Evaluation of High Performance Fortran for an Industrial Computational Fluid Dynamics Code

Thomas Brandes[1], Falk Zimmermann[1], Christian Borel[2], Marc Brédif[2]

[1] Institute for Algorithms and Scientific Computing (SCAI)
German National Research Center for Information Technology (GMD)
Schloß Birlinghoven, D-53754 St. Augustin, Germany
{Thomas.Brandes, Falk.Zimmermann}@gmd.de
[2] Service Aerodynamique Numerique PV21
MATRA BAe Dynamics France, CFD Group
37, avenue Lois Bréguet BP 1,
F-78146 Vélizy-Villacoublay Cedex, France
{cborel, mbredif}@matra-def.fr

Abstract. The PHAROS project, funded by the European Unions ESPRIT program for research and development in information technology, aimed to assess High Performance Fortran (HPF) as a paradigm for porting large FORTRAN 77 scientific applications to distributed memory architectures, in comparison to message-passing programming. The AEROLOG computational fluid dynamic software developed by MATRA was one of these applications that has been ported to HPF. It is devoted to the study of compressible fluid flows around complex geometries. This paper describes the port of the AEROLOG code to HPF based on the decomposition of subdomains. It outlines the parallelization strategy, the changes of the data structures and the tuning of the boundary conditions for the subdomains. Performance results for industrial test cases with different HPF compilers are given and compared with the results of the message-passing version.

1. Introduction

High Performance Fortran (HPF) [7,8] is a data parallel, high level programming language for parallel computing that is expected to be more convenient in terms of portability and maintainability than explicit message passing and to allow higher productivity in software development. But the porting of key commercial applications to HPF is still of critical importance for the continuing development and acceptance of HPF as a standard and for the improvement of HPF compilers.

The ESPRIT project “Open HPF Programming Environments” (PHAROS) aimed to assess HPF as a paradigm for porting large FORTRAN 77 scientific applications to shared and distributed memory architectures, in comparison to message-passing programming. The PHAROS project was funded by the European Union’s ESPRIT program for research and development in information technology. It was a two years project, running from January 1996 until December 1997.

To this end, four major commercial FORTRAN 77 application codes have been successfully ported to HPF (structural analysis, CFD and electromagnetism application codes). These codes already had message-passing parallel versions. The comparison of HPF to message-passing considered factors such as:

- the porting effort;
- the performance of the resulting code;
- the portability and maintainability of the resulting code.

One of the PHAROS applications was the AEROLOG computational fluid dynamics software of MATRA BAe Dynamics [1,5]. The AEROLOG code is a proprietary CFD software devoted to the study of compressible fluid flows around complex geometries. For many years, it has been systematically applied during the aeronautical development programs MISTRAL, MICA, and APACHE, reducing the experimental studies and consequently cutting down costs and delays.

Together with HPF experts and HPF tool providers, the version AEROLOG-v3.2e (e for Euler inviscid fluid model) has been ported to HPF. It is an industry relevant subset of the latest release AEROLOG-v3.2 which includes all the functionalities used in today’s applications. This full release AEROLOG-v3.2 is composed of 102 subroutines with about 19000 lines of source code written in standard FORTRAN 77. The reduced code AEROLOG V3.2e is composed of 55 subroutines with about 11500 lines of source code. It is equivalent to the content of the message passing version of the AEROLOG code.

In accordance to the workplan of the PHAROS project, the rest of this paper is organized as follows. Section 2 describes the AEROLOG software and section 3 outlines the initial port to HPF. The code review in section 4 identified the problems of the initial version and resulted in the code tuning presented in section 5. We discuss our expectations for the next generation of HPF compilers in section 6 to overcome the still existing problems. Finally, we compare in section 7 the results of the HPF versions with the MPI version and conclude in section 8.

2. Description of the AEROLOG Software and the Test Cases

2.1 The AEROLOG CFD Code

The AEROLOG-v3.2 code allows the simulation of steady or unsteady inviscid and compressible fluid flows over three-dimensional geometries by solving the Euler system of partial differential equations. It utilizes an explicit time integration scheme of Lax-Wendroff type. It is second order accurate in time and allows steady flow simulation with the local time-stepping technique or unsteady simulation with a uniform time step limited by the so-called CFD stability condition. Another functionality is the finite volume space integration scheme. It is a three-dimensional extension of the cell vertex Ni scheme [4] which is second order accurate in space on a Cartesian grid. The formulation is fully conservative so that shock and expansion waves are automatically captured.

The data layout is based on a multidomain meshing strategy. The global mesh is composed of an assembly of locally structured three-dimensional mesh blocks (I, J, K, families of mesh lines). All types of degenerations are allowed on the mesh boundaries, like mesh plans degenerating into a mesh line or a point. This is very useful for the meshing of complex geometries, but it requires the implementation of the convenient matching conditions.

The most time consuming part of the code is the subroutine that computes the time increments for the physical variables at each time step. It is composed of a succession of calls to subroutines that can be sorted into two groups:

- The “local” routines are called independently over subdomains. This group uses typically up to 90% of the total CPU time within the FORTRAN 77 code.
- The “boundary” routines perform the boundary conditions: flow conditions (inflow, outflow, walls, etc.) and numerical matching conditions (interfaces between subdomains). These routines make an intensive use of indirect addressing and involve dependencies between data belonging to different subdomains.

2.2 CFD Test Cases

Three meshes of increasing sizes (see Table 1) have been build around the same geometry of blunt body. The free flow conditions are: Mach number equal to 2.96 and angle of attack equal to 10o. With these conditions, the fluid flow over the blunt body shows strong shock and expansion waves: characteristic of MATRA industrial applications.

Test Case Name	Sizes and Number of Subdomains	Total Mesh Points	Processing Nodes
Small	(65 x 33 x 9) x 8	154440	1-8
Medium	(65 x 65 x 9) x 8	304200	2-16
Large	(129 x 65 x 9) x 8	603720	4-64

Table 1. Industrial Test Cases.

Due to the coarse grain parallelization strategy over subdomains (see next section), the number of HPF processes is limited by the number of subdomains. In order to run a number of HPF tasks higher than the initial number of subdomains, a pre-processor can be applied [6]. This pre-processor takes as input a mesh file with its associated topological description and generates automatically a new mesh data set with respect to the three following constraints :

- generate the given number of mesh blocks,
- optimize the load balancing (i.e. homogeneous mesh block sizes),
- minimize the size of blocks interfaces.

3. Initial Port to HPF

3.1 Porting to Fortran 90

Initially, the code was ported from FORTRAN 77 to Fortran 90. Apart from inserting F90 syntax, e.g. array syntax and interface blocks, we replaced the old one dimensional “work-array” with dynamic arrays. Some of the arrays became allocatable arrays, other ones, especially for local data, became automatic arrays. These changes made the code more flexible as the static size of the workspace is no longer given. But they were also absolutely necessary to allow the HPF distribution of the mesh data in a useful way.

The porting to Fortran 90 was supported by the Foresys (FORtran Engineering SYStem) tool from SIMULOG [11] that is a reverse-engineering, migration and development support system for Fortran. It was especially useful to generate interface blocks with intentions for the dummy arguments, and to take advantage of new syntax and new language features.

3.2 Coarse Grain Parallelization Strategy

For the HPF parallelization we have chosen the following strategy:

- The loops over the different subdomains calling the local routines provide *coarse grain* parallelism without communication. The HPF mapping directives have to guarantee that all data belonging to one subdomain is completely mapped to the same processor.
- The matching conditions of the interfaces of adjacent subdomains are difficult to handle. The initial strategy was the replication of the data and computations involved in the boundary conditions.

In the AEROLOG code, the data of the two- or three-dimensional subdomains is linearized and stored in a one-dimensional array. For using the coarse grain parallelism, it is essential that we can distribute the data in the program in such a way that one subdomain is completely owned by the processor that will work on this data. As the subgrids have different sizes, it would be necessary that HPF supports generalized block distributions where the user can pass to the compiler the corresponding block sizes for each processor. Unfortunately, none of the commercial HPF compilers supported this feature already during the project time. For this reason, we had to reorganize the one-dimensional mesh data arrays to two-dimensional ones. The second dimension corresponds to the subdomains numbering and will be distributed by `BLOCK` (see **Fig. 1**). This imposed significant changes to the code, not only for the arrays containing mesh data, but also for all integer arrays used for indirect addressing of mesh data.

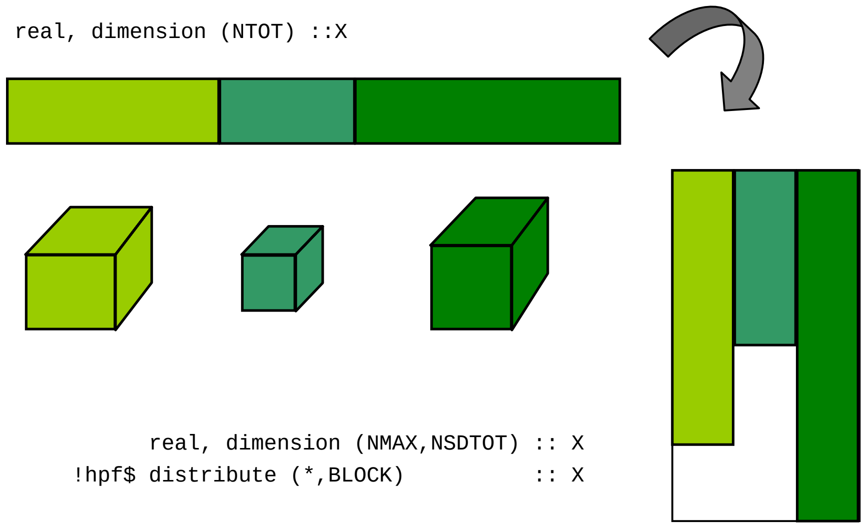

Fig. 1. New data structures for the mesh data and their distribution.

3.3 Coarse Grain HPF Implementation

With the help of the `INDEPENDENT` directive, we enabled the parallelization of the loops over the subdomains. The local routines are defined as `PURE` routines to allow their parallel execution for the different subdomains (see **Fig. 2**). Furthermore, the local routines have not to be parallelized at all and do not need any HPF directive.

The AEROLOG code takes advantage of sequence association. The subdomains are implicitly reshaped within the local subroutines. Within one subroutine, one subdomain is always considered as a three-dimensional rectangular grid. Though the HPF standard does not allow sequence association for mapped arguments, we could rely on it as long as it is only used for a single subdomain that is completely mapped to one processor.

For the boundary routines, the values of the boundary nodes of the different subdomains are gathered from the distributed mesh data. For the initial HPF port, the data and computations are replicated on all nodes and every processor updates the values of its boundary mesh nodes. The gathering of the distributed data is realized by replication of the whole mesh data. As the mesh data is not distributed within the boundary routine, implicit remapping at subroutine boundaries is utilized (see **Fig. 3**).

```
      integer  :: NSDTOT    ! total number of subdomains
      integer  :: NMAX      ! maximal size of one subdomain
      real, dimension (NMAX,NSDTOT) :: F !  mesh data, e.g. force
!hpf$ distribute F (*,block)             !  distribute the subdomains
      integer, dimension(NSDTOT) :: IM, JM, KM ! sizes
      ...
!hpf$ independent
      do NSD = 1, NSDTOT
        call LOCAL_ROUTINE (F(1,NSD),IM(NSD),JM(NSD),KM(NSD), ...)
      end do
      call BOUNDARY (F, NMAX, NSDTOT, ...)
      ....

      pure subroutine LOCAL_ROUTINE (F,IM,JM,KM,...)
      integer, intent(in) :: IM, JM, KM
      real, dimension(IM,JM,KM), intent(inout) :: F
      ...
      end subroutine LOCAL_ROUTINE
```

Fig. 2. Outline of the initial HPF AEROLOG Code.

```
      subroutine BOUNDARY (F, NMAX, NSDTOT, ISD_B, IJK_B, NB)
      integer, intent (in)    :: NMAX, NSDTOT, NB
      integer, dimension (2, NB), intent (in)  :: ISD_B, IJK_B
      real, dimension (NMAX, NSDTOT), intent (inout) :: F
!hpf$ distribute F(*,*)                ! replicated mesh data
      real    :: X1, X2, X
      integer :: IB, IJK1, ISD1, IJK2, ISD2
      do IB=1,NB
        IJK2 = IJK_B(2,IB); ISD2 = ISD_B(2,IB)
        IJK1 = IJK_B(1,IB); ISD1 = ISD_B(1,IB)
        X = (F(IJK1,ISD1) + F(IJK2,ISD2)) * 0.5
        F (IJK1,ISD1) = X; F(IJK2,ISD2) = X
      end do
      end subroutine
```

Fig. 3. Computation of boundary conditions in the AEROLOG code.

4. Code Review

We tested the initial HPF port with the following compilers:

- NAS HPFPlus by NA Software Liverpool, Release 2.01, a commercial HPF compiler that was also the target compiler for all HPF codes in the PHAROS project;
- PG HPF by Portland Inc., Oregon [9], AIX Rel. 2.2-1, another commercial HPF compiler;
- ADAPTOR HPF compiler, version 5.1 (Oct. 1997) [3], developed at SCAI in GMD, a research compiler that is available as public domain.

All results have been measured on the IBM SP2 at the GMD. We give the execution times in seconds for 5 iterations on the 'small' test case that works on 8 subdomains, every subdomain contains *65 x 33 x 9* mesh points (see also Table 1).

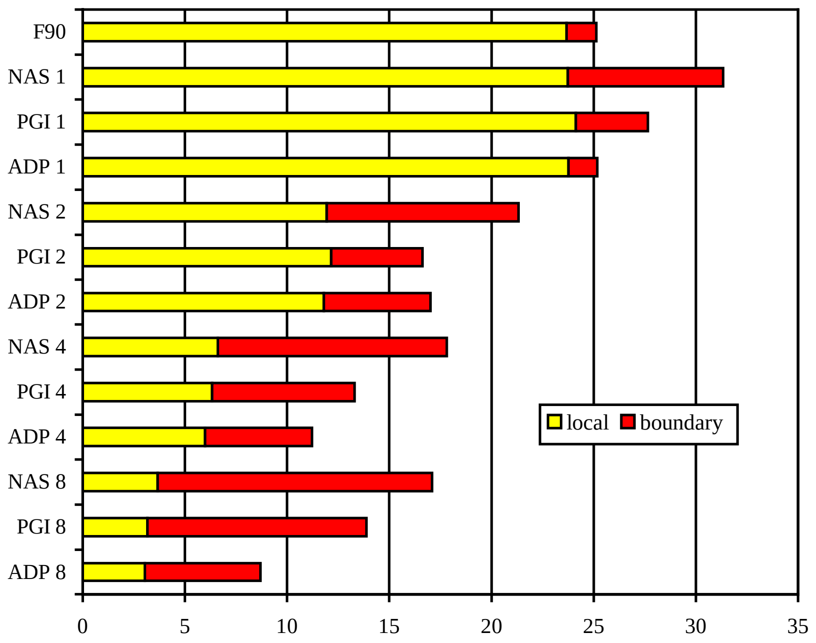

Fig. 4. Execution times (in seconds) of initial HPF version.

Fig. 4 shows the execution times of the initial HPF version, compiled by the native Fortran 90 compiler (xlf) and by the different HPF compilers running on 1, 2, 4, and 8 processors. The HPF version (considered as a Fortran 90 version without directives) has nearly the same performance as the original FORTRAN 77 version. The execution times, separated for the local and boundary routines, show that the local routines are parallelized perfectly. They scale well and the HPF parallelization causes no overhead.

But all boundary routines do not scale. In contrary, the execution time increases with the number of processors. This is due to the replication of distributed data that involves an all-to-all communication. Furthermore, it shows that the compilers have already different support for this kind of structured communication that follows a fixed communication pattern where every processor knows which data has to be sent and to be received.

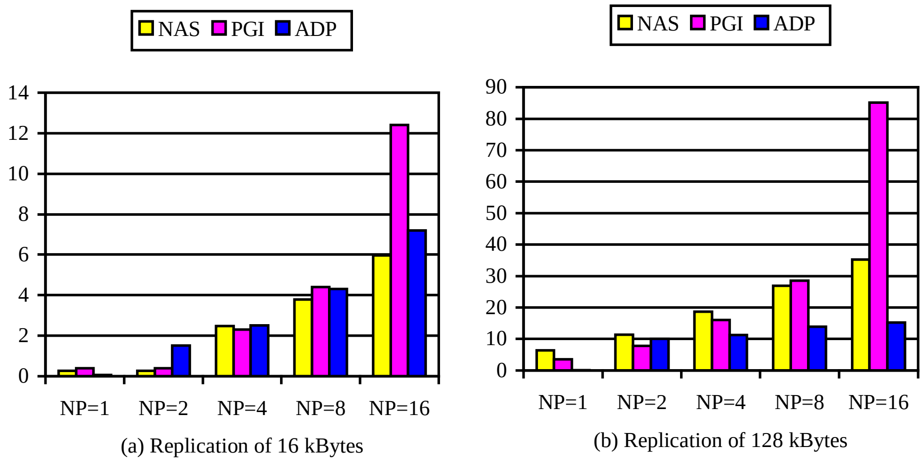

Fig. 5. Replication of distributed data.

In order to estimate the cost of replications, we benchmarked a sample code that performs only data replications of arrays with varying sizes. **Fig. 5** shows how much time (in milliseconds) the replication of distributed data needs for the different compilers and for the different number of processors. Array sizes of 16 Kbytes and 128 Kbytes are considered. The time for replicating distributed data increases with the size of the array and with the number of processors. The ADAPTOR runtime system is able to recognize at runtime that the replication of distributed data on a single processor does not require any copying at all.

5. Tuning of the HPF Code

As the results of the initial HPF port show, only the tuning of the boundary routines is necessary. We considered two strategies:

- We let the mesh arrays distributed for the boundary routines and relied on the capabilities of the HPF compiler to deal with unstructured communication. Unfortunately, all HPF compilers failed to generate more efficient code than for the initial

HPF version. Especially the NAS HPFPlus compiler provided absolutely no efficient support for indirect addressing.

- Instead of replicating the whole array containing the mesh data, we compressed the full mesh data to the boundary data before replicating it. This solution needed new data structures for packing and unpacking of boundary data (see **Fig. 6**). The packing of the data can be done independently for all subdomains. This approach required only HPF features that were supported by all HPF compilers.

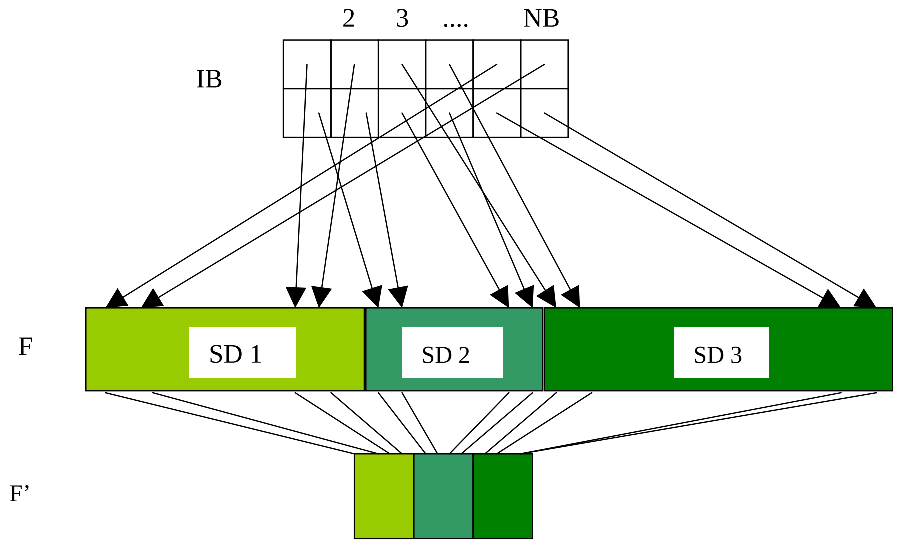

Fig. 6. Packing of boundary data.

By the packing of the boundary data, much less data has to be replicated between the different processors. Only the boundary data and not the whole mesh data is exchanged between the processors. The results shown in **Fig. 7** verify the effectiveness of the chosen approach. Compared to the sequential version, speedups from 4 to 5 on 8 processors are achieved.

6. Expectations for the Next Generation of HPF Compilers

The current tuned HPF version is still not fully portable between the different HPF compilers as the calling of local subroutines within an independent loop is supported differently. The NAS HPFPlus compiler did not act at all upon the `INDEPENDENT` directive, but scheduled the local computations, defined as `HPF_SERIAL` routines and not as `PURE` routines, on the processors owning the subdomain. The need for the slightly different versions of the HPF code should become redundant with the next releases of the HPF compilers.

Considering HPF 2.0 [8], we expect support for general block distributions. This would avoid the additional dimension for the subdomains and there would be no more wasting of memory in case of different subdomain sizes (see also **Fig. 1**). This feature is absolutely necessary to combine the evolution of the serial and the HPF version of the AEROLOG code.

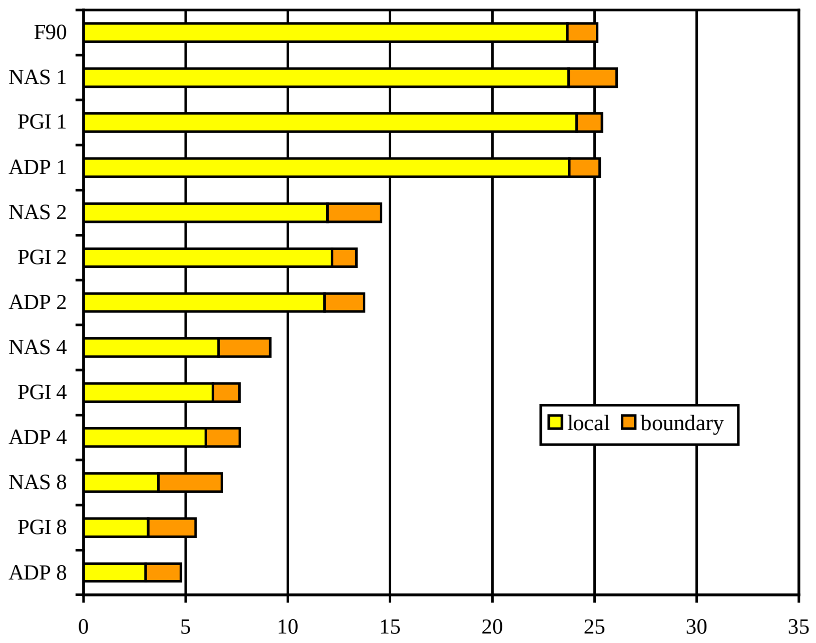

Fig. 7. Execution times (in seconds) of tuned HPF version.

The Amdahl limit restricts the maximum speed-up as long as the boundary computations are not parallelized. But then unstructured communication has to be supported. Therefore the compiler has to build a schedule required for accessing remote items of distributed arrays and for communication optimization. Unfortunately, schedules cannot be worked out at compile time, but only at run-time, when the values of the indirection arrays are known. The code design which first builds the schedule, then uses it to carry out the actual communication and computation, has been coined as the *inspector/executor* scheme [10]. The PGI and ADAPTOR compiler followed this design, but only the latest release of ADAPTOR tool provided sufficient support for reusing communication schedules by an additional directive [2].

Fig. 8 presents results for the 'medium' test case (100 iterations), for the local routines (local) and for different implementations of the boundary routines. The replication strategy of the initial and tuned HPF version does not scale. The unstructured communication (unstr.) for the parallelized boundary computations scales, but produces an unacceptable overhead due to the high costs for building the communication schedule. If the communication schedule can be reused, e.g. by tracing modifications of the indirection array as described in [2], the unstructured communication produces good results (traced).

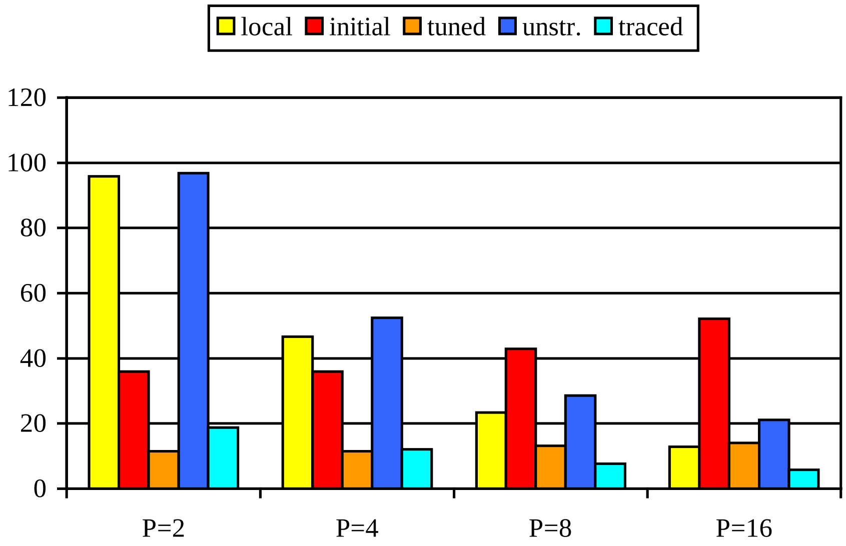

Fig. 8. Different tuning strategies of the boundary computations (ADAPTOR).

7. Benchmarking and Comparison with Message Passing

The porting of the AEROLOG code to HPF required important code changes as well as the message passing port, but it could be done step by step, always having a running version. This porting included useful code cleaning and modernized memory management. The replacement of the super-array technique in favor of Fortran 90 dynamic allocation of the local arrays brings simplification and flexibility to the code. But many code changes were only required due to the limited capabilities of the HPF compilers.

The HPF and the message passing version achieve nearly the same performance for smaller number of processors. But the message passing version of the AEROLOG code scales better (see **Fig. 9**). It parallelizes also the boundary routines and takes advantage of reusing explicitly communication schedules. But with an HPF compiler that supports unstructured communication and reuses schedules the scalability of the

MPI version can be nearly achieved as the results with the ADAPTOR compilation system verify.

From the code development and maintenance point of view, it is possible to replace smoothly the FORTRAN 77 reference code in favor of the Fortran 90/HPF code. The benefits of this migration will be the merging of the sequential/parallel shared/distributed memory versions of AEROLOG, which have reached very different levels of development at the moment. The migration of the complete AEROLOG code (implicit solver, Navier-Stokes solver, etc.) is eased by the choice of the coarse grain parallelization strategy based on the multidomain approach. In particular, it is not necessary to rewrite the local algorithms which can remain FORTRAN 77, saving a lot of porting efforts and bug risks. Experimental results with the ADAPTOR compiler have also shown that the HPF version is well suited for shared memory architectures by translating the HPF directives into parallelization directives for the native compiler. Due to the shared memory, runtime support for unstructured communication is not necessary.

In any case, we have seen from the PHAROS final benchmarks that it is not possible at the moment to get rid of the message-passing version of the code, which is the only one able to run efficiently enough on massively parallel computers. Enhancements of the HPF compiler technology are still required to a complete migration to HPF.

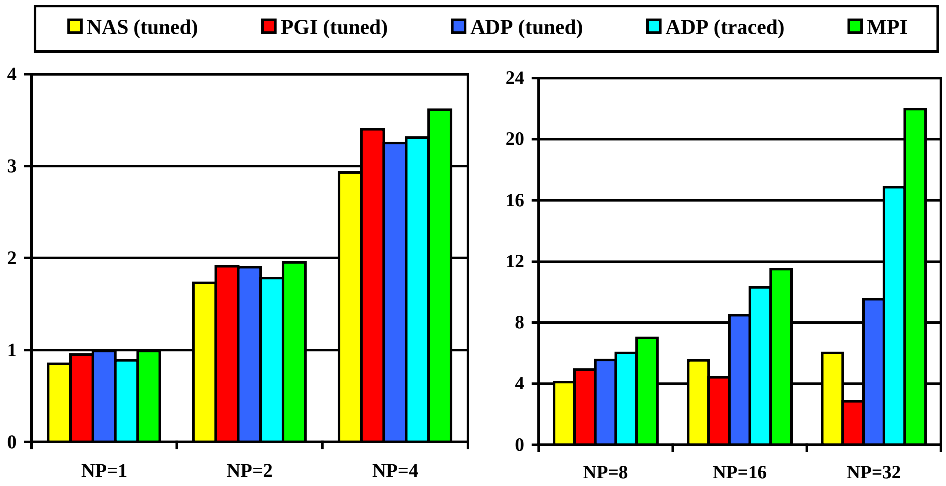

Fig. 9. Speedups on industrial test cases.

8. Conclusions

With the end of the PHAROS project, we have an HPF version of the AEROLOG code that runs with at least three HPF compilers and produces acceptable results for a limited number of processors. The porting effort was higher than expected because code restructuring was required in order to achieve the HPF implementation of the coarse grain parallel strategy. As HPF concepts are rather complicated for non specialists, the know-how transfer from tool providers and experts to the end-user was very important and might be considered as a major benefit of the PHAROS project.

At this time, the code is not fully portable as different language features are used for the two commercial HPF compilers. This is not only due to the missing support in the compilers, but also due to fact that the HPF standard was not not rigid enough, so that HPF directives led to various interpretations. With future releases of the HPF compilers, these problems will disappear. The tuning of the boundary conditions required a lot of effort. This effort might be less with advanced HPF compilers where unstructured communication is better supported.

The HPF version can directly be compiled for a serial machine achieving the same performance than the original code. While the independent computations over the subdomains scale well, the boundary conditions remain the critical part, even in the tuned version. Replication of mesh data is rather expensive, unstructured communication is not well supported. Due to the replication of the boundary computations, the scalability of this version is limited in any case.

Nevertheless, experimental results with the research compilation system ADAPTOR verify that a scalable and efficient HPF parallelization of the AERLOG software is possible if general block distributions and unstructured communication are sufficiently supported.

In conclusion of this project, we can state that HPF is a useful paradigm for porting large FORTRAN 77 applications to parallel architectures and in the long run the better alternative. But an efficient and portable parallelization and a higher productivity in software development can only be achieved if HPF compilers improve substantially.

Acknowledgements

The support of the POPCORN group (SCAI, GMD) for access to the IBM SP2 at the GMD is appreciated.

References

1. Borel, C., Brédif, M.: High Performance Parallelized Implicit Euler Solver for the Analysis of Unsteady Aerodynamic Flows. In Eccomas'92 : Conference Proceedings, pp. 1069-1076. Elsevier Science Publisher. 1992.
2. Brandes, Th., Germain, C.: A Tracing Protocol for Optimizing Data Parallel Irregular Computations. Accepted for publication at EuroPar'98, Southampton, Sep. 1998.
3. Brandes, Th., Höver-Klier, R.: ADAPTOR User's Guide (Version 5.1). Technical documentation, GMD, Oct. 1997. Available via anonymous ftp from ftp.gmd.de as gmd/adaptor/docs/uguide.ps.
4. Brédif, M., Chattol J., Koeck, P., Werlé, C: Simulation d'un système de Déviation de Jet à l'Aide des Équations d'Euler. *AGARD CP*, 412:13.1--13.11, 1986.
5. Brédif, M., Chapin, F., Borel, C., Simon, P.: Industrial Use of CFD for Missile Studies: New trends at MATRA BAe Dynamics France, in NATO-AVT panel on Missile Aerodynamics (11-15 May 98, Sorrento, Italy). Conference Proceedings, to be published.
6. Choukroun, F., Roux, F.X., Borel, C., Brédif, M.: Implementation of an Industrial CFD Code on a Massively Parallel Computer with Distributed Memory. In Parallel CFD'93 : New Algorithms and Applications, pp. 271-276, Elsevier Science, 1995.
7. High Performance Fortran Forum: High Performance Fortran Language Specification, Version 1.1, Department of Computer Science, Rice University, Nov. 1994.
8. High Performance Fortran Forum: High Performance Fortran Language Specification, Version 2.0, Department of Computer Science, Rice University, Jan. 1997.
9. PGHPF: Reference Manual, User's Guide, Technical Report, The Portland Group, Inc., Oregon, Nov. 1994.
10. Mirchandaney, R. et al: Principles of run-time support for parallel processing. In ACM Int. Conf. on Supercomputing, pages 140--152, 1988.
11. Foresys: SIMULOG. Foresys 2.0 User's Guide. Technical report, SIMULOG, 1998. http://www.simulog.fr/foresys.