23

Intelligent Parallelization within the Vienna Fortran Compilation System [*]

Jan Hulman, Stefan Andel, Barbara M. Chapman, and Hans P. Zima

Institute for Software Technology and Parallel Systems
University of Vienna, Brünner Strasse 72,
A-1210 VIENNA, AUSTRIA
E-Mail: hulman@par.univie.ac.at

Abstract

Current research and development in the area of software support for programming of distributed memory systems aims to produce compilers for languages with a single thread of control and extensions to specify the mapping of data to the processors. Compilers for these languages, in particular, High Performance Fortran (HPF), require much advanced analysis and must rely heavily on the heuristic application of program transformations; it is expected that early implementations will not handle many kinds of programs well.

In this paper we introduce an alternative approach to support programming of such machines, which makes use of expert system technology. We argue that a knowledge-based approach to this task will contribute to more powerful and intelligent automatic parallelization systems in the future. We outline the salient features of a new knowledge-based parallelization environment being built around VFCS. The basic design decisions for this environment are discussed and major components of its first prototype are described. The tool is designed not only to support the compilation of HPF and similar languages, but also to perform fully automatic compilation of sequential languages without extensions.

[*]The work described in this paper was carried out as part of the research project "Performance Prediction and eXPert Adviser for Parallel Programming environment", funded by the FWF, Vienna, Austria, under the grant number P9205-PHY. The authors assume all responsibility for the contents.

1 Introduction

Modern distributed-memory multiprocessor systems (DMMPs), such as those offered by Intel, Meiko, Thinking Machines and NCUBE, are able to provide their users with a sustained high performance at comparatively reasonable cost. Recent systems are scalable to large numbers of processors; some systems, (e.g. Meiko's CS-2) offer both scalar and vector processing nodes.

It is, however, very difficult for a programmer to utilize the power of these machines. Established programming paradigms based on message-passing require the user to distribute both data and work explicitly among processors. This task is often likened to assembly language programming on sequential computers. The programming difficulties are further increased when the nodes of a system are no longer homogeneous. As a result, application development on DMMPs is a laborious, time-consuming and error prone process.

A long-term goal of language and compiler research is to make program development on DMMPs as productive as programming on conventional sequential architectures. One approach to this task is to provide appropriate extensions to existing programming languages, so that the user can write programs with a single thread of control and global data references only. It is then the task of system software to convert the user program into a form suitable for execution on a DMMP; we refer to this process as *parallelization*. This programming paradigm promises not only faster and easier program development, but also greater flexibility and portability of codes.

In this paper we describe the design of an environment for parallelization which uses expert system technology in an attempt to improve the state of the art. Much of the discussion focusses on a specific component of this, a new tool currently being implemented at the University of Vienna. This tool, the eXPert Adviser (hereafter referred to as *XPA*), is being built using expert system methodology. XPA cooperates with the other tools in this environment in an attempt to both make the parallelization process more automatic and convenient, as well as to improve the quality of the code generated.

This paper is organized as follows. In Section 2 we give a short overview of conventional approaches to the task of parallelization. We discuss the limitations and difficulties of this approach, and in Section 3, we argue for the use of expert system technology to support the restructuring of code for DMMPs. We then discuss the parallelization environment to which XPA belongs, and with whose components it cooperates to parallelize Fortran codes (Section 4).

The rest of the paper is devoted to the design of XPA, its solution strategy, and the methods by which it interacts with the other tools in the environment. This includes an outline of the approach it takes to derive data distributions for the arrays in the source program. We conclude with a discussion of related work and a brief summary of our approach.

2 Traditional Approach to Parallelization

Early parallelization systems for Fortran programs, such as Superb [25] and the MIMDizer [20], were complex interactive systems which supported the user in the task of transforming programs. The user was required to provide a specification of the manner in which the program's data was to be distributed to the target machine.

The experience gained with these systems was formalized in several proposals for Fortran language extensions [14, 26], which permitted the user to describe the desired data distribution. Cooperation of major computer vendors and academic institutions has resulted in an agreement on a basic set of such language extensions, High Performance Fortran (HPF) [15]. Current research and development at a number of laboratories and compiler companies is aimed at implementing the features of this language. In contrast to the early systems, which relied on a significant user involvement, most of these efforts will rely entirely upon traditional compiling techniques and advanced analysis and transformation capabilities.

However, many existing programs will have to be rewritten to fulfil restrictions imposed by HPF, or to adapt the code to use constructs that can be reasonably handled by the parallelizers which will soon be on the market.

Although in the long term there will be better hardware support, and both programming methodology will adapt somewhat to the capabilities of parallelizers, and parallelizers themselves will become considerably more sophisticated, much work remains to be done before such tools will be able to adapt a broad range of codes automatically for efficient execution on a parallel machine. Both this goal, and that of fully automatic parallelization (without user specification of a data distribution), require a deep understanding of coding practices and techniques, the usefulness of program transformations, how they may interact, and how a global strategy for parallelization may be realized in individual program parts. Individual parts of the parallelization process may have conflicting aims, and tradeoffs may be required. It is currently not well understood how to apply some of the possible transformations. A further problem is that many of the decisions which need to be require information which cannot be derived analytically from the program code alone. This implies either the continued involvement of the user in the process, or an environment in which there are other tools able to derive missing data.

It is not a reasonable solution for the user to control the actions of a transformation system, however, because of the **knowledge gap** involved: an applications programmer is not likely to be familiar with the individual transformations, let alone know how to use them to obtain parallel code.

This has led us to consider alternative approaches to the problem.

3 The Usefulness of an Expert System Approach

The task of parallelization, in particular fully automatic parallelization, is a complex one in which there are a number of subgoals, which may conflict, and between which there are strong interrelations. In such situations the expert system methodology, based upon an explicit representation and exploitation of the knowledge related to a particular problem domain, has proved to be suitable and fruitful in a number of problem areas (particularly CAD/CAM) [17]. We believe that a knowledge-based approach will create a framework for the solution of each of the problems associated with automatic parallelization.

By applying it to this problem domain, we hope not only to develop a *rapid prototype* of a system which is able to perform fully automatic parallelization, but also to gain a better understanding of this task and good strategies for its solution. Solution strategies that are well understood can be utilized by future compilers which employ more traditional methods. It can make a significant contribution to recognizing major influences on the selection of strategies for application for target architectures. Further, this approach can help to create an environment in which it is possible to evaluate the effect of data distributions and transformations once they have been applied.

A substantial amount of knowledge was required to build the current generation of parallelizing systems, and it has been incorporated into them. Unfortunately, the parallelizers do not explicitly store information about the target machine's hardware and software, or about programming methodology. Instead, this information is hard-wired into the program, scattered into many thousands of lines of code, where it cannot be easily accessed, modified or extended; nor can be used to reason about the transformation process. Such a system is hard to modify - the implicit nature of the knowledge makes it inflexible. For the same reason, a system dedicated to one machine cannot be easily retargeted.

In view of the rapid evolution of the target machines, a process in which the very machine characteristics which have the largest influence on parallelization strategies are changing, it seems desirable to make both this knowledge and the manner in which parameters of the machine play a role explicit. The *explicit representation of knowledge* can have a significant impact on the system's portability, and supports the goals of easy modification and rapid prototyping.

Heuristics and methods of parallelizing specific constructs are known; they can serve as important shortcuts and hints. Efficient parallelization strategies for particular program patterns are also known from experience and experiments, and these can be incorporated into an **intelligent parallelization system**.

4 The Parallelization Environment

In this section we describe the parallelization environment by outlining the capabilities of the existing components. This environment is currently being extended by a prototype implementation of the XPA.

The tool around which the parallelization environment is centered is the **Vienna Fortran Compilation System (VFCS)** [8], a *source-to-source translator* from Vienna Fortran and HPF into several dialects of **Message Passing Fortran**, extensions of Fortran by a set of constructs for explicit message passing. The current system can target several existing DMMPs, as well as providing a machine-independent target in the form of the Parmacs macros. VFCS can be used as a command line compiler for Vienna Fortran and HPF programs, in which case it applies a standard set of program transformations and optimizations to the code, or it can be used as an interactive parallelization system where the user has full control of the parallelization process. It generates code according to the **SPMD** (Single Program Multiple Data) model of computation, in which the arrays of the original program are partitioned and mapped to the target processors. Essentially the same parametrized program is then executed on all processors, whereby non-local data is accessed by compiler-inserted communication statements.

VFCS derives a good deal of information on the source code it processes, including internal data structures such as the syntax tree, flow graph, call graph, and symbol table, and a precise dependence information resulting from data flow and data dependence analysis.

However, many of the decisions which have to be made during parallelization require knowledge of the program's behavior at run time which cannot be derived from the program by analytical methods alone. To support the derivation of such information a set of performance tools is incorporated in the parallelization environment. These performance tools are able to cooperate with VFCS, and may be invoked from within it. As we shall see, they will also cooperate with XPA, for which they represent a further source of knowledge.

The performance tools of the parallelization environment are the following.

The **Weight Finder** [11] is a profiling tool which derives profiling information by a combination of static performance analysis and dynamic techniques. It executes a sequential version of the input program on sample data. It is possible to determine which profiling information it collects about a particular program via a set of parameters.

P^{3}T, a parameter based performance prediction tool [13, 12], may be applied to a (partially) parallelized program to statically compute a set of parameters to help evaluate the behaviour of the parallel program on a target machine or to judge the performance impact of a particular parallelization step. These parameters can be selectively determined for individual program constructs and parts of the program. They are partially target machine specific. The resulting detailed information can help determine which parts of a program need to be improved and the kind of improvement which is needed.

The final performance tool is **MEDEA**, a postmortem analyzer of actual program performance, which is able to take a trace file generated using VFCS and perform a statistical analysis and workload modeling of its run time behaviour [21].

We discuss XPA, the final tool in the parallelization environment, and its interaction with the remaining tools in the following section.

5 Implementation of XPA and Integration With Tools

In this section the experimental prototype of XPA is described. This first version serves as a flexible and **expandable** skeleton tool enabling further experimentation with knowledge-based support for parallelization of Vienna Fortran codes and for rapid implementation (prototyping) of new ideas.

The skeleton comprises substantial components and it will be gradually extended and refined according to the future requirements.

5.1 Major Design Principles

XPA is designed as a **separate subsystem** of the parallelization system with a precisely specified interface to other components of the system. Among other things, this enables realization of XPA using a specialized (and independent) knowledge representation and inference tool.

The important design requirement for XPA is that there must be a flexible means of communication between it and both the user and other parts of the system. This is realized by an **interactive interface** to the other components of the system, to the user and to the developer of the knowledge base. This flexible interface also forms the basis for **gradual evolution** of XPA from a system with intensive user interaction to a largely independent automatic system. The interactive mode of interface is important especially for the development and refinement of the knowledge base in situations where several or most of the relevant aspects of parallelization are not yet understood or they are only vaguely specified. This permits easier experimentation with the system and the acquisition of the relevant knowledge. As soon as the experimentation grants yields a fixed and refined procedure for solving a particular problem, this stable procedure is incorporated into the knowledge base in the form of rules and the interface to the user is replaced by the corresponding interface routines, which gather the necessary information and its control flow from other parts of the parallelization system without user assistance. Based on this experimentation, the interface routines are also gradually enhanced.

XPA's knowledge base contains very different components and it is expected to grow considerably because of the complexity of the parallelization problem domain. A flat organization of the knowledge base with simple objects will not suffice in this situation. Therefore, the knowledge base is **hierarchically organized**, and makes use of complex, structured objects [19].

5.2 Implementation Environment

The **ProKappa Expert System Development Environment** from IntelliCorp [22] has been chosen for the XPA implementation. This professional environment meets our design requirements. We describe some of its most important features and outline their exploitation in the framework of XPA:

- both ProKappa and VFCS are **C programmed systems**. This greatly facilitates their integration. In particular, direct access to internal VFCS data structures and procedures is easily achieved, and C procedures may be used to realize parts of XPA when appropriate.
- The ProKappa environment incorporates a full and effective implementation of the **object-oriented paradigm**. ProKappa's object system supports natural, flexible, modular, and expandable representation of XPA information components, such as program constructs (do-loops, statements), data and control flow information, data dependencies, transformations, performance parameters, data distributions, and target architecture features. The methods and monitors are used in automatic filling and checking of values in object slots.
- **Active Relations** allow specification of simple local relations among objects on the slots level. They are convenient in the framework of XPA to accomplish automatic local information distribution and classification, mainly for preprocessing of information obtained from other subsystems, e.g. for classification of array access patterns for a selected do-loop construct from the code currently being processed.
- The shell's **inference engine** supports both **forward** and **backward chaining** of

rules, with **automatic backtracking** and searching for alternative solutions. The **ProTalk descriptive language** allows the specification of rules in a humanly readable form. Pattern matching drives the application of rules. Rules are grouped into rulesets; the invocation or disabling of each ruleset can be explicitly controlled. In XPA, the ruleset mechanism is used to split the large solution space into several smaller ones, thereby reducing the solution complexity and enhancing modularity. The focus of the solver is moved according to the current subgoal by enabling or disabling particular rulesets. The XPA contains the supersets of rules for following solvers: a data distribution solver, transformation solver, performance estimator and evaluator, pattern matcher, special construct solver.

- **Developer and End User Interface** tools support not only a comfortable environment for the creation and testing of ProKappa applications, but also facilitate construction of the interactive interface between XPA and the user of the parallelization system.

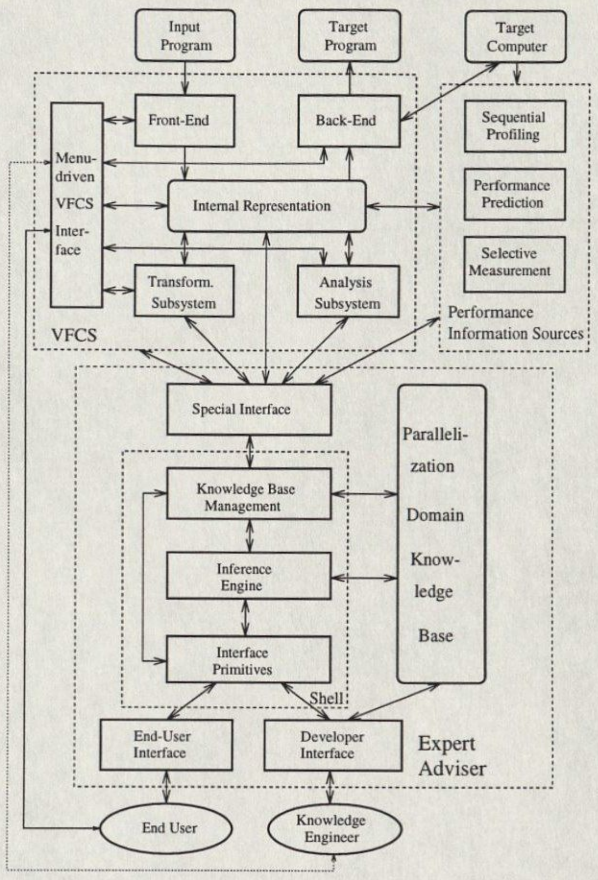


Figure 1: Knowledge-Based Parallelization Environment

5.3 The Structure of XPA

A block diagram of XPA and its connections to the other components of the parallelization environment is given in Figure 1. The facilities of the ProKappa development environment (**Knowledge Base Management** support, the **Inference Engine** and **Interface Primitives**) are customized to meet the specific requirements for the knowledge base representation and the inference process of XPA.

A special communication module is needed to accomplish the high level of cooperation required between XPA and the other subsystems of the parallelization environment. This **special interface module** comprises not only **tools for the interchange of data**, but also includes **mechanisms for a dynamic transfer of control flow** among the cooperating subsystems. This is crucial for achieving a flexible interactive access to the transformation system as described above. The main purpose of this special interface, however, is to establish *consistency* and *synchronicity* of the contents of the knowledge base with the internal representation of the program in other parts of the parallelization system.

To simplify the special interface module and to make the data acquisition transparent to the knowledge base programmer, separate hierarchical levels of the knowledge base are implemented. **Knowledge-driven data acquisition** is achieved by the fragmentation of data into smaller pieces, whose transfer can be accomplished by simpler interface routines, and which also makes an intelligent and complex grouping of the acquired information into structured objects possible. Note that individual slots of the object describing some entity can be filled in quite different stages of the inference over the knowledge base, corresponding, for example, to the gradual refinement of the solution proposed by the system.

5.4 XPA's Sources of Information

From a theoretical point of view, the problem of finding the optimal parallelization strategy for a particular program can be seen as searching through a *solution space*. The solution space for parallelization is very large. If the system is to find the optimal solution in a reasonable time, it needs additional information that can be used to cut off solution paths which are not promising as soon as possible. The system has to focus itself on the most important solution path. Hence, the system needs to have effective feedback of information from other parts of the parallelization system.

One important source of information already mentioned are the internal structures of the VFCS. From these, XPA can obtain basic information about the structure of the program, its control and data flow, and the results of data dependence, etc.

Since it is the goal of XPA to find an optimal parallelization strategy for a particular input program, and target computer configuration, the available information must be specific to a class of machines and be adaptable to the given configuration. It is vital in this context that XPA is able to receive or derive information on how a particular transformation, sequence of transformations or a specific data distribution will influence the performance of the code if applied, and whether it improves or deteriorates the target program.

The optimal situation is when such information can be obtained analytically based only on internal VFCS structures. Note that the acquisition of information about potential performance gain can involve non-trivial computation. Many known heuristics can be applied. In more complicated cases, the specialized ruleset of the XPA knowledge base must be used; this is particularly likely when parameters of the target computer architecture should be taken into account. Whereas a code analysis suffices in many cases, there exist situations where either an analytical solution or heuristics are not known, or the precision of the estimated performance gain is not satisfactory. In these cases, the performance prediction tools and the selective target code measurements are required to provide more information to the inference process.

At our institute, various subsystems of the parallelization environment have been developed, or are under development to support both the user and XPA during parallelization (see 4). They provide XPA with the following information:

- **sequential profile data for the input program** [11] - besides timing of important programming constructs, this tool provides both frequency information on a statement basis and actual true ratios of conditional statements. Within XPA, components of the sequential profiling tool are utilized to automatically identify the most computation intensive parts of an actual program.

- **parallel performance parameters** [12] - include work distribution, the number of data transfers, the amount of data transferred, transfer times, network contention and the number of cache misses. The parameters are derived without execution of the input program. These tools use both analytical methods [13] and simulation [3] to derive performance related parameters. They are of particular importance and usefulness for the evaluation and comparison of different data distributions, which largely determines not only the load balance on the executing processors but also the amount and the structure of the communication between them. For many restructuring transformations, whose purpose is often to improve data locality of the resulting node processor code and to exploit vector capabilities of single processors of the target machine (e.g. loop interchange, loop unroll, unroll and jam, strip mine), the current performance prediction tools are still not very satisfactory. This is partially due to the significant influence of the target processor architecture and to an imprecise knowledge of the vectorizing and pipelining capabilities of the target node compiler and their impact on program performance. These features cannot be simply modeled. One possible solution is to dedicate a part of XPA's knowledge base to this performance evaluation and use an extensive set of specialized measurements on the real target machine.

- **selective measuring of target machine performance** - the most precise information, generated by the selective instrumentation and successive execution of parallel code and the postprocessing of the resulting trace file by statistical methods [21] (including clustering) [7].

5.5 Knowledge Base of XPA

To make the parallelization process more automatic, a knowledge-based parallelization tool has to include various qualitatively different aspects of parallelization. The knowledge-based approach aims to see the parallelization process as a whole. A necessary precondition for that is that the knowledge about different parts of this process are contained in a common base. A uniform representation is required to allow the connection of fragmented, partially known solutions of individual parallelization steps, as have been developed in the past. Moreover, it must provide a framework for covering both human knowledge of parallelization (parallelization experience, heuristics, shortcuts) and the precise analysis and transformation techniques contained in current parallelizers. Hence, the implicitly imprecise, and often ad hoc, human solutions can be automatically improved by exploiting precise explicit methods, and vice versa.

The knowledge base for automatic parallelization consists of the following knowledge clusters:

- **configuration description** - a description of the input and output programming languages and target architecture parameters; it is supposed that XPA will be used in different programming environments (**VFCS, and other HPF compilers**) and that the target code may be generated for various target machines; so as a consequence the knowledge base must incorporate language specific features such as the data distribution types available with their characteristics, the language constructs for expressing the number and structure of processors, and array dimension alignment, and target machine characteristics such as the number of processors and its topology, as well as it communication characteristics.

- **input program features** - this knowledge is extracted automatically by the special interface from the input program's internal representation; in some cases it can be acquired by communication with the user (we will try to reduce communication with the user, however, by filling the knowledge base appropriately when possible); the special interface will have to be developed separately for different compilers in the environment; some knowledge can be deduced; some knowledge can be gained by the usage of special tools: the (**Weight Finder**) can be used, for example, to acquire information about some program fragments (do-loops, statements, procedures).

- **performance parameters and related**

knowledge - parameters gained from performance prediction tools; these parameters characterize the input program or current program version by expressing the expected performance features of the program under parallization; the knowledge used in evaluation of these parameters and expressing the feedback to parallization process is also included.

- **data distribution knowledge** - the library of array reference patterns with appropriate data distribution forcing (of course in the context of array sizes and the target machine); the library of code patterns with target machine optimized codes including data distributions; the knowledge for intraprocedural and interprocedural distributions propagation.

- **transformation descriptions** - information on the transformations available for selection in compiler environment (preconditions for their applicability, summary of their effect on performance) and their possible combinations (overall performance of such combination or refutation of certain combinations of transformations).

- **state of parallelization process** - log information recording transformations were already applied and generated recommendations, the current topic in the parallelized program, which units have been processed, which integrated phases have been done, current status of analysis information.

- **application domain dependent information** - knowledge about the application domain to which the input program belongs can have a significant impact on the reasoning process; some applications domains can be characterized by special computational patterns often used in algorithms and by the similar structure of data used in them (e.g. symmetric matrices) and where there is experience in parallelizing the programs belonging to the domain, this knowledge is incorporated in the knowledge base.

- **general strategy knowledge** - is based on known configuration of the target machine, the input language and transformation system features. Incorporating such knowledge can drastically decrease a search space of the solution. Also already known global heuristics (or strategies) for the certain configuration can be applied directly.

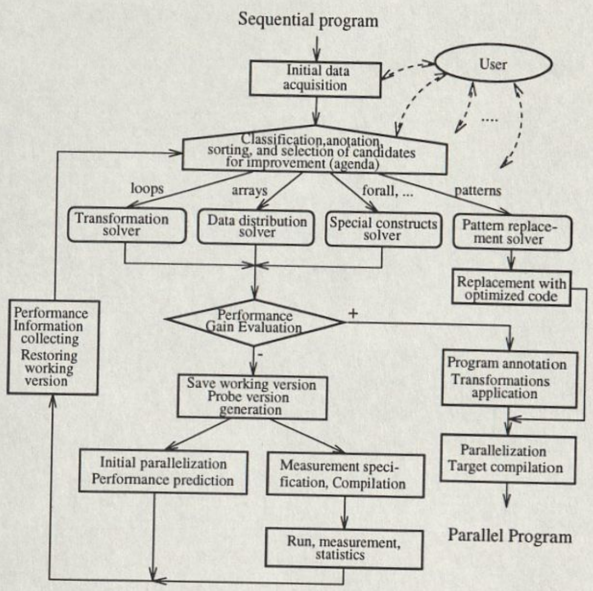
The knowledge concerning parallelization domain takes two forms: objects and rules. Roughly speaking, the objects describe the current state of the parallelization process, and the rules describes relations among objects, or they correspond to very elementary actions for changing objects.

Based on the given parallelization goal and information stored in the knowledge base, the *inference engine* performs *reasoning* over them. Note that the inference over the knowledge base is the most important mechanism for achieving solutions. Upon the strength and flexibility of the inference engine features depends the total performance of the solution searching.

5.6 Solution Flow

A functional view of the steps taken by XPA during the parallelization session is shown in Figure 2.

The initial data acquisition module first runs the VFCS Front End on the input program. Sequential profiling is then performed. After these, the internal representation of the source program is available, the call graph has been constructed and data flow and dependence analysis has been computed. The abstract syntax tree has been annotated with profiling results. Based on these internal VFCS structures, the XPA creates its initial internal program representation in the form of interconnected objects. We note that items of information from VFCS are not simply copied:



Sequential program
Initial data acquisition
User
....
Classification,anotation, sorting, and selection of candidates for improvement (agenda)
loops
arrays
forall, ...
patterns
Transformation solver
Data distribution solver
Special constructs solver
Pattern replace-ment solver
Replacement with optimized code
Performance Information collecting
Restoring working version
Performance Gain Evaluation
+
-
Program annotation Transformations application
Save working version Probe version generation
Initial parallelization Performance prediction
Measurement speci-fication, Compilation
Parallelization Target compilation
Run, measurement, statistics
Parallel Program

Figure 2: XPA - a functional view

they are preprocessed to achieve the required level of abstraction. The resulting profile information is also included in objects. When ready. the object structure is scanned to find parts suitable for successive improvement. This scanning is accomplished by the use of pattern matching. The selected parts are then classified according to the solver suitable for their further processing, and they are annotated with the expected performance gain (computed or estimated). Thus obtained, *candidates for improvement* are sorted according to an estimate of their contribution to the performance of the complete program. The candidates are then submitted in this order to the appropriate solver.

Figure 2 shows four solvers. Our current work is concentrated on the first two: the *transformation solver* and *distribution solver*. The other two are included for the sake of completeness. The *special constructs solver* is expected to be dedicated for processing of special constructs: VFCS has routines for dealing with irregular computations [6], and a number of distribution methods which are considerably more complex than the widely understood block and (block-cyclic) distributions, such as those for distributing work spaces [16] . Analysis of their use is deferred to this solver. The *pattern replacement solver* will replace specific recognized code patterns with pre-optimized and parametrized code for the target machine.

It is the task of the transformation solver to find an optimal sequence of code restructuring transformations, and the task of the data distribution solver is to generate the best data distribution(s) for the input program. The decisive criterion is the target program's performance. Although the diagram shows these solvers as separate modules, in reality there is a strong interrelation between them, since achieving the optimal solution requires a sophisticated matching of transformations and data distributions. Each solver represents an iterative process (reasoning) over the knowledge base. A more detailed discussion of the solvers exceeds the framework of this paper.

As explained in the previous section, feed-back from tools for sequential profiling, performance prediction and selective measurements by the appropriate module is often necessary. Note that the preciseness of performance information for the parallel program increases from the first of these tools to the last. But the cost which must be paid for such precision is the increasing computational time taken to gather the information and the complexity of the task. A decision to perform performance prediction or make selective measurements must be made carefully and sparingly. XPA should use these tools seldom in an actual parallelization, compared to the other sources of information, and only when the state of its solution flow strongly requires information of this kind to proceed with the solution, and the required performance parameter cannot be reasonably obtained or deduced from available information.

5.7 Automatic Data Distribution within XPA

The data distribution process can be divided into the following four subgoals:

- finding the relative locations of different arrays elements through dimension alignment
- finding the appropriate number of processors of target machine on which the program will execute
- finding their structure reflecting the way these processors will be accessed
- finding the appropriate mapping of the array elements to processors of the parallel machine

The aim of automatic data partitioning is to reach the automatical solution of above mentioned subtask and insertion of appropriate Vienna Fortran constructs into the input program to express the solution of these subtask. The solution should exhibit the acceptable performance of resulted parallelized program. The method by which the data are distributed largely determines the process structure of the parallel program and in particular, the size of the workload for each process and the communication required; hence, the data distribution also determines the overall performance of the parallelized program. Some aspects of the data distribution are known to be solved analytically [10]; the computational cost of solving more general cases is prohibitive.

The automatic data distribution within XPA is performed in several phases. The first phase is dedicated to the automatic *array dimension alignment.* This and next phase is performed on the important program parts (found with the help of serial profiler). When contradictory align preferences for the alignment of some array dimension in some program part occur, the tool chooses the most appropriate one. The library of data refference patterns with the alignment preferences forced by these references is used.

The second phase is focused on finding the appropriate *types of data partitioning* and their *specifications* for arrays in these important program parts. Array sizes and communication characteristics of different communication kinds which are accessable on the target machine are taken into account.

In the third phase, the data distributions are propagated between the individual program parts and throughout the call graph [9]. This step have to solve the potential *conflicts* arising when different data distributions for some array have been found in different program sections. *Array redistribution* is also considered as a method for solving conflicts during the *inter and intra-procedural propagation.*

In the fourth phase data distributions found are *evaluated and tuned.* The information obtained from performance prediction tool or parallel profiler is used to find the program parts and data references that were the sources of poor decisions in the previous phases; that allows a more intelligent search for the cause of poor decisions and a corresponding improvement in the system's behaviour.

6 Related Work

Knowledge-based techniques have previously been applied in few systems which restructure code for vector or parallel machines.

The expert adviser EAVE was developed to assist in the transformation of program for input to the IBM 3090 VF; it also looks for patterns within the code, and has rules telling it how code can be transformed to obtain an equivalent form that can be efficiently vectorized [2].

Wang and Gannon [23] proposed an expert system for the hierarchical parallelization of programs to run on different multi-processors architectures. The proposal concentrates on the formal representation and specification of the knowledge about the transformation process. Wang has constructed an advanced knowledge acquisition tool in Prolog for collecting knowledge about various target machines architectures [24].

The experimental InParS system [1] implements heuristic state search algorithm A*. It is applied to automatically generate the sequence of transformations for optimal vectorization on IBM 3090/180S VF machine. A simple performance prediction is used to drive the search process.

The rule based transformation system ParTool [5] for semiautomatic SPMD code generation is being developed at the Delft University of Technology. This system provides advanced pattern matching, but unfortunately lacks a flexible control mechanism for application of the rules.

7 Conclusion

In this paper we have discussed a knowledge based approach to parallelization and have introduced the design rationale and features of XPA, a tool which is being developed as part of an advanced parallelization environment for DMMPs, whose core is the VFCS parallelizer. XPA is designed to interact closely with other tools of the environment to support the automatic parallelization of data-parallel numerical programs written either in Fortran, Vienna Fortran or HPF. We hope that its development will not only improve our understanding of the techniques which a parallelizer can effectively employ, but also bring us a step closer to full automatic parallelization. Further, it is intended that this system be relatively easy to adapt to different architectures, so that efficient parallelizers for emerging systems with different architectural parameters can be quickly constructed.

This system makes extensive use of knowledge-based and performance analysis techniques.

References

[1] Al-Ayyoub A., Yazici A.:"Knowledge-Based Program Parallelization", Computer Science Department, Bahrain University, Isa Town, February 1992, 16 pp.

[2] Bose P.: "Interactive Program Improvement via EAVE: an Expert Adviser for Vectorization", Proc. of the Int. Conf. on Supercomputing, St. Malo, July 1988, pp. 119-130.

[3] Blasko R.: "Parametrization and Abstract Representation of Parallel Fortran Programs for Performance Analysis", Proc. of the AICA'93 Conf., Parallel and Distributed Architectures and Algorithms, Sept. 1993, Gallipoli-Lecce, Italy, pp. 75-91.

[4] Blasko R., Zima, H.: "Performance Prediction and Expert Adviser for Automatic Parallelization of Fortran Programs", to appear.

[5] Breebaart L.C., Paalvast E.M., Sips H.J.: "A Rule Based Transformation System for Parallel Languages", In: Proc. of Third Workshop on Compilers for Parallel Computers, Vienna, Austria, ACPC/TR 93-8, July 1992, pp. 13-21.

[6] Brezany P., Sipkova V., Das R., Saltz J.: "Compilation of Vienna Fortran Forall Loops", submitted to Supercomputing'93, Portland, Oregon, 1993.

[7] Calzarossa M., Serazzi G.: "Workload Characterization: A Survey", Proceedings of the IEEE, 1993, to appear.

[8] Chapman B., et al: "VIENNA FORTRAN Compilation System. Version 1.0. User's Guide", Department of Statistics and Computer Science, University of Vienna, Austria, January 1993, 192 pp.

[9] Chapman B., Fahringer T., Zima H.: "Automatic Support for Data Distribution on Distributed Memory Multiprocessor Systems", Sixth Annual Workshop on Languages and Compilers for Parallel Computing,

Portland OR, August 1993, Springer Verlag LNCS, to appear

[10] Chen M., Li J.: "Optimizing Fortran 90 Programs for Data Motion on Massively Parallel Systems", Technical Report YALE/DCS/TR-882, Yale University, January 1992.

[11] Fahringer T.: "The Weight Finder - An advanced Profiler for Fortran Programs", Proc. of AP'93, Saarbrücken, Germany, 1993.

[12] Fahringer T., Blasko R., Zima H.P.: "Automatic Performance Prediction to Support Parallelization of Fortran Programs for Massively Parallel Systems", The Sixth ACM Int. Conf. on Supercomputing, Washington D.C., July 1992, 10p.

[13] Fahringer T., Zima H.: "A Static Parameter based Performance Prediction Tool for Parallel Programs", Proc. of the 7th ACM International Conference on Supercomputing, Tokyo, Japan, July 1993.

[14] G. Fox, S. Hiranandani, K. Kennedy, C. Koelbel, U. Kremer, C. Tseng, and M. Wu: "Fortran D language specification", Department of Computer Science Rice COMP TR90079, Rice University, March 1991.

[15] High Performance Fortran Forum: High Performance Fortran Language Specification, Version 1.0", Rice University, Houston, May 1993

[16] Gerndt M.: "Parallelization of Multigrid Programs in Superb", Technical Report ACPC/TR 90-6, Austrian Center for Parallel Computation, October 1990.

[17] Gupta A., Prasad B.E. (eds.): "Principles of Expert Systems", IEEE Press, 1988.

[18] Harandi M.T., Ning J.Q.: "Knowledge-Based Program Analysis", IEEE Software, January 1990, pp. 74-81.

[19] Liebowitz J., De Salvo D.A. (eds.): "Structuring Expert Systems", Prentice-Hall Inc., 1989.

[20] *"MIMDizer User's Guide, Version 8.0"*, Applied Parallel Research Inc., Placerville, CA., 1992.

[21] Pantano M., Blasko R., Moritsch H., Fahringer T.: "VFCS-MEDEA Integration. Instrumentation. Parameters and Events Specification.", Institute for Software Technology and Parallel Systems, University of Vienna, Austria. Internal report, 1993.

[22] "ProKappa User's Guide. Version 2.0". IntelliCorp, Inc., Publ. Number PK2.0-UG-2, October 1991.

[23] Wang K.Y., Gannon D.: "Applying AI Techniques to Program Optimization for Parallel Computers", Hwang. K., DeGroot D. (Eds.): Parallel Processing for Supercomputers and Artificial Intelligence, McGraw-Hill Pub. Company, 1989, Chap.12., pp.441-485.

[24] Wang K.Y.: "Intelligent Program Optimization and Parallelization for Parallel Computers", Tech. Rept. No. CSD-TR-91-030, Department of Computer Science, Purdue University, April 1991.

[25] H. Zima, H. Bast, and M. Gerndt. "Superb: A tool for semi-automatic MIMD/SIMD parallelization", *Parallel Computing*, 6:1–18, 1988.

[26] H. Zima, P. Brezany, B. Chapman, P. Mehrotra, and A. Schwald: "Vienna Fortran – a language specification", ICASE Internal Report 21, ICASE, Hampton, VA, 1992.

[27] Zima H., Chapman B.: "Compiling for Distributed-Memory Systems", Proc. of the IEEE, Special Section on Languages and Compilers for Parallel Machines, February 1993.