TransTool: A Restructuring Tool for the Parallelization of Applications Using High Performance Fortran *

A. Darte, F. Desprez, J.C. Mignot and Y. Robert †
LIP, URA CNRS 1398, INRIA Rhône-Alpes
Ecole Normale Supérieure de Lyon, 69364 Lyon Cedex 07, France
e-mail: Firstname.Lastname@ens-lyon.fr

October 16, 1996

Abstract

In this paper, we present the TransTool project, whose aim is to design a restructuring tool for the transformation of Fortran 77 programs into High Performance Fortran (HPF). TransTool includes an editor, a parser, a dependence analysis tool and an optimization kernel. Moreover, we provide the users with a clean open interface, so that developers of tools around HPF can easily integrate their software within our tool.

Keywords: Semi-automatic parallelization, High Performance Fortran (HPF), Restructuring Tools.

1 Introduction

The parallelization of applications on high performance distributed memory computers has gained importance in the industry since the emergence of projects like Europort. These projects have shown that the port of industrial applications on parallel machines was a difficult task. The High Performance Fortran (HPF) language has opened new perspectives but its use is still constrained by the availability of efficient compilers and tools.

It is unlikely that we will ever see a "black box" able to parallelize a non-trivial serial code into a high-performance parallel code. The user needs help the compiler by giving information about his code. This can be done for example via directives inserted in the source file like in HPF. But it is now admitted that the insertion of such directives is a non trivial task for average users, who do not have a deep knowledge of parallelization techniques. Therefore, interactive parallelization tools have a great importance in parallel computing, even in the limited field of numerical problems.

The remainder of this paper is organized as follows. In Section 2, we explain how difficult it may be for the user to write a "good" HPF program, thereby motivating an interactive tool like TransTool. Section 3 gives a short survey of tools for data-parallel programming. Section 4 presents the HPFIT project, a collaborative action whose kernel is TransTOOL. In Section 5, we give some

*This work is partly supported by the CNRS–ENS Lyon–INRIA project *ReMaP* and by the *LHPC–EuroTOPS* project.

†Currently on leave at the University of Tennessee at Knoxville.

details about the development of a parallel application and TransTool itself. Section 6 presents two research directions in the TransTool optimization kernel and Section 7 offers some conclusions and ideas for future work.

2 Squeezing parallelism out of HPF programs

In this section, we explain why interactive tools and automatic parallelization techniques are needed to squeeze (at least some) parallelism out of HPF programs. Not only the efficient compilation of HPF programs is a difficult task by itself, but also source-to-source transformations are required to feed the compiler with programs in a form suitable for parallelization. We assume the reader has a basic knowledge of the HPF language, including alignment, distribution and parallelization directives such as `ALIGN, DISTRIBUTE, FORALL, DO INDEPENDENT`. Please refer to [28, 21] for an overview of HPF.

Consider the following simple example:

do $i = 1,\ M$
  do $j = 1,\ N$
    $u(i,j) = f(u(i-1,j), u(i,j-1))$
  enddo
enddo

None of the two loops can be made parallel, because of dependences: to compute node (i,j) (i.e. the value of $u(i,j)$), we must have already computed node $(i-1,j)$ (hence the loop on i is sequential) and node $(i,j-1)$ (hence the loop on j is sequential). This means that the HPF compiler is most likely to generate a parallel code that would run more slowly than the sequential version (because of the communication overhead). However, all iterations (i,j) such that $i+j=C$ are clearly independent (think of a anti-diagonal wavefront progressing among nodes). To exhibit this hardly hidden parallelism, let

$$\begin{pmatrix} t \\ p \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} i \\ j \end{pmatrix},$$

and rewrite the loop as:

do $t = 2,\ M+N$
  forall $(p = \max(1, t-N) : \min(t, N))$
    $u(t-p,p) = f(u(t-p-1,p), u(t-p,p-1))$
enddo

Now we can *distribute* the columns of u to the processors, to get a source code that a HPF compiler will know how to parallelize efficiently. We point out that the transformations that we have applied on the original loop nest make use of rather sophisticated mathematical tools. M and N are parameters that are unknown at compile-time, and rewriting the loop nest requires computing parameterized loop bounds. Also, we have chosen a transformation matrix $(i,j) \longrightarrow (t,p)$ whose determinant is equal to 1. This is not always possible, and non-unimodular transformations are more complicated to deal with.

Let us move on to a slightly more challenging example, due to Peir and Cytron [32]:

do $i = 0$, N
  do $j = 0$, N
    $\{ S_1 \}$ $a(i,j) = b(i,j-6) + d(i-1,j+3)$
    $\{ S_2 \}$ $b(i+1,j-1) = c(i+2,j+5)$
    $\{ S_3 \}$ $c(i+3,j-1) = a(i,j-2)$
    $\{ S_4 \}$ $d(i,j-1) = a(i,j-1)$
  enddo
enddo

Here, the user cannot guess the answer to the following questions:

- Which nodes (i,j) can be executed in parallel ?
- How to align the four arrays a, b, c et d so as to minimize communication cost ?

However, this loop nest looks like the previous one: it is perfectly nested, and dependences between statements are easy to compute: they are translations. Applying the same kind of matrix transformations, we would "automagically"[1] transform the loop nest into something like:

```
   REAL a(n,n),b(n,n),c(n,n),d(n,n)
!HPF$ TEMPLATE BCLT_0_template(n+3,n+21)
!HPF$ DISTRIBUTE BCLT_0_template(BLOCK,*)
!HPF$ ALIGN a(i1,i2) WITH BCLT_0_template(i1,i2+21)
!HPF$ ALIGN b(i1,i2) WITH BCLT_0_template(i1+2,i2+7)
!HPF$ ALIGN c(i1,i2) WITH BCLT_0_template(i1+3,i2)
!HPF$ ALIGN d(i1,i2) WITH BCLT_0_template(i1,i2+21)
   DO T = 21, 8*n-26
!HPF$ INDEPENDENT
      DO P = ceiling(max((-n+T+5.0)/7.0,2.0)), floor(min(T/7.0-1.0,n-3.0))
        a(P,T-7*P) = (b(P,T-7*P-6)+d(P-1,T-7*P+3))
        b(P+1,T-7*P-1) = c(P+2,T-7*P+5)
        c(P+3,T-7*P-1) = a(P,T-7*P-2)
        d(P,T-7*P-1) = a(P,T-7*P-1)
      END DO
   END DO
```

Only after such a major source-to-source transformation will it be possible for the HPF compiler to generate efficient parallel code. There remains much more to do. For instance, it is of dramatic importance that the compiler is able to determine that the remaining communications are indeed shifts along the computational domain, and to generate the corresponding optimized macro-communication code. Generating a collection of point-to-point communications instead would lead to very poor performance.

Finally, consider the following loop nest:

do $i = 0$ **to** N do
  do $j = 0$ **to** M do
    $\{ S_1 \}$ $b(i+j-2,1,3-j) = g_1(a(j-1,i+6),$

[1] See [13, 15, 14] for technical references.

$$a(i+j, -4i-5j), c(i-1, j+3, i-j))$$

do $k = 0$ **to** $N + M$

$\{ S_2 \}\ b(i+2, j+5, k-3) = g_2(a(j-k+1, i-j+k-2))$

$\{ S_3 \}\ c(i+j+k, j-1, i-k) = g_3(a(i+j+k, 0))$

enddo

enddo

enddo

Now, we face a non perfectly nested loop nest with affine expressions everywhere! g_1, g_2 and g_3 are arbitrary functions, what really matters is how arrays a, b and c are accessed. We still need to map these arrays onto a processor grid so as to minimize communication overhead. With this example, we are close to the current limitations of state-of-the-art techniques. See [20] and the references therein for further material on mapping techniques.

To summarize this section, we state the following:

- HPF requires the user to provide alignment, distribution and parallelization directives. In many cases it is not realistic to assume that the user will be able to do so.
- Even when "good" directives are provided, there are many optimizations (such as message vectorization, for instance) that are mandatory to achieve reasonable performance.

The good news are that automatic parallelization techniques can help fill the gap between the user and the compiler. But there are many program transformations to be tried and compared, hence the need for an interactive parallelization tool to help the user find his way.

3 Previous Work

A lot of work has been done since the early eighties around tools for supercomputer programming. These early tools need to be enhanced to follow the development of parallel computing. In this section, we present some tools for the parallelization of applications written in Fortran 77 and HPF. For a comprehensive survey of HPF tools, see [31].

One of the first projects around an "intelligent" editor for the parallelization of applications written in Fortran77 was the ParaScope editor (PED) from Rice University [27]. PED was designed for shared memory machines. It has been built from different other projects at Rice, like $\mathbb{R}_n$, PFC and PTOOL. PED allowed the search of a dependence graph whose size was limited by some filtering information. Several optimizations were added to the tool like loop restructuring, loop parallelization, dependence deletion and memory optimization. PED has been using an incremental analysis to update its text and dependence panes.

The D Editor [22] has been partly developed from PED also at Rice University. This editor has been designed for the parallelization of applications written in Fortran D, a data-parallel language which turned out to be a major input to the design of HPF. The D-editor contains an interprocedural analysis tool, the Fortran D compiler and tools for automatic data distribution, data-race detection, static performance estimation and performance profiling. The graphical display is derived from PED and contains five panes: an overview pane provides a summary of the loops and subroutines in the program (loops which restrict parallelism are highlighted); a dependence pane displays the data dependences carried on the selected loop; the communication pane displays all the communications associated with the selected loop; the data layout pane displays the data decomposition information for each array of the loop; and finally the source pane shows the actual program code. The performance analysis environment Pablo has also been integrated within the D editor [1].

The Vienna FORTRAN Compilation System (VFCS) [35] is a source-to-source compilation system based on Vienna Fortran, another extension set for Fortran, similar to FortranD and HPF. It contains a compiler, an interactive performance estimator P3T [11] and a performance measuring system (VFPMS).

ForgeExplorer is an interactive parallelizer based on an interactive source code browser. It also performs loop level transformations.

The Annai tool environment [12], developed by CSCS in a collaboration with NEC, uses MPI as the communication interface for various distributed memory platforms: NEC Cenju-3, Cray T3D, Intel Paragon, and Unix multiprocessor/networked workstations. It consists of an extended HPF compiler (with extensions for irregularly structured computations), a parallel debugger and performance monitor and analyzer, designed with important feedback from application developers.

The Computer Aided Parallelization Tools (CAPTools [23]) is a recent set of tools developed at the University of Greenwich. This tool can show dependences using a graphical display. The user can interact inside the parallelization process by giving information about values or ranges of variables, by "deleting" dependences, ...It has a Partitioner window to partition his code and data structures, a communication browser to see the generated communication routines calls. The code generated by CAPTools is written in F77 and explicit communications routines calls.

To conclude this section, we point out that other important compilation projects are being undertaken, like SUIF [3] and Paradigm [4].

4 The HPFIT Project

TransTool is the kernel of the **HPFIT**[2] project [6, 7]. Its aim is to provide a set of interactive tools integrated in a single environment to help users to parallelize scientific applications to be run on distributed memory platforms. This tool will also be used as an interface to a large number of existing tools like HPF compilers, computation libraries, simulators, data visualization tools, monitoring systems, profilers, and so on. These tools will interact in a coherent environment. This environment should allow scientists with sequential source codes to produce good parallel versions. Furthermore HPFIT will enable users to control the parallelization of their application, in order to make it even better.

HPFIT will not be a black box taking a sequential application as input and magically producing an efficient parallel application as output. Rather, HPFIT will be a tool environment to support and ease the development, tuning and maintenance of HPF applications. Though it is intended to be an open environment that allows integration of new tools, we focus in the first version of our tool on the following items:

- source editing with analysis information,
- dependence analysis of particular loop nests,
- automatic detection of INDEPENDENT loops,
- automatic detection of pipelined computations patterns and code generation,
- support for data mapping (adding data distribution directives),
- visualization and evaluation of data mappings,

[2] **H**igh **P**erformance **F**ortran **I**ntegrated **T**ools.

- interfaces to existing parallel libraries (e.g. ScaLAPACK, ...),
- interfaces to HPF compilers,
- simulation, monitoring, performance analysis and interface with profiling tools,
- compilation and execution interface.

The parallelized code is a data parallel program with explicit data mapping and explicit data parallelism. As the data parallel paradigm might not be the most efficient one in some situations, we will provide a library interface to codes written in other languages or other parallel programming styles, in particular message passing libraries, and efficient parallel computation libraries.

As most applications considered for parallelization are written in Fortran, and because a lot of work has been done for automatic parallelization of this language, our target language is HPF. Since HPF provides the `EXTRINSIC` mechanism, we can interface it with existing parallel libraries and other programming styles. Some HPF compilers are being released. These compilers are designed by software companies (like pghpf from PGI, DEC and IBM HPF compilers, ...) or universities (like ADAPTOR from GMD/SCAI [10], VFCS from the University of Vienna, sHPF from the University of Southampton, Fortran D from Rice University, PARADIGM from the Center for Reliable and High-Performance Computing at the University of Illinois at Urbana-Champaign, HPFC from the Ecole des Mines de Paris, ...). HPFIT should make possible for the user to use any of these HPF or HPF-like compiler, and all those to come, even if most of them will not allow some functionalities like monitoring, and translation of sequential library calls. We intend to make use of this variety of possible "post-compilers" in order to run HPF codes on nearly all kinds of architectures. For instance the ADAPTOR compilation system not only generates message passing code for MIMD system with distributed memory, but can also generate code for MIMD systems with shared or distributed shared memory. The first version of HPFIT (Version 1.0) will support two compilers: ADAPTOR from GMD/SCAI and pghpf from Portland Group.

The HPFIT project is based on several other projects developed in different universities. At the moment, 4 research groups have decided to work on the project and to design shared interfaces. These groups are the **LIP** in Lyon, France, the **LaBRI** in Bordeaux, France, the **LIFL** in Lille, France, and the **GMD/SCAI** in Bonn, Germany. The developed tools can be used either in a stand-alone fashion or within the HPFIT interface.

5 TransTool

The development of a "real" application is conducted in a cycle involving several steps. This cycle is shown on Figure 1.

Tools can be designed to help users especially during the distribution phase. The programmer must find a "good" distribution of data structures. This distribution should lead to the highest parallelism, and should reduce the number and volume of communications to its minimum. This can be evaluated using various tools (monitoring tools, profilers). From this distribution, the user is able to write HPF directives in the declaration part of the code. These distributions should be propagated inside the routines. They can sometimes differ and the programmer is allowed to redistribute the data inside the code. The code is then compiled to obtain a source code in Fortran 77 with message-passing calls. It is sometimes possible to modify the resulting code to insert optimizations. The code can then be executed or simulated. If traces have been generated, the user can have an idea of the behavior of his code, and the quality of the distribution (and

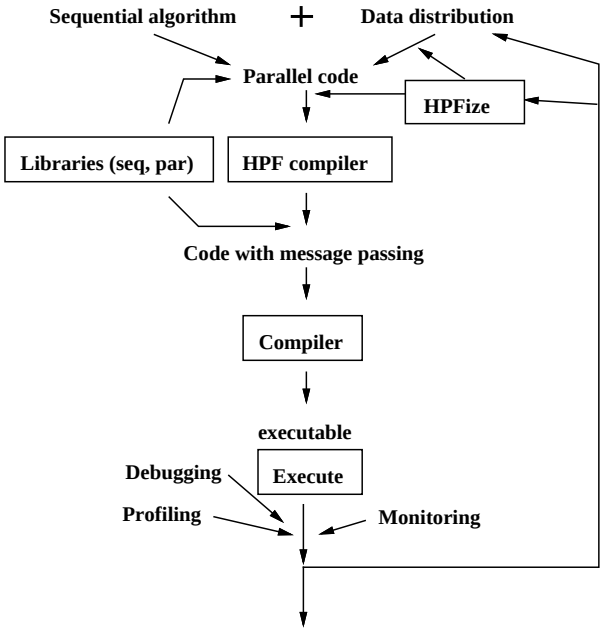

Figure 1: Development cycle of an application on a distributed-memory platform.

the optimizations) can be improved. This cycle can be executed several times to obtain the best performance. One question is: how portable is the resulting code?

One problem with HPF is that a HPF compiler is not required to follow the user's advice (stated as directives). This can be a problem because the user can have a false view of his code, "corrupted" by the compiler. That is why we think that a strong interaction between the compiler and the interactive parallelization tool is necessary.

TransTool, developed at LIP, is the kernel of the HPFIT project. At the moment, it contains a powerful editor (XEmacs), the F77 parser (from the **ADAPTOR** compiler developed at the GMD/SCAI lab.), the dependence analyzer (**Petit** at the University of Maryland) and an optimization kernel. At the moment, this kernel allows to do some parallelism detection and optimizations of pipelined computations (see Section 6). TransTool provides an interface showing the results of the parsing and of the dependence analysis. Figure 2 gives a snapshot of the TransTool screen.

5.1 The XEmacs Editor

The sequential program source is displayed by an XEmacs editor. Using a powerful editor enables us to share previously developed modes, to let the user configure its editor with his preferences (by using his regular .emacs file). The TransTool edition is suited to the target language using a modified F90 mode. HPF keywords are highlighted. The user has the opportunity to click on a program component to trigger some actions (like choosing a loop nest for the dependence analysis).

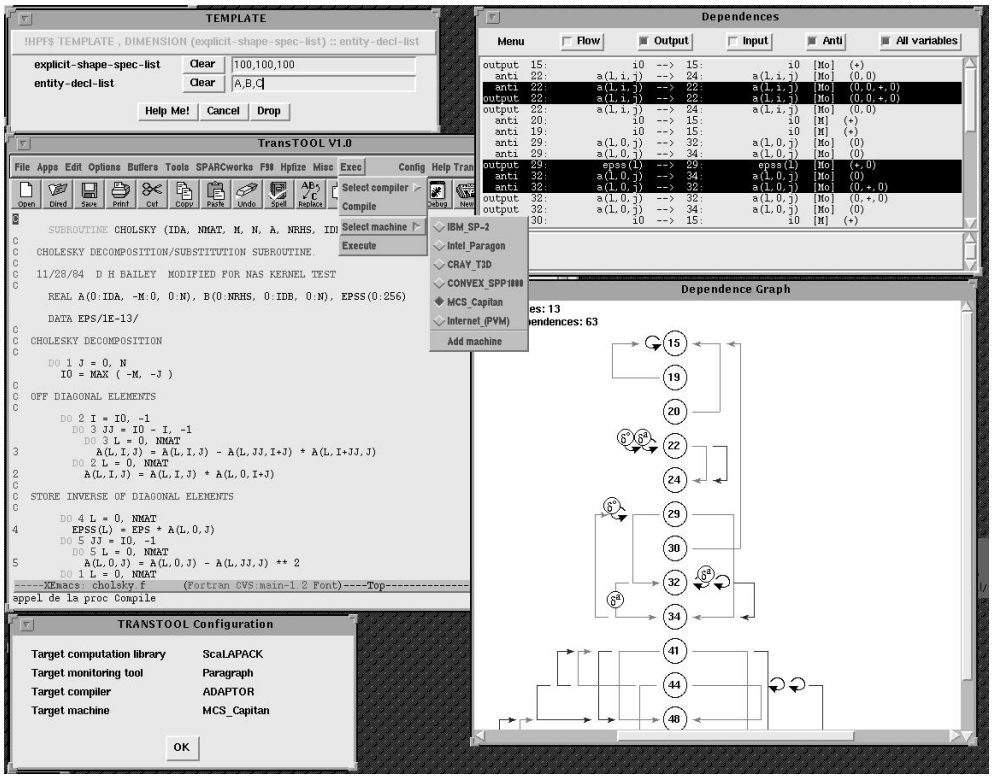

Figure 2: Snapshot of TransTool.

5.2 HPFize

Numerous applications have been developed on sequential, vector or parallel machines. These codes have to be modified to be executed on DPMCs. Writing a program in HPF consists in including compilation directives into the source code. It is interesting to simplify the way the user inserts such directives. This is achieved using a graphical environment that is able to get the necessary pieces of information from the user and from the program itself (after parsing and dependence analysis).

To summarize, the first functionality of this part of TransTool is to help the user to insert HPF basic components into his "old Fortran" source code (and give defaults values as much as possible): let the user insert, with some assistance, directives and constructions like template, processor, align, distribute, forall and calls to intrinsic procedures.

The main research topic around the semi-automatic insertion of HPF directives is the automatic distribution of matrices using dependence analysis, previous distributions and target machine parameters.

5.3 Dependence Analysis

Dependence analysis is a crucial part of the semi-automatic parallelization of a code. Many tools for dependence analysis have been designed and Petit [25] is one of them. Petit is a version of Michael Wolfe's Tiny tool extended by the Omega Project at the University of Maryland. This

tool uses the Omega library [26] to compute the dependences.

We use Petit via its batch interface to compute the dependences of selected loop nests. The user can choose a loop nest and ask for the dependences. Then, a graphical interface allows the user to see the dependences, to select some dependences according to some criterion (for example, choosing only the flow dependences, ...), to see the sink and target of dependences on the editor.

When a loop nest is chosen, the corresponding sub-program is rebuilt from the Abstract Syntax Tree and transformed into the Petit language using f2p.

Figure 2 shows some of the dependence analysis windows of TransTool.

5.4 Parsing and Unparsing

ADAPTOR (Automatic DAta Parallelism translaTOR) is a public domain compilation system developed at GMD for compiling data parallel HPF programs to equivalent message passing programs [10]. The compiler tools used for ADAPTOR can be retrieved and used to build other tools. In TransTool, we use the front end which is able to parse a Fortran 77 source file, to generate the Abstract Syntax Tree (AST) and to unparse the AST in a output file. We use the AST and the unparsing functionality to build the sub-programs for the dependence analysis. If the source file is an HPF source code, the directives are also in the AST. We will use them for the semi-automatic parallelization.

We also use ADAPTOR for the compilation of the code generated by HPFIT.

5.5 Translation of Calls to Sequential Libraries

Many real applications are currently using sequential libraries like BLAS or LAPACK. These libraries are available on many existing machines. Parallel versions of these libraries are available, like the PBLAS and ScaLAPACK [24]. They are highly optimized, and reach very high efficiencies close to peak performance together with a good scalability. It is impossible to reach the same efficiency using usual compilers. A problem appears when one wants to transform a source code into HPF: these libraries are added during the link phase, and thus their source code is not available for automatic parallelization during the upstream operations. Moreover, even if we have the source of the sequential routine, its automatic parallelization will lead to poor performance. If a parallel version of the library exists, we must use it for the parallelization of the application.

Some work has been done around HPF interfaces of parallel libraries [9, 30]. This is the best way to obtain a portability between HPF compilers and parallel libraries. One of the goals of TransTool is to offer the users the opportunity to automatically translate library calls from sequential to parallel versions (via their HPF interfaces). Furthermore, TransTool will allow the user to insert redistribution directives if it appears to be necessary. It will be a semi-automatic translation linked with the compilation. Moreover, TransTool will allow to insert the source code of subroutines which do not have their parallel implementation. Then the source will be "HPFized" and distribution will be inherited. One problem is that HPF handles many more distributions than those supported by ScaLAPACK. This will imply the development of conversion routines.

5.6 Execution Interface

TransTool has been designed to be a self-contained environment. From the XEmacs editor, the user should be able, taking a sequential Fortran 77 code, to semi-automatically generate a HPF source code, to compile this code and to execute the SPMD program on the different machines he has access to. To this purpose, we have designed an interface to give the parameters of the machines

(how to start a computation on the machine, how you allocate nodes, ...), to compile the HPF code by choosing among available HPF compilers, and to start the program on a remote machine.

5.7 Developer's Toolkit

HPFIT will provide a standard interface to many other tools used in the parallelization of applications like performance monitors, trace analyzers, and simulation tools. In this first version, HPFIT is not a totally new environment built from scratch but an "intelligent" interface into which existing or new tools are being plugged. Thanks to this interfacing, we will be able to add new tools as they appear.

The TransTool Developer's Toolkit (T^3) is the set of interfaces which can be used to build new tools from TransTool, or to integrate new functionalities inside the editor. Currently, the Toolkit has interfaces to:

- the XEmacs editor,
- the parser,
- the dependence analysis tool.

These interfaces are written with either C, TCL or Lisp, so they can be used in a C program, a tcl script or within the XEmacs editor.

The first version (V 1.0) of TransTool and its developer's Toolkit is available on the Web[3].

6 Optimization Kernel

Our main interest in TransTool is to validate recent research results on real applications, and to be able to integrate the corresponding (limited size) software developments within existing, more complete and more powerful tools.

In this section, we present two research topics, i.e. parallelism detection for automatic generation of `INDEPENDENT` directives, and the pipelined loops detection for the automatic generation of calls to the LOCCS library.

6.1 Parallelism Detection in Nested Loops

One of the objectives of the TransTool project is to develop and integrate strategies for automatically (or semi-automatically) transforming sequential Fortran pieces of code into codes with HPF directives. The goal is to help the programmer to recognize parallelism at the loop level, and to automate the corresponding loops transformations for him.

Since our target language is HPF, we have to keep in mind that only transformations that can be expressed in HPF and that can be efficiently compiled by an HPF compiler are suitable. In particular, we do not currently address the following topics: parallelism exploited in doacross loops, software pipelining, minimization of synchronization barriers: the first two topics because such parallelism cannot be easily and efficiently implemented in a data-parallel language like HPF (it is easier to exploit it when *compiling* HPF), the third topic because HPF codes are usually compiled into SPMD codes, synchronized by nature.

[3] URL `http://www.ens-lyon.fr/~desprez/FILES/RESEARCH/SOFT/TransTool/`

Our main goal is to expose to the programmer the maximal parallelism that can be detected. We are interested only in understanding if a large set of independent computations can be detected, and if they can be described by parallel loops. In other words, we aim at detecting loops that, in HPF, can be preceded by the directive `!HPF$ INDEPENDENT` (denoted by DOPAR in the pseudo-code below).

6.1.1 Fine-grain Parallelism

In many applications, there is no need to use sophisticated dependence analysis techniques and parallelization algorithms for detecting full parallelism. A simple algorithm such as Allen and Kennedy's algorithm [2] is sufficient for most of the loops. Our implementation of Allen and Kennedy's algorithm will be used as a comparison base to evaluate how often more sophisticated algorithms are needed. In this context, we recently showed that, as long as dependence level is the only information available, Allen and Kennedy's algorithm detects maximal parallelism (see [16]).

In some loops however, some more accurate representation of dependences is needed. Techniques based on the hyperplane method [29] have been developed in the past so as to exploit a more accurate description of dependences such as the description by direction vectors (see Wolf and Lam's algorithm [34]). This last algorithm is able to take into account the information given on all components of distance vectors (which is not possible with Allen and Kennedy's algorithm and level of dependences), but it is not able to use the information concerning the structure of the dependence graph (which is the basis of Allen and Kennedy's algorithm for applying loop distribution).

We thus proposed a novel algorithm, Darte and Vivien's algorithm, that combines and subsumes both algorithms [17]. We found that this algorithm optimally exploits the structure of the graph and the information on direction vectors. It is even optimal for a more accurate representation of dependences that we called PRDG (polyhedral reduced dependence graph), roughly speaking, approximations of dependences by non parameterized polyhedra, defined by vertices, rays and lines.

6.1.2 Medium-grain Parallelism

In HPF, codes with single innermost parallel loops are often not parallel enough to offer good performance. In this case, the grain of parallelism must be increased, either by trying to move up the parallel loop to the outermost possible level, or by using blocking (tiling) techniques.

We studied this tiling problem in [5] in the simple case of uniform loop nests, and it turns out that Darte and Vivien's algorithm can be easily adapted to the tiling technique, as Wolf and Lam's algorithm that was developed with a "tiling spirit". Actually, the detection of parallel loops and the detection of maximal tiling, related to maximal sets of permutable loops, are two equivalent problems.

We are currently implementing in TransTool, a tiling version of Darte and Vivien's algorithm: it informs the programmer of the maximal parallelism he can hope for, and proposes loop transformations that reveal maximal parallelism, either as fine-grain parallelism, or as medium-grain parallelism.

A lot of problems remain to be solved such as the choice of the block size for tiling, the choice of a suitable mapping (with possibly temporary arrays), ... As the reader can notice, the above algorithms are able to generate `!HPF$ INDEPENDENT` directives, but they do not address the generation of directives such as align or distribute. This is still left to the programmer: he has to choose the mapping that exploits at best the parallelism that has been detected.

We conclude this section by a very simple example, that illustrates the type of codes that can be generated by our parallelism detection algorithm. Figure 3(a) shows the original code, and

```
DO i = 1, n
  DO j = 1, n
    a(i, j) = a(i - 1, j + 1) + b(i - 1, n)
    b(i, j) = a(i, j - 1) + b(i - 1, j - 1)
  ENDDO
ENDDO
```

(a)

```
DO i = 1, n
  b(i, 1) = a(i, 0) + b(i - 1, 0)
  DOPAR j = 2, n
    a(i, j - 1) = a(i - 1, j) + b(i - 1, n)
    b(i, j) = a(i, j - 1) + b(i - 1, j - 1)
  ENDDOPAR
  a(i, n) = a(i - 1, n + 1) + b(i - 1, n)
ENDDO
```

(b)

Figure 3: Original code and code with fine-grain parallelism

Figure 3(b) the code with fine-grain parallelism. Note that Allen and Kennedy's algorithm would also find one parallel loop in this example. However, the fact that loop distribution can be avoided cannot be found by Allen and Kennedy's algorithm.

6.2 Detection of Pipelined Loops and Code Generation

Parallel distributed memory machines improve performance and memory capacity but their use adds an overhead due to the communications. To obtain programs that perform and scale well, this overhead must be minimized. Part of the job is devoted to communication libraries, which should provide efficient point-to-point and macro-communications. Another important issue is to "hide" communication as much as possible, by overlapping them with independent communications.

Asynchronous communications can be used to overlap computations and communications. The call to the communication routine (send or receive) is then issued as soon as possible in the code. A wait routine is used to check for the completion of the communication. Unfortunately, this is not always legal due to the dependences between computations and communications. Pipeline schemes are also sometimes found within the code. These schemes lead to a sequentiality in the execution of the whole algorithm.

The optimization we have added is what we call *Macro-pipeline Overlap*. There is a sequentiality within the code (see Figure 4 (A)). Processor P1 must wait for processor P0 to complete his computation and send the results, to receive the data and start to work. As soon as it has finished, it sends the results to processor P2 which, in turn, starts to work on the received data. The total execution time is higher than the sequential one because of the overhead of the communications. One first solution is to start the communications as soon as possible, i.e. as soon as one processor has computed one data item. For each data item computed, an other one is sent to the following processors so they can start as soon as possible. This is called a fine-grain pipeline ((B) on Figure 4). This solution adds an overhead because of the communication startup time. This time is usually higher than the cost of the communication of one element. Thus, the total time can be higher than the one without pipelining. A trade-off has to be found which minimizes the execution time. This is a coarse grain pipeline ((C) on Figure 4).

A typical example of a code that may benefit from a macro-pipeline optimization is the ADI algorithm given on Figure 5.

We have designed a library for the optimization of pipelined computations called the LOCCS [18] [4]. This library has been integrated in the ADAPTOR compiler [8] and we are currently working on

[4] Low Overhead Communication and Computation Subroutines.

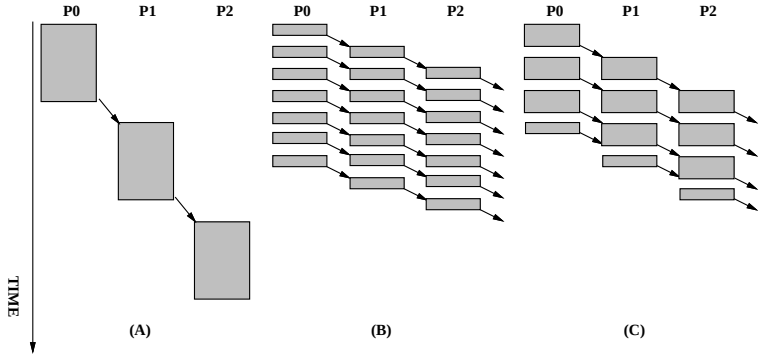

Figure 4: Macro-pipeline.

```
      PARAMETER (N=...)
      REAL, DIMENSION (N,N)  :: A, B
!HPF$ DISTRIBUTE (*,BLOCK) :: A, B
      ...
!     sweep along the columns
      DO I = 2, N
        DO J = 1, N
          A(I,J) = A(I,J) - A(I-1,J)*B(I,J)
        END DO
      END DO
!     sweep along the rows
      DO J = 2, N
        DO I = 1, N
          A(I,J) = A(I,J) - A(I,J-1)*B(I,J)
        END DO
      END DO
```

Figure 5: High Performance Fortran Version of the ADI algorithm.

```
      PARAMETER (N=...)
      REAL, DIMENSION (N,N)  :: A, B
!HPF$ DISTRIBUTE (*,BLOCK) :: A, B
      ...
      DO I = 2, N    ! parallel execution
        DO J = 1, N
          A(I,J) = A(I,J) - A(I-1,J)*B(I,J)
        END DO
      END DO

      CALL DALIB'LOCCS'DRIVER (BLOCK, 2, 0,
          A(:,2:N), [0,1], B(:,2:N), [0,0])
      ...

              EXTRINSIC  (HPF'LOCAL)  SUBROU-
TINE BLOCK (A, B)
      REAL A(:,:), B(:,:)
!HPF$ DISTRIBUTE *(*,BLOCK) :: A, B
      DO J=lbound(A,2),ubound(A,2)
        DO I=lbound(A,1),ubound(A,1)
          A(I,J) = A(I,J) - A(I,J-1)*B(I,J)
        END DO
      END DO
      END
```

Figure 6: ADI algorithm using the LOCCS library.

its integration within TransTool. Within ADAPTOR, the LOCCS library consists in a driver routine whose input parameters provide information about distributed matrices and their distributed dimension(s, and a routine which is called at each step of the macro-pipeline. Within the driver routine, choices are made to use macro-pipelining or not, and also on the way of doing this optimization. There are several reasons to use a library instead of generating the code directly in the SPMD source code. The first one is the ease of use for the programmer of an HPF compiler. Instead of generating several lines of code, the compiler only has to generate a subroutine call, to fill the parameters and to generate the computation routine. Another reason is to be able to perform run-time optimizations like, for example, the dynamic computation of the optimal grain size as a function of the network load, cache effects, and so on.

We have obtained very good results using this library in the ADAPTOR compiler, for example with the ADI algorithm given in Figure 5. There are two strategies to solve this problem, one using a redistribution (transposition) and the other one using our library to have an optimized

pipelined execution. The pipelined execution achieves a near-optimal speed-up and a dynamic data remapping is not necessary in this case.

Now we need to integrate the LOCCS inside TransTool. First we need to find what Tseng called *Cross Processor Loops* (CP loop) in [33]. A loop is a CP loop if it has a true dependence carried by the loop and of course if its iteration crosses the processors boundaries. If a loop is a CP loop, one processor needs the results of the computation of its left or right neighbor to start to work. However, telling that a loop is a CP loop is not sufficient to say that a macro-pipeline execution is efficient. We are working on an algorithm that detects loops that can benefit from a macro-pipeline execution.

For the computation of the optimal granularity of the pipeline, we will use the OPIUM library [19]. The granularity will be also tuned at run-time depending on cache effects or network trafic.

The user will be able to give to TransTool the parameters of the target machine (parameters of a communication, costs of an average computation). These parameters will be used for the optimization of the code generation. For example, when using PVM on a network of workstations connected via Ethernet, no macro-pipeline should be used because of the huge costs of communication startups.

7 Conclusion and Future Work

In this paper, we have presented the first versions of the HPFIT project and of TransTool. HPFIT will provide one interface to many other tools used in the parallelization of applications like performance monitors, traces analyzers, and simulation tools. In this first version, HPFIT is not a totally new environment built from scratch but an "intelligent" interface into which existing or new tools are being plugged. Thanks to this interfacing, we will be able to add new tools as they appear.

Data distribution and alignment is one of the most important problem for the parallelization of applications using HPF. It is known to be a NP-complete problem in most cases; however for classical problems, heuristics can be found that will lead to good performance. Thus we need to add a tool for semi-automatic data distribution (within HPFize). Another problem that has already been raised by Kennedy et al. in [22] is an interaction between the HPF compiler and the editor. This is not a trivial work as the compiler can make huge transformations to obtain an SPMD code with local arrays and calls to communications routines.

Converting dusty F77 to Fortran 90 seems to be a useful intermediate step in the parallelization process. For example, data parallelism could be expressed by array syntax and FORALL loops (Fortran 95). Part of this work is clearly not our job (cleaning F77) but we could add some fonctionnalities to integrate F90 constructs in the HPFize part of TransTool, by taking those loops nests that can be transformed.

A lot of work remains to be done is the field of tools for semi-automatic parallelization of applications. We hope that a collaboration between several laboratories will lead to an interesting and performant tool made available to the whole community.

Acknowledgments

We would like to thank Gilles Lebourgeois, Olivier Reymann, Georges-André Silber, Lionel Tricon and Julien Zory for their work within the TransTool project.

References

[1] V.S. Adve, J.C. Wang, J. Mellor-Crummey, D.A. Reed, M. Anderson, and K. Kennedy. An Integrated Compilation and Performance Analysis Environnement for Data Parallel Programs. Technical Report CRPC-TR94513-S, Center for Research on Parallel Computation, Rice University, December 1994.

[2] J.R. Allen and K. Kennedy. Automatic Translations of Fortran Programs to Vector Form. *ACM Toplas*, 9:491–542, 1987.

[3] P. Amarasinghe, J.M. Anderson, M.S. Lam, and C.-W. Tseng. The SUIF Compiler for Scalable Parallel Machines. In *Seventh SIAM Conference on Parallel Processing for Scientific Computing*, February 1995.

[4] P. Banerjee, J.A. Chandy, M. Gupta, E.W. Hodges-IV, J.G. Holm, A. Lain, D.J. Palermo, S. Ramaswamy, and E. Su. The PARADIGM Compiler for Distributed-Memory Multicomputers. *IEEE Transactions on Computers*, 28(10):37–47, October 1995.

[5] Pierre Boulet, Alain Darte, Tanguy Risset, and Yves Robert. (pen)-Ultimate Tiling? *Integration, the VLSI Journal*, 17:33–51, 1994.

[6] T. Brandes, S. Chaumette, M.-C. Counilh, A. Darte, J.C. Mignot, F. Desprez, and J. Roman. HPFIT: A Set of Integrated Tools for the Parallelization of Applications Using High Performance Fortran: Part I: HPFIT and the TransTOOL Environment. In J.J. Dongarra and B. Tourancheau, editors, *Third Workshop on Environments and Tools for Parallel Scientific Computing*, Faverges, August 1996. SIAM.

[7] T. Brandes, S. Chaumette, M.-C. Counilh, A. Darte, J.C. Mignot, F. Desprez, and J. Roman. HPFIT: A Set of Integrated Tools for the Parallelization of Applications Using High Performance Fortran: Part II: Data Structures Visualization and HPF Extensions for Irregular Problems. In J.J. Dongarra and B. Tourancheau, editors, *Third Workshop on Environments and Tools for Parallel Scientific Computing*, Faverges, August 1996. SIAM.

[8] T. Brandes and F. Desprez. Implementing Pipelined Computation and Communication in an HPF Compiler. In *Europar'96 Parallel Processing*, volume 1123 of *Lecture Notes in Computer Science*, pages 459–462. Springer Verlag, August 1996.

[9] T. Brandes and D. Greco. Realization of an HPF interface to ScaLAPACK with Redistributions. In H. Liddel, A. Colbrook, B. Hertzberger, and P. Sloot, editors, *High-Performance Computing and Networking (HPCN)*, volume 1067 of *Lectures Notes in Computer Science*, pages 834–839. Springer Verlag, 1996.

[10] Th. Brandes and F. Zimmermann. ADAPTOR - A Transformation Tool for HPF Programs. In K.M. Decker and R.M. Rehmann, editors, *Programming Environments for Massively Parallel Distributed Systems*, pages 91–96. Birkhäuser, April 1994.

[11] B.M. Chapman, T. Fahringer, and H.P. Zima. Automatic Support for Data Distribution on Distribution on Distributed Memory Multiprocessor Systems. Technical Report TR 93-2, Institute for Software Technology and Parallel Systems, University of Vienna, Austria, August 1993.

[12] C. Clémançon, A. Endo J. Fritsher, A. Müller, R. Rühl, and B.J.N. Wylie. The "Annai" Environment for Portable Distributed Parallel Programming. In *28th Hawaii International Conference on System Sciences (HICSS-28)*, volume II, pages 242–251. IEEE Computer Society Press, January 1995.

[13] Alain Darte and Yves Robert. Constructive methods for scheduling uniform loop nests. *IEEE Trans. Parallel Distributed Systems*, 5(8):814–822, 1994.

[14] Alain Darte and Yves Robert. Mapping uniform loop nests onto distributed memory architectures. *Parallel Computing*, 20:679–710, 1994.

[15] Alain Darte and Yves Robert. Affine-by-statement scheduling of uniform and affine loop nests over parametric domains. *J. Parallel and Distributed Computing*, 29:43–59, 1995.

[16] Alain Darte and Frédéric Vivien. On the Optimality of Allen and Kennedy's Algorithm for Parallelism Extraction in Nested Loops. In *Proceedings of Europar'96*, Lyon, France, August 1996. Springer Verlag. To appear.

[17] Alain Darte and Frédéric Vivien. Optimal Fine and Medium Grain Parallelism in Polyhedral Reduced Dependence Graphs. In *Proceedings of PACT'96*, Boston, MA, October 1996. IEEE Computer Society Press. To appear.

[18] F. Desprez. A Library for Coarse Grain Macro-Pipelining in Distributed Memory Architectures. In *IFIP 10.3 Conference on Programming Environments for Massively Parallel Distributed Systems*, pages 365–371. Birkhaeuser Verlag AG, Basel, Switzerland, 1994.

[19] F. Desprez, P. Ramet, and J. Roman. Optimal Grain Size Computation for Pipelined Algorithms. In *Europar'96 Parallel Processing*, volume 1123 of *Lecture Notes in Computer Science*, pages 165–172. Springer Verlag, August 1996.

[20] Michèle Dion, Cyril Randriamaro, and Yves Robert. How to optimize residual communications ? In *10th International Parallel Processing Symposium*, pages 382–391. IEEE Computer Society Press, 1996. Extended version available as ENS Lyon RR 95-27, October 1995. To appear in JPDC.

[21] Ian Foster. *Designing and building parallel programs.* Addison-Wesley, 1995.

[22] S. Hiranandani, K. Kennedy, C.-W. Tseng, and S. Warren. Design and Implementation of the D Editor. In J.J. Dongarra and B. Tourancheau, editors, *Second Workshop on Environments and Tools for Parallel and Scientific Computing*, pages 1–10, Townsend, TN, May 1994. SIAM.

[23] C.S. Ierotheou, S.P. Johnson, M. Cross, and P.F. Leggett. Computer Aided Parallelization Tools (CAPTools) - Conceptual Overview and Performance on the Parallelization of Structured Mesh Codes. *Parallel Computing*, 22:163–195, 1996.

[24] J.Choi, J. Demmel, I. Dhillon, J. Dongarra, S. Ostrouchov, A. Petitet, K. Stanley, D. Walker, and R.C. Whaley. LAPACK Working Note: ScaLAPACK: A Portable Linear Algebra Library for Distributed Memory Computers - Design Issues and Performances. Technical Report 95, The Universtity of Tennessee - Knoxville, 1995.

[25] W. Kelly, V. Maslov, W. Pugh, E. Rosser, T. Shpeisman, and D. Wonnacott. *New User Interface for Petit and Other Extensions.* CS Dept, University of Maryland, April 1996. http://www.cs.umd.edu/projects/omega.

[26] W. Kelly, V. Maslov, W. Pugh, E. Rosser, T. Shpeisman, and D. Wonnacott. *The Omega Library - Version 1.0 - Interface Guide.* CS Dept, University of Maryland, April 1996. http://www.cs.umd.edu/projects/omega.

[27] K. Kennedy, K.S. McKinley, and C.-W. Tseng. Interactive Parallel Programming Using the ParaScope Editor. *IEEE Transactions on Parallel and Distributed Systems*, 2(3):329–341, July 1991.

[28] Charles H. Koelbel, David B. Loveman, Robert S. Schreiber, Guy L. Steele Jr., and Mary E. Zosel. *The High Performance Fortran Handbook.* The MIT Press, 1994.

[29] Leslie Lamport. The Parallel Execution of DO Loops. *Communications of the ACM*, 17(2):83–93, February 1974.

[30] P.A.R. Lorenzo, A. Muller, Y. Murakami, and B.J.N. Wylie. High Performance Fortran Interfacing to ScaLAPACK. Technical Report TR-96-13, Swiss Center for Scientific Computing (CSCS), Manno, Switzerland, 1996.

[31] J.-L. Pazat. Tools for High Performance Fortran: A Survey. Technical report, IRISA, Rennes, France, 1996. http://www.irisa.fr/pampa/HPF/survey.html.

[32] J.K. Peir and R. Cytron. Minimum distance: a method for partitioning recurrences for multiprocessors. *IEEE Transactions on Computers*, 38(8):1203–1211, August 1989.

[33] Chau-Wen Tseng. *An Optimizing Fortran D Compiler for MIMD Distributed-Memory Machines.* PhD thesis, Rice University, 1993.

[34] M.E. Wolf and M.S. Lam. A Loop Transformation Theory and an Algorithm to Maximize Parallelism. *IEEE Transactions on Parallel and Distributed Systems*, 2(4):452–471, 1991.

[35] H. Zima and B. Chapman. Compiling for Distributed Memory Systems. Technical report, Institute for Statitics and Computer Science, Vienna, Austria, May 1994.