NASoftware Limited
Roscoe House, 62 Roscoe Street, Liverpool L1 9DW. UK
Tel: +44 151 709 4738 Fax: +44 151 709 5645

Developing the NASL HPF Compiler

L.M. Delves
N.A. Software Ltd

Presented at the HPF+ Workshop
Amsterdam, April 1998

April 8th 1998

File: ESPRIT/HPF+/workshop.pap

1 Background

1.1 NASL Involvement in HPF

NASL has been involved with the development of Fortran90 and HPF compilers from the release of the relevant language definitions:

1990 Fortran90 Standard completed; FortranPlus compiler already under development

1992 First release of NASL FortranPlus compiler

1992-3 HPF1 defined

1992-5 ESPRIT PPPE project, within which the NASL HPFPlus compiler was developed to prototype form, with initial release shortly after

1996-7 Pharos project: experimental ports of industrial codes

1996-7 HPF+ project: extensions to HPF, and development of innovative compilation techniques

1997-8 : HEIDI project: "production" port of Alfa Romeo Aero combustion code

The NASL HPFPlus compiler (the marketing name is confusingly similar to the HPF+ project name!) remains the only European–developed HPF compiler to reach marketing status.

1.2 The HPF Marketplace

The Fortran90 and HPF markets are still small, but are growing steadily.

Fortran90 market :

Our FortranPlus Release 2 compiler implements the full Fortran95 standard and sells well, especially on PC Linux and Win95/NT boxes. Industry still largely uses Fortran77; but there is growing interest in porting industrial codes to Fortran90/95, and we get a healthy income from this activity. Most academic Fortran users now seem to be migrating to Fortran90/95; and the commercial future looks bright to us.

HPF Market :

There is widespread academic interest in HPF; and our largest HPF market is to academic research groups carrying out relatively large scale applications work on PC clusters. These groups place a high value on ease of programming new applications; generally speaking they do not have large existing codes to port; and provided performance is "reasonable" do not require, or make, direct comparisons with a message–passing solution. Moreover, the market involved is now growing quite fast: the possibility of building large high performance PC clusters composed of commodity low cost PC boxes with commodity interconnect (Myrinet, gigabit ethernet, etc) has excited substantial interest worldwide and has already spawned its own jargon (Beowulf systems; POPC (Pile Of PC) systems and (academic) user communities. These systems typically use the Linux operating system where we already have a substantial presence.

But industrial firms studying parallel architectures normally have large, often old, applications which they market. They seek reassurance that existing F77 codes can be ported cost effectively *and* with performance comparable to that given by MPI. The Pharos ports (see Section 4.1), and the HPF+ project, have generated much interest and companies are now seeking trial ports such as that being carried out in HEIDI. The HPF+ project gave us the opportunity to seek efficiency for a wider range of codes.

2 NASL Work in the HPF+ Project

Our role in the HPF+ project was primarily to:

1. Participate in the HPF+ Language Development
2. Include as many as possible of the new compilation techniques developed by Vienna, into our compiler

The language developments have been very important for us, and are described in a separate paper. In this paper we discuss the development work carried out on our HPF compiler.

3 Compiler Work

At project start we had reached Release 1.2 of our compiler; this was we believed able to handle industrial codes such as were in Pharos. However, with our initial study of the HPF+ kernels it became clear that it was not able to handle the constructs involved. Work on a new version of the compiler began (both within, and outside, the project) and involved the following stages:

1. A design review was carried out to ensure enhanced flexibility and an easier development path for future versions.

   This review process led to the conclusion that the necessary changes would need to be implemented in two stages (this later became three stages as the scale of the required effort became apparent). We planned a new Release 2 compiler with target delivery schedule mid–97, and a Release 3 compiler with a target delivery schedule of end-98.

2. Release 2 included implementation of initial optimisations (mask merging) to improve overall efficiency, and provision of low level HPF constructs to make it *possible* (though hard work) to achieve good performance for a wider range of codes. This release was based on design work early in the project, ie on our then available expertise.

   The HPF constructs implemented included:

   (a) Multi–statement FORALL - enabling a range of parallel DO loops to be written in a form the compiler could handle efficiently;

(b) An innovative implementation of `EXTRINSIC(HPF_SERIAL` procedures. Intelligent placement of the serial process (by inspecting the actual procedure arguments) allowed a wide range of DO loops to be parallelised with some programmer assistance;

(c) `EXTRINSIC HPF_LOCAL` procedures –also put into the Vienna VFC compiler to provide a facility for very irregular codes.

Release 2.1 was issued in beta form in October 1997. It proved very capable of handling the Pharos codes, and via the availability of `HPF_LOCAL` began to address the issues posed by the HPF+ kernels although still lacking the functionality needed to handle them at a high level.

3. Within HPF+ the design of Release 3 has also been completed [1]. This design is based on experience gained within HPF+, including a very detailed study of the HPF+ kernels. It includes for the first time an implementation of `INDEPENDENT, NEW`, and `REDUCTION`, together with `PUREST` procedures and a substantially enhanced treatment of indirect array accessing. `GEN_BLOCK` and `INDIRECT` distributions are also handled.

   It is hoped to have this release ready around the end of 1998. Even then we do not expect to show good performance on the HPF+ kernels without substantial low level restructuring of the kernel sources, since the HPF+ extensions `HALO, REUSE` will not be implemented within Release 3 due to the effort needed. We are convinced of the need for these extensions; provided that the effort can be made available they are planned for Release 4.

4 Examples of Compiler Developments

The improvements in Release 2.1 of our compiler are of immediate benefit to users. We illustrate the improvement in the context both of the HPF+ irregular kernels, and the industrial ports carried out in the Pharos project.

4.1 Improvements on the Pharos Codes

The PHAROS ESPRIT project ported four industrial codes using the NASL HPF toolset:

- SEM3D (SEMCAP/ESI): Electromagnetic Response Simulation
- DBETSY3D (debis/Mercedes): Linear Elastostatics
- CANT-SD (CISE): FE static and dynamic analysis
- AEROLOG (Matra BAe): high speed CFD flows

The project ran in parallel with HPF+, and benefited substantially from the compiler development carried out within HPF+. By the end of the project, Release 2.01 of our HPF compiler was available and all four ports were satisfactorily completed, with three of the four giving good speedups (the fourth, CANT-SD, showed poor speedups with the message–passing version too). There remained a performance drop of between nil and a

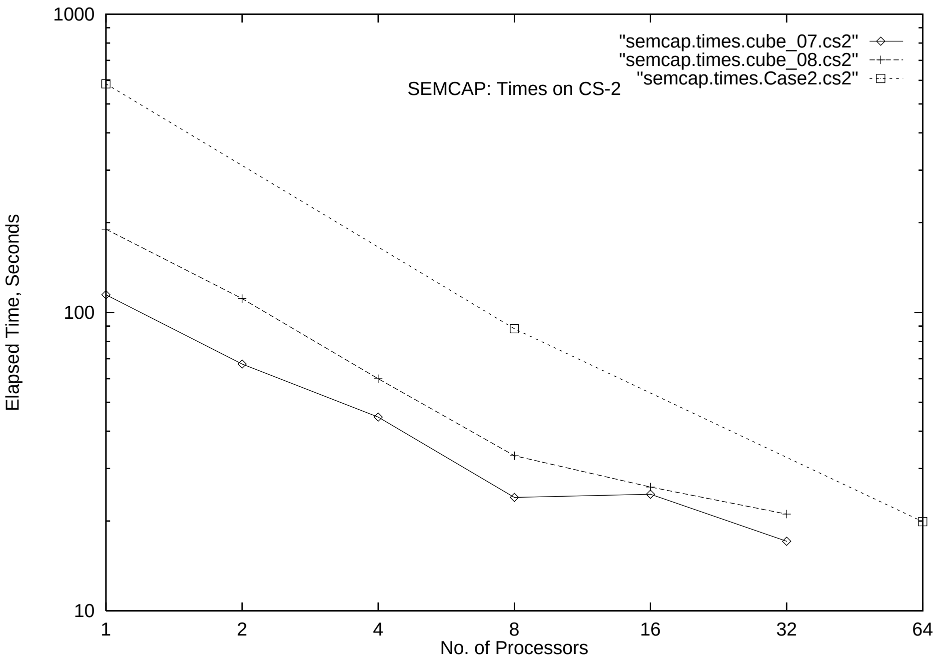

Figure 1. Performance of the Pharos SEMC3D code on the Meiko CS2

factor 2-3 compared with the MPI codes, stemming not from the HPF compiler but from the backend Fortran90 compilers available on the Meiko CS2 – this lack of performance of Fortran90 compilers has shown up in HPF+ too.

Figure 1 shows the performance achieved on the SEMC3D code on the Meiko CS2 at Vienna, with three different datasets. The scalability shown is very good, in particular for the production dataset *Case2*. The speedup figures are as good as is achieved with the message–passing version of the code. Figure 2 gives results for the Aerolog code on the IBM SP2 at GMD Bonn, including a direct comparison of our compiler with that from PGI. This code relies more heavily on indirect addressing (for the boundary condition handling). The results show that we scale substantially better than PGI, though there is still a performance gap relative to MPI.

Thus Pharos showed that Release 2 of our compiler can handle "sympathetic" industrial codes quite effectively, although much remains to be done to match the use of MPI for every code. These good results were very substantially aided by the work within HPF+.

4.2 Performance on the HPF+ Kernels

At project start, our compiler could not handle *any* of the HPF+ kernel codes. In general, these kernel codes still cause us problems because Release 2.1 of our compiler can handle irregular codes only via use of the low level `HPF_LOCAL`, `HPF_SERIAL` constructs, while the kernels themselves use a high level style with more left to the compiler. However, we are substantially better now: as exemplified in the results given in [2], Release 2.1 is able to produce respectable performance figures for the HPF+ kernels provided that the effort needed to introduce `HPF_SERIAL` or `HPF_LOCAL` procedures as necessary, is available.

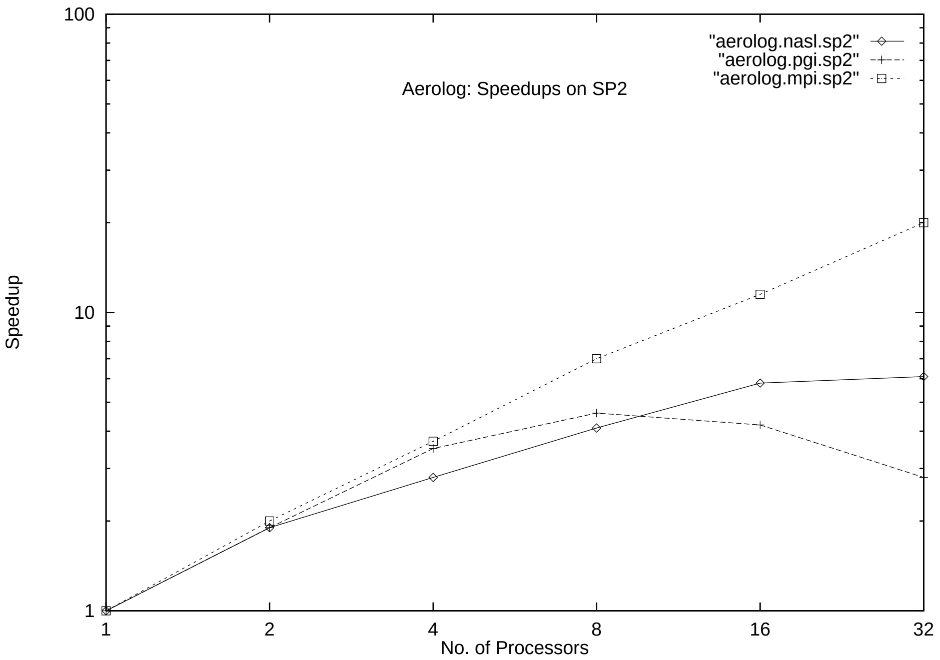

Figure 2. Comparison with pgiHPF and MPI for the Aerolog code

Figures 3 and 4 show the performance on the ECMWF LG kernel, which *was* implemented by ECMWF using `HPF_LOCAL`; and on the AVL kernel. In both cases the results obtained using our Release 2.1 compiler are similar to those obtained with the Vienna VFC compiler. These results are we believe quite gratifying, and demonstrate the progress made during the project; but they do not remove the commercial need to give programmers the ability to handle such difficulties at a higher level.

References

[1] *The Design of Release 3 of the NASL HPF Compiler*, Internal Note, D. Watson, April 1998

[2] *Benchmarking the HPF+ Kernels with the NASL Release 2.1 Compiler*, HPF+ Working Paper, G. Howard and L.M. Delves, March 1998

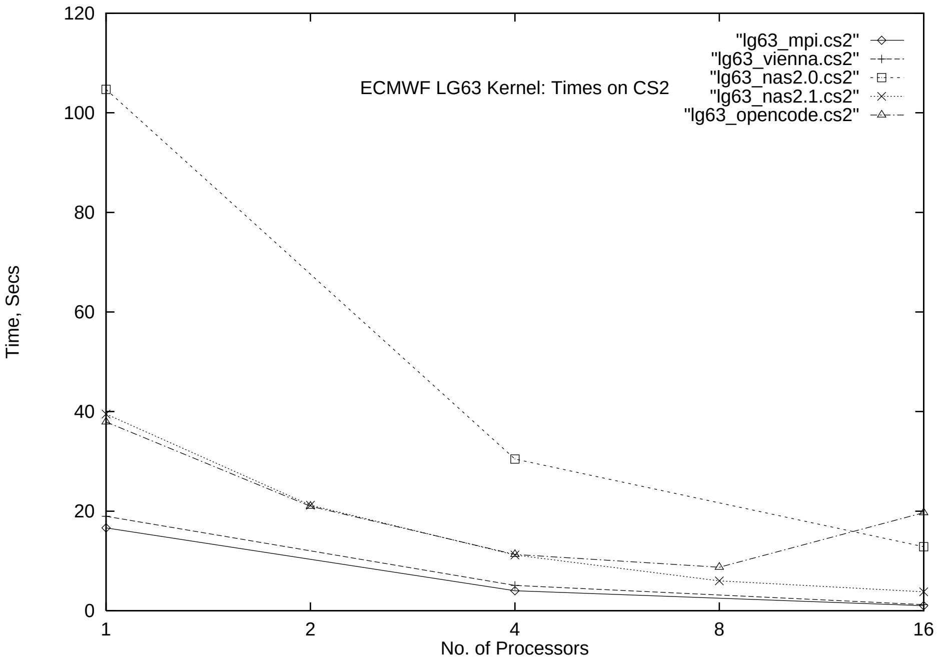

Figure 3. Results for the ECMWF LG Kernel on the CS2

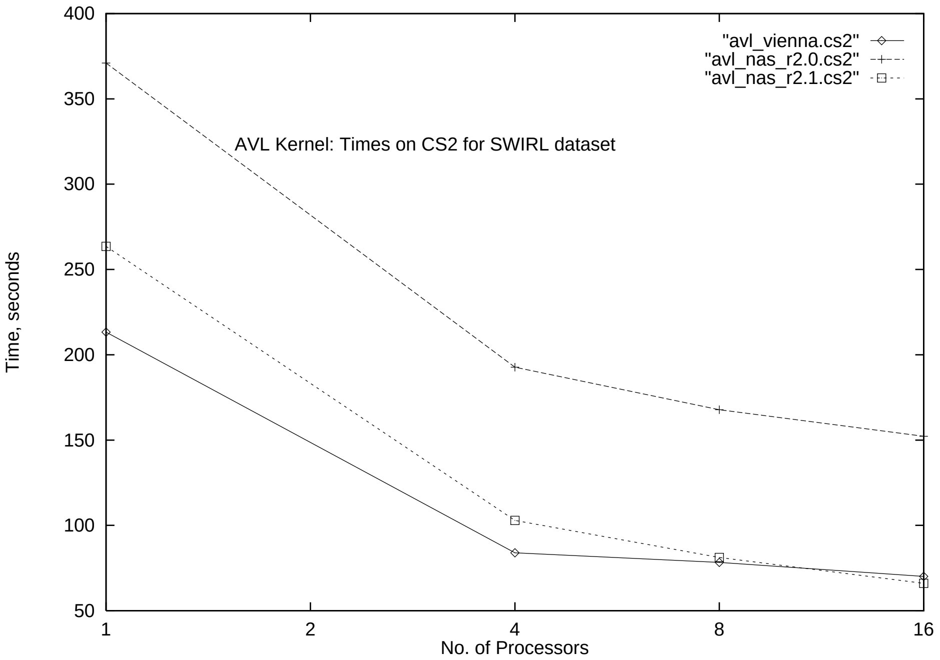

Figure 4. Results for the AVL Kernel on the CS2