# Parallelizing Compiler Framework and API for Power Reduction and Software Productivity of Real-time Heterogeneous Multicores

Akihiro Hayashi, Yasutaka Wada, Takeshi Watanabe, Takeshi Sekiguchi, Masayoshi Mase, Jun Shirako, Keiji Kimura, and Hironori Kasahara

Department of Computer Science and Engineering, Waseda University,
3-4-1 Okubo, Shinjuku-ku, Tokyo, Japan,
{ahayashi,yasutaka,watanabe,
takeshi,mase,shirako,kimura}@kasahara.cs.waseda.ac.jp,
kasahara@waseda.jp,
http://www.kasahara.cs.waseda.ac.jp/

**Abstract.** Heterogeneous multicores have been attracting much attention to attain high performance keeping power consumption low in wide spread of areas. However, heterogeneous multicores force programmers very difficult programming. The long application program development period lowers product competitiveness. In order to overcome such a situation, this paper proposes a compilation framework which bridges a gap between programmers and heterogeneous multicores. In particular, this paper describes the compilation framework based on OSCAR compiler. It realizes coarse grain task parallel processing, data transfer using a DMA controller, power reduction control from user programs with DVFS and clock gating on various heterogeneous multicores from different vendors. This paper also evaluates processing performance and the power reduction by the proposed framework on a newly developed 15 core heterogeneous multicore chip named RP-X integrating 8 general purpose processor cores and 3 types of accelerator cores which was developed by Renesas Electronics, Hitachi, Tokyo Institute of Technology and Waseda University. The framework attains speedups up to 32x for an optical flow program with eight general purpose processor cores and four DRP(Dynamically Reconfigurable Processor) accelerator cores against sequential execution by a single processor core and 80% of power reduction for the real-time AAC encoding.

**Keywords:** Heterogeneous Multicore, Parallelizing Compiler, API

## 1 Introduction

There has been a growing interest in heterogeneous multicores which integrate special purpose accelerator cores in addition to general purpose processor cores on a chip. One of the reason for this trend is because heterogeneous multicores allow us to attain high performance with low frequency and low power consumption. Various semiconductor vendors have released heterogeneous multicores such as CELL BE[15], NaviEngine[11], Uniphier[13], GPGPU[9], RP1[20] and RP-X[21].

However, the softwares for heterogeneous multicores generally require large development efforts such as the decomposition of a program into tasks, the implementation of accelerator code, the scheduling of the tasks onto general purpose processors and accelerators, and the insertion of synchronization and data transfer codes. These software development periods are required even for expert programmers.

Recent many studies have tried to handle on this software development issue. For example, NVIDIA and Khronos Group introduced CUDA[3] and OpenCL[7]. Also, PGI accelerator compiler[19] and HMPP[2] provides a high-level programming model for accelerators. However, these works focus on facilitating the development for accelerators. Programmers need to distribute tasks among general purpose processors and accelerator cores by hand. In terms of workload distribution, Qilin[10] automatically decides which task should be executed on a general purpose processor or an accelerator at runtime. However, programmers still need to parallelize a program by hand. While these works rely on programmers' skills, CellSs[1] performs an automatic parallelization of a subset of sequential C program with data flow annotations on CELL BE. CellSs automatically schedules tasks onto processor elements at runtime. The task scheduler of CellSs, however, is implemented as a homogeneous task scheduler, namely the scheduler is executed on PPE and just distributes tasks among SPEs.

In the light of above facts, further explorations are needed since it is the responsibility of programmers to parallelize a program and to optimize a data transfer and a power consumption for heterogeneous multicores. One of our goals is to realize a fully automatic parallelization of a sequential C or Fortran77 program for heterogeneous multicores. We have been developing OSCAR paralleling compiler for homogeneous multicores such as SMP servers and real-time multicores[5, 8, 12]. These works realize automatic parallelization of programs written in Fortran77 or Parallelizable C, a kind of C programming style for parallelizing compiler, and power reduction with the support of both OSCAR compiler and OSCAR API(Application Program Interface)[6]. This paper describes an automatic parallelization for a real heterogeneous multicore chip. Though prior work demonstrates the performance of automatic parallelization of a Fortran program on a heterogeneous multicore simulator[18], this paper makes the following contributions:

- A proposal of an accelerator-independent and general purpose compilation framework including a compilation framework using OSCAR compiler and an extention of OSCAR API[8] for heterogeneous multicore
- An evaluation of a processing performance and a power efficiency using 3 Parallelizable C applications on the newly developed RP-X multicore chip[21].

In order to build an accelerator-independent and a general-purpose compilation framework, we take care of utilizing existing tool chains such as accelerator compilers and hand-tuned libraries for accelerators. Therefore, this paper firstly

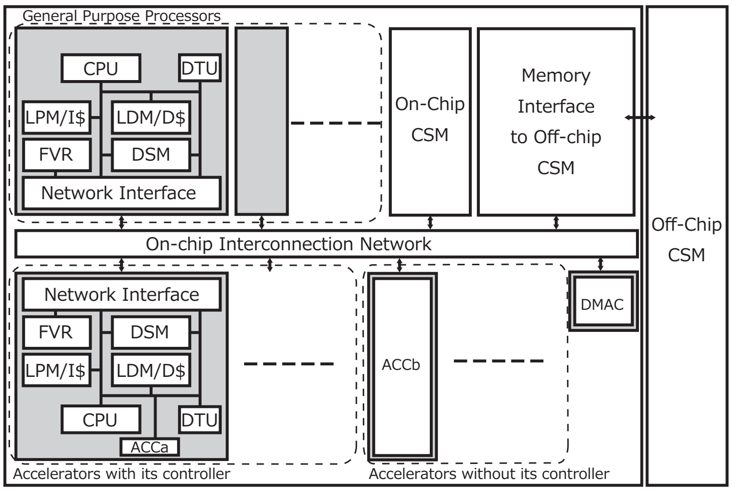

General Purpose Processors
CPU
DTU
LPM/I$
LDM/D$
FVR
DSM
Network Interface
On-Chip CSM
Memory Interface to Off-chip CSM
Off-Chip CSM
On-chip Interconnection Network
Network Interface
FVR
DSM
LPM/I$
LDM/D$
CPU
DTU
ACCa
ACCb
DMAC
Accelerators with its controller
Accelerators without its controller

**Fig. 1.** OSCAR API Applicable heterogeneous multicore architecture

defines an general-purpose architecture and compilation flow in Section 2. Secondly, we defines distinct responsibilities among these tool chains and interface among them by extending OSCAR API in Section 3.

## 2 OSCAR API Applicable Heterogeneous Multicore Architecture and Overview of the Compilation flow

This section defines both target architecture and compilation flow of the proposed framework. In this paper, define a term "controller" as a general purpose processor that controls an accelerator, that is to say, it performs part of coarse-grain task and data transfers from/to the accelerator and offload the task to the accelerator.

### 2.1 OSCAR API Applicable Heterogeneous Multicore Architecture

This section defines "OSCAR API Applicable Heterogeneous Multicore Architecture" shown in Fig.1.. The architecture is composed of general purpose processors, accelerators(ACCs), direct memory access controller(DMAC), on-chip centralized shared memory(CSM), and off-chip CSM. Some accelerators may have its own controller, or general purpose processor. Both general purpose processors and accelerators with controller may have a local data memory (LDM), a distributed shared memory (DSM), a data transfer unit (DTU), a frequency voltage control registers (FVR), an instruction cache memory and a data cache memory. The local data memory keeps private data. The distributed shared memory is a dual port memory, which enables point-to-point direct data transfer and low-latency synchronization among processors. Each existing heterogeneous multicore can be seen such as CELL BE[15], MP211[17] and RP1[20] as a subset

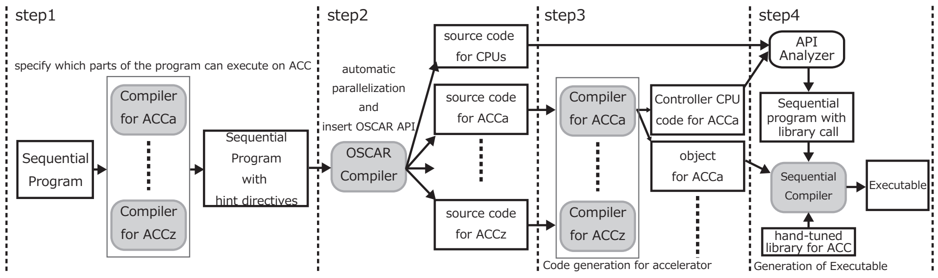


**Fig. 2.** Compilation flow of the proposed framework

of OSCAR API applicable architecture. Thus, OSCAR API can support such chips and a subset of OSCAR API applicable heterogeneous multicore.

## 2.2 Compilation Flow

Fig.2. shows the compilation flow of the proposed OSCAR heterogeneous compiler framework. The input is a sequential program written in Parallelizable C or Fortran77 and the output is an executable for a target heterogeneous multicore. The following describes each step in the proposed compilation flow.

**Step 1:** Accelerator compilers or programmers insert hint directives immediately before loops or function calls , which can be executed on the accelerator, in a sequential program.

**Step 2:** OSCAR compiler parallelizes the source program considering with hint directives: the compiler schedules coarse-grain tasks[18] to processor or accelerator cores and apply the low power control[8]. Then, the compiler generates a parallelized C or Fortran program for general purpose processors and accelerator cores by using OSCAR API. At that time, the compiler generates C source codes as separate files for accelerator cores. Each file includes functions to be executed on accelerators when a function is scheduled onto accelerator by the compiler.

**Step 3:** Each accelerator compiler generates objects for its own target accelerator. Note that each accelerator compiler also generates both data transfer code between controller and accelerator, and accelerator invocation code.

**Step 4:** An API analyzer prepared for each heterogeneous multicore translates OSCAR APIs into runtime library calls, such as pthread library. Afterwards, an ordinary sequential compiler for each processor from each vender generates an executable.

It is important that the framework also allows programmers to utilize existing hand-tuned libraries for the specific accelerator. This paper defines a term "hand-tuned library" as an accelerator library which includes computation body on the specific accelerator and both data transfer code between general purpose processors and accelerators and accelerator invocation code.

```
int main() {
   int i, x[N], var1 = 0;
   /* loop1 */
   for (i = 0; i < N; i++) { x[i] = i; }
   /* loop2 */
#pragma oscar_hint accelerator_task (ACCa) \
                           cycle(1000,((OSCAR_DMAC()))) workmem(OSCAR_LDM(), 10)
   for (i = 0; i < N; i++)  { x[i]++; }
   /* function3 */
#pragma oscar_hint accelerator_task (ACCb) \
                           cycle(100, ((OSCAR_DTU()))) in(var1,x[2:11]) out(x[2:11])
   call_FFT(var1, x);
   return 0;
}
```

```
void call_FFT(int var, int* x) {
#pragma oscar_comment "XXXXX"
   FFT(var, x);  //hand-tuned library call
}
```

**Fig. 3.** Example of source code with hint directives

## 3 A Compiler Framework for Heterogeneous Multicores

This section describes the detail of OSCAR compiler and OSCAR API.

### 3.1 Hint Directives for OSCAR Compiler

This subsection explains the hint directives for OSCAR compiler that advice OSCAR compiler which parts of the program can be executed by which accelerator core.

Fig.3. shows an example code. As shown in Fig.3., there are two types of hint directives inserted to a sequential C program, namely "accelerator_task" and "oscar_comment". In this example, there are "#pragma oscar_hint accelerator_task (ACCa) cycle(1000, ((OSCAR_DMAC()))) workmem(OSCAR_LDM(), 10)" and "#pragma oscar_hint accelerator_task (ACCb) cycle(100, ((OSCAR_DTU()))) in(var1, x[2:11]) out(x[2:11])". In these directives, accelerators represented as "ACCa" and "ACCb" is able to execute a loop named "loop2" and a function named "function3", respectively. The hint directive for "loop2" specifies that "loop2" requires 1000 cycles including the cost of a data transfer performed by DMAC if the loop is processed by "ACCa". This directive also specifies that 10 bytes in local data memory are required in order to control "ACCa". Similarly, for "function3", it takes 100 cycles including the cost of a data transfer by DTU. Input variables are scalar variable "var1" and array variable "x" ranging 2 to 11. Also, output variable is array variable "x". "oscar_comment" directive is inserted so that either programmers or accelerator compilers give a comment to accelerator compiler through OSCAR compiler.

### 3.2 OSCAR Parallelizing Compiler

This subsection describes OSCAR compiler.

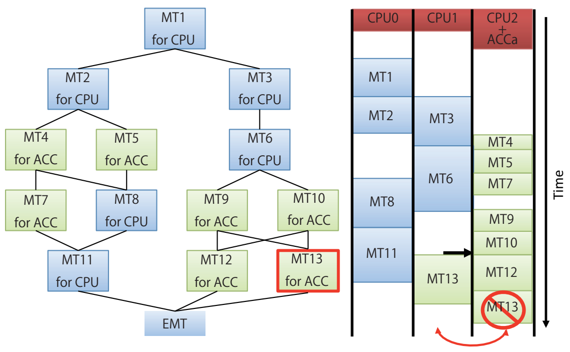


**Fig. 4.** An Example of Task Scheduling Result

First of all, the compiler decomposes a program into coarse grain tasks, namely macro-tasks (MTs), such as basic block (BPA), loop (RB), and function call or subroutine call (SB). Then, the compiler analyzes both the control flow and the data dependencies among MTs and represents them as a macro-flow-graph (MFG). Next, the compiler applies the earliest executable condition analysis, which can exploit parallelism among MTs associated with both the control dependencies and the data dependencies. The analysis result is represented as a hierarchically-defined macro-task-graph (MTG)[5]. When the compiler cannot analyze the input source for some reason, like hand-tuned accelerator library call, "in/out" clause of "accelerator_task" gives the data dependency information to OSCAR compiler. Then, the compiler calculates the cost of MT and finds the layer which is expected to apply coarse-grain parallel processing most effectively. "cycle" clause of "accelerator_task" tells the cost of accelerator execution to the compiler.

Secondly, the task scheduler of the compiler statically schedules macro-tasks to each core[18]. Fig.4. shows an example of heterogeneous task scheduling result. First the scheduler gets ready macro-tasks from MTG(MT1 in Fig.4 in initial state). Ready tasks satisfy earliest executable condition[4]. Then, the scheduler selects a macro-task to be scheduled from the ready macro-tasks and schedules the macro-task onto general purpose processor or accelerator considering data transfer overhead, according to the priorities, namely CP length. The scheduler performs above sequences until all macro-tasks are scheduled. Note that a task for an accelerator is not always assigned to the accelerator when the accelerator is busy. At this case, the task may be assigned to general purpose processor to minimize total execution time.

Thirdly, the compiler tries to minimize total power consumption by changing frequency and voltage(DVFS) or shutting power down the core during the idle time considering transition time[16]. The compiler determines suitable voltage and frequency for each macro-task based on the result of static task assignment

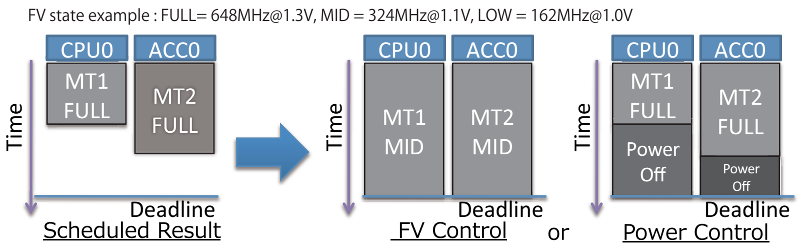


**Fig. 5.** Power control by compiler

in order to satisfy the deadline for real-time execution(Fig.5.). In Fig.5., FULL is 648MHz and MID is 324MHz, respectively. Each of which is used in RP-X described in Section4.

Finally, the compiler generates parallelized C or Fortran program with OSCAR API. OSCAR compiler generates the function which includes original source for accelerator. Generation of data transfer codes and accelerator invocation code is responsible for accelerator compiler.

OSCAR compiler uses processor configurations, such as number of cores, cache or local memory size, available power control mechanisms, and so on. This information is provided by compiler options.

## 3.3 The Extension of OSCAR API for Heterogeneous Multicores

This subsection describes API extension for heterogeneous multicores to be the output of OSCAR compiler. Thee extension is very simple. Only one directive "accelerator_task_entry" is added to OSCAR homogeneous API. This directive specifies the function's name where general purpose processor invokes an accelerator.

Let us consider an example where the compiler parallelizes the program in Fig.3. We assume a target multicore includes two general purpose processors, one ACCa as an accelerator with its controller and one ACCb as an accelerator without its controller. One of general purpose processors, namely CPU1, is used as controller for ACCb in this case. Fig.6. shows as example of the parallelized C code with OSCAR heterogeneous directive generated by OSCAR compiler. As shown in Fig.6., functions named "MAIN_CPU0()", "MAIN_CPU1()" and "MAIN_CPU2()" are invoked in omp parallel sections. These functions are executed on general purpose processors. In addition, hand-tuned library "oscartask_CTRL1_call_FFT()" executed on ACCa is called by controller "MAIN_CPU1()". "MAIN_CPU2" also calls kernel function "oscartask_CTRL2_call_loop2()" executed on ACCb. "accelerator_task_entry" directive specifies these two functions. "controller" clause of the directive specifies id of general purpose CPU which controls the accelerator. Note that there exists "oscar_comment" directives at same place shown in Fig.3.. "oscar_comment" directives may be used to give

```
int main() {
#pragma omp parallel sections
  {
#pragma omp section
    { MAIN_CPU0(); }
#pragma omp section
    { MAIN_CPU1(); }
#pragma omp section
    { MAIN_CPU2(); }
  }
  return 0;
}
```

```
int MAIN_CPU1() {
  ...
  oscartask_CTRL1_call_FFT(var1, &x);
  ...
}
int MAIN_CPU2() {
  ...
  oscartask_CTRL2_call_loop2(&x);
  ...
}
```

Source Code for CPUs

```
#pragma oscar accelerator_task_entry controller(2) \
                                        oscartask_CTRL2_loop2
void oscartask_CTRL2_loop2(int *x) {
  int i;
  for (i = 0; i <= 9; i += 1) { x[i]++; }
}
```

Source Code for ACCa

```
#pragma oscar accelerator_task_entry controller(1) \
                                        oscartask_CTRL1_call_FFT
void oscartask_CTRL1_call_FFT(int var1, int *x) {
#pragma oscar_comment "XXXXX"
  oscarlib_CTRL1_ACCEL3_FFT(var1, x);
}
```

Source Code for ACCb

**Fig. 6.** Example of parallelized source code with OSCAR API

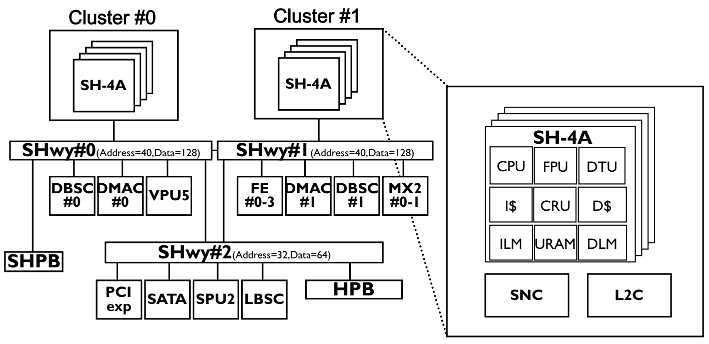


**Fig. 7.** RP-X heterogeneous multicore for consumer electronics

accelerator specific directives, such as PGI accelerator directives, to accelerator compilers. Afterwards, accelerator compilers generates the source code for the controller and objects for the accelerator, interpreting these directives.

## 4 Performance Evaluations on RP-X

This section evaluates the performance of the proposed framework on 15 core heterogeneous multicore RP-X[21] using media applications.

### 4.1 Evaluation Environment

The RP-X processor is composed of eight 648MHz SH-4A general purpose processor cores and four 324MHz FE-GA accelerator cores, the other dedicated hardware IP such as matrix processor "MX-2" and video processing unit "VPU5", as shown in Fig.7.. Each SH-4A core consists of a 32KB instruction cache, a 32KB data cache, a 16KB local instruction/data memory(ILM and DLM in Fig.7.), a 64KB distributed shared memory(URAM in Fig.7) and a data transfer unit. Furthermore, FE-GA is used as an accelerator without controller because FE-GA is directly connected with on-chip interconnection network named "SHwy#1", a split transaction bus. With regard to the power reduction control mechanism of RP-X, DVFS and clock gating for each SH-4A core can be controlled independently using special power control register by a user. DVFS for FE-GAs can be controlled by a user. This hardware mechanism is low overhead, for example frequency change needs a few clocks. This paper evaluates both generating the object code by accelerator compiler and using the hand-tuned library on RP-X processor. We evaluate the processing performance and the power consumption of the proposed framework using upto eight SH-4A cores and four FE-GA cores.

### 4.2 Performance by OSCAR compiler with Accelerator Compiler

An "optical flow" application from OpenCV[14] is used for this evaluation. The algorithm is a type of object tracking system, which calculates velocity field between two images. The program is modified in Parallelizable C[12] in this evaluation. This program consists of the following parts: dividing the image into 16x16 pixel blocks, searching a similar block in the next image for every block in the current image, shifting 16 pixels and generating the output. OSCAR compiler parallelizes the loop which searches a similar block in the next image. In addition, FE-GA compiler developed by Hitachi analyzed that the sum of absolute difference(SAD), which occupies a large part of the program execution time, is to be executed on FE-GA. FE-GA compiler also automatically inserts the hint directives to the C program. OSCAR compiler generates parallel C program with OSCAR heterogeneous API. The parallel program is translated into parallel executable binary by using API analyzer which translates the directives to library calls and sequential compiler and FE-GA compiler translates the program parts in the accelerator files to FE-GA binary. Input images are two 320x352 bitmap images. Data transfer between SH-4A and FE-GA is performed by SH-4A via data cache.

Fig.8. shows parallel processing performance of the optical flow on RP-X. The horizontal axis shows the processor configurations. For example, 8SH+4FE represents for the configuration with eight SH-4A general purpose cores and four FE-GA accelerator cores. The vertical axis shows the speedup against the sequential execution by a SH-4A core. As shown in Fig.8, the proposed compilation framework achieves speedups of up to 12.36x with 8SH+4FE.

### 4.3 Performance by OSCAR compiler and Hand-tuned Library

In this evaluation, we evaluate two applications written in Parallelizable C. The one is the optical flow from Hitachi Ltd. and Tohoku university, and the other is AAC encoder available on a market from Renesas Technology.

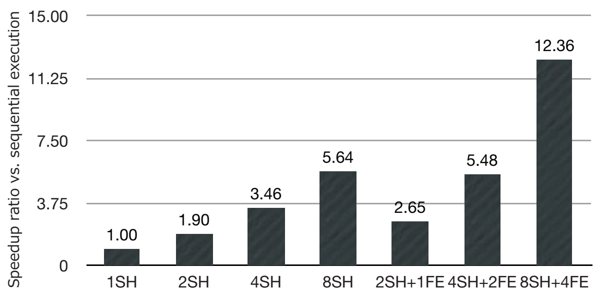


**Fig. 8.** Performance by OSCAR compiler and FE-GA Compiler(Optical Flow)

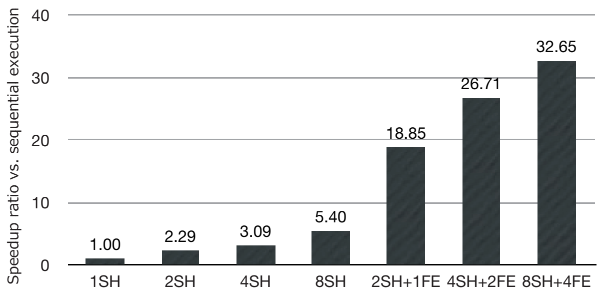


**Fig. 9.** Performance by OSCAR compiler and Hand-tuned Library(Optical Flow)

There are a few differences between the optical flow program used in this section and the program in Section4.2: In the optical flow program for this section, shift amount is 1 pixel, the input of the application is a sequence of images, and hand-tuned library for FE-GA is utilized. OSCAR compiler parallelizes the same loop, which is shown in the previous subsection. The hand-tuned library, which executes 81 SAD functions in parallel, is used for FE-GA. The hint directives are inserted to the parallelizable C program. OSCAR compiler generates parallel C program with OSCAR API or directives for these library function calls. The directives in the parallel program is translated to library calls by using API analyzer. Then, sequential compiler generates the executables linking with hand-tuned library for SAD. Input image size, number of frames and block size is 352x240, 450, 16x16, respectively. Data transfer between SH-4A and FE-GA is performed by SH-4A via data cache. AAC encoding program is based on the AAC-LC encode program provided by Renesas Technology and Hitachi Ltd. This program consists of filter bank, midside(MS) stereo, quantization and huffman coding. OSCAR compiler parallelizes the main loop which encodes a frame.

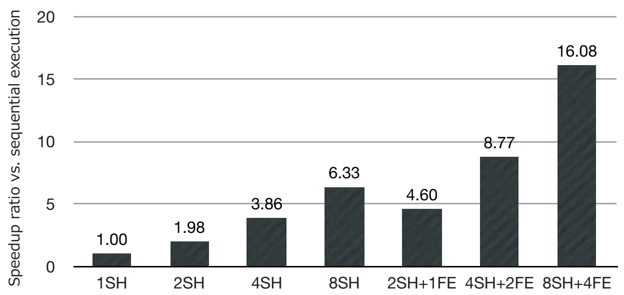


**Fig. 10.** Performance by OSCAR compiler and Hand-tuned Library(AAC)

The hand-tuned library for filter bank, MS stereo and quantization is used for FE-GA. Data transfer between SH-4A and FE-GA is performed by DTU via distributed shared memory.

Fig.9. shows parallel processing performance of the optical flow at RP-X. The horizontal axis shows the processor configurations. For example, 8SH+4FE represents for the configuration with eight SH-4A general purpose cores and four FE-GA accelerator cores. The vertical axis shows the speedup against the sequential execution by a SH-4A core. As shown in Fig.9, the proposed framework achieved speedups of up to 32.65x with 8SH+4FE.

Fig.10. shows parallel processing performance of the AAC at RP-X. As shown in Fig.10, the proposed framework achieved speedups of up to 16.08x with 8SH+4FE.

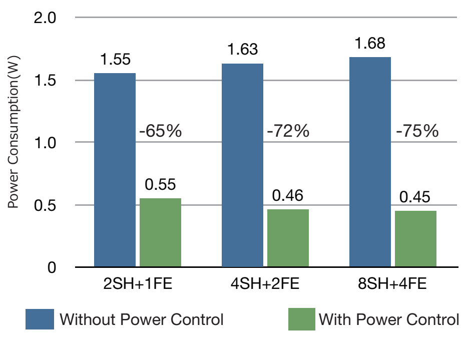
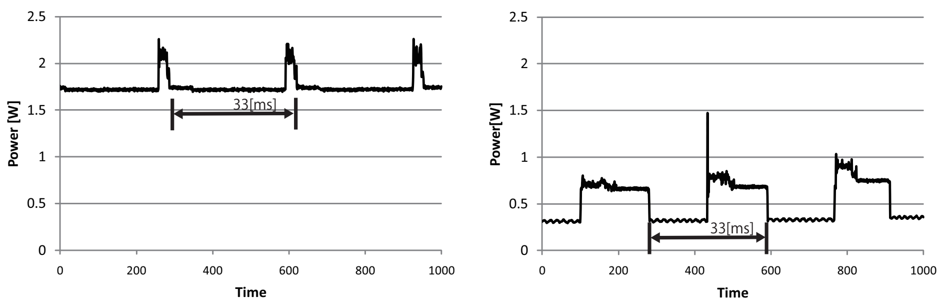


**Fig. 11.** Power reduction by OSCAR compiler's power control (Optical Flow)



a) Without Power Saving(Average:1.68W) b) With Power Saving(Average:0.45W)

**Fig. 12.** Waveforms of Power Consumption(Optical Flow)

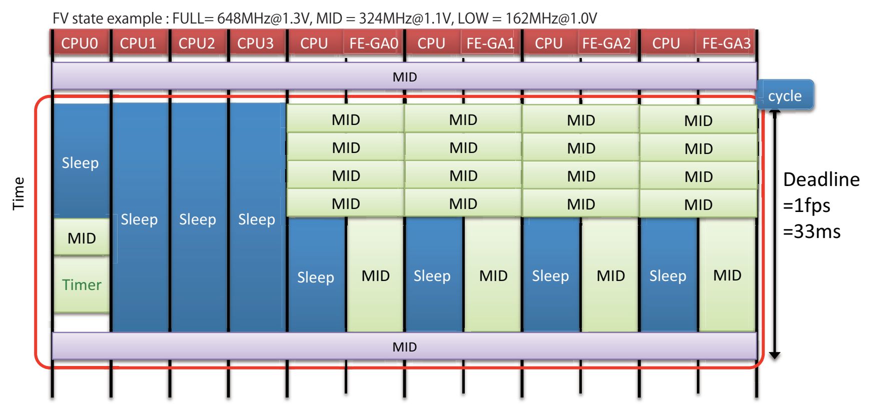


**Fig. 13.** Power Control for 8SH+4FE(Optical Flow)

## 4.4 Evaluation of Power Consumption

This section evaluates a power consumption by using optical flow and AAC encoding for real-time execution on RP-X. Fig.11 shows the power reduction by OSCAR compiler's power control, under the condition satisfying the deadline. The deadline of the optical flow is set to 33ms for each frame processing so that standard 30 [frames/sec] for moving picture processing can be achieved. The minimum number of cores required for the deadline satisfaction of optical flow calculation is 2SH+1FE. As shown in Fig.11, OSCAR heterogeneous multicore compiler reduces from 65% to 75% of power consumption for each processor configuration. Although power consumption is increased by the augmentation of processor core, the proposed framework reduces the power consumption.

Fig.12 shows the waveforms of power consumption in the case of optical flow using 8SH+4FE. The horizontal axis and the vertical axis show elapsed time and a power consumption, respectively. In the Fig.12, the arrow shows a processing

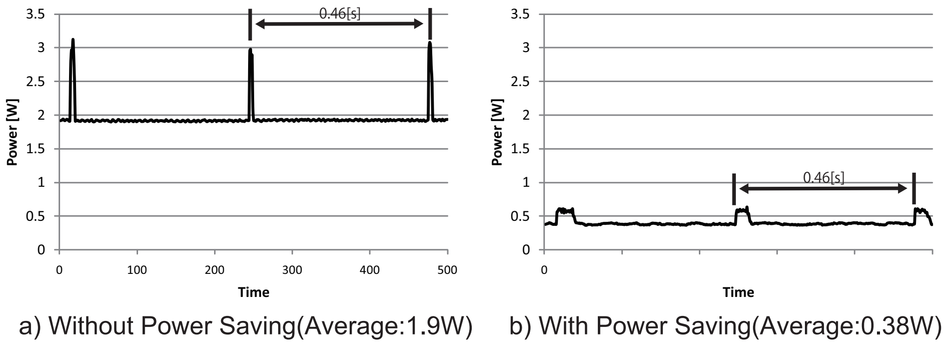


**Fig. 14.** Waveforms of Power Consumption(AAC)

period for one frame, or 33ms. In the case of applying power control(shown in Fig.12. b), each core executes the calculation by changing the frequency and the voltage on a chip. As a result, the consumed power ranges 0.3 to 0.7[W] by OSCAR compiler's power control. On the contrary, in the case of applying no power control(shown in Fig.12. a), the consumed power ranges 2.25[W] to 1.75[W].

Fig.13 shows the summary of frequency and voltage status for optical flow calculation with 8SH+4FE. In this figure, FULL is 648MHz with 1.3V, MID is 324MHz with 1.1V, and LOW is 162MHz with 1.0V. Each box labeled "MID" and "timer" "Sleep" represents macro-task. As shown in Fig.13., four SAD tasks are assigned to each FE-GA, and the tasks are executed at MID. All SH-4A core except "CPU0" is shutdown until the deadline comes. "CPU0" executes "timer" task for satisfying the deadline. In other words, "CPU0" boot up other SH-4A cores when the program execution reaches the deadline. Note that FE-GA core is not shutdown after task execution because DVFS is only applicable.

For AAC program, an audio stream is processed per frame. The deadline of AAC is set to encode 1 [sec] audio data within 1 [sec]. Fig.14 shows the waveforms of power consumption in the case of AAC using 8SH+4FE. In the case of applying power control(shown in Fig.14. b)), each core execute the calculation by changing the frequency and the voltage on a chip. As a result, the consumed power ranges 0.4 to 0.55[W]. On the contrary, in the case of applying no power control(shown in Fig.14. a), the consumed power ranges 1.9[W] to 3.1[W]. In summary, the proposed framework realizes the automatically power reduction of heterogeneous multicore for several applications.

## 5 Conclusions

This paper has proposed OSCAR heterogeneous multicore compilation framework. In particular, this paper introduces (1)the general purpose and multi-platform automatic compilation flow using OSCAR compiler and various accelerator compilers or hand-tuned libraries and (2)the heterogeneous extension of OSCAR homogeneous API. In this paper, we have evaluated the processing performance and the power efficiency of the proposed framework using RP-X, 15 core heterogeneous multicore chip, as an example. The developed framework automatically gave us speedups of up to 32x for an optical flow program with eight general purpose processor cores and four accelerator cores against sequential execution. Also, it shows 80% of power reduction by automatic DVFS for the real-time AAC encoding execution mode with eight general purpose processor cores and four accelerator cores compared with no power control.

## Acknowledgement

This work has been partly supported by the METI/NEDO project "Heterogeneous Multicore for Consumer Electronics" and MEXT project "Global COE Ambient Soc". Specifications of OSCAR API[6] heterogeneous multicore extension are developed by NEDO Heterogeneous multicore architecture and API committee at Waseda university. The authors specially thanks to the members of the API committee from Fujitsu Laboratory, Hitachi, NEC, Panasonic, Renesas Technology, and Toshiba. The hand-tuned library for FE-GA is provided by Hariyama Lab. at Tohoku university and Hitachi.

## References

1. Bellens, P., Perez, J.M., Badia, R.M., Labarta, J.: Cellss: a programming model for the cell be architecture. In Proceedings of the 2006 ACM/IEEE Conference on Supercomputing(SC'06) (2009)
2. Dolbeau, R., Bihan, S., Bodin, F.: Hmpp(tm):a hybrid multi-core parallel programmingg environment. In: GPGPU '07: Proceedings of the 1st Workshop on General Purpose Processing on Graphics Processing Units (2007)
3. Garland, M., Grand, S.L., Nickolls, J., Anderson, J., Hardwick, J., Morton, S., Phillips, E., Zhang, Y., Volkov, V.: Parallel computing experiences with cuda. IEEE Micro 28(4), 13–27 (2008)
4. Kasahara, H., Honda, H., Mogi, A., Ogura, A., Fujiwara, K., Narita, S.: A multi-grain parallelizing compilation scheme for OSCAR (Optimally scheduled advanced multiprocessor). In: Proceedings of the Fourth International Workshop on Languages and Compilers for Parallel Computing. pp. 283–297 (August 1991)
5. Kasahara, H., Obata, M., Ishizaka, K.: Automatic coarse grain task parallel processing on smp using openmp. Proc of The 13th International Workship on Languages and Compilers for Parallel Computing(LCPC2000) (2000)
6. kasahara.cs.waseda.ac.jp: Oscar-api v1.0. http://www.kasahara.cs.waseda.ac.jp/
7. khronos.org: Opencl. http://www.khronos.org/opencl/
8. Kimura, K., Mase, M., Mikami, H., Miyamoto, T., Kasahara, J.S.H.: Oscar api for real-time low-power multicores nad its performance on multicores and smp servers. Proc of The 22nd International Workship on Languages and Compilers for Parallel Computing(LCPC2009) (2009)

9. Luebke, D., Harris, M., Govindaraju, N., Lefohn, A., Houston, M., Owens, J., Segal, M., Papakipos, M., Buck, I.: Gpgpu: General-purpose computation on graphics hardware. In: 2006 ACM/IEEE Conference on Supercomputing, SC'06 (11 November 2006 through 17 November 2006 2006)
10. Luk, C., Hong, S., Kim, H.: Qilin: Exploiting parallelism on heterogeneous multiprocessors with adaptive mapping, microarchitecture. 2009. MICRO-42. Proceedings. 42th Annual IEEE/ACM International Symposium on Microarchitecture (2009)
11. Masayasu, Y., Takeshi, S., Toshiaki, T., Yasuhiko, K., Toshinori, I.: Naviengine 1, system lsi for smp-based car navigation systems. NEC TECHNICAL JOURNAL 2(4) (2007)
12. Mase, M., Onozaki, Y., Kimuraa, K., Kasahara, H.: Parallelizable c and its performance on low power high performance multicore processors. In: Proc. of 15th Workshop on Compilers for Parallel Computing (Jul 2010)
13. Nakajima, M., Yamamoto, T., Yamasaki, M., Hosoki, T., Sumita, M.: Low power techniques for mobile application socs based on integrated platform "uniphier". In: ASP-DAC '07: Proceedings of the 2007 Asia and South Pacific Design Automation Conference (2007)
14. opencv.org: Opencv. http://opencv.org/
15. Pham, D., Asano, S., Bolliger, M., Day, M.N., Hofstee, H.P., Johns, C., Kahle, J., Kameyama, A., Keaty, J., Masubuchi, Y., Riley, M., Shippy, D., Stasiak, D., Suzuoki, M., Wang, M., Warnock, J., Weitzel, S., Wendel, D., Yamazaki, T., Yazawa, K.: The design and implementation of a first-generation cell processor. In: 2005 IEEE International Solid-State Circuits Conference, ISSCC (6 February 2005 through 10 February 2005 2005)
16. 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 4339 pp. 362–376 (2007)
17. Torii, S., Suzuki, S., Tomonaga, H., Tokue, T., Sakai, J., Suzuki, N., Murakami, K., Hiraga, T., Shigemoto, K., Tatebe, Y., Obuchi, E., Kayama, N., Edahiro, M., Kusano, T., Nishi, N.: A 600mips 120mw 70 $\mu$ a leakage triple-cpu mobile application processor chip. ISSCC (2005)
18. Wada, Y., Hayashi, A., Masuura, T., Shirako, J., Nakano, H., Shikano, H., Kimura, K., Kasahara, H.: Parallelizing compiler cooperative heterogeneous multicore. In: Proceedings of Workshop on Software and Hardware Challenges of Manycore Platforms, SHCMP'08 (Jun 2008)
19. Wolfe, M.: Implementing the pgi accelerator model. In: GPGPU '10: Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units (2010)
20. Yoshida, Y., Kamei, T., Hayase, K., Shibahara, S., Nishii, O., Hattori, T., Hasegawa, A., Takada, M., Irie, N., Uchiyama, K., Odaka, T., Takada, K., Kimura, K., Kasahara, H.: A 4320mips four-processor core smp/amp with individually managed clock frequency for low power consumption. IEEE International Solid-State Circuits Conference, ISSCC (Feb 2007)
21. Yuyama, Y., Ito, M., Kiyoshige, Y., Nitta, Y., Matsui, S., Nishii, O., Hasegawa, A., Ishikawa, M., Yamada, T., Miyakoshi, J., Terada, K., Nojiri, T., Satoh, M., Mizuno, H., Uchiyama, K., Wada, Y., Kimura, K., Kasahara, H., Maejima, H.: A 45nm 37.3gops/w heterogeneous multi-core soc. IEEE International Solid-State Circuits Conference, ISSCC (Feb 2010)