Evaluation of Automatic Power Reduction with OSCAR Compiler on Intel Haswell and ARM Cortex-A9 Multicores

Tomohiro Hirano[1], Hideo Yamamoto[1], Shuhei Iizuka[1], Kohei Muto[1], Takashi Goto[1], Tamami Wake[1], Hiroki Mikami[1], Moriyuki Takamura[2], Keiji Kimura[1], and Hironori Kasahara[1]

[1] Green Computing Systems Research Center
Waseda University - Tokyo, Japan - Tel./Fax. +81-3-3203- 4485/4523
{hirano,shuhei,kmuto,tgoto,waketama,mikami}@kasahara.cs.waseda.ac.jp,
magoroku15@gmail.com,
{keiji,kasahara}@waseda.jp,
http://www.kasahara.cs.waseda.ac.jp/
[2] FUJITSU LABORATORIES LTD.
takamura_moriyuki@v06.itscom.net

Abstract. Reducing power dissipation is one of the most important issues that need to be addressed to improve the performance of all computing systems, such as supercomputers, cloud servers, desktop PCs, medical systems, and wearable devices. Exploiting parallelism and decreasing redundant power dissipation by fine grain power control for multicore/manycore systems are promising approaches, which can ensure continuous performance improvements and reduce power dissipation. However, the manual development of parallelized applications and the embedding of power control code are both time-consuming and error-prone. The OSCAR automatic parallelization compiler has been developed to overcome these problems, which facilitates automatic low-power optimization in addition to parallelization. Though the OSCAR compiler allows these optimization, the suitability of the power optimization method for various platforms is unclear because each architecture has its own power control functionality interface. Therefore, we investigated low-power optimization with the OSCAR compiler on Intel Haswell and ARM multicore platforms to determine the efficiency of the compiler in exploiting the power control functionality of these platforms. The evaluations showed that the power consumption was reduced by 44.2% on the Intel Haswell platform having three-cores with the H.264 decoder and by 68.4% with Optical Flow on three-cores with power control compared with three-cores without power control. On the ARM cortex-A9, having three-cores with power control obtained a power reduction of 57.9% with the H.264 decoder and 67.2% with Optical Flow. These results show that the OSCAR multi-platform API resolves differences between architectures and reduces the power consumption on multiple platforms.

Key words: automatic parallelization, power control, power reduction, multicore processor, multiple platforms

1 Introduction

Multicore processors have been implemented in various computing systems, which range from large servers to small smart-phones and tablets[1]. These processors have been used to obtain higher performance for over a decade. However, the issue of power consumption by computing systems is an increasingly serious problem that affects the overall affordability of computing, because the use of more processor cores yields higher performance but it also increases power usage.

To reduce the power consumption by multicore in smart devices, recent multicore processors such as the Samsung Exynos 5 Octa use the `big.LITTLE` architecture[2], while NVIDIA introduced variable symmetrical multi-processing in Tegra 3. Both approaches use low-power cores in addition to the standard processor cores when performance is less important. In addition, Intel implemented an integrated voltage regulator in Haswell generation processors to facilitate more fine-grained power control[3].

Recent software development environments for multicores support parallelization by parallel application program interface (API) as in OpenMP[4] and CUDA[5] while they still require manual parallelization for application developers. Furthermore, these development environments do not provide interfaces for power control mechanisms implemented in current multicores. This means fine-grained power controls considering synchronization, deadlines, and so on in a parallelized program can not be applied on current multicores.

The combination of the Optimally SCheduled Advanced multiprocessoR (OSCAR) automatic parallelizable compiler and the OSCAR API allows the parallelization of an application program automatically on various platforms[6–8]. In addition to parallelization, the OSCAR compiler also reduces power consumption by inserting power control codes for dynamic voltage and frequency scaling (DVFS), clock gating, and power gating, into the parallelized program. Especially for the case of power control, each architecture has its own control interface. In order to utilize these architecture dependent power control interfaces, the OSCAR compiler generates a parallelized and power optimized program annotated with the OSCAR API directives, which work as interfaces between the OSCAR compiler and various multicores. Then, these directives are translated into runtime library calls for a target architecture. Thus, the OSCAR compiler can provide applicability for various architectures.

In this paper, we show the evaluation of power reduction control using the OSCAR compiler on an Intel Haswell processor for server and desktop computers, and with an ARM Cortex-A9 multicore for smartphones using real-time applications. We also show the OSCAR compiler and the OSCAR API can fully utilize the different power control mechanisms in both processors.

The remainder of this paper is organized as follows. Section 2 provides an overview of the OSCAR compiler and API. Section 3 explains the runtime platform in a current Linux system and the interface used to call clock gating from applications. Section 4 presents the results of the performance evaluation using two different platforms. Finally, our conclusions are given in Section 5.

2 OSCAR Compiler and OSCAR API

In this section, we present an overview of the power reduction scheme provided by the OSCAR automatic parallelization compiler and the OSCAR API.

2.1 Multigrain Parallel Processing with the OSCAR Compiler

The OSCAR compiler exploits multigrain parallelism, which comprises coarse grain task parallelism, loop iteration level parallelism, and statement level near-fine grain parallelism.

To exploit multigrain parallelism, the OSCAR compiler decomposes a sequential C or Fortran program into coarse-grained tasks called macro-tasks(MTs), such as basic blocks, loops, and subroutine calls. The OSCAR compiler analyzes the control flow and the data dependencies among MTs, thereby generating a macro-flow-graph (MFG). Next, the compiler analyzes the earliest executable condition[9] by exploiting the parallelism among MTs by analyzing both the control dependencies and the data dependencies together. The results of the analysis are represented as a macro-task-graph (MTG) for an MFG.

If an MT is a subroutine call or a loop that includes coarse grain task parallelism, the OSCAR compiler hierarchically generates MTs inside the MT. In addition, loop iteration level parallelism is translated into coarse-grained task parallelism by decomposing a loop into multiple loops.

These MTs are assigned to the processor cores using static scheduling or dynamic scheduling, where a dynamic scheduling routine is generated for each source program by the OSCAR compiler[10].

2.2 Low-Power Optimization by the OSCAR Compiler

When the OSCAR compiler applies static scheduling to real-time application programs with deadlines, such as those considered in this work, the compiler tries to apply frequency reduction with voltage scaling to the critical path of the schedule generated to satisfy the deadline[11]. Next, the compiler applies frequency reduction, clock gating, or power gating to the MTs, as well as to the busy wait loops used for synchronization that are not present on the critical path, while considering the overhead of the power state transitions[12].

2.3 OSCAR API

The OSCAR API comprises a set of directives that support power control, DMA transfer, group barriers, and local and distributed shared memory management on various shared memory multiprocessor and multicores for servers, desktop computers, and embedded systems[8].

The OSCAR API uses `section`, `flush` and `threadprivate` directives in OpenMP. In addition to these directives, `distributedshared` and `onchipshared` are added to utilize distributed shared memory and on-chip shared memory while

`threadprivate` is used for local data memory. Furthermore, the OSCAR API employs user-level power control in addition to thread control and memory allocation. The OSCAR compiler generates a parallelized program by inserting these compiler directives.

The API translator translates the directives of the OSCAR API into runtime library calls. The translator was developed specifically for embedded systems that lack OpenMP compilers. In this case, an ordinary sequential compiler such as gcc finally generates the parallelized executable binary.

The OSCAR API provides the `fvcontrol` and `get_fv_status` directives for power control. These `fvcontrol` directives set the power status of a hardware module in a target system to a specified value. `get_fv_status` acquires the current power status from a specified hardware module.

The API translater translates the `fvcontrol` and `get_fv_status` directives into `oscar_fvcontrol()` and `oscar_get_fv_status()` functions, respectively. These functions wrap the operations for the power control interface of the target system.

3 Runtime Support for Power Control

Power consumption of applications can be optimized by the OSCAR compiler in a specific environment by power control mechanisms like DVFS, clock gating, and power gating through the `oscar_fvcontrol()` function.

To fully utilize the power control by the OSCAR compiler, `oscar_fvcontrol()` must be appropriately implemented to support power control mechanisms provided in each target multicore with low overhead. This section provides an overview of the target architectures, the power control frameworks that are available for these architectures, and the interfaces implemented for DVFS and clock gating.

3.1 Power Control Frameworks Available in Linux

We describe the power control frameworks that are currently available in Linux platforms. P-state controls the processor frequency, which corresponds to the workload. The processor frequency is controlled by the P-state driver: cpufreq[14]. The power of the target device is reduced by controlling the frequency and its corresponding voltage.

In an ordinary Linux system, dynamic frequency scaling is achieved using an on-demand governor, which monitors the utilization rate of each core and the upper or lower frequencies are set dynamically when the load exceeds or falls below the thresholds. In addition, the userspace governor described in this paper allows the frequency to be specified by the user's program via cpufreq.

3.2 Intel-Specific Power Control Interface

This section describes the `MWAIT` interface which is implemented in OSCAR runtime. `MWAIT` is an instruction to transit to the C-states[13], where the C-

states are low-power idle states that save power. For example, the C1 state is an auto-halt mode and the C3 state is a deep sleep mode, where numerically higher C-states comprise greater power saving actions, but with higher latency.

In the current Linux implementation for Haswell, the `MWAIT` instructions change the processor power state into a C-state as well as returning to the P-state when there is a change in content of a specific address checked by `MONITOR`.

`MONITOR` and `MWAIT` are available at "Ring 0" and applications cannot use these instructions. Thus, the kernel module is developed to access `MWAIT` and `MONITOR` from user applications via "ioctl" system call.

Figure 1 shows the method for realizing clock gating, or a transition to a C-state, as well as a transition to the P-state from a C-state using the OSCAR API `fvcontrol` directive. If the processor core-0 is clock gated by a directive `fvcontrol(0,CPU,0)`, `fvcontrol` calls a function `slave_to_master(*flag)`. The `slave_to_master(*flag)` function calls a `MWAIT` instruction in the kernel module via "ioctl". The `MWAIT` instruction makes the `MONITOR` to watch a specific address pointed by "flag". Next, `MWAIT` changes the state of the core-0 into a C-state.

When the core-1 changes the state of the core-0 from a C-state into the P-state via a directive `fvcontrol(0,CPU,100)`, the core-1 calls the `master_to_slave(flag)` function. The `master_to_slave(flag)` function changes the content of `flag`. Next, the `MONITOR` on the core-0 detects that the content pointed by `flag` has changed and it changes the power state of the core-0 from the C-state to the previous P-state. Next, the core-1 sets the frequency of the core-0 into 100%, or 3.5[GHz], in the driver `cpufreq`. Thus, `cpufreq` can change the frequency of the core-0 into 3.5[GHz].

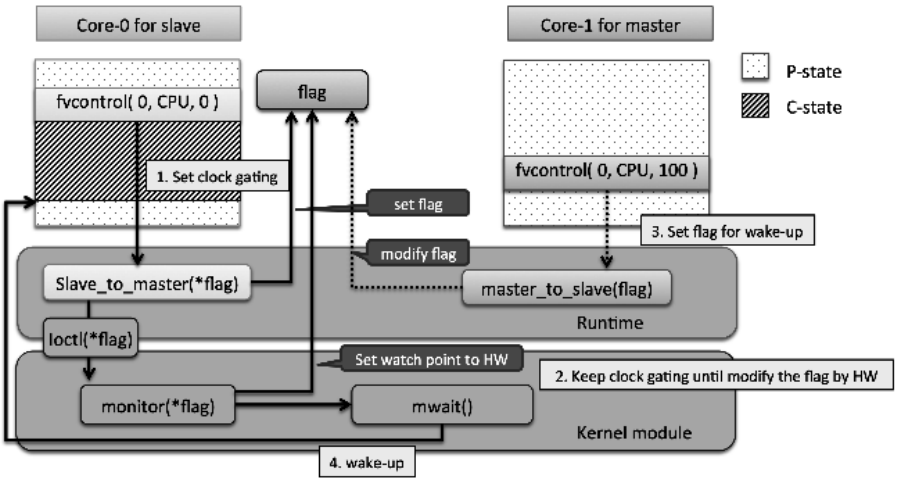

Fig. 1. Procedure for calling the MWAIT flow

Figure 2 shows the effect of clock gating using the `MWAIT` instruction. Figure 2-(a) shows the power for a busy wait loop at 3.5[GHz] in the P-state without clock gating. The average power consumption was 40[W]. Figure 2-(b) shows the same busy wait loop applying clock gating, or the C-state using MWAIT, at 3.5[GHz] in the P-state during about 10,000 clocks. In this case, the average power consumption was reduced to 28[W].

This result confirmed that the OSCAR API `fvcontrol` implemented using `MWAIT` successfully reduced the power with low overheads.

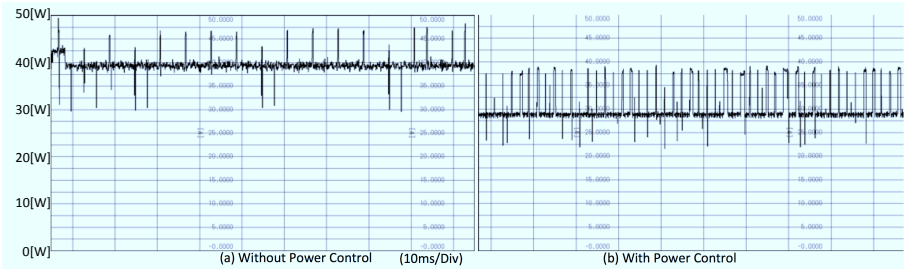

Fig. 2. Performance of clock gating using `MWAIT` at 3.5 GHz

Figure 3 shows the result of the same experiment when the frequency was 800[MHz]. Figure 3-(a) shows that the average power consumption was 7[W] when the frequency was lowest (800[MHz]) without clock gating. Figure 3-(b) shows the power when the C-state and P-state were applied for approximately 10,000 clock periods. The average power consumption was reduced to 6[W].

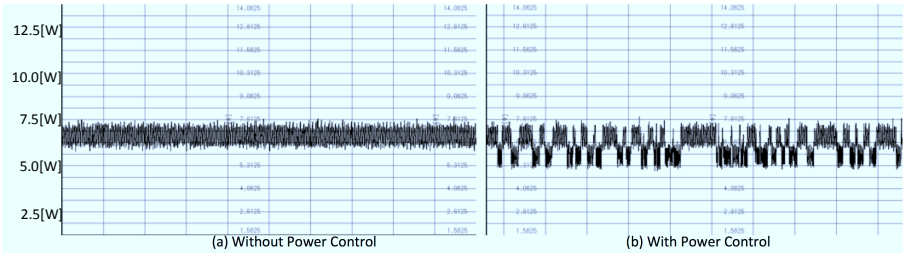

Fig. 3. Performance of clock gating using `MWAIT` at 800 MHz

This result shows that clock gating was still effective even at the lowest frequency and the lowest voltage.

3.3 ARM-Specific Power Control Interface

This section describes the wait-for-interrupt (WFI) instruction in ARM cores for clock-gating used in an optimized program by the OSCAR compiler.

The WFI instruction sends a signal to the processor, which indicates that a timing does not need to be executed. This instruction suspends the execution on the processor core and stops the clock. Specifically, the WFI instruction shuts down any process until an interrupt or a debug event occurs[15]. To utilize the characteristics of WFI as a low power optimization in the runtime library, modifications were made in the Android[16] Linux kernel to allow WFI to stop the clock at an interval of 500[μs][17].

3.4 Examples of Target Architectures

Intel and ARM platforms were prepared to investigate the differences in power consumption between a server and an embedded system. Table 1 shows the detailed configurations of the both platforms, where the power reduction control parameters comprised transitions in the frequency time, the power consumption of each frequency, and the other elements that need to be set in the OSCAR compiler.

Intel(Haswell) The H81M-A ASUS motherboard with an Intel processor Core i7 4770k was used as an example of a server environment, where the DVFS could be controlled independently for each core.

On the Intel platform, three frequency levels were tested in this experiment: full (3500[MHz]), medium (1800[MHz]), and low (800[MHz]), respectively. The frequency transition overheads were about 10,000 cycles. Moreover, each core of the `cpufreq` governor was set to ondemand when the power control was not applied. When the power control was applied, the core 0 was set to ondemand and the core 1-3 were set to the userspace for benchmark applications.

ARM(ODROID-X2) The ODROID-X2 is an evaluation board[18] for the Samsung Exynos4412 Prime chip[19, 20], which comprises the 4-core ARM Cortex-A9. Frequency and voltage scaling could not be used independently in this chip. Thus, the frequencies of all cores were changed together. In this evaluation, the frequencies were tested at three levels: full (1700[MHz]), medium (900[MHz]), and low (400[MHz]), respectively. As same as in the Intel platform, each core of the `cpufreq` governor was set to ondemand when the power control was not applied whereas the userspace was set when the power control was applied.

4 Performance Evaluation with Intel 4-cores and ARM Cortex-A9 4-cores

This section describes the analysis of the power consumption by the Intel 4-cores and the ARM cortex-A9 4-cores. In this evaluation, the benchmark applications

Table 1. Evaluation Environment

Platform	Intel platform	ARM platform
Platform board	H81M-A	ODROID-X2
CPU	Intel Core i7 4770k	Samsung Exynos4412 Prime
Number of cores	4	4
Maximum clock frequency	3.5 [GHz]	1.7 [GHz]
L1 Cache	I/D cache 32/32[KB/core]	I/D cache 32/32[KB/core]
L2 Cache	unified 256[KB/core]	1[MB/chip]
L3 Cache	8 [MB/chip]	N/A
DDR	16[GB]	2[GB]

described in Section 4.2 were parallelized and the power was controlled by the OSCAR compiler and the OSCAR API.

4.1 Modification of the Evaluation Boards to Measure the Chip Power Consumption

In this evaluation, the boards were modified to measure the power of the processor chips directly. In particular, a 5[mΩ] shunt resistor was attached between the power source circuit of the cores and the Power Management IC[21] on H81M-A, and a 40[mΩ] shunt resistor on ODROID-X2. Moreover, general purpose IO pins[17, 22] were used to measure the power consumption of a specific program section.

4.2 Application Programs used in the Evaluation

This section describes the specifications of the two real-time applications used in the power evaluations.

H.264 H.264 is a video compression format. The JM version module was originally developed as ISO/IEC 14496-10 for reference purposes[23, 24]. On the Intel platform, the deadline for H.264 was set to 30[fps] (33[ms] per frame) and the input file was HD720p (1280 $\times$ 720 pixels). On the ARM platform, the deadline for H.264 was set to 30[fps] (33[ms] per frame) and the input size was 256 $\times$ 128 pixels.

Optical Flow The Optical Flow is a benchmark application, which is used as a reference in OpenCV[25]. This real-time application tracks 16x16 blocks between two images by calculating the velocity fields. On the Intel platform, the deadline for Optical Flow was set to 15[fps] (66[ms] per frame) and the Input file was HD720p. On the ARM platform, the deadline for Optical Flow was set to 30[fps] (33[ms] per frame) and input frame size was 256 $\times$ 128 pixels.

4.3 Power Consumption on the Intel Platform

This section describes the power consumption by the Intel platform when Optical Flow and H.264 were executed on the Intel platform.

Figure 4 shows the power consumption of Optical Flow and H.264 with different numbers of processor cores. In Figure4, when the power control by the OSCAR compiler was applied, the power consumption was decreased along with the increasing number of cores for both applications.

For example, in the case of H.264, the power was 17.37[W] for one-core, 16.15[W] for two-cores, and 12.50[W] for three-cores, respectively. Comparing three-cores with one-core, 28.04% was reduced by the power optimization and the parallelization by the OSCAR compiler. On the other hand, the power consumption was increased along with the increasing number of cores without power control. For example, also in the case of H.264, the power was 29.67[W] for one-core, 37.11[W] for two-cores, and 41.81[W] for three-cores, respectively. Thus, the power consumption by three-cores became 40.92% higher than the case for one-core.

Similarly, in the case of Optical Flow, when the power optimization by the compiler is applied, the power for three-cores became 60.28% lower than the case of one-core, such as 9.6[W] for three-cores and 24.17[W] for one-core, respectively. By contrast, the power for three-cores became 41.96% higher than the case for one-core, such as 41.58[W] for three-cores and 29.29[W] for the case of one-core, respectively.

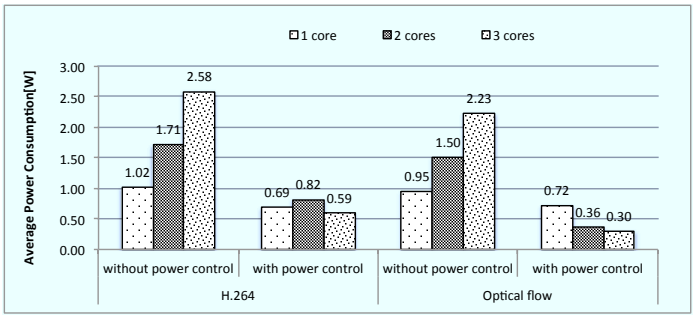

Fig. 4. The power consumption on the Intel CPU platform

Then, comparing the case of power control on three-cores with that of not using power control, 70.10% power reduction for H.264 and 76.91% power reduction for Optical Flow were achieved, respectively. Similarly, comparing with the case of three-cores with power control and that of one-core without power control, 57.87% power reduction for H.264 and 67.22% power reduction for Optical flow were achieved, respectively.

Figure 5-(a) and Figure 5-(b) show the power wave for H.264 using one-core on the Intel platform with and without power control, respectively. Similarly, Figure 6-(a) and Figure 6-(b) show those for Optical Flow.

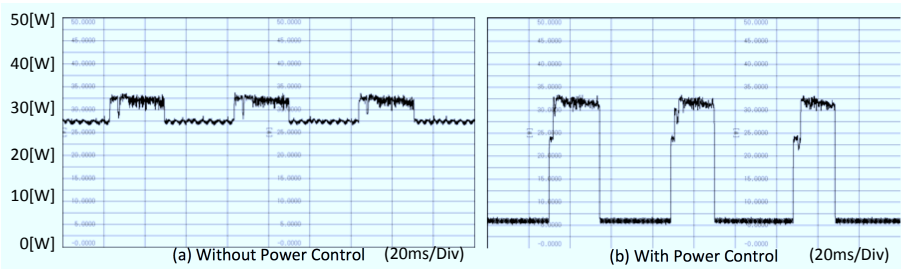

Fig. 5. Power wave of H.264 with one-core on Intel

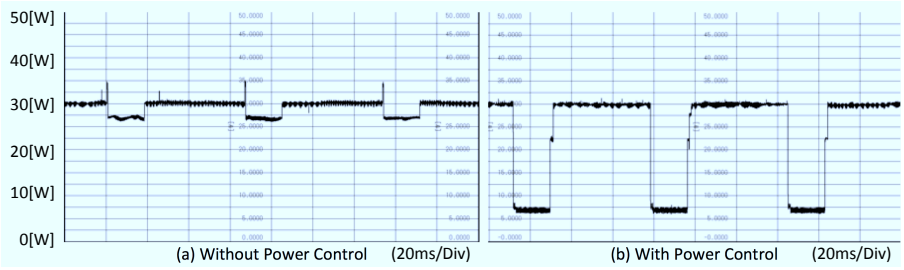

Fig. 6. Power wave of Optical Flow with one-core on Intel

Comparing these figures, the peak power of using power control and that of not using power control were almost same, which indicates both cases were driven at the highest frequency and voltage. However, the bottom power of using power control was about 5[W] while that of not using power control was 30-33[W]. This shows the power control by the OSCAR compiler is appropriately applied by using the `MWAIT` instruction.

Finally, Figure 7-(a) and Figure 7-(b) show the power wave for H.264 using three-cores on the Intel platform with and without power control, and Figure 8-(a) and Figure 8-(b) show those for Optical Flow, respectively.

Figure 7-(b) and Figure 8-(b) show the compiler tried to set the lowest clock frequency as long as possible at the calculation phase of the application, then applied clock-gating until a deadline. On the other hand, as shown in Figure 7-(a) and Figure 8-(a), the clock frequency was always high even at the waiting time for a deadline as shown at the flat line in the wave form.

These results show, in addition to parallelization, DVFS and clock gating realized by the `MWAIT` instruction, which are controlled by the OSCAR compiler, can sufficiently reduced average power consumption.

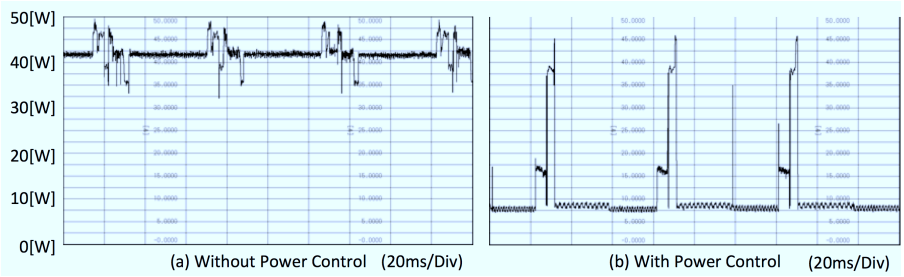

Fig. 7. Power wave of H.264 with three-cores on Intel

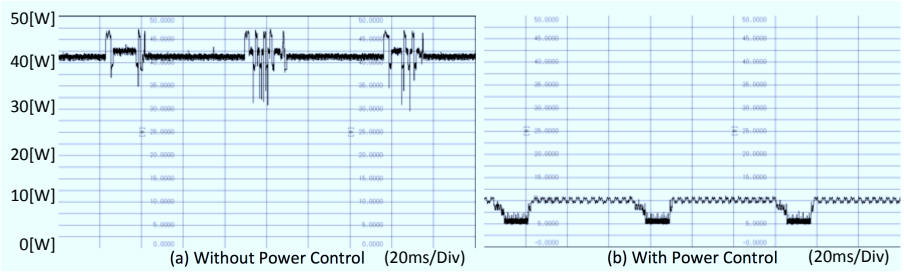

Fig. 8. Power wave of Optical Flow with three-cores on Intel

4.4 Power Consumption on the ARM Cortex-A9 multicore

This section describes the power consumption when Optical Flow and H.264 were executed on the ARM platform.

Figure 9 shows the power consumption for Optical Flow and H.264 with different numbers of cores. This Figure shows the almost same characteristics as in the Intel platform: When the power control by the OSCAR compiler was applied, the power consumption of three-cores was lower than that of one-core for both applications. Also, without the power control, the power consumption was increased along with the increasing number of cores.

For example, in the case of H.264, the power was 0.69[W] for one-core and 0.59[W] for three-cores, respectively. Comparing three-cores with one-core, 14.49%

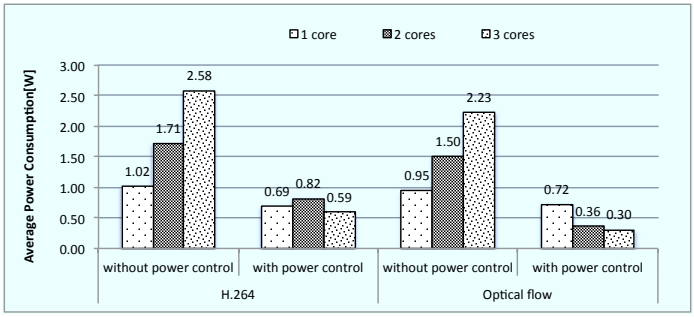

Fig. 9. The power consumption on the ARM CPU platform

was reduced by the OSCAR compiler. On the other hand, the power consumption by three-cores became 2.53 times higher than the case for one-core without power control.

Similarly, in the case of Optical Flow, when the power optimization by the compiler is applied, the power for three-cores became 58.33% lower than the case of one-core. By contrast, the power for three-cores became 2.35 times higher than the case for one-core.

For the cases of three-cores, the power control achieved 77.13% power reduction for H.264 and 86.55% power reduction for Optical Flow compared with the no-power control, respectively. Similarly, comparing with the case of three-cores with power control and that of one-core without power control, 42.16% power reduction for H.264 and 68.42% power reduction for Optical flow were achieved, respectively.

Figure 10-(a) and Figure 10-(b) show the power wave for H.264 using one-core on the ARM platform with and without power control, respectively. Similarly, Figure 11-(a) and Figure 11-(b) show those for Optical Flow. Also, Figure 12-(a) and Figure 12-(b) show the power wave for H.264 using three-cores with and without power control, and Figure 13-(a) and Figure 13-(b) show those for Optical Flow, respectively. These figures also show the almost same characteristics as in the Intel platform.

For example, comparing Figure 10-(a) and Figure 10-(b), the peak power of using power control and that of not using power control are still almost same. However, the bottom power of using power control is almost 0.0[W] while that of not using power control is 0.8[W]. The WFI instruction in the ARM core efficiently reduces the power consumption. Figure 12-(b) and Figure 13-(b) also show the compiler tried to set the lowest clock frequency as long as possible and applied clock-gating until a deadline. By contrast, the clock frequency is still high at the waiting time for a deadline.

In summary, the automatic parallelization and power optimization can efficiently reduce the power consumption on both of the Intel platform and the ARM platform in a uniform manner. The appropriately implemented runtime systems using `MWAIT` for the Intel platform and WFI for the ARM platform col-

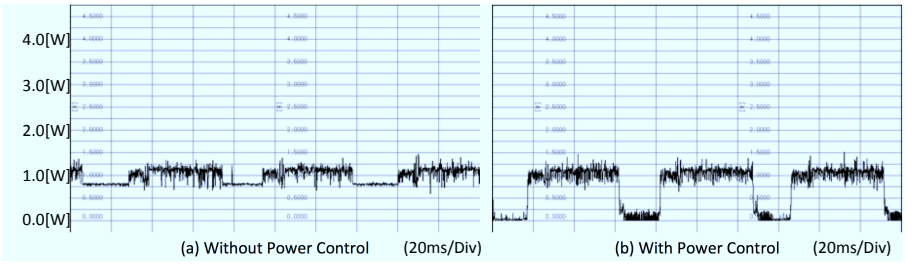

Fig. 10. Power wave of H.264 with one-core on ARM

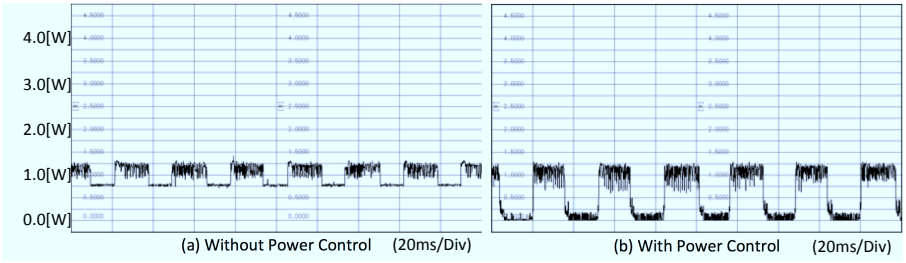

Fig. 11. Power wave of Optical Flow with one-core on ARM

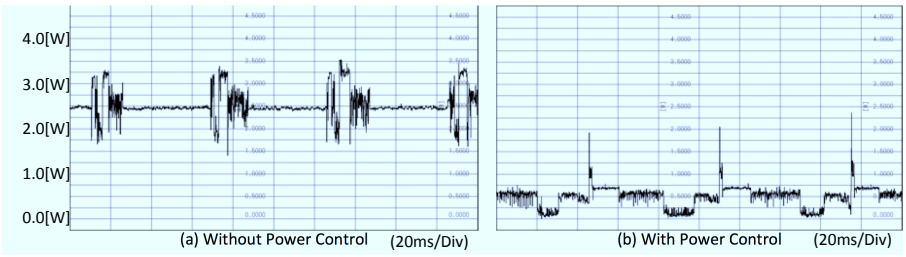

Fig. 12. Power wave of H.264 with three-cores on ARM

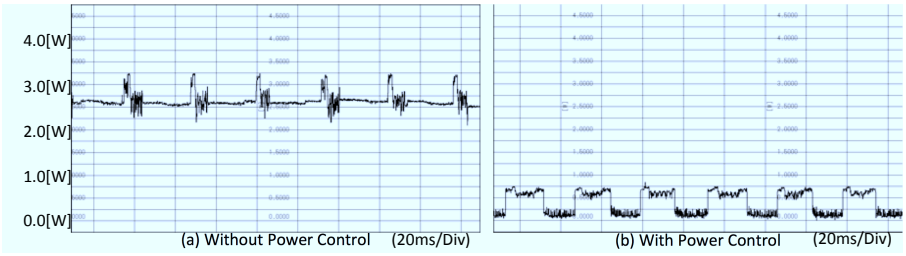

Fig. 13. Power wave of Optical Flow with three-cores on ARM

laborated with the OSCAR compiler and the OSCAR API realize these power efficient computer systems.

5 Conclusion

This paper shows the power reduction on the Intel Haswell platform using the `MONITOR` and `MWAIT` instructions, as well as on the ARM Cortex-A9 platform with WFI using the OSCAR compiler. The power consumption increased gradually without using power reduction control as the number of processor cores increased. By contrast, the power consumption decreased gradually with power reduction control by the OSCAR compiler as the number of cores increased.

The evaluation result shows the power used by Optical Flow without power control increased to 29.3[W] with one-core, 36.6[W] with two-cores and 41.6[W] with three-cores on the Haswell platform. By contrast, with power control, the power decreased to 24.2[W] with one-core, 12.2[W] with two-cores and 9.6[W] with three-cores. In particular, the decrease to 9.6[W] using three-cores with power control represented a reduction of 67.2% compared with 29.3[W] using ordinary one-core execution without power control.

In addition, on the ARM Cortex-A9 four-core, the power used by Optical Flow without power control increased to 1.0[W] with one-core, 1.5[W] with two-cores and 2.2[W] with three-cores. By contrast, with power control, the power decreased to 0.7[W] with one-core, 0.4[W] with two-cores and 0.3[W] with three-cores. In particular, the decrease to 0.3[W] using three-cores with power control represented a reduction of 68.4% compared with one-core without power control, that is 1.0[W].

The results of this performance evaluation show clearly that the OSCAR compiler significantly reduced the power consumption of real-time applications such as H.264 and Optical Flow both on Intel Haswell and ARM Cortex-A9 multicores with a uniform manner.

References

1. NVIDIA Corporation: White paper NVIDIA Tegra: Multi-processor Architecture. (2010)
2. ARM:Jeff, B.: Advances in big.LITTLE Technology for Power and Energy Savings. write paper, 1–11 (2012)
3. Kurd, N., Chowdhury, M., Burton, E., Thomas, T.P., Mozak, C., Boswell, B., Lal, M., Deval, A., Douglas, J., Elassal, M., Nalamalpu, A., Wilson, T.M., Merten, M., Chennupaty, S., Gomes, W., Kumar, R.: Haswell: A family of IA 22nm processors. Solid-State Circuits Conference Digest of Technical Papers, 112–113 (2014)
4. OpenMP, `http://openmp.org/`
5. Cuda, `http://www.nvidia.com/object/cuda_home_new.html`
6. Kasahara, H., Obata, M., Ishizaka, K.: Automatic coarse grain task parallel processing on smp using openmp. Workship on Languages and Compilers for Parallel Computing, pp. 1–15 (2001)

7. Obata, M., Shirako, J., Kaminaga, H., Ishizaka, K., Kasahara, H.: Hierarchical Parallelism Control for Multigrain Parallel Processing. Lecture Notes in Computer Science, vol. 2481, pp. 31–44 (2005)
8. Kimura, K., Mase, M., Mikami, H., Miyamoto, T., Shirako, J., Kasahara, H.: OSCAR API for Real-time Low-Power Multicores and Its Performance on Multicores and SMP Servers. Lecture Notes in Computer Science, vol. 5898, pp. 188–202. (2010)
9. Honda, H., Kasahara, H.: Coarse Grain Parallelism Detection Scheme of a Fortran Program. Systems and Computers in Japan, vol. 22, pp. 24–36. (1991)
10. Obata, M., Shirako, J., Kaminaga, H.: Hierarchical parallelism control for multigrain parallel processing. Lecture Notes in Computer Science, vol. 2481, pp. 31–44. (2005)
11. Shirako, J., Oshiyama, N., Wada, Y., Shikano, H., Kimura, K., Kasahara, H.: Compiler Control Power Saving Scheme for Multi Core Processors. Lecture Notes in Computer Science, vol. 4339, pp. 362–376. (2007)
12. Shirako, J., Yoshida, M., Oshiyama, N., Wada, Y., Nakano, H., Shikano, H., Kimura, K., Kasahara, H.: Performance Evaluation of Compiler Controlled Power Saving Scheme. Lecture Notes in Computer Science, vol. 4759, pp. 480–493. (2007)
13. Intel: Mobile 4th Generation Intel Core Processor Family, Mobile Intel Pentium Processor Family, and Mobile Intel Celeron Processor Family. Datasheet - Volume 1 of 2 (2014)
14. CPU hotplug Support in Linux(tm) Kernel, `https://www.kernel.org/doc/Documentation/cpu-hotplug.txt`
15. ARM Corporation: Cortex-A9 Technical Reference Manual, `http://infocenter.arm.com/help/topic/com.arm.doc.ddi0388i/DDI0388I\_cortex\_a9\_r4p1\_trm.pdf`
16. Google: Android Developers, `http://developer.android.com/index.html`
17. Yamamoto, H., Hirano, T., Muto, k., Muto, k., Goto, T., Mikami, H., Takamura, M., Kimura, K., Kasahara, H.: OSCAR Compiler Controlled Multicore Power Reduction on Android Platform. LCPC (2013)
18. ODROID-X2, `http://www.hardkernel.com/renewal\_2011/products/prdt\_info.php?g\_code=G135235611947`
19. Samsung Electronics: White Paper of Exynos 5. (2011)
20. Samsung Electronics: Samsung Exynos 4 Quad (Exynos 4412) RISC Microprocessor User's Manual. (2012)
21. Samsung Semiconductors Global Site, `https://www.samsung.com/global/business/semiconductor/product/poweric/overview`
22. GPIO Interfaces, `https://www.kernel.org/doc/Documentation/gpio.txt`
23. H.264, `http://iphome.hhi.de/suehring/tml/`
24. Lee, C., Potkonjak, M., Mangione-Smith, W.: MediaBench : A Tool for Evaluating and Synthesizing Multimedia and Communications Systems. Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture, pp. 330–335. (1997)
25. Opencv `http://www.opencv.org`