Advanced Parallelizing Compiler Technology

2002 Annual Report

March 2003

INTRODUCTION

The Advanced Parallelizing Compiler (APC) Technology Project formed the APC Research Body and began its work to commemorate the new millennium in September 2000, with a three year plan through the JIPDEC, in receiving a commission from the NEDO, based on the Ministry of Economy, Trade and Industry's program for the scientific technology development for industries, which creates new industries as part of the government-private sector joint research, the Millennium Project IT21.

This Research Body consists of JIPDEC, researchers dispatched from Hitachi, Ltd. and Fujitsu, Ltd., and the National Institute of Advanced Science and Technology, Waseda University as joint research establishments, with the University of Electro-Communications, Tokyo Institute of Technology and Toho University as re-outsourcing establishments.

In this project, we have conducted research and development in an aim to improve the usability of the chip multiprocessor, which is considered to be the main architecture of the next generation processors, and shared memory multiprocessor that is now incorporated into various computers such as PC and WS, and to double the effective performance, which is the substantive nature when the application program is actually used. Through this research and development, we have been able to advance the platform-free automatic parallelization software that includes the multigrain parallelization that extracts the multigrain parallelism hierarchically from the entire program, the cache and DSM optimization technology and advanced data dependant analysis.

We have been able to get such R&D results due to the encouraging support of those involved in the establishment of the global evaluation and promotion systems such as the International Cooperation Committee in which well-known researchers of the world has cooperated, and the "network concentrated research body method", which was introduced to the industrial technology system for the first time.

We believe that the achievements of this project could contribute to various application fields such as the global environment, gene-analysis, new drug development, financial engineering, automobile design, aerospace development that require such high-performance computers with improved price performance and usability. It could also contribute to promoting the research and development of various SoC fields where reduction of the development period of hardware and advanced application programs such as the next generation cell phones that are expected to incorporate chip multiprocessor, games, PDA, price performance improvement and low power consumption are desired.

C O N T E N T S

I. Development of Advanced Parallelizing Compiler Technology

1. Development of Automatic Multi-Grain Parallelizing Compiler Technology

To accelerate programs on multi-processor systems, automatic parallelizing compilers need to exploit not only simple parallelism among loop-iteration in a program but also complexed parallelism such as coarse-grain parallelism between subroutine-calls, between loops including subroutine calls or between loops and also fine-grain parallelism by sets of basic blocks. To solve this problem, we have developed the automatic multi-grain parallelizing technologies that makes the best use of multi-grain parallelism in programs and the tuning technologies for parallel processing that enables to enhance the compiler's parallelization of programs by feedbacking run-time information or user's knowledge to the compiler. In this fiscal year, which is the final year of this project, we have integrated the elements of technologies to solve these problems to evaluate and also have reflected this evaluation.

1.1. Multi-Grain Parallelism Exploitation Technology

A target of the research and development of multi-grain parallelism exploitation technology, which is the base technology of automatic multi-grain parallel processing, is to research and develop the technology for analysis of parallelism in a sequential program and efficient use of the parallelism on shared memory multiprocessor systems.

This year, as the final year of the project, technology for multi-grain parallelism exploitation scheme and integration of the scheme with data locality optimization and scheduling technologies, parallelism exploitation technology for interprocedural multi-grain parallelism exploitation technology extending interprocedural dependency analysis technology and coarse grain parallelism extraction technology among coarse grain tasks like procedures were researched and developed.

1.1.1. Technology for Multi-Grain Parallelism Exploitation Infrastructure

This section reports hierarchical parallelism control scheme for multi-grain parallel processing to use multi-grain parallelism efficiently, the integration of data locality optimization technology and scheduling technology, and the performance evaluation on shared memory multiprocessor systems. For the improvement of effective performance of multiprocessor systems, it is very important to use multi-grain parallelism, which exploits coarse grain parallelism among basic blocks, loops and subroutines, and near fine grain parallelism among statements in addition to the loop parallelism. It is too difficult for ordinary users to determine how many processors or groups of processors should be assigned to each layer according to the parallelism of the layer in order to use hierarchical parallelism efficiently in multi-grain parallel processing. To cope with this problem, this section proposes hierarchical parallelism control scheme for multi-grain parallel processing which estimates the multi-grain parallelism in each layer and determines the suitable number of processors and scheduling scheme. This technology realizes automatic multi-grain parallelization efficiently by using integrated data localization scheme.

The performance of the proposed automatic multi-grain parallelization is evaluated on IBM pSeries690 regatta with 16 processors, IBM RS6000 SP 604e High Node with 8 processors and Sun Ultra80 with 4 processors using 16 FORTRAN77 programs of SPEC CFP95 and SPEC CFP2000. As the result, this technology gave us 7.1 times speedup for 104.hydro2d compared with the maximum performance of a native loop parallelizing compiler IBM XL Fortran for AIX Version 8.1 on IBM Regatta, 4.1 times for 101.tomcatv compared with IBM XL Fortran Version 7.1 on IBM RS6000 and 5.4 times for 102.swim compared with Sun Forte 6 Update 2 on Sun Ultra80. In 16 FORTRAN77 programs of SPEC CFP95 and SPEC CFP2000, the technology of multi-grain parallelization boosted up the performance of IBM pSeris690 regatta 3.5 times in average, IBM RS6000 2.4 times and Sun Ultra80 2.0 times.

1.1.2. Medium-Grain Parallelism Exploitation Technology

In 2002 fiscal year, the module has been implemented which extracts loop level pipeline parallelism. The module transforms a loop nest as follows:

(1) Loop nests are transformed to be executed in parallel with synchronization between only near neighbor processors.

(2) Its loops are interchanged or tiled to improve data locality.

(3) Its redundant barrier synchronizations are removed to reduce the overhead.

We have also implemented pipeline parallelism in OpenMP.

Consequently, the performances of SPEC CFP2000/173.applu, SPEC CFP95/110.applu, and NPB2.3-serial/LU on the Alpha Server and SR4300 have become more than twice as the base ones.

1.1.3. Coarse-Grain Parallelism Exploitation Technology

It is important for the automatic multigrain parallelizing compiler to exploit the coarse grain parallelism such as between subroutines, loops, and basic blocks to achieve a good performance. We have developed the coarse grain parallelizing mechanism, which can extract the coarse grain parallel tasks and generate the code not only for the speculative execution schema, but also for the non-speculative execution schema.

In 2002 fiscal year, we have developed the coarse grain task selection routine. The selected tasks include loops and/or procedure calls. The routine selects them based on data and control dependence information. That is, for each basic block, the use data definition point, the output data use point, the branch statement deciding the execution of it, the branch statement deciding not to execute it, and the relation between them. At the beginning of the task selection, the top of the task is selected based on these information. At this time, it is decided whether this task will be executed speculatively or not. After that, the end of the task is decided. We have developed the code generation routine for non-speculative coarse grain tasks, which include procedure calls. This routine adds the code for a task generation, a task invocation, a task start, and synchronization to the original task code.

It was confirmed that applying this parallelizing mechanism to the main loop in SPEC CFP2000/168.wupwise, the performance was increased.

1.1.4. Technique of Analyzing Interprocedural Multi-Grain Parallelism

Loop parallelization techniques cannot extract sufficient parallelism from the programs including such sections as those outside loops or sequential loops. So, the multigrain parallelization technique, which can extract parallelism from multiple grains of sections (tasks) such as basic blocks or procedures in programs, is necessary. In this study we have been researching the technique of analyzing interprocedural multigrain parallelism using the interprocedural automatic parallelizing module WPP (Whole Program Parallelizer) as a base module.

In this fiscal year, we have developed and evaluated the parallelism analysis technique that extracts parallelism from interprocedural hierarchical tasks and outputs OpenMP programs. The algorithm of this technique is as follows.

(1) Regarding each node in the macro-flow graph (MFG) as a task, it analyzes control and data dependences between tasks.

(2) For each layer of the MFG, it applies a CP/MISF-based static task scheduling only to the layer and estimates the execution time of the program with that scheduling.

(3) It selects the layer with the schedule that provides the shortest estimated execution time of the program and it applies a task parallelization to the layer.

(4) It determines the location of each barrier in parallel regions.

(5) It generates an OpenMP program with the above extracted parallelism.

Using this technique, much thread parallelism is expected to be extracted from program sections that have no loop parallelism.

Our technique is applied to SPEC CFP95/103.su2cor benchmark program and the resulting code can extract parallelism among subroutine calls.

1.2. Data Dependency Analysis Technology

This technology is the basis of automatic parallelization. It aims to enable making of the program which cannot be parallelized up to now parallel by analyzing the data dependency more in detail and widely. This fiscal year, we have developed technologies for interprocedural data dependency analysis to enlarge the area for parallelization. And we have also developed and evaluated the predicated data-flow analysis and run-time data dependency analysis.

1.2.1. Interprocedural Data Dependency Analysis Technology

Interprocedural data dependency analysis is an essential technology to advancement of automatic parallelization. In order to extract medium and coarse grain parallelism effectively from the whole program, an analysis technology, which is not restricted to the boundary of each procedure, is required. Moreover, a mechanism for handling the result of the analysis to the succeeding module is also necessary. The result of interprocedural data dependency analysis is useful to the fundamental optimizations among procedures (e.g., constant propagation) and to the inline expansion taking into account the condition of the caller.

In this research item, we have developed not only the interprocedural data dependence analysis technology but also the framework to support it and the interprocedural fundamental optimization techniques to utilize it.

In this fiscal year, we have developed the following techniques:

(1) Interprocedural data dependency analysis

It delivers recursively data dependency information in a subprogram into the caller.

(2) Interprocedural fundamental optimizations

It contains constant propagation and folding, induction variable detection, and scalar expansion, which are using the result of the interprocedural analysis.

(3) Inline expansion using interprocedural data dependency analysis

It refers the result of constant propagation in the caller procedure.

(4) Interface and compiler driver handling the result of interprocedural analysis

The interface includes internal representation of the relation between procedures. The compiler driver makes reconfiguration of the functional modules easy.

We confirmed that these techniques encourage the exploitation of the middle and coarse grain parallelism if they work previously. The induction variable detection and the scalar expansion support DO-ALL and pipeline parallelism on the most programs of SPEC CFP95 and NAS Parallel benchmarks. And the selective inline expansion supports the multi-grain parallelism on SPEC CFP95/103.su2cor benchmark program.

1.2.2. Predicated Data-Flow Analysis Technique

The interprocedural parallelizing module WPP parallelizes loops by analyzing interprocedural data dependences. There are some loops that are not parallelized by the WPP but can be parallelized principally. That is the case where the statements with a certain data reference preventing parallelism executes only on a special condition and the condition never holds when using a given input data. In this study, we have been researching the predicated data-flow analysis in order to parallelize those loops.

In this fiscal year, we have been developing and evaluating the predicated data-flow analysis, whose algorithm is as follows.

(1) The data-flow analysis phase analyzes for each statement the array reference region with a predicate, the condition that the statement executes.

(2) The data-dependence-analysis phase calculates for each loop such a condition that there is no loop-carried dependence using the array reference regions with predicates.

(3) The code-generation phase generates multi-versioned code, in which the code selected at runtime is one of the following three loops; a serial loop and two parallel loops to which array privatization is applied or not, respectively.

Our technique is applied to some loops in SPEC benchmarks and the resulting code is parallelized, although the WPP cannot parallelize those loops.

1.2.3. Run-Time Data Dependency Analysis Technique

Most parallelizing compilers analyze loop-carried data dependences in a loop and judge the parallelizability of the loop. Those compilers, however, cannot parallelize the loop that includes an indirect-referenced array and has a possible loop-carried data dependence between two references of the array because the compilers can not judge the parallelizability of the loop at compile time. In this study, we have been researching the run-time data-dependence analysis in order to parallelize the loop.

In this fiscal year we have developed and evaluated the run-time data-dependence analysis, which outputs a code with the following structure.

(1) The code that records the accesses to variables referenced in a target loop

executing the loop in parallel.

(2) The code that checks the data dependences using the access record and re-executes the loop sequentially if the result of the check shows the sequentiality of the loop.

Our technique is applied to a loop in SPEC CFP95/103.su2cor program and the resulting code ran 1.5 times faster than the original one on Hitachi SMP machine SR8000.

1.3. Automatic Data Distribution Technology

The automatic data distribution technology is the compiler technology that partitions the data referenced in a program and assigns each of them to the local memory of the most appropriate processor. There is a gap between the logical memory view and the physical memory structure on physically distributed shared-memory processors. So, different memory models need different optimization techniques. In this fiscal year, we have developed and evaluated the automatic data distribution technique for distributed shared-memory processors and for distributed cache, and developed the optimization technique of data locality for the processors with distributed shared memories or distributed shared caches.

1.3.1. Automatic Data Distribution Technology for Distributed Shared-Memory Multiprocessors

In recent years, the distributed shared-memory multiprocessors (DSMs) have attracted attention of users because of their performance scalability and their ease of parallel programming; the former is due to physically distributed memories and the latter logically shared memories. Although usual memory-referencing instructions for DSMs can access physical memories on remote processors as well as those on local processors, any reference to remote data takes more time than one to local data. For this reason, data distribution, which determines how to assign data to processors, is important to obtain good performance for DSMs. In this study, aiming at determining the most appropriate data distribution by compilers, we have been researching the automatic data distribution techniques for DSMs.

In this fiscal year, we have been developing and evaluating our data distribution technique for indirectly referenced arrays. Our automatic data distribution technique, the first-touch control data distribution method (FTC), realizes complex data distribution accurately using the first-touch page allocation mechanism of the operating system. If a program includes an indirectly referenced array, our technique generates a code where a temporary array is used until the value of an index array of the array is determined and the indirectly referenced array is distributed by the FTC immediately after that.

In the evaluation, we compared two versions of the NPB 2.3 serial/CG (Class B) program. They are the programs to which our method is applied or not, respectively. The former version ran 6.7 times faster than the latter version on SGI(TM) Origin(TM) 2000 (32 processors).

1.3.2. Technology for Distributed Cache

It is more efficient that data are partitioned in such a way each data are accessed only from one processor. For the conflicts between processors are reduced, and each partitioned data can be allocated to the local memory of each processor. Besides it can improve data locality within each processor so much, if the partitioned data can be contracted to its minimum size.

Such an optimization has been already proposed. But in 2002 fiscal year, we proposed a new loop transformation technique to contract many more arrays, and a new loop fusion technique to reduce the size of an array much more.

Our techniques can improve the performance of SPEC CFP2000/173.applu and NPB2.3-serial/BT drastically. The results of this research have been published in the proceedings of CPC2003 (Compilers for Parallel Computing 2003) and HPCS2003 (High Performance Computing Symposium 2003).

1.3.3. Data Locality Optimization Technology

On a multiprocessor system with distributed caches, in order to realize multigrain parallel processing efficiently, it is required to develop data locality optimization technology which can reduce data transfer overhead among coarse grain tasks by using distributed caches in addition to utilization of coarse grain parallelism.

This annual report presents a data-localization scheme to utilize both coarse grain task parallelism and data locality in multigrain parallel processing. Concretely, so as to realize loop aligned decomposition on large regions in macrotask-graphs, a detection method of data-localization target region composed of consecutive-part/adjacent-part and an inter-loop dependence analysis method which analyzes iteration-based data dependencies among loops inside data-localization target regions have been developed. The data-localization scheme has been integrated into APC compiler and its effectiveness was confirmed.

1.4. Speculative Execution Technology

In this technological item, we research and develop the speculative execution scheme that is one of the element technologies of "Automatic Parallelizing Compiler". Our target of the speculative execution is not the branch prediction used in the conventional processor, that is, only instruction level speculation, but a medium grain size such as loop iteration level, and course grain size, such as between subroutines, loops, and basic blocks, is targeted.

In 2002 fiscal year, we applied the proposed speculation technique for medium-grain tasks to a real application to confirm its effectiveness. Then, we have proposed that the calculation technique of the appropriate granularity for speculated loops to gain the performance. Moreover, we have confirmed the effectiveness of the proposed scheme. As for the speculation for the course grain size tasks, we have completed the research and development of the effective speculation scheme by optimizing the task size and its initiation time. Moreover, we have completed the development of the support mechanism to apply speculative execution effectively by collecting the dynamic information of a program.

1.4.1. Speculation Techniques for Medium-Grain Tasks for Multi-Grain Parallelization

The following four features are indispensable to adopt the speculative execution for medium grain tasks: (1) dividing a program into a set of tasks that are suitable for speculative execution, (2) selecting a task to be speculated, (3) dynamic scheduling to decide the initiation time of tasks, and (4) discarding the tasks that became unnecessary.

In 2002 fiscal year, we have evaluated the proposed scheme (1)-(4) by applying them to 129.compress from SPEC CPU95 benchmark on two CPUs of IBM pSeries690 RegattaH. As a result, we have confirmed that the overhead to keep the coherence of shared variables between threads - the execution time of flush instruction of OpenMP and the waiting time of a thread initiation using spin lock - is too large to gain the speculation performance. Thus, the execution time became slow about 30% in comparison with the normal execution after applying the speculation on to the function compress that accounted for 99% of the execution time of 129.compress.

However, it has been confirmed that the 20% speedup can be done (the theoretical maximum speed improvement ratio of 129.compress by the proposed speculative execution technique is 1.3 times) when assuming that the average execution time of one iteration of the loop is 1.2μs which is about ten times of the original one. In addition, the 30% speedup, which is almost equal to the theoretical maximum speed improvement ratio, has been confirmed when assuming the average execution time of one iteration of the loop is 11μs that is about 100 times of the original one.

As a result, we have confirmed that we can obtain the speed improvement by the proposed speculative execution technique when Ti/R+To<Ti is approved, where Ti is the average execution time of one iteration of the speculated loop, To is the overhead of the flush instruction and the spin lock waiting time for one iteration, R is the theoretical speedup ratio. Therefore, if an own overhead of various platforms (overhead by the flush instruction and the spin lock) is known, judging right or wrong to apply the speculative execution became possible. The result of this research has been published in National Conference of IPSJ.

1.4.2. Speculative Execution Technology for Coarse-Grain Tasks

For coarse grain parallel execution, such as between subroutines, loops, and basic blocks, we have developed a framework that involves speculative execution and non-speculative execution, and optimizes the task size and its initiation time.

In 2002 fiscal year, we have developed the code generation routine for the speculative coarse grain tasks which are selected by the coarse grain task selection routine. That routine adds the code for the speculative task generation, the task invocation, synchronization, and the judgement of the speculation to the original code. We wrote test programs and verified the correctness of the generated code.

It was confirmed that the performance was improved when the speculative execution was suitable for the program.

1.4.3. APC supporting Architecture: Hot Trace Detector

Speculation is a technique that exploits the statistical nature of the target programs to speed up the execution. The most basic statistical nature is the trace, which is the control flow of the program consisting of multiple series of branch executions. This information can be used for optimization such as speculative instruction scheduling in the VLIW processors. We designed and implemented the Hot Trace Detector, which can mesure the frequencies of the hot (frequent) traces.

Hot Trace Detector consists of the trace extractor and the histogram generator. The trace extractor receives branch IDs whenever CPU pipeline executes branch instructions, converts them into trace ID, and sends it to histogram generator. Histogram generator creates histogram of trace IDs. The results are stored in the main memory, and are combined to generate final histogram when the execution finishes.

We have implemented Hot Trace Detector on the reconfigurable experiment facility REX that uses FPGAs to emulate processors. Signals are extracted from MIPS compatible processor and connected to Hot Trace Detector.

Compress, one of SPEC benchmark programs, was executed on the MIPS compatible processor equipped with Hot Trace Detector for the trace extraction experiment. We could successfully assure that Hot Trace Detector could extract the most frequent traces.

1.5. Scheduling Technology

This section reports a development of a multiprocessor scheduling algorithm to utilize coarse grain task parallelism, reduce overhead of data transfer and effectively use a memory near a processor like a cache memory, and the performance of the developed technology which is integrated with the technology for multi-grain parallelism exploitation infrastructure and data locality optimization technology.

This year, a scheduling scheme for cache optimization with padding among coarse grain tasks considering cache line conflict has been developed. This scheme optimizes the cache memory by using partial static scheduling scheme to assign coarse grain tasks accessing the same data to the same processors as consecutively as possible in the multi-grain parallelization. In addition, the proposed cache optimization eliminates cache line conflict misses by inter-array padding.

This scheduling technology is integrated into the multi-grain parallelizing compiler which is developed as the technology for multi-grain parallelism exploitation infrastructure and evaluated on shared memory multiprocessor systems Sun Ultra80. As the result, this technology gave us 5.1 times speedup for 101.tomcatv compared with the maximum performance of Forte 6 Update 2 on Sun Ultra80, 5.5 times speedup for 102.swim, 2.5 times speedup for 104.hydro2d and 1.2 times speedup for 125.turb3d.

1.6. Common Interface Language among Some Optimization Facilities

As interfaces for some compilation facilities, we have determined to use specifications those are shown in below.

- Specification for programming
  FORTRAN77 specification
- Specification for parallelizing
  OpenMP API V1.1 specification

We have implemented the input feature for OpenMP API V1.1 specification and the output feature for FORTRAN77 specification to make these specifications into interfaces for some compilation facilities. These implementations made some compilation facilities unite one compiler.

2. Development of Tuning Technology for Parallel Processing

Our goal in this research and development is to establish the interactive and platform-free parallelization tuning technology that speeds up the execution of a given program making the best use of dynamic information, which can not be obtained from compiler's static analyses. To achieve our goal we research and develop the following element techniques of the tuning technology for parallel processing: the program visualization technique (the technique summarizing, extracting, and visualizing the factors inhibiting parallelization), the technique for profiling and utilizing run-time information (the technique profiling run-time information and reflecting it to compiler's optimization), the feedback-directed selection technique of compiler directives (the technique tuning the combination of compiler's optimizations), and the directives for parallel tuning tool (the technique of designing and implementing the directives used in parallel and optimization tuning). In this fiscal year we have been conducting the development and evaluation of each of those element techniques.

2.1. Program Visualization Technique

To obtain high performance on multiprocessors the compiler's automatic parallelization has been widely used. The automatic parallelization, however, is not enough for getting the maximal performance of a program. So, the parallelization tuning using user's knowledge is important. To inspect the parallelism of a program easily it is important for tuning tools to provide users with helpful information such as compiler's analysis results. For example, there are some tools that show pairs of statements having data-dependence relationship prohibiting parallelization. Showing those statements helps users to find causes of prohibiting parallelization of a loop. These tools, however, can not show any statement having data-dependence relationship in a procedure called within the loop. So, users have to find such statements for themselves; that is a laborious task for them. In this study, we are aiming at developing an effective parallelization-tuning tool for this case and are researching program-slicing technique that shows statements having data-dependence relationship beyond procedure boundaries.

In this fiscal year, we have been developing the interprocedural data-dependence locator, which finds all the statements having data-dependence relationships in a loop including procedure calls even if those statements exist in a procedure called within the loop. This tool finds the statements with data dependences in the following steps.

(1) The tool finds automatically all the data-dependence relationships between assignment statements or calls to procedures in the same procedure as the target

loop.

(2) When the user specifies a call to a procedure, the tool finds all the data-dependence relationships between any statement in the callee procedure and the statement that has the data dependence with the call to the procedure.

Our technique is applied to some loops including procedure calls in SPEC benchmarks and the data dependences beyond procedure boundaries are found.

2.2. Techniques for Profiling and Utilizing Run-Time Information

When the cause of poor performance for a loop is found in the parallelization tuning, it sometimes happens that the cause is due to an inappropriate transformation by an optimizing compiler. That transformation is considered to be conducted based on indefinite information about the execution time of the loop or the loop trip counts, whose values are sometimes difficult to obtain at compile time. So, the technique for profiling, utilizing runtime information, and generating an optimized code has attracted attention of users. In this study, aiming at developing a platform-free interface that can collect loop-execution information, we have been researching the technique for profiling and utilizing runtime information.

In this fiscal year we have developed the library collecting the loop-execution information. Using this library, we can easily specify the execution time for a loop, the loop trip counts, the stride of the loop, and the execution time per iteration. The calls to this library are inserted preceding and following the loops in a program automatically by the tool realizing "the feedback-directed selection technique of compiler directives".

2.3. Feedback-Directed Selection Technique of Compiler Directives

Optimizing compilers apply many kinds of loop transformation to a given loop nest. However, it is difficult for the compilers to select the optimized loop transformation or its parameter. So, the option tuning, which determines the most appropriate compiler options or compiler directives based on runtime information, is important. There are two kinds of option-tuning tools for user programs. One optimizes compiler options, which are specified to the whole program. The other optimizes compiler directives, which are specified to each loop. The former can not specify different options for different loops. The latter does not consider the optimized combination of directives that are effective to multiple loop nests as a whole. In this study, aiming at developing an option-tuning tool that is effective to multiple loop nests and finds the optimized combination of parallel and optimization directives in a short time, we have been researching the feedback-directed selection technique of compiler directives.

In this fiscal year we have conducted the design, implementation, and evaluation of our tool. This tool has the following two features.

(1) It applies the same combination of parallel and optimization directives to each multiple loop nest in one trial and it measures the execution time of each multiple loop nest.

(2) It uses the fractional factorial design to determine the combination of directives for multiple loop nests.

Our technique is applied to some loops in SPEC95/mg benchmark program and we can determine the optimized combination of directives for multiple loop nests as a whole in a small number of combinations.

2.4. Directives for Parallelization Tuning Tool

Compiler's automatic parallelization is widely used to obtain a good performance for programs on shared-memory multiprocessors. The performance, however, is limited because some programs need dynamic information for the parallelization judgment but most parallelizing compilers just use static information. Although there are some methods that judge the parallelizability of a program at run time, they cause a runtime overhead. So, the tuning technology for parallel processing and the directives for the tuning is important: the former uses a user's knowledge about a program and the latter makes the knowledge to be reflected in the program. In this study, we have been researching some tuning directives that make possible to extract more parallelism from programs.

In this fiscal year, we have been designing the specifications of loop-tiling directives and developing the compiler technique realizing those directives, which specify the tiled loop, its tile size, and the depth of the tile-control loop. They can be inserted preceding any loop with any depth in each multiply nested loop by the tool realizing “the feedback-directed selection technique of compiler directives”. That enables the selection of the highest performance program in wider range of its candidate.

II Development of performance evaluation for parallelizing compilers

The goal of this research is to establish the technology for more objective performance evaluation of a parallelizing compiler for SMP machines. We are developing this technology along with the evaluation of Research Theme I: "Development of Advanced Parallelizing Compiler Technology". Because our Parallelizing Compiler Technology developed by this project includes several optimization functions, the evaluations of each function are needed. We use the kernel-level programs and compact applications to evaluate these functions. We planed to use full-scaled application level benchmarks for the total evaluation. This year, we added and changed some benchmarks as described in 2.1, tested them which can run on our environments, evaluated the parallelism of them based on the criterion developed last year. Also we analyzed the newly added benchmarks and predicted the performance in terms of multi-grain parallelism before the final evaluation. Finally we evaluated the APC compiler by these benchmarks.

1. Development of evaluation methods for individual functions

We have evaluated each optimization function this year. The evaluation results are described in the related section in Chapter I.

2. Development of an overall evaluation method

This year, we first built the environments for evaluating the performance of the added benchmarks, checked whether the candidate benchmarks can run on these environments, then selected the benchmarks to use in the final evaluation, and evaluated the performance of the selected benchmarks by the guidelines developed last year. We will describe our four evaluation environments in 2.3.

Our goal is to get about the double performance on the same SMP machine compared with the objects generated by the commercial compilers that were released at the time this project began (Oct, 2000). Because of the nature of parallel execution, the best performance is not always obtained by using the maximum number of CPUs. So in this situation we will use CPUs by which we can obtain the best performance by these compilers.

2.1. Choice of benchmark programs

First, it will be necessary to use the well-known benchmarks for an overall evaluation. Also the benchmarks will be needed to have some scale to evaluate the parallel execution, while we can run these benchmarks on our environments. Of course some of these benchmarks can be parallelized by the current technology, which means it is impossible to achieve the double performance by our technology. Also some benchmarks may not have any parallelism that means there is no room of applying our technology at all.

Here we define these groups of attributes as follows.

- High level parallelism benchmarks:

  Even the commercial compilers can already achieve more than 50% scalability factor of the number of CPUs.

- Low level parallelism benchmarks:

  The scalability exists but not more than 50% of the number of CPUs by the commercial compilers.

- Difficult to parallelize benchmarks:

  Parallel execution time is the same level or even slower than the serial execution by the commercial compilers.

Last year we selected the benchmarks from SPEC CFP2000 and NPB. This year we added SPEC CFP95, and changed the NPB version from 3.0 beta to 2.3, and evaluated whether these benchmarks can run on our environments each of which consists of the SMP machines and the commercial compilers. Also we evaluated the SPEC CFP2000 benchmarks on IBM pSeries690 Model681 we installed at the end of last year.

SPEC CFP2000 benchmark suite is developed by SPEC/OSG, announced in 1999 as the successor of SPEC CFP95 benchmark suite. The performance results of this suite have been published by over 10 major vendors and more than 370 systems including all the SMP machines of our environments. We selected 6 benchmarks from this suite written in FORTRAN77 as candidate of our evaluation benchmarks. FORTRAN77 is the only supported language developed by this project.

SPEC CFP95 benchmark suite was announced in 1995 for floating point operation evaluation. It consists of 10 programs all written in FORTRAN77. The performance results of this suite have been published by over 10 major vendors and more than 600 systems. SPEC CFP95 had been used as a benchmark suite for a long time since the announcement at the beginning of this project, so it is thought that many in commercial compilers are tuned for these benchmarks. We selected 10 benchmarks from SPEC CFP95.

NPB benchmark suite is provided by NAS (Numerical Aerospace Simulation) program of NASA Ames Research Center, and targets the development of 21th century's aerospace vehicle using CFD (Computational Fluid Dynamics) computation.

NPB simulates the computation and data transformation of the CFD programs and consists of 5 kernel benchmarks and 3 virtual application programs. All of these 8 benchmarks has 4 problem sizes to run, and these are identified by Class A, B, C and W. Because the IS benchmark is not written in FORTRAN77, we selected other 7 benchmarks from these 8 benchmarks. And we used Class A of these benchmarks for our evaluation.

The scalability results of SPEC CFP95 are changed from last year because we added the results of IBM pSeries690 Model681. The results are as follow:

101.tomcatv : 16.9times/28PE by SGI Origin2000
102.swim : 9.0times/8PE by COMPAQ_Alpha
103.su2cor : 2.5times/4PE by IBM RS/6000
104.hydro2d : 3.2times/8PE by COMPAQ_Alpha
107.mgrid : 3.3times/8PE by COMPAQ_Alpha
110.applu: 17.8times/31PE by SGI Origin2000
125.turb3d: 10.1times/24PE by SGI Origin2000
141.apsi : 1.0times/1PE by All Systems
145.fpppp: 1.1times/2PE by SGI Origin2000
146.wave5: 1.0times/1PE by All Systems

The scalability results of SPEC CFP2000 are below,

168.wupwise : 1.1times/20PE by SGI Origin2000
171.swim : 4.5times/7PE by IBM RS/6000
172.mgrid : 17.3times/32PE by SGI Origin2000
173.applu : 17.3times/29PE by SGI Origin2000
200.sixtrack : 1.0times/1PE by All Systems
301.apsi : 1.0times/2PE by SGI Origin2000

The scalability results of NPB are below,

EP : 1.0times/2PE by SGI Origin2000
MG : 2.9times/8PE by COMPAQ_Alpha
CG: 29.6times/27PE by SGI Origin2000
FT : 1.1times/14PE by IBM pSeries690
LU : 11.7times/31PE by SGI Origin2000
SP : 2.1times/5PE by IBM RS/6000
BT : 3.6times/29PE by SGI Origin2000

From the result of these tests we can categorize the benchmarks as follow. We use the best scalability of our four environments in this categorization. In addition, some benchmarks are classified as high-leveled parallelism benchmarks on one environment, but are classified as low-leveled parallelism benchmarks on another. For example, ‘applu’ is classified as high-leveled on SGI Origin2000, but classified as low-leveled on COMPAQ_Alpha and IBM RS/6000.

- High level parallelism benchmarks:
  101.tomcatv, 102.swim, 110.applu, 171.swim, 172.mgrid, 173.applu, CG
- Low level parallelism benchmarks:
  103.su2cor, 104.hydro2d, 107.mgrid, 125.turb3d, 145.fpppp, 168.wupwise, MG, FT, LU, SP, BT
- Difficult to parallelize benchmarks
  141.apsi, 146.wave5, 200.sixtrack, 301.apsi, EP

The guideline of the evaluation will be as follows.

For high-leveled parallelism benchmarks, it will be impossible to achieve the double performance, but any of the performance improvement will be achieved.

For low-level parallelism benchmarks, it will be the best candidate to achieve the double performance.

For difficult to parallelize benchmarks, if some of our technology can apply to these benchmarks, the performance improvement will be more than the double. In this case we can use any number of CPUs to improve the performance.

We used these 23 benchmarks for our final evaluation of this project.

2.2. Choice of compile options

To evaluate the benchmarks it is important to choose the performance compiler options. The commercial compilers used for comparison also are used as back-end compilers of APC compiler, so the same option set must be used for the final comparison for fairness of this evaluation. This means that these options set must also be robust enough.

We selected the compiler options published by each vendor’s SPEC CFP2000_base reports with automatic parallelization options of each compiler. The compiler options are as follows.

- SGI Origin2000: -Ofast=ip27 -LNO:fusion=2 -mp
- COMPAQ_Alpha: -v -arch ev6 -O5 -fkapargs='-ur=1' -fkapargs='-noconc' -omp -pthread -call_shared
- IBM RS/6000: -O5 -qarch=ppc –qhot -qsmp=noauto
- IBM pSeries690: -O5 -qfixed -qarch=pwr4 -qhot -qsmp=omp -qsmp=noauto

2.3. The evaluation environments, and the results

Our evaluation environments consist of SMP machines and a set of compilers as follows.

- SGI Origin2000
  R10000@195MHz, 32CPUs, 11GB Memory
  MIPSpro Fortran90 V7.30
- COMPAQ AlphaServer GS160 Model 6/731
  Alpha21264@731MHz, 8CPUs, 4GB Memory
  Compaq Fortran V5.4-1283-46ABA
  KAP Fortran V4.3
- IBM RS/6000
  PowerPC604@200MHz, 8 CPUs, 1GB Memory
  IBM XL Fortran 7.1.0
- IBM pSeries690 Model681
  Power4 @1.1GHz, 16 CPUs, 8GB Memory
  IBM XL Fortran 8.1

The evaluation results are follows.

- IBM RS/6000:
  13 out of 23 Benchmarks are highly parallelized by APC compiler from only 4 Benchmarks by the Original one. Average 2.4 times faster than the Original.
- IBM pSeries690:
  7 out of 23 Benchmarks are highly parallelized by APC compiler from no Benchmarks by the Original one. Average 3.5 times faster than the Original.
- COMAPQ_alpha:
  8 out of 23 Benchmarks are highly parallelized by APC compiler from 3 Benchmarks by the Original one. Average 2.1 times faster than the Original.
- SGI Origin2000:
  6 out of 23 Benchmarks are highly parallelized by APC compiler from 6 Benchmarks by the Original one. Average 2.7 times faster than the Original.

The scalability results of each environment and each benchmark are listed below. We use SPEC CFP2000_base compiler options of each compiler described in section 2.2 to get

SPEC CFP95:

101.tomcatv:

```
SGI Origin2000   :  1.0times ( 16.4times / 29PE from 16.9times / 28PE)
IBM RS/6000       :  4.0times (  7.2times /  8PE from  1.8times /  4PE)
COMPAQ_Alpha      :  3.3times (  6.0times /  8PE from  1.8times /  6PE)
IBM pSeries690   :  5.1times (  6.1times / 15PE from  1.2times /  4PE)
```

```
  102.swim:
SGI Origin2000   :   1.0times ( 22.0times / 27PE from 22.1times / 29PE )
IBM RS/6000       :   2.0times (   9.5times /   8PE from   4.8times /   6PE )
COMPAQ_Alpha      :   1.3times ( 11.5times /   8PE from   9.0times /   8PE )
IBM pSeries690   :   4.8times ( 11.1times / 16PE from   2.3times /   3PE )
  103.su2cor:
SGI Origin2000   :   0.9times (   3.7times / 11PE from   4.3times / 16PE )
IBM RS/6000       :   2.0times (   5.0times /   7PE from   2.5times /   4PE )
COMPAQ_Alpha      :   1.2times (   3.7times /   8PE from   3.0times /   8PE )
IBM pSeries690   :   3.0times (   3.0times / 14PE from   1.0times /   1PE )
  104.hydro2d:
SGI Origin2000   :   0.5times (   4.8times / 15PE from   8.9times / 27PE )
IBM RS/6000       :   3.7times (   8.6times /   8PE from   2.3times /   4PE )
COMPAQ_Alpha      :   1.0times (   3.3times /   8PE from   3.2times /   8PE )
IBM pSeries690   :   7.3times ( 10.2times / 14PE from   1.4times /   2PE )
  107.mgrid:
SGI Origin2000   :   0.9times (   7.4times / 16PE from   8.4times / 16PE )
IBM RS/6000       :   2.2times (   6.8times /   8PE from   3.1times /   4PE )
COMPAQ_Alpha      :   1.9times (   6.2times /   8PE from   3.3times /   8PE )
IBM pSeries690   :   5.4times ( 11.3times / 16PE from   2.1times /   3PE )
  110.applu:
SGI Origin2000   :   0.3times (   4.6times / 32PE from 17.8times / 31PE )
IBM RS/6000       :   2.6times (   3.7times /   8PE from   1.4times /   8PE )
COMPAQ_Alpha      :   3.4times (   3.4times /   8PE from   1.0times /   4PE )
IBM pSeries690   :   1.8times (   2.1times / 4PE from   1.2times /   3PE )
  125.turb3d:
SGI Origin2000   :   1.5times ( 14.8times / 24PE from 10.1times / 24PE )
IBM RS/6000       :   6.2times (   6.2times /   8PE from   1.0times /   1PE )
COMPAQ_Alpha      :   6.3times (   6.3times /   8PE from   1.0times /   1PE )
IBM pSeries690   : 10.1times ( 11.1times / 16PE from   1.1times /   3PE )
  141.apsi:
SGI Origin2000   :   1.0times (   1.0times /   1PE from   1.0times /   1PE )
IBM RS/6000       :   1.0times (   1.0 times /   1PE from   1.0times /   1PE )
COMPAQ_Alpha      :   1.0times (   1.0times /   1PE from   1.0times /   1PE )
IBM pSeries690   :   1.0times (   1.0times /   1PE from   1.0times /   1PE )
  145.fpppp:
    SGI Origin2000   :   1.0times (   1.1times /   7PE from   1.1times /   2PE )
    IBM RS/6000       :   1.0times (   1.0times /   1PE from   1.0times /   1PE )
    COMPAQ_Alpha      :   1.0times (   1.0times /   1PE from   1.0times /   1PE )
    IBM pSeries690   :   1.0times (   1.0times /   4PE from   1.0times /   1PE )
      146.wave5:
    SGI Origin2000   :   1.0times (   1.0times /   1PE from   1.0times /   1PE )
    IBM RS/6000       :   1.0times (   1.0times /   1PE from   1.0times /   1PE )
    COMPAQ_Alpha      :   1.0times (   1.0times /   1PE from   1.0times /   1PE )
    IBM pSeries690   :   1.0times (   1.0times /   1PE from   1.0times /   1PE )
SPEC CFP2000:
      168.wupwise:
    SGI Origin2000   : 10.8times ( 11.9times / 29PE from   1.1times / 20PE )
    IBM RS/6000       :   5.2times (   5.2times /   8PE from   1.0times /   1PE )
    COMPAQ_Alpha      :   5.8times (   5.8times /   7PE from   1.0times /   1PE )
    IBM pSeries690   :   4.4times (   4.4times / 10PE from   1.0times /   1PE )
      171.swim:
    SGI Origin2000   :   1.4times ( 13.7times / 31PE from   9.5times / 31PE )
    IBM RS/6000       :   1.6times (   7.1times /   8PE from   4.5times /   7PE )
    COMPAQ_Alpha      :   1.0times (   2.7times /   8PE from   2.7times /   8PE )
    IBM pSeries690   :   3.0times (   8.0times / 15PE from   2.7times /   7PE )
      172.mgrid:
    SGI Origin2000   :   1.0times ( 16.6times / 31PE from 17.3times / 32PE )
    IBM RS/6000       :   1.5times (   6.4times /   8PE from   4.2times /   8PE )
    COMPAQ_Alpha      :   1.1times (   4.5times /   8PE from   4.1times /   8PE )
    IBM pSeries690   :   3.5times ( 10.1times / 15PE from   2.9times /   7PE )
      173.applu:
    SGI Origin2000   :   0.4times (   6.1times / 31PE from 17.3times / 29PE )
    IBM RS/6000       :   2.9times (   4.6times /   8PE from   1.6times /   7PE )
    COMPAQ_Alpha      :   2.9times (   3.5times /   8PE from   1.2times /   4PE )
    IBM pSeries690   :   1.8times (   2.7times / 10PE from   1.5times /   6PE )
      200.sixtrack:
    SGI Origin2000   :   1.0times (   1.0times /   1PE from   1.0times /   1PE )
    IBM RS/6000       :   1.0times (   1.0times /   1PE from   1.0times /   1PE )
    COMPAQ_Alpha      :   1.0times (   1.0times /   1PE from   1.0times /   1PE )
    IBM pSeries690   :   1.0times (   1.0times /   1PE from   1.0times /   1PE )
      301.apsi:
```

```
    SGI Origin2000   :   1.0times (   1.0times /   1PE from   1.0times /   2PE )
    IBM RS/6000      :   1.0times (   1.0times /   1PE from   1.0times /   1PE )
    COMPAQ_Alpha     :   1.0times (   1.0times /   1PE from   1.0times /   1PE )
    IBM pSeries690   :   1.0times (   1.0times /   1PE from   1.0times /   1PE )
NPB:
      EP :
    SGI Origin2000   : 20.8times ( 20.8times / 32PE from   1.0times /   2PE )
    IBM RS/6000      :   7.5times (   7.5times /   8PE from   1.0times /   1PE )
    COMPAQ_Alpha     :   6.9times (   6.9times /   8PE from   1.0times /   1PE )
    IBM pSeries690   : 14.1times ( 14.1times / 16PE from   1.0times /   1PE )
      MG :
    SGI Origin2000   :   5.6times (   6.7times / 32PE from   1.2times /   2PE )
    IBM RS/6000      :   3.3times (   3.3times /   8PE from   1.0times /   3PE )
    COMPAQ_Alpha     :   0.6times (   1.7times /   7PE from   2.9times /   8PE )
    IBM pSeries690   :   2.1times (   2.1times / 8PE from   1.0times /   1PE )
      CG :
    SGI Origin2000   :   1.1times ( 33.0times / 27PE from 29. 6times / 27PE )
    IBM RS/6000      :   1.2times (   5.5times /   8PE from   4.5times /   7PE )
    COMPAQ_Alpha     :   1.3times (   5.9times /   8PE from   4.5times /   8PE )
    IBM pSeries690   :   1.5times (   4.3times / 15PE from   2.9times /   7PE )
      FT :
    SGI Origin2000   :   1.0times (   1.0times /   1PE from   1.0times /   1PE )
    IBM RS/6000      :   1.0times (   1.0times /   1PE from   1.0times /   1PE )
    COMPAQ_Alpha     :   1.0times (   1.0times /   1PE from   1.0times /   1PE )
    IBM pSeries690   :   0.9times (   1.0times /   1PE from   1.1times / 14PE )
      LU :
    SGI Origin2000   :   0.4times (   4.3times / 13PE from 11.7times / 31PE )
    IBM RS/6000      :   1.8times (   2.7times /   8PE from   1.5times /   8PE )
    COMPAQ_Alpha     :   2.7times (   3.0times /   8PE from   1.1times /   4PE )
    IBM pSeries690   :   2.4times (   3.4times / 12PE from   1.4times /   2PE )
      SP :
    SGI Origin2000   :   7.0times ( 16.0times / 32PE from   2.3times / 31PE )
    IBM RS/6000      :   2.5times (   5.3times /   8PE from   2.1times /   5PE )
    COMPAQ_Alpha     :   1.6times (   3.5times /   8PE from   2.2times /   8PE )
    IBM pSeries690   :   2.0times (   3.2times /   7PE from   1.6times /   3PE )
      BT :
```

SGI Origin2000 : 1.0times (3.7times / 18PE from 3.6times / 29PE)
IBM RS/6000 : 0.9times (1.0times / 1PE from 1.1times / 7PE)
COMPAQ_Alpha : 0.8times (1.0times / 1PE from 1.3times / 4PE)
IBM pSeries690 : 1.2times (1.4times / 5PE from 1.2times / 7PE)

2.4. Analysis of SPEC CFP95 benchmarks

The ten benchmark programs (101.tomcatv, 102.swim, 103.su2cor, 104.hydro2d, 107.mgrid, 110.applu, 125.turb3d, 141.apsi, 145.fpppp, 146.wave5) in SPEC CFP95 written in Fortran77 have been selected as the programs which are used for the overall performance evaluation. This section shows the properties of these programs by measuring the parallel processing times of these programs.

2.4.1. Target benchmark programs

All of the ten benchmark programs are written in FORTRAN77.

The smallest program, 101.tomcatv, consists of 106 lines, 0 subroutines/functions and one file. The largest program, 146.wave5, consists of 6,347 lines, 76 subroutines , 15 functions and 2 files (Table 2.4.1-1).

Table 2.4.1-1 Size of programs in CFP95

Program	# Lines	# Subroutines	# Functions	# Files
101.tomcatv	106 (190)	0	0	1
102.swim	260 (449)	5	0	1
107.mgird	330 (484)	11	0	1
125.turb3d	1280(2100)	22	0	1
103.su2cor	1558(2332)	20	4	2
104.hydro2d	1710(4292)	36	5	2
145.fpppp	2135(2790)	10	27	38
110.applu	2423(3868)	15	0	1
141.apsi	4225(7361)	93	3	1
146.wave5	6347(7764)	76	15	2

2.4.2. Measurement environment

The multiprocessor system described below is used for this measurement.

Sun Ultra80, 4CPU, SunOS5.8 7/01

Following list shows the parallelizing compiler and the parallelizing preprocessor which are used for the measurement.

- Sun Forte Developer 6 update 2, f95 parallelizing compiler
- Visual KAP for OpenMP 3.9 parallelizing preprocessor

Sun Forte compiler generates a parallel object code by parallelizing a serial source program and also generates a parallel object code from OpenMP source program.

Visual KAP for OpenMP is a preprocessor which parallelizes a serial source program and generates OpenMP code.

Three types of object codes listed below are executed on the machine.

- The serial object code (S) which is generated by Forte compiler from the CFP95 serial source program.
- The parallel object code (P)which is parallelized and generated by Forte compiler from the CFP95 serial source program.
- The parallel object code (K) which is generated by Forte compiler from the OpenMP code parallelized by Visual KAP from the CFP95 serial source program.

Table 2.4.2-1 Compiling methods

Code	Compilation
Serial (S)	f95 –fast <CFP95 prog.>
Parallel (P)	f95 –fast –autopar –reduction –stackvar < CFP95 prog.>
Paralle (K) by KAP	Parallelizing <CFP95 prog.> to OpenMP by Visual KAP for OpenMP f95 –fast –mp=openmp –explicitpar –stackvar <OpenMP prog.>

KAP Options ：**Optimization (5) Scalar Opt.(1) Roundoff(3)**
Fuse Loops(High)IPA (High) Unrolling(0) /WK,/nocmpoptions

2.4.3. Measurement results and property of each program

Based on the measurement results, we can categorize these programs into three groups: highly loop-parallelized programs(102.swim, 107.mgrid, 103.su2cor, 104.hydro2d), moderate loop-parallelized programs (101.tomcatv, 110.applu, 125.turb3d) and almost sequential programs (141.apsi, 145.fpppp, 146.wave5). The characteristics of these groups are described below.

- 102.swim, 107.mgrid, 104.hydro2d:

  Parallelized code compiled by each compiler can get the high parallel processing effect. It suggests that these programs can be fully parallelized by the loop-level parallelizing compiler. These programs should be used not for the loop-parallelism detection but for the memory access optimization by the multi-grain parallel processing.
- 101.tomcatv, 110.applu, 125.turb3d:

  Parallelized code compiled by each compiler can get the moderate parallel processing effect. It suggests that these programs can not be fully parallelized by the loop-level parallelizing compiler. These programs should be used for the evaluation of the memory access optimization, the inter procedure analysis and the coarse grain parallel processing effect by the multi-grain parallel processing.

  For example, the fact that the loops 101, 201, 301 and 401 in the subroutine

TURB3D, which have enough loop parallelisms, can not be parallelized by Forte compiler due to the subroutine calls in the loops implies that the 125.turb3d are suitable for the evaluation of the function to detect parallelism of loops which contain subroutine calls.

- 141.apsi, 145.fpppp, 146.wave5:

  Parallelized code compiled by each compiler can not get parallel processing effect at all. It suggests that these programs can not be parallelized by the conventional parallelizing compilers at all and can be appropriate for the overall evaluation of the multigrain parallelizing compiler.

2.4.4. SPEC CFP95 and CFP2000 for the overall performance evaluation

We have selected 10 programs from SPEC CFP95 and 6 programs from SPEC CFP2000 for the overall performance evaluation. The set of theses programs covers the large range in the size of programs (from 106 lines of 101.tomcatv to 41,418 lines of 200.sixtrack). Although, for four programs (swim, mgrid, applu, apsi), the same codes from both CFP95 and CFP2000 are doubly selected, it is valuable to contain all of them for the measurement because each code has the distinct data set and the run path.

2.5. Analysis of NAS benchmarks

2.5.1. Performance estimates of the multigrain parallelization

We performed experiments to estimate performance of the multigrain parallelization with data localization, the loop aligned decomposition, on IBM pSeries690, which is selected as a multiprocessor system for performance evaluation of the APC compiler. The loop aligned decomposition is a data localization scheme to improve performance of the multigrain parallelization. The scheme tries to improve cache locality by parallelizing a sequence of DOALL loops so that iterations assigned to the identical processor read/write the same region of data. The analytical results for coarse grain parallelism of benchmark programs, which we obtained in 2001, show that we need to perform the multigrain parallelization with some program data localization, such as the loop aligned decomposition, in order to exploit coarse grain parallelism effectively improving the performance on the NAS Parallel Benchmarks.

In the experiment, we compared execution time of multiple SP codes, which were parallelized as follows: (1) the multigrain parallel code, in which a sequence of multiple loops is manually parallelized by the multigrain parallelization with the loop aligned decomposition, (2) the automatic parallel codes, which are automatically parallelized by vendor parallelizing compilers, the IBM XL Fortran 7.1 and the Visual KAP for OpenMP 3.9. The result showed that the multigrain parallelization with the loop aligned decomposition yielded speedup which is 10%-40% faster than the automatic parallelization by the vendor parallelizing compilers.

2.5.2. Analysis of loop parallelism

We analyzed loop parallelism on the NAS Parallel Benchmarks in order to discuss the performance evaluation results of the APC compiler. The analysis is performed by comparing two versions of parallelized codes in the NAS Parallel Benchmarks 2.3: the codes manually parallelized to exploit effective loop parallelism and the codes automatically parallelized by the Visual KAP for OpenMP 3.9. (We denote "KAP" in the rest of the section.)

The results showed that the performance of the code automatically parallelized by KAP was lower than the manually parallelized codes. Also, the results revealed reasons for performance degradation in the codes parallelized by KAP:

- KAP exploits loop parallelism as much as the manually parallelized codes, but it generates many codes for synchronization and thread fork/join, which increase overhead. (CG, MG, LU, SP, BT)
- KAP does not precisely analyze parallelism of loops that include subroutine calls. (FT)
- KAP parallelizes small loops that do not yield performance improvement and they increase overhead. (FT)

III Lists of Accomplishment and Other Reference Materials

1. Fujitsu Ltd.

（Infrastructure Technology for Multi-grain Parallelism Exploitation）

・International Conference(1)

（1）Akira Hosoi, and Toshihiro Ozawa,:"A New Array Contraction Method",10th International Workshop on Compilers for Parallel Computers (CPC2003),pp. 127-136, Amsterdam, The Netherlands, January 2003.

・Symposium with Review(1)

（1）Akira Hosoi, and Toshihiro Ozawa,"A New Array Contraction Method":High Performance Computing Symposium 2003 (HPCS2003), pp. 125-132, January 2003.

（Overall Evaluation Method）

・International Conference(1)

（1）Naoki Sueyasu, Hidetoshi Iwashita, Kohichiro Hotta, Matthijs van Waveren, and Kenichi Miura,:"Scalability of SPEC OMP on Fujitsu PRIMEPOWER", In Proc. of 4th European Workshop on OpenMP (EWOMP2002), Rome, Italy, September 2002.

2. Hitachi,Ltd.

（Feedback-directed Selection Technique of Compiler Directives）

（1）Kiyomi Wada, Makoto Satoh, Takayoshi Iitsuka:"Feedback-directed Selection Technique of Compiler Directives for Parallelized Compiler",IPSJ SIG NOtes, Vol.2003, No.10,2003-ARC-151, Jan. 2003.

（Program Visualization Technique）

（1）Makoto Satoh, Kiyomi Wada:" Development of Interprocedural Data-Dependence Locator in Parallel Tuning Tool Aivi",IPSJ SIG NOtes, 2003-HPC-93 (to appear), Mar. 2003.

（Automatic Data Distribution Technique）

（1）Takashi Hirooka :"Optimization for Indirect Array References Using First Touch Control",IPSJ Transactions on High Performance Computing Systems Vol.43 No.SIG 06 - 005(JSPP2003), May 2002.

3. Waseda University,Kasahara Laboratory

（Technology for Multigrain Parallelism Exploitation Infrastructure）

・Papers (8)

（1）Takeshi Kodaka, Takayuki Uchida, Keiji Kimura, Hironori Kasahara, "JPEG Encoding Using Multigrain Parallel Processing on a Single Chip Multiprocessor", Trans. of IPSJ on High Performance Computing Systems, pp. 153--162, Vol. 43, No. Sig. 6 (HPS5), Sep., 2002.

（2）Motoki Obata, Jun Shirako, Hiroki Kaminaga, Kazuhisa Ishizaka, Hironori Kasahara, “Hierarchical Parallelism Control Scheme for Multigrain Parallelization”, Trans. of IPSJ, Vol. 44, No. 4, Apr., 2003 (to appear).

（3）Keiji Kimura, Takeshi Kodaka, Motoki Obata, Hironori Kasahara, “Multigrain Parallel Processing on Compiler Cooperative OSCAR Chip Multiprocessor Architecture”, The IEICE Transactions on Electronics, Special Issue on High-Performance and Low-Power System LSIs and Related Technologies, Apr., 2003 (to appear).

（4）Motoki Obata, Jun Shirako, Hiroki Kaminaga, Kazuhisa Ishizaka, Hironori Kasahara, “Hierarchical Parallelism Control for Multigrain Parallel Processing”, Proc. of 15th International Workshop on Languages and Compilers for Parallel Computing (LCPC2002), Aug., 2002.

（5）Hironori Kasahara, Motoki Obata, Kazuhisa Ishizaka, Keiji Kimura, Hiroki Kaminaga, Hirofumi Nakano,Kouhei Nagasawa, Akiko Murai, Hiroki Itagaki, Jun Shirako,“Performance of Multigrain Parallelization in Japanese Millennium Project IT21 Advanced Parallelizing Compiler”, Proc. of 10th International Workshop on Compilers for Parallel Computers (CPC) Amsterdam, Netherland, January 2003.

（6）Keiji Kimura, Takeshi Kodaka, Motoki Obata, Hironori Kasahara, “Multigrain Parallel Processing on OSCAR CMP”, Proc. of International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems (IWIA'03), Jan., 2003.

（7）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), Warsaw, Poland, Sep., 2002. (See also Invited Talks)

（8）“NEDO-1 Advanced Parallelizing Technology”, IPSJ-IEICE FIT2002 (Forum on Information Technology), National Project Introduction, Tokyo Institute of Technology, Sep.27, 2002

・Symposium (1)

（1）Takeshi Kodaka, Takayuki Uchida, Keiji Kimura, Hironori Kasahara, “JPEG Encoding using Multigrain Parallel Processing on a Shingle Chip Multiprocessor”, Joint Symposium on Parallel Processing 2002 (JSPP2002), May., 2002.

・Technical reports (6)

(1) Yasutaka Wada, Hirofumi Nakano, Keiji Kimura, Motoki Obata, Hironori Kasahara, “Evaluation of Overhead with Coarse Grain Task Parallel Processing on SMP Machines”, Technical Report of IPSJ, ARC2002-148-3, May., 2002.

(2) Jun Shirako, Hiroki Kaminaga, Noriaki Kondo, Kazuhisa Ishizaka, Motoki Obata, Hironori Kasahara, “Coarse Grain Task Parallel Processing with Automatic Determination Scheme of Parallel Processing Layer”, Technical Report of IPSJ, ARC2002-148-4, May., 2002.

(3) Motoki Obata, Jun Shirako, Kazuhisa Ishizaka, Hironori Kasahara, “Performance of OSCAR Multigrain Parallelizing Compiler on SMPs”, Technical Report of IPSJ, ARC2002-149-20(SWoPP2002), Aug., 2002.

(4) Takeshi Kodaka, Takahisa Suzuki, Keiji Kimura, Hironori Kasahara, “Multigrain Parallel Processing on Motion Vector Estimation for Single Chip Multiprocessor”, Technical Report of IPSJ, ARC2002-150-6, Nov, 2002.

(5) Keiji Kimura, Takeshi Kodaka, Motoki Obata, Hironori Kasahara, “Multigrain Parallel Processing on OSCAR Chip Multiprocessor”, Technical Report of IPSJ, ARC2002-150-7, Nov, 2002.

(6) Jun Shirako, Kouhei Nagasawa, Kazuhisa Ishizaka, Motoki Obata, Hironori Kasahara, “Inline Expansion for Improvement of Multi Grain Parallelism”, Technical Report of IPSJ, ARC2003-151-2 (SHINING2003), Jan., 2003.

・Invited talks (6)

(1)“Multigrain Parallel Processing in Millennium Project IT21 Advanced Parallelizing Compiler”, Sig. on Autonomous Distributed Systems, Nagoya University, Aug. 30,2002., Hosted by Prof. Toshio Fukuda

(2) “ Performance of Multigrain Parallelization in Japanese Millennium Project IT21 "Advanced Parallelizing Compiler" ” , Computer Engineering Seminar, Univ. Illinois at Urbana-Champaign, Sep. 3, 2002. Hoseted by Prof. David Padua

(3) “ Multigrain Parallel Processing in Japanese Millennium Project IT21 "Advanced Parallelizing Compiler" ” , Distinguished Lecture ECE Graduate Seminar, Purdue University, Sep. 5, 2002. Hosted by Prof. Rudolf Eigenmann

(4) “ OSCAR Multigrain Parallelizing Compiler for Chip Multiprocessors to High Performance Severs ”, Polish-Japanese Institute of Information Technology (PJIIT), Sep., 2002. Hosted by Prof. Marek Tudruj

(5) “ Multigrain Parallelizing Compiler for Chip Multiprocessors to High Performance Severs ” , Intel ICRC, the People's Republic of China, Nov.6, 2002

（6）“ Multigrain Parallelization in Japanese Millennium Project IT21 "Advanced Parallelizing Compiler" ” Chinese Academy of Science (ICT), the People's Republic of China, Nov.7, 2002

・Invited survey Paper (1)

（1）Hironori Kasahara, ”Advanced Automatic Parallelizing Compiler Technology”, IPSJ（Information Processing Society of Japan）, Vol.44, No.4, Apr., 2003

・Research Report(1)

（1）Hironori Kasahara, "Multigrain Parallel Processing inAdvanced Paralleling Compiler Project",Investigation Research IV on Highend Computing Technology", JIPDEC AITEC, Mar. 2003.

（Scheduling Technology）

・Papers (4)

（1）Takao Tobita, Hironori Kasahara, “Performance Evaluation of Minimum Execution Time Multiprocessor Scheduling Algorithms Using Standard Task Graph Set”, Trans. of IPSJ, Vol. 43, No. 4, Apr., 2002.

（2）Kazuhisa Ishizaka, Hirofumi Nakano, Satoshi Yagi, Motoki Obata, Hironori Kasahara, “Coarse Grain Task Parallel Processing with Cache Optimization on Shared Memory Multiprocessor”, Trans. of IPSJ, Vol. 43, No. 4, Apr., 2002.

（3）Hirofumi Nakano, Kazuhisa Ishizaka, Motoki Obata, Keiji Kimura, Hironori Kasahara, “Static Coarse Grain Task Scheduling with Cache Optimization Using OpenMP”, Proc. of WOMPEI, 2002.

（4）Takao Tobita, Hironori Kasahara, “A standard task graph set for fair evaluation of multiprocessor scheduling algorithms ” , Journal of scheduing, John Wiley & Sons Ltd, Oct., 2002.

・Technical reports (2)

（1）Kazuhisa Ishizaka, Hirofumi Nakano, Motoki Obata, Hironori Kasahara, “Cache Optimization among Coarse Grain Tasks considering Line Conflict Miss”, Technical Report of IPSJ, ARC2002-149-25(SWoPP2002), Aug., 2002.

（2）Hirofumi Nakano, Takeshi Kodaka, Keiji Kimura, Hironori Kasahara, “Data Localization using Coarse Grain Task Parallelism on Chip Multiprocessor”, Technical Report of IPSJ, ARC2003-151-3 (SHINING2003), Jan., 2003.

・Technical Report(1)

（1）Kazuhisa Ishizaka, Motoki Obata, Kasahara Hironori,"Inter-array Padding for Data Localization using StaticScheduling", Technical Report of IPSJ, May, 2003.

4. Waseda University,Yamana Laboratory

・Technical Report

（1）F.Saito, K.Kitamura, H.Yamana:"Necessity for Confidence in Multiple PHT Branch Predictors",IPSJ Tech. Report(ARC),Vol.2002, No.81, pp.55-60 (2002.8)

（2）F.Saito, T.Hiruta, H.Yamana:"Hybrid Branch Predictors Evaluation on Prediction Accuracy",IPSJ Tech. Report(ARC), Vol.2002, No.112, pp.89-94(2002.11)

（3）S.Ishikawa, H.Yamana:"A Speedup Technique to Difficultly Pallalyzing Loops Using Speculative Execution",IPSJ 65th National Conf.,3ZA-4(2003.3)

5. Toho University, Yoshida Laboratory

(Data Locality Optimization Technology)

・Journal(1)

（1）Akimasa Yoshida: "Execution Scheme Using Task Overlapping Assignment for Hierarchical Coarse Grain Parallel Processing", IPSJ Journal, Vol.43, No.4, pp.926-935, Apr. 2002.

・International Conference(1)

（1）Akimasa Yoshida: "An Overlapping Task Assignment Scheme for Hierarchical Coarse Grain Parallel Processing", Proc. of 10th International Workshop on Compilers for Parallel Computers, Jan. 2003.

・Technical Report(1)

（1）A.Yoshida, T.Aramaki, T.Ohmori: "Layer Unified Scheduling for Coarse Grain Task Parallel Processing", SIG Notes of IPSJ, 2002-ARC-149-21, Aug. 2002.

６. Tokyo Institute of Technology

・International Conference(2)

（1）Kento Aida, Yoshiaki Futakata, Shinji Hara,“High-performance Parallel and Distributed Computing for the BMI Eigenvalue Problem,” Proc.　16th IEEE International Parallel and Distributed Processing Symposium,Apr. 2002

（2）Kento Aida, Wataru Natsume, Yoshiaki Futakata,“Distributed Computing with Hierarchical Master-worker Paradigm for Parallel Branch and Bound Algorithm,” Proc. 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid,(accepted)

・Technical Report(2)

（1）Wataru Natsume, Kento Aida, "Grid Computing scheme for the BMI Eigenvalue Problem with Hierarchical Master-Worker Paradigm," IPSJ SIG Notes, 2002-HPC91, pp.73-78, Aug. 2002

（2）Masaki Kan, Kento Aida, "Evaluation methodology of parallel simulation accuracy for the desaster relief act," IPSJ SIG Notes, 2002-HPC91, pp.185-190, Aug. 2002

７.　The University of Electro-Communications

International Conference(1)

（1）Macro-Data-Flow using Software Distributed Shared Memory Hiroshi Tanabe, Hiroki Honda, Toshitsugu Yuba Information Processing Society of Japan, SIG Notes, Vol.2003, No.27, pp.37-42 (2003)

－　禁無断転載　－

アドバンスト並列化コンパイラ技術

発　行　平成１５年３月

発行所　財団法人日本情報処理開発協会

〒105－0011　　東京都港区芝公園３－５－８

機械振興会館内

電　話　　03－3432－9390

ＦＡＸ　　03－3431－4324