Optimizations for ARM11 MPCore on Computational Capabilities and Interrupt Processing

Wei Wang, Yu Zhou, Sidan Du*
Dept. of Electronic Science and Engineering
Nanjing University
Nanjing, China
coff128@nju.edu.cn

Abstract—**In this contribution, several optimization strategies for fully exploiting the performance of ARM11 MPCore have been discussed. Experimental studies are conducted for the basic computation performance and the multimedia processing performance of ARM11 MPCore. The basic computation operation includes integer operation and floating-point operation. Results show that an efficiency gain of a factor 3.8 for integer operation could be achieved when being optimized by using the OpenMP parallel computing model, and for floating-point operation, the value is 3 to 5 when adopting the VFP (vector float point) optimization method. With regard to the multimedia processing, two optimization methods are put forward, namely, the DCT optimization with VFP and the parallelization of audio decoding, video decoding and playing. The experimental results show that both methods could efficiently improve the performance of multimedia processing for ARM11 MPCore. In addition, considering that the interrupt load imbalance on multi-core processor may bottleneck the entire system's performance improvement, workload balancing methods basing on interrupt affinity are proposed for ARM11 MPCore, and results indicate these methods are effective.**

Keywords-ARM11 MPCore; vector float point; parallelization; interrupt affinity; load balancing

I. INTRODUCTION

In the last decades, various applications including network, multimedia, and digital signal process are widely implemented on embedded platform. As a result, the requirements of embedded processor's performance, as well as the high power efficiency, are growing rapidly. In order to solve this problem, the MPSOC (multi processor system on chip) has been introduced to design the embedded system [1]. Based on this, ARM has launched the MPCore [2], [3], [4], which is a synthesizable multiprocessor based on the ARMv6 and implements the ARM11 microarchitecture. Compared with the ARM9 processor family, the MPCore features configurable level 1 caches, vector floating point coprocessors (VFP), programmable distributed interrupt controller, etc. These features are expected to make MPCore have an excellent performance.

To give a full play to the MPCore's performance, the optimizations for current embedded software system must be considered. Typically, the traditional embedded applications are normally programmed using single-threaded programming model because of the constraints of single-core architecture. That is to say, they only have one thread and only use one processor when running, which means they could not run efficiently on the MPCore platform. So, multi-threaded programming model must be adopted. Furthermore, in order to fully exploit the advantage of the VFP module, the floating-point computation which is very important to multimedia processing must be optimized. The relationship between performance enhancing and power consumption of parallelized implementations on the MPCore platform had been studied in [5]. References [6], [7], [8] proposed parallel performance optimization methods on x86 platform, but did not apply them to embedded platform.

On the other hand, interrupt load balancing strategy should be considered on multi-core platform. In the Linux kernel supporting SMP, CPU0 is responsible for handling all interrupts by default configuration in order to maximize the use of CPU cache coherency [9]. References [10], [11] studied the network performance of SMP, and then proposed that this default configuration could cause interrupt load imbalance between CPUs, and lead to a poor network throughput. Many researches about interrupt load balancing methods on x86 platforms have been studied, but there is little on ARM platform yet.

The paper is organized as follows. Section II shortly discusses the basic architecture of the MPCore which could facilitate the understanding of our work. The following chapters III & IV perform experimental studies of the basic computation performance and the multimedia processing performance, and then propose optimization schemes. Section V discusses the interrupt affinity mechanism and brings forward the interrupt load balancing strategies. Finally, a conclusion is given in section VI.

II. MPCORE ARCHITECTURE

The ARM11 MPCore processor implements ARM architecture v6k with Java extensions as well as DSP and SIMD ISA extensions. This includes the 32-bit ARM instruction set, 16-bit Thumb instruction set, 8-bit Java instruction set, and a range of SIMD DSP instructions that operate on 16-bit or 8-bit data values in 32-bit registers. Furthermore, the ARM Intelligent Energy Manager (IEM) mechanism which includes a shutdown of unused resources

978-1-4244-4996-5/09/$25.00 ©2009 IEEE

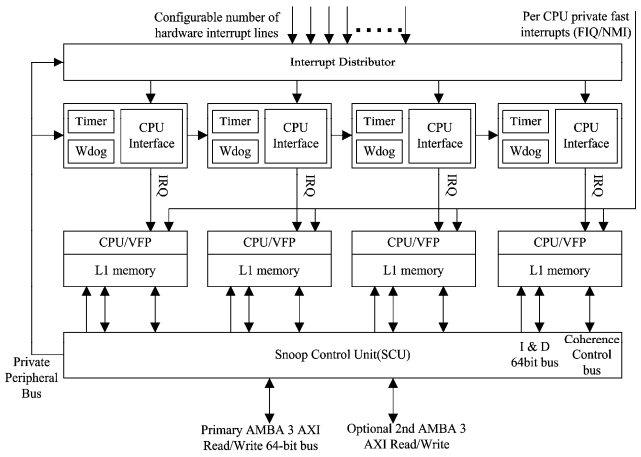

Figure 1. Architecture of ARM11 MPCore

and a dynamic voltage and frequency scaling support is implemented to gain high-performance and low-power. A block diagram of the MPCore architecture is shown in Fig. 1. The processor features:

- Up to four ARM11 CPUs,
- A snoop control unit (SCU) that is responsible for maintaining coherency among ARM11 CPUs L1 data caches,
- A private timer and a private watchdog per CPU,
- Single or dual 64-bit AMBA 3 AXI bus,
- Optional vector floating point coprocessors (VFP),
- Programmable distributed interrupt controller with support for legacy ARM interrupts.

The vector floating point unit provides a performance-power-area solution for embedded applications and high performance for general-purpose applications. It supports single-precision and double-precision add, subtract, multiply, divide, multiply and accumulate, and square root operations. Three separate pipelines are implemented and each pipeline can operate independently of the others and in parallel with them:

- Multiply and accumulate (FMAC) pipeline,
- Divide and square root (DS) pipeline,
- Load/store (LS) pipeline.

The distributed interrupt controller which is compatible with previous ARM interrupt handling architectures manages all interrupts. It provides:

- Masking of interrupts,
- Prioritization of the interrupts,
- Distribution of the interrupts to the target CPUs,
- Tracking the status of interrupts,
- Generation of interrupts by software.

The controller consists of one interrupt distributor and four CPU interfaces. The interrupt distributor centralizes all interrupt sources for the ARM11 MPCore processor before dispatching the highest priority ones to each individual CPU, while the CPU interfaces handle interrupt priority masking and pre-empted interrupts.

III. PERFORMANCE OF BASIC COMPUTATION

Two mainly aspects should be considered when analyzing and optimizing the basic computation performance of ARM11 MPCore, namely, integer operation and floating-point operation [12]. In order to present a brief view, the MPcore's computation performance is benchmarked and compared with the S3C2410A implementing ARM9 architecture. The embedded Linux operation system and GCC compiler are chosen as a base platform to accomplish this job, and both of S3C2410A and MPCore's processor frequency are set to 200M HZ. According to the specific features of MPCore's hardware architecture, when carrying out the optimization, parallel processing, hardware floating-point operation, etc., are adopted to improve the computation performance.

A. Integer operation

Regarding the integer computing capability, three operations including addition, multiply and division are concerned. To generate a benchmark report, MOPS (million operations per second) is taken as the performance indicator. The test utilities programmed in C language are used to count how many operations have been executed within a certain time interval, and the MOPS can be calculated through the count results. Because ARM11 MPCore can be configured up to four processor cores, common programming model that can only make full use of one processor is not suitable here. Parallel computing model must be adopted in order to take full advantage of MPCore's four processors. For this purpose, the OpenMP programming model [13] is introduced to optimize the test utilities on MPCore platform. The experimental results are shown in Fig. 2.

As can be seen from the results, the performance of single-threaded programs on MPCore does not improve obviously compared to the S3C2410A. This little part of the performance increase is mainly benefit from MPCore's newest pipeline and memory architecture. But if employing the OpenMP parallel programming model, the improvement becomes evident. For the MPCore platform only, the efficiency of the program with OpenMP (4 threads) could improve by 3.8 times than the one programmed single-threaded.

B. Floating-Point Operation

In this part, similar experiments are made. The performance of floating-point addition, multiply and division operations are

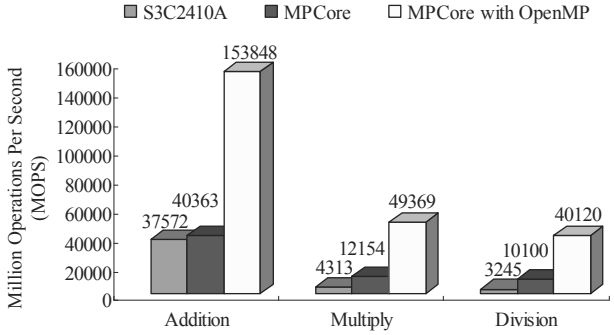

Figure 2. Analysis and optimization of integer computaion

taken into account. Typically, there are three ways to implement the floating-point computation. The first one is using software simulation for processors without floating-point processing unit. The second one is directly using the floating-point instruction set in assembly language for those processors with floating-point processing unit. The final one is programming in C language under the same conditions with the second method, and then translating the C statements to the floating-point instructions by compiler. The last approach has the best software portability, but has the less efficiency than the second way.

During the experiments, software simulation method is applied to S3C2410A. As to the MPCore, two kinds of tests are carried out. Firstly, the software simulation method is adopted merely just as the S3C2410A platform. Secondly, the VFP module is used to optimize the floating-point computation and the third approach mentioned in the previous section is employed. For the purpose of using VFP instructions on the MPCore platform, the options –mfpu=vfp –mfloat-abi=softfp must be used when compiling the test utilities with GCC. Fig. 3 gives the benchmark results of the floating-point operation.

The results indicate that the use of VFP is capable of enhancing the floating-point computation performance by about 3 to 5 times on MPCore platform, but the effect on the division operation is not obvious. Also, it is clear to conclude that even without the use of VFP, the MPCore's floating-point computation capability could improve by about 20 times compared with the S3C2410A. The reason for this can trace to the introduction of two level caches and the optimized system architecture of MPCore.

IV. PERFORMANCE OF MULTIMEDIA PROCESSING

Due to architecture limitations, the traditional embedded systems normally have poor floating-point computation capability which is very important for multimedia processing. Therefore, when designing an embedded system, dedicated DSP chips are used. However, as a newest embedded processor, the ARM11 MPCore has introduced multi-core technology as well as the VFP unit which could greatly enhance the floating-point computation capability. These features can be used to improve the traditional multimedia software and make the embedded system decode and play multimedia without dedicated DSP chips.

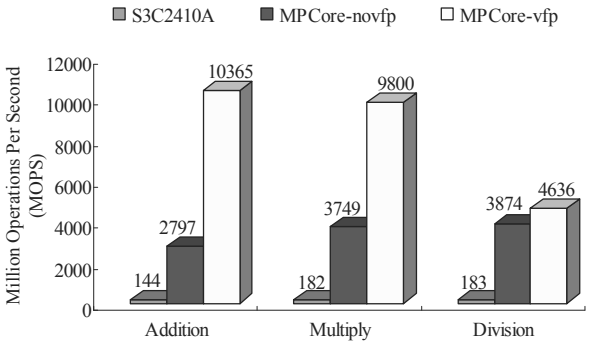

Figure 3. Analysis and optimization of floating point computation

A. DCT optimization

During the process of video decoding, DCT calculation is one of the most time-consuming parts. Conversely, it also means that a much better decoding efficiency could be achieved by carefully optimizing the DCT operation. The DCT computation involves floating-point operations, and therefore the VFP module could be used to optimize DCT calculation on MPCore platform. In this part, the DCT computing capability is benchmarked both on the MPCore and S3C2410A platform, and the Cooley-Tukey algorithm [14] which has the complexity of O (N log N) is chosen to compute DCT in the test utilities. The results are shown in Fig. 4.

From the results, it is concluded that when both adopting software simulation method to implement floating-point computation, the MPCore performs about 6.8 times better than the S3C2410A in computing DCT. For the MPCore platform only, when introducing the VFP module, the performance could improve by about 3.6 times.

B. Multimedia parallel processing optimization

Traditional media players are generally programmed by single-threaded, which could not run efficiently on multi-core platform. Considering that the MPCore can be configured up to four processor cores, multi-threaded programming model should be adopted to optimize the media players. Typically, the process of multimedia decoding can be divided into three parts, i.e. audio decoding, video decoding and playing. So when optimizing the media players, three separate threads are used to parallel multimedia decoding. Each thread is bound to a certain processor and manages one job. An MPEG4 format media file of VGA (640x480), QVGA (320x240), 25fps, 210.7 kbps is used to accomplish our experiments. The results are shown in Tab. I.

The test results enable us to conclude that the performance of media player with multiple threads can be improved about 50 percent than the one with single thread. But the playback is not very smooth. The reason is that the MPCore prototyping test chip has bottlenecks on screen display. If introducing the acceleration module to the LCD controller, it can be predicted that the performance of multi-threaded media player will improve more obviously.

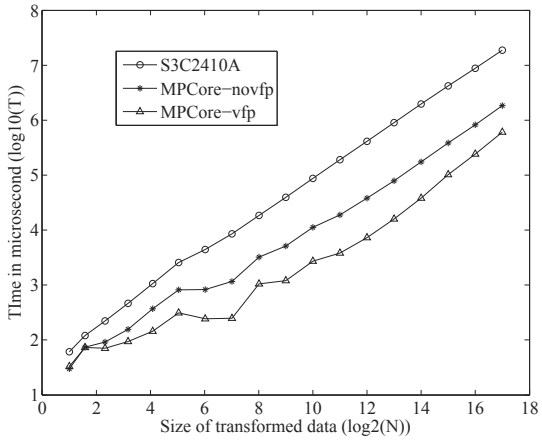

Figure 4. Analysis and optimization of DCT

TABLE I. PARALLELIZATION OPTIMIZATION OF MULTIMEDIA PROGRAM

Media Player	Different Resolutions	
	VGA	*QVGA*
Single-threaded	4.2fps	20.2fps
Multi-threaded	6fps	25fps

V. PERFORMANCE OF INTERRUPT PROCESSING

As previously mentioned, interrupt load imbalance will restraint the performance improvement on multi-core platform. To solve this problem, interrupt tasks must be distributed across all CPUs rather than just sending to CPU0. A mechanism called interrupt affinity [15] implemented in the Linux kernel would be used to achieve it. Interrupt affinity defines a CPU bitmask in hexadecimal for every interrupt, and the value of bitmask determines which processor the interrupt should be sent to. Under this architecture, interrupt tasks can be bound to specific processors to obtain the best execution performance and the interrupt loads can be balanced between CPUs.

On the MPCore platform, the realization of interrupt affinity mechanism is dependent on the distributed interrupt controller. According to the implementation rule, each interrupt has a configurable CPU targets table maintained by the interrupt CPU targets registers [16]. The table describes which CPU the interrupt should be sent to. There can be up to 64 interrupt CPU targets registers, and each one has 32 bits. As shown in Fig. 5, each of the registers can be divided into four domains by 8-bit width. Each domain corresponds to an interrupt. Every bit in the domain represents one specific CPU, and its value determined whether this CPU is allowed to receive and handle interrupt. For example, if the value of a domain is set to 0x2, it means the corresponding interrupt will be sent to the CPU1 for processing.

There are two ways to balance the interrupt load by using interrupt affinity mechanism.

A. Statically load balancing

When an interrupt task has fixed load, and there are only limited interrupt tasks to be handled, all the tasks can be assigned to several different CPUs by statically setting the affinity bitmasks. References [10], [17] studied the network performance of SMP under different conditions, and proposed that when setting the affinity bitmasks statically, the network

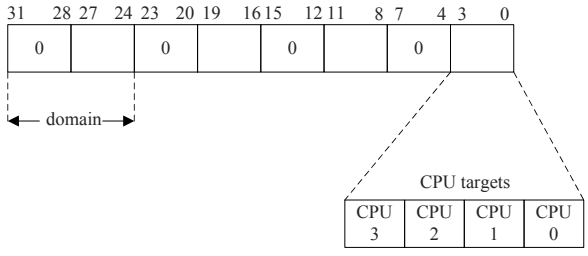

Figure 5. Interrupt CPU target registers

throughput could be received 45% and 25% performance increase respectively.

B. Dynamically load balancing

In order to distribute the interrupt load more evenly across CPUs, dynamically load balancing method should be adopted. There are many commonly used dynamic load balancing algorithms [18]. In this paper, the following actions are taken:

A dedicated kernel thread is created to handle interrupt load balancing. At regular intervals, the thread collects the occurrence frequency of each active interrupt, and compares the load of each CPU to determine whether the system load is in equilibrium. Here, a threshold is defined. Only when the difference between maximum load and minimum load exceeds the threshold, the balancing method should begin to work. In this case, the thread sorts the load of each CPU, as well as the occurrence frequency of each active interrupt, and then binds all the active interrupts to different CPUs. The binding rule is: the interrupt with the highest occurrence frequency will be bound to the CPU with the lowest load, while the interrupt with the second highest occurrence frequency will be bound to the CPU with the second lowest load, etc. The binding operation is accomplished by modifying the affinity bitmask, and the total occurrence frequency of all interrupts on one CPU at a certain interval is chosen as the load indicator.

On the MPCore platform, in order to test the efficiency of the load balancing method, the number of interrupts processed by every CPU is recorded when the system has run for a while. The results are shown in Tab. II. It can be concluded that the dynamically load balancing method is effective in distributing interrupt tasks across several processors and avoiding single processor saturating with heavy loads.

VI. CONCLUSION

Attribute to the introduction of multi-core architecture, the vector floating point unit and the programmed distributed interrupt controller, the performance of the ARM11 MPCore has been significantly improved than the ARM9 processors. Nevertheless, optimization schemes still should be considered to take full advantage of the MPCore.

In this paper, the OpenMP parallel computing model is used to optimize the integer operation, and an efficiency gain of a factor 3.8 is got. Furthermore, by adopting the hardware

TABLE II. ASSIGNMENT OF ACTIVE INTERRUPTS ON EACH CPU

IRQ	CPUs			
	CPU0	*CPU1*	*CPU2*	*CPU3*
33	39697	38135	37724	36987
36	470	1258	771	262
37	12	0	0	0
39	116	83	67	0
40	10	0	0	0
41	3978	2167	3107	1598

floating-point instruction set to optimize the floating-point computation capability, a speedup of a factor 3 to 5 is achieved. Concerning the multimedia processing, two optimization methods are proposed for the ARM11 MPCore platform, i.e., the DCT optimization with VFP and the parallelization of audio decoding, video decoding and playing. The experimental results show that both methods could efficiently improve the performance of multimedia processing on the ARM11 MPCore platform. Finally, by proposing that the traditional interrupt processing mechanism may cause workload imbalance on multi-core platform, interrupt load balancing methods which could efficiently enhance the parallelism performance of the ARM11 MPCore are adopted.

With the efforts above, the performance of the ARM11 MPCore could be fully excavated, and it can be expected that the ARM11 MPCore will have a good application prospect.

ACKNOWLEDGMENT

The work is supported by National Natural Science Foundation of China (Grant No. 60832003). And the authors acknowledge the support of the Samsung (China) research center for providing the RealView emulation baseboard with ARM11 MPCore core tile.

REFERENCES

[1] W. Wolf, "The future of multiprocessor systems-on-chips", in Design Automation Conference, 41st Conference on (DAC'04), 2004, pp. 681-685.

[2] K. Hirata, J. Goodacre, "ARM MPCore: The streamlined and scalable ARM11 processor core", in Design Automation Conference (ASP-DAC'07), January 2007, pp. 747-748.

[3] "ARM11 MPCore processor technical reference manual", ARM Limited, Lit.-Nr.:ARM DDI 0360D, 2006.

[4] "Core tile for ARM11 MPCore user guide", Ref: DUI 0318C, 2006.

[5] H. Blume, J. V. Livonius, L. Rotenberg, T. G. Noll, H. Bothe, J. Brakensiek, "Performance and power analysis of parallelized implementations on an MPCore multiprocessor platform", in International Conference on Embedded Computer Systems, July 2007, pp. 74-81.

[6] J. Marathe, F. Mueller, "Source-code-correlated cache coherence characterization of OpenMP benchmarks", IEEE Transactions on Parallel and Distributed Systems, vol. 18, no. 6, pp. 818-834, June 2007.

[7] R. E. Grant, A. Afsahi, "A comprehensive analysis of OpenMP applications on dual-core Intel Xeon SMPs", in IEEE international Parallel and Distributed Processing Symposium, March 2007, pp. 1-8.

[8] I. Dorta, C. Leon, C. Rodriguez, "A comparison between MPI and OpenMP barnch-and bound skeletons", in Proceedings of the Eighth International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS'03), April 2003, pp. 66-73.

[9] A. Foong, J. Fung, D. Newell, S. Abraham, P. Irelan, "Architectural characterization of processor affinity in network processing", in IEEE International Symposium on Performance Analysis of Systems and Software, March 2005, pp. 207-218.

[10] W. B. Zheng, H. N. Ding, Q. H. Li, D. Y. Zhang, "Research on IP forwarding performance of Linux", Journal of Xi'an Jiao Tong University, vol. 38, no. 2, pp. 124-127, February 2004.

[11] V. Vishwanath, T. Shimizu, M. Takizawa, K. Obana, J. Leigh, "Towards terabit/s systems: performance evaluation of multi-rail systems", High-Speed Networks Workshop, May 2007, pp. 51-5.

[12] Y. Zhou, S. D. Du, "The analysis and optimization on performance of ARM11 MPCore", Journal of Nanjing University (Nature Sciences), vol. 45, no. 1, pp. 5-10, January 2009.

[13] E. Ayguade, N. Copty, A. Duran, J. Hoeflinger, L. Yuan, F. Massaioli, "The design of OpenMP tasks", IEEE Transactions on Parallel and Distributed Systems, vol. 20, no. 3, pp. 404-418, March 2009.

[14] R. Bernardini, G. M. Cortelazzo, G. A. Mian, "A sequential multidimensional Cooley-Tukey algorithm", IEEE Transactions on Signal Processing, vol. 42, no. 9, pp. 2430-2438, September 1994.

[15] W. Wang, S. D. Du, "Research on interrupt affinity based on MPCore and Linux", Journal of Nanjing University (Nature Sciences), vol. 45, no. 1, pp. 24-30, January 2009.

[16] "Iterrupts on MPCore development boards", ARM DAI 0176C, 2006.

[17] A. Foong, J. Fung, D. Newell, "An in-depth analysis of the impact of processor affinity on network performance". in Proceedings of 12th IEEE International Conference on Networks, November 2004, vol. 1, pp. 244-250.

[18] G. Z. Peng, Y. L. Qiu, D. C. Peng, "Several random load balancing algorithms", Computer Engineering, vol. 27, no. 2, pp. 22-24, February 2001.