International Journal of Internet, Broadcasting and Communication Vol.13 No.3 104-108 (2021)
http://dx.doi.org/10.7236/IJIBC.2021.13.3.104

IJIBC 21-3-13

A Technique for Improving the Performance of Cache Memories

Doosan Cho

Professor, Department of Electrical & Electronic Engineering, Sunchon National Univ., Korea
E-mail: dscho@scnu.ac.kr

Abstract

In order to improve performance in IoT, edge computing system, a memory is usually configured in a hierarchical structure. Based on the distance from CPU, the access speed slows down in the order of registers, cache memory, main memory, and storage. Similar to the change in performance, energy consumption also increases as the distance from the CPU increases. Therefore, it is important to develop a technique that places frequently used data to the upper memory as much as possible to improve performance and energy consumption. However, the technique should solve the problem of cache performance degradation caused by lack of spatial locality that occurs when the data access stride is large. This study proposes a technique to selectively place data with large data access stride to a software-controlled cache. By using the proposed technique, data spatial locality can be improved by reducing the data access interval, and consequently, the cache performance can be improved.

Keywords: *Cache memory, performance, data access pattern, memory architecture, array data, data layout*

1. Introduction

With the development of mobile devices, the personal computing market has gradually disappeared, and devices such as smart phones/watches and artificial intelligence speakers have become the mainstream of the market. As of 2020, 300 million personal computers are sold per year, and 1.3 billion smartphones are sold annually as a single item. These mobile devices are designed and developed according to the performance and low power requirements and cost required by the market. It is particularly difficult to meet low-power requirements at the design stage, as a lot of power is required to provide high performance. Therefore, low-power optimization is performed on various components of the system, and the display component and the memory component are representative optimization targets. In a display, a method of adjusting brightness according to lighting is the most used representative method. In a memory system, a method of extracting power consumption is commonly used as a method of minimizing the number of accesses of an off-chip memory such as a memory card. To implement this, it is necessary to effectively use on-chip memory such as cache memory. For this purpose, various studies have been published.

[1] proposed a low-power support technology for memory in IoT or edge computing, and [2] introduced

Manuscript Received: June. 25, 2021 / Revised: June. 29, 2021 / Accepted: July. 3, 2021
Corresponding Author: dscho@scnu.ac.kr
Tel: +82-061-750-3577, Fax: +82-061-750-3570
Professor, Department of Electrical & Electronic Engineering, Sunchon National University, Korea

considerations in memory for medical system use. In [3], an effective use of low-cost, high-performance system memory was proposed, and in [4] and [5], memory system support technology using machine learning and artificial intelligence was introduced. In [6], it was introduced that energy and performance can be improved by the recalculation technique for removing spill variables generated by the compiler. In [7][8][9], memory design support and low-cost memory system optimization techniques were introduced using a compiler framework such as LLVM.

In this study, we describe the data placement optimization technique required to move the array data stored in the off-chip to the software-controlled cache memory. Since this can be applied simultaneously with the above-mentioned various existing techniques for supporting a low-power memory system in a mobile device, a further improved result can be obtained as a result.

2. Data Arrangement

In the hardware-controlled cache, a cache line can be composed of 32, 64, or 128 bytes. Assuming that the cache line is 64 bytes, 16 variables of 4 bytes can be stored at once. For example, if accesses such as a[8i]*b[j] are cached, most accesses of b[j] will go into the cache line, but accesses of variable a[8i] will yield cache misses at every third access. The use of software-controlled cache rather than hardware-controlled cache shows better performance improvement in applications where data access is irregular or data locality is poor with the cache line size.

The advantage of a software-controlled cache is that it can be optimized the alignment of data variables as they are read from storage. By doing so, the accesses a[8i]*b[j] can be optimized to spm_a[i]*b[j] with the proposed technique.

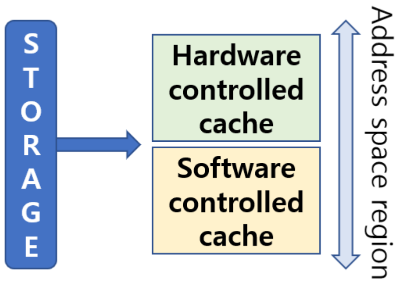

Figure 1. Memory architecture

The index variable of the array variable consists of three elements: offset, stride, and induction variable. With these three elements, data is copied from storage to cache and used for CPU operations. Stride represents the interval between values accessed when the address space of memory is viewed as a one-dimensional address space. Thus, it can be determined whether to place the data in HCC (Hardware-Controlled Cache) or SCC (Software-Controlled Cache) based on the stride and cache line size. Figure 1 shows the memory structure used in this study. In the target memory system, the hardware-controlled cache and the software-controlled cache are configured by dividing the main memory address space to optimally support system performance.

Figure 2 shows an example of accesses a[8i]*b[j] code. In the loop code, the data access pattern can be formulated as offset, access stride, and induction variable. A[8i] is data accessed by offset 0, stride 8, and induction variable i. B[j] is data accessed by offset 0, stride 1, and induction variable j. If the stride is out of the cache line like 8, the cache hit rate is low, so these data are allocated to the SCC. However, since only necessary data is copied, the data access pattern A[8i] stored in the SCC is changed to SCC_A[i]. The data access pattern code change algorithm is illustrated in Algorithm 1.

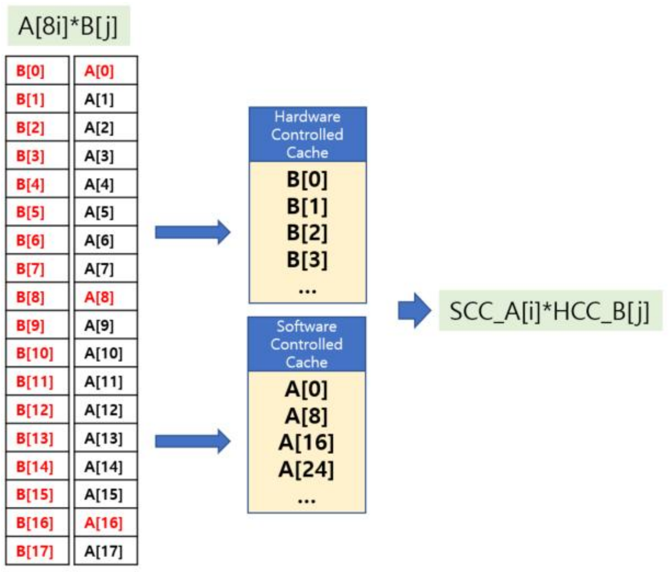
A[8i]*B[j]
B[0] B[1] B[2] B[3] B[4] B[5] B[6] B[7] B[8] B[9] B[10] B[11] B[12] B[13] B[14] B[15] B[16] B[17]
A[0] A[1] A[2] A[3] A[4] A[5] A[6] A[7] A[8] A[9] A[10] A[11] A[12] A[13] A[14] A[15] A[16] A[17]
Hardware Controlled Cache
B[0] B[1] B[2] B[3] ...
Software Controlled Cache
A[0] A[8] A[16] A[24] ...
SCC_A[i]*HCC_B[j]

Figure 2. An example for the memory architecture

Algorithm 1

```
Input: array access codes
Output: optimized array access codes

Begin
Get_data_access_pattern(stride, offset, induction_variable)

If( cache_miss(cache_line_size, stride) >= 50%)
    DMA_Copy(data, SCC)
Else
    DMA_Copy(data, HCC)

If( Confirm(data, SCC) )
    Optimize_access_pattern(data, stride)
End
```

In Algorithm 1, Get_data_access_pattern is a function that provides array access information stride, offset, and induction_variable in the loop code provided as inputs. Based on the array data access information given from the function, hardware-controlled cache simulation is applied, and if the cache miss rate exceeds 50%, data should be copied to the software-controlled cache by using DMA. Confirm(data, SCC) function checks whether the data copied to the SCC is correct, and if it is then it changes the stride of the data to 1, which is the stride placed on SCC.

3. Results and Conclusion

In order to evaluate the proposed technique, a simulation was performed in the environment of the memory structure as shown in Fig. 1. The in-house simulator operates with 18 cycles of main memory and 1 cycle of cache memories. HCC and SCC are configured with a setting based on 90nm process technology, and the main memory uses micron SDRAM setting. The cache memory architecture used in this experiment is a direct mapped hardware-controlled cache memory of 1 KB with 64 bytes of cache lines. A software-controlled cache memory of the same size as HCC is used. When the proposed technique is applied to a given loop code including array multiplication operation, the degree of performance improvement was calculated compared to the case with only the hardware control cache memory used. The evaluation program codes used performs 4x4, 16x16, and 32x32 matrix multiplication operations. Matrix multiplication consists of stride 8 and stride 16 data accesses, so the cache miss rate varies by 50% depending on the cache line 64B. Therefore, it is possible to clearly confirm how effectively the proposed technique is applied when the cache misses are high with data access stride 8 and stride 16.

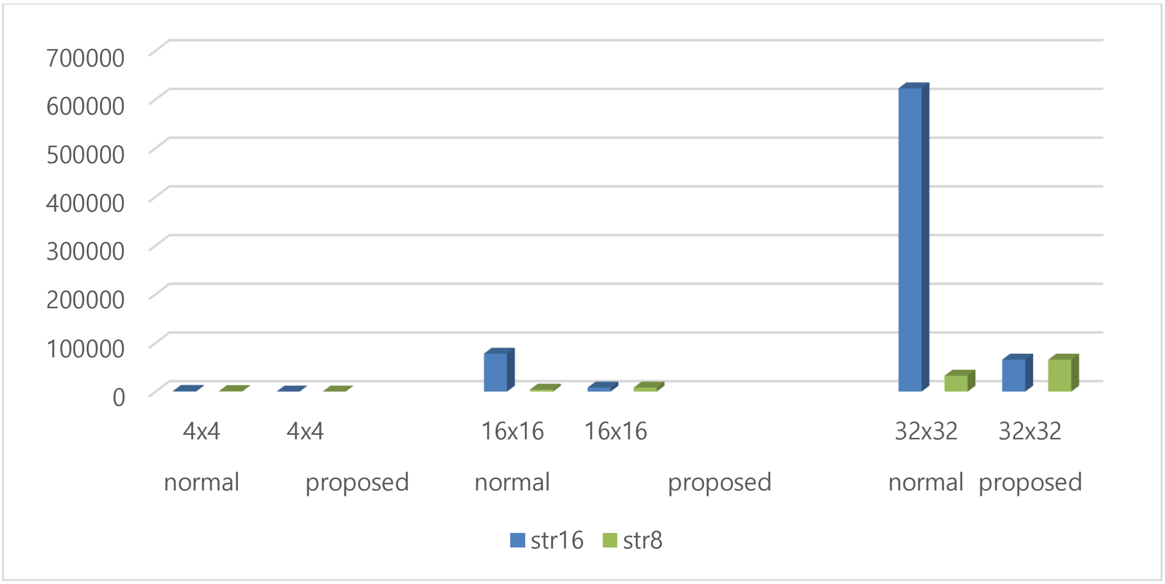

Figure 3. Experimental results

Figure 3 shows that are the execution time results of the proposed technique and the normal technique. With a traditional HCC, ***normal*** is a result of execution time of the matrix multiplication with a general array placement. ***proposed*** is a result improved by placing data to a software-controlled cache when a cache miss rate is greater than 50%. If the stride is 8, the cache miss rate is low because large portion of data can be included in the cache line 64B. Stride 16 exceeds the cache line, and the cache miss rate could be high. In the case of 32x32 ***normal***, the execution time is the highest because it has the highest amount of computation and the highest cache miss rate. In the case of ***proposed***, it was confirmed that both str8 and str16 can provide excellent cache hits with all the matrices size.

When the proposed technique was applied, an average performance improvement of 89% was confirmed. In applications where hardware-controlled cache memory performance is insufficient, such as video streaming codes, the proposed technique may provide the verified performance. This study introduces a software optimization technique required for a hybrid cache memory. It is expected that the hybrid cache memory can be effectively used with stream applications when the provided performance is insufficient. The proposed technique can be implemented as an independent source code optimizer or can be implemented in a compiler.

Acknowledgement

This work was supported by a research promotion program of SCNU.

References

[1] D. Cho, "A Study on Improvement of Low-power Memory Architecture in IoT/edge Computing", Journal of the Korean Society of Industry Convergence, Vol. 24, No. 1, pp. 69-77, Feb. 2021.
DOI: https://doi.org/10.21289/KSIC.2021.24.1.69

[2] J. Cho, J. Lee, D. Cho, "Efficient memory design for medical database", Basic & Clinical Pharmacology & Toxicology, Vol. 125, pp. 198, July. 2019.

[3] J. Cho, JM Youn, D. Cho, "An Automatic Array Distribution Technique for Multi-Bank Memory of High Performance IoT Systems", World, Vol. 3, No. 1, pp. 15-20, Nov. 2019.
DOI: https://gvpress.com/journals/WJWDE/vol3_no1/vol3_no1_2019_03.html

[4] D. Cho, "Study on Memory Performance Improvement based on Machine Learning", The Journal of the Convergence on Culture Technology, Vol. 7, No. 1, pp. 615-619, Feb. 2021.
DOI: https://doi.org/10.17703/JCCT.2021.7.1.615

[5] D. Cho, "Memory Design for Artificial Intelligence", International Journal of Internet, Broadcasting and Communication, Vol. 12, No. 1, pp. 90-94, Dec. 2020.
DOI: https://doi.org/10.7236/IJIBC.2020.12.1.90

[6] J. Yoon, D. Cho, "A spill data aware memory assignment technique for improving power consumption of multimedia memory systems", Multimedia Tools and Applications, Vol. 78, No. 5, pp. 5463-5478, Mar. 2019.
DOI: https://doi.org/10.1007/s11042-018-6783-x

[7] J. Cho,, J. Lee, D. Cho, "Low-End Memory Subsystem Optimization Process", International Journal of Smart Home, Vol. 13, No. 2, pp. 11-16, Oct. 2019.
DOI: http://dx.doi.org/10.21742/ijsh.2019.13.2.02

[8] J. Cho, D. Cho,, Y. Kim "Study on LLVM application in Parallel Computing System", The Journal of the Convergence on Culture Technology, Vol. 5, No. 1, pp. 395-399, Feb. 2019. DOI: http://dx.doi.org/10.17703/JCCT.2019.5.1.395

[9] J. Cho, D. Cho, "Development of a Prototyping Tool for New Memory Subsystem", International Journal of Internet, Broadcasting and Communication, Vol. 11, No. 1, pp. 69-74, Jan. 2019.
DOI: http://dx.doi.org/10.7236/IJIBC.2019.11.1.69