Low-Power Low-Latency Data Allocation for Hybrid Scratch-Pad Memory

Meikang Qiu, *Senior Member, IEEE*, Zhi Chen, and Meiqin Liu, *Senior Member, IEEE*

***Abstract*—This letter aims at developing new memory architecture to overcome the daunting memory wall and energy wall issues in multicore embedded systems. We propose a new heterogeneous scratch-pad memory (SPM) architecture that is configured with SRAM, MRAM, and Z-RAM. Based on this architecture, we propose two algorithms: a dynamic programming (MDPDA) and a genetic algorithm (AGADA) to allocate data to different memory banks, therefore, reducing memory access cost in terms of power consumption and latency. Extensive experiments are performed to show the merits of the hybrid SPM memory architecture and the effectiveness of the proposed algorithms.**

***Index Terms*—Data allocation, dynamic programming, genetic algorithm, hybrid memory, multicore, scratch-pad memory (SPM).**

I. Introduction

FOR multicore embedded systems, low-power and low-latency memory access are the two most important metrics of performance. Today, the development of current embedded systems is substantially hindered by the daunting *memory wall* and *power wall* issues.

Therefore, how to develop alternative cost-efficient techniques to replace the current hardware-managed cache memory is really challenging. *Scratch-pad memory* (SPM), a software-controlled on-chip memory, has been widely employed by key manufacturers. Also, heterogeneous architecture [1] and hybrid memory architecture have emerged as the potential solution.

To take advantage of the benefits of hybrid SPM memory, we must strategically allocate data on each memory module so that the total memory access cost can be minimized.

Traditional hybrid memory data management strategies, such as data placement and migration [2], [3], are unsuitable for hybrid SPMs, because they were mainly designed for hardware caches and were unaware of write activities. Fortunately, embedded system applications can fully take the advantage of compiler-analyzable data-access patterns, which can offer efficient data allocation mechanisms for hybrid SPM architecture.

There are many approaches in the literature to solve data allocation problems, such as ILP, dynamic programming, and heuristic approaches. ILP methods have garnered wide interests in recent years, because it can achieve optimal solutions for the problems in consideration. However, in the data allocation context for multicore architectures, both time and space complexities are critical factors. Heuristic methods are fast and require less memory, but usually perform poorly for guaranteeing good solutions. Generally, a dynamic programming algorithm can derive optimal solutions for problems at the acceptable time overhead. It is an important technique aimed at addressing optimization problems by breaking them down into some subproblems which can be solved optimally within polynomial complexity.

We design a *multidimensional dynamic programming data allocation* (MDPDA) strategy to allocate data on different memory modules in polynomial time. Compared with the dynamic programming proposed by Sha *et al.* [4], MDPDA is used for a more complicated architecture, targeting the data allocation problem for the heterogeneous on-chip SPM memory with SRAM, MRAM (*magnetic RAM*), and Z-RAM (*zero-capacitor RAM*). Also, MDPDA focuses on multicore embedded systems while [4] targets single processor platforms. To the best of our knowledge, this is the first paper to address the data allocation issue for multicore embedded systems with hybrid SPMs comprising three types of memory modules. The goal of our proposed algorithm is to minimize the overall cost (energy and latency) incurred by memory accesses.

Dynamic programming consumes a significant amount of time and space. Being awaren of this issue, we design a genetic algorithm, called *adaptive genetic algorithm for data allocation* (AGADA) because it can obtain near-optimal solutions with moderate time and space overhead [5]. Our genetic algorithm AGADA inherits the prominent merits of traditional ones, such as accurate solutions and fast convergence. It involves the following basic elements: chromosome, initialization, selection, reproduction, and termination.

The major contributions of this paper are the following. 1) We propose a hybrid SPM architecture that consists of SRAM, MRAM, and Z-RAM. This architecture produces high-access performance with low-power consumption. 2) We propose a multidimensional dynamic programming (MDPDA) data allocation strategy to reduce memory access latency and power consumption, along with cutting the number of write activities on MRAM. The reduction of writes on MRAM

Manuscript received May 18, 2014; accepted June 21, 2014. Date of publication August 05, 2014; date of current version November 20, 2014. This work was supported by the NSF-CNS-1359557 and the Open Research Project of the State Key Laboratory of Industrial Control Technology, Zhejiang University, China ICT1441. This manuscript was recommended for publication by Z. Shao.

M. Qiu is with the Computer Science Department, Pace University, New York, NY 10038 USA (e-mail: mqiu@pace.edu).

Z. Chen is with University of Kentucky, Lexington, KY 40506 USA.

M. Liu is with College of Electrical Engineering, Zhejiang Univesity, ZJ 310027, China (e-mail: liumeiqin@zju.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/LES.2014.2344913

will efficiently prolong their lifetime. 3) We further propose an adaptive genetic algorithm (AGADA) with very limited sacrifice in the accuracy of solutions.

II. System Model

A. Hardware Model

The architecture of a target multicore system with hybrid SPMs is as follows. Each core is tightly coupled with an on-chip SPM that consists of a SRAM, a MRAM, and a Z-RAM. We call a core that accesses the SPM owned by itself as *local access*, while accessing an SPM held by another core is referred to as *remote access*. Generally, the remote access is supported by an on-chip interconnect. All cores access the off-chip main memory (usually a DRAM device) through a shared bus. There is a multichannel ring structure to allow the communication between any two cores without intervention from other cores. Consequently, we can safely assume that the data transfer cost between cores is constant. Generally, accessing the local SPM is faster and dissipates less energy than fetching data from a remote SPM, while accessing the off-chip main memory incurs the longest latency and consumes most energy.

B. Problem Statement

The *cost optimization problem of memory access* incurred by data allocation in a multicore with P cores can be defined as: Given the number of data N, the initial data allocation on the on-chip memory units of all cores and the off-chip main memory, the capacity of each core's SRAM and MRAM, the number of cores P, the number of reading and writing references to each data of each core, the cost of each memory unit access, and the cost of moving data between different memory units, the question is how do we allocate each data to the hybrid memory units of each core so that the total memory access cost can be minimized and the write activities on MRAMs be reduced? In this problem, we assume each core can access the off-chip main memory, the SRAM and MRAM in its local SPM, and every remote SPM has its own cost.

The *objective function* of the target problem is described as: Given the number of local reads N_{LR}, local writes N_{LW}, remote reads N_{RR}, remote writes N_{RW}, the cost of local read C_{LR}, local write C_{LW}, remote read C_{RR}, remote write C_{RW}, and the cost of data movement C_{Move}, the cost of memory access (CM) for a specific data can be formulated as (1)

$$\mathrm{CM} = N_{\mathrm{LR}} \times C_{\mathrm{LR}} + N_{\mathrm{LW}} \times C_{\mathrm{LW}} + N_{\mathrm{RR}} \times C_{\mathrm{RR}} + N_{\mathrm{RW}} \times C_{\mathrm{RW}} + C_{\mathrm{Move}}. \quad (1)$$

III. Description of the Multidimensional Dynamic Programming Algorithm MDPDA

In this section, we present our MDPDA algorithm in detail. We will first build an *allocation cost* table. Then, with the help of this table, we give the procedures of MDPDA algorithm.

A. Allocation Cost Table

Assume there are N data blocks need to be allocated to a system with P cores. Each core has a proposed hybrid SPM configured from a SRAM, a MARM, and a Z-RAM. In order to calculate latency and energy consumption for each data conveniently, we build an allocation cost table to represent the cost of allocating each data block to different memory modules. We use a function $Map(b_i, x)$ to represent the cost (either latency or energy consumption) of mapping data block b_i to memory module x, and the value of the function can be read from the allocation cost table directly. Let Ci_j represent the memory j of the SPM in core i, where $\forall i, j, \{i, j | i < P, j \in \{MM, S, M, Z\}\}$, P is the nubmer of cores, MM, S, M, Z are short for the main memory, SRAM, MRAM, and Z-RAM, respectively. It is much lower that the latency of allocating other blocks to Core2's MRAM, because it is originally stored in there. Therefore, there is no data moving latency and energy consumption to allocate block B_{18} to Core2's MRAM. Since the latency cost is proportional to energy consumption, the reduction of latency will also contribute to reduction in energy consumption. Therefore, for simplicity, we just use latency for demonstration.

B. Recursive Formulation

The most critical part of a dynamic programming algorithm is the construction of the recursive formulation, which breaks down the target problems

$$\begin{cases} AllocMem(b_i, MM) = AllC[b_i - 1, s_1, m_1, z_1, \ldots, z_n] \\ AllocMem(b_i, C1_S) = AllC[b_i - 1, s_1 - 1, m_1, z_1, \ldots, z_n] \\ AllocMem(b_i, C1_M) = AllC[b_i - 1, s_1, m_1 - 1, z_1, \ldots, z_n] \\ AllocMem(b_i, C1_Z) = AllC[b_i - 1, s_1, m_1, z_1 - 1, \ldots, z_n] \\ \ldots \\ AllocMem(b_i, CP_Z) = AllC[b_i - 1, s_1, m_1, z_1, \ldots, z_n - 1] \end{cases} \quad (2)$$

$$\begin{cases} f(b_i, MM) = AllocMem(b_i, MM) + Map(b_i, MM) \\ f(b_i, C1_S) = AllocMemf(b_i, C1_S) + Map(b_i, C1_S) \\ f(b_i, C1_M) = AllocMemf(b_i, C1_S) + Map(b_i, C1_M) \\ f(b_i, C1_Z) = AllocMemf(b_i, C1_S) + Map(b_i, C1_Z) \\ \ldots \\ f(b_i, CP_Z) = AllocMemf(b_i, C1_S) + Map(b_i, CP_Z) \end{cases} \quad (3)$$

$$AllC[b_i, s_1, m_1, z_1, \ldots, z_n] = \min(f(b_i, MM), f(b_i, C1_S) f(b_i, C1_M), \ldots, f(b_i, CP_Z)). \quad (4)$$

First, we define a memory allocation function $AllocMem(n, x)$, which represents the total cost of the first $n-1$ blocks when the nth block is allocated to memory x. Then, we define a total cost function $f(n, x)$ to represent the total allocation cost of the first n data blocks when the nth block is allocated to memory x. For example, $f(4, C1_S)$ indicates the total allocation cost of the first four data blocks when block four is allocated to Core1's SRAM. We define a multidimensional matrix, $\vec{AllC}$, to store the total cost for data allocation. The dimension of $\vec{AllC}$ is $N \times size(C1_S) \times size(C1_M) \times size(C1_Z) \times \ldots \times size(CP_Z)$, where P is the number of input data blocks and $size(x)$ is the size of the memory x. For example, $AllC[4, 1, 2, 1, \ldots, 1]$ indicates the total cost for allocating the first 4 data blocks to on-chip hybrid SPMs, when the available space of the Core1's SRAM, Core1's MRAM, Core1's Z-RAM,, and Core P's Z-RAM is 1, 2, 1,…, 1, respectively. Then, we can compute the

total allocation cost by allocating block i to different memory modules as (3).

Equation (3) shows that the minimum allocation cost is always preserved, since the total allocation cost by adding the current data block is always selected from the best allocation scheme of the previous blocks and the current one. Equation (2)–(4) jointly exhibit the recursive formulation to derive the minimum total allocation cost for the target problem. In this equation, $AllC[bi, s_1, m_1, \ldots, z_n]$ records the minimum allocation cost when the available memory block for each of on-chip memory module are $s_1, m_1, \ldots, z_n$, respectively. Initially, if all memory blocks in SPMs (including SRAM, MRAM, and Z-RAM) are unavailable (which means $s_1 = m_1 = \ldots = z_n = 0$), then all the blocks will be assigned to the shared off-chip main memory. The allocation of a specific block is always determined by the optimal allocation of the previous data block.

For any other item in the matrix, the total cost is determined by both the allocation of the previous blocks and the cost of allocating this block to different memory modules. There are totally $3 \times P + 1$ choices to assign a data block, where P is the number of cores in the target CMP system. The dynamic programming algorithm always selects the combination that can achieve the minimum total cost for all present data blocks. According to the built recursive formulations, we describe the *multidimensional dynamic programming data allocation* (MPPDA) algorithm. The input of the MDPDA algorithm is N data blocks obtained by profiling tools, the constructed allocation cost table, and the total cost table. The output of the algorithm is the minimum total cost (latency or energy consumption) for the N data blocks.

IV. Description of the Genetic Algorithm AGADA

The whole procedure of our AGADA algorithm is described as follows. First, we need to generate the initial population. In this procedure, a number of chromosomes will be generated randomly. These chromosomes are random permutations of pairs of data and all memory units of a multicore system. After the initialization, the fitness value of each individual will be calculated according to (6). Then, a search process will be iteratively applied to determine the best solution for the data allocation problem until a termination condition is reached. The termination criterion includes two conditions: 1) the number of new generations exceeds a predefined maximum number of iterations; and 2) after a certain number of search (typically 500 or even more), a better solution is still unreachable. In each generation, the crossover and mutation operation will be carried out in terms of the predefined crossover rate PC and mutation rate PM. Finally, based on the new population, the fitness value of each individual will be calculated and the selection operation will be employed to generate a new population.

A. Initialization

The population size $PopSize(PS)$ usually depends on the proposed problem and is determined experimentally [6]. To accelerate the process of data allocation and the implementation of genetic operations, we will use the greedy algorithm in [2] to generate the initial population. A whole population will be generated from these initial individuals by randomly swapping the memory positions of genes.

B. Fitness Function

In general genetic algorithms, the fitness function is typically obtained from the objective function that needs to be optimized. The fitness of an individual u is regarded to be better than the fitness of another individual v if the solution corresponding to u is closer to an optimal solution than v. According to Darwin's principle of survival of the fittest, the individual with a greater fitness value will have higher likelihood to survive in the next generation than the counterpart with a lower fitness value. We define the fitness function as (5)

$$FT(i) = M - Total\_Cost(i) \tag{5}$$

where M represents maximum total cost have observed by this generation and $FT(i)$ represents the fitness value of chromosome i. $Total\_Cost(i)$ is the total cost of memory access to the chromosome i. Essentially, it equals to the total memory access cost of each gene (data) in this chromosome. We calculate the total cost by using (6)

$$Total\_Cost(i) = \sum_{j=1}^{N} CM(j), \text{for chromosome } i \tag{6}$$

where N is the number of data items and $CM(j)$ is the memory access cost of data j that is defined as (1).

C. GA Operations

1) Selection: The selection process is carried out to form a new population, through strategically choosing some chromosomes from the old population with respect to the fitness value of each individual. It is utilized to enhance the overall quality of the population. Based on the natural selection rule, many methods are exploited to select the fittest chromosomes, such as roulette wheel selection, Boltzman selection, rank selection, and elitism, etc. In our genetic algorithm, we will use a *rank-based roulette wheel selection scheme* with elitism to select chromosomes. In this method, an imaginary wheel with a total 360 degrees applied, on which all chromosomes in the population are placed, and each of them occupied a slot size according to the value of the corresponding fitness function.

2) Crossover: Crossover is a crucial step after selection. Generally, it is employed to more broadly explore the search space. We can find the individual with higher fitness function with this operation. Conventionally, crossover operation includes signal point crossover, two point crossover, and uniform crossover. The rationale is that the "good" characteristics of the parents should be well preserved and passed down to children. However, the rational selection may lead to the local optimal problem. To avoid this problem, the crossover operations are carried out with a specific probability, which is often referred to as *crossover rate*, denoted by PC. We randomly select pairs of chromosomes as parents to generate new individuals. In this section, we will use an *adaptive cycle crossover strategy* to perform the crossover operation with a tunable crossover rate which is proposed in [7].

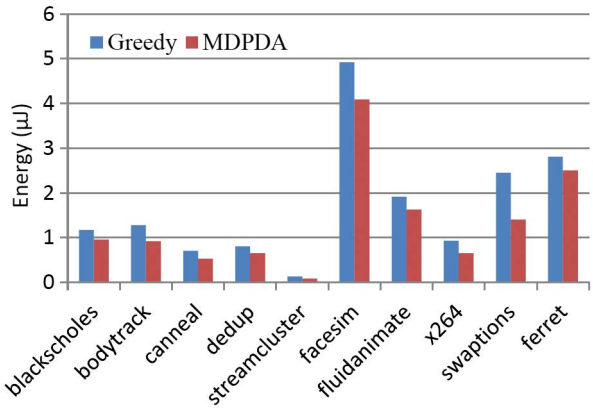

Fig. 1. Comparison of MDPDA and the greedy algorithm on quad-core system.

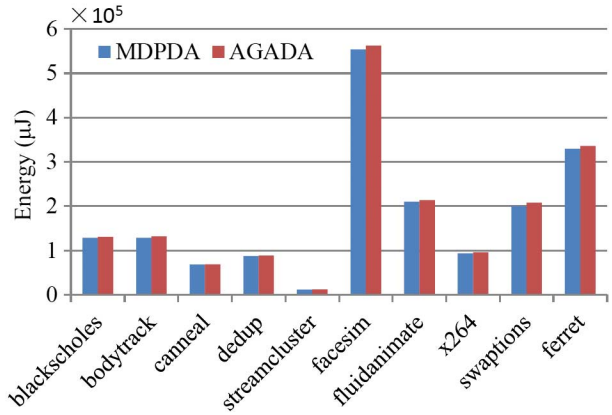

Fig. 2. Comparison of AGADA and MDPDA.

3) Mutation: After the crossover operation, a genetic mutation will be performed to recover some good features eliminated by the crossover and prevent the premature convergence to a local optima. It is archived by randomly flipping bits of a chromosome. Similar to the crossover, it is happened in a certain specific probability that is called *mutation rate* and denoted as PM. We will swap the number of memory units of two genes to achieve the mutation.

V. Experimental Results

We evaluate our algorithm across a host of benchmarks selected from PARSEC [8]. We run these workloads on M5 simulator [9] and obtain the memory traces for them. We implemented both of the MDPDA algorithm, the adaptive genetic algorithm, and the greedy algorithm as stand-alone programs. These programs take the memory traces we have collected as inputs. We also use a modified version of CACTI [10] to get the memory parameters, including memory read/write latency, energy consumption, and leakage power, for the simulations by using 65 nm technology. The following parameter specifications are used in our simulations for the AGADA algorithm: 1) Population size: 300; 2) Crossover rate: 0.8; 3) Mutation rate 0.02; 4) Selection method: rank-based roulette wheel; 4) maximum generation: 1000.

A. Comparison of MDPDA and the Greedy Algorithm [2]

Fig. 1 demonstrates that the MDPDA algorithm outstrips the greedy algorithm [2] in terms of energy efficiency. On average, the MDPDA algorithm can reduce the dynamic power consumption for quad-core system 24.18%, compared to the greedy algorithm [2]. From other results we have done, we can see that the reduction in energy consumption is proportional to the reduction in memory access latency. The reduction is mainly because of the optimal allocation of the MDPDA for each data block at each step.

B. Comparison of AGADA and MDPDA

Fig. 2 shows that dynamic energy consumption of the AGADA algorithm is approximate to that of the optimal dynamic programming algorithm MDPDA, with respect to the seven applications selected from PARSEC. On average, the AGADA consumes 2.21% more dynamic power than that of the multidimensional dynamic programming algorithm counterpart. However, considering the high time and space complexity of the multidimensional dynamic algorithm revised from [4], the AGADA algorithm is more competitive in overall performance.

VI. Conclusion

This letter proposed a novel hybrid SPM architecture comprising SRAM, MRAM, and Z-RAM to reduce memory access latency and energy consumption by taking the best of each type of memory. Based on the hybrid SPM architecture, we proposed two novel algorithms, one is a multidimensional dynamic algorithm called MDPDA, and the other is an adaptive genetic algorithm called AGADA, to efficiently allocate data on each memory unit of the heterogenous SPMs. AGADA has much smaller space complexity compared to MDPDA. Experimental results demonstrate the effectiveness of the proposed architecture and algorithms.

References

[1] M. Qiu and E. H.-M. Sha, "Cost minimization while satisfying hard/soft timing constraints for heterogeneous embedded systems," *ACM Trans. Design Autom. Electron. Syst. (TODAES)*, vol. 14, no. 2, pp. 1–30, 2009, (Best Paper Award).

[2] S. Udayakumaran and R. Barua, "Compiler-decided dynamic memory allocation for scratch-pad based embedded systems," *CASES*, pp. 276–286, Oct. 2003.

[3] G. Sun, X. Dong, Y. Xie, J. Li, and Y. Chen, "A novel architecture of the 3D stacked MRAM l2 cache for CMPs," in *Proc. ISCA*, 2009, pp. 239–249.

[4] J. Hu, C. J. Xue, Q. Zhuge, W.-C. Tseng, and E. H.-M. Sha, "Towards energy efficient hybrid on-chip scratch pad memory with non-volatile memory," in *Proc. DATE*, 2011, pp. 1–6.

[5] Z. Chen, M. Qiu, J. Niu, Z. Lu, and Y. Zhu, "Data allocation using genetic algorithm for MPSoC systems with hybrid scratch-pad memory," in *Proc. IEEE RTAS*, Beijing, China, Apr. 2012, pp. 61–64.

[6] E. S. H. Hou, N. Ansari, and H. Ren, "A genetic algorithm for multiprocessor scheduling," *IEEE Trans. Parallel Distrib. Syst.*, vol. 5, no. 2, pp. 113–120, Feb. 1994.

[7] M. Srinivas and L. M. Patnaik, "Adaptive probabilities of crossover and mutation in genetic algorithms," *IEEE Trans. Syst., Man Cybern.*, vol. 24, no. 4, pp. 656–667, Apr. 1994.

[8] "Parsec," [Online]. Available: http://parsec.cs.princeton.edu/

[9] N. L. Binkert, R. G. Dreslinski, L. R. Hsu, K. T. Lim, A. G. Saidi, and S. K. Reinhardt, "The M5 simulator: Modeling networked systems," *IEEE Micro*, vol. 26, no. 4, pp. 52–60, Jul-Aug. 2006.

[10] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, "Optimizing NUCA organizations and wiring alternatives for large caches with CACTI 6.0," in *Proc. MICRO*, Dec. 2007, pp. 3–14.