Selective Inline Expansion for Improvement of Multi Grain Parallelism

Jun Shirako[†], Kouhei Nagasawa[††], Kazuhisa Ishizaka[†], Motoki Obata[†††], Hironori Kasahara[†]
Department of Computer Science, Waseda University[†]
3-4-1 Ohkubo, Shinjuku-ku, Tokyo, 169-8555, Japan[†]
Fujitsu Limited[††] Hitachi, Ltd.[†††]
{shirako,nagasawa,ishizaka,obata,kasahara}@oscar.elec.waseda.ac.jp

ABSTRACT
This paper proposes a selective procedure inlining scheme to improve a multi-grain parallelism, which hierarchically exploits the 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. Using the proposed scheme, the parallelism among different layers(nested levels) can be exploited. In the evaluation using 103.su2cor, 107.mgrid and 125.turb3d in SPEC95FP benchmarks on 16 way IBM pSeries690 SMP server, the multi-grain parallel processing with the proposed scheme gave us 3.65 to 5.34 times speedups against IBM XL Fortran compiler and 1.03 to 1.47 times speedups against conventional multi-grain parallelization.

KEY WORDS
Parallel Processing, Automatic parallelizing compiler, Multi grain parallel processing, Program restructuring

1 Introduction

In automatic parallelizing compilers for multi-processor systems, the loop parallelization techniques have been researched extensively [1]. However, the loop parallelization techniques are almost matured and new generation parallelization techniques like multi-grain parallelization are required to extract more parallelism inside a program. As the compiler trying to exploit multiple levels of parallelism, NANOS compiler[2] extracts the multi-level parallelism including the coarse grain task parallelism by using extended OpenMP API. PROMIS compiler[3] hierarchically combines Parafrase2 compiler using HTG and symbolic analysis techniques with EVE compiler for the fine grain parallel processing. The OSCAR multigrain parallelizing compiler[4, 5] extracts two parallelism in addition to the loop parallelism. One is the coarse grain task parallelism among loops, subroutines and basic blocks. Another is near fine grain parallelism among statements inside a basic block.

In multi-grain parallelization in OSCAR compiler, a sequential program is decomposed into coarse grain tasks called macro-tasks(Basic Block, Repetition Block and Subroutine Block). Considering data dependence and control flow among macro-tasks, the compiler generates macro-task graph which extracts the coarse grain task parallelism. For efficient multi-grain parallelization, the compiler should determine how many processors are assigned to macro-tasks and which parallelizing technique, such as coarse grain, loop, near fine grain, is applied to each nested program layer, according to the parallelism of each level of macro-task graph [6, 7]. Intend to optimize task size and parallelism considering over-head of communication and synchronization, partitioning and scheduling schemes[8] and the lazy task creation scheme[9] have been proposed for SISAL language and fine grain tasks. However, for ordinary Fortran language and multi-grain tasks, no research has been performed.

Also, inter-procedure analysis and inline expansion of subroutines containing large parallelism is important to exploit multi-grain parallelism. About the inter-procedure analysis, a lot of schemes to parallelize loops have been researched. For example, SUIF compiler[10] performs analysis in two passes(bottom-up pass and top-down pass for the call graph), applies selective procedure cloning to avoid the loss of analysis precision considering code size.

This paper presents a selective inline expansion scheme to improve multi-grain parallelism and processor assignment method to each hierarchical macro-task graph. Also, the performance of the proposed scheme is evaluated on IBM 16 way SMP server pSeries690.

2 Coarse grain task parallel processing

This section describes the overview of the coarse grain task parallel processing in OSCAR multi-grain parallelizing compiler[11].

2.1 Generation of macro-tasks

OSCAR compiler first generates the macro-tasks(MTs), namely, block of pseudo assignment statements "BPA"(similar to basic blocks), repetition blocks "RBs"(or loops) and subroutine blocks(caller of subroutine) "SBs" from a source program. Furthermore, compiler hierarchically decomposes the body of a sequential repetition block

and a subroutine block.

2.2 Extraction of coarse grain task parallelism

After generating macro-tasks, the data dependency and the control flow among macro-tasks for each nested layer are analyzed hierarchically and represented by the macro flow graph(MFG)[4, 12, 11]. Then, to extract coarse grain parallelism among macro-tasks, Earliest Executable Condition analysisis applied to Macro flow graphs. Earliest Executable Condition represents the conditions on which macro-task may begin its execution earliest. By this analysis, macro-task graph(MTG)[4, 12, 11]. is generated from macro flow graph. Macro-task graph represents coarse grain parallelism among macro-tasks.

2.3 Macro-task scheduling to processor groups and processor elements

To execute each nested macro-task graph efficiently, the compiler has to group processors hierarchically. OSCAR compiler groups processor elements(PEs) into Processor Groups(PGs) virtually, then a macro-task is assigned to the PG. If a hierarchical macro-task graph can be generated inside a macro-task, processor elements in a processor group can be also grouped into multiple processor groups to execute hierarchical macro-tasks.

3 Automatic determination of parallel processing layer

The compiler must decide the number of PGs and PEs appropriately considering each parallelism of MTG. This section presents the solution of this problem.

3.1 Calculation of parallelism of each MTG

First, the proposed determination scheme estimates parallelism of macro-task graph(MTG) from sequential execution cost and critical path length, or CP length which is the longest path length from entry node to exit node, of MTG. MTG's sequential execution cost means the sum of all MT's execution time in the MTG, as to control flow and control branch probability. The way to calculate execution cost of macro-tasks is as follows. An execution cost of block of pseudo assignment statements(BPA) is the total cost of all instructions inside BPA. A processing cost of repetition block "RBs" is the cost of loop body(MTG) multiplied by the number of the loop iterations. An execution cost of subroutine block(SB) is the execution cost of a called subroutine. By using these sequential execution cost, the parallelism is defined as follows. MTG_i's sequential execution cost is Seq_i, the coarse grain task parallelism $Para_i$ is defined as

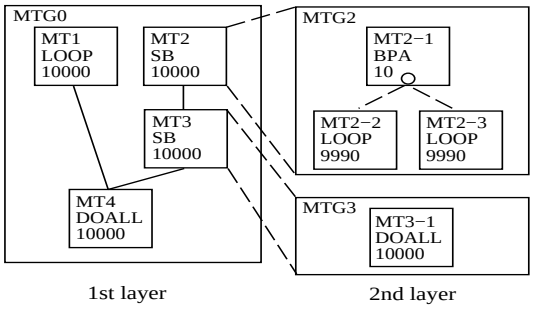

Figure 1. Hierarchical MTG for assignment of processors

$$Para_i = Seq_i \ / \ CP_i$$

where, critical path length is CP_i. $\lceil Para_i \rceil$ shows minimum number of PGs to execute MTG_i in CP_i.

Second, $Para\_ALD(Para\ After\ Loop\ Division)$ is defined as total parallelism of coarse grain task and loop iteration level parallelism. In the proposed processor groups determination scheme, T_{min} represents a minimum task cost for loop parallelization considering overheads of synchronization and task scheduling on each target multiprocessor system. To convert loop parallelism to coarse grain task parallelism, a parallelizable RB is divided into sub RBs having larger cost than T_{min}. However, in this phase the compiler doesn't actually divide RBs. It divides only for the calculation of $Para\_ALD$. This coarse grain parallelism among divided sub RBs is named converted loop parallelism. CP_ALD_i is defined as critical path length of MTG_i, assuming the RBs are divided. $Para\_ALD_i$ is represented by

$$Para\_ALD_i = Seq_i \ / \ CP\_ALD_i$$

$Para$ and $Para\_ALD$ represent the parallelism of MTG. On the other hand, as the enough number of processors to use all parallelism of a MT and the MTGs included in it,

$$Hierarchical\_Para\_max_i = \lceil Para_i \rceil \ \times \\ Converted\_Loop\_Para_i \ \times \\ \max_j(Hierarchical\_Para\_max_{i\_j})$$

is defined. Here, $Converted\_Loop\_Para_i$ is MT_i's converted loop parallelism. $\max_j(Hierarchical\_Para\_max_{i\_j})$ is the maximum $Hierarchical\ \_Para\_max$ in macro-tasks of MTG_i.

3.2 Determination scheme of the number of PGs and PEs

Using $Para$, $Para\_ALD$, and $Hierarchical\_Para\_max$, the proposed scheme determines the combination of the number of processor groups and processor elements for each macro-task graph. This section describes the process of this determination.

step.1

Let's assume the number of processors available for MTG_i is N_{Avail_PEi}. The numbers of PGs and PEs of

Table 1. Parallelism of each layer in fig.1

	$Seq.cost[u.t]$	$CP[u.t.]$	$CP\_ALD[u.t.]$	$Para$	$Para\_ALD$	H_Para_max	(PG, PE)
MTG_0	40000	30000	21000	1.33	1.90	20	(2PG, 2PE)
MTG_2	10000	10000	10000	1.00	1.00	1	(1PG, 1PE)
MTG_3	10000	10000	1000	1.00	10.00	10	(2PG, 1PE)

MTG_i are denoted as N_{PGi} and N_{PEi}. In this scheme, N_{PGi} and N_{PEi} should meet next conditions.

$$Para_i \leq N_{PGi} \leq Para\_ALD_i$$

and

$$N_{PGi} \times N_{PEi} = N_{Avail\_PEi}$$

Here, the combination of (N_{PGi}, N_{PEi}) which maximumizes N_{PGi} is selected. Because MTGs in upper level naturally have larger execution costs than lower level of MTGs. It means the overheads of task scheduling and synchronization can be kept relatively small in upper level of MTGs.

step.2

Here, N_{PEi} is the number of processors assigned to MT_{i_j}, or the jth macro-task of MTG_i. $Hierarchical\_Para\_max_{i\_j}$ shows the enough number of processors to execute MT_{i_j} efficiently. Therefore, $\max_j(Hierarchical\_Para\_max_{i\_j})$ represents the upper limit of N_{PEi}. If N_{PEi} is

$$N_{PEi} > \max_j(Hierarchical\_Para\_max_{i\_j}),$$

the possibility of overheads of unnecessary synchronization and task scheduling is high, because of too many processors. To avoid such case, N_{PEi} is changed to $\max_j(Hierarchical\_Para\_max_{i\_j})$ and N_{PGi} is adjusted to $Hierarchical\_Para\_max_i$.

By applying **step.1** and **step.2** to each macro-task graph from upper to lower, all combinations of (N_{PGi}, N_{PEi}) are determined.

3.3 An example of assignment of processors

Next, this section explains the determination scheme of the number of PGs and PEs using Figure 1. This figure consists of hierarchical macro-task graphs, namely the first level of MTG_0 and the second level of MTG_2 and MTG_3. A node represents a macro-task, a solid edge represents data dependency and a dashed edge represents control dependency. DOALL is a parallelizable RB, LOOP is a sequential RB which can't be parallelized. Let's assume $T_{min} = 1000[u.t.](u.t.$: unit time), the inner MTGs of MT_1, MT_4, MT_{2_2}, MT_{2_3} and MT_{3_1} have no parallelism. The number of available processors is supposed as 4 in MTG_0. The parallelisms of these MTGs are shown in Table 1.

In MTG_0, the number of available processors is 4, namely $N_{Avail\_PE0} = 4$. Since $Para_0 = 1 \leq N_{PG0} \leq Para\_ALD_0 = 2$ and $N_{PG0} \times N_{PE0} = 4$, ($N_{PG0}$, N_{PE0}) = (2PG, 2PE) is chosen by **step.1** of subsection 3.2. $Hierarchical\_Para\_max_3 = 10 > 2$, MT_3 needs 2 processors. Then, **step.2** determines MTG_0 is executed by(2PG, 2PE).

Next let's see MTG_2. Since $N_{Avail\_PE2} = 2$ and $Para\_ALD_2$ is 1, (N_{PG2}, N_{PE2}) = (1PG, 2PE) is chosen by **step.1**. However, because $Hierarchical\_Para\_max$ is 1, N_{PE0} is changed to 1 by **step.2**. Therefore, (1PG, 1PE) is defined as the combination of PG $\cdot$ PE of MTG_2. Also, MTG_1 included in MT_1 is determined as (1PG, 1PE). For MTG_2, $N_{Avail\_PE2} = 2$ and $Para_3 = 1 < N_{PG3} < Para\_ALD_3 = 10$, so (2PG, 1PE) is selected. The results of these determinations of (N_{PG}, N_{PE}) are shown in table 1.

After these determination, the macro-tasks like DOALLs are divided by N_{PG} of MTG including them. Then, MT_4 is divided by $N_{PG0} = 2$, becomes MT_{4a} and MT_{4b}. Also, MT_{3_1} is divided by 2, becomes MT_{3_1a} and MT_{3_1b}.

Here, let's compare loop parallelization and multi-grain(coarse grain and loop) parallelization using an example, executing the program of figure 1 on 4 processors machine. In the ordinary loop parallelization, MT_1 and MT_2 are executed by single processor. MT_{3-1} in MTG_3 and MT_4 are DOALL, and are executed by 4 processors in parallel. Therefore, the total execution time is $10000 + 10000 + 10000/4 + 10000/4 = 25000[u.t.]$. On the contrary when multi-grain parallelization is used, a processor group(which is named PG0) processes MT_2, MT_3, MT_{4a} and another processor group(which is named PG1) processes MT_1, MT_{4b}. The total execution time is the processing time of PG1's processing time, $10000 + 10000/2 + 5000/2 = 17500[u.t.]$. Then multi-grain parallel processing can shorten execution time by $7500[u.t.]$, compared with the loop parallelization.

4 Inline expansion scheme to increase parallelism

Multi-grain parallel processing can use loop parallelism and coarse grain task parallelism hierarchically. However, the scheme mentioned above gives priority to upper MTG, may not assign enough processors to execute the MTG in lower nested level. When the subroutine blocks(SBs) in lower nest level have large parallelism, processors can execute much more efficiently by inlining these SBs and increasing the parallelism of upper MTGs. This section describes the selective inlining scheme to improve multi-grain

parallelism.

4.1 Parallelism included by MT

Section 3 defined $Hierarchical\_Para\_max$ as the upper limit of the number of the processors for using the all parallelism of a macro-task efficiently. Also, in order to represent the necessary number of processors to use all parallelism among a macro-task, $Hierarchical\_Para$ is defined as follows. First, it's assumed all parallelizable RBs in MT_i is divided so that a divided portion is larger than T_{min}. $Hierarchical\_CP_i$ is defined as the total sum of critical path from lower level to MT_i's level. In other words, when $Hierarchical\_CP_{i\_j}$ is considered as the minimum execution cost of MT_{i_j}(which is inner macro-task of MTG_i), $Hierarchical\_CP_i$ is the longest path length of these minimum execution cost. If MT_i is unparallelizable RB, $Hierarchical\_CP_i$ is multiplied by the number of iterations. $Hierarchical\_Para_i$ is defined as

$$Hierarchical\_Para_i = Seq_{MT_i} \; / \; Hierarchical\_CP_i$$

where Seq_{MT_i} is MT_i's sequential execution cost. In the following discussion, $\lceil Hierarchical\_Para_i \rceil$ is considered as the lower limit of the number of processors to execute MT_i in $Hierarchical\_CP_i$.

$Para\_inl\_ALD_i$ is defined as the parallelism of MTG_i after all subroutine blocks(SBs) are inline expanded. All SBs are inlined, and each parallelizable RB in MTG_i is converted to sub RBs, then the critical path length of This MTG_i is defined as $CP\_inl\_ALD_i$. $Para\_inl\_ALD_i$ is represent by

$$Para\_inl\_ALD_i = Seq_i \; / \; CP\_inl\_ALD_i$$

If MT_i is subroutine block(SB), $Para\_inl\_ALD_i$ stands for the total parallelism of coarse grain task and loop which is available in upper level of MTG by inlining.

4.2 Selective inlining considering parallelism

The proposed scheme applies inline expansion only to SBs which requires more processors than assigned by the determination scheme presented in section 3, using $Hierarchical\_Para$ and $Para\_inl\_ALD$. Therefore, this selective inlining scheme can avoid the overheads of task scheduling and load imbalance of MTs aggravation. The selective inline expansion method is described as follows.

step.1

Parallel processing layer determination scheme described in section 3 is applied to a target program, until the MTG_i whose N_{PGi} is greater than 2 is found. Let's assume SB_{i_j} is the MT_{i_j} composed of subroutine. If there is any SB_{i_j} which need more processors than N_{PEi} in MTG_i, namely

$$Hierarchical\_Para_{i\_j} > N_{PEi}$$

the proposed inlining scheme think that inline expansion is effective for SB_{i_j}, and this MTG_i is a candidate for inlining.

step.2

If SB_{i_j} is inlined, the parallelism of MTG_i is changed. Therefore, the compiler must determine the appropriate number of PGs and PEs again. Let's introduce N_{PG_i}' as the predicted value of the number of PGs and N_{PE_i}' as the predicted value of the number of PEs after inline expansion. These parameters are determined as follows.

$$Para_i \leq N_{PG_i}' \leq Para\_inl\_ALD_i$$

and

$$N_{PG_i}' \times N_{PE_i}' = N_{Avail\_PEi}$$

The combination of (N_{PG_i}', N_{PE_i}') which has the maximum N_{PGi} is selected. N_{PG_i}' and N_{PE_i}' are only predicted value. The definite number of PGs and PEs are calculated again after actual inline expansion.

step.3

The SBs which satisfies the next 2 condition in MTG_i is selected for inline expansion.

$$Hierarchical\_Para_{i\_j} > N_{PE_i}' \quad (1)$$

and

$$Para\_inl\_ALD_{i\_j} \geq 2 \quad (2)$$

The condition (1) represents the total parallelism of MT_{i_j} is greater than the assigned processors. The condition (2) shows the parallelism can be increased by inlining MT_{i_j}.

Step.1 and **step.3** in the proposed scheme are applied to each macro-task graph from upper to lower, and is able to select all subroutine blocks which need inline expansion. After selecting subroutines, the determination scheme of PGs and PEs defines the actual number of PGs and PEs for each layer.

For example, if this scheme is applied to the macro-task graphs in subsection 3.3, only MT_3 is inlined. Because MT_3 is assigned to 2 processors by the determination scheme of PGs and PEs,

$$MT_3\text{'s } Hierarchical\_Para_3 = 10 > 2$$

and

$$MT_3\text{'s } Para\_inl\_ALD_3 = 10 > 2$$

Therefore, the number of processors to execute MT_3 is increase from 2 to 4, the parallelism of MT_3 becomes more available.

5 Performance evaluation

This section describes the performance of the proposed selective inline expansion scheme implemented in OSCAR multi-grain parallelizing compiler, on IBM high-end SMP server pSeries690.

5.1 Evaluation environment

In this evaluation, OSCAR compiler with the proposed scheme is used as a parallelizing pre-processor and generates a parallelized program using OpenMP API. The OpenMP program uses the "one time single level thread generation" scheme which allows us to minimize thread

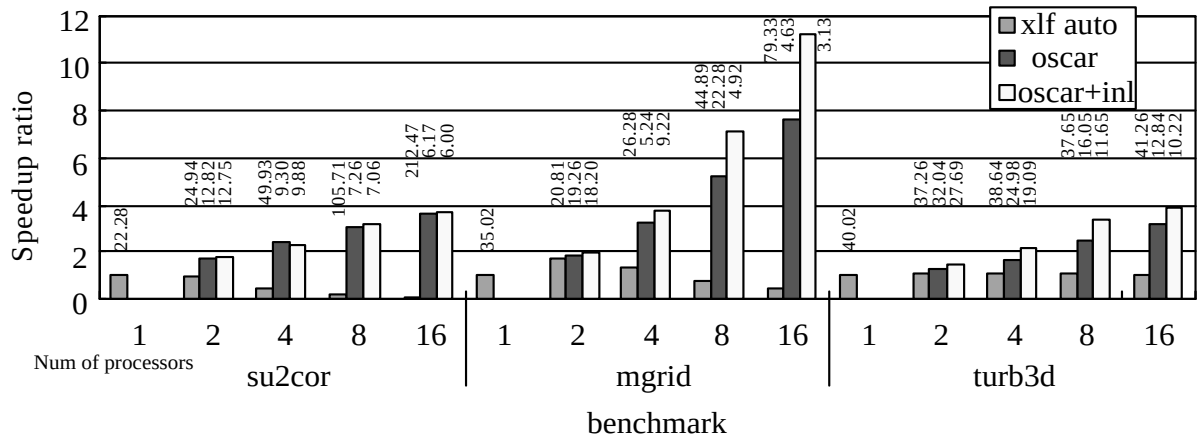

Figure 2. Speedup ratio of SPEC95 benchmarks up to 16PE on regattaH

generation overhead by forking and joining parallel threads at the beginning and the end of the program only once and realize hierarchical coarse grain task parallel processing[13, 14].

The generated OpenMP program is compiled by IBM XL Fortran version 8.1 latest automatic parallelizing compiler, and executed on IBM high-end SMP server pSeries690 with 16 processors, or 8 Power4 chips.

As the target programs of this evaluation, 103.su2cor, 107.mgrid, and 125.turb3d from SPEC95FP benchmarks are used, which have coarse grain parallelism over different nest levels. The compile-options for sequential execution by the XL compiler is "-O5 -qarch=pwr4", when 107.mgrid and 125.turb3d is compiled. 103.su2cor which can't be compiled by "-O5" is compiled using "-O4 -qarch=pwr4". Also, the compiler options of "-O5 -qsmp=auto -qarch=pwr4"is used for mgrid and turb3d, and "-O5 -qsmp=auto -qarch=pwr4"is used for su2cor, for automatic loop parallelization by the XL compiler. On the other hand, for compilation of the OpenMP codes generated by OSCAR compiler, "-O3 -qsmp=noauto -qarch=pwr4" is used for the 3 programs. In this evaluation, array renaming is applied for su2cor program, and loop distribution for turb3d. These restructured source codes are used for both OSCAR compiler and the XL compiler.

5.2 Performance

Figure 2 shows the speedup ratio against sequential execution of su2cor, mgrid and turb3d for 1, 2, 4, 8 and 16 processors. In this graph, the left bars show the performance of the XL compiler's automatic loop parallelization, the middle bars show the performance of OSCAR compiler with the determination scheme of the number of PGs and PEs, the right bars are the performance of OSCAR compiler using both the determination scheme with the selective inlining scheme. The figures over each bars represent the execution time[s].

Tables 2, 3 and 4 show the parallelism and the determined combination of PGs and PEs for 16 processors. The leftmost column represents the name of subroutine or loop, (PGs, PEs) shows the combination of the number of processor groups and processor elements. H_Para means $Hierarchical\_Para$.

In the figure 2, the minimum execution time for mgrid by the XL compiler up to 16 processors was 16.72[s] by using 3 processors. The minimum processing time by OSCAR compiler without the proposed scheme was 4.63[s] by 16 processors. Also, the proposed inline expansion reduced the minimum execution time by OSCAR compiler into 3.13[s]. This is 5.34 times speedup compared with the XL compiler, 1.47 times speedup compared with OSCAR compiler without inlining.

Table 3 shows the first layer of subroutines RESID, RPRJ3, PSINV and INTERP have large parallelism, so they were processed by combination of (16PG, 1PE) by OSCAR compiler. However, there are statements calling COMM3 whose $Hierarchical\_Para$ is 44.56 in these subroutines, COMM3 was assigned to the only 1 processor. Because of it, increment of speedup ratio was not enough. On the other hand, multi-grain parallelization using proposed scheme inlined COMM3, so that COMM3 can be executed by 16 processor. Therefore, OSCAR compiler with the selective inline expansion got sufficient performance up to 16 processors.

For su2cor, the minimum execution time was 6.17[s](3.67 times speedup against sequential execution) by using 16 processors, when OSCAR compiler parallelized without the proposed inlining scheme. In OSCAR compiler with the proposed scheme, the fastest execution time was 6.00[s](3.71 times speedup against sequential execution) by using 16 processors. On the other hand, the XL compiler shows the performance degradation by loop parallelization. The reason is the overheads to control and

Table 2. Combination of(PGs, PEs) and Parallelisms of su2cor

	(PGs, PEs)	H_Para	$Para\_inl\_ALD$
LOOPS	(1, 16)	125.23	1.00
DO 400	(2, 8)	192.29	3.07
PERM	***	77.42	77.00
MATMAT	***	78.90	78.00
MATADJ	***	82.06	82.00
ADJMAT	***	78.90	78.00
INT2V	(1, 8)	104.54	1.16
INT4V	(1, 8)	161.35	1.16
DO 100	(1, 8)	168.49	167.15
BESPOL	***	70.74	70.00

synchronize each thread in the XL compiler are larger than OSCAR compiler[7]. Multi-grain parallelization, for example, can execute DO 400 using 2PG by coarse grain parallel processing, and use the parallelism in lower level by 8PE. On the other hand, since the XL compiler execute only parallelizable loop which is in lower nested level by 16 processors, suffers large overhead.

Table 2 shows that DO 400 in subroutine LOOPS which consumes about 31% of total execution time has $Para = 2.47$ and $Para\_inl\_ALD = 2.47$, so the OSCAR compiler applied coarse grain task parallelization to DO 400 by combination of (2PG, 8PE). There are subroutines PERM, MATMAT, MATADJ, ADJMAT, INT2V and INT4V which have high $Hierarchical\_Para$ in this loop, and PERM, MATMAT, MATADJ and ADJMAT whose $Para\_inl\_ALD$ are larger than 2 were inlined by proposed scheme. Then the processors processing these subroutines were increased from 8 to 16, the parallelism could be exploited more efficiently. Though INT2V and INT4V call BESPOL having high parallelism, BESPOL is included in loop DO 100, the parallelism of BESPOL isn't available in DO 400. Therefore, $Para\_inl\_ALD$ of both INT2V and INT4V is less than 1.2, these subroutine was not selected for inlining.

For turb3d, the minimum processing time of the XL compiler was 37.40[s] by 3 processors. In OSCAR compiler, the fastest execution time without the inlining was 12.84[s], the minimum execution time with the inlining was 10.22[s] up to 16 processors. The proposed scheme gave 3.65 times speedup against the XL compiler, 1.26 times speedup against OSCAR compiler without the selective inlining.

In turb3d, a Repetition block(RB) in subroutine TURB3D almost occupies the total execution time. Coarse grain task parallelism $Para$ in this RB is 6; it was executed by (8PG, 2PE). However, subroutines ENR, UXW, LINAVG, MIXAVG and LIN having large $Para\_inl\_ALD$ and $Hierarchical\_Para$ are included in this RB, then these SBs were assigned to 2 processors in multi-grain parallelization without inlining.

Table 3. Combination of (PGs, PEs) and Parallelisms of mgrid

	(PGs, PEs)	H_Para	$Para\_inl\_ALD$
RESID	(16, 1)	11289.83	99.62
RPRJ3	(16, 1)	10836.49	99.60
PSINV	(16, 1)	11045.54	99.61
INTERP	(16, 1)	7784.91	99.78
COMM3	***	44.71	44.56

Table 4. Combination of (PGs, PEs) and Parallelism of turb3d

	(PGs, PEs)	H_Para	$Para\_inl\_ALD$
GOTO 1001	(8, 2)	1.19	6.00
ENR	***	99.88	2.00
UXW	***	2048.03	64.00
LINAVG	***	4091.22	64.00
MIXAVG	***	2046.59	64.00
LIN	***	4091.22	64.00

6 Conclusions

This paper has proposed the selective inline expansion scheme for hierarchical multi-grain parallel processing. In performance evaluation using 3 programs from SPEC95FP benchmarks on IBM high-end SMP server pSeries690, the proposed scheme gave us 3.81 times speedup for su2cor, 5.34 times speedup for mgrid and 3.65 times speedup for turb3d against IBM XL Fortran compiler ver.8.1 and 1.03 times speedup for su2cor, 1.47 times speedup for mgrid and 1.26 times speedup for turb3d compared with OSCAR compiler without the inlining. From these results, it was confirmed the proposed scheme could realize more effective multi-grain parallel processing. The proposed inlining scheme and determination scheme of PGs and PEs do not consider data localization, or the cache optimization. The total multi-grain parallelization with the data locality optimization is important, to improve performance of multi-grain parallelization further.

Acknowledgment

A part of this research has been supported by Japan Government Millennium Project IT21 METI/NEDO Advanced Parallelizing Compiler Project (http://www.apc.waseda.ac.jp) and STARC "compiler cooperative single chip multiprocessor".

References

[1] M.Wolfe. High Performance Compilers for Parallel Computing. *Addison-Wesley Publishing Company*, 1996.

[2] E.Ayguade, X.Martorell, J.Labarta, M.Gonzalez, and N.Navarro. Exploiting multiple levels of parallelism in openmp:A case study. *ICPP'99*, Sep. 1999.

[3] C.J.Brownhill, A.Nicolau, S.Novack, and C.D.Polychronopoulos. Achieving multi-level parallelization. *Proc. of ISHPC'97*, Nov. 1997.

[4] M. Okamoto, K. Aida, M. Miyazawa, H. Honda, and H. Kasahara. A Hierarchical Macro-dataflow Computation Scheme of OSCAR Multi-grain Compiler. *Trans. of IPSJ (japanese)*, 35(4):513–521, Apr. 1994.

[5] K.Kimura and H.kasahara. Near fine grain parallel processing using static scheduling on single chip multiprocessors. *Proc.IWIA'99*, Nov. 1999.

[6] J. shirako, H. Kaminaga, N. Kondo, K. Ishizaka, M. Obata, and H. Kasahara. Coarse Grain Task Parallel Processing with Automatic Determination Scheme of Parallel Processing Layer. *Technical Report of IPSJ, ARC2002-148-4(in Japanese)*, May 2002.

[7] M. Obata, J. Shirako, H. Kaminaga, K. Ishizaka, and H. Kasahara. Hierarchical Parallelism Control Scheme for Multigrain Parallelization. *Trans. of IPSJ (in Japanese)*, 44(4):1044–1055, Apr 2003.

[8] V. Sarkar. artitioning and Scheduling Parallel Programs for Multiprocessors. *MIT Press Cambridge*, 1989.

[9] E. M. D. Kranz and R. H. Jr. Lazy Task Creation: A Technique for Increasing the Granularity of Parallel Programs. *IEEE Transactions on Parallel and Distributed Systems 2(3):264-280*, July 1991.

[10] M.W.Hall and et al. Maximizing Multiprocessor Performance with the SUIF Compiler. *IEEE Computer,*, Dec 1996.

[11] H. Kasahara, M. Obata, K. Ishizaka, K. Kimura, H. Kaminaga, H. Nakano, K. Nagasawa, A. Murai, H. Itagaki, and J. Shirako. Multigrain Automatic Parallelization in Japanese Millenium Project IT21 Advanced Parallelizing Compiler. *Proc. of IEEE PARELEC (IEEE International Conference on Parallel Computing in Electrical Engineering)*, Sep. 2002.

[12] M. Obata, J. Shirako, H. Kaminaga, K. Ishizaka, and H. Kasahara. Hierarchical Parallelism Control for Multigrain Parallel Processing. *Proc. of 15th International Workshop on Languages and Compilers for Parallel Computing (LCPC2002)*, Aug. 2002.

[13] H. Kasahara, M. Obata, and K. Ishizaka. Coarse Grain Task Parallel Processing on a Shared Memory Multiprocessor System. *Trans. of IPSJ (japanese)*, 42(4), Apr. 2001.

[14] M. Obata, K. Ishizaka, and H. Kasahara. Automatic Coarse Grain Task Parallel Processing Using OSCAR Multigrain Parallelizing Compiler. *Ninth International Workshop on Compilers for Parallel Computers(CPC 2001)*, Jun. 2001.