1 Adaptor: A Compilation System for Data Parallel Fortran Programs

Thomas Brandes
German National Center for Computer Science (GMD)
St. Augustin, Germany
email: brandes@gmd.de

Abstract: Data parallel programming stands for single threaded, global name space, and loosely synchronous parallel computation. This kind of parallel programming has been proven to be very user-friendly, easy to debug and easy to use. But this programming model is not available for most message passing multiprocessor architectures.

Adaptor (Automatic Data Parallelism Translator) is a compilation system that transforms data parallel programs written in Fortran with array extensions, parallel loops, and layout directives to parallel programs with explicit message passing. The current version supports especially the translation of Connection Machine Fortran and High Performance Fortran programs to message passing programs.

After a short description of the system it will be shown how efficient the compilation is. Therefore many results about speed-ups and efficiencies of real codes are presented. Furthermore, the automatically generated message passing programs are compared with hand written message passing programs. The results will show that data parallel programs will be competitive to hand written message passing programs if the data parallelism in the program can be recognized and utilized by the compiler.

1.1 Introduction

MIMD (multiple instruction, multiple data) architectures with distributed memory are the kind of parallel machines that are scalable and can be used for a wide range of scientific applications. Usually, these architectures are programmed with explicit message passing between the processes running on the different processors. As the message passing programming model is very error prune and difficult to use, many efforts have been made to offer other programming models that are easier to use.

These difficulties are not given when using the data parallel programming model. This model stands for single threaded, global name space, and loosely synchronous parallel computation.

Language extensions and modifications for Fortran 90 have been defined by the High Performance Fortran Forum [14] to take advantage of data parallelism. This language High Performance Fortran (HPF) allows code tuning for various architectures and should guarantee top performance on MIMD and SIMD (single instruction, multiple data) computers with non-uniform memory access costs. Many large scientific applications are expected to be programmed in this data parallel language.

The Adaptor system (Automatic Data Parallelism Translator) makes it possible to translate these programs to message passing programs already now. It transforms data parallel programs written in Fortran 77 with array extensions, parallel loops, and layout directives to parallel programs with explicit message passing.

Therefore the code with global data references together with a user specified or implicitly defined data distribution is translated into a program with local and non-local references, where the latter are satisfied by automatically inserting message-passing statements.

Experiments with many sequential programs and their Fortran 90 counterparts have shown [6] that automatic methods could not parallelize the sequential version where it is possible for the version with explicit array operations. In Adaptor only the inherent parallelism of the array operations and of parallel loops is used. Local array operations will be distributed among the available nodes, for non-local array operations efficient communication is generated as these operations have mostly regular communication patterns (e.g. global reductions, shift and spread operations).

In the following the Adaptor system is described. Results of benchmark and real application codes are presented and useful optimization issues will be discussed.

1.2 The Adaptor Compilation System

The Adaptor system transforms Fortran 77 or Fortran 90 programs with explicit data parallelism into parallel programs for MIMD architectures with explicit message passing. Together with a run time system these programs can run on most available parallel architectures. The current version has been designed especially to translate data parallel Connection Machine Fortran (CMF) [4] programs to message passing programs, but it supports also features of High Performance Fortran (HPF) [14].

1.2.1 Properties of Adaptor

The central idea of an automatic translation is to distribute the large data structures like arrays among the available processors. This should be done in such a way that most operations can be done locally without any need of communication. Where global operations are necessary the corresponding message passing statements are inserted automatically.

Though the user will need to understand some issues of parallelism and has to know for efficiency reasons where message passing will be generated, the effectiveness of Adaptor is based on the fact that the user has not to know any message passing command and not to manage the control of the data partitioning. He can change types of variables (e.g. single to double precision) and data distributions without rewriting any other statement in his program. He has not to write two versions of code (host and node program) and many global array operations are translated to the most efficient code for the underlying architecture.

The parallel program can be written in such a way that it can be developed on a serial machine and is also suitable for vector machines or parallel machines with shared memory. Many features supported by Adaptor result also in good execution times for these architectures. By this way, it helps to design programs that run efficiently on nearly all architectures.

Adaptor takes only advantage of the parallelism in the array operations and of the parallel loops. It has no features for automatic parallelization.

The generated code of Adaptor should be as efficient as possible and competitive to a hand-coded Fortran program with message passing. Otherwise the acceptance of such a tool cannot be expected.

Adaptor supports the development of parallel codes that scale with the number of processors. No support is given for any kind of programming where the number of processors is fixed in any way. It makes heavy use of dynamic arrays and the executable version of the generated program can run for any number of processors without any recompilation.

1.2.2 Overview of Adaptor

Figure 1.1 gives an overview of the Adaptor compilation system. The essential parts are the interactive source-to-source transformation (fadapt) and the run-time system DALIB (distributed array library). For compiling and linking of the generated programs, the available Fortran compiler of the parallel machines is utilized.

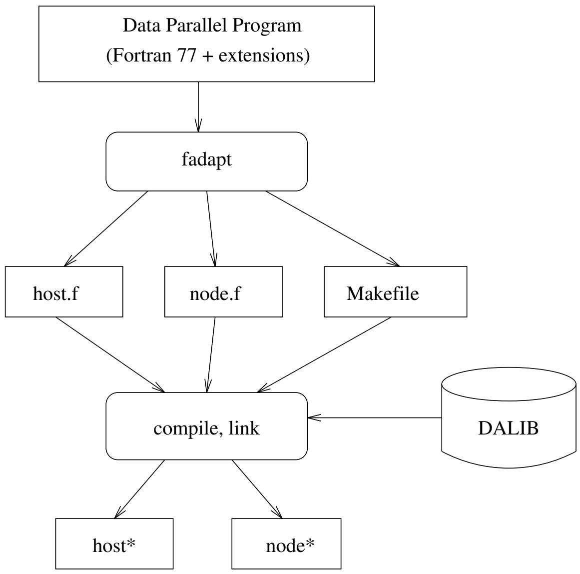

Figure 1.1 Overview of Adaptor

1.2.3 The Input Language

The input language of Adaptor can be defined as Fortran 77 with some restrictions, but with many extensions like dynamic arrays, array operations, parallel loops and layout directives of CMF, HPF and Fortran 90 [1].

The user can define host arrays, replicated arrays and distributed arrays. For the specification of data layouts in Adaptor similar directives as in CMF or HPF are used. Also the parallel `FORALL` statement supported by Adaptor has the same syntax and semantic as proposed in these data parallel languages.

Many features of Fortran 90 and HPF cannot be used with Adaptor. The most serious restrictions are that Adaptor supports no modules, no pointers, no array-valued functions and no assumed-shaped arrays.

In contrary to many other systems [18, 21, 7, 9] Adaptor supports only block distributions along one dimension. More distributions will be supported in future releases.

In CMF and HPF explicit alignment can be used to reduce communication [16] especially for a given program. For Adaptor this feature is not supported until now, but of course there is an implicit alignment of arrays that are declared and distributed in the same way.

1.2.4 Programming Models for the Generated Programs

The user can select between the following three programming models:

- If the *HOST-NODE* programming model is selected, Adaptor will generate a host program and a node program. The node program runs on all available nodes of the parallel machine, while the host program contains all I/O operations that will be executed on the front end system.

- In the *ONLY-NODE* programming model only one program will be generated that runs on all available nodes. There is no host program. The first node takes care of all I/O operations.

- A program that runs only on a single node is generated when using the *UNIPROC* programming model. The program has no communication and therefore it ought to be faster than the previous one running on a single node. By choosing this model programs with array operations that are not available in Fortran 77 can be translated to sequential Fortran 77 programs.

1.2.5 Interactive Source-to-Source Transformation

The translation of the input file can be done as a batch job, but an interactive translation is also possible. A graphical environment allows the user to select units of the source program (program, functions, subroutines) or variables in a unit to get information about them (see figure 1.2).

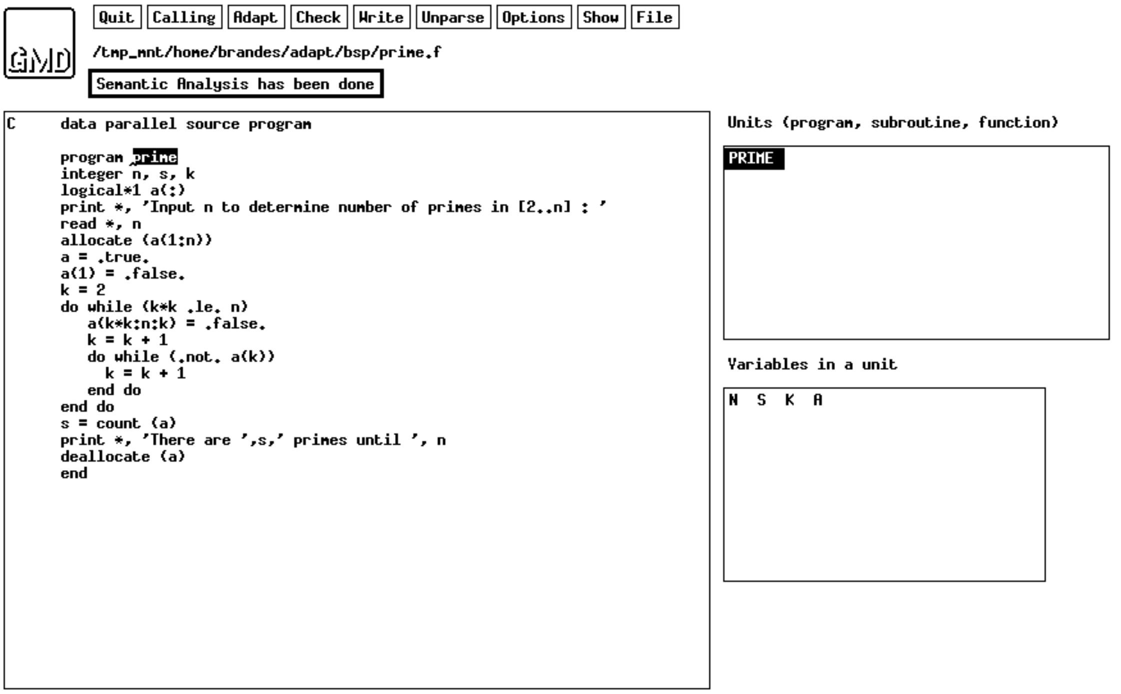

Figure 1.2 fadapt: Interactive Source-to-Source Transformation

1.2.6 Realization of the Translation

The following steps are done during the source to source transformation of Adaptor:

1. The source program is parsed and an abstract syntax tree will be generated.
2. Symbol tables are created and used for a semantic analysis.
3. The program (abstract syntax tree) is normalized to reduce the complexity of the translation.
4. The real translation on the internal abstract syntax tree and symbol tables has four phases:

   (a) In the *analysis phase* the code is checked to verify that there are no violations of the current restrictions.

   (b) In many cases array assignments need communication. In this case Adaptor tries to split up the assignment in primitive array assignments with communication and local array assignments. Sometimes new temporary arrays have to be created.

   (c) In the *initial transformation phase* local array operations will be translated to parallel loops, the forall statement will be translated to equivalent do loops. After this phase only parallel loops without communication will exist.

   (d) In the *final translation* loops will be restricted to the part of the arrays that is owned by one processor and communication statements or movements are translated to corresponding subroutine calls of the underlying run time system.

5. The new internal abstract syntax tree is unparsed back to source text.

Except the graphical interface, the whole source-to-source transformation of Adaptor is generated with a toolbox for compiler construction [11]. These tools have a great flexibility and can generate very efficient code. For the intermediate language, abstract syntax trees will be used where the program module that defines the structure of the abstract syntax trees and provides general tree manipulating procedures is also generated by a tool.

For the analysis and transformation components the compiler tool *Puma* is utilized [10]. This tool cooperates with the generator for abstract syntax trees and supports the transformation and attribution of attributed trees. It is based on pattern-matching, unification and recursion. The flexibility of this tool allows not only to have a modular design but also to extend it in a way as one would expect from a knowledge-based system [2].

1.2.7 Distributed Array Library

For the realization of the communication needed for global operations on distributed arrays, many library functions will be used that build the DALIB (distributed array library). This library contains

- low level communication (send, receive, wait, ...),
- high level communication (broadcast, reduction, barrier, ...),
- data movements based on regular and irregular communication patterns,
- timing functions and tracing facilities,
- and a parallel random number generator.

The DALIB, which can be considered as the runtime system of the whole compilation system, is implemented in C. Most part of this library is portable between the different parallel machines. Only the low level message passing commands, the timing functions and the random number generator have to be adapted to the hardware architecture.

Though the realization of the high level communication routines is based on the low level routines, these functions should be tuned for the underlying hardware architecture. As e.g. the CM-5 [5] has an own control network, broadcasts and reductions are more efficient when using this network than using the data network by message passing.

One version of the DALIB is implemented upon the public domain software PVM [20]. It guarantees the portability of the generated parallel programs to all machines where PVM is available. Another version exists for shared memory and virtual shared memory systems where the message passing is realized via a shared memory segment.

At the moment the DALIB has been implemented and tested for iPSC/860, net of SUN or IBM workstations, Alliant FX/2800, Parsytec GCel, CM-5, KSR 1 and SGI multiprocessor machines. Versions for the Intel Paragon and IBM SP1 are in development.

1.2.8 Visualization of the Run Time Behavior

If the generated parallel program is started with the trace flag switched on, a tracefile will be generated that gives information about the behavior of the parallel program. The information of the tracefile can be visualized and animated with the public domain software ParaGraph [13]. Especially the information about the utilization and communication can be used for further optimizations.

1.2.9 Availability

The source files of Adaptor, documentation files in PostScript and a number of example programs are available via 'anonymous ftp' from:

```
ftp.gmd.de        (129.26.8.90)
in subdirectory  gmd/adaptor
```

The latest version of June 1993 has more functionality, stability and more supported features than the first version that has been released in September 1992.

1.2.10 Related Work

Many other systems have been developed during the last time that also support SIMD or data parallel programming for MIMD architectures. A SIMD program is translated into an equivalent SPMD program (single program, multiple data stream). This has been done for C* [12] or for Fortran 90 with additional layout directives [18, 22]. Though the latter systems are very similar to Adaptor there is no information about the efficiency of the generated message passing programs and about their availability.

Due to the introduction of High Performance Fortran, many compilers will be available in the next future and compiler optimizations are goals of some other projects [15].

Further developments have been made to support data parallel programming in an object-oriented language like C++ [3, 17]. This approach has the great advantage that no additional preprocessor or compiler is necessary. But due to the lack of efficiency there is not a great acceptance for scientific applications until now.

1.3 Results of Benchmark Codes

For testing the Adaptor system the High Performance Fortran Benchmark Suite has been utilized [19] where many data parallel programs are given in different versions. The Purdue set (J.R. Rice set) with 14 simple data parallel problems has been used to test the efficiency of the generated message passing programs.

1.3.1 The Purdue Set

In the Purdue Set many different kernels of data parallel algorithms have been put together. Table 1.1 shows what kind of algorithms on which problem sizes have been tested.

no	short description of the problem	problem size
1	Trapezoidal rule	1048576
2	reduction function 1	1024 x 1024
3	reduction function 2	1024 x 1024
4	reduction function 3	524288
5	simple search	128 x 4096
6	tridiagonal set of lin. equations	65536
7	Lagrange interpolation	10 x 32768
8	divided differences	65536 x 8
9	finite differences	512 x 512
11	Fourier's moments	262144
12	array's construction	1023 x 511
13	floating point arithmetic	262144
14	Simpson's and Gauss' integration	262144
15	Chebyshev interpolation	16384

Table 1.1 Problems of the Purdue Set

Three different versions of the programs have been considered:

- the Fortran 77 version can be translated for one node and is used to measure real speed ups,
- the CMF version gives results for the Connection Machine and is used for Adaptor with slightly changes to get automatically generated message passing programs for different parallel machines,
- the parallel version, Fortran 77 with explicit message passing based on the message passing library PICL, is used to compare the results of Adaptor with a hand-coded message passing program.

1.3.2 Comparison of Sequential and Parallel Version

Table 1.2 shows the results of the sequential Fortran 77 version running on one node of the iPSC/860 compared with the generated message passing program of the hostless programming model. The parallel program is only running on one node.

The results show that the Adaptor version is faster than the sequential Fortran 77 counterpart for the problems 2, 3, 4, 12, 14 and 15. The following reasons are responsible for this effect:

no	sequential version	parallel version	ratio
1	284.7 s	290.6 s	0.98
2	45.9 s	33.8 s	1.36
3	47.0 s	10.3 s	4.56
4	222.1 s	185.5 s	1.20
5	175.9 s	209.3 s	0.84
6	414.7 s	1008.9 s	0.41
7	131.1 s	143.4 s	0.91
8	104.8 s	186.5 s	0.56
9	46.2 s	46.2 s	1.00
11	289.7 s	289.1 s	1.00
12	182.6 s	21.9 s	8.34
13	588.4 s	555.9 s	1.06
14	35.5 s	25.2 s	1.41
15	133.2 s	92.0 s	1.45

Table 1.2 Comparison of sequential and parallel version on one node

- In problem 2, 3 and 12 Adaptor generates a different loop nesting (innermost loop is always for the first index that results in stride 1 for the loop iterations). After loop interchanging and loop distribution the sequential version was as fast as the Adaptor version.
- In problem 4 only one loop fusion makes the sequential version as fast the generated parallel one.
- For problems 14 and 15 the Adaptor version has a vector version of the function that is integrated. By this way there is only one subroutine call instead of a call for every point.

But usually the Adaptor version is a little bit slower due to the generated communication and due to additional memory movements. Especially the problems 6 and 8 require much communication.

1.3.3 Efficiency and Scalability

Table 1.3 shows the speed ups and efficiencies for the generated message passing programs on the iPSC/860 for 8, 16 and 32 nodes. On other parallel machines similar results were achieved. Also on a net of workstations there were reasonable speed-ups for large problem sizes.

These results verify that the scalability of the data parallel programs results also in scalability of the automatically generated message passing programs.

no	8 nodes	16 nodes	32 nodes
1	7.96 (99.5 %)	15.79 (98.7 %)	31.25 (97.6 %)
2	7.41 (92.6 %)	14.08 (88.0 %)	28.17 (88.0 %)
3	6.44 (80.5 %)	12.88 (80.5 %)	25.75 (80.5 %)
4	6.08 (76.0 %)	13.16 (82.2 %)	26.13 (81.6 %)
5	7.12 (89.0 %)	14.14 (88.4 %)	27.91 (87.2 %)
6	4.86 (60.7 %)	8.75 (54.7 %)	16.17 (50.5 %)
7	7.97 (99.6 %)	15.76 (98.5 %)	31.17 (97.4 %)
8	7.61 (95.2 %)	13.92 (87.0 %)	25.90 (80.9 %)
9	7.97 (99.6 %)	15.40 (96.3 %)	28.88 (90.2 %)
11	7.94 (99.3 %)	14.90 (93.1 %)	29.20 (91.3 %)
12	6.84 (85.5 %)	10.95 (68.4 %)	16.85 (52.6 %)
13	7.91 (98.9 %)	15.80 (98.7 %)	31.71 (99.0 %)
14	7.20 (90.0 %)	14.00 (87.5 %)	25.20 (78.8 %)
15	7.48 (93.5 %)	14.84 (92.7 %)	28.75 (89.8 %)

Table 1.3 Speed-Up and Efficiency of Adaptor generated parallel programs

1.3.4 Adaptor vs. hand-coded message passing programs

The parallel programs based on PICL stand for portable hand-coded message passing programs. The most interesting results came up when comparing these programs with the automatically generated message passing programs of Adaptor.

Both versions are portable parallel programs. Both versions are able to run on different number of processors. The hand-coded version realizes this by having the whole data structure replicated on all arrays but every node works only on a subset of the arrays. This has the disadvantage that for bigger problems the code has to be rewritten completely. The Adaptor version can also be used for bigger problems with the only restriction that the program will not run on smaller machine sizes.

- A hand-coded message passing program of problem 6 was not available.
- For problem 2, 3, 4, 14 Adaptor was much more faster,
- for all other problems a little bit faster or nearly the same (problems 5, 8, 11, 12, 13).

As a hand-coded program should always be faster than an automatically generated message passing program, the results show in any case that Adaptor can generate more efficient programs than just straightforward hand written message passing programs. In section 1.4.3 more reliable results on a real application code are given.

1.3.5 Full vs. Loosely Synchronous Execution

The Connection Machine CM-5 [5] of Thinking Machines Corporation offers the data parallel programming model by using CMF and the message passing model by using Fortran 77 and the CMMD message passing library.

no	PICL: hand-coded	Adaptor: automatic
1	25.4 s	18.4 s
2	5.2 s	2.4 s
3	7.1 s	0.8 s
4	24.3 s	14.1 s
7	9.0 s	9.1 s
9	5.3 s	3.0 s
14	11.6 s	1.8 s
15	14.6 s	6.2 s

Table 1.4 Comparison with hand-coded message-passing programs

The 14 CMF applications have been compiled with the CM Fortran compiler (SIMD model). The runtimes have been compared with the runtimes of the by Adaptor automatically generated message passing programs (MIMD model). The vector units of the CM-5 nodes have not been utilized.

Table 1.5 shows the results on a CM-5 with 64 nodes. These results are preliminary as an early version of the CMF Compiler has been used and the programs have been slightly modified for the translation with Adaptor. But it shows that the execution of data parallel programs in MIMD mode is more favorable than the execution in SIMD mode, which requires more synchronization.

no	SIMD execution	MIMD execution
1	16.8 s	8.8 s
2	6.2 s	1.0 s
3	14.6 s	0.6 s
4	25.8 s	6.3 s
5	43.4 s	19.2 s
6	98.7 s	n.a.
7	24.8 s	2.7 s
8	22.8 s	10.1 s
9	8.4 s	1.7 s
11	34.2 s	15.9 s
12	28.1 s	4.8 s
13	56.0 s	21.5 s
14	4.9 s	1.8 s
15	14.1 s	3.5 s

Table 1.5 SIMD and MIMD execution on a CM-5 (64 nodes, no vector units)

If the vector units are used, both versions will be faster. But the advantages of MIMD are lost as every single node works in SIMD mode.

1.4 Results of Application Codes

1.4.1 HYDFLO: a CM Fortran Code for Fluid Dynamics

HYDFLO is a three-dimensional compressible hydrodynamics code based on an explicit two-step Lax-Wendroff finite difference scheme on a regular Eulerian mesh in a regular domain. Such a scheme operates on a compact stencil and is therefore particularly simple for parallel computing.

The given code has run efficiently on a CM and could successfully be translated by Adaptor. In general case it can be assumed that data parallel Fortran programs that run efficiently on SIMD architectures like CM or MasPar should run efficiently on MIMD architectures.

1.4.2 ESM: a Fortran 90 Code for Circulation

The Earth System Model (ESM) is part of a three-dimensional atmospheric general circulation model. This code is a finite difference approximation to the equations of hydrodynamics following the potential enstrophy conserving methods of Arakawa (UCLA). It is based on a grid-point based scheme rather than the more standard spectral decomposition schemes and uses a hydrostatic approximation.

Though the Fortran 90 version of ESM can run on MIMD architectures after using Adaptor, there have been no speed-ups. This is due to the fact that the data decomposition was not conform to the parallel loops in the code.

This effect can be explained with the following example. The code is very inefficient as the parallelism is given for the first dimension of the array, but the distribution is along the second one.

```
      real A(N,N), B(N)
!hpf$ distribute A(*,BLOCK)
!hpf$ distribute B(BLOCK)
      ....
      do J = 1, N
         A(1:N,J) = B(J) * A(1:N,J)
      end do
```

Acceptable results for the ESM code could be achieved after introducing better parallel loops.

1.4.3 IFS: a Fortran 77 Code for Weather Prediction

In a cooperation between GMD and ECMWF (European Center for Medium-Range Weather Forecasts) it is intended to parallelize the ECMWF's production code for medium-range weather forecasts, the IFS (Integrated Forecasting System). For a evaluation of Adaptor a sequential program of the 2D model of the IFS has been investigated. This version contains already all relevant data structures and algorithmic components of the corresponding 3D model.

Results on a CM-5 (without VU)	1 node	4 nodes	16 nodes	32 nodes
hand-written message passing	71.3 s	24.9 s	7.8 s	4.8 s
Adaptor generated version	64.5 s	34.0 s	4.6 s	2.6 s

Table 1.6 Comparison of hand-written and automatically generated MP programs

The same distribution and parallelization strategy has been used as proposed in a parallelization by hand [8]. Though the data structures of the code must be rearranged for Adaptor and the runtime is higher, the effort for parallelization was much less. Table 1.6 shows the runtimes on a CM-5 for a problem with 63 wave numbers, the vector units are not used.

The experiences with the IFS code have shown that Adaptor can also deal with coarse-grained parallelism from parallel loops and not only with fine-grained parallelism from array-operations. The potential of fine-grained parallelism was very low in the given code.

1.5 Summary

Adaptor is a prototype of a compilation system for High Performance Fortran that has given very useful insights how far it is possible to translate efficient data parallel programs to efficient message passage programs for MIMD architectures and in which situations hand-written message passing programs can be faster.

With these insights Adaptor itself can and will be used to implement different optimization strategies for High Performance Fortran compilers. Adaptor will also be used to define and test language extensions of interest that could be part in one of the next versions of High Performance Fortran. Especially features that deal with sparse matrices and parallel I/O are of great interest.

Acknowledgements

I thank the Central Institute for Applied Mathematics at the research center in Jülich for providing the iPSC/860 and Renate Knecht for her user support.

The following people have influenced this work by valuable discussions: James Cownie (Meiko, Bristol), Clemens-August Thole (GMD), Rolf Hänisch (GMD), Michael Gerndt (ZAM, Research Center Jülich), John Merlin (University of Southampton), and Dave Watson (NA Software, Liverpool). Many improvements of the current release have been proposed by the Adaptor users.

Many thanks are also due to Falk Zimmermann for his implementation work and for the discussions about proving correctness of the transformations realized within Adaptor.

Bibliography

[1] T. Brandes. ADAPTOR Language Reference Manual (Version 1.0). Internal Report ADAPTOR-3, GMD, June 1993.

[2] T. Brandes and M. Sommer. Realization of a Knowledge-Based Parallelization Tool in a Programming Environment. In *International Conference on Supercomputing*, Athens, Greece, June 1987.

[3] C. Chase, A. Cheung, A. Reeves, and M. Smith. Paragon: A Parallel Programming Environment for Scientific Applications Using Communications Structures. In *Proc. of 1991 International Conference on Parallel Processing*, St. Charles, Illinois, August 1991.

[4] Thinking Machines Corporation. Connection Machine Model CM-2. Technical Summary Version 6.0, TMC, November 1990.

[5] Thinking Machines Corporation. Connection Machine Model CM-5. Technical summary, TMC, November 1992.

[6] G. Fox. Achievements and prospects for parallel computing. *Concurrency: Practice and Experience*, 3(6):725–739, December 1991.

[7] G. Fox, S. Hiranandani, K. Kennedy, C. Koelbel, U. Kremer, C. Tseng, and M. Wu. Fortran D language specification. Technical Report TR90079, Department of Computer Science, Rice University, April 1991.

[8] U. Gärtel, W. Joppich, and A. Schüller. Parallelizing the ECMWF's Weather Forecast Program: The 2D Case, Technical Documentation and Results for the IFS-2D Model. Arbeitspapiere der GMD 740, Gesellschaft für Mathematik und Datenverarbeitung mbH, March 1993.

[9] H.M. Gerndt. *Automatic Parallelization for Distributed-Memory Multiprocessing Systems*. PhD thesis, University of Bonn, 1989.

[10] J. Grosch. Puma - A Generator for the Transformation of Attributed Trees. Compiler Generation Report 26, GMD, Forschungsstelle an der Universität Karlsruhe, 1991.

[11] J. Grosch and H. Emmelmann. A Tool Box for Compiler Construction. *Lecture Notes of Computer Science*, 477:106–116, October 1990.

[12] P. Hatcher, A. Lapadula, R. Jones, M. Quinn, and R. Anderson. A production quality C* compiler for hypercube machines. In *3rd ACM SIGPLAN Symposium on Principles Practice of Parallel Programming*, pages 73–82, April 1991.

[13] M. Heath and J. Etheridge. Visualizing the Performance of Parallel Programs. *IEEE Software*, pages 29–39, September 1991.

[14] High Perforamnce Fortran Forum. High Performance Fortran Language Specification. Final Version 1.0, Department of Computer Science, Rice University, May 1993.

[15] S. Hiranandani, K. Kennedy, and C.-W. Tseng. Compiler Optimizations for Fortran D on MIMD Distributed-Memory Machines. Technical report, Department of Computer Science, Rice University, 1991.

[16] K. Knobe, J. Lukas, and G. Steele. Data Optimization: Allocation of Arrays to Reduce Communication on SIMD Machines. *Journal of Parallel and Distributed Computing*, 8:102–118, 1991.

[17] M. Lemke and D. Quinlan. P++, a C++ Virtual Shared Grids Based Programming Environment for Architecture-Independent Development of Structured Grid Applications. In Springer Verlag, editor, *Lecture Notes in Computer Science, No. 634*, CONPAR/VAPP V, Lyon, France, September 1992.

[18] J. Merlin. ADAPTing Fortran 90 Array Programs for Distributed Memory Architectures. In *Proc. 1st International Conference of the Austrian Center for Parallel Computation*, Salzburg, October 1991.

[19] A. Mohamed, G. Fox, G. Laszewski, M. Parashar, T. Haupt, K. Mills, Y. Lu, N. Lin, and N. Yeh. Applications Benchmark Set for Fortran-D and High Performance Fortran. Technical Report 327, Northeast Parallel Architectures Center, 1992.

[20] V. Sunderam. PVM: a Framework for Parallel Distributed Computing. *Concurrency: Practice and Experience*, 3(10), December 1990.

[21] Thinking Machines Corporation. CM Fortran Programming Guide, Version 1.0. Manual, TMC, January 1991.

[22] M. Wu and G. Fox. Compiling Fortran 90 programs for distributed memory MIMD parallel computers. Technical Report No. SCCS-88, Syracuse Center for Computational Science, 1991.