Advanced Parallelizing Compiler Technology

2000 Annual Report

March 2001

Japan Information Processing Development Corporation

Introduction

This Advanced Parallelizing Compiler (APC) Technology Research and Development Project started its R&D activities in October, 2000, in trust by New Energy and Industrial Technology Development Organization (NEDO) as a software R & D project in Japanese Government Millennium Project.

This project researches and develops automatic parallelizing compiler technologies that enhance effective performance, cost-performance and ease of use of multiprocessor systems which have been widely used for HPC (High Performance Computers) and have attracted much attention as a fundamental architecture for advanced microprocessors like single-chip-multiprocessors used for next generation PCs (Personal Computers), cellular phones, video games and so on. The R&D is performed in a short period of three years considering hard international competitive environment.

R & D topics in APC project are

(1) Development of advanced parallelizing compiler technology

(2) Development of performance evaluation technology for parallelizing compilers

R & D status of these two topics in the first year of APC project is reported in Chapters I and II in this report.

The goals of the APC project are to enhance effective performance twice and to improve ease of use of shared memory multiprocessor systems. Toward the goals, the R&D has started smoothly.

By attaining these goals, APC project aims at strengthening of IT competitiveness. It is expected that the technology developed in this project will be widely used for various systems including future intelligent cellular phones, video games, household information electric appliances etc. in addition to HPC and will contribute strengthening of competitiveness of R&D fields, such as automobile, aviation & space, semiconductor, environment, humanoid, bio, medicine, energy, requiring high performance computers.

CONTENTS

I Development of Advanced Parallelizing Compiler Technology

To maximize the performance efficiency of a program on multi-processor systems, automatic parallelizing compilers must exploit not merely simpler loop parallelism in the program but the following complex parallelism such as the coarse-grain parallelism, which exists in subroutines themselves, in loops including calls to subroutines, and between loops; and the fine-grain parallelism, which exists among basic blocks. To solve those problems, we have been researching and developing the automatic multi-grain parallelizing technique, which makes the best use of multi-grain parallelism in a given program, and the tuning technology for parallel processing, which enables to enhance the compiler's parallelization of the program by feedbacking run-time information or user's knowledge to the compiler.In this fiscal year, we have been investigating several element techniques that enable us to realize those two technologies, conducting the fundamental examination of these techniques, and done some preliminary evaluations.

1. Development of Automatic Multi-grain Parallelizing Compiler Technology

Our goal in this research and development is to establish the automatic multi-grain parallelizing technique that makes the best use of multi-grain parallelism in a given program. To achieve our goal we research and develop the following element techniques of the automatic multi-grain parallelizing technology: the multigrain parallelism exploitation technology (the base technique in the automatic multi-grain parallelizing technology), the data-dependence analysis technique (the technique analyzing data-dependence relationship, the basic information to extract parallelism from a given program), the automatic data-distribution technique (the technique minimizing data reference time by arranging certain data as near their referencing processors as possible), the speculative execution technology (the technique minimizing program execution time by executing a task depending on another task speculatively before completion of the latter task), the scheduling techniques (the technique minimizing processors ' idle time by optimizing the execution order of tasks and the their allocation to the processors), and the common interface language among some optimization functions (the technique adding the multi-grain parallelism to the language that enables programmers to describe a platform-free parallelism). In this fiscal year, we have been conducting the fundamental examination of each of those element techniques and have done the preliminary evaluation for some of them.

1.1 Multigrain Parallelism Exploitation Technology

This section describes multigrain parallelism exploitation technology as an infrastructure of automatic multigrain parallelization. It aimes atresearch and development of parallelism analysis technique from sourceprograms and of realization scheme of multigrain parallel processing on shared memory multiprocessor systems. This year, as the first year of the project, a parallelized program generation technique using OpenMP API to exploit multigrain parallelism on shared memory multiprocessor systems available on the market, coarse grain patallelism exploitaion techniques among procedures and parallelism exploitation techniques from imperfectly nested loops considering data locality have been researched and developed.

1.1.1 Transformation techniques into OpenMP program with multigrain parallelism

This section proposes an implementation method named “one-time single level thread generation” for a coarse grain task parallel processing scheme on an off the shelf shared memory multiprocessor system.

The coarse grain task parallel processing is important to improve the effective performance of wide range of multiprocessor systems from a single chip multiprocessor to a high performance computer beyond the limit of the loop parallelism. The loop parallelization techniques, such as Do-all and Do-across, have been widely used in Fortran parallelizing compilers for multiprocessor systems. However, these compilers cannot parallelize loops that include complex loop carrying dependences and conditional branches to the outside of a loop. Considering these facts, the coarse grain task parallelism should be exploited to improve the effective performance of multiprocessor systems further in addition to the improvement of data dependence analysis, speculative execution and so on.

The proposed scheme decomposes a Fortran program into coarse grain tasks, analyzes parallelism among tasks by “Earliest Executable Condition Analysis” considering control and data dependencies, statically schedules the coarse grain tasks to processors or generates dynamic task scheduling codes to assign the tasks to processors and generates OpenMP Fortran source code for a shared memory multiprocessor system machine. The thread parallel code using OpenMP generated by OSCAR compiler forks threads only once at the beginning of the program and joins only once at the end even though the program is processed in parallel based on hierarchical coarse grain task parallel processing concept.

The performance of the proposed scheme is evaluated on a 8-processor shared memory multiprocessor system machine, IBM RS6000 SP 604e High Node, using OpenMP Fortran source codes generated by compiling ARC2D in Perfect Benchmarks, SWIM, TOMCATV, HYDRO2D, MGRID in SPEC 95fp Benchmarks. In this evaluation, OSCAR compiler with XL Fortran compiler gave us significant speed-up compared with the sequential execution time by native XL Fortran compiler, such as 2.6 times for ARC2D, 9.0 times for SWIM, 4.5 times for TOMCATV, 8.1 times for HYDRO2D and 6.8 times for MGRID. The evaluation shows that the proposed coarse grain task parallel processing scheme gives us 1.5 to 3 times larger speedup than the automatic loop parallelization by IBM XL Fortran compiler.

1.1.2 Technique of analyzing interprocedural multi-grain parallelism

Automatic parallelizing compilers have to extract as much parallelism from programs as possible to make efficient use of shared-memory multiprocessors (SMPs). Loop parallelization technique cannot extract sufficient parallelism from the programs including such sections as those outside loops or sequential loops. So, multi-grain 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 multi-grain parallelism using as a base compiler the interprocedural automatic parallelizing compiler WPP (Whole Program Parallelizer), which was developed in part in RWC (Real World Computing) project in 1997-1999.

In this fiscal year we have been examining how to implement the technique of analyzing interprocedural dependences between tasks. This technique uses a hierarchical control-flow graph (CFS:Control Flow Summary graph) made of the whole program by WPP. Regarding each node of CFS as a task, the technique analyzes control and data dependences between nodes, and connects two nodes with dependence relationship using dependence edge. There are three types of tasks: basic blocks, loops, and procedure calls. Nodes corresponding to loops or calls to certain procedures are linked to the CFS graphs made of the bodies of the loops or the procedures, respectively. Some of the features of our techniques are as follows. To each array with a dependence relationship between certain two nodes, the following three data are registered on the dependence edge connecting these two nodes; the dependence types (flow, anti, and output), the array name, and the reference region of the array; all of which can be used to privatize the array. Array reference regions are represented using the linear inequality system, which improves the precision of dependence analysis. Using our technique, much task parallelism is expected to be extracted from program sections that have no loop parallelism.

1.1.3 Parallelization technique for an imperfectly nested loop nest

It is not always a best way to extract parallelism from each perfectly nested loop nest of a sequential program. We plan to extract parallelism from each imperfectly nested loop nest, which is a set of perfectly nested loop nests surrounded by at least one common loop. In our method, first every possible parallelism is extracted from each imperfectly nested loop nest. We finally choose the best parallelism for each imperfectly nested loop nest, considering the locality of a whole program.

To accomplish above, we have done the following works in this term:

（1）The basic design and prototype implementation of data dependence analysis routine of an imperfectly nested loop nest

This routine decides, by array subscripts within an imperfectly nested loop nest, whether it has data dependence or not. If it has the data dependence, this calculates what kind of dependence it has. This routine does not use an imprecise data dependence information such as a dependence vector, so we can do more efficient and aggressive parallelization.

（2） he basic design and prototype implementation of finding the possibility of the decomposition of computation and data

We find some possibilities of computation and data decomposition with the minimum communication in such a way that, for example computations using the same data are assigned to the same processor.

1.2 Predicated data-flow analysis technique

In this technological item, we research and develop both the predicated data-flow analysis technique and the run-time dependence analysis technique, each of which is the extension of the existing data-dependence analysis technique. In this fiscal year, we have been examining how to implement predicated data-flow analysis and have done the preliminary evaluation.

1.2.1 Predicated data-flow analysis technique

To speed up programs on parallel machines, it is important to exploit as much parallelism of the programs as possible. The WPP which has been developed in part in RWCP project, parallelizes loops by analyzing inter-procedural data dependences. There are some loops, which are not parallelized by WPP but can be parallelized principally. Such is the case where the data reference preventing parallelism executes only on a condition and the condition never holds when using an input data. In this study, we have been researching and developing predicated data-flow analysis in order to parallelize those loops.

In this fiscal year, we have been examining how to implement predicated data-flow analysis. Predicated data-flow analysis is implemented as follows. First, data-flow analysis phase analyzes for each statement an array reference region with a predicate, the condition the statement with the reference region executes. Second, 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. Last, code-generation phase generates multi-versioned code, in which at run time the condition is tested and selected is one of three loops; a serial loop and two parallel loops to which array privatization is applied and is not, respectively.

As a preliminary evaluation, we applied our technique by hand to such a loop in SPECfp95/apsi that WPP cannot parallelize. In three versions of loops in the resulting code, the parallel loop to which array privatization is applied is selected at run time. We ran the program on eight processors and found that the scalability of the loop is expected to achieve about six but that of the whole program 1.03.

1.2.2 Parallelism analysis techniques for Fortran90

Recently the number of scientific application programs written in Fortran90 has been increasing on multiprocessors. For instance, four of 14 programs of SPECfp2000, a famous benchmark suite in the field of scientific computations, are written in Fortran90. As this benchmark suite is planned to be used for the evaluation in this project, it is necessary for our automatic parallelizing compiler to support Fortran90. However, the interprocedural automatic parallelizing compiler WPP which has been developed in part in RWC project in 1997-1999, cannot extract parallelism from Fortran90 programs because it supports only FORTRAN77. In this study, aiming at extracting more parallelism from Fortran90 programs we have been developing parallelism analysis techniques for Fortran90, expanding interprocedural analysis techniques of WPP to each Fortran90 specification.

In this fiscal year we have been prototyping three techniques; the cloning technique for the procedure which contains recursive calls, module variables, and allocatable arrays; the loop parallelization technique for the programs which contains recursive calls, module variables, and optional arguments; and the technique for limiting parallelization to loops including no references to variables such as structures. These techniques are realized by detecting recursive calls in the call graph, analyzing data dependences for module variables, analyzing correspondence relationships between arguments and parameters of optional variables, analyzing descriptors of dynamic arrays, and so on. These techniques are expected to extract more parallelism from Fortran90 programs than before.

1.3 Automatic data distribution technique

Automatic data distribution technique is to distribute the data on the memory by the compiler for each parallel task to acquire the necessary data smoothly. There is a gap between the logical memory structure and the physical one on parallel computer systems. The different technique is needed according to the target memory model. We have studied the data decomposition technique for distributed shared memory system and distributed cache memory system, and the optimization techniques of data locality for distributed shared memory.

1.3.1 Automatic Data Distribution Technique for Distributed Shared-Memory Multiprocessors

In recent years DSMs(distributed shared-memory multiprocessors) have attracted attention of users because of their performance scalability due to physically-distributed memories and their ease of parallel programming due to logically-shared memories. Although usual memory-reference instructions can access physical memories of remote processors as well as those of local processors on DSMs, references to remote memories require more time than those to local memories. For this reason, data distribution assigning each data onto appropriate processors is important in order to obtain good performance on DSMs. In this study, aiming at determining the most appropriate data distribution by our compiler, we have been researching and developing automatic data distribution technique for DSMs.

In this fiscal year, we have been proposing and evaluating automatic data distribution technique for indirect referenced arrays and have been prototyping data-distribution –control libraries for software DSMs (SDSMs). Our automatic data distribution technique, the first-touch control data distribution method (FTC), distributes complex data accurately using first-touch page allocation policy of the operating system. In case of the programs including indirect referenced arrays, temporary arrays are used before some values are stored into index arrays of the arrays and the indirect referenced arrays are distributed by FTC immediately after that. Moreover, in order to extend this technique to the other platforms, SDSMs, we have been prototyping the library that explicitly controls data distribution on SDSMs.

As preliminary evaluation, we compared two versions of NPB2.3serial/CG program. One is made by applying our technique by hand to indirect referenced arrays in the program. The other is made by applying regular data distribution to the arrays by using data distribution directives. The former version is expected to run 1.5 times faster than the latter version on SGI™ Origin™ 2000 (16 processors).

1.3.2 Data decomposition technique for distributed cache memory

We need not only to extract sufficient parallelism from the programs, but also improve the locality of the parallelized program for achieving high performance on parallel machines such as SMP. It is necessary for reducing true/false sharing by the decomposition of the computation and data, and/or the data layout arrangement.

To accomplish above, we have done the following work in this term to improve the locality between imperfectly nested loop nests:

（1） he basic design and prototype implementation of data dependence analysis routine among the imperfectly nested loop nests

This routine analyzes the data dependencies among the several imperfectly nested loop nests.

（2） he basic design and prototype implementation of locality improvement routine to reduce true sharing

・to reduce true sharing

Parallelization technique for an imperfectly nested loop nest enables to improve the locality within one imperfectly nested loop nest. We try to improve the locality among several imperfectly nested loop nests. This chooses computation and data decomposition for each imperfectly nested loop nest to improve the locality of a whole program and to reduce true sharing.

・to reduce false sharing

data fragment in such a way that the contiguous array elements are assigned to the same processor.

1.3.3 Optimization Techniques of Data Locality for Distributed Shared Memory

Current parallelizing compilers parallelize many types of Do-loops by using strong data dependence analysis and loop restructuring techniques. However, parallelism outside Do-loops, for example, coarse-grain parallelism among loops, subroutines and basic blocks, has not been effectively exploited. Therefore, in order to improve the effective performance of multiprocessor systems, it is important to hierarchically exploit coarse-grain parallelism in addition to loop parallelism. In the hierarchical coarse grain parallel processing, to reduce data transfer overhead, it is necessary that a compiler automatically decomposes data and computation and assigns them to each processor and its local memory so that data transfer overhead can be minimum.

This annual report proposes a data-localization scheme for macrotask-graph with data dependence edges in hierarchical coarse-grain parallel processing. In this scheme, first, multiple coarse-grain tasks having data dependencies are decomposed by Loop-aligned-decomposition method in each macrotask-graph layer. Especially, the proposed Loop-aligned-decomposition method can be applied to large regions each of which is composed of multiple loops. Next, the compiler assigns macrotasks with strong data dependence to the same processor by Data-transfer-gain/CP scheduling method, so that data transfer overhead is reduced by using caches or local memories. Finally, this report describes the performance evaluation on a multi-processor system SGI Origin2000. The evaluation shows that hierarchical coarse-grain parallel processing with data-localization can reduce execution time remarkably. The result of this research was published in SIG Notes of IPSJ.

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 2000 fiscal year, we began to develop the algorithm of speculative execution for the medium grain tasks such as loop iteration level, and examined the support mechanism to apply speculative execution effectively.

1.4.1 Speculation Techniques for Medium Grain Tasks for Multigrain Parallelization

To do the speculative execution for medium grain tasks, the following four features are indispensable: (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 algorithm to decide the initiation time of tasks, and (4) the discarding mechanism of tasks that became needless. The research and development of both (1) and (2) was done in 2000 fiscal year.

We focused on the loop that includes the loop carried dependence with unknown data dependence distance.

Firstly, the loop is divided into “loop control part” and “loop body part”. Data speculation technique is applied to the loop body part at the same time as speculatively executing the loop control part. As for the loop where the loop carried dependence does not exist, speculative execution easily becomes possible by applying control speculation technique to “loop control part”. Moreover, the loop by which the data dependence distance is fixed can be handled as Doacross type loop even if we do not apply speculation.

Secondly, we examined the effect of speculation extent by applying data speculation technique to the variables with the loop carried dependence. Although the prediction techniques such as a stride value and the constant value are proposed, efficient prediction technique which uses a minimum resource is indispensable to efficiency and speculative execute a program with the limited hardware[1]. Then, the loop is considered to be a Doacross type loop by assuming that control speculation technique is applied to the side where the loop continues.

Concretely, the loop carried dependence which becomes the decision factor of delay d (the initiation interval of each loop iteration) is distinguished in order to shorten the total execution time of the loop which is treated as Doacross type loop after applying the control speculation technique to its control part. Then, the variables that should be applied data speculation technique are judged their priority.

By applying the above-mentioned techniques to the SPEC Benchmark test, we have confirmed that we are able to speedup about twice in comparison with the normal execution without speculation when assuming that there is no overhead of speculative execution.

1.4.2 Parallelizing Compiler for Chip-Multi-Processors with Execution Profiling and Speculative Execution Support

In our Advance Parallelizing Compiler project, a platform-free parallelizing compiler targeted for existing SMP-type commercial parallel machines is being developed. Parallel computer architectures themselves are improved rapidly, however. Therefore, it is also important to understand new parallelizing compilation techniques targeted for future parallel machines.

In the project at Electrotechnical Laboratory, we develop an extended OpenMP compiler targeted for our Chip-Multi-Processors with Execution Profiling and Speculative Execution Support to enable the utilization of the new features. Our goals of this development are(1)to get better understanding of new parallelizing compilation and parallel tuning techniques,(2)to get better understanding of the performance and applicability potential of the new techniques, and 3) to contribute to the whole project with these technical feedbacks.

The results of the project of this year include (1) the development of a new program model

[1] Intended for the processor with the prediction hardware. The trace information is used as the input data to the compiler intended for the processor without the prediction hardware. In this case, acquiring the trace for all variables is not practical, but the trace information of only the variable that a compiler judges as important should be acquired.

suitable both for execution profiling and speculative execution, (2) the initial investigation of extended fetures of OpenMP language for our new CMP, and (3) facilitating the compiler development environment for our platform.

Koike, H.：Investigating a Chip-Multi-Processors with Execution Profiling and Speculative Execution Support, IPSJ SIG Notes, 2001-ARC-141, pp.47-52, Jan. 2001 (in Japanese).

1.5 Scheduling Techniques

To efficiently execute programs in parallel on a multiprocessor system, a minimum-execution-time multiprocessor scheduling problem must be solved which determines the assignment of task sets to the processors and the execution order of the tasks so that the execution time is minimized. It is known that the time complexity of this problem is strong NP-hard for a general problem which assumes arbitrary task processing time, arbitrary number of processors, arbitrary shapes of task graphs, and arbitrary inter-processor data communication time.

Because of the intractability of the scheduling problem, many heuristic algorithms have been proposed. An objective understanding of the evaluations of the scheduling algorithms is difficult because these algorithms have not evaluated under the same conditions. It is necessary for fair performance evaluation to use a lot of task graphs with various shapes and arbitrary communication time available for every researcher.

This section proposes a standard task graph set with data communication time “STG_dt ver.0.1” for fair evaluation of the performance of scheduling algorithms considering task graph generation methods used in previous researches and describes the evaluation of heuristic algorithms, which consider data transfer overheads hiding by use of overlap with task processing, such as CP/DT-w-PS (with poststore), CP/DT/MISF-w-PS, DT/CP-w-PS, DT/CP/MISF-w-PS, CP/ETF-w-PS, and ETF/CP-w-PS using the standard task graph set.

As the results of evaluation, it was confirmed that CP/ETF-w-PS algorithm, which gives the highest priority to the tasks having longest critical path (CP) length, allows us to have the best performance for the problems having small CCR (the ratio of task processing cost and data communication cost). Also it was confirmed that ETF/CP-w-PS algorithm, which gives the highest priority for a pair of task and processor requiring the least data communication cost, achieves the best performance for the problems having large CCR.

1.6 Common interface language amang some optimization unctions

The Advanced Parallel compiler consists of several parallelization techniques. We select the OpenMP API(Application Program Interface) as a interface among these parallelization functions.

1.6.1 Making Extensions of a parallel programming language

The OpenMP API aims at shared memory parallel multiprocessor machines, therefore the OpenMP API is not used on DSM (distributed shared memory) parallel multiprocessor machines, to which APC project applies its automatic parallelization function.

In this research, we designed a prototype of a set of extensions of the OpenMP API for DSM parallel multiprocessor machines using the basic design which had been proposed by us in the RWCP (Real World Computing Partnership). In this prototype, we added specifications of processor group, data distribution, and explicit communication to the original OpenMP API to apply to DSM parallel multiprocessor machines.

We developed a prototype compiler for these extensions, and examined performance of these extensions on a DSM parallel multiprocessor machine. The performance results show that these extensions are effective for the DSM parallel multiprocessor machine.

We reported this result at the SHINING 2001 of the information processing society of Japan.

2. Development of Tuning Technology for Parallel Processing

In the tuning technology for parallel processing, we research and develop three techniques: the program visualization technique, the dynamic information utilization technique, and the combinatorial parallelization technique. In this fiscal year, we started the research and development of the program visualization technique and we have been examining how to implement the technique and prototyping its user-interface.

2.1 Program Visualization Technique

In this technological item, we research and develop the program visualization technique that provides users with an easy way to view the parallel structure of a given program and to extract useful information for tuning from the program. In this fiscal year, we have been examining how to implement interprocedural data-dependence-locating technique, which finds all the statements having data dependence relationship in procedures called within a loop, and have been prototyping its user-interface. The data-dependence-locating technique is a base one for realizing the program slicing technique that extracts from a program the set of statements concerning the determination of values of the variables preventing parallelization.

2.1.1 Program Slicing Technique

To get high performance on multiprocessors any program must be parallelized. Compiler’s automatic parallelization provides good but limited performance; parallelization tuning using user’s knowledge is indispensable for getting the maximal performance of the program. To inspect the parallelism of a program efficiently 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 that have data dependence relationship prohibiting parallelization. Showing those statements helps users to find causes of prohibiting parallelization. These tools, however, can not show any statement having data dependence relationship in a procedure called within a 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 examining how to implement interprocedural data-dependence-locating technique, which finds all the statements having data dependence relationship in procedures called within a loop and have been prototyping its user-interface. Interprocedural data-dependence-locating technique executes on-demand. First, it finds data dependence relationship between a statement S and a call to a procedure P. Second, when users select the call, it compares the array reference regions of statement S and of all the statements in procedure P. Last, it finds all of those statements in procedure P that have data dependence relationship with statement S. We have been prototyping a user-interface for the technique. One window lists a call sequence starting from a loop in question to the statement having data dependence relationship and the data dependence information. Another window displays two source programs, each of which includes either of the pair of statements having data dependence relationship. Using this technique and its user-interface we can expect the parallelization tuning to be done efficiently for loops including procedure calls.

II Research regarding performance evaluation of parallelizing compilers

The goal of this research is to establish technology for more objective performance evaluation of a parallelizing compiler for SMP systems, through the evaluation of Research Theme I: "Development of Advanced Parallelizing Compiler Technology", using the technology that is developed. For this end, developments regarding (1) the selection of benchmark programs, (2) the definition of run-rules, and (3) the definition of the performance metric are now under way. To obtain the base data necessary for selecting benchmark programs, surveys of parallelizing compilers and the existing benchmark programs have been made this year.

1. Evaluation method for individual functions

In order to obtain the base data for the kernel programs which are to be used for the evaluation of individual functions of a parallelizing compiler, surveys of the kernel programs from the two benchmarks and of the commercial parallelizing compiler have been performed.

1.1 Surveys of benchmark programs

The two benchmarks described below, including the kernels to be used for the evaluation of individual functions, have been selected as candidates, and some surveys of the programs, the run-rules, and the performance metrics regarding that benchmarks have been performed.

1.1.1 SPEC CPU2000

In 1999, the SPEC CPU2000 benchmark was announced by SPEC/OSG (The Standard Performance Evaluation Corporation/ Open Systems Group) as the follower of SPEC CPU95. Since SPEC CPU95 has been widely used as the benchmark for commercial computers, SPEC CPU2000 is considered acceptable for use as the benchmark for the evaluation of the individual functions of paralellelizing compilers.

The benchmark programs in SPEC CPU2000 consist of CINT2000, which is designed for the performance evaluation of the integer operations, and CFP2000, which is designed for the performance evaluation of the floating-point operations. CINT2000 includes twelve application programs, eleven of which are written in C and one of which is written in C++. The number of lines of each program ranges from thousands to tens of thousands. CFP2000 includes fourteen application programs, six of which are written in Fortran77, four of which are written in Fortran90, and four of which are written in C. The number of lines of each program ranges from hundreds to tens of thousands. This benchmark has many types of programs and is full of kernels that can be used for the evaluation of individual functions.

In SPEC CPU2000, the run-rules on the setting optimization-option flags are defined and two types of the performance metrics are used based on the flag setting. Such principles regarding the run-rules and the metrics should be used in the evaluation of individual functions.

In this survey, the trial parallelization of the SPEC CPU 2000 programs was conducted using the real parallelizing compiler (as described in 1.2).

1.1.2 NAS Parallel Benchmarks (NPB)

NPB, which were announced in 1991 by the NASA Ames Research Center NAS(Numerical Aerospace Simulation) program, provide the benchmarks for parallel computers.

The programs in NPB are designed to simulate the CFD(Computational Fluid Dynamics). NPB consist of five parallel kernels for the core calculation of CFD program and three pseudo-applications emulating the relationship between the data transfers and the calculations in the CFD programs. Originally NPB were definitions of the application used for the evaluation, now they include the parallel codes and the serial codes. The number of lines in each serial kernel code ranges from hundreds to one thousand and the number of lines in each serial pseudo-application is about three thousands.

One of the main characteristics of NPB is that it focuses on one application program and evaluates the performance of each part of the program in detail.

1.2 Surveys on functions of high-end compilers

To select kernels for evaluation of individual functions, some surveys of functions of Visual KAP are performed using benchmarks described in 1.1.

1.2.1 Survey on functions of Visual KAP using SPEC CPU2000

As an example of a high-end parallelizing compiler, Visual KAP for OpenMP compiler, which is developed by Kuck & Associates, Inc (KAI), was selected, and the survey of Visual KAP's parallelizing functions has been made by using it to compile SPEC CFP2000 programs.

KAI has been found by Professor David Kuck at Illinois Univ., where much of the leading research on parallelizing compilers has been done, and is one of the oldest independent software vendors whose specialty is the development of parallelizing compilers.

Visual KAP is a parallelizing compiler developed by KAI. The compiler inputs sequential Fortran77/90 programs and applies scalar optimization and parallelization. Visual KAP can parallelize serial Fortran programs automatically and outputs optimized and parallelized programs written in the OpenMP directives. The newest version of the Visual KAP for OpenMP is version 3.9, and runs on the Windows 2000/NT 4.0.

In this survey, the paralellelizing functions of Visual KAP have been investigated by parallelizing the programs in “wupwise”, “sixtrack”, “facerec”, and “lucas” selected from SPEC FP2000.

(1) Coarse grain level parallelization

The coarse grain parallelisms in a program should be used effectively for auto multi-grain parallel processing. The coarse grain parallelisms in a program can be described with the “sections” directive of OpenMP and can be parallel-processed. In the present investigation, based on the results of compiling the example programs with coarse grain parallelisms using Visual KAP, it was revealed that Visual KAP cannot parallelize the programs at the coarse grain level.

(2) Loop parallelization

Loop parallelization is one of the parallelization techniques on which substantial research has been performed. In this survey, in order to investigate the loop parallelization functions of Visual KAP, all programs in “wupwize” and “sixtrack” (both written in Fortran 77) and from “facerec” and “lucas” (both written in Frotran90) have been compiled using Visual KAP. As a summary of the results, the numbers of loops which have been recognized as parallel loops and those parallelized using the “C$OMP DO” directive are 19 out of a total of 70 loops in “wupwize”, 438 loops out of a total of 1498 loops in “sixtrack”, 84 loops out of a total of 189 loops in “facerec”, and10 loops out of a total of 104 loops in “lucas”.

Moreover, for some typical loops and several characteristic loops from the programs in SPEC FP2000, several detailed investigations, such as those concerning types of parallelization and the cause of the inhibition of parallelization, have been performed.

Based on results obtained by observing the functions of visual KAP, the candidate kernels for evaluating the functions of the up-to-date compilers have been obtained.

1.2.2 Survey on functions of Visual KAP using NPB

In the same way as 1.2.1, we investigated the performance of loop parallelization techniques in the Visual KAP by NAS Parallel Benchmarks. We have omitted the investigation for coarse grain parallelization, since the investigation in 1.2.1 shows the Visual KAP does not perform the coarse grain parallelization.

We have tried to parallelize seven codes that are included in the serial version of NAS Parallel Benchmarks 2.3 (NPB2.3-serial). The investigation for the code, IS, were not performed because the code was written in C.

The each kernel code, CG, EP, FT or MG, is composed of less than 100 loops. (The number of loops includes inner loops in a nested loop.) The CG has 41 loops and the Visual KAP parallelized 18 loops among them. The EP has eight loops and one loop among them is parallelized. The FT is composed of 50 loops and the Visual KAP parallelized 12 loops among them. Also, the Visual KAP parallelized 22 loops out of a total of 93 loops in the MG. The application codes, BT, LU and SP are composed of 100 through 300 loops respectively. The Visual KAP parallelized 24 loops out of a total of 227 loops in the BT, 40 loops out of a total of 172 loops in the LU, and 47 loops out of a total of 314 loops in the SP, respectively.

Next, we investigated the parallelization techniques applied to the parallelized loops and reasons that inhibited the parallelization. Finally, we conclude that the codes in NPG2.3-serial can be a candidate for the benchmark in the evaluation of individual technique.

2. Overall performance evaluation method

In order to obtain the base data for the benchmark programs that are to be used for evaluation of the overall performance of a parallelizing compiler, surveys were made of two benchmarks and of several commercial parallelizing compilers.

2.1 Surveys of benchmark programs

Large and real application programs are required for the evaluation of the overall performance. Two benchmarks described below were selected as the candidates for this evaluation, and surveys of the programs, run-rules, and performance metrics regarding those benchmarks have been made.

2.1.1 Perfect Benchmarks

Perfect Benchmarks (PERFormance Evaluation for Cost-effective Transformations) are benchmarks of scientific calculation in the high performance computing field. It has been developed by the CSRD (Center for Supercomputing Research and Development) of Illinois Univ. in cooperation with IBM Kingston Lab., NASA Ames Lab, Princeton Univ., Cray Research, and California Technology Univ., and were announced in 1993.

Perfect Benchmarks consist of 13 real application programs which are written in Fortran77 and the number of lines in each program ranges from hundreds to tens of thousands, so this benchmark seems to be appropriate for the evaluation of the overall performance.

The run-rules involve two conditions based on the hand optimization level; one allows code-optimization by hand and the other prohibits hand code optimization. When the hand optimizations are conducted, it is required that the optimization history which includes information regarding the types of optimization used, the performance gain with those optimizations, and the handling costs for those optimizations be reported. This kind of run-rule should be used for the evaluation of the overall performance.

2.1.2 SPEChpc96

The SPEChpc96 benchmark was announced by SPEC/HPG (Standard Performance Evaluation Corporation / High Performance Group) in 1995.

The membership of SPEC/HPG consists of researchers involved with other benchmark projects such as the Perfect Benchmarks project and of researchers from other companies and laboratories. SPEChpc96 is designed to evaluate the performance of high-end parallel/distributed computers, and can measure real performance, instead of peak performance, using real industrial programs.

SPEChpc96 consists of three large application programs written in Fortran and C. The number of lines of each program ranges from tens of thousands to hundreds of thousands. One of the advantages of this benchmark is that the programs are written in both serial and parallel manners. Thus, it is possible to evaluate the performance of a parallelizing compiler by comparing the parallel processing times of the parallelizing code generated by the compiler and the times of the hand-parallelized code.

2.2 Survey of high-performance parallelizing compilers

The diffusion of SMP multiprocessor systems accelerated development of high-performance parallelizing compilers. These compilers are classified into two types; compilers that generate a parallelized machine code and compilers that generate a source program with compiler directives for parallel execution. The compiler directives that inserted into a source program by the latter compiler have had various formats, each of which has been specified by an individual compiler developer. However, compiler researchers/vendors currently try to make the standard format for the directives, OpenMP, and develop compilers that generate OpenMP programs, or programs with OpenMP directives. We investigated the high-performance parallelizing compilers that generate OpenMP programs, since an OpenMP program is used as an intermediate representation among compiler modules developed in our project.

We have found two commercial parallelizing compilers that generate OpenMP programs; the Visual KAP for OpenMP developed by Kuck & Associates, Inc., US and the VAST/toOpenMP developed by Pacific-Sierra Research Corporation, US. Also, we have found state-of-the-art parallelizing compilers developed in research projects, the Polaris developed by the University of Illinois and the Purdue University, the CAPTools developed by the University of Greenwich, and the OSCAR Compiler developed by the Waseda University.

We have installed the Visual KAP for OpenMP, which is a representative commercial parallelizing compiler, and the Polaris, which is a representative state-of-the-art parallelizing compiler.

2.2.1 Visual KAP for OpenMP

The Visual KAP for OpenMP has five parallelization modes, each of which performs different strength of parallelization, and the user is able to specify the mode through GUI. Parallelization techniques applied by each mode are as follows (techniques applied in the highly parallelized mode include those applied in the lower mode): (1) the Low level optimization, which performs the recognition of induction variables and the loop interchange of simple loops, (2) the Medium level optimization, which performs lifetime analysis of scalar variables, (3) the High level optimization, which interchanges triangular loops and analyzes high level data dependence test for loop index variables, (4) the Higher level optimization, which performs the loop interchange of reduction loops and floating conditional branch statements to out of loops, (5) the Highest level optimization, which performs the array expansion. Also, the users are able to individually specify the details of other optimization/parallelization techniques, such as scalar optimization, optimization for the round-off error, the loop fusion, the loop unrolling, the inter-procedure analysis, and cache optimization.

We can say that the performance of optimization/parallelization techniques implemented in the Visual KAP for OpenMP represents the performance of high-level commercial parallelizing compilers currently developed. Thus, we conclude that the performance of this compiler can be a performance metric in the synthesis evaluation of parallelizing compiler in our project.

2.2.2 Polaris

The Polaris is the state-of-the-arts parallelizing compiler developed in the University of Illinois and the Purdue University. The Polaris inputs a sequential Fortran source program and generates a parallelized program with compiler directives for parallel processing on SMP. Currently, there are two versions of compilers, the Polaris Release 1.6.5 distributed by the University of Illinois and the Purdue version developed by the Purdue University, and the Purdue version generates OpenMP programs. The Purdue version provides user interface on the web page, and the user can operates the compilation of source programs though web browser.

The Polaris performs not only conventional loop transformation techniques, such as the loop interchange, the loop fusion and the loop coalescing, but also the high-level analysis and transformation techniques, such as the recognition of induction variables/substitution, the recognition of reduction loops, array expansion/privatization, and the non-linear and symbolic data dependence testing.

The performance of the Polaris is verified in many literatures and it is recognized as the parallelizing compiler that has highest performance in the world. Thus, we conclude that the performance of this compiler can be a performance metric in the synthesis evaluation of parallelizing compiler in our project.

Waseda University

2000 Papers

- Kazuhisa Ishizaka, Hironori Kasahara, Motoki Obata, ``Coarse-grain Task Parallel Processing using the OpenMP backend of the OSCAR Multigrain Parallelizing Compiler'', Proc. of Third International Symposium, ISHPC 2000, October 2000.
- Keiji Kimura, Hironori Kasahara, ``Evaluation of Single Chip Multiprocessor Core Architecture with Near Fine Grain Parallel Processing'' Proc. of International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems (IWIA'01), Jan., 2001.
- Hiroshi Koide, Hironori Kasahara, ``Meta-scheduling -- Trial for Automatic Distributed Computing'', bit, pp. 10-14, Kyoritsu Shuppan, Vol. 33, No. 4, Mar., 2001.
- Hiroshi Koide, Nobuhiro Yamagishi, Hiroshi Takemiya, Hironori Kasahara, ``Evaluation of the resource information prediction in the resource information server'', Trans. of IPSJ: Programming Vol.42,Mar. 2001.
- Keiji Kimura, Takayuki Kato, Hironori Kasahara, ``Evaluation of Processor Core Architecture for Single Chip Multiprocessor with Near Fine Grain Parallel Processing'', Trans. of IPSJ, Vol. 42, No. 4, Apr., 2001.(To be appeared)
- Hironori Kasahara, Motoki Obata, Kazuhisa Ishizaka, ``Coarse Grain Task Parallel Processing on a Shared Memory Multiprocessor System'', Trans. of IPSJ, Vol. 42, No. 4, Apr., 2001.(To be appeared)

Technical Reports

- Kazuhisa Ishizaka, Satoshi Yagi, Motoki Obata, Akimasa Yoshida, Hironori Kasahara, “ Evaluation of coarse grain task parallel processing on the shared memory multiprocessor system ” , Technical Report of IPSJ, ARC-141-7, Jan. 2001
- Akimasa Yoshida, Satoshi Yagi, Hironori Kasahara, “ A Data-Localization Scheme for Macrotask-Graph with Data Dependencies on SMP ” , Technical Report of IPSJ, ARC-141-6, Jan., 2001

Annual Convention

- T. Yamaguchi, Y. Tanaka, T. Tobita, H. Kasahara, ``Performance Evaluation of Scheduling Algorithms with Data Transfer Using Standard Task Graph Set'', Proc. 62th Annual Convention IPSJ 2Q-01, Mar., 2001.
- T. Hayashi, Y. Moda, H. Koike, T. Tobita, H. Kasahara, ``Meta-Scheduling Method for Heterogeneous Distributed Environment Using OSCAR Fortran Multi-Grain Parallelizing Compiler'', Proc. 62th Annual Convention IPSJ 3R-01, Mar., 2001.

- Y. Moda, T. Hayashi, H. Koike, T. Shikashima, H. Tsutsui, H. Kasahara, ``CPU Load Prediction on Hererogeneous Distributed Computing Environments Using dataFOREST Data-Mining Tool", Proc. 62th Annual Convention IPSJ 3R-02, Mar., 2001.
- H. Nakano, K. Ishizaka, M. Obata, K. Kimura, H. Kasahara, ``A Static Scheduling Method for Coarse Grain Tasks considering Cache Optimization on Multiprocessor Systems", Proc. 62th Annual Convention IPSJ 4R-02, Mar., 2001.
- T. Tanaka, H. Funayama, T. Tobita, H. Kasahara, ``Performance Evaluation of Preload-Poststore Scheduling Algorithm Considering Memory Capacity", Proc. 62th Annual Convention IPSJ 4R-03, Mar., 2001.
- T. Kodaka, K. Kimura, N. Miyashita, H. Kasahara, ``Near Fine Grain Parallel Processing on Multimedia Application for Single Chip Multiprocessor", Proc. 62th Annual Convention IPSJ 3P-08, Mar., 2001.
- N. Matsumoto, K. Kimura, H. Kasahara, ``Performance Evaluation of Single Chip Multiprocessor Memory Architecture for Near Fine Grain Parallel Processing", Proc. 62th Annual Convention IPSJ 4P-01, Mar., 2001.
- N. Miyashita, K. Kimura, T. Kodaka, H. Kasahara, ``A Data Transfer Unit on the Single Chip Multiprocessor for Multigrain Prallel Processing", Proc. 62th Annual Convention IPSJ 4P-02, Mar., 2001.

Invited Talks

- ``OSCAR Multigrain Parallelizing Compiler and Single Chip Multiprocessor", University of Illinois at Urbana-Champaign, Nov.3, 2000
- ``OSCAR Multigrain Parallelizing Compiler and Single Chip Multiprocessor", Data Processing Center, Kyoto University, Jan.30, 2001.
- ``Overview of METI/NEDO Millenium Project ``Advenced Parallelizing Compiler"", Japan Information Processing Development Center Research Institute for Advanced Information Technology, Mar.2, 2001.

Toho University

- Akimasa Yoshida, Satoshi Yagi, Hironori Kasahara, A Data-Localization Scheme for Macrotask-Graph with Data Dependencies on SMP, Technical Report of IPSJ, ARC-141-6, Jan., 2001.