A METHODOLOGY FOR THE OPTIMIZATION OF MULTI-PROGRAM SHARED SCRATCHPAD MEMORY

J. F. Yang, H. Jiang
School of Electronic Information
Wuhan University
Wuhan, Hubei Province, 430072, China
Emails: yjf@whu.edu.cn, jh@whu.edu.cn

W. Hu
College of Computer Science and Technology
Wuhan University of Science and Technology
Emails: huwei@wust.edu.cn

Submitted: December 19, 2010 Accepted: February 11, 2011 Published: March 1, 2011

Abstract- Processors used in wireless and ad hoc networks bring more strict requirements on performance and power consumption. The hardware and software need to coordinate more efficiently to meet such requirements. With the rapid development of semi-conductor technology, more memory can be integrated into the processor. ScratchPad Memory (SPM) is a kind of the on-chip memory, is a SRAM based memory with fast response, small on-chip area and low power consumption. It is still a

big challenge on how to take advantages of SPM because SPM must be explicitly used by software to achieve high performance. This paper proposes a novel methodology to share SPM during multiprograms. The applications are analyzed and memory objects are generated. During the executions, multi-programs can share SPM through these memory objects. The experimental results show that our approach can reduce both the execution time and the energy consumption effectively.

Index terms*:* methodology, optimization, scratchpad memory, high performance.

I. INTRODUCTION

With the rapid development of semi-conductor, Morse's Law will still be valid within the next decade [1]. When the performance is enhanced, the price and chip area are coming down. Processors will have stronger performance, but it also requires less power consumption especially for high performance embedded systems [2]. The advance in VLSI can help us to achieve such goals: the integration density of VLSI is still in substantial increase [3]. More components with different functionalities can be integrated into the processors, which can improve the efficiency of the on-chip area for better performance of the whole system.

"Memory Wall" is always the bottleneck between the memory and processors [4]. The faster increase of processor speed makes the gap bigger between memory and processors. Such speed gap has been a major problem to further enhance the performance. On-chip memory can solve this problem partially. Memory integrated on chip will consume less energy and have better performance [5]. The speed of SRAM is 10-100 time faster than DRAM while its price is also 20 time high than DRAM [6]. So SRAM is used as on-chip memory.

ScratchPad Memory (SPM) is a type of on-chip SRAM. It is different from cache, which is also on-chip SRAM. Cache is controlled by hardware and apparent to programmers. But SPM is controlled by software and should be explicitly used [7]. On-chip area occupied by SPM is smaller compared to cache and then more memory can be integrated. At the same time, SPM has lower power consumption than cache with simpler design [8]. So SPM can be used to improve the performance of the system.

Many approaches have been proposed to take advantages of SPM. Programs must be analyzed by compilers. The slices or objects including code and data selected from programs are titled as

memory objects. These memory objects can be allocated to SPM. Some proposed approaches proposed that memory objects could be assigned to SPM through compilers [9-12]. The allocation of SPM was fixed and can not be changed during the execution and the assigned SPM could not be used by other memory objects. Some approaches were proposed to allocate SPM dynamically [13-16]. SPM spaces can be shared by different memory objects. There are also some approaches focused on array and loop [17-20]. Because SPM is now used in embedded systems mainly, the existing research also focuses on embedded systems mainly. The experimental results from these approaches show that the utilization of SPM will improve the performance.

In this paper, we propose an optimization methodology for multi-program shared SPM. Our methodology will analyze the programs and share SPM among different programs. It can effectively reduce the energy consumption and improve the performance. At the same time, our methodology can also be used in not only embedded nodes for wireless and ad hoc networks but also the other types of embedded systems even desktop computers and the servers which have SPM as on-chip memory.

The paper is organized as follows. In Section II, we describe the related work. The architecture model is described in Section III. In Section IV, we depict our optimization methodology. In Section V, we discuss our experimental results. And at last, we draw our conclusions and give the future work in Section VI.

II. RELATED WORK

Because SPM should be explicitly used, the main idea of SPM based optimization is to analyze the programs and control the mapping of code or data of these programs to SPM. The programs will be divided into different memory objects. Some memory objects can be allocated to SPM space if they are hotspot of the programs. The programs must be modified during the optimization by the compilers. The main challenge is how to deal with mapping on memory objects to SPM. Code is less modified during the execution and has good temporal and spatial locality. The optimization of data will be more complex for more types and the different access patterns.

SPM based optimization was first proposed by [7] and they described the detail in article by [10]. Their approach focused on the constants, global variables and arrays. The allocation of constants and global variables relied on their access frequency analyzed by compilers. And the allocation

of arrays depended on the sizes, life cycles and access conflicts etc. The works of [9, 11, 12] were similar to Panda's approach. Such approaches will allocate a memory objects to fixed SPM address and the address will never be changed during the life cycle of the programs.

SPM can also be divided into different banks for optimization [21-25]. Their main objective is energy-saving as well as performance improvement. The different banks of SPM have different characteristics in power consumption. Memory objects will be allocated to corresponding banks according to their features through compilers. The banks with higher power consumption will be used fewer in order to save energy.

The works of [13-16] are the typical approaches as overlapped SPM allocation. [14] proposed the approach that the elements of an array could be divided into different sub-arrays according to their life cycles. And then these new arrays were renamed and allocated to the same SPM space if they had no conflicts in life cycles. Thus different memory objects could share the same SPM space. [26] extended the granularity to pages. Programs were divided into pages, which had the same size with the pages managed by MMU. A special management unit was designed for the cooperation with MMU. [27] allocated the code of a program to SPM and provided a LC copy instruction to support the dynamic transmission of the code at run-time. The works of [28-30] allocated the most frequent used slice to SPM through inserting special instructions in selected locations of a program.

There are many arrays and loops in many programs. The performance can be improved by SPM based optimization for these arrays and loops. But not all the arrays and loops can be allocated to SPM. The arrays and loops will be analyzed. Arrays are divided into small arrays and/or the loops are transformed [17-20]. And then the selected memory objects can be allocated to SPM and share SPM space during execution. [31] analyzed the arrays of multimedia programs and adopted SPM as the shared hierarchy buffer for arrays. It is the application-specific optimization.

As the analysis in work by [32], SPM based optimization can improve the performance of the systems. In this paper, we propose our methodology for SPM based optimization for processors used in wireless and ad hoc networks as the optimization for a single node. Different from the existing research, our methodology provides the architecture of the optimization to share SPM among multi-programs. And it can be used for embedded systems as well as desktop computers and some other computing systems.

III. ARCHITECTURE MODEL

The processor model used is shown in Figure 1. There are a processor core, cache and SPM on a single chip. The on-chip components connect to off-chip main memory by bus. Cache and SPM will fetch data from main memory. Cache and SPM has the same response time, which is several cycles and both of them are faster than off-chip memory, which is about 10-29 cycles [7].

Data can not in SPM and cache simultaneously in order to ensure the consistency of data in on-chip memory. It means data in SPM can not be sent to cache and vice versa. When there are requests for data, the requested data will be sent to processor core from SPM or from cache if there is cache hit. If both SPM and cache are missed, the requests will obtain the responses from off-chip memory.

SPM has a continuous section of whole address space. Though SPM can be used for optimization, the size of onchip SRAM is very small compared to off-chip DRAM. Thus SPM space only occupies a very small section of the whole address space. It is also the reason why SPM based optimization must resort to compiler analysis for accurate allocation.

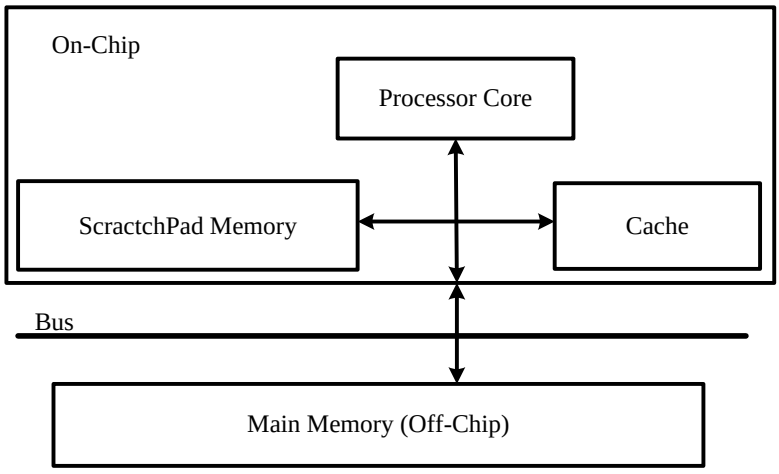


Figure 1. Processor model

IV. OPTIMIZATION METHODOLOGIES

a. Methodology overview

When there are more than one programs running in parallel, the memory objects will belong to different programs. These memory objects should share SPM for higher performance. Each

program can be divided into two parts: one part consists of the memory objects allocated to SPM and the other is the code and data which reside in main memory. If the size of SPM is big enough, each program can have its own SPM space. Otherwise, different programs must share the same SPM space.

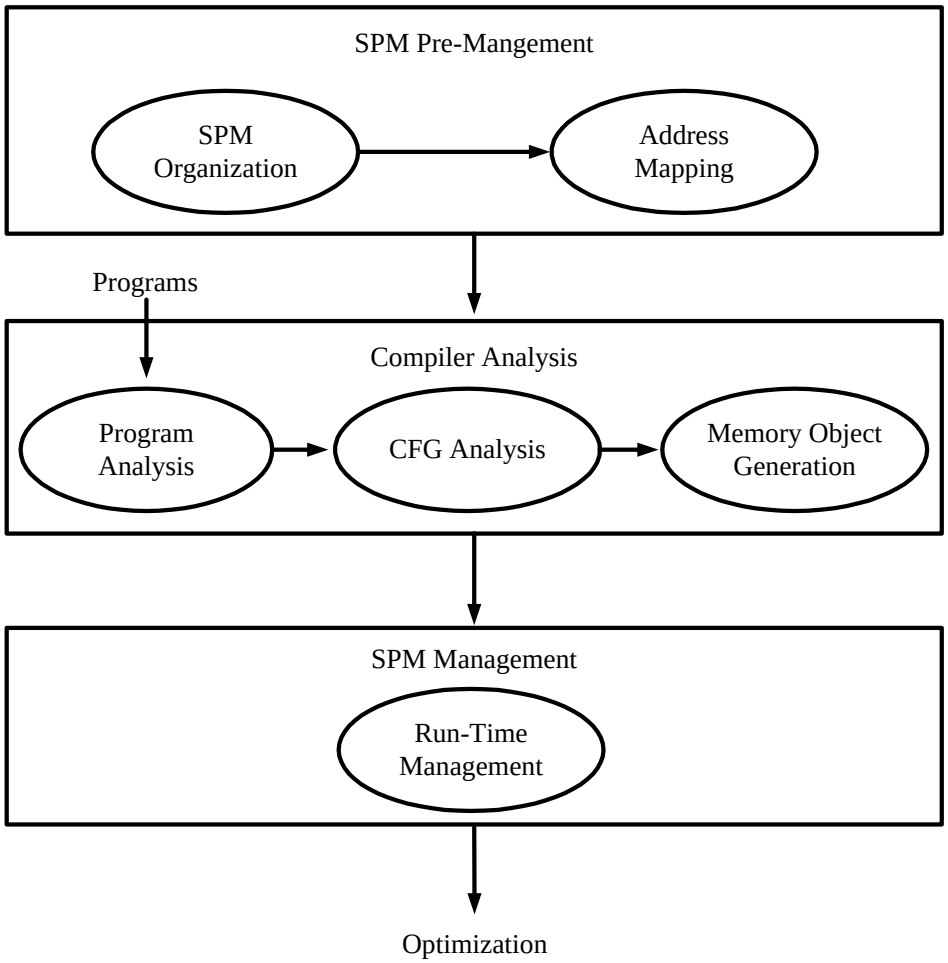


Figure 2. The optimization methodology

Our methodology has three basic phases for SPM based optimization as shown in Figure 2. The three phases of the basic methodology are SPM premanagement, compiler analysis and SPM management respectively.

SPM will be organized first for the optimization. After SPM pre-management phase including SPM organization and address mapping, SPM space will be related to physical and virtual address space respectively. Then SPM can be used for memory objects. Compiler analysis phase

has three steps including program analysis, CFG (Control Flow Graph) analysis and memory object generation. Programs should be analyzed and the memory objects will be generated as the potential allocation objects for SPM. SPM management phase has only single step: run-time management. This phase is responsible for the management of memory objects when the programs are running. In addition, we introduce a special management unit titled as SPMG for the effective management of SPM in our methodology.

b. SPM pre-management and address mapping

SPM is divided into different pages and the page size is same to the page size of main memory. All the pages are managed by SPMG at run-time. For all the pages of SPM space, there is a table titled SPT (ScratchPad Memory Page Table). There are two basic statuses of the pages: free pages and busy pages. Each item of SPT will be related to one page. And each item has six fields: page number, next page, page property, process ID, memory object and the preserved field. Page number is the number of this page in all the pages. Next page points to the next page in SPM space. Page property specifies the status of the page. Process ID is the ID of the process which is the owner of this page. Memory object field stores the memory objects allocated to SPM. And preserved field is reserved for properties defined by the users.

There are more than one memory objects in a single page possibly. And the memory objects in the same page belong to the same process for a SPM page will be allocated to a process only. Each memory object in a SPM page are linked and has five fields including memory object ID, which is used to mark the memory object; pointer of next memory object address, which is used to link to the next memory object; size, which is used to record the size of the memory object; property, which is used to store the other properties of the memory object; and at last is the memory object itself.

After SPM is used in the system, a part of the memory space will be assigned to SPM space. When the memory objects are allocated to SPM, their addresses are still in the unified address space. The physical addresses of these memory objects belong to the SPM space. The translation from linear addresses to physical addresses will be done by SPMG including the modification of the page tables of MMU.

c. Compiler analysis

Memory objects are produced from the programs and will be scheduling to the SPM space during the execution. Programs are analyzed to obtain the potential memory objects for the optimization. This work is done by compilers. The analysis flow is shown in Figure 3.

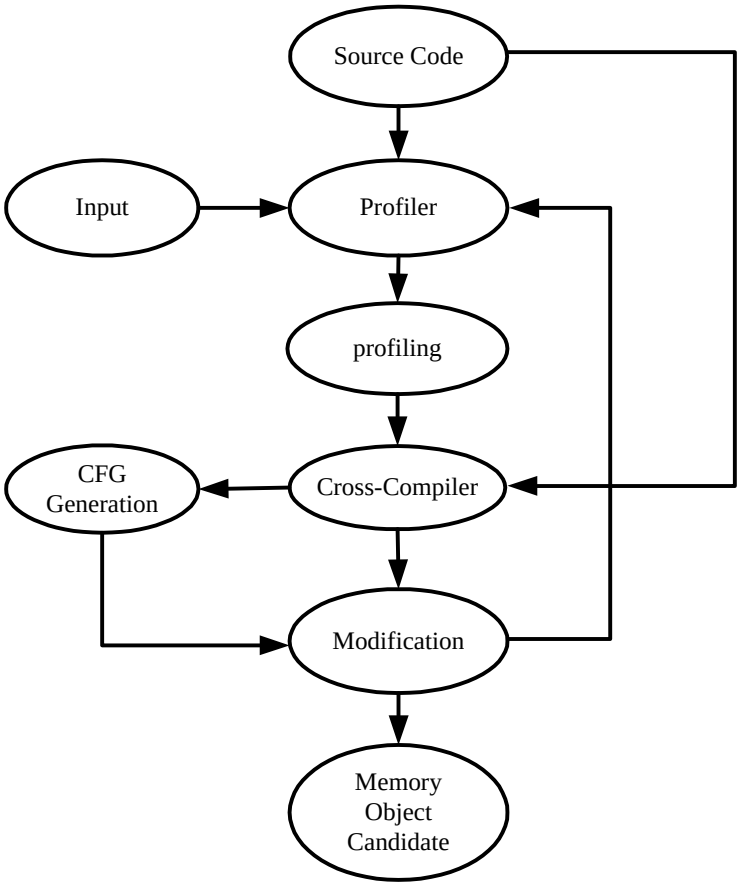


Figure 3. Program analysis

In this flow, source code of the program is the analysis object. Profiler will generate the profiling information of the programs. And then the cross-compiler will generate the CFG (Control Flow Graph) of the program according to the profiling information and modify the source for the optimization. And at last the memory objects will be found out. This flow will be cyclic to analyze the program as accurately as possible. Profiler is very important in the analysis. The program will run many times to obtain enough information according to different input data. After enough executions, the detail information of the program will be obtained. The information collected includes the characteristics, the access frequency and life cycle of the variables, stack/heap, data and code.

However, not all the memory objects can be placed to SPM for the limited space. The generated memory objects are taken as the candidates for optimization. At the same time, the compiler will insert special instructions into the source code which are used as the run-time ones for the memory objects scheduling. According to the size of SPM on the chip, some memory objects can be selected and allocated to SPM. But if the SPM space is not enough for all the memory objects, some of them have to be allocated to main memory as the common situations.

d. Memory object generation

After CFG is generated, the memory objects will be selected according to it. For each CFG, the following rules are used to select the memory objects:

(1) The size of a single memory object will never be bigger than the physical capacity of a SPM page. If the size is too big, the management will be more complex. The memory objects have to be swapped between the SPM space and main memory. This is time-consuming work. So we add this rule for simplicity.

(2) The most frequent used data should be selected as the memory object candidates. This information can be obtained from the profiler. When such memory objects are placed into SPM, the access time to main memory can be saved, which is more longer than SPM access time. And the program will be faster than before.

(3) The overhead should be small. Some instructions will be inserted into the program in order to allocate memory objects to SPM. These instructions will also execute during the execution of the programs. They will consume the CPU time, which is the overhead of the extra instructions. The overhead introduced by these instructions should be as small as possible.

All the information of the memory object candidates is stored and managed by SPMG. SPMG is responsible for the management of SPM space and memory objects. When the capacity of available SPM increases, SPMG can select more memory objects to SPM. Or some objects will be swapped from SPM to main memory according to the run-time environment.

e. Run-Time Management

When programs are running, the best case is that all of the memory objects are allocated to SPM for higher performance. When all memory objects are placed on chip, the memory wall can be overcome. However, not all the memory objects can use SPM as on-chip memory for the limit of

the size of SPM. SPMG will collect and manage the run-time information and then different programs can share SPM. SPMG will communicate with scheduling module of the operating system and obtain the run-time information of different processes. SPMG can also obtain the information of memory objects according to the prior phases of our methodology. And then SPMG will decide which memory objects can be allocated to SPM.

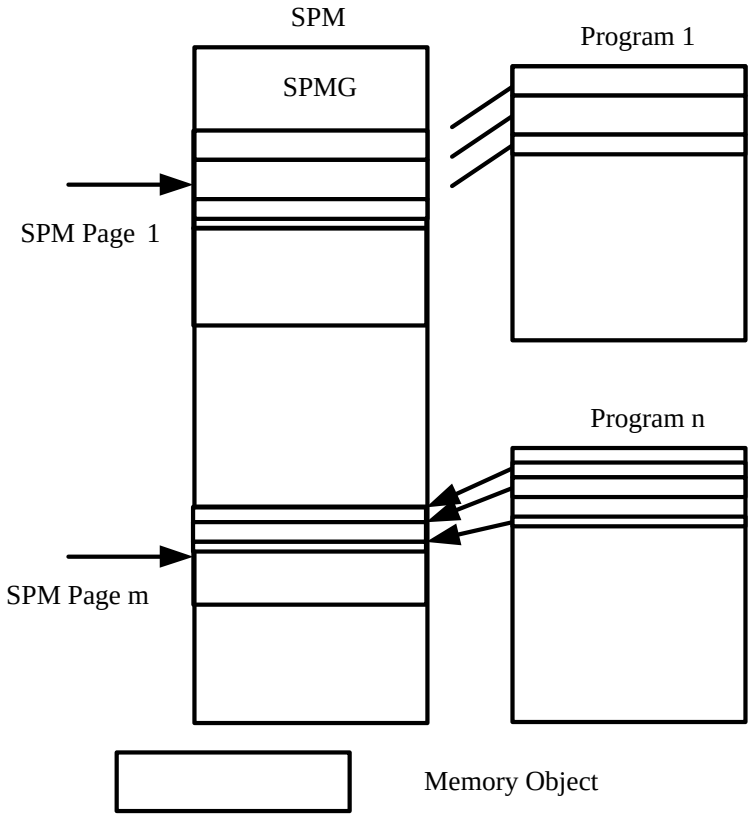


Figure 4. Run-time management

SPMG will assign the first free SPM page to the memory objects. If the process has been assigned a page, SPMG will check whether the free space of this page can hold the memory objects. If the existing page has enough space, the memory objects will be allocated to this page. Otherwise, a new free page will be found and allocated to this process. While, if a process exit, all the pages it owns will be reclaimed by SPMG.

As Figure 4 shows, SPMG will allocate SPM space to the programs. It resides in SPM space from the beginning. During the execution of the programs, the inserted instructions will request the SPM space for the programs. SPMG will check the record of the SPM for idle space for the

allocations. When there are idle SPM pages or the page belong to a program has idle space, SPMG will allocate the space to the new memory objects. Because SPM space is managed by SPMG, it will share the SPM to different programs by allocating different space of SPM to the memory objects from different programs.

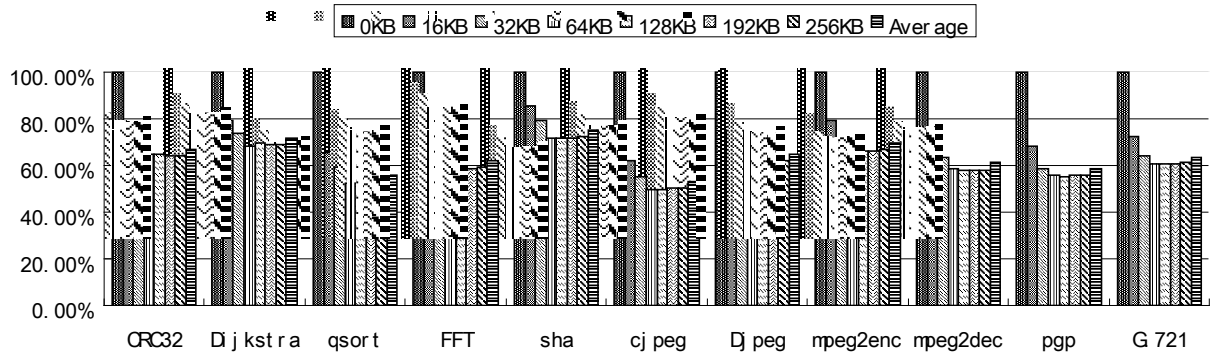
V. EXPERIMENTAL RESULTS

The hardware platform used in our experiments is the PXA 272, which is based on embedded processor and with 256 KB SPM [33]. SPM of PXA 272 has four SRAM banks and each bank can be set as standby mode or active mode. We adopt modified embedded Linux [34]. And the eleven benchmarks are selected from two benchmark sets MediaBench [35] and MIBench [36] respectively as shown in Table 1.

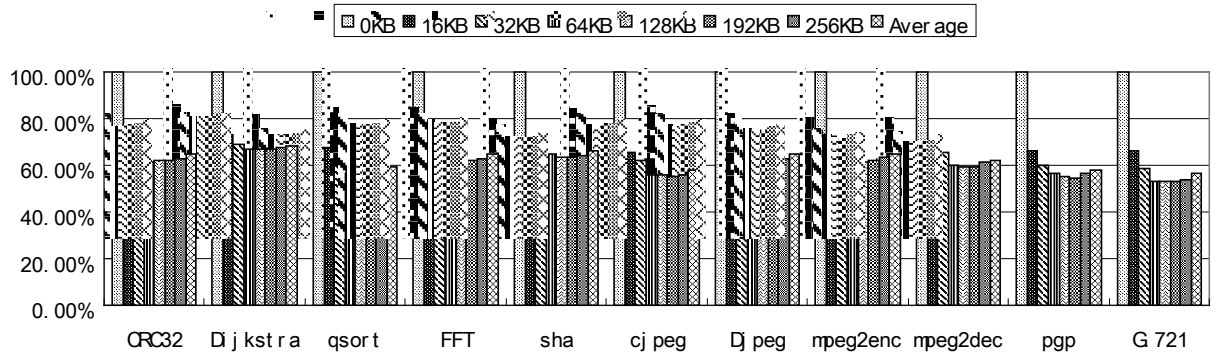
Table 1. Benchmarks

Name	Source	Description
CRC32	MIBench	32-bit standard CRC verification
Dijkstra	MIBench	Shortest path algorithm
Qsort	MIBench	Quick sort algorithm
FFT	MIBench	FFT
Sha	MIBench	Security encryption algorithm
Cjpeg	MediaBench	JPEG compress algorithm
Djpeg	MediaBench	JPEG decompress algorithm
mpeg2enc	MediaBench	MPEG compress algorithm
mpeg2dec	MediaBench	MPEG decompress algorithm
Pgp	MediaBench	PGP encryption algorithm
G.721	MediaBench	G.721 audio compress algorithm
CRC32	MIBench	32-bit standard CRC verification

We test the eleven benchmarks on the hardware platform. Figure 5 to Figure 8 show the experimental results under different configurations. The size of SPM is 0KB, 16KB, 32KB, 64KB, 128KB, 192KB and 256KB. And the parallel programs in the system are one, two, four and eight. Figure 4 shows that when there is only a single benchmark, SPM can improve the performance both in execution time and power consumption. Because there is only a single program, SPM space is relatively enough for this program, so when the size of SPM increases, all the memory objects candidates can be placed into SPM space. Thus when this situation occurs, it means that the increase of the size of SPM can not improve the performance of the program further. When all the memory objects are allocated to SPM, the improvement will be stable. It also means that a single program will result in a waste of SPM if there is sufficiently enough SPM. It is same to power consumption experiments.

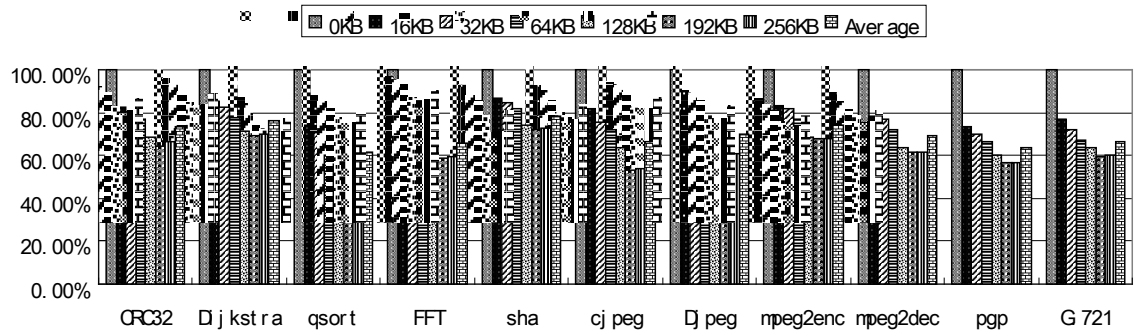


(a) Execution Time Optimization of A Single Program

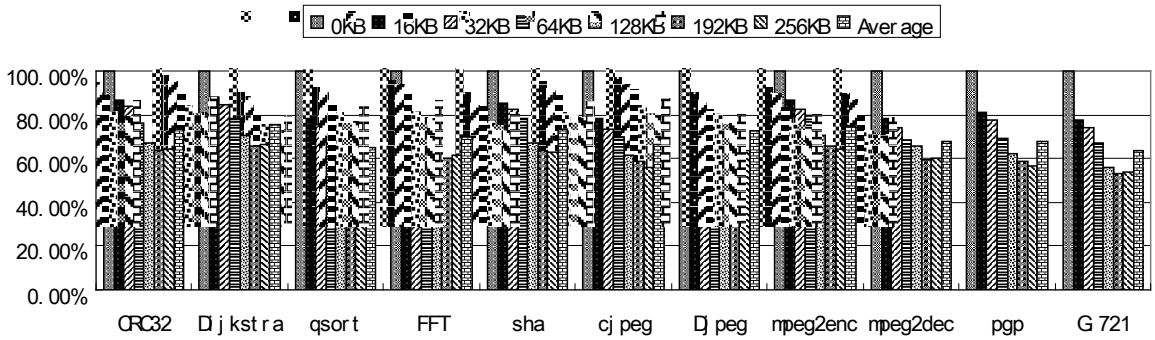


(b) Power Consumption of A Single Program

Figure 5. Optimization of a single program

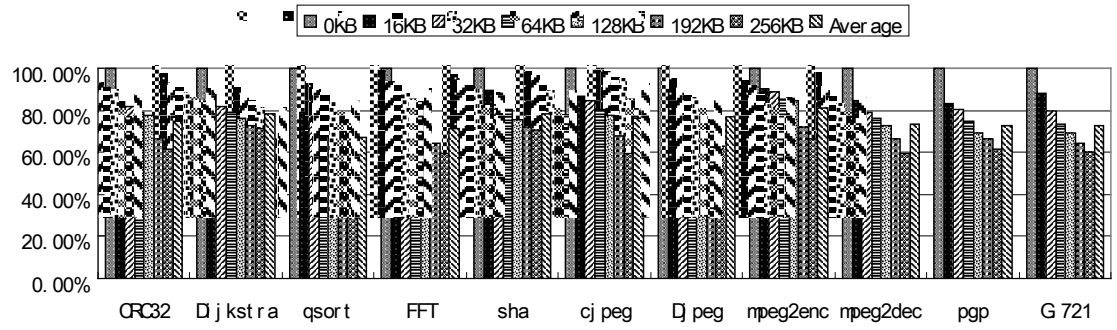


(a) Execution Time Optimization of two Parallel Program

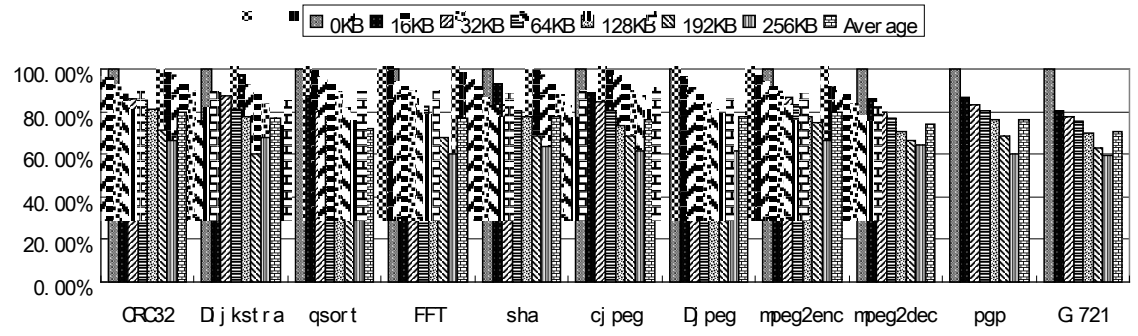


(b) Power Consumption of two Parallel Program

Figure 6. Optimization of two Parallel program

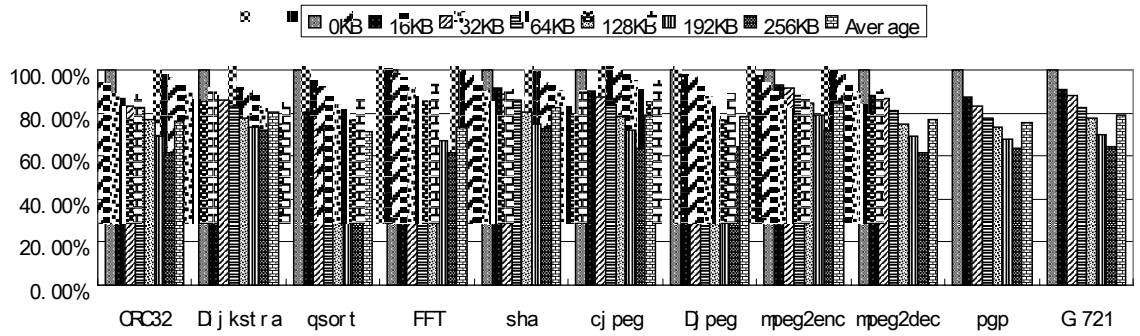


(a) Execution Time Optimization of four Parallel Program

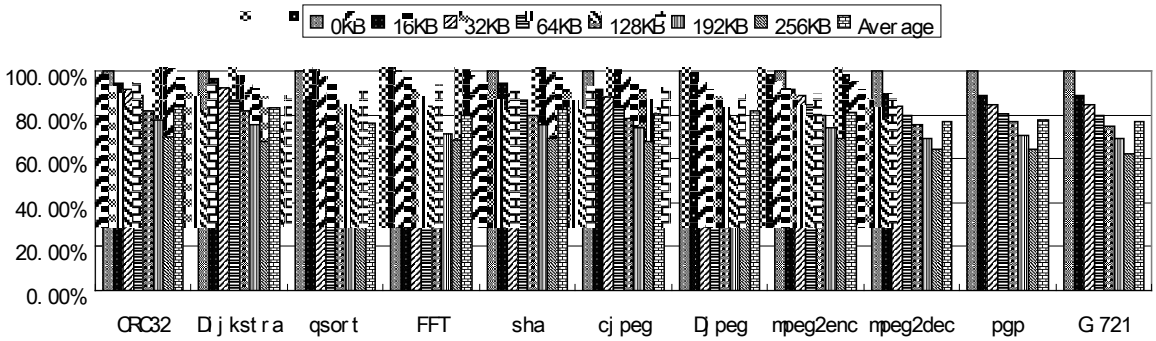


(b) Power Consumption of four Parallel Program

Figure 7. Optimization of four Parallel program

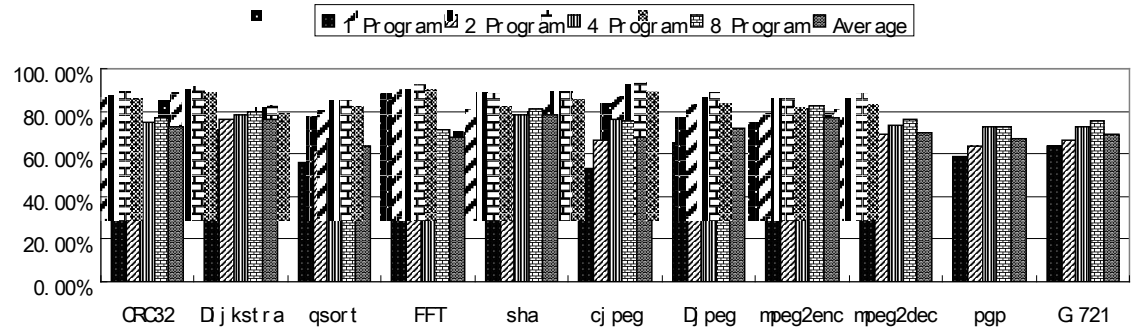


(a) Execution Time Optimization of eight Parallel Program

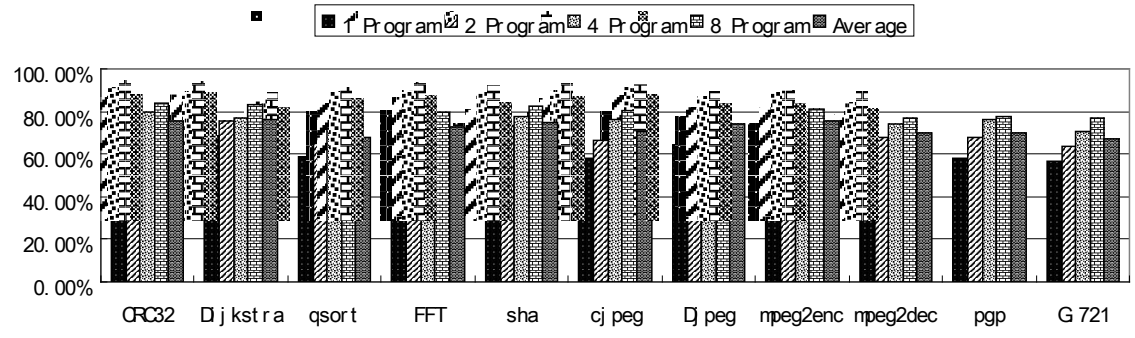


(b) Power Consumption of eight Parallel Program

Figure 8 Optimization of eight Parallel program



(a) Average Execution Time Optimization of Multi-Program



(b) Average Power Consumption of Multi-Program

Figure 9. Average optimization of multi-programs

From Figure 5, Figure 6, Figure 7 and Figure 8, we can see that the performance of the programs is improved while the power consumption is reduced with the increase in SPM capacity. When the number of parallel programs increases to two, the whole performance of each program decreases for they will compete for the limited SPM space especially when the size of SPM does not increase. When more SPM space is provided to the programs, more memory objects can be placed into SPM and the performance is improved. As we can see from four parallel programs and eight parallel programs, they have similar optimization results. However, if the number of memory objects of the parallel benchmarks is too much and the size of SPM is limited, there will be many swap operations between SPM and main memory among different programs. The result is that the improvement in performance and power is also very limited. In our experiments, when there are four or eight parallel programs and the size of SPM is 16KB, the improvement of the optimization is only about 10%. When the size increases, the improvement will also increase.

When there are more programs, the management by SMPG will become more complex. There may be more memory objects from different programs. SMPG should manage SPM space carefully for memory security. It will spend more time in SPT management.

Notice that in each test case, there is still a strange phenomenon: when the size of SPM is very large for the programs, the optimization will decrease slightly. According to our research, we think the reason is that when the size of SPM becomes larger compared to the number of memory objects, SPMG has to spend more time in managing SPM itself and more memory means more power consumption. Thus if more suitable memory object candidates can be generated, it will improve the performance when the size of SPM is large enough.

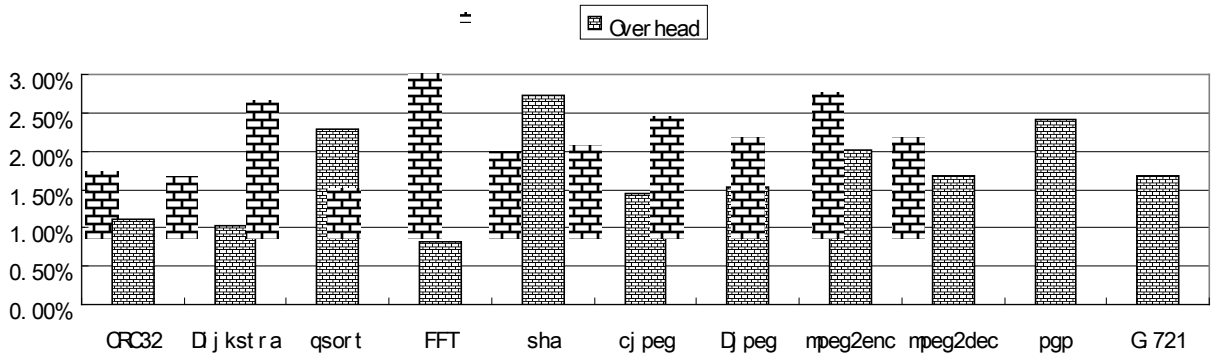


Figure 10. Overhead of the optimization

We also test the overhead of the optimization. The source of overhead comes from the SPMG operations on memory objects and the modification on the source code of the programs. Figure 10 shows that the overhead is no more than 3 percent both in execution time and power consumption. It is acceptable compared to the improvement of the optimization.

VI. CONCLUSIONS AND FUTURE WORK

Memory system is always the bottleneck between the memory and processors. On-chip memory is an active solution for this problem. SPM is software controlled memory, which is also on-chip memory. The existing research shows that SPM can improve the performance and reduce the power consumption of the systems. In this paper, we propose a novel methodology for SPM based optimization, which can be used not only in embedded systems but also in desktop computers and server. Our experimental results show that this methodology can reduce both the execution time and the power consumption effectively.

There is still much work to do in the future. First, more automatic tools should be provided for the optimization. Second, the corresponding interface should be designed for programmers. And at last, this methodology should be extended to support multicore/manycore processors.

ACKNOWLEDGEMENTS

We would like to thank Prof. Chengcheng Guo and Puliu Yan for their great supports in the experiments. This work was also partially sponsored by Intel project “Research on Service Discovery in Vehicule Ad hoc Network”.

REFERENCES

[1] G. E. Moore. “No exponential is forever: but "Forever" can be delayed!”, IEEE Intl. Solid-State Circuit Conf., Lisbon, Portugal, February 12-15, 2003, pp. 20-23.

[2] C.M. Kirsch and R. Wilhelm. “Grand challenges in embedded software”, 7th ACM&IEEE Intl. Conf. on Embedded software, Salzburg, Austria, September 30 - October 03, 2007, pp. 2-6.

[3] ITRS. “International Techbology Roadmap for Semiconductors”, 2009, http://www.public.itrs.net.

[4] W.A. Wulf and S.A. Mckee. “Hitting the Memory Wall: Implications of the Obvious”, ACM Computer Archtiecture News, Vol. 23, Issue 1, March, 1995, pp. 24-29.

[5] W.R. Fujita and K. Yanagisawa. “Low-power and high-speed advantages of DRAM-logic integration for multimedia systems”, IECE Transactions on Electron, Vol. 80, Issue 12, December, 1997, pp. 1523–1531.

[6] J. Hennessy and D. Patterson. Computer Architecture A Quantitative Approach, Palo Alto, CA: Morgan Kaufmann, 3 edition, 2002.

[7] P.R. Panda, N.D. Dutt. and A. Nicolau. “Efficient Utilization of Scratch-Pad Memory in Embedded Processor Applications”, 1997 European conference on Design and Test, Paris , France, March 17-20, 1997, pp. 7-11.

[8] S. Steinke, L. Wehmeyer, B. Lee and P. Marwedel. “Assigning Program and Data Objects to Scratchpad for Energy Reduction”, IEEE Conf. on Design, automation and test in Europe, Paris , France, March 4-8, 2002, pp. 409-415.

[9] M. Verma, S. Steinke and P. Marwedel. “Data partitioning for maximal scratchpad usage”, 2003 conference on Asia South Pacific design automation, Kasuga, Japan, January 21-24, 2003, pp. 77-83.

[10] S. Anantharaman and P.R. Panda. “An Efficient Data Partitioning Method for Limited Memory Embedded Systems”, ACM SIGPLAN Workshop on Languages, Compilers, and Tools for Embedded Systems, Montreal, Canada, June 19–20, 1998, pp. 108-122.

[11] O. Avissar, R. Barua and D. Stewart. “An optimal memory allocation scheme for scratch-pad-based embedded systems”, IEEE Trans. on Embedded Computing Sys. Vol. 1, Issue 1, November, 2003, pp. 6-26.

[12] K.D. Cooper and T.H. Harvey. “Compiler-controlled memory”, eighth international conference on Architectural support for programming languages and operating systems, San Jose, California, United States, October 02 - 07, 1998, pp. 2-11.

[13] M. Kandemir and A. Choudhary. “Compiler-directed scratch pad memory hierarchy design and management”, 39th conference on Design automation, New Orleans, Louisiana, USA, June 10 - 14, 2002, pp. 628-633.

[14] O. Ozturk, M. Kandemir and I. Kolcu. “Shared scratch-pad memory space management”, 7th International Symposium on Quality Electronic Design, San Jose, CA, March 27-29, 2006, pp. 576-584.

[15] A. Ramachandran and M.F. Jacome. “Xtream-fit: an energy-delay efficient data memory subsystem for embedded media processing”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 24, Issue 6, June, 2005, pp. 832-848.

[16] A. Ramachandran and M.F. Jacome. “Energy-delay efficient data memory subsystems: suitable for embedded media ‘processing’”, IEEE Signal Processing Magazine, Vol. 22, Issue 3, May, 2005, pp. 23-37.

[17] F. Angiolini, L. Benini and A. Caprara. “Polynomial-time algorithm for on-chip scratchpad memory partitioning”, 2003 international conference on Compilers, architecture and synthesis for embedded systems, San Jose, California, USA, October 30 - November 01, 2003, pp. 318-326.

[18] F. Angiolini, L. Benini and A. Caprara. “An efficient profile-based algorithm for scratchpad memory partitioning”, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2005, Vol. 24, Issue 11, November, 2005, pp. 1660-1676.

[19] P.R. Panda, N.D. Dutt and A. Nicolau. “Data memory organization and optimizations in application-specific systems”, IEEE Design & Test of Computers, Vol. 18, Issue 3, May, 2001, pp. 56-68.

[20] M. Kandemir and I. Kadayif. “Compiler-directed selection of dynamic memory layouts”, ninth international symposium on Hardware/software codesign, Copenhagen, Denmark, April 25-27, 2001, pp. 219-224.

[21] E. Brockmeyer, M. Miranda, H. Corporaal and F. Catthoor. “Layer Assignment echniques for Low Energy in Multi-Layered Memory Organisations”, IEEE Conf. on Design, Automation and Test in Europe, Munich, Germany, March 3-7, 2003, pp.1070-1075.

[22] B. Egger, J. Lee and H. Shin. . “Scratchpad memory management for portable systems with a memory management unit”, 6th ACM & IEEE Intl. Conf. on Embedded software, Seoul, Korea, October 22-25, 2006, pp. 321-330.

[23] O. Ozturk, M. Kandemir and I. Demirkiran. “Data compression for improving SPM behavior”, 41st annual conference on Design automation, San Diego, California, USA, June 07-11, 2004, pp. 401-410.

[24] A.R. Ravindran, P.D. Nagarkar and G.S. Dasika. “Compiler Managed Dynamic Instruction Placement in a Low-Power Code Cache”, ACM Intl. Symp. on Code generation and optimization, San Jose, California, March 20 - 23, 2005, pp. 179-190.

[25] I. Issenin, E. Brockmeyer, M. Miranda and N.D. Dutt. “Data Reuse Analysis Technique for Software-Controlled Memory Hierarchies”, IEEE Conf. on Design, automation and test in Europe, Vol. 1, Paris, France, February 16 - 20, 2004, pp. 202-207.

[26] M. Ruggiero, A. Guerri and D. Bertozzi. “Communication-aware allocation and scheduling framework for stream-oriented multi-processor systems-on-chip”, IEEE Conf. on Design, automation and test in Europe, Munich, Germany, March 06 - 10, 2006, pp. 3-8.

[27] B. Scholz, B. Burgstaller and J.L. Xue. “Minimizing bank selection instructions for partitioned memory architecture”, 2006 international conference on Compilers, architecture and synthesis for embedded systems, Seoul, Korea, October 22 - 25, 2006, pp. 201-211.

[28] S. Udayakumaran and R. Barua. “Compiler-decided dynamic memory allocation for scratch-pad based embedded systems”, 2003 Intl. Conf. on Compilers, architecture and synthesis for embedded systems, San Jose, California, USA, October 30 - November 01, 2003, pp. 276-286.

[29] S. Udayakumaran, A. Dominguez and R. Barua. “Dynamic allocation for scratch-pad memory using compile-time decisions”, ACM Trans. on Embedded Computing Sys., Vol. 5, Issue 2, May, 2006, pp. 472-511.

[30] S. Udayakumaran and R. Barua. “An integrated scratch-pad allocator for affine and non-affine code”, IEEE Conf. on Design, automation and test in Europe, Munich, Germany, March 06 - 10, 2006, pp. 925-930.

[31] P.R. Panda, N.D. Dutt And A. Nicolau. “Data memory organization and optimizations in application-specific systems”, IEEE Design & Test of Computers, Vol 18, Issue 3, May, 2001, pp. 56-68.

[32] M. Kandemir, J. Irwin, G. Chen and I. Kolcu. “Banked scratch-pad memory management for reducing leakage energy consumption”, 2004 IEEE/ACM International conference on Computer-aided design, San Jose, California, USA, November 07 - 11, 2004, pp. 120-124.

[33] Intel Corporation, Intel XScale® PXA27x Processor Family, http://www.intel.com/design/pca/probref/253820.htm.

[34] W. Hu, T.Z. Chen, B. Xie and Q.S. Shi. “Embedded Real-Time Linux on Chip: Next Generation Operation System for Embedded System”, 8th Real-Time Linux Workshop, Lanzhou China, October 12-15, 2006, pp. 167-172.

[35] C. Lee, M. Potkonjak and W.H. Manione-Smith. “Mediabench: A tool for evaluating multimedia and communications systems”, 30th Annu. IEEE Conf. Microarchitecture, Research Triangle Park, North Carolina, United States, December 01 - 03, 1997, pp. 330–335.

[36] M.R. Guthaus. “Mibench: A free, commercially representative embedded benchmark suite”, IEEE Annual Workshop on Workload Characterization, Austin, TX, USA, December 2, 2001, pp. 3-14.