A Novel Operating System on Chip with Information Security Support for Embedded System

Wei Hu, Tianzhou Chen, Qingsong Shi, Gang Wang, Nan Zhang, Jijun Ma, Yi Lian
College of Computer Science, Zhejiang University, Hangzhou, Zhejiang, P.R.China
Email: ehu@zju.edu.cn

***Abstract*—System-on-chip (SOC) has provided more powerful functions for embedded systems. Scratchpad memory (SPM), which is software-controlled on-chip memory, is used in embedded systems to reduce the speed gap between the processors and the memory and the power consumption of memory. ChipOS, a novel operating system on chip with information security support for embedded system, is proposed in this paper. It has a microkernel residing in SPM. This microkernel can execute absolutely. And it can also encapsulate the processor resources and provide virtual resources to general operating system (GPOS). At the same time, ChipOS can provide information security service through the encapsulation. The experimental results show that ChipOS has better response time and lower power consumption compared with GPOS and the security framework of ChipOS will protect GPOS with flexibility.**

***Index Terms*—scratchpad memory, operating system, information security, power-efficient, system-on-chip, embedded system**

I. INTRODUCTION

Embedded system has made great advance in recent years. There are also more intense requirements from the high performance and low power consumption. System-on-chip (SOC) has provided more powerful functions for embedded systems and the embedded system designers can adopt the new features of on-chip memory to improve the memory hierarchy in embedded systems. Memory issues are very important in embedded system design for the performance, power consumption and the cost of implementation. The most important on-chip memory is the on-chip SRAM, which is also called scratch-pad memory (SPM). SPM has faster access speed and lower power consumption compared with the traditional DRAM based memory and it does not need the extra tag bits and the comparators compared with caches. The study on the comparison of SPM and cache shows that SPM has 34% smaller die and 40% lower power consumption than cache with the same capacity [1]. At the same time, the use of cache in embedded system is still controversial [2].

SPM is software-controlled memory which is different from cache which is also on-chip SRAM but it is hardware-controlled [3]. The programmers or the compilers have to control the space allocation and recall. Though SOC makes on-chip memory possible, the capacity of SPM is still small compared with the whole address space.

How to take advantages of this type of on-chip memory is still under discussion for its special features. Most previous efforts on SPM utilization focus on the optimization of the applications. The basic idea is that the programs are scanned and analyzed and then the frequent used parts of the programs including code and data will be selected as the memory objects. Most researches have to resort to the help of the compiler to decide which segments of the programs are the potential memory objects. These memory objects can be allocated SPM statically or dynamically according to different approaches. For example, global variables in the same program are singled out and allocated to SPM to improve the system' performance when the utilization of SPM is first explored [4]. The main problems are that how to find the memory object and how to allow the memory objects from different programs sharing the SPM space. They are the hot topics in SPM research.

In this paper, we propose a new approach to utilize SPM to improve the performance and reduce the power consumption. Embedded operating system is the most important software in the whole system and it is also the most used software. If the architecture of EOS can be re-designed and take the advantages of SPM, the performance of the whole system will be improved greatly. Different from the existing approaches, we take the embedded operating system (EOS) into account.

The paper is organized as follows. In Section 2, we describe the related work. In Section 3, we define the architecture of our research. The microkernel and the components of ChipOS are presented in Section 4. In Section 5, the encapsulation based on ChipOS is described. The ChipOS based grid computing is depicted in Section 6. And in Section 7, we discuss our experimental results. At last, wee draw our conclusions in Section 8.

II. RELATED WORK

Different approaches have different ideas to solve the map from the memory object to the memory space [4] in SPM utilization. Static optimization is first proposed to

doi:10.4304/jsw.4.10.1053-1060

allocate SPM space. The compilers will scan the programs and select the potential memory objects. These memory objects will be allocated to SPM space and occupy the space during the execution without being swapped out [5, 6]. However, the effects of such optimizations are also restricted for the limited SPM space. Only a few memory objects will be the "children of fortune". Dynamic optimization is proposed to solve the problems in the static optimizations. The memory objects will be pre-analyzed by the compilers. The potential memory objects will be found out as many as possible during the compile process. The memory objects with non-overlapped life time can share the same SPM space [7, 8]. Or some memory objects can be kicked out of the SPM space and their space can be allocated to new memory objects [9, 10]. The dynamic allocation will increase the utilization rate through the share of SPM space.

The current approaches have to re-compile the programs for many times for the different types of embedded processors. The cost is too high to use in real systems. Furthermore, the current approaches do not take EOS into account. EOS is always in execution to maintain the stability of the systems. If EOS can be allocated to SPM, the on-chip memory will make EOS have better performance. In this paper, a novel operating system on chip with information security support for embedded system is proposed titled as ChipOS. ChipOS has improved performance and lower power consumption based on the optimization from SPM.

III. ChipOS Architecture

A. Processor Model

The processor model used for ChipOS design is shown in Fig. 1. There are a processor core, cache and SPM on chip. Generally the off-chip main memory is realized by DRAM. Cache and SPM has the same access time, which is one cycle while the access time of main memory is about 10 – 20 cycles [3]. The data stored in SPM will not be sent to cache. When there are data requests, the data will be sent to processor core from cache if cache hits or from SPM if the data are in SPM. Otherwise, the data will be fetched from the off-chip memory.

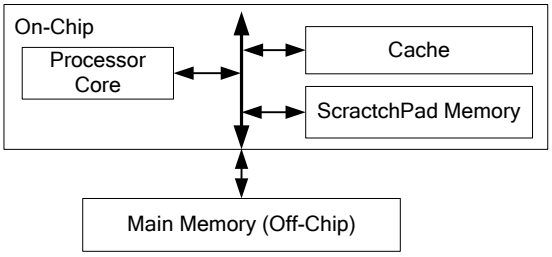

Figure 1. Processor model of the embedded processor with SPM.

The address space of SPM is mapped to the uniform space of the whole system. SPM space will only occupy a very small part of the whole address space for its limited capacity. In fact, many embedded processors have adopted SPM as the fixed configuration on chip. These processors include low-end processors such as some DSP [11] and some high-end processors such as Intel/Marvell XScale PXA 272 [12]. In some processors, cache is replaced by SPM directly such as Motorola 68HC12 [13].

B. ChipOS Architecture

Traditionally, EOS resides in main memory. So EOS can have abundant functions for the large capacity of main memory. However, SPM has only limited capacity and the new ChipOS has to be suitable to such restriction. There are three principles when we design ChipOS. (1) ChipOS should be small. (2) The architecture should be optimal; (3) It is organized by components. As shown in Fig. 2, ChipOS has four parts: microkernel, components, security and the extensions.

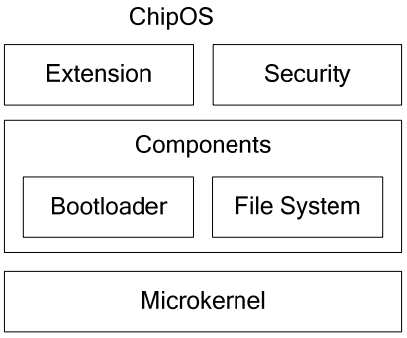

Figure 2. The architecture of ChipOS.

Microkernel is the core of ChipOS. It plays the most important role in ChipOS architecture. This microkernel is designed to support the basic service of an embedded operating system [14]. The traditional EOS is analyzed and the most important functions are selected and kept in the microkernel. Components are some important modules, which can also be provided by the off-chip system. The current components are the ChipOS bootloader [15] and the file system [16]. They have close relationship with microkernel. ChipOS is designed with information security support. When it executes in parallel with the general purpose operating system (GPOS), ChipOS can detect the viruses or the intrusions to GPOS [17][18].

ChipOS only has limited functions compared with the traditional EOS. ChipOS can encapsulate the on-chip resources and then provide the virtual resources to GPOS [19]. The traditional EOS can execute based on these virtual resources. Thus more than one EOS can execute in parallel. The encapsulation of ChipOS can also provide ChipOS extension. The current extension is that ChipOS can provide a novel way to construct the grid computing infrastructure based on the ChipOS supported dual OS [20]. In the following sections, the four parts of ChipOS will be described in detail.

IV. Microkernel and Components of ChipOS

A. Microkernel

Micorkernel will reside in SPM during the execution. It only provides the basic functions as a tidy embedded

operating system. Thus microkernel can be divided into five modules. The five modules are resource management module, task management module, SPM management module, power management module and scheduler module. Each module only has a single function. This will be helpful to reduce the size of this microkernel. When they are mapped to SPM space, they have their own spaces, which are called regions.

Resource management module is responsible for the management of the on-chip resources and recording the use of the resources. Microkernel is the software which is closest to the processor core. The resources on chip are under the control of microkernel. Thus the resource management module is designed to manage these resources. It will maintain a resource table according to the records. There are two advantages: first, the resource table will be provided to the task management module for task scheduling and second, it is possible to encapsulate the on-chip resources to provide virtual resources through the resource table.

Task management module is responsible for the scheduling of the tasks in the system. The size of SPM is limited, so the number of the tasks executing in SPM will be restricted. A set division based scheduling algorithm is used in microkernel [21]. The tasks are divided into different sets according to their features and then the different sets have different scheduling algorithms based on various resource requirements. Then the scheduling sequences can be calculated as preparation partly. It will provide real-time response for embedded system.

SPM management module is responsible for the management of the SPM space. The information of SPM space will be collected and maintained by SPM management module. When there is more than on one process in parallel to share SPM, this module will also manage the operations such as request and swap etc. on the memory objects of these processes.

Power management module is responsible for the power control for the system. Power consumption is very important in embedded system. Power management module has adopted dynamic voltage scaling (DVS) technology [22] to reduce the power consumption. The frequency and voltage of the processor core will be scaled down in the appropriate moment during the execution to reduce the power consumption. When the system tends to be busy again, the frequency and voltage will be restored to normal status.

ChipOS has different components, while the size of SPM is associated with the type of the processors. Though ChipOS is designed optimally, the size of SPM will limit the loading of some components. Scheduler module is responsible for the scheduling of the components when SPM space is not enough or there are some special requirements to load or swap out some components.

B. Component-Based ChipOS

The different modules will be extracted and re-organized according to the design of ChipOS. These modules will be more independent from each other. Thus they can be abstracted as the components of ChipOS. The components are the basic units of ChipOS. The component-based ChipOS can be adjusted to meet the different requirements from the using environments through the configuration of the components. And at the same time, the old components can be replaced by the new components. This will improve the performance and make ChipOS be scalable.

All the components will have the standard interface by definitions. The information of the components can be obtained from these standard interfaces. These interfaces can also be used to manage the components and they will be the channels of the data transmission among the components. Furthermore, the components can be organized in the component library. And some components can also be stored in the flash on chip. They can be scheduled into SPM according to the requirements of the configuration.

The five different modules of the microkernel are the components of ChipOS. These components are resource management component, task management component, SPM management component, power management component and scheduler component. That is why scheduler module can switch the components in and out of SPM space. There are also some other components of ChipOS. In the following sub-sections, the ChipOS bootloader and ChipOS based file system will be described in detail.

C. ChipOS Bootloader

Bootloader is used in embedded system to initialize the hardware and load the operating system for different hardware platform of embedded systems. Traditionally, bootloader can often be split into two stages. In the first stage, the binary code of the bootloader is found and loaded into memory from the flash memory. And then bootloader will detect, check and initialize the hardware resources in the system. In the second stage, bootloader will find the compressed kernel image and appropriate root file system image and load them into memory. Once these operations are completed, bootloader will hand the control of the system to the kernel. In this stage, bootloader will reside in SDRAM.

The performance of the second stage of bootloader can be improved for the residence in SDRAM. SDRAM has much access time compared with SPM. The microkernel of ChipOS is also an operating system for embedded system. So it needs the bootloader. And at the same time, GPOS in parallel with ChipOS will also need bootloader to complete the complicated functions before its startup. The bootloader of ChipOS can be more effective through SPM. The key is that the second stage of bootloader should be done in SPM, which is faster than flash or SDRAM.

ChipOS bootloader has three load stages. The basic loader is stored into the flash memory as the traditional bootloader. In stage 1, the basic loader first initializes the hardware resources and then the ChipOS laoder will copy the code of ChipOS (microkernel) inot SPM. ChipOS first starts up in stage 2. And then if the GPOS also needs to load, the stage 3 begins. In stage 3, a loader for GPOS is created and it will load the GPOS into memory. After

ChipOS is loaded, a special user interface is provided to the users for their commands. And thus different GPOS kernels can be downloaded from remote hosts and then loaded in memory.

D. ChipOS Based File System

Traditional embedded file systems reside in main memory to provide the service. Thus these file systems have to endure the long delays of the main memory accesses. According to the design of ChipOS, SPM will also make the embedded file system on chip possible. ChipOS based file system, which is also called SPM file system (SPMFS) is designed to improve the performance of the file system. SPMFS only has the key functions of the file system. It is also a component titled micro file system component (MFSC). MFSC is the core component of the SPMFS.

There are three layers of SPMFS. MFSC is designed to be stored in the flash memory on chip for fast startup. MFSC itself will not be modified during the execution to avoid the writing back operations. If there are enough SPM space, MFSC will have its own space to manage the file data. It will maintain an open file table and special buffers to manage the files and provide faster responses. MFSC is designed as small as possible. It can be scheduled into SPM by the scheduler module of the microkernel. The common functions of embedded file system are provided in basic file system in main memory. And when there is no SPM space for MFSC's execution, basic file system can provide all the functions and work as the traditional file systems. Flash memory will provide large storage as the third level.

V. ENCAPSULATION

ChipOS is designed based on the on-chip memory. It can control the on-chip resources according to its features. ChipOS is close to the processor compared with GPOS. And the microkernel of ChipOS also has the resource management module and maintains the resource table. ChipOS can encapsulate the on-chip resources and provide virtual resources to GPOS. The structure of the encapsulation is shown in Fig. 3.

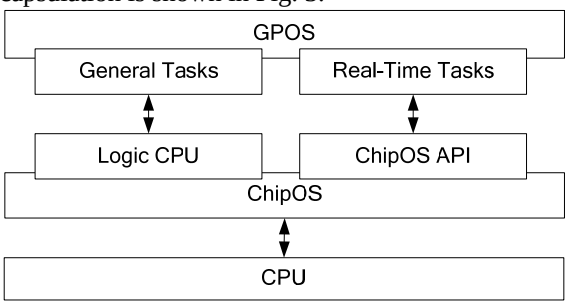

Figure 3. Encapsulation structure

ChipOS resides in SPM and controls all the resources on chip such as SPM, on-chip flash and on-chip processing units. GPOS resides in main memory, which is SDRAM. And ChipOS encapsulates these resources and organizes them as a logic CPU. This logic CPU is provided to GPOS. The physical CPU is controlled by ChipOS and concealed to GPOS. GPOS itself executes on the logic CPU and GPOS will dispatch its general tasks to execute on the logic CPU as this logic CPU is a real hardware resource. These general tasks all execute on GPOS. Some special tasks of GPOS especially the real-time tasks have specific solution. ChipOS will provide ChipOS API to these tasks. The real-time tasks will execute in SPM through ChipOS. Thus SPM can improve the real-time response. Through ChipOS API, real-time tasks will have better performance while the general tasks will not be impacted.

There are also some restrictions to protect GPOS when ChipOS encapsulate the processor. First, ChipOS can reserve some main memory space for its future use, but it must comply with the memory policies of GPOS. And it can not modify the main memory occupied by GPOS without restrictions. Second, ChipOS can manage files and share some files with GPOS through SPMFS. But the off-chip file system of GPOS can not be modified by ChipOS directly. The operations on such files including read/write can only be executed by owner GPOS. At last ChipOS will never be able to impact or change the scheduling algorithms of GPOS. It can only schedule the real-time tasks from GPOS through the special interfaces (ChipOS API). The encapsulation by ChipOS can provide new security mechanisms as described in Section VI. And the encapsulation provides the possibility for multi-GPOS in parallel. ChipOS can encapsulate the resources as several or more logic processors and then support more than one GPOS. The virtual resources can meet the requirements of multi-GPOS cooperation.

VI. CHIPOS SUPPORTED SECURITY

ChipOS can support security through the encapsulation of the hardware resources. The current security support in ChipOS is the virus detection and intrusion detection as depicted in this section respectively.

A. Virus Detection Framework Based on ChipOS

The virus detection framework based on ChipOS is designed as a Demilitarized Zone (DMZ), which is a computer host or small network inserted between the private network and the outside network as the isolated zone [17]. The encapsulation provided by ChipOS can separate GPOS from the real processor. GPOS can only access the resources through the logic processor. Then all the accesses from the GPOS can be observed by ChipOS and all the accesses to the resources will also be monitored by ChipOS. Our DMZ is based on such analysis and the DMZ framework is shown in Fig. 4.

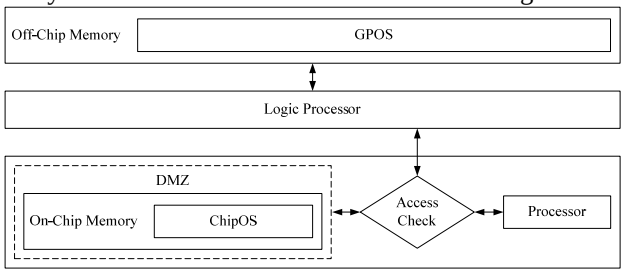

Figure 4. DMZ framework

DMZ has a single entrance for GPOS. During the execution, GPOS can stop and then trap itself into ChipOS via interrupts when there are some suspicious activities detected. And then ChipOS will have the control. First ChipOS will save the context of trapped GPOS and then ChipOS scans the memory and GPOS to find the source of the suspicious activities. When the detection is completed, GPOS can resume through restoring its context saved by ChipOS if there are no serious problems. If the problems can not be fixed, ChipOS will still control the system and give the error messages to the users. The users can stop GPOS and restart it or have some other operations.

DMZ provides an anti-virus framework to embedded systems. ChipOS itself can not scan the software and find the virus. The anti-virus software can be loaded to ChipOS for the scan, detection and cleaning jobs. ChipOS is just a platform. It means that the virus detection programs are provided by third part companies and the maintenance and update of these programs are also done by these companies.

B. Intrusion Detection Architecture Based on ChipOS

Intrusion detection system (IDS) is designed to protect the systems from the network attacks. Traditional IDSs are running on operating systems to work and this is not a safe and dependable way [18]. Operating system and IDS themselves will also be attacked perhaps. ChipOS is an underlying operating system and transparent to GPOS. It provides the opportunity to design a different architecture for IDS. IDS based on ChipOS can have direct control of the hardware resource and will not be attacked by the programs from GPOS. Fig. 5 shows the new IDS architecture based on ChipOS. In this architecture, IDS is located in SPM.

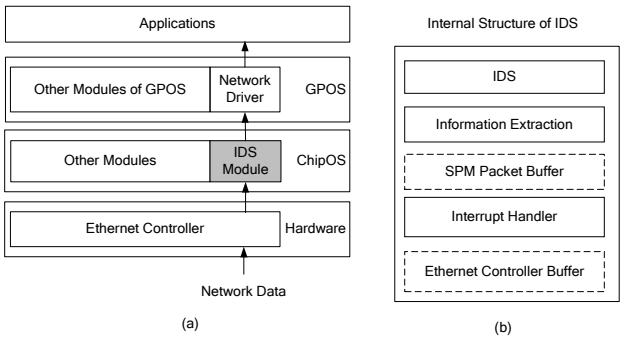

Figure 5. Intrusion detection architecture based on ChipOS

There are four layers in the intrusion detection architecture based on ChipOS as shown in Fig. 8 (a): hardware, ChipOS, GPOS and the applications. IDS is located in ChipOS as a module shown as the shadow area. SPM will make IDS have faster access time.This architecture is not an implementation of IDS itself. It provides an architecture to locate the detection system just as same as the virus detection framework based on ChipOS.

Our design can work as normal when there are some errors in GPOS for its location in SPM. SPM management module can allocate SPM space to IDS, monitor such space and then lock or protect such space. As shown in Fig. 5 (b), our design has the following modules including Ethernet controller buffer, interrupt handler, SPM packet buffer, information abstraction and IDS. The buffers are hardware components. And the interrupt handler and DIS are software components. The network data will be first received in the Ethernet controller buffer. Then interrupt handler will receive the interrupt signals and the raw data in Ethernet controller buffer will be copied to SPM packet buffer. Information extraction is responsible for splitting data stream into the single packets. At last, these packets will be sent to IDS. IDS will analyze these packets and find out the potential attacks. Thus IDS can complete its work with the help of ChipOS and improve its performance via SPM.

VII. CHIPOS BASED GRID COMPUTING

ChipOS based grid computing is the extension of ChipOS and it has three layers as shown in Fig. 6. The Internet and ChipOS are organized as the basic fabric of the grid. The second layer is the GPOS layer. In this layer, GPOS will execute as normal as there are no ChipOSs. The third layer consists of the applications.

All the nodes in such a grid have the same structure. And each node in this fabric has three parts: ChipOS, GPOS and the applications. So each node will have a ChipOS residing in SPM. ChipOS provides grid API as the interfaces to the upper-level. Some basic functions are also provided by the grid API. GPOS will provide the traditional grid service to the applications. Grid service is a part of GPOS and it communications with ChipOS through the interfaces between grid service (in GPOS) and grid API (in ChipOS).

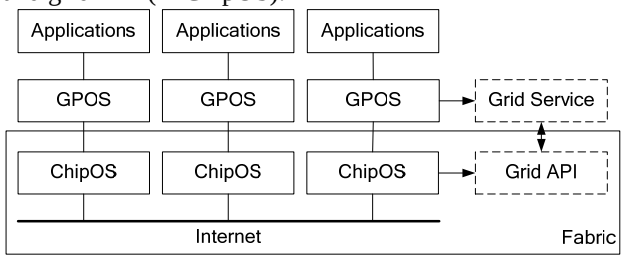

Figure 6. Layers of ChipOS based grid computing

The main change between the traditional grid computing and ChipOS based grid computing is that some basic functions of grid middleware are directly integrated into ChipOS for its higher performance. ChipOS is responsible for the following functions: package management, connectivity check, transportation and necessary communication among ChipOSs. The other functions are provided by GPOS.

In ChipOS based grid computing, the task distribution process is different as described in the following. (1) Grid application will call the grid service and grid service will notify ChipOS through grid API; (2) When ChipOS is ready, it will collect the data from grid application and distribute the tasks to the other nodes in the fabric; (3) When the nodes complete their tasks, the results are returned to the original ChipOS; and (4) at last, the results will be returned to the grid application.

VII EXPERIMENTAL RESULTS

A. Experiment Setup

The hardware platform used in the experiments is the Intel/Marvel PXA272 embedded processor [19], which has 256K SPM. SPM in PXA 272 is organized as banks. Each bank can be set in standby mode and the data will be stored in the banks in run mode. So the size of SPM in PXA272 can be adjusted during the execution. GNU tool chain [23] is adopted as the development toolkit. The implementation of ChipOS is based on the embedded Linux [24]. Eleven benchmarks are selected from two benchmark sets MIBench [25] and MediaBench [26].

TABLE I. 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

B. Parameters of ChipOS

ChipOS is cut down from embedded Linux. Only several most important functions are kept in ChipOS. The reserved modules are organized as the microkernel. The other modules including the security support and the extensions are configured as the components of ChipOS. The basic parameters of ChipOS are shown in Table 2. The faster access time of SPM helps ChipOS to improve its parameters. The size of ChipOS is very small compared with the other operating systems as shown in Fig. 7. But ChipOS has only several functions and it can not have the same functions to them according to the restriction of the limited size of SPM.

TABLE II. PARAMETERS OF CHIPOS*

Item	Parameter
Load Time	3 ms on average
Kernel Size	About 8 KB
Switch Time	4 us on average

*The frequency of CPU is 520 MHz

C. Experimental Results and Analysis of Microkernel

The size of SPM is set to 0KB (without SPM), 16KB, 32KB, 64KB, 128KB, 192KB and 256KB. The size of cache is set to 32KB and the states of cache are on or off respectively during the experiments. The benchmarks are scheduled by the microkernel to test the scheduling time. Power consumption of the scheduling is also tested. Fig. 8 shows the results of the task scheduling by the microkernel.

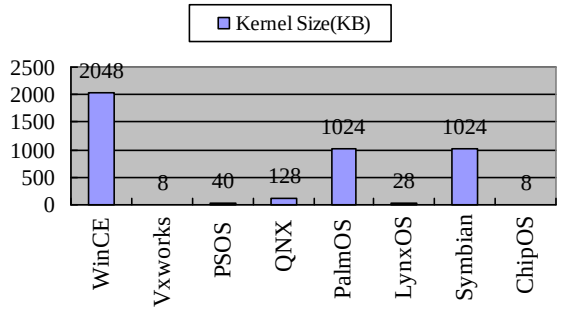

Figure 7. Kernel size of different embedded operating system

Fig. 8 (a) shows that the task scheduling time on average is reduced 20.3% (cache off) and 25.9% (cache off). The faster access time of SPM helps task management of ChipOS to reduce the switch time. When the size of SPM is increased, more related contexts can be stored in SPM and the time is further reduced. But when all the related contexts are contained, the optimization results will be constant. Fig. 8 (b) shows the power consumption of task management optimization. The average power consumption is reduced 26.1% (cache off) and 30.7% (cache on). The reduced main memory access times are the main reason of such improvements.

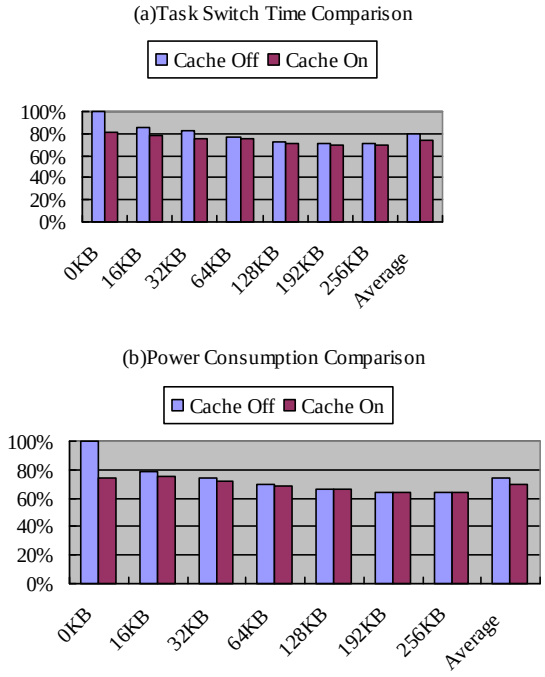

Figure 8. The experimental results of task management optimization

The execution time and the power consumption of microkernel are shown in Fig. 9. When the size of SPM is too small to contain all the components (without SPM or less than the size of microkernel), ChipOS has to fetch

data from main memory. The access time and the power consumption are both very large. This will make the advantages of ChipOS be deprived and lose the optimization. When ChipOS can execute in SPM completely, it can work effectively. The execution time is reduced 26.7% (cache off) and 30.8% (cache on) and the average power consumption is reduced 31.9% (cache off) and 36.3% (cache on) respectively. When the whole microkernel is in SPM, the optimization is better than only task management module in SPM.

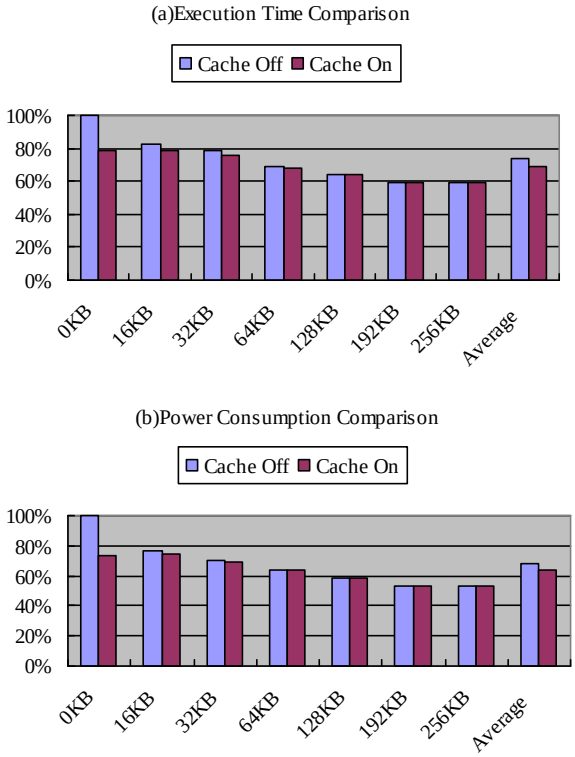

Figure 9. The experimental results of the microkernel

A data access program is created on both SPMFS and YAFFS2 [27] to test the performance of the file systems based on Nand Flash as shown in Fig. 10. Compared with YAFFS2, the performance of SPMFS is improved to especially when the sizes of the files are less than 256KB, which is the maximum size of SPM. One Mbytes data flow is constructed, which contains about 10% packets with intrusive data. The intrusive data consist of "bash" and consecutive "nop" etc., which are often used in buffer overflow attacks. For simplicity, the simplest comparison algorithm is adopted to detect the attacks by compared the packets with the signatures in signature records. This simple IDS is also used in GPOS for comparison.

Fig. 11 shows that the comparison stage consumes much time of GPOS IDS and ChipOS based IDS (56% for ChipOS based IDS and 86% for GPOS IDS). The reason is that duplicate loops used by the comparison algorithm are time consuming. ChipOS based IDS copies a bulk of data instead of copying the packets one by one. So ChipOS will have shorter latency in store to SDRAM stage. Copy to SPM stage only consumes 6.2% of the whole SPMOS based IDS latency. ChipOS based IDS only spends about 500ms in detecting instead of 6037ms by GPOS IDS. The performance of ChipOS based IDS is better than GPOS IDS.

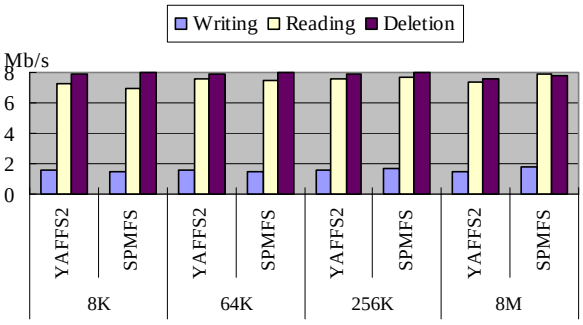

Figure 10. The comparison of YAFFS2 and SPMFS

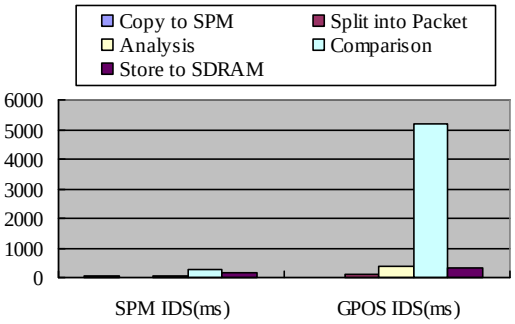

Figure 11. The comparison of IDS in ChipOS and GPOS

The experimental results show that ChipOS can take advantages of the characteristics of SPM. This make ChipOS can reduce the memory footprints and have better performance.

VIII Conclusions and Future Work

SOC provides different types of on-chip memory for embedded systems. Though SPM is software-controlled on-chip memory, it has more flexibility compared with the hardware-controlled on-chip memory. In this paper, we propose ChipOS for embedded system. ChipOS has a microkernel and it consists of different components. Our experimental results show that ChipOS has better performance compared with GPOS. Our future work is to expand ChipOS architecture to embedded multi-core systems. Many multi-core processors have SPM as the local memory. ChipOS is very small and it is suitable to work in such environments.

Acknowledgment

This work was supported by National Nature Science Foundation of China with granted No. 60673149, the National High-Tech Research and Development Program of China (863) with granted No. 2007AA01Z105 and the Research Foundation of Education Bureau of Zhejiang Province with granted No. Y200803333.

References

[1] R. Banakar, S. Steinke, B.S. Lee, M. Balakrishnan and P. Marwedel, "Scratchpad memory: A design alternative for cache on-chip memory in embedded systems", *In Proc. of 10th Intl. Symp. on Hardware/Software Codesign (CODES)*, pp. 73-78, ACM Press, 2002.
[2] J. Hennessy, D. Patterson, *Computer Architecture A Quantitative Approach*, Palo Alto, CA: Morgan Kaufmann, 3 edition, 2002.
[3] P.R. Panda, N.D. Dutt and N. Alexandru, "Efficient Utilization of Scratch-Pad Memory in Embedded Processor Applications", *In Proc. of the 1997 European Conf. on Design and Test*, pp. 7-11, 1997.
[4] F. Angiolini, M. Francesco and F. Alberto et al, "A post-compiler approach to scratchpad mapping of code", *In Proc. of the 2004 Intl. Conf. on Compilers, architecture, and synthesis for embedded systems*, pp. 259-267, 2004.
[5] K.D. Cooper and T.H. Harvey, "Compiler-controlled memory", *In Proc. of the 8th Intl. Conf. on Architectural support for programming languages and operating systems*,pp. 2-11, 1998.
[6] L. Wehmeyer, U. Helmig and P. Marwedel, "Compiler-optimized usage of partitioned memories", *In Proc. of the 3rd workshop on memory performance issues*, pp. 114-120, 2004.
[7] M. Kandemir, "A. Choudhary. Compiler-directed scratch pad memory hierarchy design and management", *In Proc. of the 39th Conf. on Design automation*, pp. 628-633, 2002.
[8] O. Ozturk, M. Kandemir and I. Kolcu, "Shared scratch-pad memory space management", *In Prec. of 7th Intl. Symp. on Quality Electronic Design*, pp. 576-584, 2006.
[9] M. Ruggiero, A. Guerri and D. Bertozzi et al, "Communication-aware allocation and scheduling framework for stream-oriented multi-processor systems-on-chip", *In Proc. of Conf. on Design, Automation and Test in Europe*, pp. 3-8, 2006.
[10] B. Scholz, B. Burgstaller and J.L. Xue, "Minimizing bank selection instructions for partitioned memory architecture", *In Proc. of the 2006 Int. Conf. on Compilers, architecture and synthesis for embedded systems*, pp. 201-211, 2006.
[11] Philips. LPC2290 16/32-bit Embedded CPU, http://www.semiconductors.philips.com/acrobatdownload/ datasheets/LPC2290-01.pdf, 2004.
[12] Intel Corporation. XScale® PXA27x Processor Family, http://www.intel.com/design/pca/probref/25382 0.htm.
[13] Motorola Corporation. CPU12 Reference Manual, http://e-www.motorola.com/brdata/pdfdb/microcontrollers/16bit/68hc12family/freemat/cpu12rm.pdf, 2000.
[14] T.Z. Chen, Y. Lian and W. Hu, "Chip OS: New Architecture for Next Generation Embedded System", *In Proc. of the 2006 Intl. Conf. on Embedded Systems and Applications*, 2006.
[15] N. Zhang, and J.J. Ma et al, "SPM-Based Boot Loader", *In Proc. of the 2008 Intl. Conf. on Embedded Software and Systems*, China, pp. 164-168, 2008.
[16] T.Z. Chen, F. Sha, W. Hu and Q.S. Shi, "A new type of embedded file system based on SPM", *In Proc. of the 3rd Intl. Conf. on Embedded Software and Systems*, Korea, pp. 174-180, 2007.
[17] T.Z. Chen, J.J. Ma, N. Zhang and Q.S. Shi, "A virus detection framework based on SPMOS", *In Proc. of the 2008 Intl. Conf. on embedded Software and Systems*, China, pp. 610-615, 2008.
[18] Q.S. Shi and D. Chen et al, "SPMOS-based Intrusion Detection Architecture", *In Proc. of the 5th IEEE Intl. Symp. on Embedded Computing*, China, pp. 82-87, 2008.
[19] Q.S. Shi and D.G. Feng et al, "Dual OS Support Peripheral Device Encapsulation", *In Proc. of the 5th IEEE Intl. Symp. on Embedded Computing*, China, pp. 67-72, 2008.
[20] J.J. MA, M. Cao, W. Ma and T.Z. Chen, "ChipOS based Grid Computing", *In Proc. of 7th WSEAS International conference on Simulation, Modelling and Optmization*, Beijing, China, pp. 171-176, 2007.
[21] T.Z. Chen, W. Hu, B. Xie and L.K. Yan, "A Real-Time Scheduling Algorithm for Embedded Systems with Various Resource Requirements", *In Proc. of International Workshop on Networking, Architecture, and Storages*, Shenyang, China, pp. 43-46, August 2006.
[22] T.Z. Chen and J.W. Huang et al, "Using Dynamic and Static Approach with Compiler-Assisted in Power Management for DVS-Enabled Embedded System", *WSEAS Trans. on Comp.*, vol 6, pp. 967-972, June 2007.
[23] CodeSourcery. http://www.codesourcery.com/
[24] W. Hu, T.Z. Chen, B. Xie and Q.S. Shi, "Embedded Real-Time Linux on Chip: Next Generation Operation System for Embedded System", *In Proc. of 8th Real-Time Linux Workshop*, pp. 167-172, China, 2006.
[25] M. R. Guthaus et al, "Mibench: A free, commercially representative embedded benchmark suite", *In Proc of IEEE Annu. Workshop on Workload Characterization*, pp. 3-14, 2001.
[26] C. Lee, M. Potkonjak, W. H. Manione-Smith, "Mediabench: A tool for evaluating multimedia and communications systems", *In Proc of 30th Annu. IEEE Conf. Microarchitecture*, pp. 330–335, 1997.
[27] Aleph One Company, "The Yet Another Flash Filing System(YAFFS)", http://www.aleph1.co.uk/yaffs/

Wei Hu was born in Xinyang, Henan Province of China in 1979. He got his bachelor degree in 2002, master degree in 2005 and Ph.D degree in 2008 with the major of computer science in Zhejiang University at Hangzhou, Zhejiang province.

He was an ASSISTANT when he was the Ph.D candidate. And he became the teacher after obtaining his Ph.D degree. Now he is the LECTURER of Zhejiang University in Hangzhou. His current research is computer architecture and embedded system.

Dr. Hu is the member of IEEE, ACM and CCF (China Computer Federation).

Tianzhou Chen was born in Lishui, Zhejiang Province of China in 1970. He entered Zhejiang University mixed class (Honors Program of Engineering), in 1990, and obtained his B.S. degree in computer software in 1994. He studied for M.S. degree in computer architecture, and received his PH.D degree in computer application from Zhejiang University in 1998.

He is a PROFESSOR of computer science at Zhejiang University. His current research is computer architecture and embedded system.

Prof. Chen is the member of IEEE, ACM and senior member of CCF (China Computer Federation).

Qingsong Shi was born in Hangzhou, Zhejiang Province of China in 1963. He obtained his B.S. degree in 1986.

After he obtained his B.S. degree, he became a LECTURER of computer science at Zhejiang University. And now he is an ASSOCIATE PROFESSOR of computer science at Zhejiang University. His current research is computer architecture and embedded system.

Assoc. Prof. Shi is the member of IEEE, ACM and CCF (China Computer Federation).