OpenMP Implementation and Performance on Embedded Renesas M32R Chip Multiprocessor

Yoshihiko Hotta, Mitsuhisa Sato, Yoshihiro Nakajima, Yoshinori Ojima
Graduate School of Systems & Information Engineering,
University of Tsukuba
{hotta,msato,yoshihiro,ojima}@hpcs.is.tsukuba.ac.jp

Abstract

CMP (Chip Multiprocessor) is a promising processor architecture, not only for high performance but also for reducing power and energy consumption in embedded applications. We have implemented an OpenMP compiler for an embedded Renesas M32R chip multiprocessor as a parallel programming environment. In this paper, we report the preliminary performance of OpenMP benchmarks, including scientific and multimedia applications on the M32R CMP. We found that OpenMP allows users to easily obtain reasonable performance improvement using multiple CPUs in the CMP with just a few directives inserted. Also, we discuss the possibility of OpenMP thread run-time scheduling and some compilation techniques for power-aware computing on the CMP.

1. Introduction

Recently, many processor developers have announced a chip multiprocessor (CMP) architecture, especially for embedded applications. ARM offers MPCore [1], which is a CMP with four processors in a single chip. Even in the high-performance computing field, Intel announced that they will change their high-end processor s road map to CMP due to the chip s high power dissipation and low clock frequency [8]. As more transistors areintegrated onto a bigger die, the CMP will become a promising alternative to the complex superscalar microprocessor that is commonly used today. While there seems to be a trend toward CPUs with wider instruction issues as well as support for a larger amount of speculative execution, a CMP composed of simpler processors may exploit thread-level parallelism efficiently. For such CMPs, parallel (or multi-threaded) programming is required. OpenMP is an easy-to-use simple parallel programming environment that makes use of the multiple CPUs in the CMP.

In this paper, we report an implementation of OpenMP and its preliminary performance on a Renesas M32R chip multiprocessor, which is intended for embedded systems. M3T-M32700UT is a system designed to develop several embedded applications using M32R CMP. M3T-M32700UT has several devices, such as Ethernet, LCD, and a video camera, and it runs on the embedded Linux.

One of our research interests is power-aware computing using a CMP. Recently, there has been tremendous interest in power-aware computing for mobile embedded systems such as PDAs and cellular phones. Even in high-performance parallel computing systems, it is very important to reduce the power consumption for cooling and high-density packaging. Our recent research [2] shows that a cluster of low-power processors can obtain better performance than a single high-performance processor in terms of the power-performance ratio. CMP allows us to exploit both high performance and low power since it can achieve high performance while keeping a low clock frequency by having multiple processors on a single chip.

An interesting feature of OpenMP is that the execution of OpenMP programs is independent of the number of actual physical processors. This feature can be a useful tradeoff between application performance and power consumption in the CMP. For example, when an application is required to reduce the power consumption, which may cause the application to run slowly, we can run the OpenMP program at low power by stopping the power supply to some of processors.

The roles of OpenMP for embedded CMP are as follows:

- Providing a portable parallel programming environment. OpenMP provides a standard API in C/C++ and Fortran for a shared memory machine used for embedded applications.
- Providing an easy to use parallel programming model. The original sequential program may be parallelized easily by inserting OpenMP directives. Actually, the POSIX thread library is often used to develop

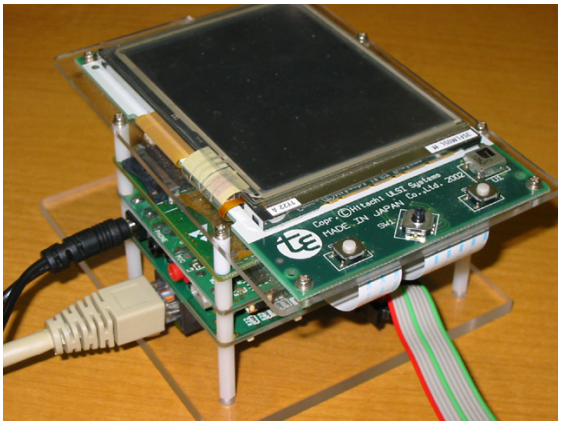

Figure 1. Photo of M3T-M32700UT

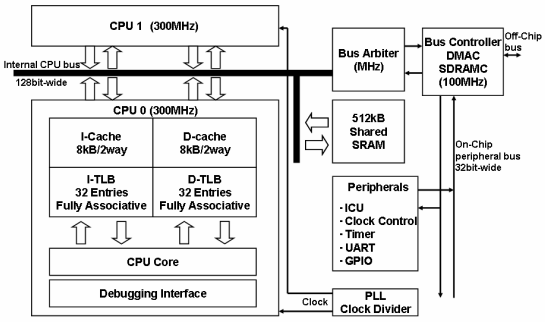

Figure 2. Block Diagram of Renesas M32R CMP (M32700)

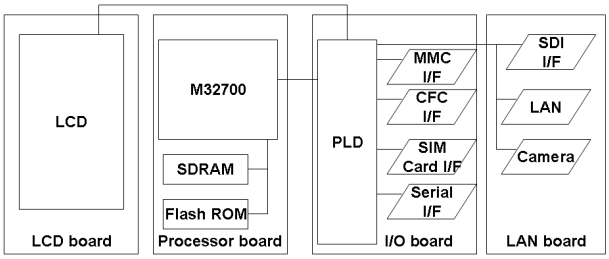

Figure 3. M3T-M32700UT block diagram

feature	spec
CPU	Renesas M32R CMP (M32700)
clock	300MHz (in this paper at 200MHz)
cache I/D	8KB/8KB 2way
Memory	16MB SDRAM / FLASH:4MB
LAN	100Base-TX
USB	Host2.0
module	CF/MMC/eTRON slot, LCD, AR camera
size	60mm * 60mm

Table 1. Specification of M3T32700UT

embedded applications. An OpenMP-style programming model can reduce the cost of parallel programming by eliminating tedious codes for managing and synchronizing threads.

- Exploiting parallelism on embedded multimedia applications. Time-consuming multimedia applications often have much parallelism so that the parallelization makes these applications faster.
- Enabling flexible run-time thread scheduling for real-time and power-aware computing. As described above, sophisticated run-time thread scheduling may allow a tradeoff between application performance and power consumption by controlling the CPU status in the CMP.
- New features of OpenMP will be possible to extend OpenMP for real-time embedded applications.

In this paper, we discuss some techniques for reducing power and energy consumption using OpenMP. Reducing power and energy is important for embedded systems to make a compact package while maintaining performance. The rest of the paper is organized as follows. The specification of our target system M3T-M32700UT is presented in the next section. To implement an OpenMP compiler, we used the Omni OpenMP compiler, which is described in Section 3. Section 4 presents the preliminary performance evaluation using some scientific and multimedia benchmarks. Section 5 discusses the OpenMP issue for power-aware computing in CMP. Finally, we present our concluding remarks and future work in Section 6.

2. M3T-M32700UT: Renesas M32R Chip Multiprocessor System

M3T-M32700UT is a system that consists of an M32R CMP processor [3], LCD, network interface, etc. The M32R processor is a product of Renesas Technology and is a very low-power processor for embedded systems. Table 1 shows the specification of the M32R CMP processor. Figure 1 shows a photo of the M3T-M32700UT.

The M32R32700 single-chip multiprocessor contains two M32R 32-b CPU cores and 512 KB of shared SRAM, and was designed for embedded microcontrollers and a system-on-chip (SoC) core. Since the software for an embedded system has increased in complexity, embedded processors are required to increase performance. At the same time, low power dissipation is still a key feature for battery-operated applications.

This chip is fabricated in a 0.15 μm CMOS process. Figure 2 shows a block diagram of the chip. Two CPU cores and the shared SRAM are connected via a 128-bit bus. The pipeline is a 7-stage pipeline. The CPU core has 2-way set-associated I/D cache memory, and 32-bit full-associated memory. The power consumption is about 800 mW (in the latest M32R processor, CPU clock: 400 MHz, bus clock: 100 MHz) Table 1 shows the M3T-M32700UT specification. This system has an LCD, network interface, camera, etc., for developing applications. As an operating system, Linux/M32R [5] is the ported Linux for the M32R. The Linux/M32R supports both the 2.4 and 2.6 kernel. Several device drivers (e.g., LAN, CF, USB, MTD) have already been ported. This system is designed for development of an embedded application. We used this system for the performance evaluation of OpenMP applications.

3. OpenMP Implementation

To implement OpenMP on the M32R CMP, we used the Omni OpenMP compiler [7]. The Omni OpenMP compiler is a translator that takes OpenMP programs as input to generate a multi-threaded C program with run-time library calls. The generated program is compiled by the native cross compiler and linked with the run-time library. The run-time library includes microtasking and synchronization primitives on top of the different thread libraries, which includes POSIX threads. Since Linux/M32R supports POSIX threads, it was very easy to port the run-time library to the system.

Table 2 shows the overhead of Synchbench using the EPCC microbenchmark. For reference, we also measured the performance on a Pentium-based SMP platform (Pentium III 550 MHz). The overhead of this result, we found dramatically difference between M32R CMP (2 processors) and Pentium III (2 processors). In particularly "CRITICAL" and "LOCK/UNLOCK" spends over few hundreds of times time. On the other hand, we found little difference in "FOR" and "BARRIER" overhead. But in all case, we found the ratio of each processors overhead is large than the difference of processors clock frequency. So, this result means as compared with high performance processor, M32R CMP has large overhead in case of using OpenMP.

processor (2 proc.)	M32R	PenIII	ratio
PARALLEL	843.4	4.7	179.4
FOR	17.7	1.3	13.6
PARALLEL FOR	857.8	5.2	164.9
BARRIER	13.0	0.9	14.4
SINGLE	58.9	1.4	42.0
CRITICAL	313.8	0.7	448.3
LOCK/UNLOCK	313.5	0.7	447.8
ORDERED	10.3	0.4	25.7
ATOMIC	295.4	0.9	328.2
REDUCTION	855.7	6.8	125.8

Table 2. Results of EPCC micro benchmark [micro sec]

	single	2 proc.	speedup
Laplace	109.2	56.7	1.9
IS CLASS S	0.31	0.19	1.63

Table 3. The execution time and speedup of benchmarks [sec]

4. Preliminary Performance Evaluation

For the performance evaluation, we used some benchmarks including Laplace, NPB (NAS parallel benchmarks)3.1 [6] and MediaBench [4]. NPB and Laplace are taken as scientific workloads, MediaBench is a multimedia application. We used an m32r-linux-gcc compiler (gcc3.2 based) as a backend.

4.1 Laplace and NPB IS

The Laplace benchmark is a simple Laplace equation solver with 4-point stencil operation (512*512 and 20 iterations). NPB IS is an integer sort program taken from NPB 3.1. The size of the benchmark is class S. Each benchmark has different characteristics: Laplace is a computation-intensive program and NPB IS is a memory-intensive program. Table 3 shows the result of these benchmarks. We found that performance dramatically improved by using two processors. In the Laplace benchmark, we achieved a speedup that was almost double. In the IS benchmark, however, the speedup was limited to 1.6. Therefore, we found less speedup in the memory-intensive program than in the computation-intensive program.

4.2 MediaBench

For evaluating the performance of multimedia applications, which are the main targets of embedded systems,

function	single	2 proc.	speedup
motion-estimation	1.823	1.009	1.81
predict	0.001	0.001	0.95
dct-type-estimation	0.018	0.019	0.97
transform	0.374	0.018	1.99
putpict	0.231	0.223	1.04
iquant	0.018	0.020	0.92
itransform	0.038	0.050	0.76
calcSNR	0.025	0.026	0.95
Total	2.762	1.55	1.65

Table 4. The execution time of mpeg2encode on Pentium III SMP (550Hz) [sec]

function	single	2 proc.	speedup
motion-estimation	9.03	4.94	1.83
predict	0.12	0.12	1.01
dct-type-estimation	0.46	0.53	0.87
transform	102.62	73.06	1.40
putpict	2.32	3.81	0.62
iquant	0.13	0.13	1.00
itransform	0.25	0.26	0.99
calcSNR	18.61	11.59	1.60
Total	131.25	94.48	1.41

Table 5. result of mpeg2encode

we used MediaBench. MediaBench is a benchmark for embedded multimedia applications. This benchmark contains applications collected from available image processing, communications and DSP applications. It includes MPEG, JPG, GSM, G.721, PGP, and others. While all of MediaBench may be parallelized by OpenMP, we used the MPEG2 encode, which is a very suitable benchmark for evaluating the benefit of OpenMP on M3T-M32700UT. We parallelized the MPEG2 encode by OpenMP. MPEG is the dominant standard for high-quality digital video transmission. The important computing kernel is a discrete cosine transform for coding and an inverse transform for decoding. The two applications used are mpeg2enc and mpeg2dec for encoding and decoding, respectively. In this paper, we showed the performance of mpeg2enc. Figure 4 shows the MPEG2 encoding stages. This benchmark contains motion-estimation, predict, dct-type-estimation, transform, putpict, iquant, itransform, and calcSNR functions. The motion-estimation function is the calculation of a motion vector of the luminance frame in each macroblock. The basic premise of motion estimation is that, in most cases, consecutive video frames will be similar except for the changes induced by objects moving within the frames. predict predicts the motion compensation. dct-type-estimation is the selection DCT mode from a frame or field DCT. transform is the discrete cosine transformation (DCT) using DCT mode. putpict is the quantization of DCTed blocks and output of bit stream code. iquant is the inverse quantization to the quantized blocks. itransform is the inverse DCT to the dequantized blocks for reconstructions. calcSNR is the calculation of the signal to noise Ratio (SNR). The input data size is four frames of 352*240 in MediaBench.

For reference, we also measured the performance on a Pentium-based SMP platform (Pentium III 550 MHz). Table 4 shows the result of mpeg2enc on this system. We found that the motion-estimation took a large part of the execution time, so we parallelized motion-estimation and transform. For example, we show the code transformation on motion-estimation.

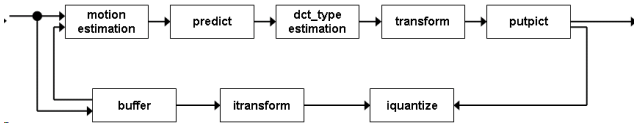

Figure 4. Stages of MPEG2 encode

```
/*loop through all macro-blocks of the picture*/
#pragma omp parallel private(i,j,myk)
{
 #pragma omp for
 for (j=0; j<height2; j+=16)
 {
  for (i=0; i<width; i+=16)
  {
        ... loop body ...
  }
 }
}
```

Table 5 shows the execution time of each function of mpeg2encode on the M32R CMP. The performance is low mainly because of its low clock frequency. In the case of the single processor, the ratio of execution time of motion-estimation, transform, and calcSNR changes compared to that in the Pentium III. This is because the M32R does not have a floating point unit. The execution time of functions that contain many floating point calculation becomes long. In the M32R CMP, we also parallelized calcSNR for performance improvement. For the two-processor execution, we found a performance improvement of the parallel version of about 30% in total execution time. While the performance improvement of the motion-estimation and the calcSNR functions is reasonable, the speedup in the transform function is limited to 1.40. We found that the transform function spends almost as much time as the DCT transformation calculation (over 99%) in a function. we guess that the transform cannot achieve more speedup than the other function because M32R has only L1 small cache.

It should be noted that OpenMP enables us to obtain performance improvement by adding only a small amount of

code. We believe that OpenMP helps the programmer to parallelize the embedded application very easily for an embedded CMP.

5. OpenMP on CMP for Power-Aware Computing

We are interested in power-aware computing using CMP for embedded applications. In this section, we describe several issues regarding compilation technique and OpenMP for power-aware computing.

5.1 Processor run-time scheduling for real-time applications

The typical requirement in real-time applications is to execute a reserved job within a certain period. Although parallel execution runs a program faster than sequential execution does, it is not necessary to execute in parallel with respect to power efficiency. Even though it may take a long time to meet the deadline, the program can run on one CPU with other CPUs in the standby mode. For such run-time thread scheduling, co-scheduling with OpenMP run-time and kernel thread scheduling on the CMP will be required.

In our experiment, we measured the power consumption. The power consumption of the system was 3 W all the time (we measured DC power including CPU, LCD, network and some devices). Unfortunately, we did not find much difference between the OpenMP execution and the sequential execution, because most of the power was used for devices such as the LCD and the network, and the power consumption of the CPUs is only on the order of a few hundred mW.

In addition, there is no kernel support to put the CPU in standby mode. In the near future, we will investigate kernel support and OpenMP thread scheduling for power-aware computing.

5.2 Compiler optimization techniques for low-power computing

Much compiler research has already been done in low-power embedded computing. In embedded applications, not only power consumption but also energy consumption is very important. Power consumption is significant for cooling and packaging, energy consumption is significant for battery-life.

Valluri [10] investigated the impact of using a different level of optimization in the compiler for system power and energy. This research found that most compiler optimizations also benefit the energy by reducing the number of operations, such as dead-code elimination and constant propagation. Higher IPC makes power higher, but the execution time reduces dramatically by compiler optimization. The ratio of execution time reduction is dramatically larger than that of power addition. So we can reduce the energy by compiler optimization. Steinke et al. [9] applied register pipelining to reduce power and energy. Their technique is to replace the memory load with register read (i.e., use cheaper instructions).

According to our previous research, cache optimization such as cache blocking is more effective in reducing the energy in a low-power embedded processor than in a high-performance processor. In a system using embedded processor power consumption in the case of cache hit is smaller than that of cache miss hit. And almost embedded processor has only L1 cache memory, so program can cause cache miss hit and memory bandwidth is small. From that reason, performance is dramatically improved by cache-optimization in embedded processor than conventional high performance processors. While the effect of execution time in case of cache-blocking of high performance processor is about 2 times faster than non-blocking, that of embedded processor is over 9 times faster than non-blocking. As a result, we can reduce energy dissipation dramatically in case of cache-blocking optimization.

So, the cache-blocking technique and memory optimization by the compiler is very effective for power-aware computing. The OpenMP compiler might help code transformations in the above compiler optimization for reducing power consumption.

6. Concluding Remarks and Future Work

We have implemented an OpenMP compiler on an embedded M32R CMP system, and have evaluated its preliminary performance using scientific and multimedia benchmarks. OpenMP is very useful for parallelizing embedded applications as well as scientific applications on the CMP. We found that reasonable improvement can easily be obtained just by inserting OpenMP directives. We have proposed the possibility of OpenMP run-time scheduling for a tradeoff between performance and power in real-time embedded applications in power-aware computing. The most current OpenMP compilers and run-time systems are designed for shared memory multiprocessors. If CMP has a sophisticated mechanism for fast synchronization on a chip, an OpenMP implementation could be redesigned to make use of such fast synchronization and low-latency communications between threads in the run-time system and compiler. Flexible thread run-time scheduling is also an interesting design issue of OpenMP to exploit nested dynamic parallelism using fast synchronization of the CMP. For reducing power consumption, CMP architecture is very effective. In CMP, OpenMP has a large impact for the application engineer, and with power-effective scheduling using

OpenMP, we can greatly reduce power and energy.

Acknowledgment

We would like to thank Renesas Technology Corp. for providing M3T-M32700UT for this research. This research is partly supported by the Grant-in-Aid of Ministry of Education, Culture, Sports, Science and Technology in Japan, No. 14208026, 2002-2004.

References

[1] ARM MPCore. http://www.arm.com/.

[2] Y. Hotta, M. Sato, T. Boku, D. Takahashi, and C. Takahashi. Measurement and chracterization of power consumption of microprocessors for power-aware computing. In *CoolChips7*, April 2004.

[3] S. Kaneko et al. A 600MHz Single-Chip Multiprocessor with 4.8GB/s Internal Shared Pipelined Bus and 512kB Internal Memory. In *2003 International Solid-State Circuits Conference*, volume 1, pages 254–255, September 2003.

[4] C. Lee, M. Potkonjak, and W. H. Mangione-Smith. Mediabench: A tool for evaluating and synthesizing multimedia and communicatons systems. In *International Symposium on Microarchitecture*, pages 330–335, 1997.

[5] Linux/M32R Project. http://www.linux-m32r.org/.

[6] NAS Parallel Benchmarks. http://www.nas.nasa.org/NAS/NPB.

[7] OpenMP Forum. http://www.openmp.org.

[8] R. Sasanka, S. V.Adve, Y.-K. Chen, and E. Debes. The energy efficiency of cmp vs smt for multimedia workloads. In *ISC'04*, July 2004.

[9] S. Steinke, N. Grunwald, L. Wehmeyer, R. Banakar, M. Balakrishnan, and P. Marwedel. Reducing Energy Consumption by Dynamic Copying of Instructions onto Onchip Memory. In *the International Symposium on System Synthesis*, October 2002.

[10] M. Valluri and L. John. Is Compiling for Performance == Compiling for Power? In *The 5th Annual Workshop on Interaction between Compilers and Computer Architectures (INTERACT-5)*, January 2001.