Random Sampling SPM Management Algorithm For Single Processing Core

Kavita Tabbassum, Shahnawaz Farhan, Suhni Abbassi, Zulfiqar Maher, Saima Tunio

Abstract: This research is aimed at the dynamic management of SPM on a single processing core. A dynamic SPM dynamic management strategy based on random sampling is proposed. The dynamic memory access characteristics displayed during the execution of the program and are used to manage SPM and make the SPM management free from depending on Profiling information and compilers. The difference between this method and the traditional SPM management strategy is that it utilizes the hardware support provided by DataUnit, and performs complete runtime management of SPM through software and hardware coordination, which can better reflect the dynamic changes of program access during program execution. . Furthermore, this paper extends the random sampling SPM allocation algorithm to a multi-tasking environment, simulates the multi-tasking environment by modifying the small real-time operating system RTOS, and designs a multi-task test program set based on RTOS as needed. The performance of SPM is performed using a random sampling algorithm in the single task environment.

.

Index Terms: Cache , Core- working set, Multi-core processor, Memory Architecture, , Memory Management, On- chip memory , Scratchpad Memory.

1 INTRODUCTION

In the hierarchical memory architecture, the L1 on-chip memory is composed of SPM and Cache. In the on-chip memory, SPM can assist the Cache to store certain data types suitable for memory in SPM. This paper proposes a dynamic management strategy based on random sampling combining software and hardware for the single processing core (Cell) internal SPM management. It dynamically predicts the core working set by means of runtime detection program access characteristics (Core Working Set) and guide SPM space allocation. A single processing core internal DataUnit can provide the necessary address sampling hardware logic to ensure the efficiency of runtime SPM management. This method is different from the traditional SPM distribution in that it adopts the software and hardware coordination method to predict the dynamic memory access feature during the program execution to guide the SPM management, without relying on the Profiling information and the compiler for SPM management. Experiments show that this method can give full advantages of SPM in terms of power consumption and area, so as to improve the overall operating efficiency of the system. By comparing this method with a classic compiler-based SPM management strategy, it is proved that the proposed method improves the flexibility and versatility of SPM management, especially for applications. Program portability and cross-platform provide excellent support.

2. Literature Review

2.1 Proposal of the Core Working Set Concept

The concept of the Working Set was first proposed in [1] in 1968 to describe a collection of all memory addresses accessed during program execution over a certain period of time. This concept has been used until now. There are far-reaching implications for runtime system and memory management research, but there are some limitations: the working set concept treats memory addresses in all working sets as equally important address units, regardless of their respective Access frequency and access cost. However, for most computer workloads, program access is not evenly distributed throughout the working set, but tends to some more centralized address areas. This core work is closely related to the principle of program locality. The set phenomenon has been verified in various programs [2]. proposes the following concepts based on the set of important addresses present in the real load of the core working set: Definition.1. The core working set refers to a subset of the traditional working set [3], the address of which can satisfy the vast majority of the program's memory access request. Definition.2. During the running of the program, the ratio of the number of data blocks (Blocks) corresponding to 50% of memory accesses to the total number of data blocks is N1/2. Definition.3. During the running of the program, the proportion of the number of memory accesses that can be satisfied by 50% of the data blocks is W1/2. In the literature [4], the above statistics are performed for 20 typical applications in SPEC2000. The results show that:

(1) In most applications, there is obvious memory locality, 11 of which have N1/2 <3%, which means that more than half of the memory access requests occur in only 3% of the total memory addresses;

(2) Among the 12 applications, W1/2<1%, which means that 50% of the addresses can only satisfy about 1% of the access requests, and the number of accesses to each address does not exceed 1;

(3) Memory accesses are not evenly distributed throughout the main memory space, but are concentrated in a small area, which is the core working set mentioned earlier.

The core working set theory is an extension of the working set, which can better reflect the local characteristics of memory access in the actual load, and lays a theoretical foundation for the related optimization work carried out by using this feature.

2.2 Core Working Set Existence Verification

In order to have a more intuitive understanding of the concept of the core working set, the typical memory access of four embedded programs are recorded through Trace, to verify and analyze the existence and characteristics of the core working set. Four typical applications were selected from the commonly used embedded test program set MiBench [5]: basicmath, dijkstra, stringsearch and matrix, and tested in the ARM926EJ-S based system simulator, and the experimental platform related to it. During the test, all the memory access information during the running of the program is recorded and analyzed. In the SPM allocation strategy, the core idea is to prioritize the most frequently accessed data in the execution of the program to the SPM on-chip memory, thereby minimizing the excessive latency and energy consumption caused by off-chip access to the DRAM. In terms of how to define whether

data is frequently accessed, you can consider using the concept of the core working set to reasonably distinguish between cold and hot data when the program is running, in order to maximize the efficiency brought by SPM allocation.

3. SPM Allocation Algorithm Based on Random Sampling

The random sampling algorithm is a program-time adaptive SPM allocation algorithm, which can effectively adjust the data content in the SPM by effectively utilizing the dynamic changes of the memory access characteristics displayed during the running of the program. This section first proves the mathematical principle of the random sampling algorithm, and then introduces the specific implementation of the algorithm from two aspects of hardware and software.

3.1 Theoretical Derivation of SPM Random Sampling Algorithm

The random sampling method can dynamically determine the frequently accessed data area according to the program runtime information, need access to status information. In the actual memory access process, a large number of accesses are only performed for a small number of data blocks, and the memory access address "captured" by a lower probability is most likely to belong to a frequently accessed memory block. The random sampling method can dynamically predict the core working set of the program running time after a few times of address detection during the running of the program. This article uses a random sample to specify that a block of data belongs to the core working set. Dynamic prediction of core working sets using random sampling is the basis for dynamic SPM management.

3.2 Data Unit Hardware Support for SPM Random Sampling Algorithm

Each processing core inside is called a Cell. Each Cell is internally composed of three independent unit modules: InterUnit, ProcUnit, and Memory Management Unit (DataUnit), where DataUnit is responsible for the necessary hardware support for dynamic memory management. In the single processing core, L1 on-chip memory is composed of Cache/SPM, and MMU in DataUnit supports page-type virtual memory management with different granularity. For the random sampling algorithm proposed in this paper, DataUnit can also provide a special Memory Random Sampling Unit (MRSU) to support it. This software and hardware coordination design helps to compensate SPM management through pure software and the runtime overhead that comes with it. The SPM management hardware logic block diagram of the sampling random sampling algorithm is shown in Figure 3, where the MMU, MRSU and Address Comparator are all part of the DataUnit in the Cell. The memory access request issued by the ProcUnit in the single processing core is translated into the physical address by the memory management unit (MMU) address translation in the form of an object reference (Offset) and an offset (Offset) Physical Address, the physical address is directly compared with the address range of the SPM to determine whether the address is located in the on-chip SPM where SPM is mapped into memory.

In a fixed Register to facilitate address comparison, in a specific implementation may be employed SPM address comparison register (SPM Address Comparator) to store the SPM fixed address range.The result of the address comparison, there are two possibilities: if the memory access address is on-chip SPM, the read and write data directly in the SPM and returns ProcUnit processed; if the memory access address is not in the SPM, then the next Access is also located on-chip Cache to determine whether the data can be hit in the Cache,But the difference is that, when the data access is not hit in the Cache will be the failure of the physical address as MRSU enable signal to inform DataUnit open a random sampling address.

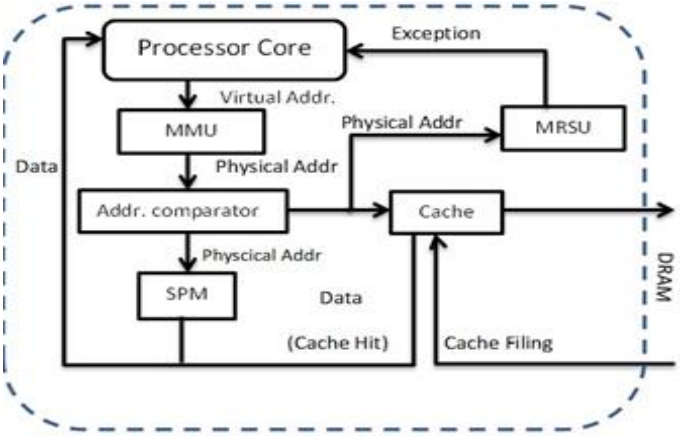

Figure 3: *Hardware Block Diagram of SPM Management Algorithm Based on Random Sampling*

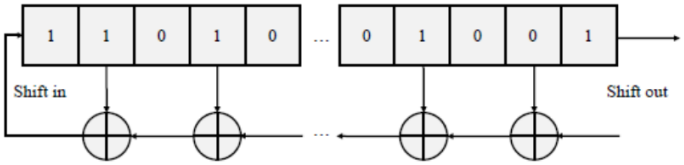

Figure .4*: Schematic Diagram of the Working Principle of the Linear Shift Register*

A MRSU internal core member is a random number generator for generating a random number which is located within a certain range in each clock cycle. Random number generated by the random number generator with a preset threshold value to determine whether the random sampling, if the comparison is required after the address sampling, by MRSU failure will be served by the physical address via the Cache address as a sample to the processing core for subsequent processing. ProcUnit physical address received at this time as the Open Signal SPM allocation, triggering a software interrupt by the software management mechanism which is responsible for the data transmitted from the L3 DRAM chip to the on-chip SPM in ProcUnit.

MRSU the most important feature is a random number generator, the hardware design can be used in a linear shift register[6] (Linear Feedback Shift Register, LFSR) random number generation, lfsr implementation principle shown in Figure.4 shown.Lfsr can generate a sequence of random numbers loop, when selecting the appropriate feedback function, loop sequence generated lfsr need to go through a long cycle to cycle once.By pre-set a value located LFSR cycle t, and T and lfsr random number generated by the comparison, if the random number generated by the lfsr access count is greater than the preset threshold T, then generate a MRSU enable letter addressed address space, it can be SPM address range is stored in a fixed Register to facilitate the address comparison, inThe result of the address

comparison, there are two possibilities: if the memory access address is on-chip SPM, the read and write data directly in the SPM and returns ProcUnit processed; if the memory access address is not in the SPM, then the next Access is also located on-chip Cache to determine whether the data can be hit in the Cache,But the difference is that, when the data access is not hit in the Cache will be the failure of the physical address as MRSU enable signal to inform DataUnit open a random sampling address. No.For example: you can use a 16-bit shift register to generate a random number from 0 to 65535, and according to the needs of the preset threshold value T is set to 328, which can produce an approximate P = 0.005 random sampling frequency.Using lfsr random number generator logic is very simple, easy to integrate in the DataUnit SPM allocation hardware support.

3.3 Software Management of SPM Random Sampling Algorithm

The software management of the random sampling SPM allocation algorithm involves two aspects: (1) data exchange between off-chip DRAM data and SPM data; (2) address redirection after data migration. The data is exchanged between the off-chip DRAM and the on-chip SPM in units of blocks, and the granularity of the blocks can be configured by software according to the size of the data blocks in the page table. The virtual memory mechanism provided by MMU in DataUnit can support multiple granularity of page mode. Through the initial configuration of page table, different size blocks can be selected as the basic unit of data exchange. There are two cases in the data exchange process: if there are free blocks in the SPM, the off-chip data is directly copied to the free blocks in the SPM; if there are no free blocks in the SPM, a replacement algorithm needs to be selected in the SPM. One piece eliminates SPM and then allocates new data blocks to SPM. In the SPM software management mechanism, the bitmap is used to record the usage of the SPM, and the bitmap is set to an array bitmap [SPM_BLK_NUM] whose size is equal to the number of SPM blocks, where bitmap[i]=1, indicating that the i-th block in the SPM is occupied; Bitmap[i]=0, indicating that the block is idle. After the data exchange between the data and the SPM in the L3 DRAM, the redirection problem of the subsequent access address is generated. By modifying the content of the corresponding item in the page table, the access to the original off-chip address can be mapped to the SPM. The MMU in the DataUnit is responsible for maintaining the mapping between virtual and real addresses. It can be represented by the following two-group: <virtual_addr, physical_addr>. In the initial state, the data in the bitmap is 0, indicating that all blocks in the SPM are free. The process of data exchange from off-chip to SPM can be implemented by Algorithm.1. Where S represents the set of SPM data blocks and si represents the ith data block in the SPM.

Algorithm.1. SPM Data Exchange Algorithm
Input: off-chip data block address mem_addr
Output: block first address spm_addr written on SPM
(1) If the SPM has a free block, traverse the SPM bitmap to find the first free area si, and record the block first address spm_addr;
(2) If there is no free block in the SPM, a random replacement algorithm is used to find a si from the SPM, the block data is exchanged to the off-chip idle location and the first address spm_addr of the data block in the SPM is recorded;
(3) Transfer the off-chip data to SPM through the Load/Store instruction, and modify the corresponding entry of the page table to <mem_addr, spm_addr>;
(4) Return to spm_addr.

The dynamic SPM management proposed in this paper needs to use the MMU to perform virtual and real address translation. By re-establishing the mapping relationship between the virtual address and the physical address, subsequent access to the "moved" data area can be performed on the SPM, thereby reducing the number of off-chip accesses. When all the locations in the SPM are occupied, it is necessary to replace a piece of SPM data into the L3 DRAM, which is basically consistent with the principle of the Cache replacement algorithm. At this time, an equal size space can be allocated in the off-chip memory space as needed to store the data replaced from the SPM, thereby "filling" the data to be allocated in the SPM to a suitable position in the DRAM. How to select the data block that is replaced by SPM is the key. You can use the LRU and Round-Robin methods in the Cache replacement strategy to select the appropriate data block for replacement. For the sake of simplicity in the subsequent experiments in this paper, basic random selection is adopted. The replacement of SPM data blocks.

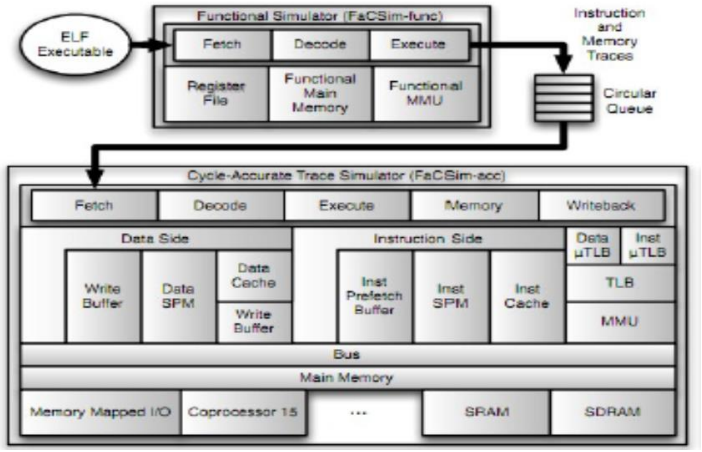

***Figure.5**: ARM performance simulator FaCSim internal structure block diagram*

3.4 Performance Analysis

This section uses performance simulation to evaluate the performance of the random sampling SPM allocation algorithm. In the aspect of performance evaluation, the execution time and energy consumption of SPM, Cache and SPM+Cache using SPM random sampling algorithm are compared, and the validity and rationality of random sampling algorithm for SPM allocation are verified. Because lacks prototype system to verify this part of SPM management strategy, this paper uses software simulation method to analyze the performance of random sampling algorithm on ARM926EJ-S based platform. In this paper, by modifying the simulator FaCSim, hardware logic similar to DataUnit for SPM dynamic memory management is added to ARM to verify the rationality of this software and hardware collaborative SPM management strategy in TriBA.

3.4.1 Experimental Platform

3.4.1.1 Introduction to Simulator FaCSim

The performance simulation was performed using the ARM9 simulator FaCSim [7]. FaCSim is capable of performing Cycle Accurate simulations on the ARM926EJ-S[8] processor. The simulator uses Written in C++ to provide accurate and efficient simulation and simulation of the ARM architecture. Such as as shown in Figure.5, it is divided into a functional front end (Front-End) and a timing accurate back end (Back-End). The front-end mainly simulates the main functions of the pipeline in the embedded processor, and performs function simulation on the execution of the instruction, and passes the number of time beats (including the execution of instructions, memory access, data transmission, etc.) consumed by each execution stage to the count. The back end and the back end calculate the total number of time beats required during the execution of the entire instruction to achieve performance simulation. The processor emulated by FaCSim includes a complete pipeline structure and a detailed simulation of the memory hierarchy, which provides simulation of SPM (Tightly Couple Memory, TCM) and Cache, especially at the on-chip memory level. Since the SPM allocation algorithm used in this paper requires runtime completion, MMU is required to support address redirection. See Table .1 for the system configuration of the simulation experiment.

Table.*1 Simulation Environment System Parameters*

Memory Hierarchy	FaCSim
Frequency	140 MHZ
On-chip memory	256 MB
L1 data/instruction Cache	32K, 32B line 4-way
L1 data/instruction Cache	write back
Cache	32K, 32B line 4-way
SPM	32KB
SPM	Random
TLB	64 entries, 2-way

The simulator uses data/instruction Cache and data/instruction SPM as the on-chip memory hierarchy. This section only targets he management of the data SPM is experimentally verified, and the related management of the command SPM is similar. This paper implements the random sampling algorithm by modifying the hardware structure and software strategy of the SPM module. In particular, in terms of comparing experimental setups, consider using three different on-chip memory configurations: SPM, Cache, and Cache+SPM. In the experiment, the performance of 32KB SPM and 32KB Cache is used for performance comparison, and in the case of mixed mode, 16KB SPM+4KB Cache is adopted to meet the requirement that Cache+SPM is smaller than or equal to the equivalent size Cache.

Table .*2: Introduction to the Experimental Test Procedure*

Benchmarks	Category	Execution time(Cycles: x 106)
basicmath	MiBench	128.2
bitcount	MiBench	976.2
blowfish	MiBench	1.4
fft	OOPACK	25.0
dijkstra	MiBench	100.5
Md5	OOPACK	1.8
matrix	OOPACK	70.9

3.4.1.2 Introduction to the Experimental Test Set

This section selects some of the programs currently used in the widely used embedded program set MiBench and OOPACK for performance analysis. MiBench is a free embedded benchmark program launched by the School of Electrical Engineering and Science at the University of Michigan. It was widely used in various fields of embedded research since its introduction in 2001. OOPACK is a collection of embedded programs developed to compare performance differences between object-oriented and process-oriented programs. It mainly includes the core code of four different calculation types of Max, Matrix, Iterator and Complex. In this section, the following applications are selected as the experimental test cases in the above two test program sets. The specific information of each program is shown in Table.2.

3.4.1.3 Energy Consumption Model

The experimental part mainly compares and analyzes the power consumption of on-chip memory of Cache, SPM and SPM + Cache. Firstly, it briefly introduces its energy consumption model. In the hybrid on-chip memory hierarchy consisting of SPM + Cache, E_{mem} _ subsystem represents the energy consumed by the memory system, which includes the sum of the energy consumed by the three parts of Cache, SPM and DRAM, calculated by the formula (2):

$E_{total} = E_{Scratchpad} + E_{cache} + E_{DRAM}$ ********* (1)

$E_{Scratchpad} = e_{scratch} * (read + write)$ ******* (2)

$E_{cache} = e_{cache} * (hit + miss * linesize)$****** (3)

$E_{DRAM} = e_{DRAMread} * read + e_{DRAMwrite} * write + T_{total} * P_{standby}$ **** (4)

The power consumption of Cache and SPM can be obtained by formula (4) and formula (5), where hit and miss represent the number of access cache failures and failures during program simulation, and ecache represents the energy consumed by each access to Cache. The number of bytes contained in the Cache line. The DRAM power consumption is calculated by the formula (3). The read and write times of the DRAM are expressed as read and write respectively. The energy consumed for each read and write is: $read_{DRAM}$ and $write_{DRAM}$, and *$totalstandby_{TP}$ indicates the static power consumption of the DRAM.

3.4.2 Experimental Setup and Results Analysis

This section first determines the reasonable sampling frequency as the basis of the subsequent experiments; and compares the SPM and Cache managed by the random sampling algorithm and the SPM dynamic management strategy proposed in the literature [9], which proves the rationality of the random sampling algorithm. Advantages; finally analyzed the impact of different SPM size on system execution time.

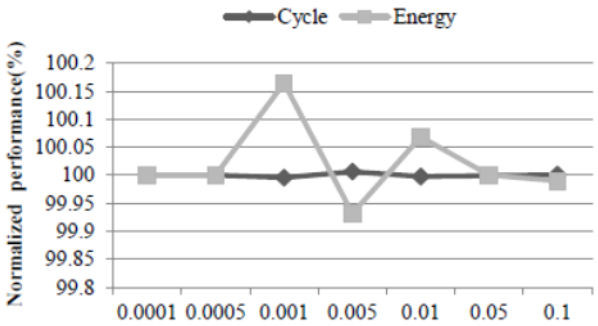

Figure.6*: Performance comparison at different sampling frequencies*

3.4.2.1 Effect of Sampling Frequency on System Performance

This section compares the effects of random sampling algorithms on program performance at different sampling frequencies. Figure.6 shows the variation in execution time and power consumption of the test program dijkstra at seven sampling frequencies. The abscissa indicates different sampling frequencies, and the ordinate indicates normalized execution time and energy consumption. The results show that the execution time varies slightly with different probability of sampling; the energy consumption is the lowest at P=0.005, but the sampling frequency has a limited impact on performance. Unless otherwise stated, the subsequent experiments used P=0.005 as the sampling frequency.

3.4.2.2 Performance Comparison with Cache

Execution time and energy consumption are important indicators to measure the overall performance of the processor. The performance evaluation in this section mainly evaluates and analyzes the random sampling SPM allocation algorithm from two aspects of energy consumption and execution time. The evaluation uses a comprehensive indicator reflecting the execution time and energy—energy delay product EDP [10] (Energy Delay Product, EDP = Energy ×Delay) as the unified dimension. This experiment compares the EDP relationship between 32KB data SPM and equivalent size data cache. As shown in Figure.7, the SPM with a random sampling allocation strategy has an EDP that is about 11% lower than the traditional Cache. Experiments show that the number of off-chip accesses is significantly reduced after using SPM, and most of the memory that was originally hit in the Cache is converted to SPM with lower unit access energy consumption, thus reducing the overall energy consumption of the memory system. In general, the random sampling algorithm reduces system power consumption while keeping execution time relatively stable[11].

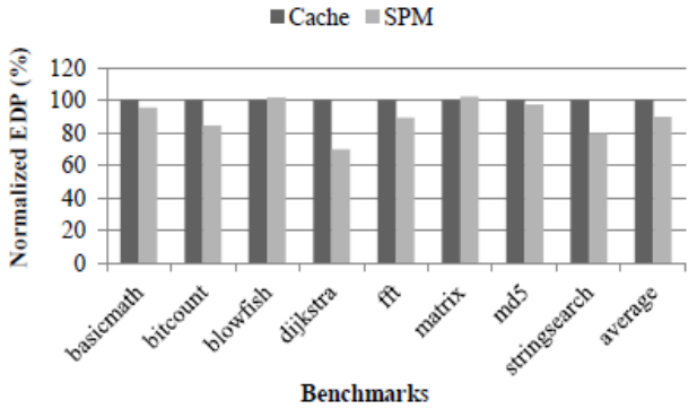

***Figure.7**: Comparison of Cache and SPM energy delay product EDP*

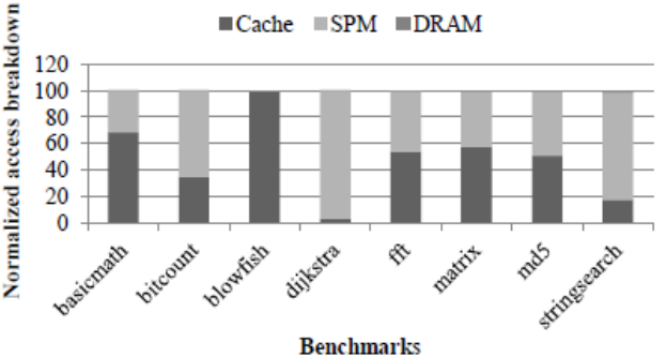

***Figure.8**: Memory model distribution of access levels at each level*

Figure.8 shows the distribution of memory access during the running of all test programs. As shown in the figure, the programs with a higher SPM access rate are dijkstra and stringsearch, while the programs with a higher Cache access rate have blowfish. The SPM and Cache access ratios of other programs are roughly equivalent. The reason for this result is related to the local characteristics of the program. The dijkstra and stringsearch access characteristics show a linear distribution trend, and the locality is worse than other programs. For this kind of program with poor locality, the random sampling algorithm will appear more frequent swap-in and swap-out operations. The failure rate in SPM is higher, and a large number of memory accesses are still in the Cache. This experiment demonstrates that SPM with a random sampling allocation strategy has a better effect on Irregular data access. Due to the existence of on-chip cache SPM and Cache, an average of more than 99% of memory accesses can be hit in the on-chip memory hierarchy, greatly reducing the overhead of off-chip access. Figure.9 shows the ratio of the energy consumed by the three memory tiers of SPM, Cache, and DRAM in the test vector to the total energy. Both Sijk energy consumption of dijkstra and stringsearch is higher than other memory levels. As can be seen from Figure.8, the main reason is that SPM occupies the vast majority of the memory access; while in other programs, Cache energy consumption still dominates. Table.3 shows the comparison of the on-chip area of SPM and Cache under the same size conditions. As shown in the table, the SPM area of the same size is about 23% to 55% of the Cache. The following data demonstrates that using SPM as an alternative to Cache has a smaller area overhead.

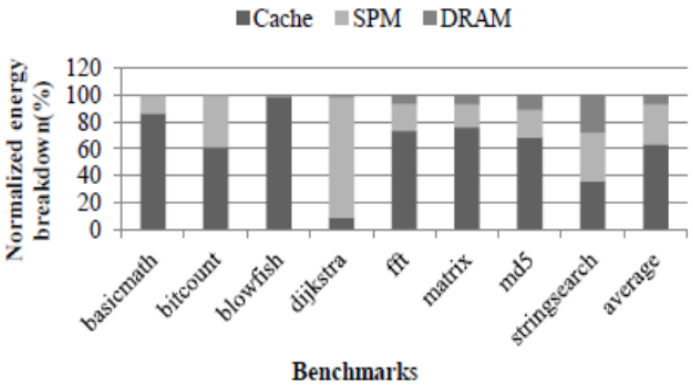

***Figure .9**: Schematic diagram of the relationship between energy consumption of Cache, SPM and DRAM*

CONCLUSION

This paper is aimed at the dynamic management of SPM on a single processing core. A dynamic SPM dynamic management strategy based on random sampling is proposed for single core. The dynamic memory access characteristics displayed during the running of the program are used to manage SPM and get rid of SPM management. Depend on Profiling information and compilers. The difference between this method and the traditional SPM management strategy is that it utilizes the hardware support provided by DataUnit, and performs complete runtime management of SPM through software and hardware coordination, which can better reflect

the dynamic changes of program access during program running. The performance of SPM is performed using a random sampling algorithm in the task environment. Experiments show that the random sampling algorithm not only has good support for single-task environment, but also applies to multi-tasking environment. Through the analysis of the experimental results, it can be concluded that the SPM and Cache hybrid L1 memory can achieve a compromise between performance and power consumption in the case of single processing core. By rationally utilizing the hardware support provided, software and hardware collaboration can be constructed. The runtime efficient SPM management algorithm also provides a solution for dynamically managing SPM on a single processing core.

ACKNOWLEDGMENT

The authors wish to thank IICT, Mehran University of Engineering & Technology, Jamshoro, Pakistan and ITC, Sindh Agricultural University Tandojam, Pakistan for their continuous support.

REFERENCES

[1] Grosser T., Groesslinger A. and Lengauer C., (2013) “Polly performing polyhedral optimizations on a low-level intermediate representation”, Parallel Processing Letters, 22(04):1250010.

[2] Ji, W., Deng, N., Shi, F., Zuo, Q., & Li, J. (2011), Dynamic and adaptive spm management for a multitask environment. J. Syst. Archit. 57, 181–192.

[3] Kahle J.A., Day M.N., Hofstee H.P., Johns C.R., Maeurer T.R. and Shippy D., (2005), Introduction to the cell multiprocessor, IBM J. Res. Dev. 49, 589–604.

[4] Kannan A., Shrivastava A., Pabalkar A. and Lee J.E., (2019), “A software solution for dynamic stack management on scratch pad memory”, Proceedings of the 2009 Asia and South Paci_c Design Automation Conference, pp. 612-617. IEEE Press.

[5] Khader M., Ahsan K. and Tarek T., (2014), On-Chip Power Minimization Using Serialization-Widening with Frequent Value Encoding,‖ VLSI Design, vol. 2014, Article ID 801241, 14 pages.

[6] Khahro K. T., Talpur S., and Khahro S. F., (2019), "State of Art Innovative Technique for Management of Scratchpad Memory (Scratch), "Microprocessors and Microsystems” 70, pp.31-37..

[7] Kunzman D. M. and Kalé L. V., (2011), Programming Heterogeneous Clusters with Accelerators Using Object-Based Programming,‖ Scientific Programming, vol. 19, no. 1, pp. 47-62.

[8] Lee J., Kim H. and Vuduc R., (2012), “When prefetching works, when it doesn't, and why”, ACM Trans. Archit. Code Optim., 9(1), pp. 2:1-2:29.

[9] Lee J., Kim J., Jang C., Kim S., Egger B., Kim K., and Han S., (2018), Facsim: a fast and cycle-accurate architecture simulator for embedded systems,‖ in LCTES ’08: Proceedings of the 2008 ACM SIGPLAN-SIGBED conference on Languages, compilers, and tools for embedded systems. New York, NY, USA: ACM, pp. 89–100.

[10] Lee J., Lee J., Seo S., Kim J., Kim S. and Sura Z., (2015), COMIC++: A software SVM system for heterogeneous multicore accelerator clusters. In: 2010 IEEE 20th International Symposium on High Performance Computer Architecture (HPCA 2020), pp. 1–12. IEEE, Los Alamitos.