Proc. of 20th ACM International Conference on Supercomputing Workshop on Advanced Low Power Systems (ALPS2006), Cairns, Australia, July 2006

Performance Evaluation of Compiler Controlled Power Saving Scheme

Jun Shirako†, Munehiro Yoshida†, Naoto Oshiyama†, Yasutaka Wada†,
Hirofumi Nakano†, Hiroaki Shikano†,††, Keiji Kimura†, Hironori Kasahara†
†Dept. of Computer Science, Waseda University
3-4-1 Ohkubo, Shinjuku-ku, Tokyo, 169-8555, Japan
††Central Research Laboratory, Hitachi, Ltd.
1-280, Higashi-Koigakubo, Kokubunji-shi, Tokyo 185-8601, Japan
{shirako,munehiro,oshiyama,yasutaka,hnakano,shikano,kimura,kasahara}@oscar.elec.waseda.ac.jp

Abstract

Multicore processors, or chip multiprocessors, which allow us to realize low power consumption, high effective performance, good cost performance and short hardware/software development period, are attracting much attention. In order to achieve full potential of multicore processors, cooperation with a parallelizing compiler is very important. The latest compiler extracts multilevel parallelism, such as coarse grain task parallelism, loop parallelism and near fine grain parallelism, to keep parallel execution efficiency high. It also controls the voltage and clock frequency of processors carefully inside an application program to reduce energy comsunption. This paper evaluates performance of compiler controlled power saving scheme which has been implemented in OSCAR multigrain parallelizing compiler. The developed power saving scheme realizes voltage/frequency control and power shutdown of each processor core during coarse grain task parallel processing. In the performance evaluation, when it is assumed that static power is one-tenth of dynamic power, OSCAR compiler with the power saving scheme achieved 61.2 percent energy reduction for SPEC CFP95 applu without performance degradation on 4 processors and 87.4 percent energy reduction for mpeg2encode, 88.1 percent energy reduction for SPEC CFP95 tomcatv and 84.6 percent energy reduction for applu with the real-time deadline constraint on 4 processors.

1 Introduction

Multicore processors which integrate multiple processor cores are attracting much attention, since they allow us to realize low power consumption, high effective performance, good cost performance and short hardware/software development period, with compiler supports. For example, Fujitsu FR-V[25], ARM MPCore[3], IBM, SONY and Toshiba Cell[5], Intel Xeon dual-core[21] and IBM Power5+[13] have been developed for consumer electronics, PCs, servers and so on. In order to achieve efficient parallel processing on multicore processors, cache and local memory optimization to cope with memory wall problem and minimization of data transfer among processors using DMAC (Direct Memory Access Controller) are necessary, in addition to the extraction of parallelism from an application program. There have been a lot of researches to extract parallelism for multicore processors in the areas of loop parallelizing compilers [22, 4, 7]. 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[6] extracts the multi-level parallelism including the coarse grain task parallelism by using extended OpenMP API. Also, OSCAR multigrain parallelizing compiler [9, 8, 14] 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. Furthermore, OSCAR compiler automatically determines the suitable number of processors for each part of a program, considering processing overhead and the global cache memory optimization over different loops.

So far, improving processing performance has been one of the most important problems. However, recently, reduction of power consumption is getting important. For the power saving, various methods by hardware and OS have been proposed. Adaptive Processing[1] 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 Con-

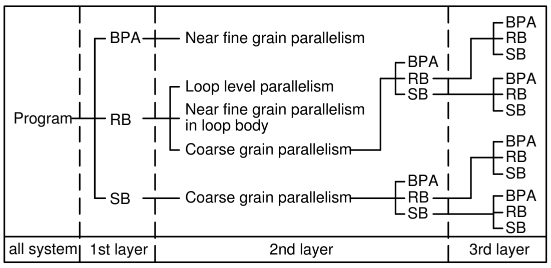

Figure 1. Hierarchical Macro Task Definition

trol [26] settles on the fitting voltage and frequency for each domain of processors using instruction issue queue occupancies as feedback signals.

This paper describes a compiler controlled power saving scheme for a multicore processor, which realizes voltage/frequency (V/F) control and power shutdown under the constraints of the minimum time execution or real-time execution with deadline.

2 Multigrain parallel processing

The developed power saving scheme in OSCAR compiler[11, 12] 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 Generation of macro-tasks [9, 8, 14]

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)[14]. 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.

2.2 Extracting 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 [9, 8, 14]. Then, to extract coarse grain task parallelism among macro-tasks, Earliest Executable Condition analysis [9, 8, 14] 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)[9, 8, 14] is generated for each macro flow graph. Macro-task graph represents coarse grain parallelism among macro-tasks.

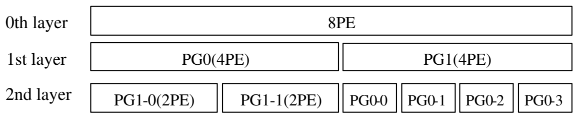

Figure 2. Hierarchical processor grouping

2.3 Hierarchical Processor grouping

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 to processors the grouped processor layer. OSCAR compiler with the automatic parallelized layer determination scheme [23, 24] 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

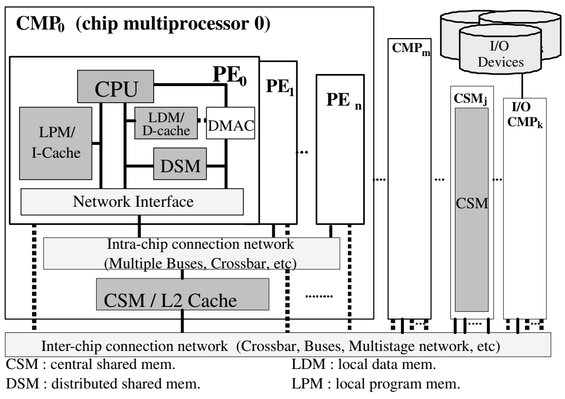
CMP$_0$ (chip multiprocessor 0)
CPU
PE$_0$
PE$_1$
PE $_n$
LPM/ I-Cache
LDM/ D-cache
DMAC
DSM
Network Interface
Intra-chip connection network (Multiple Buses, Crossbar, etc)
CSM / L2 Cache
CMP$_m$
I/O Devices
CSM$_j$
CSM
I/O CMP$_k$
Inter-chip connection network (Crossbar, Buses, Multistage network, etc)
CSM : central shared mem.
LDM : local data mem.
DSM : distributed shared mem.
LPM : local program mem.

Figure 3. OSCAR multicore processor

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 processor 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 reduction 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 static and dynamic power consumption. Also, execution at lower voltage and frequency may reduce the total energy consumption in real time processing with the deadline constraint. The compiler controlled power saving scheme realizes the following two modes of power saving. The first is the fastest execution mode that doesn't apply the power reduction 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.

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

3.1 Target model for the power saving scheme

In this paper, it is supposed that the target multicore processors have the following functions with the hardware supports like OSCAR multicore processor shown in Figure 3. The OSCAR (Optimally Scheduled Advanced Multiprocessor) architecture has been proposed to support optimization of multigrain parallelizing compiler [15, 9, 8], especially static and dynamic task scheduling [17, 15, 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) [17, 15, 16, 19]. 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 separately.
- The voltage can be changed with the frequency.
- Each processor can be powered on and off individually.

Here, each memory, DMAC and Network are not the target of the power saving scheme described in this paper. There are a lot of approaches for voltage and frequency (V/F) control. The developed 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, and then the static power becomes 0. These parameters and the number of V/F states can be changed, according to architectures and technology. This scheme also considers the state transition overhead that is given for each state.

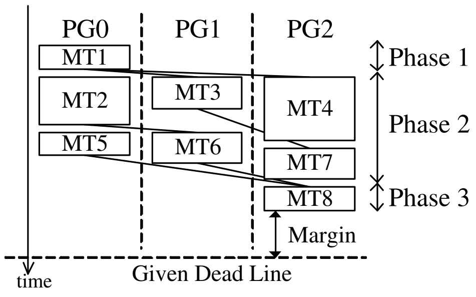

Figure 4. static scheduled MTG

3.2 Target MTG for the power 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 developed 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 constraints for target MTG

The developed scheme determines suitable voltage and frequency for each MT on a MTG based on the result of static task assignment. In other words, the developed 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 developed scheme is applied.

T_i : execution time of MT_i after V/F control

T_{start_i} : start time of MT_i

T_{finish_i} : finish time of MT_i

At the beginning of the developed scheme, T_i is not yet

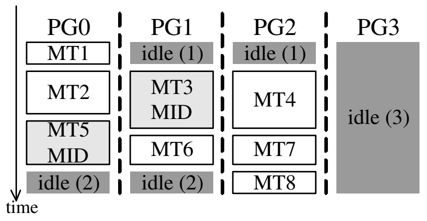

Figure 5. Result of V/F control

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.

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

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

According to these rules,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 finish time of 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 developed scheme determines suitable clock frequency for MT_i to satisfy the condition.

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 target 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

Table 2. dynamic/static Power and overhead

dynamic power	220[mW]
static power	2.2, 22, 66, 110[mW]
overhead of V/F control	0.1[ms]
overhead of power shutdown	0.2[ms]

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 executed 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 reduction can be expected by applying V/F control to $Phase_i$ with larger $Gain_i(F_m)$.

Next, to estimate the margin of the target MTG, the minimum value of T_{MTG} is calculated as 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, $T_{MTG\_margin} = 0$, then each Phase has to be executed at FULL. When $T_{MTG\_margin} > 0$, the developed scheme turns down the voltage and frequency of each Phase, according to $Gain_i(F_m)$. If a Phase has a single MT, the frequency of the MT is the same as the Phase. If a Phase includes some MTs and corresponds to the max term, the developed scheme also defines Phases for each argument of the max, and then determines clock frequency to execute these Phases. The algorithm to determine frequency for each Phase and MT is described in [11, 12].

3.5 Power supply control

Next, power supply control to reduce unnecessary energy consumption including static power consumption by idle processors is applied. The idle time occurs, when a PG (processor group) is waiting for other PGs to execute the MTs (1), finished all scheduled MTs (2) or has no MTs (3).

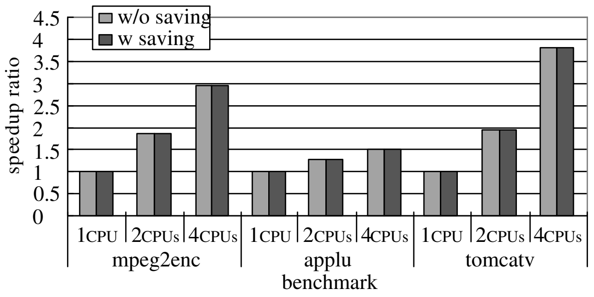

Figure 6. Speedup in the fastest mode

The gray parts of Figure 5 are the idle in each case. The power of the PG is turned off, if the idle time is longer than the frequency transition overhead and the energy becomes lower by power shutdown considering the overhead.

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 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.

4 Performance evaluation

This section describes the performance of OSCAR multigrain parallelizing compiler with the compiler controlled power saving scheme. The evaluation was performed by using the static scheduler in the compiler. For this evaluation, the parameters such as frequencies, voltages, dynamic energies, and static powers shown in Table 1 were used. In this paper, only energy for processors was evaluated. The state transition overhead of V/F control and power shutdown, dynamic and static power is shown in Table 2. The dynamic power at FULL frequency was measured by using Wattch[2]. Cooperative Voltage Scaling[18] was referenced to determine the parameters like the transition overhead, attribute of voltage/frequency and dynamic power at MID and LOW frequency. Static power is set to 2.2[mW] (1% of dynamic power), 22[mW] (10% of dynamic power), 66[mW] (30% of dynamic power) or 110[mW] (50% of dynamic power), supposing various type of multicore processors from low power oriented multicores to high performance multiprocessors. In this evalu-

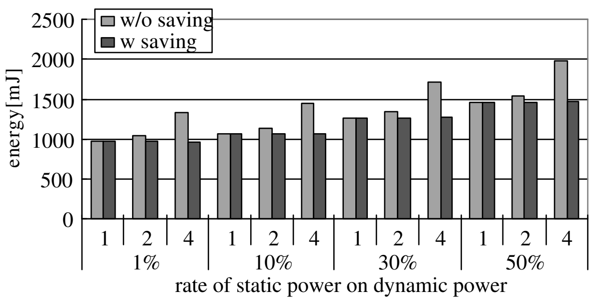

Figure 7. Energy of mpeg2encode (fastest)

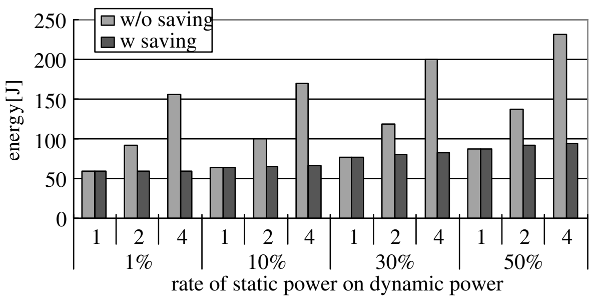

Figure 8. Energy of applu (fastest)

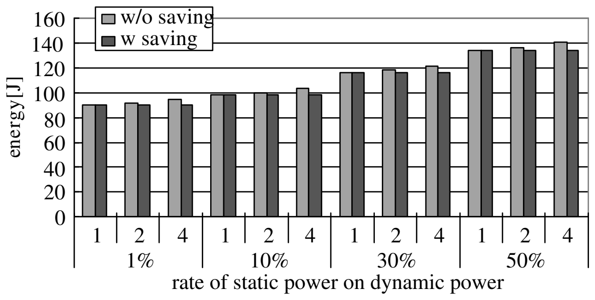

Figure 9. Energy of tomcatv (fastest)

Table 3. Energy reduction for 4 CPUs(fastest)

program	static	w/o saving	w saving	reduction
mpeg2	1 %	1336[mJ]	973[mJ]	27.2 %
	10 %	1455[mJ]	1071[mJ]	26.4 %
	30 %	1720[mJ]	1278[mJ]	25.7 %
	50 %	1985[mJ]	1476[mJ]	25.6 %
applu	1 %	156[J]	58.5[J]	62.4 %
	10 %	170[J]	65.9[J]	61.2 %
	30 %	201[J]	81.9[J]	59.2 %
	50 %	231[J]	95.1[J]	58.9 %
tomcatv	1 %	94.8[J]	90.4[J]	4.66 %
	10 %	103[J]	98.4[J]	4.65 %
	30 %	122[J]	116[J]	4.65 %
	50 %	141[J]	134[J]	4.64 %

ation, MediaBench mpeg2encode which was rewritten in Fortran[20], SPEC95 CFP applu and tomcatv were used. For applu, inline expansion and loop aligned decomposition for the data localization[10] were applied. Also, the main loop in applu was divided into the static part without conditional branch and the dynamic part with branches, in order to apply the power saving scheme.

4.1 Performance in the fastest execution mode

Figure 6 shows the speedup ratio of each program for 1, 2 and 4 processors in the fastest execution mode, when it is assumed static power is equal to 1% of dynamic power. The left bars represent the results of OSCAR compiler without the power saving scheme, the right bars show the results of OSCAR compiler using the compiler controlled power saving scheme. As shown in Figure 6, there is no performance degradation by using the power saving scheme in the fastest execution mode. When static power was changed to 22[mW] (10% of dynamic power), 66[mW] (30% of dynamic power) or 110[mW] (50% of dynamic power) assuming high performance processors, there were also no performance losses. Figure 7, 8 and 9 show the total energy consumption of mpeg2encode, applu and tomcatv for 1, 2, and 4 processors, changing the rate of static power on dynamic power to 1%, 10%, 30% or 50%. In mpeg2encode and applu, the power saving scheme using 2 or 4 processors reduces the energy consumption at each rate of static power. These applications have sequential parts which can't be parallelized, and then there is a certain amount of processor idle time. The power saving scheme applied V/F control and power shutdown, using this idle time. The developed scheme reduced the consumed energy by 5.5 % (from 1138[mJ] down to 1075[mJ]) for 2 processors, 26.4 % (from 1455[mJ] down to 1071[mJ]) for 4 processors in mpeg2encode and 35.4 % (from 101[J] down to 65.2[J]) for 2 processors, 61.2 % (from 170[J] down to 65.9[J]) for 4 processors in applu, assuming the rate of static power on dynamic power is 10 %. The energy reduction rate for 4 processors changing static power is shown in Table 3.

On the other hand, tomcatv has large parallelism to run all processors almost every time during the program execution. Therefore, all processors must execute at full speed to attain the minimum execution time. The parallel execution time of tomcatv with 4 processors is about one quarter of

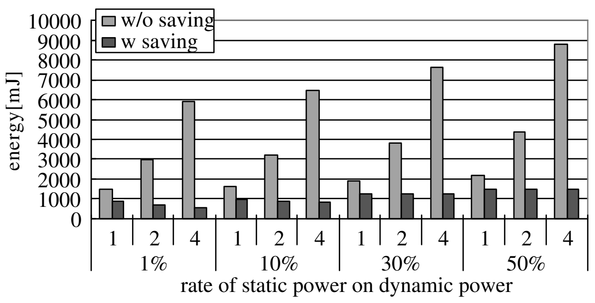

Figure 10. Energy of mpeg2encode (deadline)

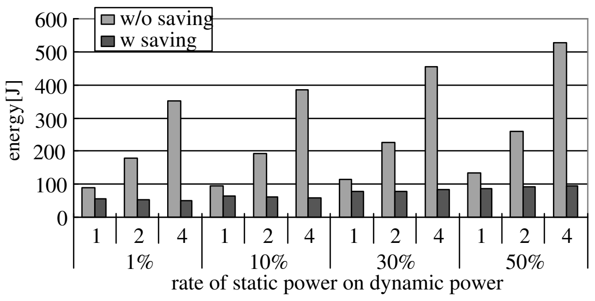

Figure 11. Energy of applu (deadline)

the 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 the sequential execution.

4.2 Performance in real-time execution mode with deadline constraints

Next, the evaluation results of real-time execution mode with the deadline constraint are described. Figure 10, 11 and 12 show the total energy consumed until the real-time deadline. Here, the deadline was set to 150% of the sequential execution time. The left bars represent the results of OSCAR compiler without any power saving scheme. In this case, all processors run at FULL frequency until the deadline. The right bars show the results of OSCAR compiler using the developed power saving scheme in real-time deadline mode. These figures show the power saving scheme drastically reduced energy consumption, because the developed scheme applied V/F control and power shutdown as far as the execution time didn't exceed the deadline. Furthermore, the energy consumption of mpeg2encode or tomcatv executed in parallel is lower than the energy of the sequential execution, when static power is set to 2.2[mW], 22[mW] or 66[mW]. The developed power saving scheme

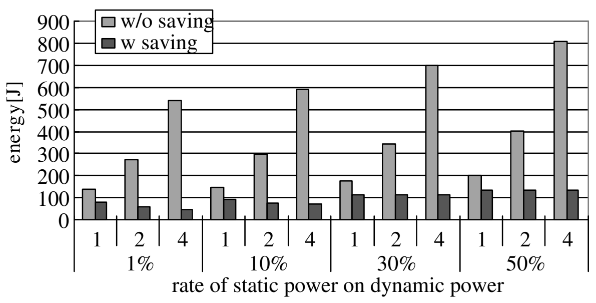

Figure 12. Energy of tomcatv (deadline)

Table 4. Energy reduct. for 4 CPUs(deadline)

program	static	w/o saving	w saving	reduction
mpeg2	1 %	5929[mJ]	592[mJ]	90.0 %
	10 %	6458[mJ]	815[mJ]	87.4 %
	30 %	7632[mJ]	1262[mJ]	83.5 %
	50 %	8806[mJ]	1476[mJ]	83.2 %
applu	1 %	354[J]	49.5[J]	86.0 %
	10 %	385[J]	59.5[J]	84.6 %
	30 %	455[J]	81.7[J]	82.1 %
	50 %	525[J]	95.1[J]	81.9 %
tomcatv	1 %	542[J]	48.3[J]	91.1 %
	10 %	591[J]	70.2[J]	88.1 %
	30 %	698[J]	114[J]	83.7 %
	50 %	805[J]	134[J]	83.3 %

in real-time processing mode reduced energy by 73.3 % (from 3229[mJ] down to 861[mJ]) for 2 processors, 87.4 % (from 6458[mJ] down to 815[mJ]) for 4 processors in mpeg2encode, 68.9 % (from 193[J] down to 59.8[J]) for 2 processors, 84.6 % (from 385[J] down to 59.5[J]) for 4 processors in applu and 73.8 % (from 295[J] down to 77.3[J]) for 2 processors, 88.1 % (from 591[J] down to 70.2[J]) for 4 processors in tomcatv, assuming the rate of static power on dynamic power is 10 %. Table 4 shows the energy reduction for 4 processors changing static power.

The execution time with the developed power saving scheme was less than the deadline in all the cases where static power was changed. This means the developed scheme could satisfy the deadline constraints.

5 Conclusions

This paper evaluated performance of compiler controlled power saving scheme for various type of multicore processors from the low power oriented to the high performance oriented, changing the quantity of static power. The scheme gave us good processing performance and low energy consumption for all the cases.

The evaluation assuming static power was 10% of dynamic power has shown that the compiler controlled power saving scheme gave 61.2 percent energy reduction for SPEC CFP95 applu using 4 processors without performance degradation and 87.4 percent energy reduction for mpeg2encode, 88.1 percent energy reduction for SPEC CFP95 tomcatv and 84.6 percent energy reduction for applu using 4 processors with the real-time deadline constraint.

The power saving scheme described here only controls processors in static scheduling. The development of the power saving scheme for dynamic scheduling and the saving method for the resources other than processors 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] D. H. Albonesi and et al. Dynamically tuning processor resources with adaptive processing. In *IEEE Computer*, Dec. 2003.

[2] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proc. of the 27th ISCA*, Jun. 2000.

[3] J. Cornish. Balanced energy optimization. In *International Symposium on Low Power Electronics and Design*, 2004.

[4] 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.

[5] D. P. et al. The design and implementation of a first-generation cell processor. In *In Proceeding of the IEEE International Solid-State Circuits Conference*, 2005.

[6] M. Gonzalez, X. Martorell, J. Oliver, E. Ayguade, and J. 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.

[7] 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.

[8] H.Kasahara and et al. A multi-grain parallelizing compilation scheme on oscar. *Proc. 4th Workshop on Language and Compilers for Parallel Computing*, 1991.

[9] 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.

[10] K. Ishizaka, T. Miyamoto, M. o. 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.

[11] J.Shirako, N.Oshiyama, Y.Wada, H.Shikano, K.Kimura, and H.Kasahara. Compiler control power saving scheme for multi core processors. In *Proc. of 18th International Workshop on Languages and Compilers for Parallel Computing(LCPC2005)*, Oct. 2005.

[12] J.Shirako, N.Oshiyama, Y.Wada, H.Shikano, K.Kimura, and H.Kasahara. Parallelizing compilation scheme for reduction of power consumption of chip multiprocessors. In *Proc. of 12th International Workshop on Compilers for Parallel Computers (CPC)*, Jan. 2006.

[13] R. Kalla, B. Sinharoy, and J. Tendler. Ibm power5 chip: a dual-core multithreaded processor. *IEEE Micro*, 24(2):40–47, 2004.

[14] H. Kasahara. Advanced automatic parallelizing compiler technology. *IPSJ MAGANIE*, Apr 2003.

[15] 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.

[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] H. Kasahara, S. Narita, and S. Hashimoto. Architecture of oscar. *Trans of IEICE*, J71-D(8), Aug. 1988.

[18] H. Kawaguchi, Y. Shin, and T. Sakurai. uitron-lp: Power-conscious real-time os based on cooperative voltage scaling for multimedia applications. In *IEEE Transactions on multimedia*, Feb. 2005.

[19] 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.

[20] T. Kodaka, H. Nakano, K. Kimura, and H. Kasahara. Parallel processing using data localization for mpeg2 encoding on oscar chip multiprocessor. In *Proc. of International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems*, Jan. 2004.

[21] I. Multi-core. http://www.intel.com/multi-core/.

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

[23] 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.

[24] J. shirako, K. Nagasawa, K. Ishizaka, M. Obata, and H. Kasahara. Selective inline expansion for improvement of multi grain parallelism. *PDCN2004*, Feb. 2004.

[25] A. Suga and K. Matsunami. Introducing the fr 500 embedded microprocessor. volume 20, pages 21–27, 2000.

[26] 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.