Compiler Control Power Saving Scheme for Multi Core Processors

Jun shirako[†], Naoto Oshiyama[†], Yasutaka Wada[†], Hiroaki Shikano[‡], Keiji Kimura[†‡], and Hironori Kasahara[†‡]

[†]Dept. of Computer Science, [‡]Advanced Chip Multiprocessor Research Institute
Waseda University
3-4-1 Ohkubo, Shinjuku-ku, Tokyo, 169-8555, Japan
{shirako,oshiyama,yasutaka,shikano,kimura,kasahara}@oscar.elec.waseda.ac.jp

Abstract. With the increase of transistors integrated onto a chip, multi core processor architectures have attracted much attention to achieve high effective performance, shorten development period and reduce the power consumption. To this end, the compiler for a multi core processor is expected not only to parallelize program effectively, but also to control the voltage and clock frequency of processors and storages carefully inside an application program. This paper proposes a compilation scheme for reduction of power consumption under the multigrain parallel processing environment that controls Voltage/Frequency and power supply of each processor core on a chip. In the evaluation, the OSCAR compiler with the proposed scheme achieves 60.7 percent energy savings for SPEC CFP95 applu without performance degradation on 4 processors, and 45.4 percent energy savings for SPEC CFP95 tomcatv with real-time deadline constraint on 4 processors, and 46.5 percent energy savings for SPEC CFP95 swim with the deadline constraint on 4 processors.

1 Introduction

According to the increase of transistors integrated onto a chip, a chip multiprocessor architecture, or multicore architecture, that can achieve higher performance and save the power consumption is collecting much attention as future processors. To realize efficient parallel processing on multiprocessor systems, cache and local memory optimization to cope with memory wall problems and minimization of data transfer among processors using DMAC (Direct Memory Access Controller), in addition to the extraction of parallelism from an application program. For the exploitation of parallelism for multiprocessors, there have been a large number of researches in the areas of loop parallelizing compilers [1–3]. However, the loop parallelization techniques are almost matured and new generation of parallelization techniques like multi-grain parallelization are required to attain further speedup. There are a few compilers trying to exploit multiple levels of parallelism, for example, NANOS compiler[4] extracts the multi-level parallelism including the coarse grain task parallelism by using extended OpenMP API and OSCAR multigrain parallelizing compiler [5–7] extracts coarse grain task parallelism among loops, subroutines and basic blocks

and near fine grain parallelism among statements inside a basic block, in addition to the loop parallelism. Also, OSCAR compiler realizes the automatic determination of parallelism of each part of a program and the number of required processors to process the program part efficiently with the global cache memory optimization over different loops.

This required number of processors determination scheme determines the suitable number of processors to execute each part of a program and stops the unnecessary processors to minimize processing overhead and reduce power consumption by shutting off power supply for idle processors.

For the power saving techniques, various methods have been proposed. Adaptive Processing[8] estimates the workload of computing resources using counters for cache misses and instruction queues and powers off unnecessary resources. Online Methods for Voltage and Frequency Control [9] settles on the fitting voltage and frequency for each domain of processors using instruction issue queue occupancies as feedback signals. As the compiler algorithm for CPU energy reduction, compiler-directed DVS(dynamic voltage scaling)[10] is known. This method gets the relations between frequency and execution time for each part of a program by profiling. It solves minimization problem of total energy consumption and determines the suitable frequency for each part.

This paper proposes a static compiler control scheme of power saving for a multi core processor without profiling, which realizes

- power supply cutoff for unnecessary processors
- voltage/frequency(V/F) control of each task or of each processor in an application program under the constraints of the minimum time execution or the satisfaction of real-time deadline

2 Multigrain parallel processing

The proposed power saving scheme is mainly used with the coarse grain task parallelization in the multigrain parallel processing. This section describes the overview of the coarse grain task parallel processing.

2.1 Generating macro-tasks [5–7][11, 12]

In multigrain parallelization, a program is decomposed into three kinds of coarse grain tasks, or macro-tasks, such as block of pseudo assignment statements(BPA) repetition block(RB), subroutine block(SB)[7]. Macro-tasks can be hierarchically defined inside each un-parallelizable repetition block, or sequential loop, and a subroutine block as shown in Figure 1. Repeating the macro-task generation hierarchically, the source program is decomposed into the nested macro-tasks as in Figure 1.

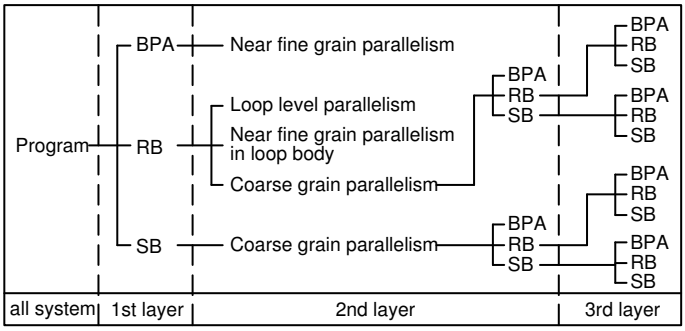

Fig. 1. Hierarchical Macro Task Definition

2.2 Extraction of coarse grain task parallelism

After generation of macro-tasks, the data dependency and the control flow among macro-tasks are analyzed in each nested layer, and hierarchical macro flow graphs(MFG) representing control flow and data dependencies among macro-tasks are generated [5–7]. Then, to extract coarse grain task parallelism among macro-tasks, Earliest Executable Condition analysis [5–7] which analyzes control dependencies and data dependencies among macro-tasks simultaneously is applied to each Macro flow graph. Earliest Executable Conditions are the conditions on which macro-task may begin its execution earliest. By this analysis, a macro-task graph(MTG)[5–7] is generated for each macro flow graph. Macro-task graph represents coarse grain parallelism among macro-tasks.

2.3 Processor groups and Processor elements

To execute hierarchical macro-task graphs efficiently, the compiler groups processors hierarchically. This grouping of processor elements(PEs) into Processor Groups(PGs) is performed logically, and macro-tasks are assigned to processor groups in each layer.

Figure 2 shows an example of a hierarchical processor groups. For execution of a macro-task graph in the 1st nest level, or 1st layer, the 8 processors are grouped into 2 processor groups each of which has 4 processor elements. This is represented as (2PGs, 4PEs). The macro-task graph in the 1st nest level is processed by the 2PGs. For each macro-task graph in the 2nd nest level, 4 processors are available. In the Figure 2, the grouping of (4PGs, 1PE) is chosen for the left PG and (2PGs, 2PEs) is chosen for the right PG.

2.4 Automatic determination scheme of parallelizing layer

In order to improve the performance of multigrain parallel processing, it is necessary to schedule the tasks on the macro-task graph with the extracted parallelism

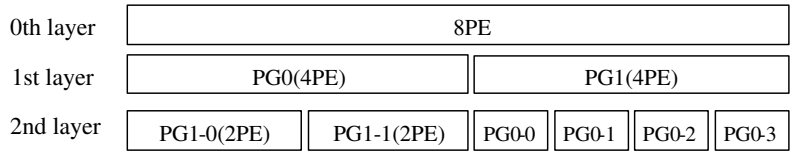

Fig. 2. Hierarchical definition of processor groups and processor elements

to processors the grouped processor layer. OSCAR compiler with the automatic parallelized layer determination scheme [11, 13] estimates the parallelism of each macro-task graph and determine the suitable (PGs, PEs) grouping. This scheme determines the suitable number of processors executing each macro-task, considering trade-off between parallelization and scheduling and data transfer overhead. Therefore, OSCAR compiler doesn't assign tasks to the excessive processors to reduce parallel processing overhead.

2.5 Macro-Task Scheduling

In the coarse grain task parallel processing, a macro-task in the macro-task graph is assigned to a processor group. At this time, static scheduling or dynamic scheduling is chosen for each macro-task graph.

If a macro-task graph has only data dependencies and is deterministic, the static scheduling is selected. In this case, the compiler schedules macro-tasks to processer groups. The static scheduling is effective since it can minimize data transfer and synchronization overhead without runtime scheduling overhead.

If a macro-task graph is un-deterministic by conditional branches among coarse grain tasks, the dynamic scheduling is selected to handle the runtime uncertainties. The dynamic scheduling routines are generated by the compiler and inserted into a parallelized program code to minimize scheduling overhead.

This paper proposes the power saving static scheduling scheme for the determinable macro-task graphs.

In the following sections, MT represents macro-task, MTG is macro-task graph, PG is processor group, PE is processor element, BPA is block of pseudo assignment statements, RB is repetition block and SB is subroutine block.

3 Compiler control power saving scheme

The multigrain parallel processing can take full advantage of multi level parallelism in a program. However, there isn't always enough parallelism in all part of a program for available resources. In such a case, shutting off the power supply to the idle processors, to which tasks are not assigned, can reduce power consumption. Also, execution at lower voltage and frequency may reduce the total energy consumption in real time processing with the deadline constraint. The proposed scheme realizes the following two modes of power reduction. The first

Table 1. The rate of frequency, voltage, dynamic energy and static power

state	FULL	MID	LOW	OFF
frequency	1	1/2	1/4	0
voltage	1	0.87	0.71	0
dynamic energy	1	3/4	1/2	0
static power	1	1	1	0

is the fastest execution mode that doesn't apply the power saving scheme to the critical path of a program to guarantee the fastest processing speed. The second is real-time processing mode with deadline constraint that minimizes the total energy consumption within the given deadline.

3.1 Target model for the proposed power saving scheme

In this paper, it is supposed that the target multi core processors have the following functions with the hardware supports like OSCAR multi core processor shown in Figure 3. The OSCAR(Optimally Scheduled Advanced Multiprocessor) architecture has been proposed to support optimization of multigrain parallelizing compiler [14, 5, 6], especially static and dynamic task scheduling [15, 14, 16]. In the OSCAR architecture, simple processor cores having local and/or distributed shared memory both of which are double mapped to the global address space so that can be accessed by remote processor cores DTC(Data Transfer Controller), or DMAC, are connected by interconnection network like multiple busses or cross bar switches to control shared memory(CSM) [15, 14, 16, 17]. In addition to the traditional OSCAR architecture, in this paper, the following power control functions are supported.

- The frequency for each processor can be changed in several levels individually.
- The voltage can be changed with the frequency.
- Each processor can be powered on and off individually.

There are a lot of approaches for voltage and frequency(V/F) control. The proposed power saving scheme assumes frequency changes discretely, and the optimal voltage is fixed for each frequency. Table 1 shows an example of the combinations of voltage, dynamic energy and static power at each frequency, which supposes FULL is 400MHz, MID is 200MHz and LOW is 100MHz at 90nm technology. For the table, dynamic energy rate for each frequency is the rate of energy consumption to the energy consumption at FULL. The power supply is shut off completely at OFF, then the static power becomes 0. These parameters and the number of frequency states can be changed, according to architectures and technology. This scheme also considers the state transition overhead that is given for each state.

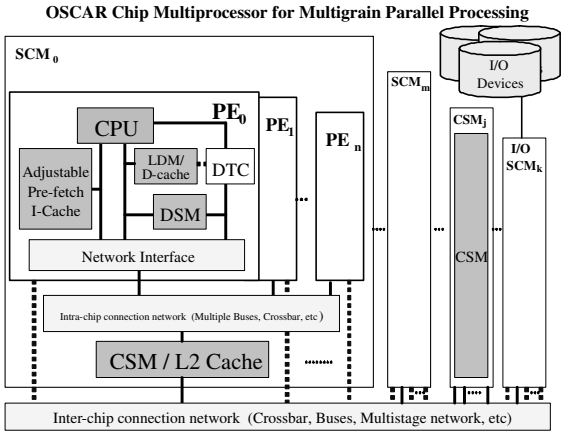

Fig. 3. OSCAR architecture(Chip multiprocessor)

3.2 Target MTG for the proposed control scheme

OSCAR compiler selects dynamic scheduling or static scheduling for each MTG, as to whether there is runtime uncertainty like conditional branches in the MTG. The proposed scheme can be only applied to static scheduled MTGs. However, separating the parts without branches from dynamic scheduled MTG, this scheme is applied for the static scheduling parts of MTGs. In the static scheduling at the compile time, execution cost and consumed energy of each MT is estimated. The cost and energy at each frequency level like "FULL" and "MID" can be calculated using the previously prepared parameter table for each target multicore processor of each instruction cost embedded in the compiler.

3.3 Deadline constraint of target MTG

The proposed scheme determines suitable voltage and frequency for each MT on a MTG based on the result of static task assignment. In other words, the proposed power saving scheme is applied for the static task schedule like Figure 4 generated by static task scheduling algorithms to minimize processing time including data transfer overhead, such as CP/DT/MISF, DT/CP, ETF/CP, which have been used for a long time in OSCAR compiler. Figure 4 shows MTs 1, 2 and 5 are assigned to PG0, MTs 3 and 6 are assigned to PG1, MTs 4, 7 and 8 are assigned to PG2 by the static scheduling algorithms. The best schedule is chosen among different schedules generated by the different heuristic scheduling algorithms. In Figure 4, edges among tasks show data dependence.

First, the following is defined for MT_i, in order to estimate the execution time of the target MTG to which the proposed scheme is applied.

T_i : execution time of MT_i after V/F control

T_{start_i} : start time of MT_i

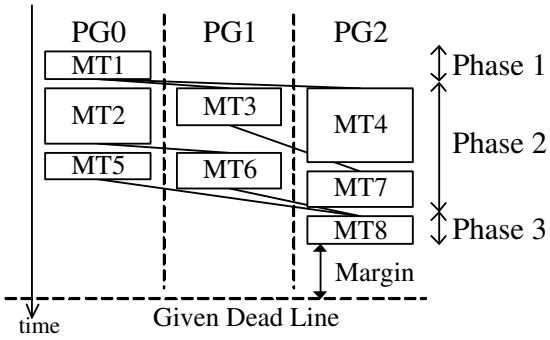

Fig. 4. static scheduled MTG

T_{finish_i} : finish time of MT_i

At the beginning of the proposed scheme, T_i is not yet fixed. The start time of the target MTG is set to 0. If MT_i is the first macro-task executed by a PG and has no data dependent predecessor. T_{start_i} and T_{finish_i} are represented as shown below.

$T_{start_i} = 0$

$T_{finish_i} = T_{start_i} + T_i = T_i$

For instance, the MT_1 is the entry node of MTG, so it is the first and has no data dependent predecessor. Then, $T_{start_1} = 0$, $T_{finish_1} = T_1$. In other case, the previous macro-task which is assigned to the same PG as MT_i is represented as MT_j. The data dependent predecessors of MT_i are defined as $\{MT_k, MT_l, ...\}$. Then, MT_i starts when MT_j, MT_k, MT_l, ... finish.

$T_{start_i} = max(T_{finish_j}, T_{finish_k}, T_{finish_l}, ...)$

$T_{finish_i} = T_{start_i} + T_i$

In Figure 4, MT_2 and MT_3 start execution immediately after the time MT_1 is finished. So, the start time is represented as $T_{start_2} = T_{start_3} = T_{finish_1} = T_1$, the finish time is $T_{finish_2} = T_{start_2} + T_2 = T_1 + T_2$, $T_{finish_3} = T_{start_3} + T_3 = T_1 + T_3$. MT_6 is started after MT_2 and MT_3, then $T_{start_6} = max(T_{finish_2}, T_{finish_3}) = max(T_2 + T_1, T_3 + T_1)$. In addition, the common term of the arguments in max may be put out of max. Then, $T_{start_6} = max(T_2+T_1, T_3+T_1) = max(T_2, T_3)+T_1$. As the same way, the finish time of MT_8 which is the exit node is represented as $T_{finish_8} = T_1 + T_8 + max(T_2 + T_5, T_6 + max(T_2, T_3), T_7 + max(T_3, T_4))$

The exit node is generally represented by

$T_{finish_{exit}} = T_m + T_n + ... + max_1(...) + max_2(...) + ...$

The start time of the entry node is 0, therefore $T_{finish_{exit}}$ expresses the execution time of the target MTG, defined as T_{MTG}. The given deadline for the target MTG is defined as $T_{MTG\_deadline}$. Then, the next condition should be satisfied.

$T_{MTG} \leq T_{MTG\_deadline}$

The proposed scheme determines suitable clock frequency for MT_i to satisfy the condition.

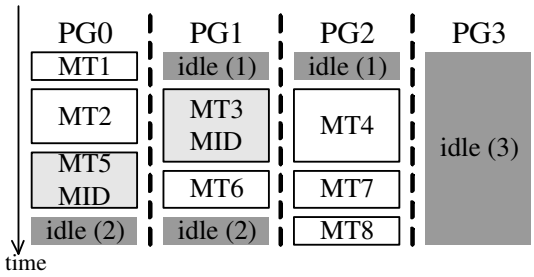

Fig. 5. Result of FV control

3.4 Voltage / frequency control

This paragraph describes how to determine the voltage and frequency to execute each MT using next conditions. The execution time of MT_i is T_i, the execution time of target MTG is T_{MTG}, the real-time deadline of the terget MTG is $T_{MTG\_deadline}$, then

$T_{MTG} = T_m + T_n + ... + max_1 + max_2 + ...$ - - - (a)

$T_{MTG} \leq T_{MTG\_deadline}$ - - - (b)

For sake of simplicity, the MTs corresponding to each term of the expression (a) such as T_m, T_n, ..., max_1, max_2, ... are called Phase. Each term represents the different part of T_{MTG}. Therefore, the different Phase is not executed in parallel on any account as shown in Figure 4. The following parameters for $Phase_i$ at frequency F_n are defined.

$T_{sched_i}(F_n)$: scheduling length at F_n

$Energy_i(F_n)$: energy consumption at F_n

$T_{sched_i}(F_n)$ represents the execution time when the whole $Phase_i$ is processed at F_n. $T_{sched_i}(FULL)$ is the minimum value of the term in the expression (a). $Energy_i(F_n)$ expresses the total energy consumption as $Phase_i$ is excuted at F_n.

Here, it is considered to change frequency from F_n to F_m. The scheduling length is increased from $T_{sched_i}(F_n)$ to $T_{sched_i}(F_m)$. The energy is decreased from $Energy_i(F_n)$ to $Energy_i(F_m)$. Using these values, $Gain_i(F_m)$ is defined as

$$Gain_i(F_m) = -\frac{Energy_i(F_m) - Energy_i(F_n)}{T_{sched_i}(F_m) - T_{sched_i}(F_n)}$$

$Gain_i(F_m)$ represents reduction rate of energy on scheduling length when F_n is changed into F_m. Therefore, if the increases of scheduling length are same, the more energy consumption can be prevented by prioritizing $Phase_i$ with larger $Gain_i(F_m)$.

Next, to estimate the margin of the target MTG, the minimum value of T_{MTG} is calculated. This is equal to the summation of $T_{sched_i}(FULL)$. Then, using this minimum value and $T_{MTG\_deadline}$, the margin T_{MTG_margin} is defined as

$T_{MTG\_margin} = T_{MTG\_deadline} - \sum T_{sched_i}(FULL)$

As the target MTG must finish in minimum execution time, $T_{MTG\_margin} = 0$,

Table 2. Power and frequency transition overhead

dynamic power	220[mW]
static power	2.2[mW]
overhead(FULL - MID - LOW)	0.1[ms]
overhead({FULL, MID, LOW} - OFF)	0.2[ms]

then each Phase has to be executed at FULL. When $T_{MTG\_margin} > 0$, the proposed scheme turns down the voltage and frequency of each Phase, according to $Gain_i(F_m)$. If Phase has a single MT, the frequency of MT is the same as the Phase. If Phase includes some MTs and corresponds to max term, the proposed scheme also defines Phases for each argument of max, then determines clock frequency to execute these Phases. The algorithm to determine frequency for each Phase is described below. The initial value of each frequency is FULL.

Step.1 Determining each frequency of Phase

Step.1.1 selecting target Phase

This step considers only a Phase whose frequency isn't fixed. F_n is represented as current frequency and F_m is defined as one step lower than F_n, then $Phase_i$ having the maximum $Gain_i(F_m)$ is selected as the target Phase. goto **Step.1.2**

Step.1.2 determining effectiveness for target Phase

For target Phase, the conditions to change the frequency from F_n to F_m is as follows.

1. Including the frequency transition overhead, the target Phase can finish at F_m within the T_{MTG_margin}.
2. The energy at F_m with overhead is lower than the energy at F_n.

If both conditions are satisfied,
then the frequency of target Phase is changed to F_m. goto **Step.1.3**
else the frequency of target Phase is confirmed as F_n. goto **Step.1.4**

Step.1.3 updating the margin of MTG

The required time to execute the target Phase at F_m is calculated, then the required time is subtracted from T_{MTG_margin}. If F_m is the lowest frequency, the frequency of target Phase is confirmed as F_m. goto **Step.1.4**

Step.1.4 determining exit

The conditions to exit are as follows.

1. The frequency of all Phase is confirmed.
2. T_{MTG_margin} is 0.

If either of these conditions is satisfied,
then goto **Step.2**
else goto **Step.1.1**

The remained margin is given $Phase_i$ which satisfies next conditions, if T_{MTG_margin} is not 0 at the end.

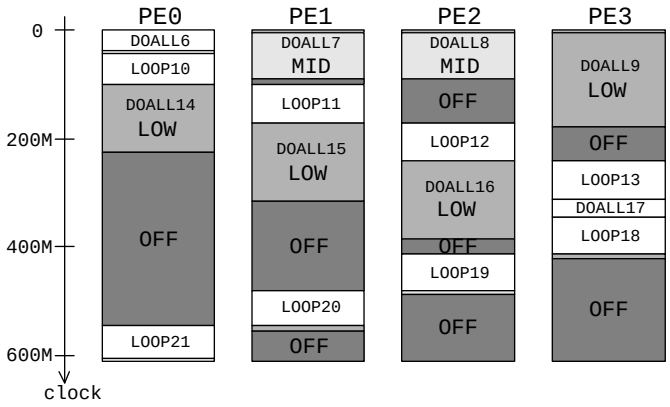

Fig. 6. FV control of applu(4proc.)

– The frequency is not the lowest.
– $Gain_i(F_m)$ is the maximum.

Step.2 Voltage/frequency control within each Phase

In the proposed scheme, the following algorithm is applied to each Phase.

Step.2.1 classifying Phases

If Phase includes only a single MT,

then the frequency of the MT is the same as Phase. **exit**

else goto **Step.2.2**

Step.2.2 Voltage/frequency control of max term

Phase includes some MTs and corresponds to max term, the proposed scheme calculates the executing time of this Phase at the already determined frequency in **Step.1**. Then, the calculated execution time is defined as $T_{max_i\_deadline}$.

$max_i = max(arg_{i\_1}, arg_{i\_2}, ...) \leq T_{max_i\_deadline}$

$arg_{i\_j} = T_{i\_j\_m} + T_{i\_j\_n} + ... + max_{i\_j\_1} + max_{i\_j\_2} + ...$

Therefore, arg_{i_j} should meet the next condition.

$T_{i\_j\_m} + T_{i\_j\_n} ... + max_{i\_j\_1} + max_{i\_j\_2} ... \leq T_{max_i\_deadline}$ - - - (c)

The MTs corresponding to each term in the expression (c) are also considered as Phase, then **Step.1** is applied to determine the frequency of each Phase. At this time, the execution time of each arg_{i_j} at FULL frequency is calculated. Then each arg_{i_j} is applied **Step.1** in descending order of the execution time, or ascending order of the margin. Some Phases in different *arg*s may include the same macro-tasks in common. However, once the frequency of a macro-task has been determined, the frequency isn't changed.

Applying **Step.1** and **Step.2** recursively, the suitable frequency of all MTs are determined.

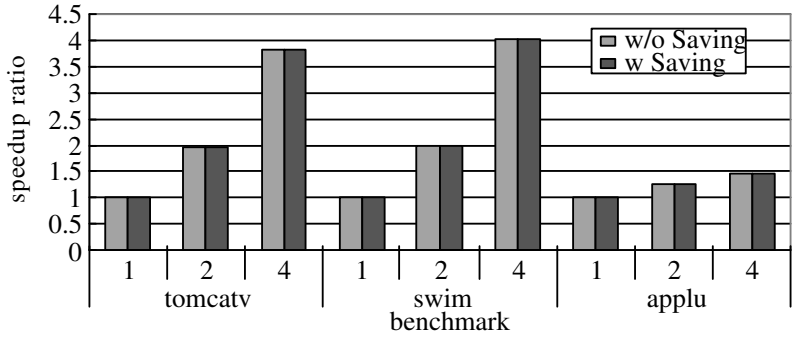

Fig. 7. Speedup in fastest execution mode

3.5 Power supply control

This paragraph explains power supply control to reduce unnecessary energy consumption including static leak current by idle processors. The cases where the idle time occurs in a MTG are,

1. before MT with data dependency is executed,
2. after all MTs in a PG are finished,
3. the idle time created by the determination scheme of parallelizing layer, which is described in paragraph 2.4.

The gray parts of Figure 5 are the idle in each case. Here, the PG3 is the processor group determined as unnecessary. In the idle time which meets the next conditions, the power of the processor is turned off.

- The idle time is longer than the frequency transition overhead.
- The energy becomes lower by power-off.

3.6 Applying power saving scheme to inner MTG

If a MT_i includes a MTG_i inside, it may be more effective to control each MT_{i_j} in MTG_i than to process the whole MT_i at the same clock frequency. Therefore, the deadline for MTG_i is defined as $T_{MTG_i\_deadline}$, which is given by T_i. Then, MTG_i is applied the proposed power saving control described in paragraph 3.4 and 3.5. Comparing both case to execute the whole MT_i at the same frequency and case to apply the power saving control to MTG_i, the more effective one is selected.

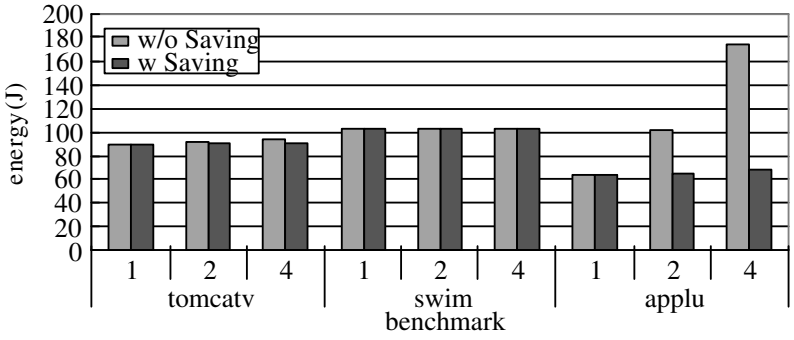

Fig. 8. Energy in fastest execution mode

4 Performance evaluation

This section describes the performance of OSCAR multigrain parallelizing compiler with the proposed power saving scheme. The evaluation are performed by using the static scheduler in the compiler. For this evaluation, the parameters for frequencies, voltages, dynamic energies, and static powers shown in Table 1 are used. In this paper, only energy for processors was evaluated. The state transition overhead with frequency, dynamic and static power is shown in Table 2. The dynamic power at FULL frequency is measured by using Wattch[18]. Cooperative Voltage Scaling[19] is vebered to determine the parameters like the transition overhead, attribute of voltage/frequency and dynamic power at MID and LOW frequency. Application programs, such as applu, tomcatv and swim from SPEC95 CFP, are used in the evaluation. For applu, inline expansion and loop aligned decomposition for the data localization[12] are applied. Also, the main loop in applu is divided into the static part without conditional branch and the dynamic part with branches, in order to apply the proposed scheme.

4.1 Performance in the fastest execution mode

Figure 7 shows the speedup ratio of each program, and Figure 8 shows the total energy consumption for 1, 2 and 4 processors in the fastest execution mode. In these graphs, the left bars represents the results of OSCAR compiler without the proposed power saving scheme, the right bars show the results of OSCAR compiler using the proposed scheme. As shown in Figure 7, there is no performance degradation by using the power saving scheme in the fastest execution mode, while the energy consumption is reduced as shown in Figure 8. The proposed scheme reduced the consumed energy by 36.3 %(from 102[J] down to 65.0[J]) for 2 processors, 60.7 %(from 174[J] down to 68.4[J]) for 4 processors in SPEC95 applu, 1.56 %(from 92.1[J] down to 90.6[J]) for 2 processors, 4.64 %(from 95.0[J] down to 90.6[J]) for 4 processors in tomcatv.

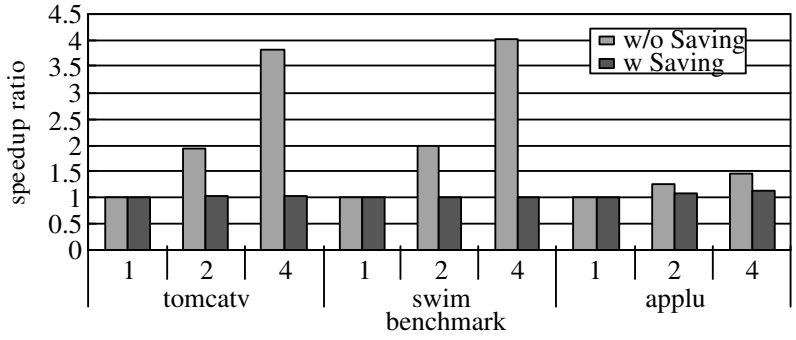

Fig. 9. Speedup in deadline mode

The reason why the proposed scheme can not reduce the energy consumption in tomcatv and swim is that the both application programs have large parallelism and the all processors must execute in "FULL" mode to attain the minimum execution time. The parallel execution time of these programs with 4 processors is about one quarter of sequential execution time. Therefore, though the power consumption is quadrupled by using 4 processors, the total energy consumption is almost equal to the energy of sequential execution.

On the other hand, there is a certain amount of idle time in applu. Therefore, the following controls were made. Figure 6 shows the main loop to which the power saving scheme is applied for 4 processors. The DOALL6, LOOP10-13, DOALL17, LOOP18-21, DOALL22 had no margin, then their frequencies were set to FULL. MID or LOW was chosen for other MTs according to each margin of task. Furthermore, the proposed scheme shut off the power supply in the idle times.

4.2 Performance in real-time processing with deadline constraints

Next, the evaluation results of real-time execution mode with the deadline constraint are described. Figure 9 shows the speedup ratio and Figure 10 shows the total energy consumption with the real-time deadline that was set to equal to the sequential execution time. The speedup ratio could be kept almost 1, as shown in Figure 9. This means the proposed scheme could satisfy the deadline constraints, or the sequential processing time.

Figure 10 shows that the saved power for real-time processing mode were 37.8 %(from 102[J] down to 63.3[J]) for 2 processors, 62.2 %(from 174[J] down to 65.8[J]) for 4 processors in applu, 21.6 %(from 92.1[J] down to 72.2[J]) for 2 processors, 45.4 %(from 95.0[J] down to 51.9[J]) for 4 processors in tomcatv, and 23.7 %(from 103[J] down to 78.7[J]) for 2 processors, 46.5 %(from 103[J] down to 55.2[J]) for 4 processors in swim.

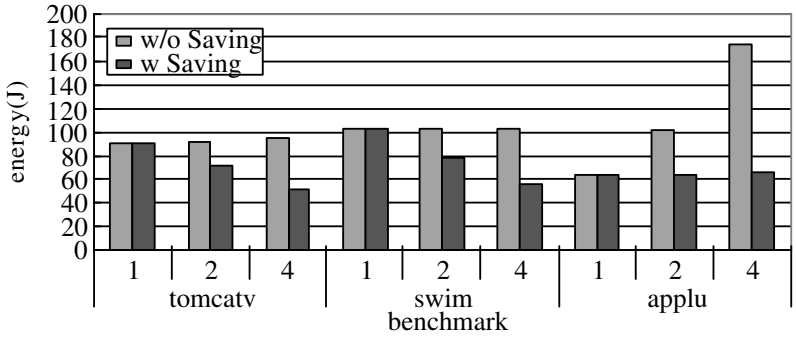

Fig. 10. Energy in deadline mode

These results shows the proposed scheme could realize large power reduction for programs with large parallelism under the real-time execution mode.

5 Conclusions

This paper has proposed compiler control power saving scheme for multi core processors. The proposed scheme can be applied for both the fastest parallel executing mode and the real-time execution mode with deadline constraint. The scheme gives us good effective performance and low energy consumption for the both modes.

The evaluation using OSCAR multigrain parallelizing compiler has shown the proposed scheme gave 60.7 percent energy savings for SPEC CFP95 applu using 4 processors without the performance degradation, and 45.4 percent energy savings for SPEC CFP95 tomcatv using 4 processors with real-time deadline constraint, or the sequential processing time, and 46.5 percent energy savings for SPEC CFP95 swim using 4 processors with the deadline constraint.

The detailed evaluation using an actual multi core processor and the implement of the dynamic scheduling are the future works.

Acknowledgments

A part of this research has been supported by NEDO "Advanced Heterogeneous Multiprocessor", STARC "Automatic Parallelizing Compiler Cooperative Single Chip Multiprocessor" and NEDO "Multi core processors for real time consumer electronics".

References

1. M.Wolfe. High performance compilers for parallel computing. *Addison-Wesley Publishing Company*, 1996.

2. R. Eigenmann, J. Hoeflinger, and D. Padua. On the automatic parallelization of the perfect benchmarks. *IEEE Trans. on parallel and distributed systems*, 9(1), Jan. 1998.
3. M. W. Hall, J. M. Anderson, S. P. Amarasinghe, B. R. Murphy, S. Liao, E. Bugnion, and M. S. Lam. Maximizing multiprocessor performance with the suif compiler. *IEEE Computer*, 1996.
4. Marc Gonzalez, Xavier Martorell, Jose Oliver, Eduard Ayguade, and Jesus Labarta. Code generation and run-time support for multi-level parallelism exploitation. In *Proc. of the 8th International Workshop on Compilers for Parallel Computing*, Jan. 2000.
5. H. Honda, M. Iwata, and H. Kasahara. Coarse grain parallelism detection scheme of a fortran program. *Trans. of IEICE*, J73-D-1(12):951–960, Dec. 1990.
6. H.Kasahara and et al. A multi-grain parallelizing compilation scheme on oscar. *Proc. 4th Workshop on Language and Compilers for Parallel Computing*, 1991.
7. Hironori Kasahara. Advanced automatic parallelizing compiler technology. *IPSJ MAGANIE*, Apr 2003.
8. David H. Albonesi and et al. Dynamically tuning processor resources with adaptive processing. In *IEEE Computer*, Dec. 2003.
9. Q. Wu, P. Juang, M. Martonosi, and D. W. Clark. Formal online methods for voltage/frequency control in multiple clock domain microprocessors. In *Eleventh International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct. 2004.
10. Chung-Hsing Hsu and Ulrich Kremer. The design, implementation, and evaluation of a compiler algorithm for cpu energy reduction. In *The ACM SIGPLAN Conference on Programming Language Design and Implementation*, Jun. 2003.
11. M. Obata, J. Shirako, H. Kaminaga, K. Ishizaka, and H. Kasahara. Hierarchical parallelism control for multigrain parallel processing. In *Proc. of 15th International Workshop on Languages and Compilers for Parallel Computing*, Aug. 2002.
12. K. Ishizaka, T. Miyamoto, M. obata J. Shirako, K. kimura, and H. Kasahara. Performance of oscar multigrain parallelizing compiler on smp servers. In *Proc. of 17th International Workshop on Languages and Compilers for Parallel Computing*, Sep. 2004.
13. Jun shirako, Kouhei Nagasawa, Kazuhisa Ishizaka, Motoki Obata, and Hironori Kasahara. Selective inline expansion for improvement of multi grain parallelism. *PDCN2004*, Feb. 2004.
14. H. Kasahara, H. Honda, M. Iwata, and M. Hirota. A compilation scheme for macro-dataflow computation on hierarchical multiprocessor system. *Proc. Int Conf. on Parallel Processing*, 1990.
15. H. Kasahara, S. Narita, and S. Hashimoto. Architecture of oscar. *Trans of IEICE*, J71-D(8), Aug. 1988.
16. H. Kasahara, H. Honda, and S. Narita. Parallel processing of near fine grain tasks using static scheduling on oscar. *Proceedings of Supercomputing '90*, Nov. 1990.
17. K. Kimura, W. Ogata, M. Okamoto, and H. Kasahara. Near fine grain parallel processing on single chip multiprocessors. *Trans. of IPSJ*, 40(5), May. 1999.
18. David Brooks, Vivek Tiwari, and Margaret Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proc. of the 27th ISCA*, Jun. 2000.
19. Hiroshi Kawaguchi, Youngsoo Shin, and Takayasu Sakurai. uitron-lp: Power-conscious real-time os based on cooperative voltage scaling for multimedia applications. In *IEEE Transactions on multimedia*, Feb. 2005.