OSCAR Compiler Controlled Multicore Power Reduction on Android Platform

Hideo Yamamoto[1], Tomohiro Hirano[1], Kohei Muto[1], Hiroki Mikami[1], Takashi Goto[1], Dominic Hillenbrand[1], Moriyuki Takamura[2], Keiji Kimura[1], and Hironori Kasahara[1]

[1] Green Computing Systems Research and Department Center
Waseda University Tokyo, Japan
{hideo,hirano,kmuto,mikami,tgoto,dominic}@kasahara.cs.waseda.ac.jp,
{kimura,Kasahara}@kasahara.cs.waseda.ac.jp,
http://www.kasahara.elec.waseda.ac.jp/
[2] Fujitsu Laboratories Ltd.
takamura.moriyu@jp.fujitsu.com

Abstract. In recent years, smart devices are transitioning from single core processors to multicore processors to satisfy the growing demands of higher performance and lower power consumption. However, power consumption of multicore processors is increasing, as usage of smart devices become more intense. This situation is one of the most fundamental and important obstacle that the mobile device industries face, to extend the battery life of smart devices. This paper evaluates the power reduction control by the OSCAR Automatic Parallelizing Compiler on an Android platform with the newly developed precise power measurement environment on the ODROID-X2, a development platform with the Samsung Exynos4412 Prime, which consists of 4 ARM Cortex-A9 cores. The OSCAR Compiler enables automatic exploitation of multigrain parallelism within a sequential program, and automatically generates a parallelized code with the OSCAR Multi-Platform API power reduction directives for the purpose of DVFS (Dynamic Voltage and Frequency Scaling), clock gating, and power gating. The paper also introduces a newly developed micro second order pseudo clock gating method to reduce power consumption using WFI (Wait For Interrupt). By inserting GPIO (General Purpose Input Output) control functions into programs, signals appear on the power waveform indicating the point of where the GPIO control was inserted and provides a precise power measurement of the specified program area. The results of the power evaluation for real-time Mpeg2 Decoder show 86.7% power reduction, namely from 2.79[W] to 0.37[W] and for real-time Optical Flow show 86.5% power reduction, namely from 2.23[W] to 0.36[W] on 3 core execution.

Key words: smart device, automatic parallelization, API, power control, power reduction, multicore processor, Android, WFI

1 Introduction

Multicore processors have been attracting much attention and applied into a wide variety of systems, such as personal computers, high performance computers, cloud servers and even embedded systems including smartphones, tablets and automobiles[1–3]. In recent years, smart devices such as smartphones and tablets have already been transitioning from single core processors to multicore processors to satisfy the growing demands of higher performance and lower power consumption. However, power consumption of multicore processors on the smart devices is increasing, as usage of these devices become more intense. This situation is one of the most fundamental and important obstacle that the mobile device industries face. Extending the battery life is a crucial problem for current smart devices. To avoid the increasing power consumption, low power architectures like big.LITTLE[4] from ARM[5] have been introduced in the mobile device industries. Some of the multicore processors that apply these architectures are the NVIDIA Tegra3[6] and the Samsung Exynos 5 Octa[7].

Although recent smart devices apply multicore processors in an attempt to gain higher performance and lower power consumption, the anticipated results require further advancement of cooperative hardware-software environment. To realize such an environment, parallelization of software is crucial to fully utilize the capability and potential of multicore processors. Current methods of parallelization include OpenMP and MPI; however, manual optimization of software lowers productivity and become extremely difficult when complexity of software heightens. In order to ease software optimizations for multicore processors, automatic parallelization compilers are needed. Previous and current works of compilers include the SUIF Compiler[8], Polaris Compiler[9], PLUTO[10], and the OSCAR Automatic Parallelizing Compiler[11, 12]. Especially in the works of the OSCAR Compiler, it has realized an automatic power reduction scheme using DVFS, clock gating, and power gating[13, 14]. The significance of this compiler lies in the fact that it can both automatically parallelize an application and control power at the same time. As for power reduction, other works propose a compile-time static approach using detailed information of the program behavior from compiler analysis[15–18]. Moreover, dynamic compiler approaches using the information obtained at runtime and compile-time, have also been proposed on a single processor execution[18, 19].

This paper evaluates the power reduction control by the OSCAR Automatic Parallelizing Compiler on ODROID-X2[20], an Android[21] development platform using real-time applications. Furthermore, by using WFI (Wait For Interrupt) instructions, a pseudo clock gating method was developed, which enables clock gating at an 500[us] interval. Compared to the power control of current Android platforms, this method proves higher power reduction. To attain precise power measurements, a new power measurement method was developed utilizing the GPIO. This proposed method allows synchronization between the program and the waveforms in the power measurements, which no other work has done before to the best of the author's knowledge.

This paper gives an overview of the current power control on Android platforms in Section 2, the methodology in Section 3, an overview of the evaluation environment Section 4, evaluation results in Section 5, and the conclusion of this paper in Section 6.

2 Power Control on Current Android Platforms

This section provides an overview of the current power control on Android platforms. The base of Android is made of Linux, and power control on Android is realized through `cpufreq`, `cpuidle`, and `hotplug`[22].

CPUFreq. The `cpufreq` architecture allows frequency scaling of a target CPU and is a basic driver installed in the Linux kernel. Controlling of frequency and its corresponding voltage results in lower power of the target device. On the Android device, dynamic frequency scaling is realized by using the `ondemand` governor. This governor monitors the current usage on each core at certain time intervals. When the load exceeds or falls below the threshold, frequencies are made higher or lower dynamically.

CPUIdle. Many CPUs on Android devices support multiple idle levels, which are differentiated by power consumption and the exit latencies from that idle level. The `cpuidle` manages the level of idle on each core of the CPU and realizes low power on the device. Linux determines to go idle when no processes are there to execute. The levels of idle state is determined by the number of function units that go to sleep on the CPU. Power consumption is very low when many function units go to sleep, but returning from sleep takes much time. On the other hand, when small amounts of function units go to sleep, power consumption is not lessened much, but returning from sleep is very quick. When the idle state continues for a certain time period, the depth of the idle state goes deeper by default.

HotPlug. The `hotplug` is an extended function of `cpufreq`, which was developed specifically for the power control of multicore processors. When `cpufreq` sets a core to the maximum frequency that runs for a certain period of time, the `hotplug` adds another core to distribute the load. Similarly, when `cpufreq` sets a core to the minimum frequency and the load stays low, the `hotplug` shuts down excess cores to reduce power consumption.

However, utilizing these assets as power control inside applications takes some to some tens of milliseconds, which is not suited for fine power control by a compiler.

3 Power Reduction Control by the OSCAR Compiler

This section provides an overview of the power reduction scheme realized in the OSCAR Automatic Parallelizing Compiler and the OSCAR API. Furthermore, an explanation of the pseudo clock gating method controlled by the OSCAR Compiler and the precise power measurement method will be given.

3.1 Multigrain Parallel Processing and Low Power Optimization by the OSCAR Compiler

The OSCAR (Optimally Scheduled Advanced multiprocessor) Compiler exploits multigrain parallelism, which consists of coarse grain task parallelism, loop iteration level parallelism, and statement level near-fine grain parallelism. In order to exploit multigrain parallelism, OSCAR compiler first decomposes a sequential C or Fortran program into coarse grain tasks named macro tasks(MTs), such as basic block(BB), loop(RB), and subroutine call(SB). Using these MTs, the OSCAR compiler would then analyze both the control flow and the data dependencies among them, creating a macro-flow-graph (MFG). After creating the MFG, the compiler applies the earliest executable condition analysis[23], which can exploit parallelism among MTs associated with both the control dependencies and the data dependencies. The analysis result is represented as a macro-task-graph (MTG).

If a MT is a subroutine call or a loop that has coarse grain task parallelism, the OSCAR compiler hierarchically generates inner MTs inside that MT. Also, loop iteration level parallelism is translated into coarse grain task parallelism by loop decomposition.

These MTs are assigned to the processor cores, which is grouped into processor groups (PG) logically and hierarchically considering the parallelism in each layer of the hierarchical MTG. If the MTG fluctuates at runtime or has conditional branches, dynamic scheduling is applied. Otherwise, static scheduling is applied to the MTG[24].

If there are idle or busy-waiting periods between MTs in a statically scheduled MTG, the compiler tries to minimize total power dissipation by prolonging the execution time of MTs with DVFS or applying clock gating and power gating during the idle periods. This execution mode is named as the fastest execution mode[14]. Note that the OSCAR compiler carefully controls DVFS, clock gating and power gating not to prolong the program execution time in the case of the fastest execution mode.

Similarly, if the deadline of an MTG is given and there are sufficient idle periods until the deadline, the compiler also applies DVFS over MTs on the critical path and applies its clock gating and power gating over idle periods not on the critical path, so that total energy consumption can become as little as possible. This execution mode is named as the realtime execution mode[14].

For example, a power-optimized MTG with a deadline is processed iteratively as in the case of a movie player, this execution mode is called as real-time

execution mode. The experimental evaluations in this paper use the real-time execution mode.

3.2 OSCAR Application Programming Interface

The OSCAR API (Application Programming Interface) is a parallel API for executing the optimized code generated by the OSCAR compiler on various shared memory multiprocessor and multicore systems, including server, desktop computers and embedded systems[13].

The OSCAR API consists of a set of compiler directives based on a subset of OpenMP. The OSCAR API employs user-level power control in addition to thread creation and memory allocation considering local memory and distributed shared memory. The OSCAR compiler generates a parallelized program by inserting these compiler directives. Then, an OpenMP compiler compile this parallelized program into executable binary in the case of server platforms.

The standard API translator, which translates directives of OSCAR API into runtime library calls, has been also developed especially for embedded systems. In this case, an ordinary sequential compiler like gcc finally generates the parallelized executable binary for the target system.

For power control, the OSCAR API provides `fvcontrol` and `get_fv_status` directives. The `fvcontrol` directive sets the power status of a hardware module in a target system to a specified value. The `get_fv_status` acquires the current power status from a specified hardware module.

The power status notation used in these directives is an integer value ranging from `-1` to `100`. The value from `0` to `100` represents the percentage of clock frequency of the specified hardware module. For example, `0` represents clock gating, `100` is the maximum clock frequency, and `50` is half of the maximum clock frequency. In addition, `-1` denotes power gating.

The standard API translator translates `fvcontrol` and `get_fv_status` directives into `oscar_fvcontrol()` and `oscar_get_fv_status()` functions, respectively. These functions wrap the runtime library calls for the target system.

3.3 Pseudo Clock Gating Method Using WFI

This section explains the newly developed pseudo clock gating method that has been implemented in `oscar_fvcontrol` for the OSCAR API.

The current power control method that can be utilized on consumer Android devices is the `cpufreq`. However, this method require millisecond level latency, which prevents high power reduction using the OSCAR compiler. In order to reduce as much power consumption as possible on the Android platform, a new clock gating method was developed. This method utilizes the WFI (Wait For Interrupt) instruction, which is supported by the ARM architecture. The WFI instruction gives a signal to the processor as a hint that there is no process to be executed. This instruction suspends the execution on the processor core and stops the clock. Specifically, the WFI instruction shuts down any instruction

issue of a new process until an interrupt or a debug event occurs[5]. To utilize the characteristics of WFI as a low power optimization, additional functions were inserted in the Linux Kernel to enable the WFI instruction to be issued directly from the applications. Furthermore, this clock gating method is able to stop the clock at a 500[us] interval.

Figure 1-(a) shows the measurements of the electric current on each number of cores without the implementation of the pseudo clock gating method. The graph shows that as number of cores in usage increase, the electric current consumed on the board increases from 500[mA] for 1 core to 2000[mA] for 4 cores. However, by implementing the pseudo clock gating method, as shown in Figure 1-(b), the electric current stays just below 500[mA] even if the number of cores in usage increase. Figure 1-(b) also shows that the proposed method stops the clock at an interval of 500[us]. Compared to the power gating method using `cpufreq`, which requires over 10[ms], the new pseudo clock gating method is higher in precision and speed. By implementing this new method into the runtime library, higher power reduction can be obtained on an Android platform by the OSCAR Compiler.

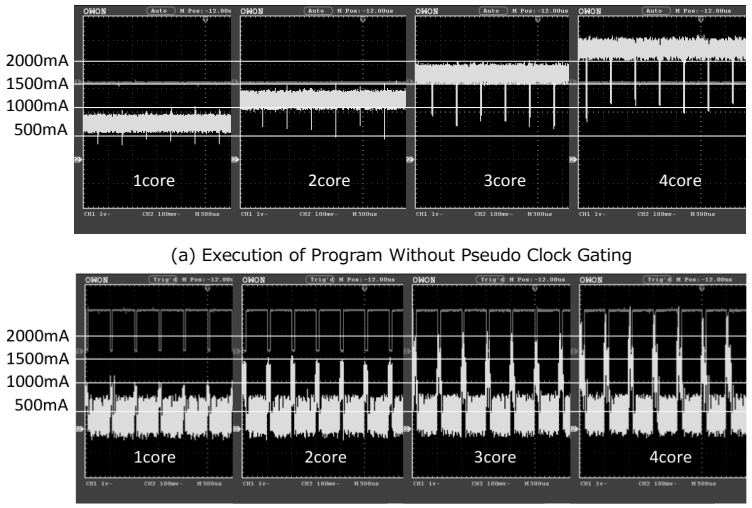

Fig. 1. Comparison of Power Waveform With/Without Pseudo Clock Gating

3.4 Precise Power Measurement Method Using GPIO

This section explains the development of the precise power measurement method using the GPIO (General Purpose Input Output)[25] pins on the ODROID-X2. GPIO pins on chips are usually used for debugging or testing on embedded

systems by inputting commands and triggering interrupts. The Linux kernel driver can control these GPIO pins. A state of a GPIO pin can be seen as an event rising up and down on the voltage measurements.

A GPIO control function is prepared to change the state of GPIO from user applicatoins. The GPIO control function changes the state of the GPIO by setting `1` or `0` as a parameter. Figure 2 shows how the GPIO is utilized on the power measurements. Fig. 2-(a) shows a program example of inserting GPIO control functions inside the MPEG2 Decoder program. Function `gpio_value` changes the state of GPIO from the application. Variable `gpio` specifies the GPIO pin number and the second argument specifies the value to write into the GPIO register. Fig. 2-(b) shows the waveforms of the GPIO and the MPEG2 Decoder in execution. The top voltage waveform shows the output state of the GPIO. Similarly, the bottom power waveform shows the power consumption of MPEG2 Decoder. This figure shows how precise and efficient the usage of GPIO is in power measurements. The marks on the voltage waveform, (i) and (ii), corresponds to the exact location of where the GPIO control functions were processed. On the other hand, the precise power consumption can be measure by expliciting the waveform for MPEG2 Decoder between GPIO signals rising up and down. This precise measurement method using GPIO allows a causal relationship and synchronization between the program and the power measurements.

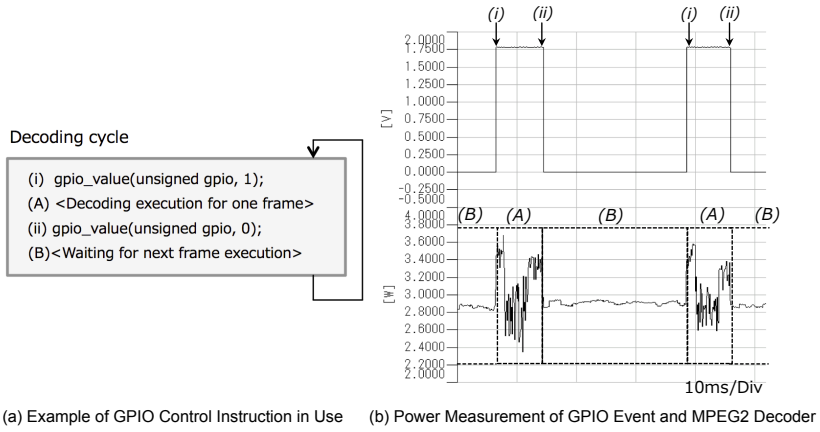

Fig. 2. Example of GPIO Control Functions

4 Evaluation Environment

This section provides an overview of the environment used for power measurements. Although there are many different kinds of smart devices with different chips, a platform which enables power measurements are very rare, or do not

exist in the consumer market. Therefore, to take power measurements on an Android platform, the evaluation environment itself had to be developed.

4.1 The Development of the Evaluation Environment

The ODROID-X2[20] is a development board, which has the Samsung Exynos4412 Prime chip. Within the Exynos4412 Prime[26], there are 4 ARM Cortex-A9 cores each with a maximum clock frequency of 1.7GHz with 1MB shared L2 cache memory, 2GB of dual channel LPDDR2 RAM is equipped on the board. The frequency and voltage scaling cannot be controlled differently on individual cores of this chip, but on all cores at the same time.

The ODROID-X2 development board is originally not designed for measuring power consumption of any parts of the board. In order to measure the power consumption of the cores on a chip some modifications are applied around the PMIC (Power Management IC)[27], which acts as the controller of power source of the CPU. PMIC on ODROID-X2 controls each power supply of the following function units on the CPU: battery, cores, memory, interrupt controller, accelerators, and so on. The modification of the circuit connected to the PMIC is shown in the dotted line of Figure 3. The modifications applied to the development board are the following: altering the power source circuit of cores connected to the PMIC, adding a 40[mΩ] shunt resistor, and placing a 10x gain instrumentation amplifier. By placing an amplifier with 10x gain, the measurement of voltage difference between both ends of the shunt resistor become precise. This newly developed environment enables measurements of electrical current on cores from tens of milliampere to thousands of milliampere.

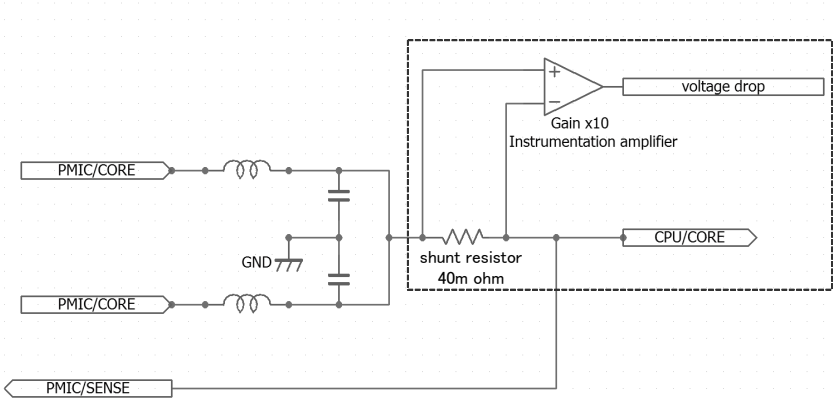

Fig. 3. Modified Circuit Diagram of ODROID-X2

4.2 Evaluated Applications on ODROID-X2

This section explains two real-time applications used for the power evaluations on the ODROID-X2.

MPEG2 Decoder. MPEG2 Decoder is a standard video coding application from MediaBench[28].

The OSCAR compiler exploits slice level parallelism from the program. The deadline set for the MPEG2 Decoder is set to 60[fps] (16.6[ms] per frame).

Optical Flow. The Optical Flow is a benchmark application referenced from OpenCV[29]. This real-time application tracks 16x16 blocks between two images by calculating the velocity fields.

OSCAR compiler exploits parallelism among calculations of velocity fields from each block in two images. The deadline for Optical Flow is set to 30[fps] (33[ms] per frame).

5 Evaluation of Power Reduction on ODROID-X2

This section presents the results of power evaluations on the modified ODROID-X2. Power consumption for each evaluation is exploited by the proposed power measurement method using GPIO mentioned in Section 3.4. The power reduction control parameters set in the OSCAR Compiler for frequencies are FULL(1700[MHz]), MID(900[MHz]), and LOW(400[MHz]). Moreover, the `cpufreq` governor on Android is set to `ondemand` for benchmark applications without power control and `userspace` for benchmark applications with power control.

5.1 Power Consumption of MPEG2 Decoder on ODROID-X2

Fig. 4 shows the power consumption results of MPEG2 Decoder for each number of processor element (PE). The power consumption of 1PE with power reduction controls consumed 0.63[W] (power reduced to 75.7%) compared to 0.97[W] on 1PE without power reduction controls. The power consumption of 2PE with power reduction controls consumed 0.46[W] (power reduced to 24.5%) compared to 1.88[W] on 2PE without power reduction controls. The power consumption of 3PE with power reduction controls consumed 0.37[W] (power reduced to 13.3%) compared to 2.79[W] on 3PE without power reduction controls. The 0.37[W] for 3PE with power control resulted in 86.7% power reduction against the ordinary 1PE execution without power control.

Fig. 5-(a) shows the power waveform of 1PE without power reduction control. MPEG2 Decoder in this figure is running at maximum frequency (1700[MHz]) and the `ondemand` governor seems to automatically lower the frequency when waiting for the deadline. Fig. 5-(b) shows the power waveform of 1PE with power reduction control. This figure shows that the OSCAR Compiler had analyzed 1PE of MPEG2 Decoder to run at FULL to meet the deadline time.

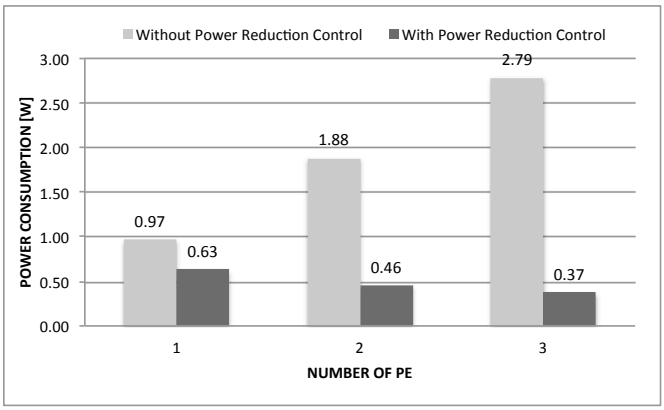

Fig. 4. Power Consumption of MPEG2 Decoder on ODROID-X2

Furthermore, by implementing the proposed pseudo clock gating method mentioned in section 3.3, power consumption decreases to approximately 0[W] when waiting for the deadline. Similarly, Fig. 6-(a) shows the power waveform of 3PE without power reduction control. In this figure, MPEG2 Decoder is running at maximum frequency using 3PE. However, Fig. 6-(b) shows that the OSCAR Compiler had analyzed 3PE of MPEG2 Decoder is fast enough to run at LOW and meet the deadline time. This figure exibits the significance of having power reduction controls at an application level on Android platforms. Furthermore, as explained in Fig. 5-(b), the pseudo clock gating lowers the power consumption to approximately 0[W] when waiting for the deadline time.

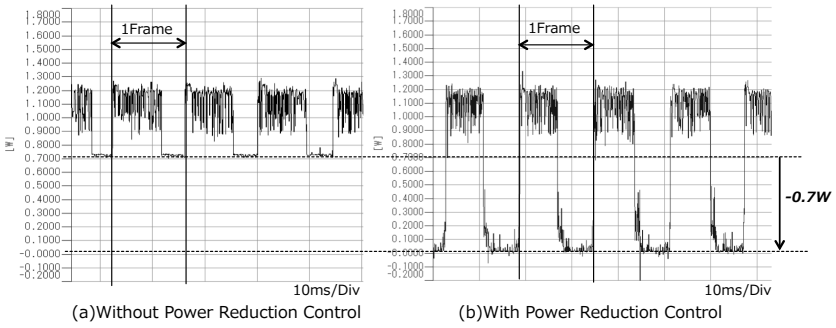

Fig. 5. Power Waveform of MPEG2 Decoder for 1PE

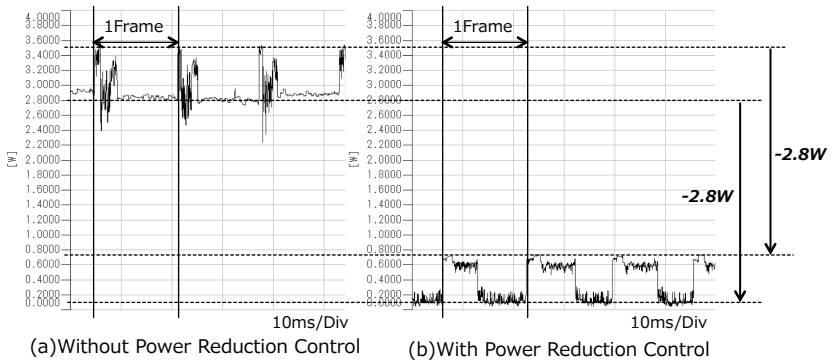

Fig. 6. Power Waveform of MPEG2 Decoder for 3PE

5.2 Power Consumption of Optical Flow on ODROID-X2

Fig. 7 shows the power consumption results of Optical Flow for each number of processor element (PE). The power consumption of 1PE with power reduction controls consumed 0.72[W] (power reduced to 75.8%) compared to 0.95[W] on 1PE without power reduction controls. The power consumption of 2PE with power reduction controls consumed 0.36[W] (power reduced to 24.0%) compared to 1.50[W] on 2PE without power reduction controls. The power consumption of 3PE with power reduction controls consumed 0.30[W] (power reduced to 13.5%) compared to 2.23[W] on 3PE without power reduction controls. The 0.30[W] for 3PE with power control resulted in 86.5% power reduction against the ordinary 1PE exeution without power control.

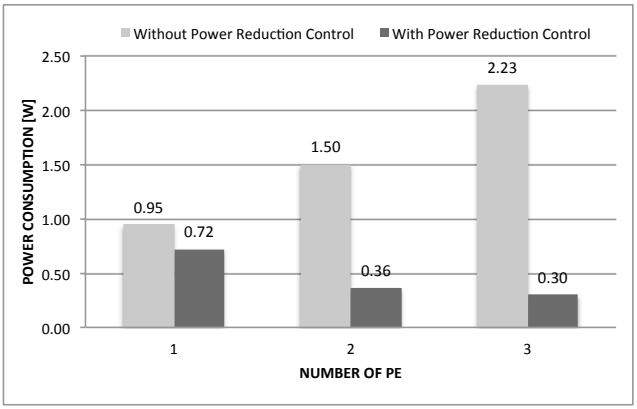

Fig. 7. Power Consumption of Optical Flow on ODROID-X2

Fig. 8-(a) shows the power waveform of 1PE without power reduction control. Optical Flow in this figure is running at maximum frequency (1700[MHz]) and the `ondemand` governor seems to lower the frequency when waiting for the deadline similar to Fig. 5. Fig. 8-(b) shows the power waveform of 1PE with power reduction control. This figure shows that the OSCAR Compiler had analyzed 1PE of Optical Flow to run at FULL to meet the deadline time. Power consumption decreases to approximately 0[W] when waiting for the deadline using the pseudo clock gating. Fig. 9-(a) shows the power waveform of 3PE without power reduction control. In this figure, Optical Flow is running at maximum frequency using 3PE. However, Fig. 9-(b) shows that the OSCAR Compiler had analyzed 3PE of Optical Flow is fast enough to run at LOW and meet the deadline time. This figure exibits the significance of having power reduction controls at an application level on Android platforms. Furthermore, as explained in Fig. 8-(b), the pseudo clock gating lowers the power consumption to approximately 0[W] when waiting for the deadline time.

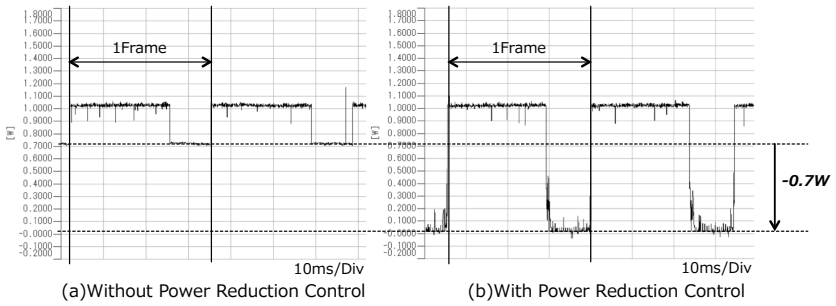

Fig. 8. Power Waveform of Optical Flow for 1PE

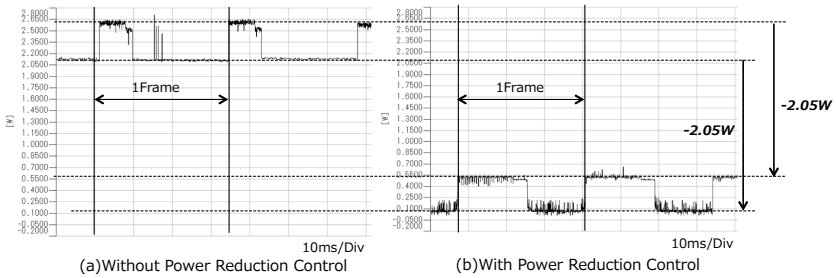

Fig. 9. Power Waveform of Optical Flow for 3PE

6 Conclusion

This paper evaluated the power reduction controls by the OSCAR Compiler on an Android platform, ODROID-X2. A pseudo clock gating method was developed using WFI to realize a low-overhead, or 100[us] transition time, power control by the compiler. All measurements in the evaluation were taken with the precise power measurement environment using the GPIO. For the evaluation, MPEG2 Decoder showed 86.7% power reduction on 3PE from 2.79[W] on ordinary execution to 0.37[W] on execution with power control by the OSCAR compiler. Similarly, Optical Flow showed 86.5% power reduction on 3PE from 2.23[W] on ordinary exection to 0.30[W] on execution with power control. The results exibit that the proposed pseudo clock gating method and the low power optimizations by the OSCAR Compiler enables significant power reduction on the Android platform.

References

1. Taylor, M., Kim, J., Miller, J., Wentzlaff, D.: The Raw microprocessor: a computational fabric for software circuits and general-purpose programs. Micro, IEEE (2002) 25–35
2. Hammond, L., Hubbert, B., Siu, M.: The Stanford Hydra CMP. IEEE (2000) 71–84
3. Friedrich, J., McCredie, B.: Design of the Power6 microprocessor. (2007) 96–97
4. Jeff, B.: Advances in big . LITTLE Technology for Power and Energy Savings. (September) (2012) 1–11
5. ARM Corporation: Cortex-A9 Technical Reference Manual `http://infocenter.arm.com/help/topic/com.arm.doc.ddi0388i/DDI0388I_cortex_a9_r4p1_trm.pdf`
6. NVIDIA Corporation: Whitepaper NVIDIA Tegra Multi-processor Architecture. 1–12
7. Samsung Electronics Co., L.: White Paper of Exynos 5. **1**(1) (April 2011) 1–8
8. Amarasinghe, S., Anderson, J.: An overview of the SUIF compiler for scalable parallel machines. **667** (1995)
9. Blume, W., Doallo, R., Eigenmann, R.: Parallel Programming with Polaris. Computer (1996)
10. Bondhugula, U., Ramanujam, J., Sadayappan, P.: Pluto: A practical and fully automatic polyhedral parallelizer and locality optimizer. Technical Report OSU-CISRC-10/07-TR70, The Ohio State University (October 2007)
11. Kasahara, H., Obata, M., Ishizaka, K.: Automatic coarse grain task parallel processing on smp using openmp. Workshop on Languages and Compilers for Parallel Computing (2001) 1–15
12. Obata, M., Shirako, J., Kaminaga, H., Ishizaka, K., Kasahara, H.: Hierarchical Parallelism Control for Multigrain Parallel Processing. Lecture Notes in Computer Science **2481** (2005) 31–44
13. 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 (2010) 188–202
14. 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 (2007) 362–376

15. Hsu, C.H., Kremer, U.: The design, implementation, and evaluation of a compiler algorithm for CPU energy reduction. Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation - PLDI '03 (2003) 38
16. Chen, G., Malkowski, K., Kandemir, M., Raghavan, P.: Reducing Power with Performance Constraints for Parallel Sparse Applications http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1420150
17. Xie, F., Martonosi, M., Malik, S.: Compile-time dynamic voltage scaling settings: Opportunities and limits. ACM SIGPLAN Notices (2003)
18. Martonosi, M., Clark, D., Reddi, V., Connors, D., Brooks, D.: Dynamic-Compiler-Driven Control for Microprocessor Energy and Performance (January 2006) http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1603505
19. Azevedo, A., Cornea, R., Issenin, I., Gupta, R., Dutt, N., Nicolau, A., Veidenbaum, A.: Architectural and compiler strategies for dynamic power management in the COPPER project. Innovative Architecture for Future Generation High-Performance Processors and Systems IWIA-01 (2001) 25–34
20. Hardkernel: ODROID-X2 http://www.hardkernel.com/renewal_2011/products/prdt_info.php?g_code=G135235611947
21. Google: Android Developers http://developer.android.com/index.html
22. Linux: CPU hotplug Support in Linux(tm) Kernel https://www.kernel.org/doc/Documentation/cpu-hotplug.txt
23. Honda, H., Kasahara, H.: Coarse Grain Parallelism Detection Scheme of a Fortran Program. Systems and computers in Japan (1991)
24. Obata, M., Shirako, J., Kaminaga, H.: Hierarchical parallelism control for multigrain parallel processing. (2005) 31–44
25. ARM Information Center: GPIO Interfaces https://www.kernel.org/doc/Documentation/gpio.txt
26. SAMSUNG ELECTRONICS: Samsung Exynos 4 Quad (Exynos 4412) RISC Microprocessor User's Manual. (October) (2012)
27. SAMSUNG ELECTRONICS: Samsung Semiconductors Global Site https://www.samsung.com/global/business/semiconductor/product/poweric/overview
28. Lee, C., Potkonjak, M., Mangione-Smith, W.: MediaBench : A Tool for Evaluating and Synthesizing Multimedia and Communications Systems. (1997) 330–335
29. : Opencv http://www.opencv.org