# A Novel Function Complexity-Based Code Migration Policy for Reducing Power Consumption

Hayeon Choi, Youngkyoung Koo, and Sangsoo Park

Original scientific paper

***Abstract*—Embedded system designs has changed greatly owing to rapid developments in both hardware and software technology. Typical designs should consider hardware limitations, such as size, weight, or battery capacity. In other words, the designs are heavily dependent on hardware components. Since hardware components can deteriorate and degenerate, hardware-aware software designs are needed to achieve power-efficient embedded systems. Previous studies usually focus on the microprocessor expecting to reduce power consumed on computation. Besides, entire program execution resulting a lot of memory accesses also consume power. Therefore, it should be considered to minimize overall power consumption for more efficient designs. Modern embedded systems often use heterogeneous memory to benefit from different characteristics of each memory device. This study aims to optimize the power efficiency of heterogeneous memory in embedded systems. We have proposed a detailed function complexity concept whose scale implies the range of power consumption in migrated memory. Afterward, function selection algorithm with function complexity selects a unique function which improve power consumption most after the migration. Several experiments and quantitative analyses with various benchmarks have been performed to validate the proposed algorithm. Consequently, migrating selected complex function successfully minimizes power consumption of an embedded system.**

***Index Terms*— embedded system, heterogeneous memory, code migration, function complexity.**

## I. INTRODUCTION

WITH the rapid developments of computer science technology, efficient designs of the systems should consider the hardware-wise properties in mind, such as size, weight, or battery capacity. In other words, hardware is the main limitation in embedded systems. However, physical hardware components can wear out or deteriorate as time goes. To achieve efficient systems, hardware-aware software design is urgently required.

Embedded systems consist of a microprocessor, a single memory or multiple memory devices, and other peripheral hardware as well as software. Most previous studies have focused on optimizing the microprocessor because it performs main computations, which consume power. However, accesses to memory devices also consumes power apart from the microprocessor. When running program codes and accessing program data during the entire execution causes a lot of memory access. Therefore, power consumption of memory should also be minimized for achieving more efficient system designs [7, 8]. Moreover, most embedded systems equipped with heterogeneous memory, that is, multiple types of memory devices. In most cases, non-volatile memory (NVM) and non-volatile memory (VM) are equipped in the systems. It enables to benefit from different characteristics of memory devices.

Our previous studies focused on migrating program code utilizing the properties of both memory devices for operating a low-power embedded system. Basic concept of code migration method refers to migrating program code from one memory device to another regarding the characteristics of each memory device. Experiments have showed that moving certain function units of a program generally consumes less power. Thus, the effect of migration depends on situations, mostly the power consumptions reduced than usual. Consequently, earlier code migration methods could affect considerably on the operation of low-power embedded systems as a positive factor [9-11].

In this paper, we propose the function complexity concept to describe the computational complication of each function and the relationship between connected functions. Furthermore, function selection algorithm based on the function complexity is proposed. With the algorithm, a novel code migration policy is performed and experimentally verified using several benchmarks.

The remainder of this paper is organized as follows: Section II.A provides a comprehensive description of the code migration method and Section II.B quantitatively analyzes motivating examples as well as code migrations. Section III.A presents the procedure of function complexity calculation, and function selection algorithm is proposed in Section III.B. Then section IV.A discusses the experimental environment for implementing a novel code migration policy.

Manuscript received January 31, 2018; revised March 3, 2018. Date of publication March 15, 2018.

H. Choi, Y. Koo, and S. Park are with the Department of Computer Science and Engineering, Ewha Womans University, Seoul, Republic of Korea.

E-mails: {hayeon.choi, kooyoungkyoung}@ewhain.net, sangsoo.park@ewha.ac.kr.

This article is an extension of the following paper: Hayeon Choi, Youngkyoung Koo, and Sangsoo Park, "Segment-aware energy-efficient management of heterogeneous memory system for ultra-low-power IoT devices," proceeding of the 2nd international Multidisciplinary Conference on Computer and Energy Science (SpliTech), pp. 1-6, 2017.

This work was supported the National Research Foundation of Korea funded by the Korean Government (NRF2017S1A5B6066963). Sangsoo Park is the corresponding author.

Digital Object Identifier (DOI): 10.24138/jcomss.v14i1.454

Section IV.B organizes the experiments for various benchmarks and the final analysis to verify the proposed algorithm. Finally, Section V concludes this study.

## II. Code Migration Methods

### A. Motivation

In general, embedded systems load and execute program codes on NVM. On the contrary, they read and write program data on VM owing to the location of data. Therefore, several accesses to NVM and VM are required for executing program codes and using program data, respectively. Code migration method loads and executes code from the memory it is originally in to another as shown in the Fig. 1. When using heterogeneous memory, the method migrates program code from NVM to VM. In particular, one of the most distinguishable property of VM is relatively low-power operation compared to NVM. Because of the structural difference, therefore, the power consumption can be reduced through code migration method. Thus, the power efficiency is dependent on the structural characteristics of the located memory device. In heterogeneous memory systems, it is especially important to consider each characteristic when executing a program in a low-power system.

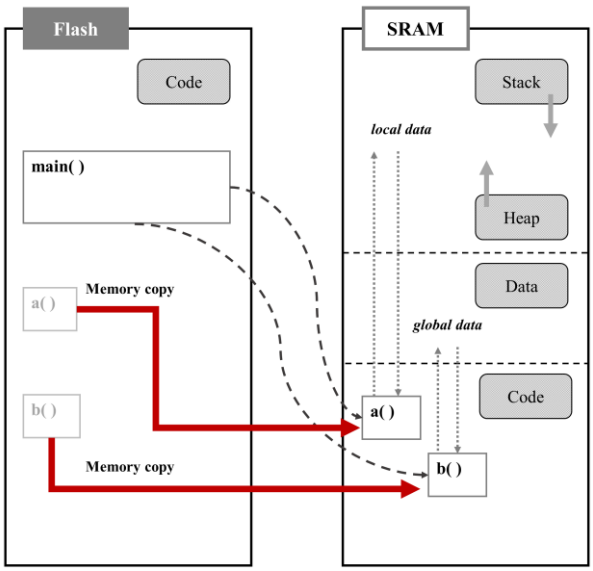



Fig. 1. General flow of a code migration method.

### B. Quantitative Analysis of Motivating Examples

An application being used can affect the power efficiency of a program. Therefore, three different programs with different time complexities were migrated to reveal the influence of the code migration method. Each motivating example has different time complexity: a constant, a square, and a coefficient. In a constant-complexity program, the problem is to determine whether a number is an odd or even. Next, in a square-complexity program, the problem is to perform a bubble sort. Lastly, a coefficient-complexity program solves the 0-1 knapsack problem.

For each example, the power consumption was measured before and after migration. Also, the power efficiency, described as the reduction ratio in the table, was calculated by the measured power consumption values. If code migration had a positive influence on optimizing power consumption, the consumed power after migration should be reduced than that before.

As shown in Table I, the result showed that each program has a different efficiency according to time complexity. When executing a constant-complexity program, more power was consumed after migration. Which implies that a negative effect was given by the code migration method. Such result demonstrated that there existed the overhead of migration which was larger than the effect of migration in this case. On the other hand, the migration method had an influence to reduce power consumption on the other two programs. A coefficient-complexity program especially had the highest reduction ratio. Therefore, the power efficiency of a program with higher time complexity has improved more after migration. It seemed that the effect of migration could be larger as the time-complexity went higher [9, 10].

TABLE I
Power Consumption and Reduction Ratio by Code Migration

| Time Complexity | Power Consumption (W) | | Reduction Ratio (%) |
|---|---|---|---|
| | Before Migration | After Migration | |
| O(*1*) | 0.0068 | 0.0071 | -3 |
| O($n^2$) | 0.0078 | 0.0077 | 1 |
| O(*n!*) | 0.008 | 0.0078 | 2.3 |

## III. Function Selection Algorithm based on Function Complexity

The motivating examples discussed in Section II.B are programs with a single function connected to the main function. Most of those programs can be stated in a simplified diagram Fig. 2 (a). However, most programs are comprised of complex additional functions as shown in Fig. 2 (b).

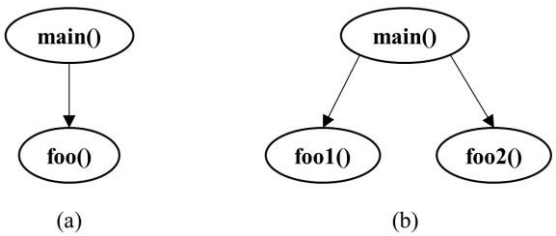



Fig. 2. (a) Diagram of a program with a single function and (b) Diagram of a program with complex functions.

As mentioned in the previous section, migrating code to VM can improve the power efficiency. In embedded systems, however, moving the whole program can be difficult or impossible because of hardware constraints such as limited capacity, especially. Therefore, the function complexity concept is newly introduced which predicts a certain function expecting to improve the power efficiency most after migration.

### A. Function Complexity Calculation

For all functions ($F_p$) in a program ($P_i$), each function complexity is calculated. A function complexity ($FC_p$) implies the combination of computational complexity and calling complexity. Therefore, it reflects not only how complicated the function itself is, but also how the function is connected to others. It is calculated by using the following equation:

$$P_i = F_1 \cup F_2 \cup \cdots \cup F_p \cup \cdots F_n \quad (1)$$
$$FC_p = CalculateFC(F_p)$$

where $P_i$ = an $i^{th}$ program with $n$ functions,
$F_p$ = a $p^{th}$ function of a program,
$FC_p$ = a final function complexity of $F_p$

Fig. 3 shows the procedure for calculating the function complexity in a program and the details of each step.

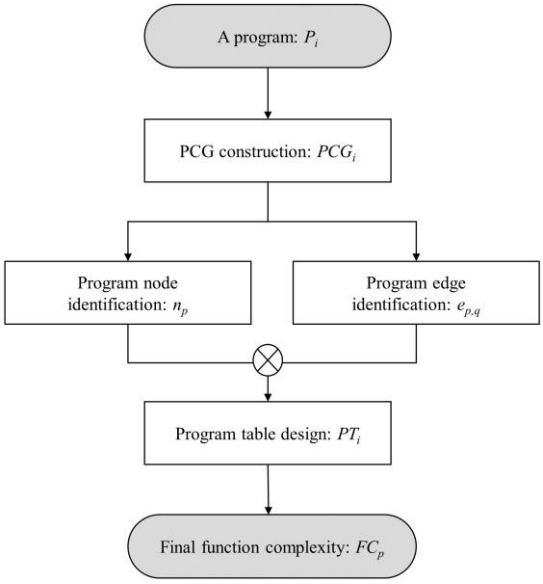



Fig. 3. Flowchart of procedure for calculating function complexity.

#### 1) PCG construction

A general program can be expressed in the form of a program call graph ($PCG_i$). Since a program consists of various functions, these functions construct a set of nodes ($N$). Moreover, these functions invoke each other and form complicated relationships with each other. Those invocation are calls between functions indicating edges ($E$). They are derived from the following formula, for each program:

$$PCG_i\,(N, E) \leftarrow P_i\,(f, c) \quad (2)$$

where $PCG_i$ = a program call graph of a $i^{th}$ program,
$f$ = #($F$), $c$ = #($calls$)

#### 2) Program node identification

Each node of a $PCG_i$ represents a different function that exists in a program. These functions have their own unique complex standards, computational complexity based on the Big-O notation. It is calculated by the time complexity for a loop statement; i.e. the following formula (3) for a for-while loop was applied:

$$n_p \mathrel{+}= R(loop_i) \quad (3)$$
$$R(loop_i) = \lfloor repetition\ of\ loop_i\ /\ incremental\ unit\ of\ loop_i \rfloor$$

where $n_p$ = a computational complexity of $F_p$,
$loop_i$ = an $i^{th}$ loop statement in a function

#### 3) Program edge identification

Understanding the relationship between various functions will significantly important in terms of the function complexity. Every edge indicates how many invocations are occurred between two nodes. Thus, the calling complexity is determined by the number of times a function $j$ calls another function $k$.

$$e_{p,q} = 1, \text{ where } F_p \textit{ calls } F_q \textit{ for constant times} \quad (4)$$
$$n, \text{ where } F_p \textit{ calls } F_q \textit{ for } n \textit{ times}$$
$$n^2, \text{ where } F_p \textit{ calls } F_q \textit{ for } n^2 \textit{ times}$$
$$\cdots$$

where $e_{p,q}$ = a calling complexity between $F_p$ and $F_q$

#### 4) Program table design

The program table ($PT$) reflects program nodes and edges derived above. This table shows functions and function calls that comprise a program. Therefore, the $PT_i$ should contain all nodes and edges of the $PCG_i$. Basically, each element $pt_{p,q}$ of the $PT_i$ reflects the relationship between $F_p$ and $F_q$, that is same the as the edge between $F_p$ and $F_q$ of $PCG_i$. In contrast, as diagonal element $pt_{p,q}$ has the same $j$ and $k$, which is the same as the $F_p$ (=$F_q$), one of the vertexes in $PCG_i$. Thus, the relation is with the function itself. All elements of $PT_i$ are constructed using (5), and the constructed table is shown in Fig. 4.

$$pt_{p,q} = n_p, \text{ where } p = q \quad (5)$$
$$pt_{p,q} = e_{p,q}, \text{ otherwise } p \neq q$$

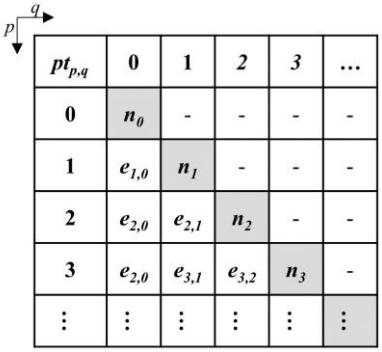



Fig. 4. Example of constructed program table.

#### 5) Function complexity calculation

Finally, the complexity of each function can be determined by the program table $PT$. As mentioned in the previous section, the function complexity is the concept which depicts computational complexity of each function and calling complexity with others. A function complexity of a node derived from a calculation of all paths backtracked to

the main function. If a node has only a single path on $PCG_i$, a function complexity is a product of whole edges on the path and the node itself. Any other nodes with multiple paths, on the other hand, should consider each path respectively. From each path, a few included edges are multiplied, and then the calculated products of paths are summed up. A final function complexity is a multiplication of summed paths and the node itself. These rules calculate a final function complexity whose detailed algorithm is given in Fig. 5.

**Algorithm 1** Function Complexity Calculation

**Input:** Program $P_i$, Function $F_p$, Program Table $PT_i$ whose element $pt_{p,q}$

```
for all program P_i do
    for all functions F_p do
        CalculateFC(F_p)
    end for
end for

function CalculateFC(F_p)
    for q = p − 1 ... 0 do
        if pt_{p,q} is not NULL then
            if q == 0 then
                return pt_{p,q}
            else
                X = CalculateFC(F_q) * pt_{p,q}
            end if
            Y += X
        end if
    end for
    FC_p = pt_{p,q} * Y
    return FC_p
end function
```

**Output:** Function Complexity $FC_p$ of Function $F_p$ in Program $P_i$

Fig. 5. Algorithm of function complexity calculation.

### B. Function selection algorithm

Based on the calculated final function complexity, the function selection algorithm predicts the function that is expected to be the most power-efficient after code migration. The selection of a unique function is conducted as followed:

At first, a function whose complexity $FC_p$ is the largest is selected. If the orders of two or more function complexities are the same, a function with larger $n_p$ is selected. It resulted from the fact that the time complexity of a function makes more big impact on the power consumption proved in our previous studies.

## IV. Case Study on Benchmark

### A. Experimental environment

Mibench is a benchmark tool used to verify the performance of embedded processors [12]. We evaluated the performance of the proposed algorithm by porting the benchmark program to the experimental environment. Table II shows the benchmark information used in the experiment.

TABLE II
DESCRIPTION OF THE BENCHMARK PROGRAMS

| Programs ($P_i$) | Benchmark program | Number of $F_p$ ($f$) |
|---|---|---|
| $P_1$ | basicmath() | 2 |
| $P_2$ | qsort() | 3 |
| $P_3$ | dijkstra() | 4 |
| $P_4$ | blowfish() | 4 |
| $P_5$ | SHA() | 5 |

The experimental board used was the MSP432P401R Launchpad of Texas Instruments. This board is a primary experimental tool for measuring power in embedded systems with heterogeneous memory. The device is comprised of a microprocessor, i.e. cost-effective and low-power Cortex- M4F, 256 kB Flash memory, and 64 kB SRAM. For optimal usage of power, SRAM can be partitioned every 8 kB; these partitions can be powered down individually. The segments of program data are mapped to each of these partitions [13, 14]. The most distinguishable attribute of the device is the embedded EnergyTrace+ hardware. It allows for real-time debugging and internal power measurement using a tool named Code Composer Studio (CCS). It is an Integrated Development Environment for Texas Instruments' microcontrollers and embedded processors. It is an advanced system development and debugging tool based on the Eclipse Framework and it had built-in EnergyTrace+ hardware for measuring the power consumption [15].

### B. Analysis of the benchmarks

To validate of the function selection algorithm, we performed experiments to ensure that the selected function most improve the power efficiency after migration. The program call graph, $PCG_i$, and program table, $PT_i$ for the benchmark programs consisting of function complexity were constructed. Then, we measured the power consumption when each function unit of the code was migrated using the experimental board. Information about each function in the benchmark program was obtained using the algorithm shown in Figs. 6-10. The control group of the experiment was the power consumption when the code was not migrated; this was referred as *basic*.

Experiments were carried out to determine the difference in power consumption before and after code migration of function units. The results showed that the power efficiency is correlated to the function complexity. Furthermore, when $n_p$ value of the function is the same, the higher $FC_p$, the larger efficiency is obtained.

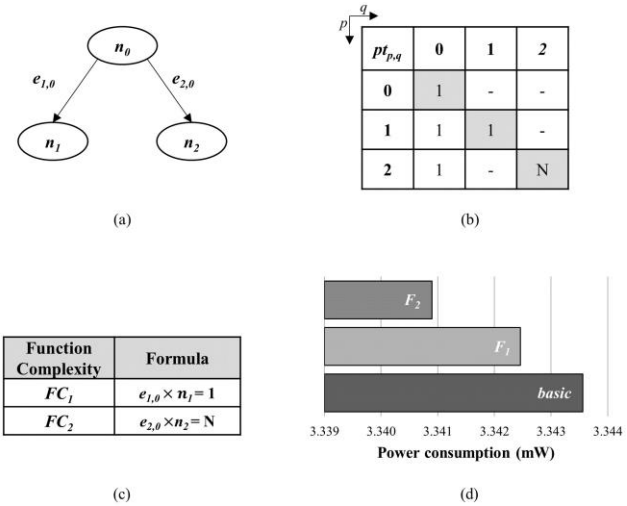
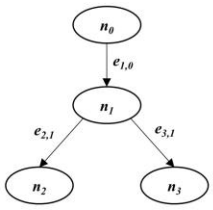



Fig. 6. Program1: basicmath.



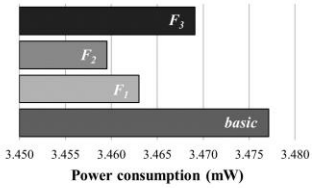

(a)

| $pt_{p,q}$ | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | 1 | - | - | - |
| 1 | 1 | N | - | - |
| 2 | - | N | N | - |
| 3 | - | N | - | 1 |

(b)

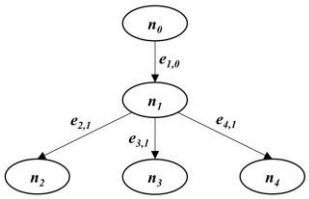
| Function Complexity | Formula |
|---|---|
| $FC_1$ | $e_{1,0}\times n_1 = \mathbf{N}$ |
| $FC_2$ | $e_{2,1}\times n_2 = \mathbf{N^2}$ |
| $FC_3$ | $e_{3,1}\times n_3 = \mathbf{N}$ |

(c)




(d)

Fig. 7. Program2: qsort.



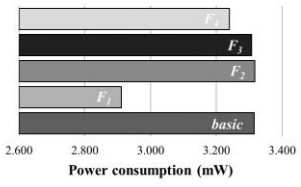

(a)

| $pt_{p,q}$ | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 1 | - | - | - | - |
| 1 | N | $N^2$ | - | - | - |
| 2 | - | 1 | 1 | - | - |
| 3 | - | N | - | 1 | - |
| 4 | - | $N^2$ | - | - | 1 |

(b)

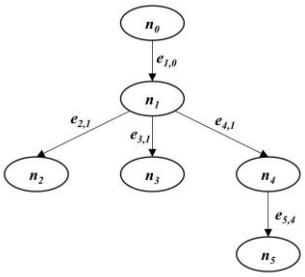
| Function Complexity | Formula |
|---|---|
| $FC_1$ | $e_{1,0}\times n_1 = \mathbf{N^3}$ |
| $FC_2$ | $e_{2,1}\times n_2 = \mathbf{N}$ |
| $FC_3$ | $e_{3,1}\times n_3 = \mathbf{N^2}$ |
| $FC_4$ | $e_{4,1}\times n_4 = \mathbf{N^3}$ |

(c)




(d)

Fig. 8. Program3: dijkstra.



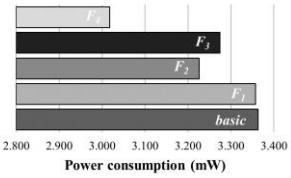

(a)

| $pt_{p,q}$ | 0 | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| 0 | 1 | - | - | - | - |
| 1 | 1 | N | - | - | - |
| 2 | N | N | N | - | - |
| 3 | N | - | - | N | - |
| 4 | - | - | N | N | 1 |

(b)

| Function Complexity | Formula |
|---|---|
| $FC_1$ | $e_{1,0}\times n_1 = \mathbf{N}$ |
| $FC_2$ | $((e_{1,0}\times e_{2,1})+e_{2,0})\times n_2 = \mathbf{2N^2}$ |
| $FC_3$ | $e_{3,0}\times n_3 = \mathbf{N^2}$ |
| $FC_4$ | $((e_{1,0}\times e_{2,1}\times e_{4,2})+(e_{2,0}\times e_{4,2})+(e_{3,0}\times e_{4,3}))\times n_4 = \mathbf{3N^2}$ |

(c)




(d)

Fig. 9. Program4: blowfish.



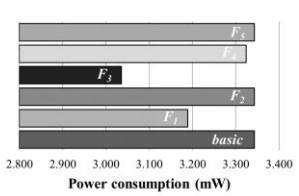

(a)

| $pt_{p,q}$ | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 0 | 1 | - | - | - | - | - |
| 1 | 1 | N | - | - | - | - |
| 2 | - | 1 | 1 | - | - | - |
| 3 | - | N | - | N | - | - |
| 4 | - | 1 | - | - | 1 | - |
| 5 | - | - | - | - | 1 | 1 |

(b)

| Function Complexity | Formula |
|---|---|
| $FC_1$ | $e_{1,0}\times n_1 = \mathbf{N}$ |
| $FC_2$ | $(e_{1,0}\times e_{2,1})\times n_2 = \mathbf{1}$ |
| $FC_3$ | $(e_{1,0}\times e_{3,1})\times n_3 = \mathbf{N^2}$ |
| $FC_4$ | $(e_{1,0}\times e_{4,1})\times n_4 = \mathbf{1}$ |
| $FC_5$ | $(e_{1,0}\times e_{4,1}\times e_{5,4})\times n_5 = \mathbf{1}$ |

(c)




(d)

Fig. 10. Program5: SHA.

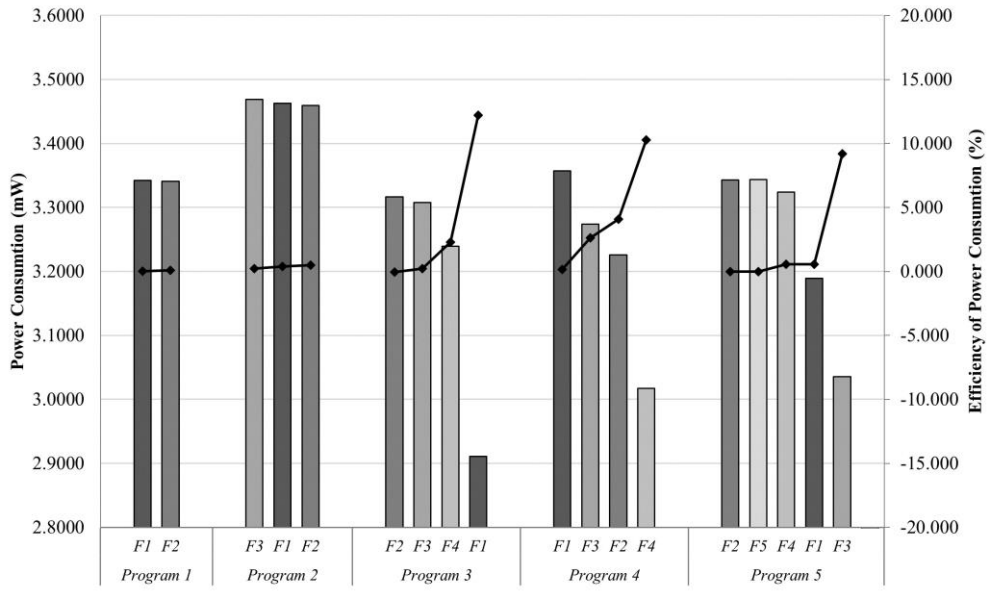



Fig. 11. Experimental result.

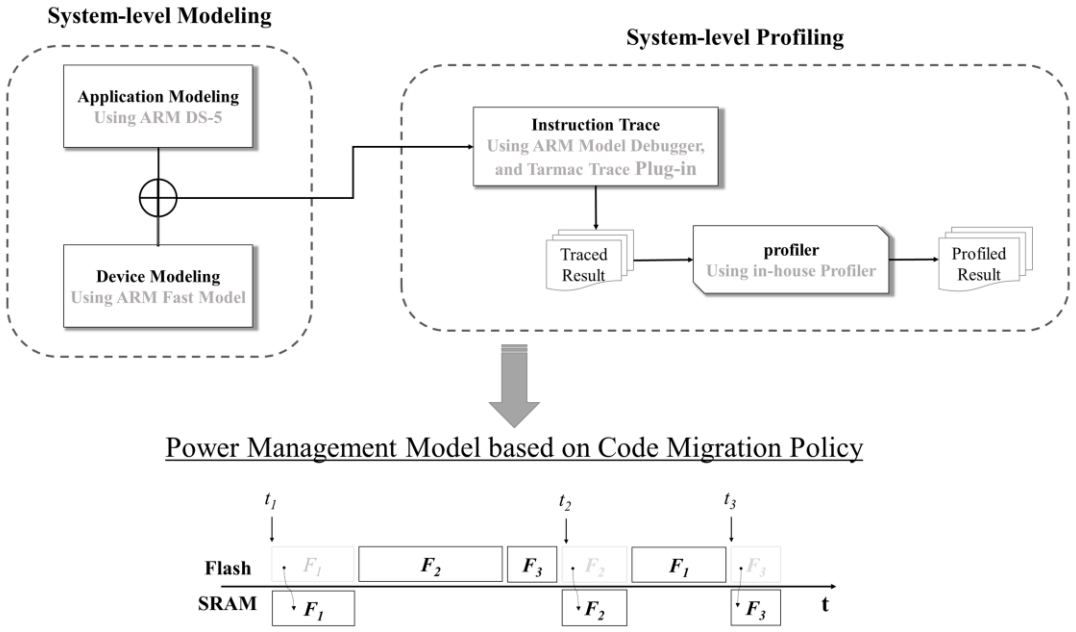



Fig. 12. Framework design of power management model based on code migration policy.

For instance, in case of a program named dijkstra, final complexities of functions are calculated in Fig. 8 (c). As the results imply that $FC_1$ and $FC_3$ are the same, $N^3$. By the function selection algorithm, $n_1$ is selected because the value of $n_1$ is larger than $n_3$. Therefore, the power consumption migrating $n_1$ to another memory device reduced most among the other functions.

## V. Conclusion

Embedded systems are subjected to various constraints, and therefore, the trade-offs involved should be studied in detail. In a real operating environment, low power consumption can help achieve long-term use with high performance. This study focuses on managing and fully utilizing existing heterogeneous memory while considering the additional cost for low-power systems. Our experiment shows that migrating certain function unit of a program in power-efficient VM is effective. Furthermore, executing a few program functions in SRAM is comparatively more power-efficient than executing them in NVM.

This study proposed a function selection algorithm for more efficient code migration of a complex program. This algorithm calculates the highest power efficiency of a function after it has been migrated. To validate the proposed algorithm, we ported a benchmark that evaluates the performance of a general process to the experimental environment. The results showed that the function with highest function complexity is the most power-efficient among all functions in a program. The proposed function selection algorithm could serve as an important measure for determining function units in code migration.

## VI. Future Work

The program code is a set of arithmetic, data transfer, and other instructions. Several previous studies have proved the effectiveness of instructions in code migration, respectively. For example, in the paper [9, 10], the power consumption can be reduced according to time complexity of a function by focusing on arithmetic instructions. Furthermore, the power consumption for data transfer instructions decreases depending on the segment type [11]. This study proposes the function complexity concept based on a deeper analysis of arithmetic instructions, and accordingly proposes a novel code migration policy.

Based on these existing studies, we present the model shown in Fig. 12. Specifically, we propose an embedded device and a profiler framework through a system-level simulation based on arithmetic and data transfer instructions for selecting program code to be migrated. This model applies the trace method to an application and analyzes the case of all commands in detail.

## References

[1] Z. Sheng, S. Yang, Y. Yu, A. Vasilakos, J. Mccann, and K. Leung, "A survey on the ietf protocol suite for the internet of things: Standards, challenges, and opportunities," Journal of the IEEE Wireless Communications, vol. 20, no. 6, pp. 91-98, 2013.

[2] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of things for smart cities," Journal of the IEEE Internet of Things, vol. 1, no. 1, pp. 22-32, 2014.

[3] L. Wang, A. Vega, A. Buyuktosunoglu, P. Bose, and K. Skadron, "Power-efficient embedded processing with resilience and real-time constraints," Proceeding of the IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED), pp. 231-235, 2015.

[4] S. Mai, C. Zhang, Y. Zhao, J. Chao, and Z. Wang, "An application specific memory partitioning method for low power," Proceeding of the 7th International Conference on ASIC, pp. 221-224, 2007.

[5] L. Liu et al., "A software memory partition approach for eliminating bank-level interference in multicore systems," Proceeding of the 21st International Conference on Parallel Architectures and Compilation Techniques(PACT), pp. 367-375, 2012.

[6] C. Chi, L. Wu, X. Zhang, and L. Pan, "Low power embedded EEPROM design for MCU in battery-less tire pressure monitoring system," proceeding of the 11th International Conference on Solid-State and Integrated Circuit Technology (ICSICT), pp. 1-3, 2012.
[7] G. Indumathi, and V. P. M. B. Aarthi alias Ananthakirupa, "Energy optimization techniques on SRAM: A survey," proceeding of the International Conference on Communication and Network Technologies (ICCNT), pp. 216-221, 2014.
[8] Z. Wang, D. A. Jimenez, C. Xu, G. Sun, and Y. Xie, "Adaptive placement and migration policy for an STT-RAM-based hybrid cache," proceeding of the 20th International Symposium on High Performance Computer Architecture (HPCA), pp. 13-24, 2014.
[9] H. Choi, Y. Koo, S. Park, "Power consumption comparisons of flash and RAM on time complexities for low power embedded systems", proceeding of the Conference on Korea Multimedia Society, vol. 19, 2016.
[10] H. Choi, Y. Koo, and S. Park, "Quantitative analysis of power consumption for low power embedded system by types of memory in program execution", Journal of Korea Multimedia Society, vol. 19, no. 2016, pp. 1179-1187.
[11] H. Choi, Y. Koo, and S. Park, "Segment-aware energy-efficient management of heterogeneous memory system for ultra-low-power IoT devices," proceeding of the 2nd international Multidisciplinary Conference on Computer and Energy Science (SpliTech), pp. 1-6, 2017.
[12] M. R. Guthaus et al., "MiBench: A free Commercially Representative Embedded Benchmark Suite," proceeding of IEEE 4th Annual Workshop on Workload Characterization, pp. 3-14, 2001.
[13] MSP432P401R LaunchPad™ Development Kit (MSP EXP432P401R), http://www.ti.com/lit/ ug/slau597a/slau597a.pdf, (accessed Apr., 9 2016).
[14] Overview for MSP432P4x, http://www.ti.com/lsds/ti/microcontrollers_16-bit_32-bit/msp/low_power_performance/msp432p4x/overview.page, (accessed Apr., 1, 2016).
[15] Code Composer Studio(CCS) Integrated Development Environment (IDE), http://www.ti.com/tool/ccstudio#TechnicalDocuments, (accessed Apr., 17 2016).

**Hayeon Choi** received her BS degree from Ewha Womans University, Seoul, Republic of Korea, in 2013. Currently, she is in the combined Master's and Doctorate program in the Embedded Software Laboratory in the Department of Computer Science and Engineering at Ewha Womans University.

**Youngkyoung Koo** received her BS degree from Ewha Womans University, Seoul, Republic of Korea, in 2017. Currently, she is an MS candidate in the Embedded Software Laboratory in the Department of Computer Science at Ewha Womans University.

**Sangsoo Park** received his BS degree from Korea Advanced Institute of Science and Technology, Daejeon, Republic of Korea, in 1998, and his MS and PhD degrees from Seoul National University, Seoul, Republic of Korea, in 2000 and 2006, respectively. Currently, he is an Associate Professor in the Department of Computer Science and Engineering at Ewha Womans University, Seoul, Republic of Korea. His research interests include real-time embedded systems and system software.