A Parallelizing Compiler Cooperative Heterogeneous Multicore Processor Architecture

Yasutaka Wada, Akihiro Hayashi, Takeshi Masuura, Jun Shirako, Hirofumi Nakano, Hiroaki Shikano, Keiji Kimura, and Hironori Kasahara

Department of Computer Science and Engineering, Waseda University
3-4-1 Ohkubo, Shinjuku-ku, Tokyo 169-8555, Japan
{yasutaka,ahayashi,masuura,shirako,hnakano,
shikano,kimura,kasahara}@kasahara.cs.waseda.ac.jp

Abstract. Heterogeneous multicore architectures, integrating several kinds of accelerator cores in addition to general purpose processor cores, have been attracting much attention to realize high performance with low power consumption. To attain effective high performance, high application software productivity, and low power consumption on heterogeneous multicores, cooperation between an architecture and a parallelizing compiler is important. This paper proposes a compiler cooperative heterogeneous multicore architecture and parallelizing compilation scheme for it. Performance of the proposed scheme is evaluated on the heterogeneous multicore integrating Hitachi and Renesas' SH4A processor cores and Hitachi's FE-GA accelerator cores, using an MP3 encoder. The heterogeneous multicore gives us 14.34 times speedup with two SH4As and two FE-GAs, and 26.05 times speedup with four SH4As and four FE-GAs against sequential execution with a single SH4A. The cooperation between the heterogeneous multicore architecture and the parallelizing compiler enables to achieve high performance in a short development period.

1 Introduction

The demands for high performance, low power consumption, cost effectiveness and short software development period have been increasing in the area of consumer electronics such as mobile phones, games, digital TVs, and car navigation systems. To satisfy these demands, multicore processors[1–8] have been attracting much attention. Especially in consumer electronics, heterogeneous multicores[9–15], that integrate general purpose processor cores and various accelerator cores such as dynamically reconfigurable processors (DRPs), digital signal processors (DSPs), graphic processors and/or matrix processors (MTXs)[16] on a chip, have been researched to achieve both high performance with low cost and low power consumption.

Many types of heterogeneous multicores are being developed such as Larrabee [9], CELL[10], Stream processor[11], MP211[12], SH4A heterogeneous multicore [13], and Single-ISA heterogeneous multicore[14]. In addition, development environments or language extensions like CUDA[17] come to be used to make it easy

P. Stenström (Ed.): Transactions on HiPEAC IV, LNCS 6760, pp. 215–233, 2011.

to use accelerator cores. However, parallelizing application programs for various configurations of heterogeneous multicore processors is human-intensive and rather difficult. Therefore, a parallelizing compiler, which automatically extracts parallelism from a sequential program and schedules tasks to heterogeneous cores considering data transfer overhead, is required to minimize development periods of application softwares.

A parallelizing compiler for heterogeneous multicore systems has to schedule tasks to various types of hardware resources. Some task scheduling algorithms have been proposed for heterogeneous systems with availability constraints of resources[18, 19]. Though some heuristic scheduling algorithms assume to be used at compile-time[20, 21], a parallelizing compiler employing such task scheduling algorithms has not been developed before.

This paper proposes a parallel compilation scheme with a static scheduling scheme[22] for coarse grain task parallel processing[23] and a heterogeneous multicore architecture to support the compiler parallelization. The developed compiler is based on the OSCAR multigrain parallelizing compiler[24]. In the proposed compilation scheme, the compiler groups general purpose processor cores on a chip hierarchically to utilize hierarchical parallelism of a program effectively. In this processor grouping, accelerator cores are not included in the hierarchical grouping of general purpose processor cores so that the compiler can improve the availability of accelerator cores even if the number of them is less than the number of general purpose processor cores. Then, the compiler applies a static scheduling scheme extended for a heterogeneous multicore processor and optimizes data transfer timings considering overlapping with task executions. The proposed heterogeneous multicore architecture is developed for cooperating with the OSCAR heterogeneous compiler. Each accelerator core on this architecture is equipped with simple processor core to control the accelerator core. This controller makes it possible that the compiler adjusts the granularity of tasks can be executed on accelerators to apply the static scheduling. In addition, each core on a chip has the same memory architecture. This memory homogeneity supports the static scheduling scheme with memory and data transfer optimization by the compiler.

The remainder of this paper is organized as follows: Overviews of a heterogeneous multicore architecture cooperative with the parallelizing compiler are discussed in Section 2. Overviews of a compilation flow for heterogeneous multicore processors are described in Section 3. A coarse grain task parallel processing method for the heterogeneous multicore architecture is explained in Section 4. A coarse grain task scheduling scheme on the heterogeneous multicore architecture considering data transfers is proposed in Section 5. An experimental performance evaluation using an MP3 encoder program is described in Section 6.

2 A Heterogeneous Multicore with OSCAR-Type Memory Architecture

The proposed heterogeneous multicore architecture (Fig. 1) is based on homogeneous OSCAR-type memory architecture[22, 25, 26]. The architecture has

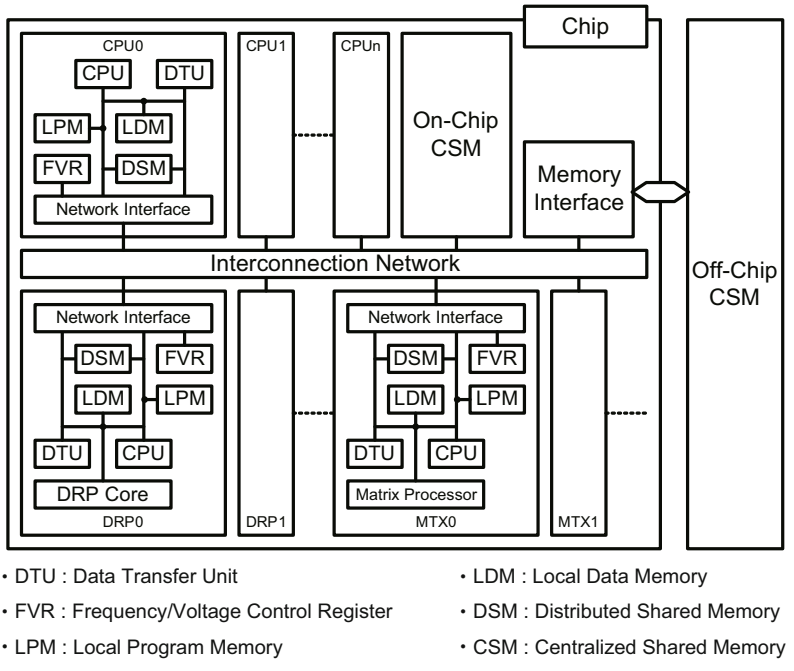

Fig. 1. OSCAR Heterogeneous Multicore Architecture

multiple processor elements (PEs) and centralized shared memories (CSM) connected by an interconnection network like multiple buses or crossbar network. A general purpose PE has a general purpose processor core (CPU). An accelerator PE has an accelerator core such as a dynamically reconfigurable processor (DRP) and a matrix processor (MTX) with a small controller processor core. This controller helps to make synchronizations among PEs, data transfers, and some simple calculations, so that OSCAR compiler can apply static scheduling scheme as explained in Section 5 even if the input program has complex control flow structure. For example, control flow structures such as conditional branches can be processed by the controller. This paper focuses on the applications that the compiler can apply the static scheduling scheme in the cooperation with this architecture.

Each general purpose and accelerator PE has a local data memory (LDM), a distributed shared memory (DSM), a data transfer unit (DTU) and frequency voltage control registers (FVR). The Local data memory stores PE private data in a program. The distributed shared memory is a dual port memory used for data transfers between PEs and low-latency synchronizations. Different from cache memory, the local data memory and the distributed shared memory are managed by software to handle real-time applications. The data transfer unit is an advanced DMA Controller which enables overlapping task execution and data transfer. OSCAR compiler controls and optimizes data transfer timings and

data allocation for the PEs using these memories and DTUs with the support of this homogeneous memory architecture. To make the task cost estimation by the compiler easy and precise, this architecture expects to use simple CPU cores as the general purpose cores.

3 A Compilation Flow for Heterogeneous Multicore Processors

For a heterogeneous multicore, the parallelizing compiler needs to know which tasks in the program can be executed and accelerated by which accelerator cores and their execution costs. However, it is difficult to develop a compiler that can deal with every type of accelerator core. In the proposed scheme, special purpose compilers for the accelerators developed by their vendors is used to find the tasks which can be accelerated, to calculate the execution costs, and to generate object code for the accelerator cores.

Fig. 2 shows a compilation flow for heterogeneous multicore processors. First, special purpose compilers for accelerator cores are used to generate source code with directives indicating task execution costs on the accelerator. This source file is input to OSCAR heterogeneous parallelizing compiler. OSCAR compiler parallelizes and optimizes the program, then generates parallel execution code for the heterogeneous multicore processor. To generate the execution code, OSCAR compiler uses the object codes generated by the special purpose compilers for accelerator cores. This paper focuses on OSCAR parallelizing compiler and the compiler cooperative heterogeneous multicore architecture assuming that the special purpose compilers are supplied by the vendors and can be used.

4 Coarse Grain Task Parallel Processing on a Heterogeneous Multicore Processor

This section presents coarse grain task parallel processing on a heterogeneous multicore processor. In this scheme, OSCAR compiler generates coarse grain tasks (Macro-Tasks, MTs) hierarchically and decides the hierarchical processor grouping to utilize parallelism among them.

4.1 Coarse Grain Task Parallel Processing

For coarse grain task parallel processing, the compiler decomposes the target program into three kinds of coarse grain tasks (Macro-Tasks, MTs), such as a block of pseudo assignment statements (BPA), a repetition block (RB), a subroutine block (SB). After generation of Macro-Tasks from the source program, the data dependencies and control flow among Macro-Tasks are analyzed, and Macro-Flow Graphs (MFGs) are generated. A Macro-Flow Graph (Fig. 3a) represents control flow and data dependencies among Macro-Tasks. Nodes represent Macro-Tasks, solid edges represent data dependencies among Macro-Tasks, and dotted edges represent control flow. A small circle inside a node represents a

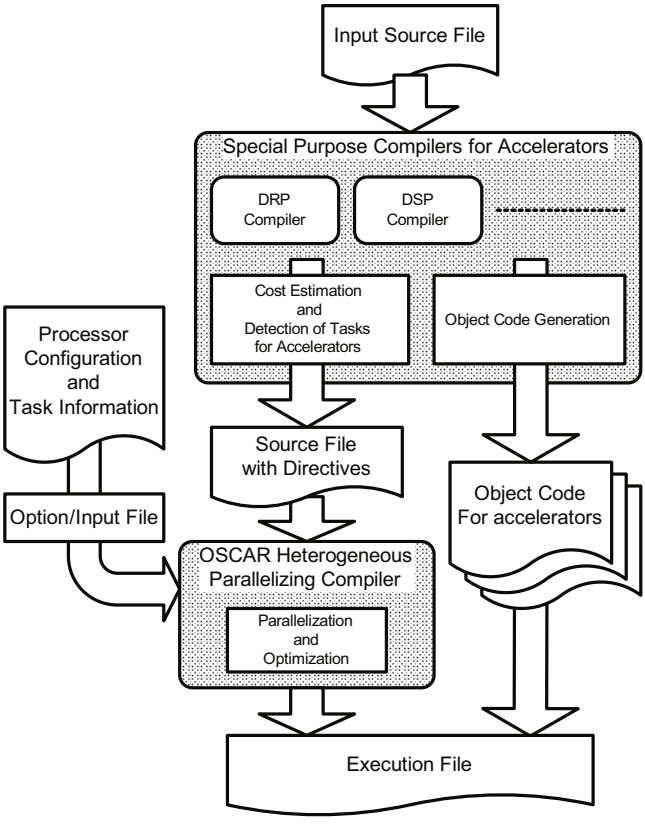

Fig. 2. A Compilation Flow for Heterogeneous Multicores

conditional branch inside the Macro-Task. Though arrows of edges are omitted in the Macro-Flow Graph, it is assumed that the directions are downward.

Then, to extract coarse grain task parallelism among Macro-Tasks from the Macro-Flow Graph, Earliest Executable Condition Analysis[27] is applied to the Macro-Flow Graph and a Macro-Task Graph (MTG) is generated (Fig. 3b). Macro-Task Graphs represent coarse grain task parallelism among Macro-Tasks. Nodes represent Macro-Tasks. A small circle inside a node represents conditional branches. Solid edges represent data dependencies. Dotted edges represent extended control dependencies. Extended control dependency means ordinary normal control dependency and the condition on which a data dependence predecessor of a Macro-Task is not executed. Solid and dotted arcs connecting solid and dotted edges have two different meanings. A solid arc represents that edges connected by the arc are in AND relationship. A dotted arc represents that edges connected by the arc are in OR relationship. Though arrows of edges are omitted assuming downward direction, edges having arrow represents original control flow edges, or branch direction in Macro-Flow Graph. If SB or RB has nested inner layer, Macro-Tasks and Macro-Task Graphs are generated hierarchically.

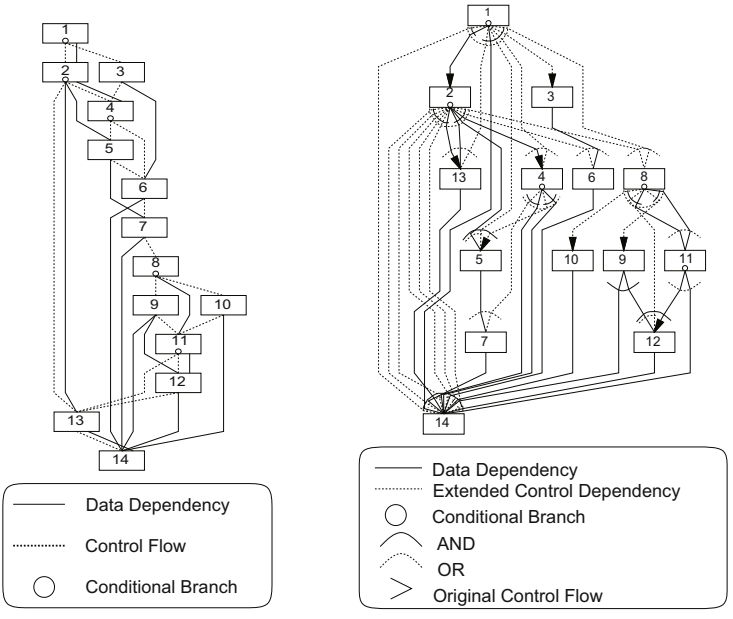

Fig. 3. Examples of Macro-Flow Graph (MFG) and Macro-Task Graph (MTG)

After generation of Macro-Tasks and Macro-Task Graphs, Macro-Tasks are assigned to Processor-Groups (PGs). A Processor-Group is a group of processor elements (PEs), and its grouping is performed logically. If the Macro-Task Graph has only data dependencies,the compiler schedules Macro-Tasks to Processor-Groups at compile time (static scheduling). The static scheduling scheme can minimize data transfer, task assignment and synchronization overhead. If the Macro-Task Graph has conditional branches among Macro-Tasks, the dynamic scheduling is applied. In this case, the compiler generates scheduling codes to assign Macro-Tasks to Processor-Groups at run-time[24, 26] though only the static scheduling scheme is used in this paper.

In addition, if SB or RB has coarse grain task parallelism inside them, PEs are grouped hierarchically and hierarchical parallelism is utilized. OSCAR compiler applies the automatic processor grouping scheme[28, 29] to decide the layers to be applied coarse grain task parallel processing and its processor grouping.

4.2 Hierarchical Processor Grouping Considering OSCAR Heterogeneous Multicore Architecture

To apply coarse grain task parallel processing with static scheduling effectively, processor elements (PEs) are grouped into Processor-Groups (PGs) hierarchically

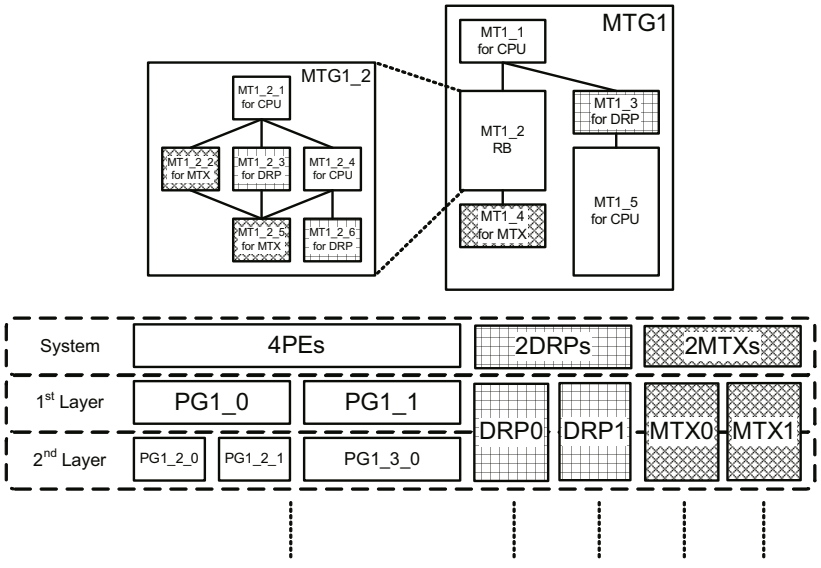

Fig. 4. An Example of Processor Grouping considering OSCAR Heterogeneous Multicore Architecture

while considering coarse grain task parallelism. General purpose PEs on a heterogeneous multicore processor are hierarchically grouped, while considering nested parallelism of the source program or nested structure of Macro-Task Graphs. Different from general purpose PEs, accelerator PEs are not grouped hierarchically, since the number of accelerator PEs is usually less than the number of the general purpose PEs and not enough to be assigned to deeply nested Macro-Task Graphs. In the proposed scheme, accelerator PEs are free from hierarchical grouping of general purpose PEs, and can be used by any nested layers of Macro-Task Graphs efficiently. After the processor grouping, OSCAR compiler schedules Macro-Tasks (MTs) to Processor-Groups or to accelerator PEs at compile time.

Fig. 4 shows an example of hierarchical processor grouping for a heterogeneous multicore processor. In this example, the multicore system has four general purpose PEs and two kinds of accelerator PEs such as two DRP cores and two matrix processor cores (MTXs)[16]. In the 1st layer Macro-Task Graph (MTG1), general purpose PEs are grouped into two Processor-Groups (PG1_0 and PG1_1) and Macro-Tasks (MTs) in this layer is assigned to these Processor-Groups. While the MT1_2 having coarse grain task parallelism internally is executed on PG1_0, PG1_0 is grouped into two Processor-Groups (PG1_2_0 and PG1_2_1) hierarchically. DRPs and MTXs are grouped according to their types or functionality, and can accept requests to execute Macro-Tasks from any Macro-Task Graph.

5 A Static Scheduling Algorithm for OSCAR Heterogeneous Multicore Architecture

This section presents a static scheduling scheme for coarse grain tasks on OSCAR heterogeneous multicore architecture. In this scheme, the compiler schedules Macro-Tasks to Processor-Groups composed of general purpose PEs and accelerator PEs to minimize execution time while considering load balancing and data transfer timing.

5.1 A Task Scheduling Scheme for Heterogeneous Multicores

To schedule Macro-Tasks in the input program to heterogeneous cores, the characteristics of Macro-Tasks and PEs must be considered. For example, some Macro-Tasks are assignable to accelerator PEs (PEs having accelerator cores), however, the other Macro-Tasks cannot be executed on accelerator PEs. In most cases, the assignable Macro-Tasks result in highly effective execution if they are assigned to accelerators. Different from accelerator PEs, general purpose PEs (PEs having only general purpose processor cores) can execute all Macro-Tasks.

The task scheduling algorithm for heterogeneous multicore implemented in OSCAR compiler consists of eight steps:

Step 1. Preparation.
- **Step 1-1.** Calculate each Macro-Task cost on a general purpose PE and each assignable accelerator PE.
- **Step 1-2.** Calculate scheduling priority of each Macro-Task (see Section 5.2 for more details).

Step 2. Initialization.
- **Step 2-1.** Set the scheduling time to zero.
- **Step 2-2.** Add the 1st layer Macro-Task Graph to the list of Macro-Task Graphs under the scheduling process.

Step 3. Extracting ready tasks.

Extract ready Macro-Tasks from the Macro-Task Graphs in the list of Macro-Task Graphs under the scheduling process. Ready Macro-Tasks are Macro-Tasks that satisfy the following requirements at current scheduling time :
- satisfying Earliest Executable Condition[27]
- having a Processor-Group or an assignable accelerator PE which is free at current scheduling time

If there is no ready Macro-Task, then go to Step 8.

Step 4. Task selection.

Select a Macro-Task to be scheduled (target Macro-Task) from the ready Macro-Tasks according to the priorities.

Step 5. Completion time estimation.

Estimate execution completion time of the target Macro-Task on
- each Processor-Group which is free at current scheduling time
- each accelerator PE which can execute the target Macro-Task

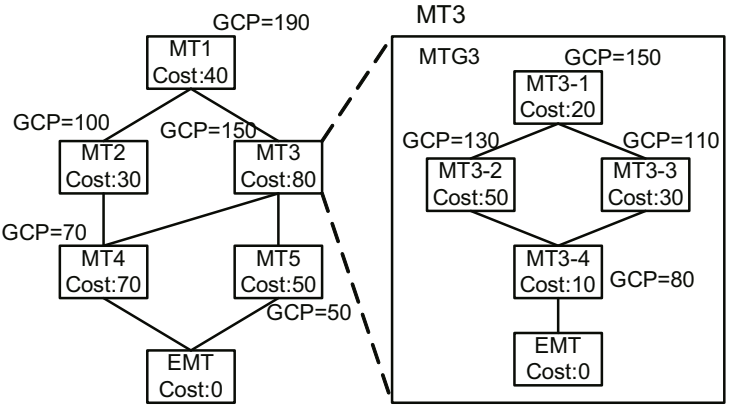

Fig. 5. Global CP Length

(To estimate the completion time, data transfer timings are calculated as mentioned in Section 5.3 and 5.4.)

Step 6. Macro-Task assignment.
Assign the target Macro-Task to the Processor-Group or the accelerator PE gives the earliest completion time.

Step 7. Examination of inside of the Macro-Task.
If the assigned Macro-Task has Macro-Task Graphs to be applied coarse grain parallel processing inside, add it to the list of Macro-Task Graphs in the scheduling process.

Step 8. Updating the scheduling time.

Step 8-1. Update the scheduling time until the time when a Macro-Task is completed next.

Step 8-2. If there is a Macro-Task Graph that all of the Macro-Tasks inside have been completed at the updated scheduling time, remove it from the list of Macro-Task Graphs under the scheduling process.

Step 8-3. If all of the Macro-Tasks are completed, then exit. If not, then go to Step 3.

5.2 A Scheduling Priority

In this paper, global critical path length (global CP length, GCP) is used as the scheduling priority. Global CP length is the longest path length from the exit node of main program to each Macro-Task calculated considering hierarchical Macro-Task Graph structure. Fig. 5 is an example of hierarchical Macro-Task Graphs and global critical path length (GCP) of each Macro-Task (MT). For example, the longest path length from the exit node (EMT) of MTG3 to MT3-3 is 40, and the longest path length from the exit node of the main program to the end of MT3 is 70. Therefore, the longest path length, or global critical path length of MT3-3 is 110, or 40 + 70.

5.3 Estimating Completion Time of a Macro-Task

In the proposed task scheduling scheme, OSCAR compiler estimates completion time of a Macro-Task considering characteristics of the Macro-Task (MT) and processor elements (PEs), and overlapping data transfers with task executions. OSCAR compiler estimates a Macro-Task cost by adding up the costs of instructions inside the task. In the case of loops, the number of iterations is estimated using initial and final value of the loop index, or the size of array accessed in the task. If more precise costs are required, profiling results may be used. In both cases, OSCAR compiler can schedule Macro-Tasks effectively if relative costs among Macro-Tasks are correct.

Before estimating the completion time of a Macro-Task, the completion time of data transfers needed by the target Macro-Task DT_{end} is estimated as:

$$DT_{end} = \max[DT_{load}, \max_{MT_p \in PRED}\{DT_{send}(p)\}],$$

where $PRED$ is the set of predecessors of the target Macro-Task, $DT_{send}(p)$ is the completion time of data transfers from a predecessor MT_p, and DT_{load} is the completion time of data loadings from CSM. These data transfers are scheduled while considering overlapping with task executions (section 5.4). Then, completion time of the target Macro-Task (MT_{fin}) is estimated as:

$$MT_{fin} = \max(T_{free}, DT_{end}) + COST_{MT} + DT_{store},$$

where T_{free} is the time that the Processor-Group or the accelerator PE comes to be free, DT_{end} is the completion time of the data transfers, $COST_{MT}$ is the execution cost of the target Macro-Task on the Processor-Group or the accelerator PE, and DT_{store} is the cost of the data storing to CSM. If the target Macro-Task has Macro-Task Graphs inside it, its completion time is estimated by applying this scheduling scheme to the Macro-Task Graphs inside recursively.

5.4 Overlapping Data Transfers and Task Executions

On a heterogeneous multicore architecture proposed in Section 2, data can be transferred by the data transfer unit (DTU)[13] effectively. In current implementation, the data transfer unit is driven at the beginning or the end of a Macro-Task execution. Once a data transfer is driven, the data transfer is performed asynchronously with task executions on CPU cores or accelerator cores. OSCAR compiler searches data transfer timing while considering the statuses of interconnection network and memory ports. Data loadings from centralized shared memory and data sending among processor elements are scheduled as earlier as possible while considering overlapping with task executions, and data storings to centralized shared memory are scheduled to be overlapped with task executions if possible.

Fig. 6 shows an example of the scheduling result with data transfers. This figure shows the case that one general purpose PE (CPU0) and one accelerator

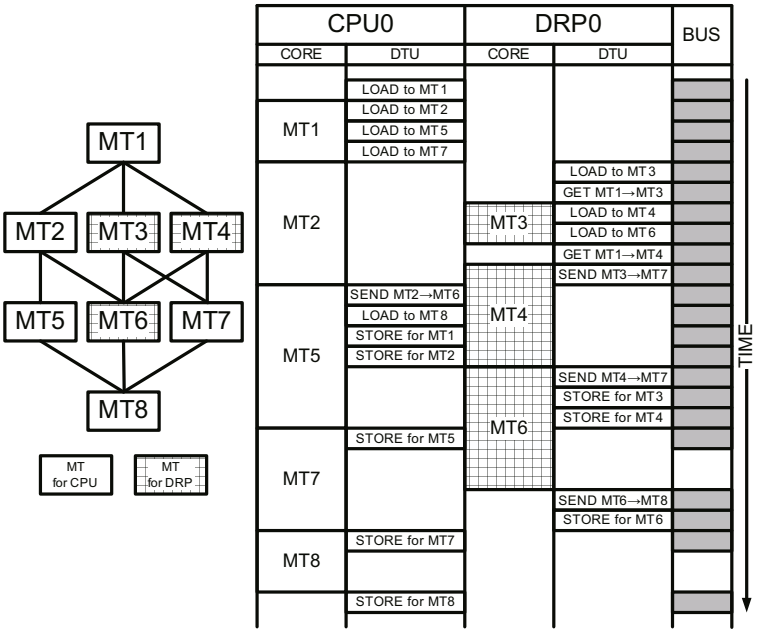

Fig. 6. An Example of Task Scheduling Result considering Overlapping Data Transfers and Task Executions

PE with DRP (DRP0) are connected with single bus. In this example, "LOAD" means data loading from centralized shared memory (CSM), "STORE" means data storing to CSM, "SEND" means data storing/sending to remote distributed shared memory (DSM) and "GET" means data loading/getting from remote DSM. For example, data loading from CSM to the PE that executes Macro-Task named MT4 ("LOAD to MT4") is overlapped with the execution of MT3.

6 Performance Evaluation

This section evaluates the performance of the proposed heterogeneous multicore architecture and task scheduling algorithm.

6.1 The Evaluated Multicore Architecture

In this evaluation, the heterogeneous multicore processor having up to eight PEs on a chip is used. SH4A[30] processor cores are used as the general purpose processor cores and controllers of accelerator PEs, and FE-GAs[13, 31] are used as DRP cores of accelerator PEs. Triple buses are used as the interconnection network, and centralized shared memory has four memory banks. Table 1 shows the minimum access costs of distributed shared memory (DSM), local data memory(LDM), and centralized shared memory (CSM). Local DSM access needs 1

Table 1. Minimum Access Costs for LDM, DSM, and CSM

DSM	1 Clock Cycle
DSM (Remote)	4 Clock Cycles
LDM	1 Clock Cycle
CSM	16 Clock Cycles (off-chip) 4 Clock Cycles (on-chip)

Table 2. MP3 Encode Parameters

# of frames evaluated	16 frames of Stereo PCM
Sample Rate	44.1 [kHz]
Bit Rate	128 [kbps]

clock cycle, remote DSM access needs 4 clock cycles, LDM access needs 1 clock cycle, on-chip CSM access needs 4 clock cycles and off-chip CSM access needs 16 clock cycles at 300MHz. A clock accurate simulator of the heterogeneous multicore architecture is used for this evaluation.

6.2 The Evaluated Application

In this evaluation, an MP3 encoder program written in FORTRAN77 is used. This program is implemented based on the "UZURA MPEG1/LayerIII encoder in FORTRAN90"[32]. Tasks can be executed by accelerator PEs are specified by compiler directives. OSCAR compiler[24] extracts parallelism from the sequential program and schedules its coarse grain tasks to the general purpose processor PEs, namely SH4A cores, and accelerator PEs with FE-GA cores. The encode parameters are shown in Table 2. The input PCM data is allocated on the centralized shared memory initially. Profiling results are used for the task scheduling. Only the main encoding loop is measured to reduce the influence of I/O and calculation of the initial values.

The Structure and Parallelism of an MP3 Encoder. Fig. 7a shows the program structure of MP3 encoder. The MP3 encoder program consists of Sub-Band Analysis, MDCT, Psycho-Acoustic Analysis, Quantization, and Huffman Coding. In the MDCT, the result from the sub-band analysis of the previous frame is used. In psycho-acoustic analysis, the result from the psycho-acoustic analysis of the previous frame is used. Except these stages that need to deliver data among frames, multiple frames can be encoded at the same time. In the program used for this evaluation, 16 frames of PCM data are encoded in the same iteration of main-loop of encoding (Fig. 7b). In this evaluation, it is assumed that local memory (local data memory and distributed shared memory) size is large enough to store the data to encode the 16 frames.

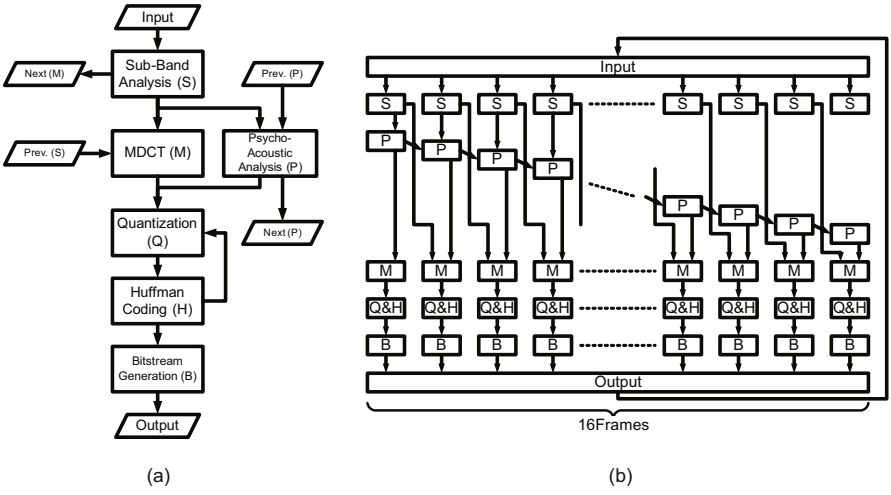

Fig. 7. The Program Structure and the Task Graph of the Evaluated MP3 Encoder

Code Mapping for the ALU Array of FE-GA. In this evaluation, "P", "M", "Q&H" and a part of "S" (Fig. 7b) can be executed and accelerated by accelerator PEs having FE-GAs. Their execution costs on accelerator PEs are precisely estimated in clock level using the amount of input data and the code mapping for the ALU array on FE-GA.

FE-GA core has 24 ALU cells and 8 MULT cells, and these cells are connected to the memory via a crossbar switch. Fig. 8 shows a part of code mapping of MP3 sub-band analysis to FE-GA. The k's loop can be accelerated with the FE-GA's ALU array (Fig. 8a). In the code mapping of this part (Fig. 8b), the k's loop is divided into two parts for the effective use of ALU/MULT cells.

Estimated speedup on accelerator PEs is shown in Fig. 9. The estimated speedup is the average of the sixteen frames used in this evaluation. A part of sub-band analysis and psycho-acoustic analysis are comparatively simple processions, which give us 67.06 times speedup and 61.76 times speedup against a single general purpose PE, respectively. Because the task of Quantization and Huffman Coding ("Q&H" in Fig. 7b) is complex, it gives us 4.36 times speedup. On the average, 6.73 times speedup is given by an accelerator PE with FE-GA core against a general purpose PE.

6.3 Performance on OSCAR Heterogeneous Multicore by OSCAR Compiler

The parallel processing performance of an MP3 encoder on the OSCAR heterogeneous multicore by OSCAR heterogeneous parallelizing compiler is shown in Fig. 10. The horizontal axis shows the configurations of the processor cores and centralized shared memory (namely, on-chip CSM or off-chip CSM). The vertical axis shows the speedup against the sequential execution using one general

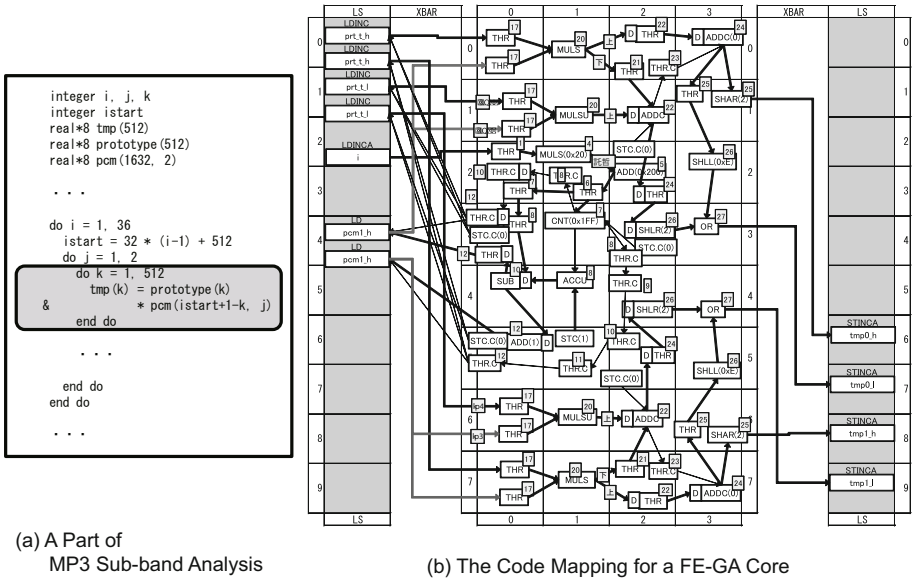

Fig. 8. A Sample Code to be Accelerated by FE-GA and its Mapping for ALU Array of FE-GA

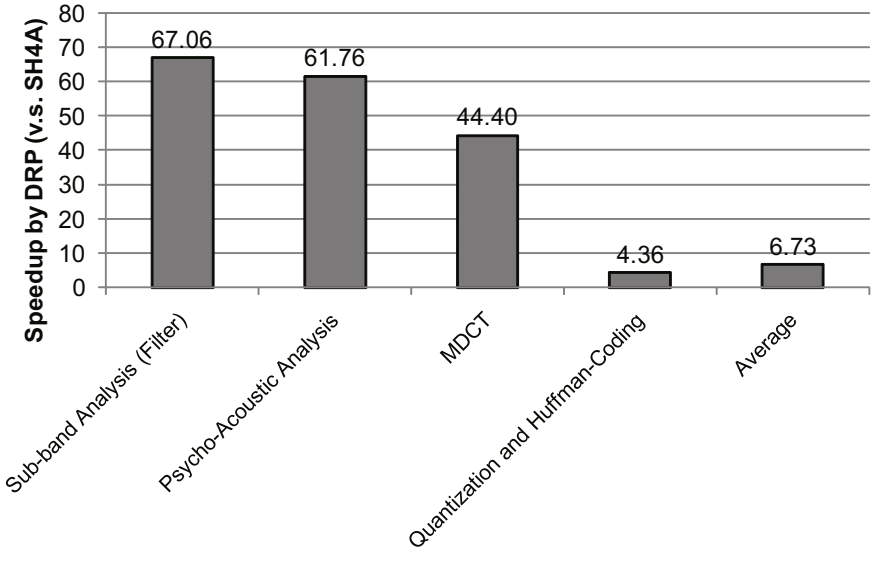

Fig. 9. Speedup of the MP3 Encoder by an Accelerator PE (DRP)

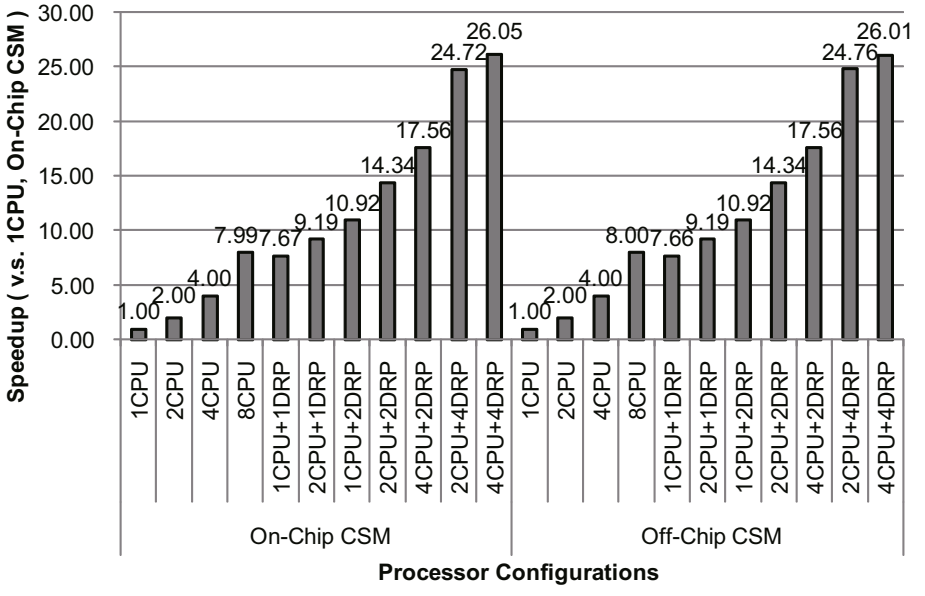

Fig. 10. The Performance of the MP3 Encoder on the OSCAR Heterogeneous Multicore using OSCAR Compiler

purpose PE and on-chip CSM. "nCPU+mDRP" are the numbers of general purpose PEs (CPUs in the figure) and accelerator PEs with FE-GAs (DRPs). "On-Chip" means on-chip CSM and "Off-Chip" means off-chip CSM.

The configurations of 2CPU, 4CPU and 8CPU with on-chip CSM give us 2.00, 4.00 and 7.99 times speedup against 1CPU, respectively. The proposed heterogeneous parallelizing compilation scheme with architecture supports is effective even on a homogeneous multicore environment. With heterogeneous configurations, the configurations of 1CPU+1DRP, 2CPU+1DRP, 1CPU+2DRP, 2CPU+2DRP, 4CPU+2DRP, 2CPU+4DRP and 4CPU+4DRP give 7.67, 9.19, 10.92, 14.34, 17.56, 24.72 and 26.05 times speedup against 1CPU, respectively. Effective use of accelerator PEs gives a much higher performance compared with the number of PEs.

Even if the CSM is out of the chip, the homogeneous configurations of 2CPU, 4CPU and 8CPU give us 2.00, 4.00, 8.00 times speedup against 1CPU, respectively, and the heterogeneous configurations of 1CPU+1DRP, 2CPU+1DRP, 1CPU+2DRP, 2CPU+2DRP, 4CPU+2DRP, 2CPU+4DRP and 4CPU+4DRP give us 7.66, 9.19, 10.92, 14.34, 17.56, 24.76 and 26.01 times speedup. This is because the data are effectively assigned to local memories (local data memories and distributed shared memories) and data transfers are overlapped with task execution with the use of data transfer units (DTUs), or DMA controllers.

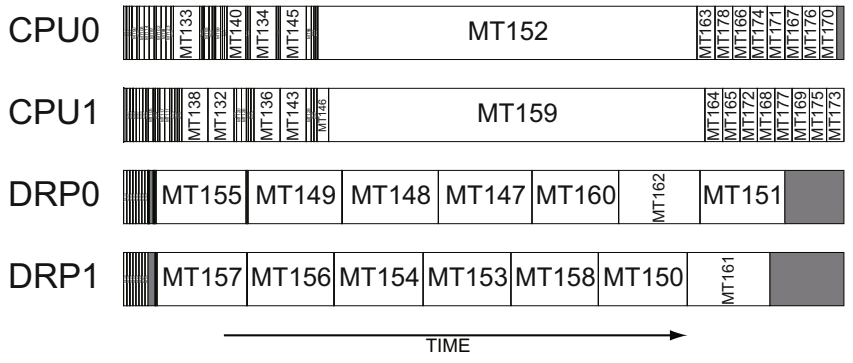

Fig. 11. The Execution Trace of the Configuration of 2CPU+2DRP with off-chip CSM

Fig. 11 shows the execution trace of 2CPU+2DRP with off-chip CSM. MT152 and MT159 are the Macro-Tasks of Quantization and Huffman Coding ("Q&H" in Fig 7b) and executed by the general purpose PEs, though they can be assigned to accelerator PEs. If they are assigned to accelerator PEs, the total processing time of the program becomes larger than the schedule. In the proposed compilation scheme, a Macro-Task, which might require more processing time if it is assigned to an accelerator PE, is automatically assigned to a general purpose PE to utilize CPUs and accelerators effectively.

7 Conclusions

This paper has proposed OSCAR heterogeneous multicore architecture and an automatic parallelizing compilation scheme using static coarse grain task scheduling. The performance is evaluated on the heterogeneous multicore processor with low power SH4A processor cores as general purpose PEs (CPUs), and FEGA dynamically reconfigurable processor (DRP) cores as accelerator PEs using an MP3 encoder implemented based on "UZURA MPEG1/LayerIII encoder in FORTRAN90". In this evaluation, the heterogeneous configurations give us 14.34 times speedup with two CPUs and two DRPs and 26.05 times speedup with four CPUs and four DRPs against sequential execution on one CPU with the on-chip centralized shared memory (CSM). Also, with the off-chip CSM, the heterogeneous multicore give us 14.34 times speedup with two CPUs and two DRPs and 26.01 times speedup with four CPUs and four DRPs by data localization to local data memories and distributed shared memories and data transfer overlapping using intelligent DMA controllers.

References

1. Hammond, L., Hubbert, B.A., Siu, M., Prabhu, M.K., Chen, M., Olukotun, K.: The stanford hydra CMP. IEEE Micro 20, 71–84 (2000)
2. ARM Limited: ARM11 MPCore Processor Technical Reference Manual (2005)

3. Friedrich, J., McCredie, B., James, N., Huott, B., Curran, B., Fluhr, E., Mittal, G., Chan, E., Chan, Y., Plass, D., Chu, S., Le, H., Clark, L., Ripley, J., Taylor, S., Dilullo, J., Lanzerotti, M.: Design of the Power6 microprocessor. In: Digest of Technical Papers of the 2007 IEEE International Solid-State Circuits Conference, pp. 96–97 (February 2007)
4. Taylor, M.B., Kim, J., Miller, J., Wentzlaff, D., Ghodrat, F., Greenwald, B., Hoffman, H., Johnson, P., Lee, J.W., Lee, W., Ma, A., Saraf, A., Seneski, M., Shnidman, N., Strumpen, V., Frank, M., Amarasinghe, S., Agarwal, A.: The raw microprocessor: A computational fabric for software circuits and general purpose programs. IEEE Micro 22, 25–35 (2002)
5. Sankaralingam, K., Nagarajan, R., Liu, H., Kim, C., Huh, J., Burger, D., Keckler, S.W., Moore, C.R.: Exploiting ILP, TLP, and DLP with the polymorphous TRIPS architecture. In: Proceedings of the 30th Annual International Symposium on Computer Architecture, pp. 422–433 (June 2003)
6. Shiota, T., Kawasaki, K., Kawabe, Y., Shibamoto, W., Sato, A., Hashimoto, T., Hayakawa, F., Tago, S., Okano, H., Nakamura, Y., Miyake, H., Suga, A., Takahashi, H.: A 51.2GOPS 1.0GB/s-DMA single-chip multi-processor integrating quadruple 8-Way VLIW processors. In: Digest of Technical Papers of the 2005 IEEE International Solid-State Circuits Conference, pp. 194–593 (February 2005)
7. Sohi, G.S., Breach, S.E., Vijaykumar, T.N.: Multiscalar processors. In: Proceedings of 22nd Annual International Symposium on Computer Architecture, pp. 414–425 (June 1995)
8. Vangal, S., Howard, J., Ruhl, G., Dighe, S., Wilson, H., Tschanz, J., Finan, D., Iyer, P., Singh, A., Jacob, T., Jain, S., Venkataraman, S., Hoskote, Y., Borkar, N.: An 80-Tile 1.28TFLOPS network-on-chip in 65nm CMOS. In: Digest of Technical Papers of the 2007 IEEE International Solid-State Circuits Conference, pp. 98–589 (February 2007)
9. Seiler, L., Carmean, D., Sprangle, E., Forsyth, T., Abrash, M., Dubey, P., Junkins, S., Lake, A., Sugerman, J., Cavin, R., Espasa, R., Grochowski, E., Juan, T., Hanrahan, P.: Larrabee: A many-core x86 architecture for visual computing. ACM Transactions on Graphics 27(3) (2008)
10. 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: Digest of Technical Papers of the 2005 IEEE International Solid-State Circuits Conference, pp. 184–592 (February 2005)
11. Khailany, B., Williams, T., Lin, J., Long, E., Rygh, M., Tovey, D., Dally, W.J.: A programmable 512 GOPS stream processor for signal, image, and video processing. In: Digest of Technical Papers of the 2007 IEEE International Solid-State Circuits Conference, pp. 272–602 (February 2007)
12. Torii, S., Suzuki, S., Tomonaga, H., Tokue, T., Sakai, J., Suzuki, N., Murakami, K., Hiraga, T., Shigemoto, K., Tatebe, Y., Ohbuchi, E., Kayama, N., Edahiro, M., Kusano, T., Nishi, N.: A 600MIPS 120mW 70μA leakage triple-CPU mobile application processor chip. In: Digest of Technical Papers of the 2005 IEEE International Solid-State Circuits Conference, pp. 136–589 (February 2005)

13. Ito, M., Todaka, T., Tsunoda, T., Tanaka, H., Kodama, T., Shikano, H., Onouchi, M., Uchiyama, K., Odaka, T., Kamei, T., Nagahama, E., Kusaoke, M., Nitta, Y., Wada, Y., Kimura, K., Kasahara, H.: Heterogeneous multiprocessor on a chip which enables 54x AAC-LC stereo encoding. In: Proceedings of the 2007 IEEE Symposium on VLSI Circuits, pp. 18–19 (June 2007)
14. Kumar, R., Tullsen, D.M., Ranganathan, P., Jouppi, N.P., Farkas, K.I.: Single-ISA heterogeneous multi-core architectures for multithreaded workload performance. In: Proceedings of the 31st Annual International Symposium on Computer Architecture, pp. 64–75 (June 2004)
15. Shikano, H., Suzuki, Y., Wada, Y., Shirako, J., Kimura, K., Kasahara, H.: Performance evaluation of heterogeneous chip multi-processor with MP3 audio encoder. In: Proceedings of the IEEE Symposium on Low-Power and High Speed Chips, pp. 349–363 (April 2006)
16. Noda, H., Tanizaki, T., Gyohten, T., Dosaka, K., Nakajima, M., Mizumoto, K., Yoshida, K., Iwao, T., Nishijima, T., Okuno, Y., Arimoto, K.: The circuits and robust design methodology of the massively parallel processor based on the matrix architecture. In: Digest of Technical Papers of the 2006 Symposium on VLSI Circuits, pp. 210–211 (2006)
17. NVIDIA Corporation: NVIDIA CUDA Compute Unified Device Architecture Programming Guide (2008)
18. Xie, T., Qin, X.: Stochastic scheduling with availability constraints in heterogeneous clusters. In: Proceedings of the 2006 IEEE International Conference on Cluster Computing, pp. 1–10 (September 2006)
19. Sih, G.C., Lee, E.A.: A compile-time scheduling heuristic for interconnection-constrained heterogeneous processor architectures. IEEE Transactions on Parallel and Distributed Systems 4, 175–187 (1993)
20. Chan, W.Y., Li, C.K.: Scheduling tasks in DAG to heterogeneous processor system. In: Proceedings of the 6th Euromicro Workshop on Parallel and Distributed Processing, pp. 27–31 (January 1998)
21. Topcuoglu, H., Hariri, S., Wu, M.Y.: Performance-effective and low-complexity task scheduling for heterogeneous computing. IEEE Transactions on Parallel and Distributed Systems 13, 260–274 (2002)
22. Kasahara, H., Honda, H., Narita, S.: Parallel processing of near fine grain tasks using static scheduling on OSCAR (Optimally SCheduled Advanced multiprocessoR). In: Proceedings of Supercomputing '90, pp. 856–864 (November 1990)
23. Kimura, K., Kodaka, T., Obata, M., Kasahara, H.: Multigrain parallel processing on OSCAR CMP. In: Proceedings of the 2003 International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems (January 2003)
24. Ishizaka, K., Miyamoto, T., Shirako, J., Obata, M., Kimura, K., Kasahara, H.: Performance of OSCAR multigrain parallelizing compiler on SMP servers. In: Proceedings of the 17th International Workshop on Languages and Compilers for Parallel Computing (September 2004)
25. Kimura, K., Wada, Y., Nakano, H., Kodaka, T., Shirako, J., Ishizaka, K., Kasahara, H.: Multigrain parallel processing on compiler cooperative chip multiprocessor. In: Proceedings of the 9th Annual Workshop on Interaction between Compilers and Computer Architectures, pp. 11–20 (February 2005)
26. Kasahara, H., Ogata, W., Kimura, K., Matsui, G., Matsuzaki, H., Okamoto, M., Yoshida, A., Honda, H.: OSCAR multi-grain architecture and its evaluation. In: Proceedings of the 1997 International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems, pp. 106–115 (October 1997)

27. Kasahara, H., Honda, H., Mogi, A., Ogura, A., Fujiwara, K., Narita, S.: A multigrain 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)
28. Obata, M., Shirako, J., Kaminaga, H., Ishizaka, K., Kasahara, H.: Hierarchical parallelism control for multigrain parallel processing. In: Pugh, B., Tseng, C.-W. (eds.) LCPC 2002. LNCS, vol. 2481, pp. 31–44. Springer, Heidelberg (2005)
29. Shirako, J., Nagasawa, K., Ishizaka, K., Obata, M., Kasahara, H.: Selective inline expansion for improvement of multi grain parallelism. In: The IASTED International Conference on Parallel and Distributed Computing and Networks, pp. 128–134 (February 2004)
30. 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. In: Digest of Technical Papers of the 2007 IEEE International Solid-State Circuits Conference, pp. 100–590 (February 2007)
31. Kodama, T., Tsunoda, T., Takada, M., Tanaka, H., Akita, Y., Sato, M., Ito, M.: Flexible engine: A dynamic reconfigurable accelerator with high performance and low power consumption. In: Proceedings of the IEEE Symposium on Low-Power and High Speed Chips, pp. 393–408 (April 2006)
32. UZURA3: MPEG1/LayerIII encoder in FORTRAN90, `http://members.at.infoseek.co.jp/kitaurawa/index_e.html`