Genetic Algorithm for Hybrid on-Chip Memory

Naveen Vijay, Silpi Dutta, Prabhakar Poudel, Ranjan J
Department of Computer Science and Engineering
T. John Institute of Technology
Bengaluru, India

Roopashree S
Assistant Professor
Department of Computer Science and Engineering
T. John Institute of Technology
Bengaluru, India

***Abstract*— In an era of computation, speed is a major criterion. With the advent of chip multiprocessor (CMP) systems, it's exigent for an innovative strategy to bypass the improficiency in present memory system & architecture. In accordance to the above, frequent on chip memory access have increased analytical challenges in delivering high memory access performance with compact power and latency. The generalized concept of Scratch Pad Memory (SPM) can be configured from SRAM, MRAM & Z-RAM to evolve a heterogeneous SPM architecture. In this paper, we focus on uplifting latency & reducing power consumption. We have used, Adaptive Genetic Algorithm for Data Allocation (AGADA) for allocating data to above mentioned memory units forming the architecture along with test results.**

Keywords — Chip multi processor, scratch pad memory, genetic algorithm, data allocation.

I. INTRODUCTION

Chip multi-processor (CMP) is a single computing component with two or more actual cores (processor units) which are used for reading and executing program instruction. This CMP has two important metrics of performance that are low power consumption and short latency memory access [1].

To overcome the problem and to bridge the processor-memory speed gap traditional computing system adopted the cache mechanism. Caches cause notorious problem to CMP system. Therefore, an alternative technique is used to replace a cache that is Scratch Pad Memory (SPM) [2], software controlled on chip memory.

This SPM has two major advantages: first, SPM does not have the comparator and tag SRAM technique and second, SPM generally guarantees single cycle access latency [5].SPM is used widely in CMP system because of SPM beneficial advantages in size, power consumption and predictability [4].

The most crucial task for compilers is to manage SPM characteristics in form of data allocation on current system. Hybrid SPM architecture must resolve certain problems like no of write operations to MRAM (Magneto resistive random access memory), memory access latency etc.

Data is allocated on each memory module to reduce the total memory access cost. Data allocation employs a method called multi dimensional dynamic programming method (MDPDA) [9] to overcome hybrid SPM data allocation problems. This method consumes a significant amount of time and space. Hence, we employed genetic algorithm for data allocation.

Genetic algorithm is a class of computational models which organize a solution candidate of a problem in a specific data structure, example: linear binary, tree, linked list etc and applying some operations on them. Generally, genetic [10] algorithm based on initialization, selection, reproduction and termination.The major contributions of this paper include:
(1) A hybrid SPM architecture that consists of SRAM, MRAM, and Z-RAM. This architecture produces high access performance with low power consumption.
(2) A novel genetic algorithm based data allocation strategy to reduce memory access latency and while reducing the number of write operations to MRAM. The reduction of writes on MRAM will efficiently prolong their lifetime.

II. CACHE MEMORY

The most widely used systems in the current generation have a very fast memory known as the Cache Memory. In computing cache is a component that stores data so future requests for that data can be served faster. Cache Memory also called CPU memory is a Random Access Memory (RAM) that a systems microprocessor can access more quickly than it can access regular RAM. The memory is typically integrated directly with the CPU chip or can be placed on a separate chip. A CPU cache is used to reduce the average time to access data from the main memory. The cache is smaller, faster memory which stores copies of data from frequently used main memory locations for further reference that could be made on that data.

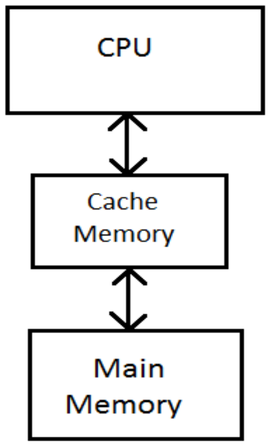

Fig 1: Cache Memory

There are two types of aftereffects for a search on a data stored on the caches which are the **cache hit** and the **cache miss** [3]. Cache hit occurs when the solicited data can be found, while a Cache miss occurs it cannot. Cache hits are served by reading data from the cache, which is faster than recomputing a result or reading from a slower data store. Whilst a solicitation for a data has been made, the data is transferred between memory and cache in blocks of fixed size called *cache lines*. When a cache line is copied from memory into the cache, a cache entry is created. A cache entry will include the copied data as well as the requested memory location. When the processor needs to read or write a location in main memory, it first checks for a correlated entry in the cache. The cache in turn checks for the contents for a requested memory locations in the cache lines that might contain that address. If the processor finds that the memory location is in the cache, then there is a cache hit or else it's a cache miss. In case of:

- A cache hit, the processor immediately reads or writes the data in the cache line.
- A cache miss, the cache allocates a new entry and copies in data from the main memory and consequently the solicitation is fulfilled from the contents of the cache.
- Though cache memories are so fast and so easy to use they have a few critical drawbacks.
- Since cache memory is not accessed using direct addressing they need to have a comparator and tag SRAM [6].
- Although, caches save a large amount of energy by not performing the complex decode operations to support the runtime address mapping for references, it is considered to be wastage of energy as their replacements save up to 40% more energy than cache memory [7].
- Access solicits to cache sometimes suffer from capacity, compulsory and conflict misses that incur a very long latency.

To prevail over the hindrance of cache memory, a new memory was developed called the Scratch Pad Memory (SPM).

III. SCRATCH PAD MEMORY

Scratch Pad Memory architecture is a potential replacement for traditional cache memory. SPM consists of a module made up of different kinds of memory units. The reason for the superiority of SPM over traditional cache memory is the fact that the various units of a SPM chips are programmed through software, in contrast to traditional homogeneous cache memory, which are programmed electronically. This helps a system with SPM to access memory with non-uniform latency.

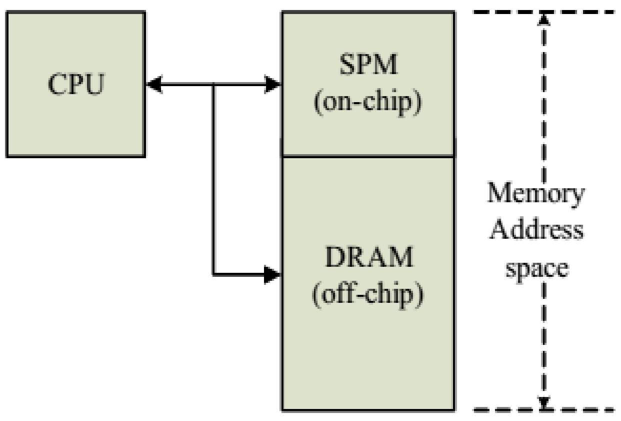

Fig 2: Scratch Pad Memory

Figure 2 shows the typical system with a scratchpad in place of cache memory. Here, the memory access space is always in a predetermined range, which facilitates access of data through direct memory addressing. This enables faster access as compared to the paged memory access method of cache.

Since programmers and compilers need to interfere in the memory retrieval process, the development of sophisticated mechanisms is a must to SPM management in order to achieve the above mentioned benefits. Data allocation in a SPM chip can be programmatically done either during compile time or during run time. Based on this, data allocation can be classified into static data allocation and dynamic data allocation respectively.

Static data allocation is the simpler of the two approaches, where the memory blocks are loaded into the memory during the initialization stage and remain unchanged during execution. This approach is fairly easy to implement and does not require as many resources as compared to dynamic data allocation.

Memory mapping is determined during run time in the case of dynamic data allocation. In addition to allocation of data dynamically, data can be reloaded into SPM at some required break points to guarantee the execution of the application. Therefore, the algorithm used for dynamic data allocation must know the contents of the memory at during allocation. This, together with the exorbitant cost for data mapping, makes dynamic allocation a problematic task.

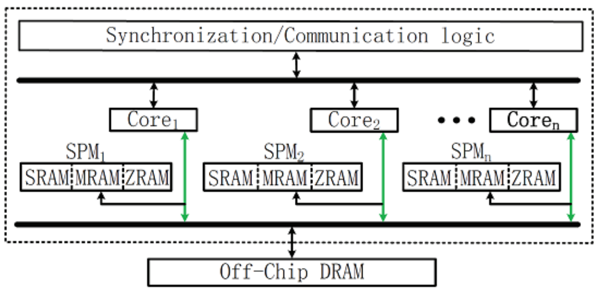

Fig 3: Multi-core hybrid memory structure

The architecture we propose consist of a SPM module made up of 3 separate memory units; namely SRAM, MRAM and ZRAM [8]; which are tightly coupled with the individual cores of the CMP system, as shown in the above figure. A core accessing an SPM coupled to it is called local access, and a core accessing a SPM coupled to another core is known as remote access. Remote access is usually implemented with the help of an interconnection among the different cores. The cores access the off-chip main memory through a shared bus. We may assume that the data transfer cost between cores is constant, since these CMP systems usually implement a multi-channel ring structure to facilitate isolated data transfer between cores. Local access is normally faster and more power efficient as compared to remote access, while the accessing of the main memory takes up the most power and time. In order to ensure a memory hit in the case of a heterogeneous module, the data must be written into the memory module in advance. We propose an Adaptive Genetic Algorithm for dynamic data allocation, which provides an efficient system to manage memory in the above SPM system.

IV. CHROMOSOME MODEL

A chromosome for the data allocation problem is a set of defined parameters which is able to represent a solution. The parameters here are the data blocks and the size of each memory module including all on-chip memory modules and the off-chip main memory. Therefore, we define a gene in a chromosome as a pair of these two parameters. That is, a chromosome represents an allocation scheme. There are numerous ways to represent a chromosome. Intuitively, we can use a matrix to represent a chromosome, where the rows indicate the main memory and all on-chip memory units of a SPM in each processor core. The columns indicate data allocation on the corresponding memories.

V. 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. 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 total
360 degrees is 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. Let PS denote the population size and **A_i** represent the angle of the sector occupied by the i^{th} ranked chromosome. The chromosome-to-sector mapping is consistent to the fitness of each chromosome, and the 1st ranked chromosome has the highest fitness value, therefore allocating to the sector 1 with the largest angle **A_1**. The $(PS)^{th}$ ranked chromosome has the lowest fitness value and is allocated to the sector PS−1withallest angle A_{PS} . The fitter an individual is, the more area of it will be assigned on the wheel, and thus the more possible that it will be selected when the biased roulette wheel is spun.

$$\rho = \frac{A_i}{A_{i+1}} \tag{1}$$

$$A_1 = \frac{1-\rho}{1-\rho^{PS}} \tag{2}$$

$$A_i = \frac{(1-\rho)}{1-\rho^{PS}} \times \rho^{i-1} \tag{3}$$

Where $A_i < 1$, $\rho < 1$, and $0 \le i \le PS$

<u>Algorithm-1:</u> <u>Algorithm for Genetic Selection</u>
Input: An old population Old_{Pop} and the size of the population *PS*.
Output: A selected chromosome *k*.

```
1: Define the total fitness SumFit as the sum of fitness values of all individuals in the current population;
2: for i = 1 → P S do
3:     $SumFit = SumFit + Old_{Pop}(i).FT$ ;
4: end for
5: Generate a random number RanN between 1 to SumFit;
6: for $k = 1 \rightarrow PS$ do
7:     if $\sum_{i=1}^{k} Old_{Pop}(i).FT \ge RanN$ then
8:         break;
9:     end if
10: end for
11: return chromosome k;
```

VI. CROSSOVER

Crossover is a crucial step after selection. 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.

The basic idea of cycle crossover works as follows.

$$PC = \frac{Q_C\left(FT_{max} - FT_{bestC}\right)}{\left(FT_{max} - FT_{avg}\right)} \tag{4}$$

Where FT_{max} is the maximal fitness value in the current population, FT_{bestC} is the fitness value of the parent with higher fitness value between the two crossover parents, FT_{avg} is the average fitness value of the current population, and␣c is a positive constant less than 1. We start at the first allele of parent 1 and copy the gene to the first position of the child. Then, we look at the allele at the same position in parent 2. We cannot copy this gene to the first position of the child because it has been occupied. We will go to the position with the same gene in the parent 1 and suppose it is at the position i. We copy the gene in parent 2 to the position i of the child. We then apply the same operation on the gene in position i of parent 2. The cycle is repeated until we arrive at a gene in parent 2 which has already been in the child. The cycle started from parent 1 is complete. The next cycle will be taken from parent 2. This crossover mechanism enables the child to efficiently inherit the characteristics from both parents. However, this approach is possible to generate invalid alleles for our data allocation problem, due to the size constraint of each memory unit. Because of the limitation of directly applying the cycle crossover method to our data allocation problem, we propose an adaptive cycle crossover strategy to guarantee valid data allocation. The critical idea of our approach is that we use a variable to keep the currently available space of each memory unit. For each genetic operation of data allocation, we will check if there is enough room for assigning the gene to the specific memory unit. If it is true, the data will be directly allocated. Otherwise, we will adaptive check the memory units of the neighboring processor cores and find a space for it. However, if all on-chip memory units, including SRAMs and MRAMs, are full, the data will be assigned to the off-chip.

```
Algorithm-2: Adaptive cycle crossover algorithm
Input: Two parent chromosomes P1 and P2.
Output: A new chromosome.
1: Assume the length of each chromosome is L.
2: while Child chromosome has empty position do
3:     for i = 1 → L do
4:         if Gene i in P1 has not been copied to the child chromosome then
5:             Keep the gene and break;
6:         end if
7:     end for
8:     if The memory unit associated with gene i is full then
9:         Adaptively search an available position from neigh- boring memory units;
10:    else
11:        Copy gene i to the same position of the child;
12:    end if
13:    Get a gene Ge at position i in P2;
14:    while Ge has already existed in the child do
15: Locate the gene Ge in P1, suppose its position is j;
16: Copy the gene Ge to the position j of the child;
17: Get a new gene Ge at position j in P2;
18: end while
19: Apply the same process on P2 to copy genes to the child chromosome;
20: end while
21: return The child chromosome;
```

The Crossover of the cycles is able to travel through both the parents. Hence the good feature of both of them can be examined carefully. But the downside of it is the relative long cost of examining each position of the parent chromosomes. Hence, we have proposed another simpler crossover operation, which is a modified version of the *Partially Mapped Crossover* (PMX). The algorithm to it is quite easily understandable.

```
Algorithm 3 Modified PMX algorithm
Input: Two parent chromosomes P1 and P2.
Output: A new chromosome C.
1: Assume the length of each chromosome is L;
2: Randomly generate a crossover point $0 \le cp \le L$;
3: for all Genes in the segment starting from the crossover point in P1 do
4:     Examine the gene at the same position of P2;
5:     if The two genes have not been copied to C then
6:         Fill the positions of the child C by swapping the two genes in P1;
7:     end if
9: end for
10: Map the remaining genes in P1 to C
11: return The child chromosome C;
```

VII. MUTATION

Once the crossover operation has been completed, the good features which are eliminated during crossover will be recovered by performing a genetic mutation. The mutation algorithm will also prevent premature convergence in a local optima. Mutation is performed by randomly flipping bits within a chromosome at a specific probability called a mutation rate, similar to crossover. Mutation rate is defined to be a tunable parameter, the value of which can be changed using the equation for PM defined as follows.

$$PM = \frac{Q_m\left(FT_{max} - FT_{bestM}\right)}{\left(FT_{max} - FT_{avg}\right)} \qquad (5)$$

where FT_{bestM} is the fitness level of the selected chromosome, and Q_m is positive constant between 0 and 1.

```
Algorithm 4 Algorithm for Genetic Mutation
Input: A Chromosome in population and mutation rate PM.
Output: A new chromosome.
1: Randomly select two genes i and j in the input Chromosome;
2: Generate a random number RanN between 0 and1;
3: if RanN ≤ PM then
4:     Form a new chromosome by swapping the memory units of gene i and gene j;
5: end if
6: return The new generated chromosome;
```

However, we must note that the probability of mutation is lesser than the probability of crossover, since for every crossover, a mutation is performed with the probability PM. Since we define the genes in this algorithm as a datum and a memory unit pair, mutation is performed by either swapping the data or the memory units of the selected gene. We will thus swap the number of memory units of two genes to achieve a mutated gene.

VIII. CONCLUSION

The whole procedure of our AGADA algorithm can be summarized. 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 CMP system. After the initialization, the fitness value of each individual will be calculated. 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:

- The number of new generations exceeds a predefined maximum number of iterations
- 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

Algorithm 5 Adaptive Genetic Algorithm for Data Allocation
(AGADA)
Input: A set of data items, a CMP system with P processor cores, and each core has a hybrid SPM. Any SPM_i has a SRAM with size of SS_i and a MRAM with size of SM_i.
Output: A data allocation.

```
1: Generate initial population;
2: $New_{PoP} \leftarrow \Phi$;
3: Determine the fitness of each individual;
4: while Termination criterion is not met do
5:     for $i = 0 \rightarrow PS$ do
6:         Randomly select two chromosomes $i$ and $j$
           from current population;
7:         Optionally apply the crossover operation
           on chromosomes $i$ and $j$ with probability
           $PC$;
8:         Optionally apply the mutation operation on
           the new chromosome with probability $PM$;
9:     end for
10:    Evaluate all individuals and perform selection;
11: end while
12: return the best allocation has obtained;
```

REFERENCES

[1] J. Montanaro, R. T. Witek, K. Anne, A. J. Black, and E. M. Cooper, “A160-MHz, 32-b, 0.5-w CMOS RISC microprocessor,” Digital Technical Journal, vol. 9, no. 1, pp. 49–62, 1997.

[2] M. Qiu, Z. Chen, and M. Liu, “Low-power low-latency data allocation for hybrid scratch-pad memory,” IEEE Embedded Systems Letters (ESL), vol. 6, no. 4, pp. 69–72, 2014.

[3] M. Qiu, Z. Chen, Z. Ming, X. Qiu, and J. Niu, “Energy-aware data allocation for mobile cloud systems,” IEEE System Journal, 2014, in press.

[4] R. Banakar, S. Steinke, B. S. Lee, M. Balakrishnan, and R. Marwedel, “Scratchpad memory: design alternative for cache on-chip memory in embedded systems,” in Proceedings of the 10 International Symposium on Hardware/Software Co-design (CODES), May 2002, pp. 73–78.

[5] P. R. Panda, N. D. Dutt, and A. Nicolau, “Efficient utilization of scratch-pad memory in embedded processor applications,” in Proceedings of the 1997 European Conference on Design and Test, Mar. 1997, pp. 7–11.

[6] C. R. Johns and D. A. Brokenshire, “Introduction to the cell broadband engine architecture,” IBM Journal of Research and Development, vol. 51, no. 5, pp. 503–520, 2007.

[7] S. Udayakumaran and R. Barua, “Compiler-decided dynamic memory allocation for scratch-pad based embedded systems,” in Proceedings of the 2003 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES), Oct. 2003, pp. 276–286.

[8] G. Sun, X. Dong, Y. Xie, J. Li, and Y. Chen, “A novel architecture of the 3D stacked MRAM l2 cache for CMPs,” in Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA), 2009, pp. 239–249.

[9] 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 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011, pp.1–6.

[10] D. Whitley, “A genetic algorithm tutorial,” Statistics and Computing, vol. 4, no. 2, pp. 65–85, 1994