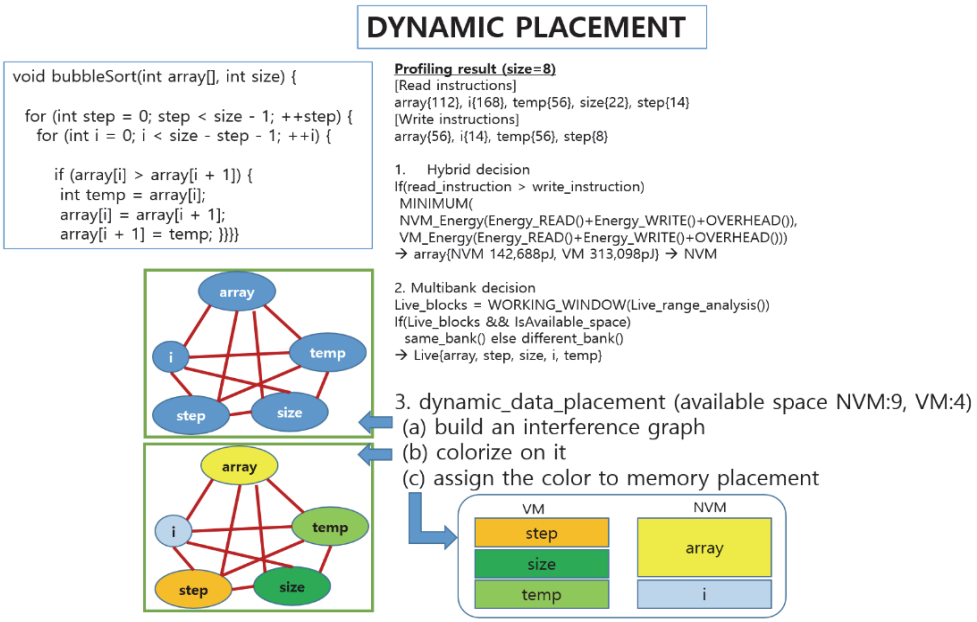

Figure 11 The overall procedure of the dynamic data placement technique for the bubble sort code example

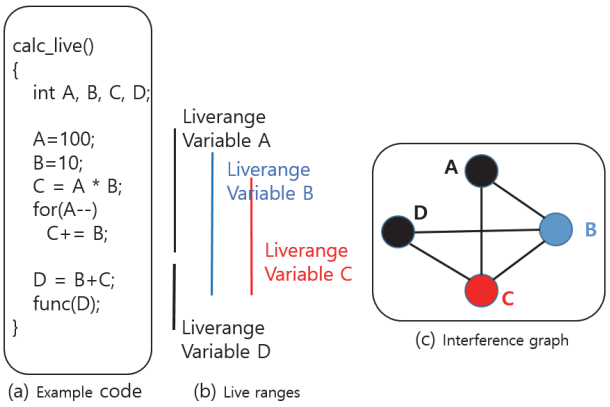

Figure 10 An interference graph generated by the live range information

The graph colouring algorithm allows all nodes in the interference graph to be assigned a minimum number of colours. NVM address space is partitioned into small partitioned sizes which are determined by profiling. The number of partitioned space is the total number of colours in the proposed algorithm 3. By doing so, read intensive data blocks are placed to NVM space.

The example of Fig. 11 shows an overall procedure of the dynamic data placement technique applied to the bubble sort code. In the case of sort size 8, the number of read/write for data blocks (variables) was obtained by profiling and the profiled information is shown in the upper right of Fig. 11. First, the hybrid decision should be determined by the technique in Algorithm 3. NVM energy and VM energy are calculated with the profiled information and energy consumption per a memory access. In this example, VM's read/write energy consumptions are 2396.16 pJ/798.72 pJ, NVM's read/write energy consumptions are 798 pJ/952 pJ. Therefore, in the case of *array* variable, it executes 112 reads and 56 writes. By the hybrid decision function, NVM_Energy and VM_Energy are calculated to 142,688 pJ and 313,098 pJ respectively. NVM provides lower energy consumption for *array*, thus, it is placed into NVM. With the same procedure, *i* is assigned to NVM, and *temp* is assigned to VM because it has the same number of reads/writes. *step* and *size* are

assigned to VM because there is no available space in NVM with size 9, even though the number of reads is larger than writes.

Second, in the multibank decision, the live range of each data block should be analysed. As shown in the code at the top left of Fig. 11, it can be easily confirmed that the definition and use of all data blocks overlap.

Therefore, all data blocks should be placed to the same memory bank in their assigned memory. With a data block holding the largest energy saving, *i* is placed to the first bank of NVM. Next, *array* tries to be placed to the same memory bank of NVM. Since NVM consists of two memory banks of size 5 and 4, thus, three elements of array are placed to the same bank of *i*, and the rest of array are placed to other available memory bank. *Temp*, *step* and *size* are all placed to the same memory bank of VM. In this example, if there are two data blocks that do not overlap their live ranges, then one memory space can be shared with them. To confirm this, we construct an interference graph. Fig. 11 shows the interference graph at the bottom left. Each data block becomes a node of the graph, and the edges represent overlapping of the live range between data blocks. Since all variables overlap in the example, we can see that all variables require independent memory space. Colouring is performed to confirm this. It can be seen that every node needs a different colour. The algorithm ends by allocating each colorized node to its own memory space. In this example, the result shows the same as the static technique, because there is no non-overlapped node in the interference graph. If there is no edge between *array* and *i*, they can share the same space on NVM. In this case, *step* and *size* can be assigned to NVM, it can lead to additional energy saving.

Table 1 Benchmarks from DSPstone

Benchmarks	Complex multiply	Complex update	convolution	Dot product	fir	lms	Matrix1	Matrix2
target variables	6 variables	4 variables	4 variables	4 variables	4 variables	8 variables	6 variables	6 variables
profiles	8 reads 2 writes	10 reads 11 writes	98 reads 98 writes	8 reads 8 writes	105 reads 105 writes	164 reads 146 writes	4410 reads 4910 writes	5010 reads 5510 writes

4 EXPERIMENTAL RESULT

This section illustrates evaluation results of the proposed two techniques in the hybrid multibank memory. In this evaluation, eight multimedia benchmarks are used from DSPstone [27]. Tab. 1 presents the benchmark codes and their profiled information. Cacti 7.0 [28] and NVSim [29] are used for the energy parameter of the hybrid multibank memory. Cacti and NVSim use a variety of memory configurations as an input. It uses the number of banks, the number of ports, the block size, fabrication technology depth, and the memory capacity. They provide its area, delay, and energy consumption for the given memory configuration.

Cacti provides the information for VM only in the hybrid multibank memory, thus, NVSim is used to generate the same information for NVM. To obtain product level's evaluation, STT-MRAM (Spin-Transfer Torque Magnetic Random Access Memory) and SRAM (Static Random Access Memory) are chosen for NVM and VM, since they are the representative architecture design in commercialized products. Tab. 2 shows characteristics of STT-MRAM and SRAM that are used for these evaluations. With a capacity of 32 kB, STT-MRAM operates read/write operations at 8.506 pJ/35.285 pJ, and SRAM operates read/write operations at 31.351 pJ/30.304 pJ. STT-MRAM provides the read operation less than 30% energy consumption of SRAM, but it provides 20% higher write operation than SRAM.

As summarized in Tab. 2, the experiments were conducted on a 32 kB STTRAM/SRAM hybrid multibank memory manufactured by a 22 nm fabrication process. The ratio of the STTRAM/SRAM hybrid bank is 3:1, which is the same as that of commercial products. Fig. 12 and Fig. 13 show the energy consumption results from applied static technique for 8 benchmarks in DSPStone. The eight benchmarks were classified into three groups, large, middle, and small according to the frequency of memory accesses. For each benchmark group, the memory size was changed to 128 kB, 64 kB, 32 kB, and 16 kB.

The baseline for evaluation of the proposed techniques is a multibank memory composed of DRAM. This is because commercial memory architectures consist of a small hybrid memory and a large amount of DRAM.

In the case of small groups benchmarks with 16 kB memory size, the experimental results show that complex_multiply improved by 69.9%, complex_update improved by 75.1%, and dot_product improved by 72% compared to the baseline. It can be seen that the static technique works well even in small memory size. In addition, when the memory size is increased to 32 kB, the energy consumption can be improved up to 85.7%, 75.1%, and 74.6% compared to the baseline respectively. Complex_update does not be improved due to the increment of memory size, because all data are stored to the 16 kB memory. Complex_multiply and dot_product can take additional improvement up to 52.6% and 9.2% respectively when the memory size is increased from 16 kB to 32 kB. As shown in the figure, increment the memory size from 64 kB to 128 kB did not get any additional energy saving. This is because all the data used in the benchmarks can be stored in 32 kB of memory.

Table 2 Characteristics for VM and NVM

	Capacity	Read/Write energy consumption	Fabrication technology depth	Leakage power
STT-MRAM	32 kB	8.506/35.285 pJ	22 nm	1.284 mW
SRAM	32 kB	31.351/30.304 pJ	22 nm	41.712 mW

With the middle group, the results of energy consumption show 32.6% improvement in convolution, 20.2% improvement in fir, and 39.8% improvement in lms compared to the baseline in 16 KB memory size. It can be seen that increasing the memory size from 16 KB to 32 KB greatly increases the improvement of energy consumption

results up to 72.3%, 58.1%, and 73.2% in convolution, fir, and lms respectively. Since the applications in the middle group use more data than the small group, it was confirmed that the amount of gain due to the increase in memory capacity was relatively large compared to the small group. Increasing the memory size from 32 KB to 64 KB increases the energy consumption improvement by 21.3%, 39.4%, and 21.5% in convolution, fir, and lms respectively. There is no gain when the memory capacity increases from 64 kB to 128 kB. As confirmed in the small group, all data of the benchmarks in middle group can be stored in 64 kB, thus, there is no additional energy saving with 128 kB memory space. It is noticed that large memory space does not lead to gain any saving. That is why system architecture should be optimized for a certain application. This is always true for small, middle and large size benchmarks.

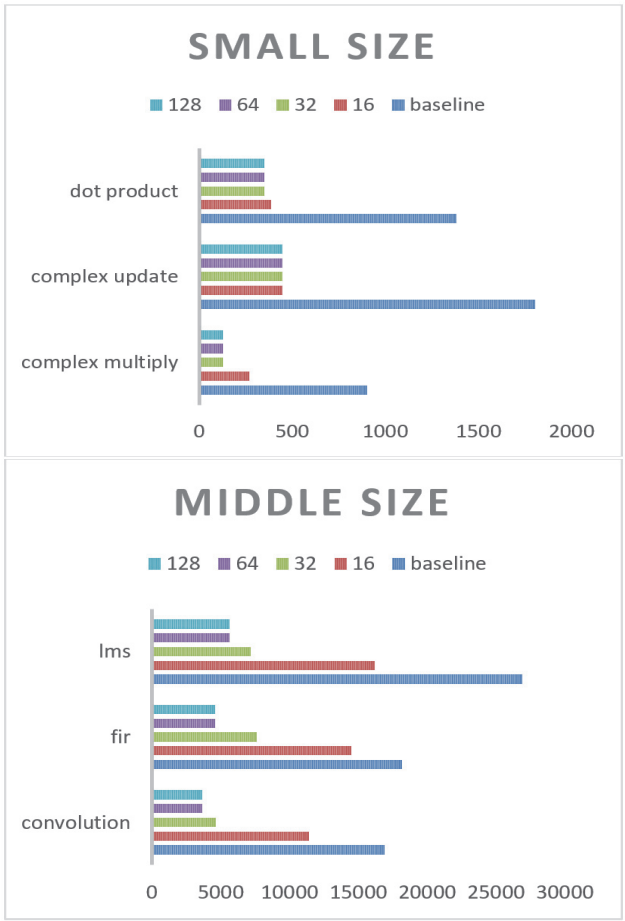

Figure 12 Energy consumption results with the static technique from small and middle benchmarks

In Fig. 13, the large group includes applications that deal with a large amount of matrix operations, and they handle ten times more data than the middle group. In the case of 16 kB memory, energy consumptions are improved by 53.1% in matrix1 and 39.7% in matrix2 compared to the baseline. With 32 kB memory, the energy consumption can be improved by 68.7% and 55% respectively. In the case of matrix2, when the memory capacity was increased to 64 KB, an additional improvement of 38.2% was obtained compared to 32 kB memory, and when the memory capacity was increased to 128 kB, an additional energy saving can be obtained by 11.2%. As a result, it was confirmed that the proposed static technique can obtain the best results in the benchmarks using a large amount of data and an optimal size of memory capacity. Specifically, 64 kB memory space is good enough to save energy for the benchmarks.

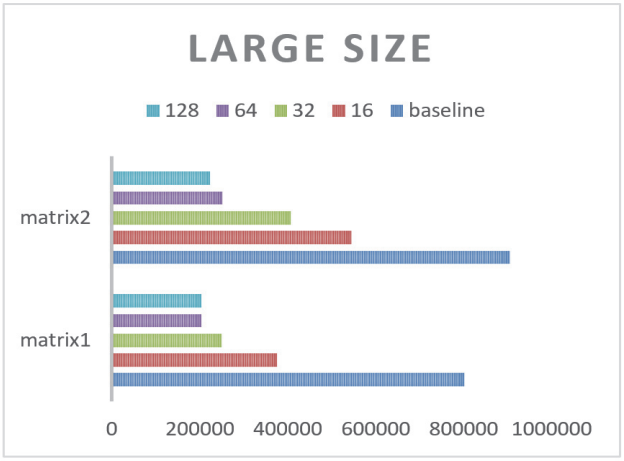

Figure 13 Energy consumption results with the static technique from large benchmarks

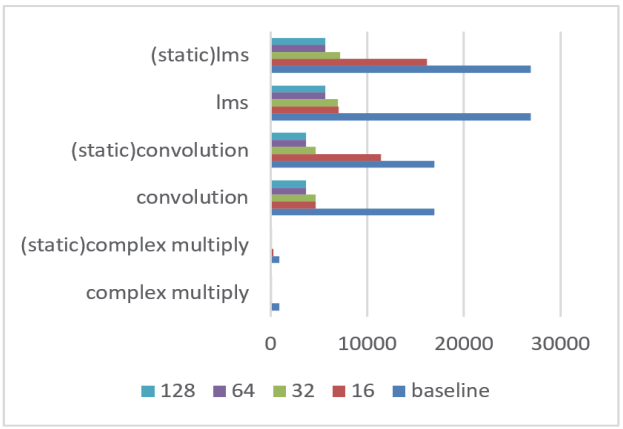

Figure 14 Energy consumption results with the dynamic technique

Fig. 14 shows the energy consumption result for the applied proposed dynamic technique compared to the static technique. By use of the graph colouring technique, the dynamic technique can take additional energy saving compared to the static technique from the benchmarks lms, convolution, and complex_multiply. With 16 kB memory space, complex_multiply improved the energy consumption by 69.9% with the static technique and 85.7% with the dynamic technique. As shown in the figure, the dynamic technique can take about 16% more energy saving than the static technique. In the same configuration, convolution was able to improve about 40% more energy saving compared to the static technique, and lms was able to improve about 34% more energy saving than the static technique. The dynamic technique can improve energy consumption averagely 30% more than the static technique.

This result presents the difference of algorithm 2 (the static technique) and algorithm 3 (the dynamic technique). The reason is that the dynamic technique can place multiple pieces of data in the same place over different time. Thus, it is possible to place more reads to NVM and more writes to DRAM. As a result, this leads to the more energy saving results.

In DSPStone benchmark, there are few variables that have full edges in the interfere graph for the variables used in each application. In an interference graph, if two nodes do not have any edges (their live ranges do not overlap), they can be placed with the same colour (memory place). With an interference graph having low edge degree, the dynamic technique can improve energy consumption more than 30% compared to the static technique. With an interference graph having high edge degree, the static technique is sufficient, since the dynamic technique does not take any chance to place variables to the same memory place at different time.

These results show that the proposed techniques can be used for battery critical applications. In particular, QOS prediction for mobile edge service [30, 31], human pose estimation [32, 33], smartphone application [34], machine learning [35], realtime performance evaluation [36], OS virtualization [37], public safety/security technique [38, 39]. The proposed techniques are essential components because low power is critical in those application areas.

5 CONCLUSION

Many studies have proposed techniques to achieve high performance and energy saving results at low cost. As prior mentioned, there is always a trade-off in these problems that cannot meet all requirements. Low cost conflicts with high performance and high performance conflicts with low power, ending with either choice. In this paper, we assume an environment that is operating at maximum performance with low-cost hardware. This work proposes two compiler optimization techniques that allocate data to efficiently use the hybrid multibank memory composed of non volatile memory and volatile memory.

The proposed techniques consist of static and dynamic techniques. In the hybrid multibank memory, the proposed techniques can achieve energy savings of 50.3% (with static technique) and 59.5% (with dynamic technique) compared to monolithic DRAM only memory bank. In wearable IoT devices, there are tons of data processing, thus, the proposed techniques can efficiently support to improve energy consumption and leakage current. As a future work, it can be evaluated in performance perspective.

Acknowledgements

This research was supported by the 2018 Yeungnam University Research Grant (218A061016, 218A380138) and the National Research Foundation of Korea(NRF) grant funded by the Korea government (MSIT) (No.2018R1D1A1B07050647).

6 REFERENCES

[1] Wolf, M. E. & Lam, M. (1991). A data locality optimizing algorithm. *In Proceedings of the SIGPLAN Conference on Program Language Design and Implementation*, *26*(6), 30-44. https://doi.org/10.1145/113445.113449

[2] Li, W. & Pingali, K. (1992). Access normalization: Loop restructuring for NUMA compilers. *In Proceedings of the Fifth International Conference on Architectural Support for ProgrammingLanguages and Operating Systems*, *11*(4), 353-375. https://doi.org/10.1145/143365.143541

[3] Kennedy, K. & McKinley, K. S. (1993). Maximizing loop parallelism and improving data locality via loop fusion and distribution. *InLanguages and Compilers for Parallel Computing*, 301-320. https://doi.org/10.1007/3-540-57659-2_18

[4] Ferrante, J., Sarkar, V., & Thrash, W. (1991). On estimating and enhancing cache effectiveness. *In Languages and Compilers for Parallel Computing, Fourth International Workshop,* 328-343. https://doi.org/10.1007/BFb0038674

[5] Gannon, D., Jalby, W., & Gallivan, K. (1988). Strategies for cache and local memory management by global program transformation. *Journal of Parallel and Distributed Computing*, *5*(5), 587-616. https://doi.org/10.1016/0743-7315(88)90014-7

[6] Yin, S., Xie, Z., Meng, C., Liu, L. & Wei, S. (2016). Multibank memory optimization for parallel data access in multiple data arrays.*IEEE/ACM International Conference on Computer-Aided Design*, 1-8. https://doi.org/10.1145/2966986.2967056

[7] See https://speech.di.uoa.gr/dsp/manuals/E1.pdf *TMS320 DSP product overview*, Texas Instruments

[8] See https://www.synopsys.com/designware-ip/processor-solutions.html

[9] MeiSerge, B., Diederik, V., Hugo, V., & Lauwereins, M. (2003). ADRES: An Architecture with Tightly Coupled VLIW Processor and Coarse-Grained Reconfigurable Matrix. *Field Programmable Logic and Application*, 61-70. https://doi.org/10.1007/978-3-540-45234-8_7

[10] Banakar, R., Steinke, S., Lee, B., Balakrishnan, M., & Marwedel, P. (2002). Scratchpad memory: a design alternative for cache on-chip memory in embedded systems. *Proceedings of the Tenth International Symposium on Hardware/Software Codesign*, 73-78. https://doi.org/10.1145/774789.774805

[11] Udayakumaran, S., Dominguez, A., & Barua, R. (2006). Dynamic allocation for scratchpad memory using compile-time decisions. *Embedded Computing System*, *5*(2), 472-511. https://doi.org/10.1145/1151074.1151085

[12] Dominguez, A., Udayakumaran, S., & Barua, R. (2005). Heap data allocation to scratch-pad memory in embedded systems. *ACM Transactions on Design Automation of Electronic Systems*, *5*(2), 115-192. https://doi.org/10.1145/1151074.1151085

[13] Xue, C. J., Zhang, Y., Chen, Y., Sun, G., J. Yang, J., & Li, H. (2011). Emerging non-volatile memories: opportunities and challenges. *In Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, 325-334. https://doi.org/10.1145/2039370.2039420

[14] Hassan, A., Vandierendonck, H., & Nikolopoulos, D. S. (2015). Software-managed energy-efficient hybrid DRAM/NVM main memory. *In Proceedings of the 12th ACM International Conference on Computing Frontiers*, 23-31. https://doi.org/10.1145/2742854.2742886

[15] Park, C., Lim, J., Kwon, K., Lee, J., & Min, S. L. (2004). Compiler-assisted Demand Paging for Embedded Systems with Flash Memory. *In Proceedings of the 4th ACM international conference on embedded software*, 114-124. https://doi.org/10.1145/1017753.1017775

[16] Hu, J., Xue, C., Tseng, W., Zhuge, Q., & Sha, E. H. (2010). Minimizing Write Activities to Non-volatile Memory via Scheduling and Recomputation. *Procedings. 8th IEEE Symposium on Application Specific Processors*, 7-12.

[17] Hu, J., Xue, C., Tseng, W., He, Y., Qiu, M., & Sha, E. M. (2010). Reducing Write Activities on Non-volatile Memories in Embedded CMPs via Data Migration and Recomputation. *Proceedings 47th IEEE/ACM Design Automation Conference*, 350-355. https://doi.org/10.1145/1837274.1837363

[18] Mi, W., Feng, X., Xue, J., & Jia, Y. (2010). Software-hardware cooperative DRAM bank partitioning for chip multiprocessors. *In Proceedings of IFIP international conference on Network and parallel computing*, 329-343. https://doi.org/10.1007/978-3-642-15672-4_28

[19] Chen, G., Wu, B., Li, D., & Shen, X. (2014). PORPLE: An Extensible Optimizer for Portable Data Placement on GPU. *47th Annual IEEE/ACM International Symposium on Microarchitecture*, 88-100. https://doi.org/10.1109/MICRO.2014.20

[20] Ved, S. & Awasthi, M. (2018). Exploring non-volatile main memory architectures for handheld devices. *Design, Automation & Test in Europe Conference & Exhibition*, 1528-1531. https://doi.org/10.23919/DATE.2018.8342258

[21] Park, H., Yoo, S., & Lee, S. (2011). Power management of hybrid DRAM/PRAM-based main memory. *Design Automation Conference*, 59-64. https://doi.org/10.1145/2024724.2024738

[22] Cho, J. & Paek, Y. (2004). Fast Memory Bank Assignment for Fixed-Point Digital Signal Processors. *ACM Transaction Design Automation Electrical Systems*, *9*(1), 52-74. https://doi.org/10.1145/966137.966140

[23] Chaitin, G. J. (1982). Register Allocation & Spilling via Graph Coloring, *SIGPLAN Notices, 17*(6), 98-101. https://doi.org/10.1145/872726.806984

[24] Braun, M. & Hack, S. (2009). Register Spilling and Live-Range Splitting for SSA-Form Programs. *In the proceedings of Compiler Construction*, 174-189. https://doi.org/10.1007/978-3-642-00722-4_13

[25] Briggs, P., Cooper, K. D., & Torczon, L. (1994). Improvements to Graph Coloring Register Allocation. *ACM Transactions on Programming Languages and Systems*, *16*(3), 428-455. https://doi.org/10.1145/177492.177575

[26] Schaumont, P. R. (2010). *Analysis of Control Flow and Data Flow*. A Practical Introduction to Hardware/Software Codesign. Springer. https://doi.org/10.1007/978-1-4419-6000-9_3

[27] Zivojnovic, V., Mart'nez, J., Schlager, C., & Meyr, H. (1994). Dspstone: a dsp-oriented benchmarking methodology. *Proceedings of Signal Processing Applications & Technology*.

[28] Balasubramonian, R., Kahng, A., Muralimanohar, N., Shafiee, A., & Srinivas, V. (2017). CACTI 7: New Tools for Interconnect Exploration in Innovative Off-Chip Memories. *ACM Transactions on Architecture and Code Optimization*. *14*(2), 14-39. https://doi.org/10.1145/3085572

[29] Dong, X., C., Xu, X. Y., & Jouppi, N. P. (2012). NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *31*(7), 994-1007. https://doi.org/10.1109/TCAD.2012.2185930

[30] Yin, Y., Zhang, W., Xu, Y., Zhang, H., Mai, Z., & Yu, L. (2019). QoS Prediction for Mobile Edge Service Recommendation with Auto-encoder. *IEEE Access*, *7*(1), 62312-62324. https://doi.org/10.1109/ACCESS.2019.2914737

[31] Yin, Y., Chen, L., Xu, Y., Wan, J., Zhang, H., & Mai, Z. (2019). QoS Prediction for Service Recommendation with Deep Feature Learning in Edge Computing Environment. *Mobile Networks and Applications*, *25*, 391-401. https://doi.org/10.1007/s11036-019-01241-7

[32] Yu, J., Guo, Y., Tao, D., & Wan, J. (2015). Human pose recovery by supervised spectral embedding. *NEUROCOMPUTING*., *166*, 301-308. https://doi.org/10.1016/j.neucom.2015.04.005

[33] Yu, J. & Hong, C. (2017). Exemplar based 3D human pose estimation with sparse spectral embedding. *NEUROCOMPUTING*, 269, 82-89. https://doi.org/10.1016/j.neucom.2016.09.137

[34] Kang, S., Ando, K., & Yuasa, T. (2015). The Planning and Implementation of Smartphone Application Designed to Efficient Donation for Direct Support to the 2011 Tohoku Earthquake-Affected Area. *International Journal of Disaster Recovery and Business Continuity, NADIA,* 1-8. https://doi.org/10.14257/ijdrbc.2015.6.01

[35] Rao, N. T. (2018). A Review on Industrial Applications of Machine Learning. *International Journal of Disaster Recovery and Business Continuity, NADIA*, *8*(1), 1-10.

[36] Kambourakis, G., Geneiatakis, D., Gritzalis, S., Lambrinoudakis, C., Dagiuklas, T., Ehlert, S., & Fiedler, J. (2010). High Availability for SIP: Solutions and Real-Time Measurement Performance Evaluation. *International Journal of Disaster Recovery and Business Continuity, NADIA*, *1*(1), 11-30.

[37] Yu, H., Xiang, X., & Shu, J. (2010). A New Global Consistent Checkpoint Based on OS Virtualization. *International Journal of Disaster Recovery and Business Continuity, NADIA*, 31-40.

[38] Baldini, G., Sallent, O., Subik, S., & Wietfeld, C. (2011). The Evolution of ICT in the Public Safety Domain: Challenges and Opportunities. *International Journal of Disaster Recovery and Business Continuity, NADIA*, 9-22.

[39] Coskun, A. & Bostanci, U. (2018). Vulnerability analysis of smart phone and tablet operating systems. *Technical gazette*, *25*(6), 1860-1866. https://doi.org/10.17559/TV-20170627175835

Contact information:

Jungseok CHO, PhD
Electrical & Electronic Engineering, Sunchon National University,
Suncheon, Jeollanam-do, South Korea
E-mail: icaroosion@naver.com

Jonghee M. YOUN, PhD, Professor
(Corresponding author)
Computer Engineering, Yeungnam University,
Gyeongsan, Gyeongbuk, South Korea
E-mail: youn@yu.ac.kr

Doosan CHO, PhD, Professor
(Corresponding author)
Electrical & Electronic Engineering, Sunchon National University,
Suncheon, Jeollanam-do, South Korea
E-mail: dscho@scnu.ac.kr