ECMWF experience with HPF+

George Mozdzynski
European Centre for Medium-range Weather Forecasts
Shinfield Park, Reading, RG2 9AX, U.K.
e-mail : George.Mozdzynski@ecmwf.int

April 14, 1998

Abstract

The HPF+ project [1] started in early 1996 to investigate how to make the HPF language more applicable to large industrial applications. One such application investigated by this project was the IFS forecast model from ECMWF. New HPF language features were implemented in an experimental HPF compiler called the Vienna Fortran Compiler (VFC) based on the requirements of the application partners in the project. This paper presents some results from this project, and in particular, provides an evaluation of an extended HPF language (HPF+) by comparing some IFS kernels against message-passing equivalents.

1 Introduction

HPF arose from the desire to have a common Fortran-based language to program the growing number of parallel computer architectures that existed in the late 1980's. These architectures included SIMD (Thinking Machines, MasPar) and MIMD (Cray C90, IBM SP1, Intel Paragon, KSR1, SGI Challenge) shared and distributed memory machines.

While the MIMD systems could be programmed using Fortran with message passing for inter-processor communication and synchronisation (e.g. PVM), this typically

[1] HPF+ is supported by the European Commision under the ESPRIT IV, Long Term Research Project nr. 21033.

involved a large programming effort and a maintenance problem, as both serial and parallel code versions were developed and had to be supported. On the other hand the SIMD systems (do any still exist?) required vendor specific compilers and libraries.

In May 1993, the first specification of HPF (v1.0) was published by the High Performance Fortran Forum (HPFF), which contained members from industry, universities and U.S. government laboratories. In November 1994, HPF v1.1 was released, which mainly incorporated corrections (and some clarifications) to the language. This specification resulted in the development of a number of HPF v1.1 compilers, however, it was also recognised that the v1.1 standard required further development to be applicable to large industrial applications. One such effort was the HPF+ project, which investigated HPF language extensions based on the requirements of a small number of such applications, one of which was the the Integrated Forecasting System (IFS) from the European Centre for Medium-Range Weather Forecasts (ECMWF).

The contributions made by the HPF+ project and other members of the HPFF resulted in the publication of the HPF v2.0 specification in January 1997. Since that time the HPF+ project has identified additional language features that will be proposed for a future specification of HPF.

It is important to note that the HPF+ project was not founded to develop a new HPF language. Its main goal was to provide input to the HPF standardisation process by investigating the requirements of a set of advanced application codes. This was achieved by testing new language features in an experimental HPF compiler called VFC [1][2].

In this paper, three kernel programs from ECMWF (named TS, LG and SL) are evaluated. These kernels represent key components of the IFS weather forecast model, which is described briefly in the following section. After this, each kernel is described separately, including descriptions of the major data structures and language features used in the HPF+ and MPI versions.

In this evaluation, the performance of the HPF+ and MPI kernels are compared against a serial version of the same kernel.

Each kernel is compiled using the same SUN F90 compiler (f90: WorkShop Compilers 4.2 10/22/96 FORTRAN 90 1.2) and optimisation level (-O3), to facilitate a more straightforward comparison of language features. It should be noted that the HPF+ compiler (VFC) is a pre-processor that generates Fortran 90 source code, including calls to a HPF+ runtime library. For the MPI kernels, version MPICH-1.0.13 has been used. The release of VFC used for the evaluation of the HPF+ kernels is version 1.00. Finally, a discussion of the main points found during the evaluation and future HPF+ requirements are presented in a summary section.

1.1 IFS

The IFS uses a spectral forecast model for the prediction of weather for a period of up to 10 days ahead. This model [3][4][5] is highly parallel and has been implemented today on both distributed and shared memory systems. Given a strong desire to protect the scientific code from details of the parallel implementation, a transposition strategy is used. With this approach, the complete data required is redistributed at various stages of a time step so that the arithmetic computations between two consecutive transpositions can be performed without any interprocess communication. Such an approach is feasible because data dependencies in the forecast model exist only in one coordinate direction, this direction being different for each algorithmic component. An overwhelming practical advantage of this technique is that the interprocessor communication is localised in a few routines. The transpositions are executed prior to the appropriate algorithmic stage, so that the computational routines (which constitute the vast bulk of the IFS source code) need have no knowledge of this activity. In terms of HPF, the transpositions are no more than redistributions that would permit more efficient data distributions to be used during an algorithmic stage.

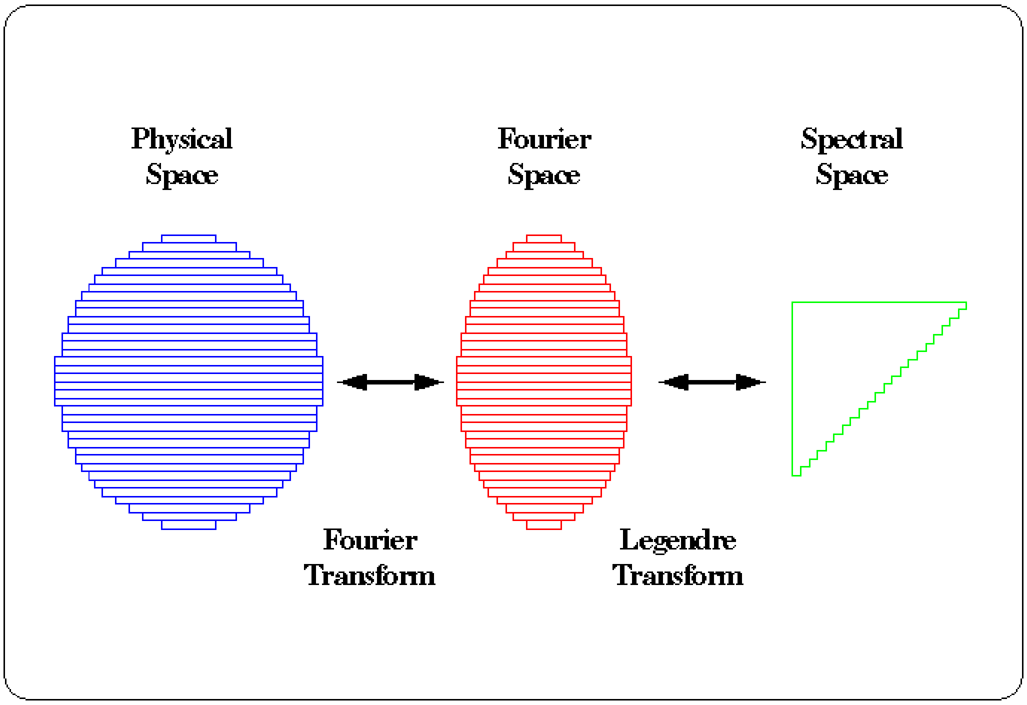

Figure 1: Spectral transform method

1.2 HPF+ Kernel Issues

It is the representation of the data spaces used in the IFS that present a technical difficulty for HPF as they are all irregular, as shown in Figure 1. Grid point space, fourier space and spectral space cannot be represented by simple BLOCK or CYCLIC distributions without the severe overhead of additional interprocessor communication during the algorithmic stages. Instead GEN_BLOCK and INDIRECT distributions are required as explained in the following sections.

As the computational routines of the IFS require no communication, they can take advantage of the efficient HPF_LOCAL extrinsic mechanism. Further, the complexity and number of these routines preclude an evaluation with an experimental HPF compiler such as VFC and for this reason are not part of this evaluation. Rather, the kernels explore the issues relating to the representation of the IFS data spaces and the transformation of these spaces in a forecast model timestep.

Finally, the kernels have all been coded using modules with allocatable data structures. This dynamic structure is exactly the same as that used in the production IFS and allows the same executable to run all forecast resolutions. For the HPF+ kernels this involves computing map information for GEN_BLOCK and INDIRECT distributions.

1.3 Grid point space

Grid point space (also known as physical space) is a quasi-regular (so-called reduced) 3 dimensional grid in which the number of grid points per latitude is reduced as the poles are approached. For example, at model resolution T213 each field has 640 grid points on equatorial latitudes and only 18 grid points on latitudes at the north and south poles. The number of grid points for a single model variable at one level in the atmosphere is shown in Table 1.

IFS Spectral Resolution	Grid Points
T63	13280
T106	35718
T213	134028

Table 1: Number of Grid Points for different IFS configurations

With the exception of the semi-Lagrangian (SL) calculations, grid-point computations only involve vertical dependencies. This would permit use of a simple BLOCK distribution for the horizontal (i.e. lat, lon) domain of grid-point space, which is represented in a single array dimension. However, the SL advection computations require access to grid point data in the upwind direction in order to compute departure points. Using a BLOCK distribution for grid-point space would result in a large communication overhead as the number of processors increase. This is due to the processors containing a decreasing number of latitudes and the partitions becoming long and thin. One way to resolve this data dependency would be to minimise the amount of data required from neighbouring processors. This would require a 2 dimensional partitioning of grid point space, which can be represented by use of an INDIRECT distribution.

1.4 Fourier space

Fourier space is similar to grid point space in its structure, with the number of fourier coefficients for each latitude reducing towards the poles. At model resolution T213, each field has 213 (complex) coefficients at the equator while only 8 at the poles. Fast fourier transforms (FFTs) are used to transform grid point space to fourier space. These require access to all grid points on a latitude, and are independent over latitudes and fields. A field is a 2 dimensional model variable at the surface or at one of 31 levels in the atmosphere. The distribution of fourier space can be achieved by using a combination of GEN_BLOCK for complete latitudes and BLOCK for fields. However, it should be noted that the fourier transforms used in the IFS vectorise over fields, and any distribution of fields (such as BLOCK) would result in shorter vector lengths and a reduction in vector efficiency. Legendre transforms (LTs) are used to transform fourier space to spectral space. These transforms require access to all latitudes for a particular wave number (m), and are independent of each other and of fields. This dependency on all latitudes is a case for another (GEN_BLOCK,BLOCK) distribution of fourier space, with a transposition to get from the previous distribution suited for FFTs to one suited for LTs.

1.5 Spectral space

Spectral space is a 3 dimensional space in which two of the dimensions represent a triangular data structure (m,n) where m is the number of zonal wave numbers and n the number of total wave numbers. The third dimension of spectral space represent levels in the atmosphere. Computations in spectral space have different data dependencies in various phases and have either no dependencies (horizonal diffusion calculation), or a dependency over n. For these reasons, spectral wave space would be distributed using GEN_BLOCK where for practical and communication efficiency reasons the zonal wave numbers (m) would be distributed as in the legendre transforms.

2 TS kernel

The Time-Step (TS) kernel is by far the most complex of the three ECMWF kernels. It uses fourier and legendre transforms to transform data between grid-point space and spectral space. In a single time-step data is transformed from grid-point space to spectral space and back to grid-point space.

Two HPF+ variants of the TS kernel were coded, with the fourier and legendre transforms being performed by using INDEPENDENT loops (variant 1) or by using the extrinsic HPF_LOCAL mechanism (variant 2).

Unfortunately, VFC 1.00 was unable to parallelise the critical INDEPENDENT loops of variant 1 due to a compiler restriction that bounds of inner loops must be invariant, that is, they cannot be dependent on outer loops. No practical workaround was found to bypass this restriction in this kernel.

On the other hand, the second variant using the HPF_LOCAL extrinsic mechanism encountered no such compiler restrictions. Fourier and spectral spaces were distributed using GEN_BLOCK distribution. This allowed the fourier and legendre transforms to execute efficiently using the HPF_LOCAL extrinsic mechanism and without any interprocessor communication. While grid-point space would ideally be distributed with an INDIRECT distribution for a general two dimensional partitioning strategy, a one dimensional approach was used to simplify the complexity of these transforms. For this reason grid-point space could be distributed with a simple BLOCK distribution. The distribution of the legendre polynomials (computed during initialisation) represented a difficulty for HPF, as they both needed to be distributed (because of their size) and to have different distributions for the initial computation (over latitudes) and in the legendre transforms (over waves). This was achieved by using an initial BLOCK distribution for the polynomials over latitudes (all waves) and then redistributing over waves (all latitudes). This was accomplished by the specification of the DYNAMIC attribute on the initial BLOCK distribution, and then by using a REDISTRIBUTE directive to perform the redistribution. This is much easier than the equivalent message passing routine (sutrle.F), which takes 150 lines to do what is achieved by a single HPF directive!

An attempt was made to port the HPF kernel (variant 2) to the pghpf (version 2.2) compiler on the QSW CS-2 at the University of Vienna. It was found that this compiler did not support GEN_BLOCK distributions. Furthermore, as there was no straightforward way to modify the kernel to get it to compile and run correctly, the port was abandoned. This is a serious problem when using HPF compilers. If you have a Fortran 90 code, you know (or are at least 99% certain) that it will compile and run correctly whatever F90 compiler you use. The same cannot be said for HPF. Are users expected to analyse which features HPF compiler suppliers have decided to implement? For this reason HPF programs today are not portable. If on the

Problem	Version		1p	4p	16p
T63	SERIAL	ms	10,857		
		s/u	1.00		
	MPI	ms	8,797	2,097	584
		s/u	1.27 (1.00)	5.28 (4.20)	18.59 (15.06)
	HPF+	ms	15,259	4,449	1,274
		s/u	0.71 (1.00)	2.44 (3.43)	8.52 (11.98)
T106	SERIAL	ms	34,953		
		s/u	1.00		
	MPI	ms	29,574	7,399	1,953
		s/u	1.18 (1.00)	4.72 (4.00)	17.90 (15.14)
	HPF+	ms	not enough	14,740	3,898
		s/u	memory	2.37 (1.00)	8.97 (3.78)
T213	SERIAL	ms	206,797		
		s/u	1.00		
	MPI	ms	not enough	52,051	12,349
		s/u	memory	3.97 (1.00)	16.75 (4.21)
	HPF+	ms	not enough	not enough	not enough
		s/u	memory	memory	memory

Table 2: Comparison of run times (in millisecs per time step) and speedups for the versions of the TS kernel for different numbers of processors on a QSW CS2

other hand, programmers restricted themselves to a common subset of HPF language features, then this would not be sufficient for programs with irregular data structures.

The performance of the TS kernel is summarised in Table 2 for three problem sizes - T63, T106 and T213, representing a small, medium and large problem - against an increasing number of processors. For each of the problem sizes, performance is shown for the serial kernel, message-passing kernel and HPF+ kernel (variant 2). In this table, timing is presented in millisecs (m/s) per time-step, and speed-up (s/u) is the performance attained relative to the serial version of the kernel. Where speedup is shown in brackets, this is relative to the same code version.

Figure 2 shows graphically the performance of the message-passing and HPF+ kernels at T63 resolution. As can be seen the MPI kernel scales very well, achieving ideal speedups relative to the serial version. Further, the single processor version of the MPI kernel is 27% faster than the original serial version. This is due to the legendre transforms (direct and inverse) running about twice as fast. This, in turn, is thought to be due to different compiler optimisation of the respective source files. Besides this there is evidence of some super-linear speedup, which is most probably a caching

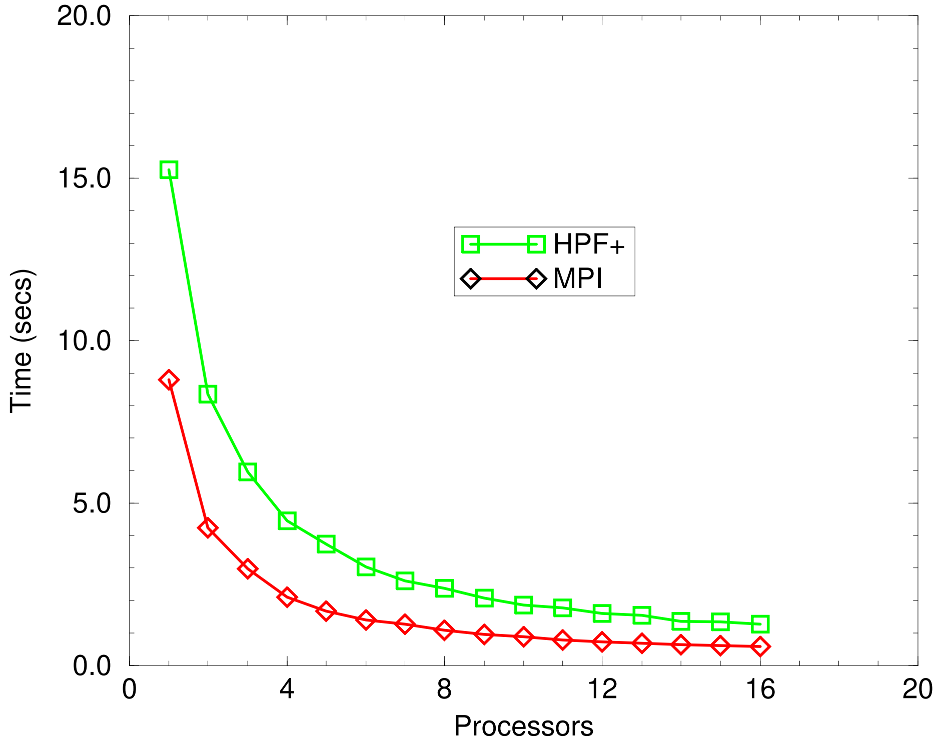

Figure 2: TS kernel performance

effect on the QSW CS-2. In comparison, the HPF+ kernel scales moderately well, achieving a speedup of 8 to 9 on 16 processors. This is no surprise, as HPF overheads will always be greater than message-passing overheads, especially for simple loops where communication is required. However, if we were to include the large amount of code in the grid-point, fourier and spectral space computations (as used in the IFS forecast model) in a HPF_LOCAL context then the message-passing and HPF+ performance would get much closer.

3 LG kernel

The LG kernel was developed to address the IFS program issues of grid-point space, namely,

- Indexed access to latitude data structures
- E-W data dependencies of Fast Fourier Transforms
- Vertical dependencies of physics calculations
- Blocking of grid-point space to conserve memory usage in physics calculations and exploit locality of reference

As an example, the original serial version of the GTOL transposition (kernel fragment) was coded as:

```
REAL ZGA(NPROMA,NGTO,NBLOCKS)
REAL ZGL(NPROMAG,NGTO)
REAL ZTMP(NGPTOTG)

DO JFLD=1,NGTO
  IB=0
  DO J=1,NGPTOTG,NPROMA
    IBEG=1
    IEND=MIN(NPROMA,NGPTOTG-J+1)
    IB=IB+1
    DO JJ=IBEG,IEND
      ZTMP(J+JJ-1)=ZGA(JJ,JFLD,IB)
    ENDDO
  ENDDO
  DO JGL=1,NDGLG
    DO JL=1,NLOENG(JGL)
      ZGL(NSTABUFL(JGL)+JL,JFLD)=ZTMP(NSTABUFG(JGL)+JL-1)
    ENDDO
  ENDDO
ENDDO
```

In the above kernel fragment the blocking of ZGA (by NPROMA) was required to conserve space in physics routines where some automatic arrays are dimensioned (NPROMA, NFLEV, NFLEV) - where NPROMA is the block size (or vector length) and NFLEV the number of model levels (31 today). Without blocking, each of these arrays at the operational T213L31 resolution would be just over 1 Gbyte. With blocking, the space used by these arrays is reduced by the factor 134028/NPROMA or NBLOCKS.

The difficulty for HPF is how to distribute ZGA, as the first and third dimensions together represent a field that we would like to distribute INDIRECT. This issue was addressed in the LG kernel by only using the blocked structure in a HPF_LOCAL context, and an INDIRECT distribution (unblocked) outside of this.

Another issue (an implementation constraint of VFC) is that the bounds of nested independent loops must be invariant, which would be violated by the (JGL,JL) loops above. The workaround for this constraint was to collapse the (JGL,JL) loops into one (J) loop and use a pre-computed index array (INDL).

The HPF+ version for the same kernel extract was therefore coded as:

```
!HPF$ PROCESSORS PROCS(NPROC), PROCSAB(NPROCA,NPROCB)
!HPF$ DISTRIBUTE(GEN_BLOCK(MAPGLA),INDIRECT(MAPFLD0)) ONTO PROCSAB :: ZGL
      REAL ZGL(NPROMAG,NGT0)
!HPF$ DISTRIBUTE(INDIRECT(MAPGP ),*) ONTO PROCS   :: ZGA
      REAL ZGA(NGPTOTG,NGT0)

!HPF$ INDEPENDENT,NEW(JFLD),ONHOME(ZGL(INDL(J),:)),REUSE
      DO J=1,NGPTOTG
        DO JFLD=1,NGT0
          ZGL(INDL(J),JFLD)=ZGA(J,JFLD)
        ENDDO
      ENDDO
```

As can be seen in the above fragment, the LG kernel uses (GEN_BLOCK,INDIRECT) distributions for ZGL. In addition, communication schedule reuse was specified by the REUSE clause on the INDEPENDENT directive.

The use of an INDIRECT distribution to distribute (NGT0) fields over NPROCB processors was required as both 3D variables (i.e. those with a vertical dimension) and 2D variables (surface fields) are included in this dimension. In this case we need to keep the same set of vertical levels for all 3D variables distributed to the same NPROCB processor. This is needed in the IFS fourier space computations, where calculations are performed in fourier space over latitudes and levels. To avoid communication we need variables with the same latitudes and levels in the atmosphere to be in the same processor.

The LG kernel was relatively straightforward to code in HPF+ and certainly easier than the equivalent message passing version. HPF_LOCAL provided a simple and easy to use facility for switching between global and local programming methods and can be used to make HPF more flexible. Porting or writing HPF programs would be made even easier if HPF extensions were exclusively restricted to directives. Extensions such as PURE or EXTRINSIC are not recognised by Fortran 90 compilers and need to be removed before such compilers are used.

Problem	Version		1p	4p	16p
T63	SERIAL	ms	15,888		
		s/u	1.00		
	MPI	ms	15,649	3,911	1,006
		s/u	1.02 (1.00)	4.06 (4.00)	15.79 (15.56)
	HPF+	ms	16,933	4,282	1,120
		s/u	0.94 (1.00)	3.71 (3.95)	14.19 (15.12)
T106	SERIAL	ms	42,645		
		s/u	1.00		
	MPI	ms	42,554	10,608	2,673
		s/u	1.00 (1.00)	4.02 (4.01)	15.95 (15.92)
	HPF+	ms	45,881	11,515	2,926
		s/u	0.93 (1.00)	3.70 (3.98)	14.57 (15.68)
T213	SERIAL	ms	164,642		
		s/u	1.00		
	MPI	ms	not enough	40,951	10,327
		s/u	memory	4.02 (1.00)	15.94 (3.97)
	HPF+	ms	not enough	44,470	11,233
		s/u	memory	3.70 (1.00)	14.66 (3.96)

Table 3: Comparison of run times (in millisecs per time step) and speedups for the versions of the LG kernel for different numbers of processors on a QSW CS2

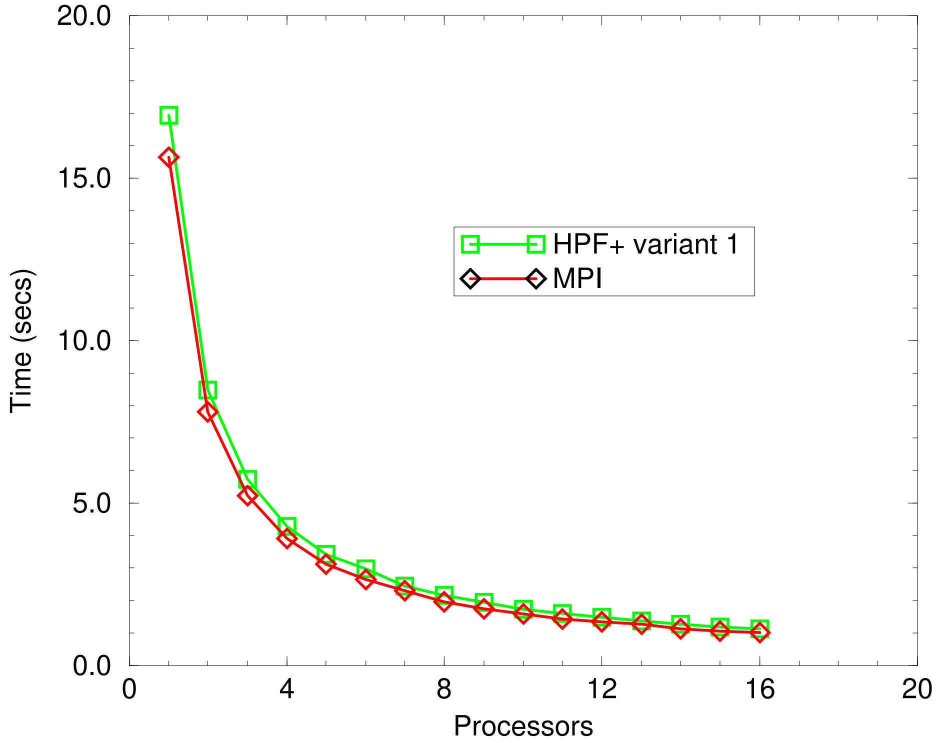

Figure 3: LG kernel performance

The performance of the LG kernel is characterised by two phases of compute (simulated FFTs and physics) where no communication is required, and two phases of communication, which are the grid-point space transpositions. As before, performance is summarised in Table 3 for three problem sizes, T63, T106 and T213, representing a small, medium and large problem, against an increasing number of processors. For each of the problem sizes, timings are presented for the serial kernel, message-passing kernel and the HPF+ kernel. In this table, times are shown in millisecs (m/s) per time-step and speed-up (s/u) is the performance attained relative to the serial version of the kernel. Where speedup is shown in brackets, this is relative to the same code version.

Figure 3 shows graphically the performance of the message-passing and HPF+ (variant 1) kernels at T63 resolution. The performance of the message-passing (MPI) kernel shows near perfect scaling on up to 16 processors with a speed-up of over 15. The HPF+ kernel also performs well against the MPI version, showing a speedup of over 14 on 16 processors.

4 SL kernel

The semi-Lagrangian (SL) calculation [6] in the IFS consists of two parts, the computation of a trajectory from a grid point backwards in time, and the interpolation of various quantities at the departure point and at the mid-point of the trajectory. The problem for distributed memory systems is that both these parts require access to grid-point data held on neighbouring processors. The message-passing approach used in the IFS today is to construct a halo of all grid columns from these processors based on the global maximum wind likely to be encountered, as shown in Figure 4. Such a halo mechanism is required, as the computation of the trajectory for each grid-point is an iterative process where only 2 or 3 iterations are performed due to the cost.

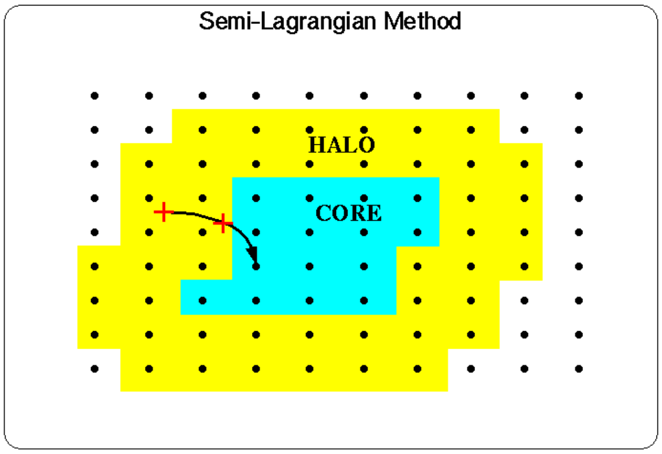

Figure 4: Semi Lagrangian method, showing computation of trajectory, requiring interpolation of grid-point data held on neighbouring processors (halo area)

For the HPF+ version, a HALO specification is used as part of an INDIRECT distribution directive. This HALO specification simply specifies a data structure that describes for each processor the list of non-local global indices that are needed. Also an UPDATE_HALO directive is specified to indicate when the halo should be updated. In the IFS, the SL halo data is only read and never written to.

It should be noted that the HPF-2 SHADOW capability is only applicable for regular nearest neighbour problems, and cannot be used in the above context where a single array dimension contains an irregular structure that is distributed INDIRECT.

Finally, the SL interpolation routines in the IFS are very complex and have many runtime controls that specify, among other things, the interpolation method to be used (e.g. 32 point stencil, 3D interpolation) and the variables to be interpolated. For this reason they are only approximated in the HPF+ kernel by a simple averaging of neighbouring grid-points, as shown in the HPF+ extract below:

```
      ...
      USE YOMHAL
      ...
!HPF$ PROCESSORS PROCS(NPROC)
!HPF$ DISTRIBUTE(INDIRECT(MAPGP),*), HALO(NSLHALO(I)%INDEX,*)
!HPF$+ ON PROCS(I) :: ZSL1
      REAL ZSL1(NGPTOTG,NFLDSLB1)
!HPF$ DISTRIBUTE(INDIRECT(MAPGP),*) ONTO PROCS :: ZSL2
      REAL ZSL2(NGPTOTG,NFLDSLB1)
      ...
      DO I=1,NSTOP
        ...
!HPF$ UPDATE_HALO :: ZSL1
        ...
        DO JSL=1,ISL
          ...
!HPF$ INDEPENDENT,NEW(JLEV,Z1,Z2,Z3,Z4,Z5,Z6),ON HOME(ZSL2(J,:)),
!HPF$+ REDUCTION(ZSL2)
          DO J=1,NGPTOTG
            DO JLEV=1,NFLEVG
              Z1=ZSL1(NSLIND(1,J,JSL),JLEV)
              Z2=ZSL1(NSLIND(2,J,JSL),JLEV)
              Z3=ZSL1(NSLIND(3,J,JSL),JLEV)
              Z4=ZSL1(NSLIND(4,J,JSL),JLEV)
              Z5=ZSL1(NSLIND(2,J,JSL),MIN(NFLEVG,JLEV+1))
              Z6=ZSL1(NSLIND(3,J,JSL),MAX(1,JLEV-1))
              ZSL2(J,JLEV)=ZSL2(J,JLEV)+
     +          (Z1+Z2+Z3+Z4+Z5+Z6)/(6.*ISL)
            ENDDO
          ENDDO
          ...
        ENDDO
        ...
      ENDDO
      ...
```

```
      MODULE YOMHAL
        TYPE PHALO
          INTEGER, POINTER    :: INDEX(:)
        END TYPE PHALO
        TYPE (PHALO),POINTER :: NSLHALO(:)
      END MODULE YOMHAL
```

The HPF+ kernel was also modified to run with the pghpf compiler (release 2.2). However, as this release did not support INDIRECT distributions (it is basically a HPF 1.1 compiler), the distribution for grid-point space was changed to BLOCK. Other changes were made to remove the halo module and associated references and associated data structure allocations. This was necessary as the compiler complained that the use of POINTER in a derived type was not supported. The pghpf compiler also required HPF distribute directives to follow the declaration of the array being distributed, which was unnecessary with VFC.

Problem	Version		1p	4p	16p
T63	SERIAL	ms	1,036.8		
		s/u	1.00		
	MPI	ms	1,103.8	276.4	104.0
		s/u	0.94 (1.00)	3.75 (3.99)	9.97 (10.61)
	HPF+	ms	14,806	3,857	1,255
		s/u	0.07 (1.00)	0.27 (3.84)	0.83 (11.80)
	PGHPF	ms	40,260	12,057	4,163
		s/u	0.03 (1.00)	0.09 (3.34)	0.25 (9.67)
T106	SERIAL	ms	3,944.9		
		s/u	1.00		
	MPI	ms	4,297.1	1,138.2	341.3
		s/u	0.92 (1.00)	3.47 (3.78)	11.56 (12.59)
	HPF+	ms	not enough	16,232	5,160
		s/u	memory	0.24 (1.00)	0.76 (3.15)
T213	SERIAL	ms	16,410.8		
		s/u	1.00		
	MPI	ms	not enough	4,308.2	1,187.6
		s/u	memory	3.81 (1.00)	13.82 (3.63)
	HPF+	ms	not enough	not enough	18,354
		s/u	memory	memory	0.89 (1.00)

Table 4: Comparison of run times (in millisecs per time step) and speedups for the versions of the SL kernel for different numbers of processors on a QSW CS2

The performance of the SL kernel is summarised in Table 4. As with the other ECMWF kernels, runs were performed on 1, 4 and 16 processors.

Figure 5 shows graphically the performance of the message-passing and HPF+ kernels at T63 resolution. The message passing version showed scaling of 10 to 14 on 16 processors for different problem sizes. It should be noted that the halo data volume does not reduce in proportion to a corresponding reduction in processor grid-points. This non-scaling issue and high communication to compute ratio mainly explain the observed performance for the message passing kernel.

Further, the HPF+ version has improved by a factor of 6 to 7 times over the previous version, which is attributed to the use of the HALO capability. In comparison the pghpf (v2.2) kernel ran about 3 times slower than the HPF+ version.

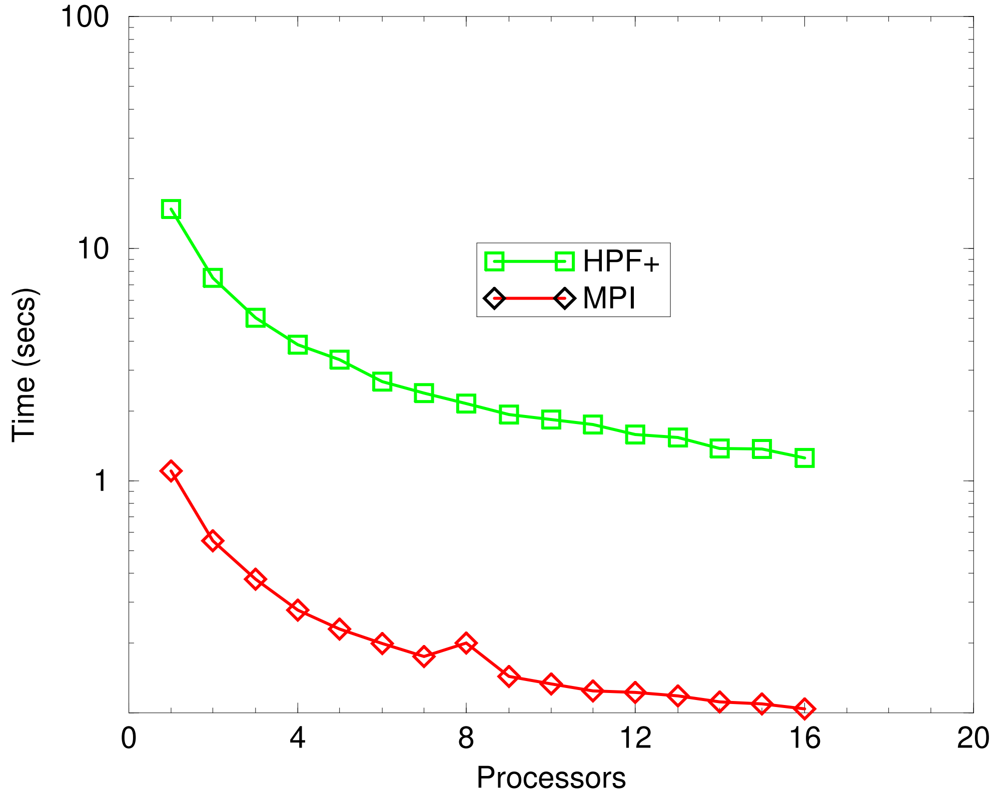

Figure 5: SL kernel performance

While the VFC improvement represents significant progress, the MPI version is still many times faster. This is not of great concern as the real SL interpolation routines used in the IFS have a computational intensity at least an order of magnitude greater than the much simplified SL kernel.

5 Summary

The VFC compiler has demonstrated that the HPF language can be extended to provide the functionality and performance required for the ECMWF IFS forecast model. In particular, the HPF+ language extensions for communication schedule reuse and halo support have been shown to be both easy to use and provide acceptable performance. Schedule reuse has been used in the TS and LG kernel, and has typically resulted in performance improvements of 5 to 6 times for the associated INDEPEN-

DENT loops. Whereas the HALO capability has been used successfully in the SL kernel.

It is also clear that the HPF-2 GEN_BLOCK & INDIRECT distributions and HPF_LOCAL extrinsic mechanism are equally important for the IFS. These features are classed as approved extensions in the HPF-2 standard, which means that it is left up to the compiler developer to decide whether to incorporate them. This presents a problem for application developers, as they have to decide whether to tailor their programs for each HPF compiler, or use a common reduced subset of the language. Hopefully these issues will be resolved as HPF compiler developers implement more of the present HPF-2 standard, and the HPF language is further enhanced by features such as those implemented in HPF+.

An important focus of the HPF+ project has been the support of nested INDEPENDENT loops in the context of multi-dimensional distributions of arrays. This has been important for all the IFS kernels, and much progress has been made to remove compiler restrictions as these kernels have evolved. Some restrictions still exist, however they can often be programmed around. This is probably the most annoying feature of programming in HPF, and much time can be spent trying to get INDEPENDENT loops to parallelise successfully. This should be compared with programming for message passing where the paradigm imposes little restrictions on Fortran 90 (communication of types being an obvious one). The counter arguement of course is that coding message passing library calls and being responsible for data partitions are equally tiresome.

It should be noted that VFC is implemented as a pre-processor that accepts HPF/HPF+ programs and generates Fortran 90 source code to be compiled by a Fortran 90 compiler. Included in the VFC generated code are calls to a VFC runtime library which are resolved at link time. This approach should be contrasted with message-passing programs which are simply compiled by a Fortran 90 compiler and linked with your favourite message-passing library, such as MPI. The VFC approach clearly has some disadvantages. The generated code is typically very difficult to understand due to its size and sometimes results in extended compilation times. An example of this can be seen with subroutine stepo3.f90 in the LG kernel, where serial compilation (f90 -O3) takes 23 secs on the QSW CS-2 for the 300 source lines. VFC compiles the same source in under 2 seconds, but generates 1400 source lines which takes 880 secs to compile (f90 -O3), a factor of 38 times longer. Fortunately very few routines are like this, but we don't need many for compilation time to be a serious problem.

Another problem occurs when we want to debug HPF+ programs. It is not possible today to debug the original HPF+ source. That leaves the voluminous VFC generated source which is unreadable by mere mortals. What do we do to resolve this? We use good old-fashioned print statements in the original source. This works in the global context of HPF programs, but requires writing to separate units in the HPF_LOCAL context.

Parallel I/O has not been addressed in the HPF+ project and equally in HPF standardisation efforts. This is a serious disadvantage for HPF, as the MPI-2 standard now supports parallel I/O which has been incorporated successfully into the production IFS. A possible solution for HPF may be to use this same interface within a HPF_LOCAL context.

Issues such as these would need to be resolved before ECMWF could consider an IFS port to HPF, or writing new applications in HPF. Nevertheless, the HPF+ project has made a big step towards such a port being practical.

6 Acknowledgements

The author would like to thank the other members of the HPF+ project for the many fruitful discussions on HPF and to the staff at the Vienna Centre for Parallel Computing for keeping the QSW CS-2 alive during the project evaluation periods. Special thanks are due to Siegfried Benkner, Viera Sipkova and other members of the VFC team in Vienna for making VFC a useable HPF compiler, and being responsive to the requirements of the application partners.

References

[1] Benkner, S. : *HPF+ : High Performance Fortran for Advanced Industrial Applications*, Proceedings HPCN'98, Amsterdam, April 1998, Springer Verlag (to appear)

[2] Benkner, S., Sanjari, K., Sipkova, V., and Velkov, B. : *Parallelizing Irregular Applications with the Vienna HPF+ Compiler VFC*, Proceedings HPCN'98, Amsterdam, April 1998, Springer Verlag (to appear)

[3] Dent, D., Mozdzynski, G. : *ECMWF Operational Forecasting on a Distributed Memory Platform : Forecast Model*, Proceedings of the Seventh ECMWF Workshop on the use of Parallel Processors in Meteorology, G-R Hoffmann and N. Kreitz eds, World Scientific, 36-51

[4] Barros, S. R. M., Dent, D., Isaksen, L., Robinson, G., Mozdzynski, G. and Wollenweber, F. : *The IFS Model: A parallel production weather code*, Parallel Computing **21** (1995), 1621-1638

[5] Dent, D., Isaksen, L., Mozdzynski, G., O'Keefe, M., Robinson, G. and Wollenweber, F. : *IFS Model – Performance Measurements*, Proceedings of the Sixth ECMWF Workshop on the use of Parallel Processors in Meteorology, G-R Hoffmann and N. Kreitz eds, World Scientific, 352-369

[6] Ritchie, H., Temperton, C., Simmons, A., Hortal, M., Davies, T., Dent, D. and Hamrud, M. : *Implementation of the semi-Lagrangian method in a high resolution version of the ECMWF forecast model*, in Monthly Weather Review 123 (1995), 489-514