World Journal of Wireless Devices and Engineering
Vol.3, No.1 (2019), pp.15-20
http://dx.doi.org/10.21742/wjwde.2019.3.1.03

An Automatic Array Distribution Technique for Multi-Bank Memory of High Performance IoT Systems

Jungseok Cho[1], Jonghee M. Youn[2], and Doosan Cho[3]

[1]*Computer Engineering, Yeungnam Univ., South Korea*
[2,3]*Electrical & Electronic Engineering, Sunchon National Univ., South Korea*
[1]*youn@yu.ac.kr*, [3]*dscho@scnu.ac.kr*

Abstract

Mobile devices designed for IoT exploit a variety of system optimization techniques to maximize performance while reducing power consumption. These technologies apply to communication modules, to memory system, and to the central processing unit. Most of the technologies are developed and applied at the design stage of the system, but not many technologies are applied at the system integration stage. In the system integration stage, the major power consuming parts are the communication part and the memory part. Since communication has a lot of variables depending on the network environment, there are some limited technologies available, but in the case of memory, a large benefit can be obtained depending on the technology applied. Mobile or IoT system's memory structures can be classified in many different ways, of which we focus on multi-bank memory. Multi-bank memory refers to a method of dividing a large memory into several smaller memories. Using multi-bank memory can reduce operating power consumption and support parallel memory accesses, resulting in improved performance, which is often used in commercial products. A compiler should generate the access instruction and data placement properly. Therefore, the system performance is determined by the compiler performance. In this paper, we introduce a compiler optimization technique for multi-bank memory to overcome the compiler performance. The proposed technique can improve energy consumption by up to 20% in multi-bank memory systems.

Keywords: *Energy consumption, IoT system, Multi-bank memory, Memory structure, Compiler technique, System optimization*

1. Introduction

With the widespread use of a high-speed networked environment where all electronic devices are connected, a variety of new functional devices are being developed and commercialized to support them. Although the functions of these electronic devices vary, the hardware characteristics of the electronic devices are developed in a very similar form. They are designed to be low power, low cost constraints because they are usually small, battery powered. Designing devices with small, low-power, low-cost computing power requires processors and memory systems to support them. There are several commercially available processing systems for this purpose. In this paper, we discuss one of them called the coarse grained reconfigurable array architecture (CGRA).

Article history:
Received (May 28, 2019), Review Result (August 29, 2019), Accepted (October 10, 2019)

Print ISSN: 2207-5968, eISSN: 2207-5976 WJWDE

CGRA is a one-chip processing solution containing several low cost processors in an array. It was developed to meet low power requirements while providing high performance by connecting several processing elements with several small computing powers in a mesh network form. The form of a memory system containing data is very important for evenly using tens or hundreds of processing elements.

To provide wide data bandwidth with a single memory chip requires multiple input / output ports and a wide multiplexer, which consumes a lot of power. To solve this problem, the memory is divided into several small spaces, which are called multi-bank on-chip memory. Initially developed in a similar design, Morphosys architecture [1] consists of 16 banks of memory, with at least one read / write port per bank connected to the processing elements. More than one read port and one write port design are usually used, because multimedia applications require a lot of read operation than writes.

Thus, each memory bank support two data read and one write at the same time. To fully utilize this hardware feature, we present an array variable replacement technique. By using the proposed technique, it can be obtained to maximally use the memory bandwidth, and thus it leads to energy saving and performance improvement. The proposed technique optimizes the data placement of the source code in the preprocessing stage of an optimizing compiler. This compiler determines final data placement and modifies data transfer code from off-chip memory to on-chip memory. Because CGRA operates with on-chip memory, before its use, the whole data must be placed to on-chip memory.

Many commercial processor architectures employ multi-on-chip memory bank designs to reduce energy consumption. Since the multi-memory bank is composed of a small number of read and write ports for each memory bank chip and a smaller multiplexer supporting the same, the operating power consumption is low and the multiple memory banks can be accessed simultaneously, thereby providing high performance support. However, there is a disadvantage that system performance depends on compiler performance because data allocation and use must be coordinated by compiler which is system software. Processors that have been successfully commercialized with superior compilers and multi-on-chip memory bank designs include the Motorola 56000 [2] and Gepard Core DSPs [3][4]. Since the design of the commercialized architecture is open, but the compiler technology supporting it is not disclosed, the latecomers have to develop their own compiler technology, but the commercialization has failed in many cases. This paper introduces a compiler technique that can support multi-bank on-chip memory at low cost.

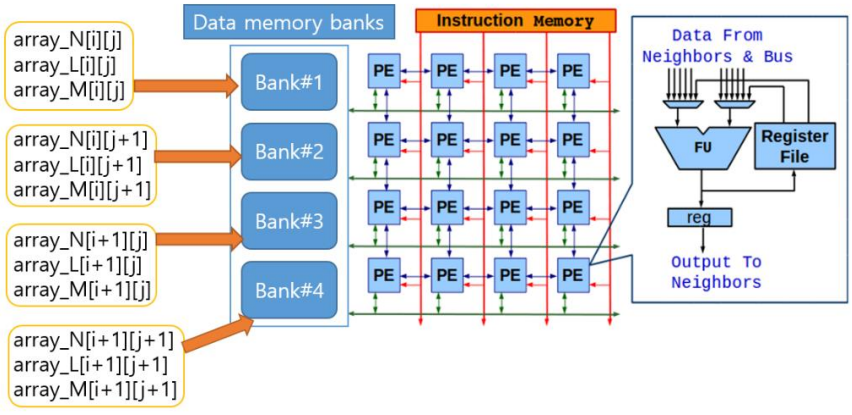

Figure 1. Multiple memory banks on a coarse grained reconfigurable array (CGRA) architecture

In an IoT-like environment, much of the data stream is taken up by the multimedia data stream rather than the device's control signals. Multimedia applications [5] implement a wide variety of algorithms, all of which share common algorithms. Representatively, techniques such as FIR, IIR, MATRIX, LAPLACE, and FOURIER are mixed. This is accomplished by large-scale add and multiply operations, which makes multi-memory bank designs a more popular choice because the wide bandwidth for the data streams has a significant impact on system performance.

A compiler makes program codes considering with the number of processing elements, mesh network and the number of ports of multiple memory banks. This code making process is called code mapping and data mapping. Our work focuses on data mapping only, since the problem of code mapping is beyond this research purpose.

Fig. 2 (a) shows matrix multiplication, which is common in multimedia codes. Fig. 2 (b) shows the code of unrolled (a). Unrolling is a technique applied to maximize data parallelism in the loop code. As shown in Fig. 2 (b), each array_N, array_L, and array_M are accessed by index variables i and j. If the accesses for the same array variable are orthogonal, they can be allocated to separated memory banks to enable simultaneous accesses. By using maximally concurrent accessing, program execution can be accelerated and energy consumption can be reduced.

```
int array_L[100][100], array_M[100][100], array_N[100][100];

for(i=0; i<100; i++)
   for(j=0; j<100; j++)
      array_N[i][j] = array_L[i][j]*array_M[i][j];

(a) Matrix multiplication code


for(i=0; i<50; i+=2)
   for(j=0; j<50; j+=2){
      array_N[i][j]     = array_L[i][j]      * array_M[i][j];
      array_N[i][j+1] = array_L[i][j+1] * array_M[i][j+1];

      array_N[i+1][j]     = array_L[i+1][j]      * array_M[i+1][j];
      array_N[i+1][j+1] = array_L[i+1][j+1] * array_M[i+1][j+1];
    }

(b) unrolled code of (a)
```

Figure 2. An example codes

[Figure 1] shows a coarse grained re-configurable array processor architecture (CGRA). CGRA consists of hundreds of processing elements and a number of memory banks to efficiently process data streams. Figure 1 shows CGRA in a simplified manner. When assigning the code in [Figure 2] (b) to CGRA, the data placement should be determined to maximize data parallelism. As shown in [Figure 1], orthogonal accesses can be allocated to different memory banks to support concurrent accesses. Thus, to improve memory access latency, the arrays that have index [i][j], [i][j+1], they should distribute into independent banks of memory. This data mapping supports parallel accessing.

Specifically, we present an algorithm that performs an array replacement for exploiting multiple memory banks driven by compiler analysis techniques. The proposed technique analyzes the indexes of program code and maps these data to different memory banks if it is

confirmed that the accesses are independent of each other. This mapping parallelizes memory accesses to improve system performance and energy consumption. We will present the experimental results based on the multimedia kernels by applying the proposed technique to a compiler platform.

2. Overall workflow

The workflow of our approach is shown in [Figure 3]. The first step in the proposed technique is to extract the loop code from a given program. Because the loop code takes more than 90% of the execution time, our technique optimized array references in loop codes. The next step is to extract the array references from extracted loop codes. If the references accessing the same array use mutually exclusive regions, it can be classified them to independent set. Then, they can be allocated to different memory banks. Otherwise, we put them to the same bank, and derive a single unified array variable placement.

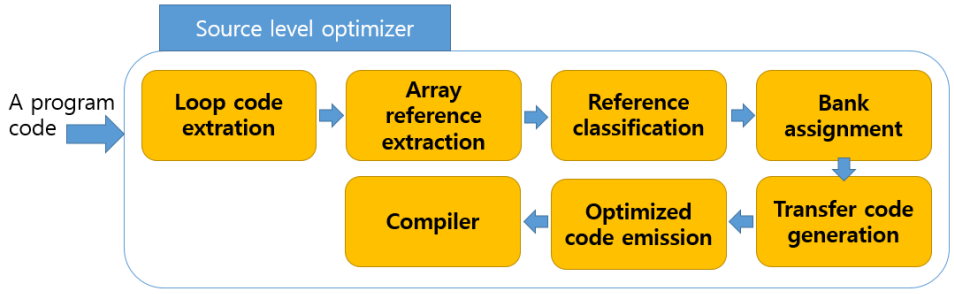

Figure 3. A workflow of the proposed approach

In the next step, array replacement is performed to determine placement of classified arrays to multiple memory banks. Since the size of each memory bank is limited, the size of the array variable that can be moved at one time should be determined, and bank assignment is also determined with the array references. Once the bank allocation is determined, the data transfer code from the off-chip memory to the on-chip memory bank must be generated and inserted into the loop code. When the source code conversion is completed, the final generated code is used as input to the compiler. Specifically, the optimizer rewrites each array reference so that the transformed subscript expression takes into account the position within the newly formed array in the mapped memory bank. Based on the transfer and allocation decision, optimizer insert array variable transfer code to/from multiple memories.

An important part of the proposed technique is to analyze how to distinguish index functions that access memory without overlapping sections. To analyze this, we represent each array index as a one-dimensional function and see if there is something in common with the set of data indexes accessed in loop space. This could be solved by linear algebra.

3. Related works

Previous studies on CGRA [6][7][8] can be largely separated into code mapping and data mapping studies. The study of code mapping focuses on efficiently allocating operations to hundreds of processing elements. Data mapping research analyzes applications and now focuses on designing optimized memory system architectures and determining data mappings optimized for existing architectures. In most cases, one-dimensional memory structures are considered, but hierarchical memory structures [9] are sometimes considered. Early CGRA

mapping studies [10] considered scratch pad memory as the main element of memory, but recent researches considering non-volatile memory elements [11] have become mainstream. To the best our knowledge, universal techniques have not yet been introduced to provide optimal data mapping for various memory structures.

Prior to data mapping, the main research flows were studies using data locality [12][13][14][15]. Data locality allows you to take advantage of many loop transformations that are positive for system performance. For example, reuse analysis is very useful when using data prefetch. This increases the hit rate of the cache memory and improves system performance. Because loop transformation technology is accompanied by code transformation, it must include proof of maintenance of correctness.

In architectures that use distributed and shared memory systems, such as CGRA, various optimization techniques are used. Data replication [16][17][18] technology is one of them. If the memory bandwidth is limited when the various processing elements share data, it is possible to replicate the data to extend the software perspective. Data renaming technology is a further improvement of the data replication technique. This is a way to improve program optimization by assigning different names to the same data rather than using the same name for multiple processing elements that use reads simultaneously. Array-scalar transformation flow analysis techniques [19][20] can optionally be applied when arrays interfere with data partitioning. Since arrays are allocated in contiguous space, various optimizations can be applied by selectively scalar-converting regularly used data. For example, code scheduling is a typical optimization technique. Scheduling analyzes data parallelism to make the best use of the hardware resources of a given architecture. In most cases, the complex dependencies of arrays limit performance significantly. Converting highly dependent data into scalars can help alleviate complex dependencies and significantly improve scheduling performance.

4. Conclusion

In this paper, we present an algorithm for efficiently determining array variable placements in multiple memory banks for array-based computations, to facilitate high-bandwidth concurrent memory accesses in modern high performance mobile architectures. The proposed technique automatically generates array variables placement to maximize data parallelism. Our technique focuses on loop nest computations, since most of execution time spend by the loop code. Depending on the memory architecture and the application, such array data replacement could be more attractive increasing array data parallelism. A major focus of our current work is to formulate this array reference replacement optimization as a simple problem. By using our technique, the experimental results show that the average improvement on energy saving is 16.1% with some loop kernel of multimedia applications. The future work is to mathematically formulate the proposed technique and generalize it and apply it to various memory architectures.

Acknowledgements

This work was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF - 2018R1D1A1 B07050054).

References

[1] Hartej Singh, Guangming Lu, Eliseu Filho, Rafael Maestre, Ming-Hau Lee, Fadi Kurdahi, and Nader Bagherzadeh, "Morphosys: case study of a reconfigurable computing system targeting multimedia applications," In Proceedings of DAC, pp.573-578, **(2000)** DOI: 10.1145/337292.337583

[2] Jean-Francois Collard and Daniel Lavery, "Optimizations to prevent cache penalties for the intel Itanium 2 processor," In Proceedings of the CGO, pp.105-114, **(2003)**

[3] P. Grun, N. Dutt, and A. Nicolau, "Access pattern based local memory customization for low power embedded systems," In Proceedings of the conference on DATE, pp.778-784, **(2001)** DOI: 10.1109/DATE.2001.915120

[4] M. Gupta and P. Banerjee, "Demonstration of automatic data partitioning techniques for parallelizing compilers on multicomputers," IEEE Trans. Parallel Distrib. Syst., vol.3, no.2, pp.179-193, **(1992)** DOI: 10.1109/71.127259

[5] Chunho Lee, Miodrag Potkonjak, and William H. Mangione-Smith, "MediaBench: a tool for evaluating and synthesizing multimedia and communications systems," In Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture (MICRO 30), Washington, DC, USA, 330-335, **(1997)** DOI: 10.1109/MICRO.1997.645830

[6] Hyunchul Park, Kevin Fan, Manjunath Kudlur, and Scott Mahlke, "Modulo graph em-bedding: mapping applications onto coarse-grained reconfigurable architectures," In Proceedings of CASES, pp.136-146, **(2006)** DOI: 10.1145/1176760.1176778

[7] A. Hatanaka and N. Bagherzadeh, "A modulo scheduling algorithm for a coarse-grain re-configurable array template," In Proceedings of the IPDPS, pp.1-8, **(2007)** DOI: 10.1109/IPDPS.2007.370371

[8] Yoonjin Kim, Mary Kiemb, Chulsoo Park, Jinyong Jung, and Kiyoung Choi, "Resource sharing and pipelining in coarse-grained reconfigurable architecture for domain-specific optimization," In Proceedings of DATE pp.12-17, **(2005)** DOI: 10.1109/DATE.2005.260

[9] Kathryn McKinley and Steve Carr, "Improving data locality with loop transformations," ACM Transactions on Programming Languages and Systems, vol.18, pp.424-453, **(1996)** DOI: 10.1145/233561.233564

[10] B. Mei, S. Vernalde, D. Verkest, H. De Man, and R. Lauwereins, "ADRES: An architecture with tightly coupled VLIW processor and coarse-grained reconfigurable matrix," In Proceeding of Field Programmable Logic, FPL, pp.61-70, **(2003)** DOI: 10.1007/978-3-540-45234-8_7

[11] Michael Joseph Wolfe, "High performance compilers for parallel computing," Addison-Wesley Longman Publishing Co., USA, **(1995)**

[12] Michael E. Wolf, Dror E. Maydan, and Ding-Kai Chen, "Combining loop transformations considering caches and scheduling," In MICRO, pp.274-286, **(1996)**

[13] Michael E. Wolf and Monica S. Lam, "A data locality optimizing algorithm," In Proceedings of the ACM SIGPLAN, pp.30-44, **(1991)**

[14] Wei Li, "Compiling for numa parallel machines", Ph.D. dissertation, Ithaca, NY, USA, **(1993)**

[15] X. Pan, A. Bacha and R. Teodorescu, "Respin: Rethinking near-threshold multiprocessor design with non-volatile memory," IEEE International Parallel and Distributed Processing Symposium (IPDPS), Orlando, FL, , pp.265-275, **(2017)** DOI: 10.1109/IPDPS.2017.109

[16] S. Lumetta, L. Murphy, X. Li, D. Culler, and I. Khalil, "Decentralized optimal power pricing: The development of a parallel program," In IEEE Parallel and Distributed Technology, pp.240-249, **(1993)** DOI: 10.1109/SUPERC.1993.1263450

[17] Kai Li, "Shared virtual memory on loosely coupled multiprocessors," Ph.D. dissertation, **(1986)**

[18] Daniel Edward Lenoski, "The design and analysis of DASH: a scalable directory-based multiprocessor," Ph.D. dissertation, Stanford, CA, USA, **(1992)**

[19] Chau wen Tseng, "Compiler optimizations for eliminating barrier synchronization," ACM SIGPLAN, pp.144-155, **(1995)** DOI: 10.1145/209936.209952

[20] V. Balasundaram and K. Kennedy, "A technique for summarizing data access and its use in parallelism enhancing transformations," In Proceedings of the ACM SIGPLAN, pp.41-53, **(1989)** DOI: 10.1145/73141.74822