Esprit 1993

High Performance Fortran Tools (HPF-Tools)

Proposal, Part 2

Contents

1 Summary **1**

2 Project Objectives **2**

3 Project Contribution to Esprit Aims and Objectives **3**

4 Dependencies and Relationships **4**

5 Baseline and Rationale of the Project **5**
5.1 Baseline 5
5.2 Rationale 7

6 International Standards **8**

7 Description of the Project **9**
7.1 Workpackage 1: HPF program development tools 9
7.2 Workpackage 2: HPF visualisation and graphics 16
7.3 Workpackage 3: HPF optimisation 21
7.4 Workpackage 4: Validation and evaluation 26
7.5 Workpackage 5: Project management 28
7.6 Risks 28

8 Project Management **29**

9 Workpackages **31**
9.1 Workpackage 1: HPF program development tools 31
9.2 Workpackage 2: HPF Visualisation and Graphics 31
9.3 Workpackage 3: HPF optimisation 32
9.4 Workpackage 4: Validation and evaluation 32
9.5 Workpackage 5: Project Management 32

10 Workpackage interdependencies and milestones **33**

11 Major Deliverables and Review Points **41**
11.1 Workpackage 1 41
11.2 Workpackage 2 42
11.3 Workpackage 3 43
11.4 Workpackage 4 43
11.5 Workpackage 5 43

12 Other dependencies **44**

1 Summary

The recently completed High Performance Fortran (HPF) informal standard for parallel programming promises to revolutionise the use of multi-processor architectures, especially distributed-memory MIMD systems. It is based on a simple and familiar programming language and offers wide portability, a simple migration method for existing Fortran codes, and the promise of high performance, so there is every chance that it will rapidly become one of the foremost languages for scientific and engineering applications on such platforms. Indeed, the lack until now of a programming language with these desirable features for DM-MIMD architectures has been the main reason for their low user acceptance, despite their potentially high performance and cost effectiveness, so the future success of these architectures is strongly dependent on the success of HPF.

However, writing efficient HPF code is by no means a trivial task. The programmer will need a good understanding of his program and of the HPF execution model in order to map data effectively and avoid programming practices that severely impair efficiency. Indeed, the high-level nature of the language means that it will be very easy to write hugely inefficient code. This suggests that automated support and guidance would be particularly beneficial for HPF program development. Another obstacle for HPF users in the near term will be the lack of a supporting development environment including execution visualisation, libraries, etc. Therefore, the aim of this project is to develop tools and software for supporting HPF programming.

In particular, the project will develop tools to help transform 'dusty deck' Fortran programs into efficient HPF. These include tools to identify and help replace 'old, deprecated' Fortran programming features that inhibit efficient parallel execution, to replace sequential code by more efficient (and clearer) data parallel array syntax and calls to intrinsic and standard library procedures, and to assist the data mapping process. As well as assisting HPF program development, these tools will improve the quality and efficiency of the resulting HPF program, which is a crucial requirement for the success of HPF.

The project will also develop tools for analysing, monitoring and visualising HPF program execution, and a library of advanced graphics routines that can be called directly from HPF programs and operate efficiently on distributed data.

Again addressing the issue of efficiency, another theme of the project is to investigate HPF compiler optimisation issues, which are largely unaddressed at the present time since HPF is still very much in its infancy.

The tools will be 'packaged' with an advanced, professional graphical user interface, and integrated into a programming environment developed in an earlier Esprit project (GIPE II) that allows uniform access and easy interoperability of all of the tools, and the easy addition of new tools as they become available. In addition, 'unpackaged', separately operable versions of the tools will be available.

All of the tools will be generic to all architectures supporting message-passing, and the program development tools will generally be applicable to all architectures.

Finally, an integral part of the project will be the evaluation of the tools and software on real applications by an independent member of the consortium (i.e. one not involved in tool development), providing feedback to the tool development.

It is not proposed to develop an actual HPF compiler in this project (which is already being undertaken in other Esprit projects), or to tie any of the tools to a particular compiler. However, the project will import (Subset) HPF compiler technology that was developed elsewhere, as a basis for tool development.

2 Project Objectives

The principal objectives of the project are as follows:

1. To develop tools to assist the conversion of standard Fortran to HPF

   These include new tools for bringing "dusty deck" Fortran to a normalised form; for semi-automatic insertion of HPF data-mapping directives; and for replacement of in-line user code by HPF intrinsic and library functions. These and other project tools will have a uniform modern user interface.

   As well as assisting HPF program development, these tools will improve the quality and efficiency of the resulting HPF program, which is a crucial requirement for the success of HPF.

2. To develop performance monitoring and visualisation tools for HPF programs.

   Performance visualisation has a proven track record in tuning message passing code, and is likely to be a key enhancement in HPF programming environments too. Since HPF will be available on a wide variety of DM-MIMD architectures, portable, system-independent tools are necessary for the analysis of the program behaviour.

3. To develop an HPF graphics library.

   This will be a library of efficient graphics routines for run-time use, which can be called directly from an HPF program. It will contain a wide range of functions for 1-d, 2-d and 3-d visualization, and the routines will be designed to be efficient on DM-MIMD computers where the data is assumed to be distributed among several processors.

4. To develop HPF optimisation techniques.

   Under this heading comes a tool for HPF source-to-source optimisation, based on an extensible data base of program transformations, and one supporting user optimisation of the intermediate Fortran code produced by HPF translators. We also include a component of work on optimised translation schemes here, supporting other tasks.

5. To build an open ended programming environment in which separately designed tools can cooperate, and a modern client/server man-machine interface gives a uniform and intuitive access to the system's functionalities. The resulting environment will bring unprecedented support to the task of building HPF scientific software.

6. To test and evaluate this software.

3 Project Contribution to Esprit Aims and Objectives

This project addresses the 'Basic High Performance Computing (HPC) tools and technologies' sub-domain of the Esprit Work Programme for 1993-4, and in particular task 3.5. The objectives of this sub-domain are described as:

> *To improve and consolidate the environment in which software and applications for HPCN systems are developed, ...*

The HPF development and visualisation tools are addressed specifically to this aim. They will will improve any programming environment for HPF.

> *...to encourage standardisation and common practices amongst a widespread group of vendors and users.*

The creation of tools for HPF encourages its widespread use and so its standardization.

> *Architecture-independent programming, ease of programming, general purpose, and predictable high level of performance will be the main focus.*

The optimisation workpackage tools will develop address performance issues, in a largely platform-independent manner.

> *To provide generic software which eases the use of a wide range of applications. To continue to stimulate the development of next generation HPCN hardware and software in accordance with user requirements.*

The evaluation and validation work will ensure that these goals are met.

The aims of task 3.5 specifically are:

> *Development of methodology, associated software tools and supporting standards which will rationalise the development of software for parallel systems and satisfy the production requirements of commercial and technical applications.*

Rationalising the development of software for parallel systems is the very reason for the existence of a company like Connexité . The tools that we develop must scale up to the requirements of commercial and technical applications. The tools's users are not necessarily programming language or system architecture specialists. The development must be guided by an intuitive interface and alternatives must be presented in a simple manner.

Portable languages like HPF are best supported by portable tools for runtime analysis. Thus architecture independent programming is supported. The ease of program development is especially supported by visualisation tools which are easy to learn and support the programming and analysis process substantially. Moreover application independent tools provide generic, general purpose software. Since they support a specific programming paradigm, the environment of HPCN systems is improved.

Run-time graphics is an important tool in a large number of fields. Graphics has been widely used in both program and algorithm debugging. Parallel graphics will thus be an important tool which will rationalize the development of software for parallel systems.

4 Dependencies and Relationships

This project will inherit considerable technical experience in building programming environments from the Gipe and Gipe II Esprit projects, as well as the experience of Connexité (a spinoff of Inria) in marketing Fortran development and reverse-engineering tools.

Technology from an HPF programming environment that is being developed by BES and partially funded in a Eureka project will be imported into the project. This will form a basis for the development of source-to-source transformation tools (task 3.3, source code optimizer, and 1.3, intrinsic/library call generator) and an HPF intermediate code optimiser (task 3.4).

Southampton University will provide as input to the project a subset HPF compiler (or rather a translator from HPF to Fortran 90 plus message-passing). This is based on modification of a system called 'ADAPT', whose development was started in the Espirt PUMA project. It will be used as the basis for HPF compiler optimisation studies in task 3.1, and in project years 2–3 it will be used to implement trace generation for the visualisation work in WP2 (tasks 2.2 and 2.3). The front end of this tool will also be used in the development of a 'data mapping assistant' in task 1.2 in project years 2–3.

Southampton University will also provide as input to the project an inter-procedural analysis tool for Fortran called 'IDA', which is currently under development in the Esprit CAMAS project and will be completed by September this year. This will be the basis for further development in task 1.1, to create an 'inter-procedural normaliser'.

The 'ParViz' project at IBM Bergen Environmental Centre will provide initial input to this project. ParViz was developed for message passing computers and has worked with Trollius and PVM on both a Transputer based parallel machine (IBM's Victor, experimental machine) and networks of workstations.

Software arising from HPF-Tools will be directly useful to other projects, especially those related to parallel porting of software. In addition, the possibilities for use as training aids in the European COMETT Training programs are numerous.

This project does not contain a component for monitoring and influencing the further HPF standardisation activities planned for next year (HPF-2). However, this is the domain of another Esprit proposal called PROPS, and it is planned that 'HPF Tools' would keep abreast of HPF-2 developments via the involvement of two partners, Southampton and Telmat, in both projects. The proposed PROPS project, which is targetted to the expansion of the HPF specifications, would also benefit from the detailed information emerging from 'HPF Tools', for example in the areas of HPF performance issues, programming styles, optimisations, etc.

5 Baseline and Rationale of the Project

5.1 Baseline

Fortran has always held a dominant position in scientific programming. Relatively unambitious in scope compared with more modern programming languages, its simplicity nevertheless lends itself naturally to automatic optimisation. A large subset of typical scientific and engineering applications do not essentially depend on features like pointers and contemporary data encapsulation techniques. Their absence has facilitated extensive static analysis and consequent transformation. This, in turn, has enabled successful automatic vectorisation of sequential code, and more limited success with semi-automatic parallelisation (cf *Vast*, *KAP*).

In an independent line of development, various vendor-specific data-parallel extensions to the Fortran have been notably successful in exploiting SIMD architectures. The outstanding contemporary example is perhaps Thinking Machines' *Connection Machine Fortran*.

Recent years have seen a number of attempts to combine features from these two threads (vectorization and data-parallelism). These systems take Fortran 77 programs extended by directives prescribing data distribution and alignment. Typically, they target distributed-memory MIMD rather than SIMD hardware. Such systems can be viewed as tools for automating the coding of an emergent paradigm in scientific MIMD computation—Single Program Multiple Data (SPMD) programming. They relieve the programmer of the need to explicitly insert communication. Archetypical examples are Rice University's *Fortran-D*, and *Vienna Fortran*.

The focus of these efforts has shifted with the acceptance of the Fortran 90 standard, which contains explicit syntax for expressing data-parallel computation. This facilitates the extraction of parallelism—because the programmer can express parallelism explicitly, there is a reduced requirement to depend solely on extensive static analysis for isolating parallelism. (Simultaneously, some of the limitations to Fortran mentioned above have been relaxed). However Fortran 90 does not provide any support for programmer-defined *placement* of data.

In the last year the *High Performance Fortran Forum* (HPFF), a coalition of many leading industrial and academic groups in the field of parallel processing, has established an informal standard called *High Performance Fortran* (HPF) for extensions to Fortran 90 for data-parallel computations on multi-processor architectures. In particular, HPF includes directives for specifying, at various levels of abstraction, how program data should be distributed over virtual or actual processor memories. The standard is supported by a majority of vendors of parallel hardware, including Cray, DEC, Fujitsu, HP, IBM, Intel, Maspar, Meiko, nCube, Sun and Thinking Machines. Several vendors have already announced their intention to develop HPF implementations, including hardware vendors DEC, Intel, Maspar, Meiko, Thinking Machines, and compiler vendors ACE, APR, KAI, Lahey, NA Software, Portland and PSR.

An official subset of HPF has been defined to facilitate early implementations. It is expected that the first true Subset HPF implementations, implemented as translators to Fortran plus message-passing, will appear at the end of 1993 or start of 1994.

The consortium of this project brings together an impressive range of experience relevant to programming in HPF.

Since 1989, the University of Southampton has been involved in the research and development of a system called *ADAPT* ("Array Distribution Automatic Parallelisation Tool") for automatically transforming Fortran 90, supplemented by data distribution declarations, to code for distributed memory message-passing architectures [Mer91]. This research started in the Esprit PUMA project, and has established Southampton as a leading group in the field of research into HPF-like languages and their implementation.

The input language for this system, *Distributed Fortran 90*, may be considered, alongside *Fortran D* and *Vienna Fortran*, as a precursor of HPF. Ours was the first research to study the implications of Fortran 90 in this context, others having concentrated on Fortran 77, and a technical report of this research was one of a few major input documents to the HPF activity [Mer92].

Southampton has had a significant input and influence on the development of HPF, through an active involvement in the HPFF group supported by the Esprit PPPE project. Southampton's contributions to HPF include the `PURE` procedure mechanism, which relaxes some of the restrictions of an otherwise predominantly SIMD programming model.

Connexité was founded in January 1992 to provide a range of products and services aiding in the development of scientific applications and the management of large-scale Fortran applications. Together with its founding partners INRIA and SIMULOG, it boasts 120 man years of research in interactive programming environments. Their existing products include the FORESYS (FORtran Engineering SYStem) line.

INRIA's expertise in programming environments began with the Mentor system and continued under the Esprit program with the Gipe projects, "Generation of Interactive Programming Environments", Gipe (1984–1989) and Gipe II (1989-1993). In the area of Software Development Environments, the major international conference is that of the ACM SIGSOFT SDE conference, which is held every two years. Gipe II results, including system architecture, programming environment techniques, and intertool protocol, have been presented at the last three SDE conferences.

While working for an American company, leading staff at BES designed and implemented a complete Fortran 90 to FORTRAN 77 translator. In conjunction with native FORTRAN 77 compilers, the translator provides a Fortran 90 programming environment portable across a wide range of computers. Currently BES is participating in a Eureka project. There it is responsible for producing an HPF programming environment based on a translator. The programming environment will be complemented with a source-level debugger and a profiler.

Siemens, who will take a leading role in the HPF "performance visualisation" tasks of the project, have previously implemented a visualisation and control environment called VISIT [IW93] for DM-MIMD programming. This provides a sound platform for visualising parallel programs, as an adjunct to optimisation. Extensive use of VISIT during the Esprit II project EDS demonstrated how software development may be rationalised by the use of adequate tools. Their close collaboration with Ken Kennedy and his group (a driving force behind HPF and state of the art parallel Fortran environments) will be of considerable benefit in their role of developing an integrated visualisation environment for the HPF community. Their associated partner TUM brings to the project long term experience in trace based monitoring [EA88] and trace compression. Southampton also has considerable experience in the area of performance visualisation systems, gained partly during the Esprit GENESIS project, within which they developed tools to transform between different trace file formats, as well as a generic visualisation tool based on ParaGraph.

IBM BSC have been involved in parallel computing, graphics and large scale scientific computing since their foundation in 1986. Their graphics research, which forms the baseline of their contribution to this project, was driven by the need to visualize the variables of models used in Environmenal Sciences. A post-processor graphics environment, ADVIZE (ADvanced VIsualiZation Environment), came out of this work. ADVIZE is an interactive 3-d visualization environment for analyzing and presenting scalar and vector data. It has been used by a number of international labs. They have parallel computing experience with both Shared Memory and Distributed Memory computers.

Finally, Telmat Informatique have been involved in the design and commercialisation of parallel systems since 1985. Those years of designing and constant refinement have provided them with a strong appreciation of the needs of their end-users in terms of programming tools. This puts them in a strong position to effectively evaluate and provide user-oriented feedback concerning the tools produced in the rest of the project.

5.2 Rationale

HPF is a significant step forward from explicit SPMD programming with message-passing, towards a style of programming familiar to the ordinary Fortran programmer. On the one hand quite general parallel algorithms can be encoded, but communication is managed entirely transparently by the compilation system. On the other hand, the language has been designed with a view to permitting direct message-passing *implementation* on distributed-memory systems, generally avoiding the overheads of simulated shared memory.

However, writing efficient HPF code is by no means a trivial task. The programmer will need a good a understanding of what the HPF mapping directives mean, in order to use them effectively. Indeed, the high-level nature of the language means that it will be very easy to write hugely inefficient code. Deceptively simple programs can translate to code involving enormous amounts of communication. The situation is aggravated by the fact that early implementations are unlikely to be highly optimising—implementors are bound to concentrate on providing the prescribed parallel semantics, before they turn to optimisation.

The project proposes to provide a set of tools for easing the transition to HPF, and tuning and debugging HPF codes.

6 International Standards

All the tools will comply with the HPF standard and their availability will encourage the use of HPF.

The visualisation tools will exploit where relevant, MPI, the emerging standard for message passing interfaces. For display, OSF/MOTIF will be adopted.

The graphics library will be initially written with X-windows/Motif. Future development could include Open GL and Phigs.

The work on tool interfaces also eploits OSF/Motif. The tool communication protocol will exploit WAIS and an industry client/server protocol (e.g., Tooltalk, CORBA).

7 Description of the Project

This section describes each Workpackage in turn, and concludes by presenting the main risks of departure from the proposed workplan and their consequences.

7.1 Workpackage 1: HPF program development tools

It is generally recognised that transforming an existing Fortran code into an efficient HPF program is not simply a matter of inserting appropriate data mapping directives. Fortran programs often contain features that, although they can be supported by HPF, can potentially lead to monstrous inefficiency. Particular examples are:

- *storage associations* (connections between sets of variables due to equivalences, or disagreement of common block declarations in different program units).
- *sequence associations* across procedure boundaries (e.g. passing an array element to a dummy array, or shape disagreement between actual and dummy arguments);

The presence of these features in an HPF code obviously greatly impairs the data mapping possibilities, and can *enormously* increase the amount of communication and reduce the potential for concurrency. Another consideration is that variables subject to such associations *must* be explicitly identified by `SEQUENCE` directives in an HPF program, this being an example where a legal Fortran program must be modified to make it a legal HPF program.

It is clear from these considerations that in general some rewriting of Fortran codes may be necessary before they can be converted to efficient HPF. Indeed, the HPF standard acknowledges this, saying that the HPF principle is to 'rewrite once' to produce a code that is portable across all architectures. It is also clear that this code conversion would be facilitated, and its quality enhanced, by software assistance.

The aim of this workpackage therefore to develop tools that will assist the user in converting a 'dusty-deck' Fortran program into an efficient (and elegant) HPF program.

It is useful to decompose this process into two stages, as shown in Figure 1:

1. The 'dusty deck' program is first transformed into an 'improved' style which is more suitable for conversion to HPF. The improved style would make extensive use of the new features of Fortran 90, both to avoid the pitfalls inherent in the restrictions of Fortran 77 (e.g. storage and sequence associations, which may be replaced by new 'cleaner' Fortran 90 features), and because many of the new features can assist data-parallel execution and HPF implementation (e.g. array syntax and intrinsics, explicit procedure interfaces).[1]

   The improved code is basically still standard Fortran (-90). It may optionally include a few HPF syntax extensions (i.e. `FORALL` statements and constructs and `PURE` procedures) which provide enhanced facilities for express data parallel operations, as well as references to HPF standard library procedures, but it does not include any HPF data mapping directives.

2. The 'improved' Fortran program is then transformed into an efficient HPF program by adding appropriate data-mapping directives, etc.

There are several advantages to this two-stage transformation:

[1] N.B. Task 4.1 will investigate the relative benefits of these and other transformations, which will feed-back to this workpackage.

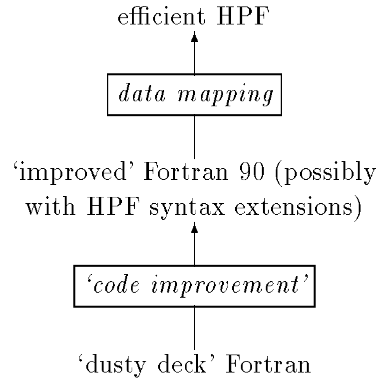


Figure 1: Transforming 'dusty deck' Fortran into efficient HPF

- In general the 'improved' program is a more suitable basis for conversion to HPF than the original (i.e. it will give rise to a more efficient HPF program) on *any* architecture. Therefore code 'improvement' need only be done once for all architectures, while the optimal HPF data mapping tends to be more machine specific.
- It will also be cleaner, clearer and more concise. (E.g. experiments in converting some 'Genesis' benchmarks to Fortran 90 array syntax have shown a reduction in code size of up to 50% and a large increase in clarity). In other words, the optimisations proposed here are all 'clean', in the sense of improving the code legibility (as opposed to some machine-specific 'dirty' optimisations which make the code illegible).

  Therefore, the user may well wish to keep the 'improved' version as the 'master' copy of the program. If HPF syntax extensions are avoided, the improved version will still be standard Fortran 90 and totally portable.
- The HPF data mapping will generally be affected by the source code 'improvement', e.g. the removal of storage associations will influence the mapping of arrays.

The tools to be developed in this workpackage will perform the following functions:

1. 'interprocedural normalisation', e.g. the identification and removal of sequence and storage associations as described above (task 1.1);
2. replacement of code by references to Fortran 90 and HPF intrinsic procedures, and HPF and other standard library procedures (task 1.3);
3. replacement of sequential code by data parallel code, e.g. Fortran 90 array syntax or HPF `FORALL`s (task 1.4);
4. semi-automatic HPF data mapping (task 1.2).

The first three fall into the 'dusty-deck' → 'improved' Fortran category, while the last helps to automate the actual HPF data mapping process.

The workpackage will also develop (in tasks 1.5 and 1.6) a uniform graphical user interface for these and other project tools, and integrate them into a programming environment that allows them to cooperate easily and allows new tools to be added when they become available.

Task 1.1: Interprocedural normalisation

A tool will be developed to perform tasks that we call ‘interprocedural normalisation’. (We have invented this name, as we are unaware of any existing tool with a similar functionality). It will use interprocedural analysis to perform the following functions:

Sequence and storage association removal. It will locate all occurrences of sequence and storage association, and in the latter case also derive ‘aggregate variable groups’ as defined in chapter 7 of the HPF standard.

It will also suggest how these associations may be removed (e.g. by using dynamic arrays to replace the Fortran 77 technique of decomposing a single static ‘workarray’ into multiple ones via storage association). If the user declines the appropriate code modification the tool will be able to insert the appropriate `SEQUENCE` directives in the program and apply, or check compliance with, the consequent mapping restrictions.

Introduction of explicit procedure interfaces into the Fortran program and the introduction of optimisations consequent on this, e.g. changing all HPF mapping directives for dummy arguments to the ‘descriptive’ form. This is a useful optimisation, as it allows argument redistribution to be performed by the caller, avoiding the need for the argument mapping to be checked at runtime on entry to the procedure.

The tool will also check that procedure references are consistent with their definitions, and check conformance with Fortran’s anti-aliasing rules for arguments that are modified (which is necessary to preserve program correctness under the types of transformations performed by an HPF, or even a vectorising, compiler.)

Data subject to the sequence and storage associations *must* be identified by `SEQUENCE` directives in HPF, so this analysis is mandatory. These associations also imply restrictions on data mapping, and can *massively* degrade efficiency in distributed memory implementations, so their removal should constitute a major optimisation; indeed, this may well be the most beneficial code improvement for the purposes of HPF on such architectures. We do not know of any currently available tools to perform these functions (e.g. Forge-90 and its associated HPF implementation, XHPF77, aim to support such associations, albeit inefficiently, rather than removing them).

The tool will be based on further development of the ‘IDA’ inter-procedural analyser from the Esprit CAMAS project, and will be completed one year into the project.

Task 1.2: Data mapping ‘assistant’

This will be an interactive tool to assist the programmer in selecting suitable HPF mappings (i.e. alignments and distributions) for the data in standard Fortran (-90) codes, and then automatically generate the appropriate HPF directives. It may be regarded as a practical and feasible intermediate step towards the longer-term goal of fully automatic data mapping, which is still very much at the research stage (and is being explored in Workpackage 5 of PPPE).

As well as helping to convert standard Fortran programs to HPF, this tool might also serve as an ‘HPF tutorial’ tool, especially if supplemented by small sample codes.

It would perform the following functions:

Alignment generation. The tool will suggest array alignments based on inspecting the array accesses in the input Fortran program. It will return a number of candidate alignments with a weighting for each, where this could be determined, based on an estimate of the ratio of aligned to unaligned accesses.

Template generation. Given a program with its data alignments explicitly declared, this will extract a smaller number of HPF templates to align the arrays to, and automatically generate the required directives.

Distribution generation. Given alignments to templates, this will inspect the data access patterns to suggest suitable candidate distributions (i.e. for each dimension: block, cyclic, block-cyclic or undistributed).

Argument mapping propagation and procedure cloning. The tool will perform inter-procedural analysis to perform a number of functions, e.g.:

- For a given procedure, it could show the set of actual arguments and their mappings (when defined) associated with each dummy argument.

  It could also generate a number of clones of the procedure, each of which has its dummy arguments mapped identically to the corresponding actual arguments for a particular set of calls, and replace those calls by calls to the appropriate clone. This is a desirable optimisation for frequently executed procedures if the number of different actual argument mappings is sufficiently small, as it avoids the overhead of either run-time redistribution (if the actual argument has a different mapping from the dummy) or the calculation of data location (if the dummy inherits its mapping).
- Alternatively, data mapping could be propagated in the opposite direction—from dummy arguments to the corresponding actual arguments at call sites. This is the natural strategy when dealing with a heavily-used procedure that has an 'optimal' mapping for its dummy arguments.
- It can propagate the mapping of global data throughout the program.

The tool will require an HPF parser, which will be obtained from the Subset HPF compiler currently under development at Southampton, which will be imported into the project. The results of the research in task 3.1 on HPF compilation techniques will be used in determining optimal data distributions.

A 'static performance estimator' for standard Fortran programs could usefully be used in conjunction with the data mapping assistant. Such a tool is being developed elsewhere (e.g. the SPE in CAMAS), and it is likely that the latter could be imported into the project with relatively little modification. This would provide estimates of the execution frequency of each statement and procedure, thus indicating critical procedures, code sections, data objects and access patterns on which effort could be concentrated.

This might thus be imported into the project and integrated with the data mapping assistant (via the environment developed in task 1.6) towards the end of the project.

Task 1.3: Intrinsics / library call generator

The use of intrinsic and library procedures is a good way of both simplifying the programming of an application (complicate algorithms are implemented with just a procedure call), and of producing efficient code (the intrinsic and library routines are generally optimized for the specific architecture being used).

Fortran 90 greatly extends Fortran 77's set of intrinsics by adding a large set of intrinsic procedures for common array operations, and HPF adds a few more of its own. HPF also incorporates a standard library of procedures to perform many useful operations in data parallel programming, which are more complex than those covered by the Fortran 90 intrinsics (e.g. combining send, parallel prefix and suffix operations, sorting). Finally, the most common numerical libraries for Fortran (NAG, IMSL, ...) will have parallel HPF versions in the future.

The source-to-source transformation system developed in task 3.2 will be used to implement an intrinsics/library call generator. This system will recognize portions of code (patterns) that can be replaced by references to intrinsic or library routines, and will make the replacements.

Data-bases of patterns/actions will be provided for:

- the Fortran 90 array intrinsics;
- HPF intrinsics;
- HPF standard library procedures;
- some standard numerical library (NAG, IMSL, ...), one of which will be chosen according to its availability and possible benefits.

Task 1.4: Adapting 'Foresys' tools for HPF

'Foresys' (FORtran Engineering SYStem) is a scientific computing environment, generated from Centaur (see task 1.5), which is developed and marketed by Connexité . It is a group of tools to help users develop and maintain Fortran code. This includes tools for parsing, visualizing and interprocedural type-checking (source validation) of code. There are also tools that help to restructure code with Fortran 77 syntax (as well as numerous industrial dialects) into code with a Fortran 90 syntax, and tools that aid in the maintenance and parallelization of code, e.g. dependency graph generation.

This task will extend the existing Foresys tools to handle HPF features and directives/constructs. For example, in addition to generating Fortran 90 array syntax, the code restructurer will be extended to generate HPF `FORALL` statements and constructs, which may express more general data-parallel computations than standard array syntax can.

The work for this subtask would begin with adapting the parser and visualizer currently used to HPF. Also a tool called 'Forest' (FOrtran RESTructurer), which is a migration tool that eliminates old constructs such as `GOTO`s and arthimetical `IF`s, would be updated. Next we would move onto an HPF dependency graph generation. Finally, we would adapt the Foresys tools to the new intertool protocol (see task 1.6). This would result in a first product, 'Foresys/HPF', half way through the project.

Task 1.5: Building Modern Interfaces for Tools

The goal of this task is to provide modern interfaces to the separately designed tools. While all the tools can be operated separately, a successful programming environment for HPF must provide a system in which the tools can cooperate and the user interface provides a uniform and intuitive access to all of the tools' functionalities.

This task will build on experience with an environment generator called Centaur, which was developed under the Esprit Gipe and Gipe II projects. The programming environment technology uses a client-server approach, and the interface follows standards such as OSF/Motif and uses intertool communication techniques. The latter involves both control integration, whereby tools communicate in an event-driven style, and data integration, which allows the tools to exchange the necessary information in a standard manner.

This task is divided into three subtasks:

1. interfacing source manipulation tools (developed within the project);
2. interfacing run time tools (developed within the project), and
3. final integration and packaging of the HPF environment.

The first subtask deals with source manipulation tools developed elsewhere in the project. For code migration tools, one wants to see the original source code corresponding to a portion of generated code. This is particularly important, as the user is not the author of the generated code. The interfacing with the HPF source-to-source optimization tool (task 3.3) is particularly interesting. The user is not a specialist and thus it is not simply a matter of presenting a menu of transformations. The idea is to aid the user by presenting directly on the source code the places where the different transformations apply, allowing him to view the resulting transformation, and then perhaps finally applying such a transformation to the code. Note that an undo facility is important for this type of tool, and that the user is establishing a dialogue via the interface with the generator of the transformations. That is, the communication with the tool is more complex since it is a real dialogue, in contrast with handling messages generated by a typical batch tool (e.g. a compiler).

The work in this subtask is obviously dependent on the other work packages. It will begin with the first components available from the 'dusty desk tools', and the prototype source-to-source optimization tool. This will result in a prototype environment that interfaces with other tools within the project. We will also provide documentation for the user interfaces developed.

The second subtask deals with the run-time tools developed within the project. The work in this subtask will be centered around the 'HPF intermediate code optimiser' developed in task 3.4. In this case, one can imagine sending directives to the tool, e.g. for setting breakpoints. Here too there is an active dialogue between the interface and the tool.

The third subtask will perform the final integration and packaging of the final versions of the project tools, and produce accompanying documentation for the HPF environment (i.e. documentation for the interactive environment rather than the tools themselves). The end product of this task will be the completed HPF environment as a product.

Task 1.6: Inter-tool Protocol

Workpackage 1 develops a number of tools for HPF program development. Getting the various tools to communicate is an important task, e.g. to show the correspondance between the original and generated code when dealing with source transformation tools.

This task is intended to set up a communication protocol to be used between the various tools in the HPF environment.

Tools provide various services, such as transformations or analyses, on source code. The output of one tool may then be used as the input for another and so on. In this case, a modification to the source program will invoke a chain of transformations. Note that fragments of code and locations in the source code must therefore be expressed in a uniform manner that all tools understand.

We will specify the interface needed for control integration (i.e. requests, acknowledgements) between various tools, and for data integration (i.e. module names, code fragments, source code locations) when these tools communicate. The latter entails a HPF abstract syntax specification. It is not necessary for all tools to operate on abstract syntax trees as an internal data structure but rather it is necessary for each tool to be able to communicate with the others in terms of such trees. This is particularly important in an interactive setting.

The major advantage of this approach is that any tool encapsulated using such an interface can then be replaced by another tool that conforms to the interface. Also the approach is open-ended: new tools may be added.

We breakdown this task into 3 subtasks or phases:

1. A specification phase to document the protocol used for control integration. As mentioned above, this will include an abstract syntax specification for HPF. In addition, a technique is needed for indicating locations in the source code. We will also define the protocol used by various tools within the environment, i.e. requests and acknowledgements that the tools can receive or issue.

2. An initial prototype implementation will be developed, based on the Sophtalk system (a control integration system that was developed in the Esprit Gipe II project, No. 2177). Sophtalk uses an encapsulation technique similar to that of HP's Softbench and has successfully been used to integrate tools in programming environments in the Gipe II project. Here we will use the Sophtalk protocol to connect various tools for the HPF environment. This subtask will result in an initial product using the Sophtalk protocol.

3. We will examine the suitability of a more robust standard for intertool communication, be it Tooltalk, CORBA, or a new intertool communication system that emerges as a standard, and then implement the communication protocol using this standard. Finally, we will adapt the previous Sophtalk version of communication between tools to use the new protocol and integrate tools, which only become available at the end of the project. This would result in a more robust final product.

7.2 Workpackage 2: HPF visualisation and graphics

This workpackage consists of four tasks, three of which will together produce an HPF performance visualisation system, and the fourth of which will produce an HPF graphics library. The library will be able to be used by programmers to help visualise their data and, although no formal connection between its task and the others has been made in the workplan, it is envisaged that the exchange of ideas between groups working on different but related forms of visualisation will prove fruitful.

Visualisation of program execution is one of the most valuable means of program diagnosis and tuning. A number of performance monitoring and visualisation tools have already been developed for explicit message-passing programs executing on DM-MIMD machines (e.g. ParaGraph), and have proved to be a useful aid in optimising, debugging and generally understanding the behaviour of such programs. Similar tools will also be extremely useful for HPF programs. Since HPF does not contain explicit message-passing, the opportunity arises to visualise program execution at a higher level than that of existing tools. HPF implementations for message-passing systems will, however, ultimately generate message-passing code, and we therefore propose to develop two tools that enable programs to be visualised at three different levels of abstraction:

1. Array level
2. Process level
3. Message-passing level

At the higher level, program execution will be able to be visualised purely in terms of language-level distributed arrays, while at the lower level it will be possible to observe the actual pattern of message-passing produced by the code. The process level lies somewhere between the two extremes, and both tools will visualise aspects of it.

The visualisation tools will display a representation of trace data generated during program execution. For array-level visualisation, the trace must be generated by library calls embedded into the code by the HPF compiler, while for the message-passing level it suffices that the trace be generated by code within the message-passing library. The two approaches are thus complementary, with the higher level form of visualisation being easier to comprehend, but less portable. In contrast to existing visualisation systems for explicitly message-passing systems, it is unavoidable that the compiler itself must be instrumented if language-level visualisation is to be achieved. In addition, the compiler must produce a cross-reference file to relate trace data to the HPF program source. In order to simplify the task of instrumenting particular compilers we will produce trace and cross-reference generating libraries that are as generic as possible, and to demonstrate the system as a working whole, and to assess the effort involved in instrumentation, we will produce an instrumented version of Southampton's research implementation of a subset HPF compiler.

The relation between the visualisation tools and the other components of the visualisation system is shown in Figure 2. Programs are represented by rectangles, and their input and output data by ellipses, with arrows indicating the flow of data between programs. The rectangle representing an executable program is an exception, since it is also data. The dashed line on the arrow pointing to it indicates that it is the output data of the compiler, rather than the compiler being its input data. The diagram actually illustrates the case of array-level visualisation. The equivalent diagram for the message-passing level is very similar except that the trace generating routines are embedded within the MPI/PVM routines.

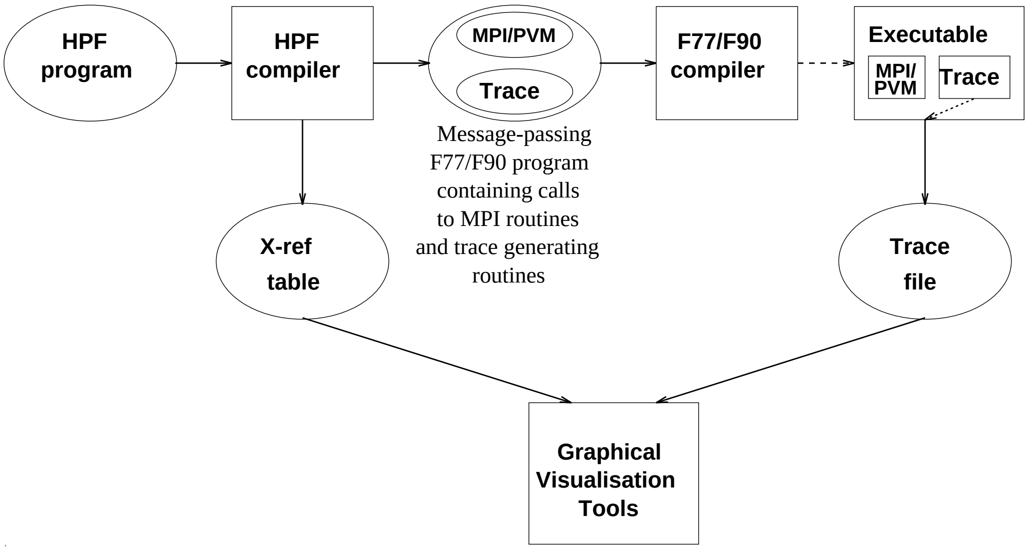


Figure 2: Components of the HPF Performance Visualisation System

Task 2.1: Trace and Cross-Reference Data

The principal aim of this task is to select trace definitions which are closest to emerging standards and to develop a library of basic trace generation and reading functions which supports those trace definitions.

For HPF monitoring we propose to adopt a "mixed" approach to trace generation, to support the different levels of abstraction of the visualisation tools. To visualise message-passing (task 2.3) only the message-passing library need be instrumented. To visualise at the array-level (task 2.2) the HPF compiler must insert trace generating code. Trace events can then represent global operations, which is conceptually simpler than trying to reconstruct global operation semantics from message-passing level trace data. Moreover, such a representation could significantly reduce the size of trace files, which is already an important consideration, and increasingly so as the speed of systems, and the number of processors in them increase.

In either case, in addition to the information in the trace, the graphical tools must be in possession of cross-reference data, relating that information back to the source code, if language-level visualisation is to be achieved. This cross-reference data is static information, and can be generated at compile-time. The precise definition of the type of cross-reference data required by the visualisation tools is intimately bound up with the choice of level of trace data to be recorded. Trace and cross-reference generating libraries will be provided to embody the design decisions that are made.

In addition to reducing the number of generated events, it is important to minimize the amount of trace data logged with the events. Minimizing trace data leeds to reduced overhead at run time (and less influence on the program run), reduced communication costs (for data collection), reduced disk usage, and faster trace interpretation. TUM will contribute their long term experience on trace based monitoring and trace compression from the ParMod project [Eic87, WSF91, Fri92].

In the first stage, the trace file formats offered already by the partners (e.g. EDS Lisp, VISIT) and undertaken by other groups (e.g. PICL, SIMPLE, self-defining records, ParMod) will be analysed. In the second stage, a trace format will be defined which is closest to emerging standards and which satisfies the requirements of HPF visualisation. Finally, a library of trace generation and reading functions will be implemented (using data compression algorithms) and integrated into the HPF tools supported.

Task 2.2: Array-Level Visualisation Tool

A tool will be developed that will present the programmer with performance data in a graphical form related to the constructs used at the source code level. This is the same approach taken by currently popular tools for message-passing programs, except that in the case of HPF the relevant constructs are distributed arrays, rather than message-passing calls. Thus the graphical displays used will be based on arrays and their alignment with templates, the distribution of templates onto abstract processors, and the allocation of abstract processors to physical processors. Animation and colour will be used to convey performance data relevant to an HPF program. This will include remote memory accesses and processor utilisation. Although remote memory accesses are implemented by message-passing, they will be represented in terms of array element accesses. For portability the tool will be based on the X Window System, which runs on systems available from a wide variety of vendors.

Since at present there are no tools of this type, there is no base of existing practice, and the first phase of this task will be concerned with designing a suitable set of graphical representations to be implemented. By the start of the project, if other tools have become available then the features they offer will be considered in the design. For example, it is expected that the Prism programming environment from Thinking Machines will incorporate some form of HPF visualisation. During the initial phase there will also be heavy interaction with the definition of the trace and cross-reference data being undertaken in task 2.1. In the second phase a prototype implementation of the design will be made. In the final phase an instrumented version of a subset HPF compiler will be produced, based on a research implementation available at Southampton, and will be used to evaluate the visualisation tool. In parallel with these two activities, a final version of the visualisation tool will be produced, incorporating ideas arising from the evaluation.

Task 2.3: Message-Passing Level Visualisation Tool

In general, HPF compilers will transform the HPF code to Fortran code with calls to a library containing functions for process generation and communication. Within this task the instrumented compiler of task 2.2 will be used, which produces code for the popular PVM communication library and later for the emerging message passing standard MPI. A tool will be developed to visualise at these levels of HPF program execution. It will be based on an existing visualisation and analysis tool VISIT [IW93], which will be provided to the project. It was also used with great success within the Esprit-II project EDS for parallel Lisp visualisation.

In the first stage, PVM/MPI will be annotated to produce VISIT events, and VISIT can be employed by the implementators of HPF for debugging and analysis. It will be specifically employable by task 3.4 to optimise message passing code. This task lets the programmer do fine-tuning on message passing level, a technique for which appropriate visualisations are essential in order to check the unchanged semantics of the program. There will be no need to make changes to the HPF compiler at this stage, as all trace data can be derived from the PVM/MPI level by annotating PVM/MPI, although in order to be able to relate the PVM/MPI information back to the HPF code, a restricted use of the cross-reference table produced by the HPF compiler will be necessary. The tool will offer, for example, the following visualisations: a tree of spawned processes, Pert charts with communication, specific displays for the language's communication and synchronisation constructs, load balancing, performance statistics. These visualisations are valuable for the HPF programmer to get a general feeling for the program performance. The visualisation can be done both in realtime and post mortem out of the trace file. Since VISIT proved to be easily adaptable to similar languages, other communication libraries can be hooked to VISIT by the respective partner.

The tool will evolve towards a higher-level HPF execution visualiser and will adapt to the trace format defined in interaction with task 2.2 by task 2.1. To achieve this goal, it will use cross-reference information provided by the compiler and runtime data from the runtime system. Together with the information from the PVM/MPI level, the tool will provide visualisations which are customized for dataparallel language executions. For example, the typical split-join behaviour of HPF programs can be presented to the HPF user for analysis. Visualisations typical for HPF programs, e.g. processor grids, will be integrated. Task 2.2 will instrument an HPF compiler and can thus provide information on typical HPF execution patterns.

In general, the programmer wants to have visualisations in order to see whether the program does the desired work in general, and to recognize imbalances and irregularities. In a further stage, the tool will be extended to allow the programmer to declare what kind of imbalances and irregularities he is interested in. Thus the 'normal' program behaviour can be cut out, and only the 'special', interesting points can be displayed and visualised. For example visualisations can be activated only for situations where there is a specified deviation (e.g. in runtime or nonlocal memory accesses) from the normal behaviour. This extension will heavily facilitate the evaluation of the tools developed in work packages 1 and 3. This will make the search for irregularities and performance bottlenecks much easier and allow for faster program tuning.

Task 2.4: HPF graphics library

Graphics is becoming increasingly important as the sizes and complexities of problems are increasing. The ability to grasp a large amount of data by using graphics is a tremendous aid to programmers.

There is a problem with using today's run-time graphics routines on DM-MIMD processors since all of the data we wish to view graphically must reside on one processor, whereas in the HPF model data are spread out among the processors, and for efficiency are kept in the same place as much as possible.

Due to the ratio of computing speed to communication speed (e.g. presently $\sim$ 50–100 double precision Mflops versus $\sim$ 40–50 Mbytes/sec for high speed connections between superscalar workstations) it is still quite costly for such computers to do much communication. If one wants to visualize large amounts of data located on separate processors during program runs it is important to reduce the amount of data transferred in order to optimize performance. This will still be true even when the communication bandwidth increases.

We propose to build a graphics library that uses the underlying communications routines to produce graphical output of data situated on separate processors while minimizing communication. The library routines will be callable directly from the HPF program, and will thus hide from the programmer the inter-processor communication needed to combine data. The library will do data reduction on the processors to reduce communication time and will use efficient algorithms to combine 3-D data properly for hidden and transparent surfaces.

The library will communicate with a 'Graphics Daemon', a process that is responsible for the collection of reduced data, combination and preparation of this data, and the display. The schematic diagram below illustrates this process. The programmer calls graphics subroutines on the top left of the figure. The graphics routines use the parallel graphics primitives and the communication routines. The graphics communication layer is one layer above the communication software layer in order to make the system easily portable. To go from one system such as PVM to another, for example, MPI, one need only change a small number of routines in one file.

$$
\begin{array}{ccc}
 & & \text{graphics server} \\
 & & \text{(hidden from user)} \\
\text{HPF graphics routines} & & \text{graphics system (X-windows)} \\
\downarrow & & \uparrow \\
\text{parallel graphics primitives} & & \text{server graphics layer} \\
\downarrow & & \uparrow \\
\text{parallel graphics communication} & & \text{graphics server communication} \\
\searrow & & \nearrow \\
 & \text{communication layer (PVM, MPI)} &
\end{array}
$$

The graphics server is sitting on the 'host' machine, i.e. the machine used to communicate with the X-windows graphics on the screen. Presently X-windows is being used as the graphics standard, however OpenGL and others will also be considered.

The HPF graphics library will be based on future development of the ParViz ('Parallel Visualization') system, which is based on PVM and X-windows. The library will contain functions such as setting colors and manipulating graphics windows. Different data can be displayed in different windows. The 2-d graphics functions will include basic functions such as lines, polylines, text, etc, and additionally higher level routines such as contours, fields/arrows, drifters, bars and different projections (rectangular, polar, etc) will be provided. The 3-d part of the package allows the display of 3-d surfaces with independent contours, 3-d lines, fields, drifters, shading, and volume visualization with transparent 'clouds' (single ray raytracing). Concurrent display of multiple grids (cartesian, spherical, etc.) will also be possible.

Owing to the background of the people involved in this task, the initial emphasis will be the types of displays used in environmental modelling (airpollution, ocean-circulation, toxic spread). However, these needs will coincide with a large group in the science community, and it is our aim to produce a general package for scientific and technical applications. The other partners in this consortium will be a great resource by giving us feedback on the direction of advanced graphics functions.

The devlopment will be performed in the following phases:

- By the end of the first year we plan to have a first prototype ready which includes a basic 2-d and 3-d components. This version will be based on X-windows graphics and PVM communication tools. The integration with HPF will also be planned. The communication layer must be made available for this to be done.
- The next year will be used to integrate the prototype with HPF.
- The third year will be devoted to advanced graphics routines and high level interaction routines, e.g. the ability to change view angle from the screen; a method of saving pictures for movie making; and making printer interfaces for colorgraphics printers.

7.3 Workpackage 3: HPF optimisation

Code optimisation has traditionally been the province of compilation systems. Recent years, however, have seen an emerging trend towards implementing very high-level languages (of which HPF is an instance *par excellence*) by source-to-source translation to other high-level languages. At the same time program transformation systems which perform optimisations at the original source level, have become increasingly prevalent. Hence, even without a specific commitment to compiler development, there is a niche for optimisation tools within the project. One task here, namely 3.3, is committed to source to source transformation *within* HPF. Another, 3.4, supports user optimisation at the intermediate code level (the message-passing code generated by an HPF translator). The first is, in principle, completely independent of the HPF compiler. The second involves limited integration with a compiler (translator).

Other workpackages (notably the tasks on language level performance visualisation) will ultimately involve *some* level of integration with an HPF compiler. Wherever such compiler dependence occurs, the intention is to clearly delimit and document the requirements of the compiler, so that appropriate "hooks" can ultimately incorporated in product compilers, developed outside the HPF Tools project. As an interim measure, existing experimental translators, imported into the project by partners, will be used as a testbed for supporting tools with compiler-dependent features. Task 3.1, on translation schemes, involves a component of "testbed" compiler support for other project tasks. Task 3.1 will also track and contribute to state of the art HPF translation schemes. The associated knowhow is needed within the project, because other tools cannot expect to generate efficient HPF code without some idea of the costs (for HPF, *strongly* dependent on the translation scheme) of particular language constructs.

Task 3.2 provides general support for source-to-source transformation tools, needed within this task and also in workpackage 1.

Task 3.1: HPF Translation Schemes

Although the main thrust of the project is on tools and other software for facilitating HPF program development, there will be one task on translation (compilation) of HPF. There are several good reasons for including such a component in the project.

- Tools such as the semi-automatic data-mapping tool (task 1.2) are strongly dependent on the translation scheme adopted by the target compiler. Depending on exactly how communication is managed and computation localised, different distribution schemes will be optimal. So, at the very least, the project will have to track the state of the art in HPF compilation strategies, to make sensible decisions in design of the program-development tools.

- Workpackages on language-level visualization and performance monitoring will ultimately require compiler integration. Availability of a research compiler within the project will serve as a testbed for this integration. Emphasis will be placed on performing the integration in a modular way, so that eventually the tools can be adopted by commercial compiler vendors with minimum modification to their products.

- HPF-like systems are still in their infancy, and there has been little study of optimisation issues. Research is urgently needed into efficient implementation of the complex HPF semantics.

Southampton expect to have a research implementation of Subset HPF available at the start of the project, which will form a suitable platform for further development within the project. The compiler will generate Fortran 90 output, with calls to MPI-compatible communication routines.

Specific areas of development in HPF translation to be addressed in this task include:

Calculation Placement. Develop methods to determine the optimal placement of sub-expression calculation, so as to minimise communication, minimise temporary storage, and increase potential concurrency. By suitably organising the placement of calculation it is possible to improve all of these aspects simultaneously.

Managing local and non-local data in loops In many cases, especially with problems involving some locality of reference, `DO` loops can profitably be split into parts operating on local data and parts involving communication. This may facilitate overlapping of communication and computation, and make calculations on internal data simpler (vectorisable?). The various proposed schemes for managing this loop reorganisation need to be assessed.

Layered Run-time Systems and Optimisation. In certain places in HPF code almost no information is statically available concerning the whereabouts of particular array elements. In this case data is accessed by very high-level run-time libraries (perhaps performing collective operations on distributed arrays), with a large performance penalty. At the opposite extreme, on sufficiently simple code an optimising compiler may be able to go straight to low level communication instructions (below the level of standard message passing interfaces). The "onion-skin" approach involves a layered run-time library, some representation of which is available to the compiler, and whose procedures can be "in-lined" and simplified level by level when appropriate static information available to the compiler. We propose to explore the feasibility and utility of this approach to communication management.

Accordingly this task will

- Track developments in the state of the art of HPF compilation.
- Investigate the issues described above, including experimental implementations of the various schemes.
- Provide testbed compiler support to other tasks requiring compiler integration.

Reports will be produced at yearly intervals.

Task 3.2: Generalised Fortran source-to-source transformation system

Many modern programming tools operate by taking as input a program in a high-level language and writing as output an equivalent program in the same language. Examples of such tools are source-to-source optimizers, vectorizers, parallelizers and in general any tools that instrument programs.

The program transformations those tools perform are traditionally hardwired in the code of the tool itself. This has several drawbacks:

- To change or add a transformation one needs to modify the code of the tool itself and recompile it.
- Incorporating program transformations as code within the tool itself is not a natural way to express them. The code that implements a given transformation can be quite obscure and error prone.

- Users cannot modify the behavior of the tool and in general it is quite complicated and laborious to put more knowledge into the tool.

An alternative approach is to define a high-level language specifically designed to express program transformations and then implement a generic tool that can interpret that language and apply the program transformations that it expresses.

BES are currently using this approach to implement an HPF to Fortran 90 translator in a project that aims to produce a programming environment for HPF, consisting of the afore-mentioned translator plus a profiler and a debugger. The system is still in its early stages, and targetted to some specific, simple transformations.

In this task we will generalize the basic system, making it applicable to more general program transformations.

The system consists of:

- a data-base of patterns to match (along with the actions to do upon finding each pattern);
- a tool that reads the data-base and the original source, looks for the patterns and performs the required actions.

For the purposes of this project the system will be extended and improved in three different ways:

1. **Extension and enhancement** of the basic system to perform the tasks required for this project. The system will need to recognize more complicated patterns, and to perform more complicated actions on finding those patterns. The work will involve extending the language used to specify the patterns and actions as necessary, and implementing those changes in the actual system.
2. **Conversion into an interactive system.** The basic system works as a batch system. An interactive user interface will be added, using the techniques and tools developed in tasks 1.5 and 1.6. This will allow the user to perform all tasks interactively and to see and act on the patterns and actions which the system performs. The concept of a hierarchy of data-bases will be introduced, and the user will be allowed to create his own data-bases.
3. **Optimisation of the system.** The main bottleneck of the system is in the pattern matching algorithm. One needs to compare each portion of the parse tree with a data-base of possible patterns to match. Those data-bases can be very large, and they will keep growing as more knowledge (pattern/action rules) is put into the system. To overcome this bottleneck the pattern matching algorithm itself will be optimised.

By building different data-bases, several source-to-source code transformation tools will be created. These can be seen as different tools, depending on the emphasis given to the different functions performed, and the data-base used.

This task is thus a supporting task for tasks 1.3 and 3.3. Specific source to source program transformation tools will be created by writing suitable data-bases of program transformations. There will also be feedback from tasks 1.3 and 3.3 into this task, indicating how the system needs to be extended to perform the more complicated program transformations required therein. The basic system, already developed, will enable the simultaneous start of all these tasks.

One of the strong points of this system will be its expandability: the user will be able to easily create his own data-bases or to add or replace rules in existing data-bases.

Task 3.3: HPF source-to-source optimiser

Optimization of high-level programs is usually done in one of two ways. The compiler may create optimized object-code directly, by analysing the program and internally applying program transformations that optimize the code.

The alternative is to use a source-to-source optimizer. In this case the program, written in a high-level language, is read and analyzed. The optimizer then finds program transformations that optimize the code and outputs the resulting program in the same high-level language.

In this task we will develop such a source-to-source optimizer for the HPF programming language. The system will use the generalized source-to-source transformation system developed in task 3.2, together with a data-base of optimizing source-to-source transformations.

The purpose of this task is to identify source-to-source optimizations that optimize HPF code, and to create a basic source to-source optimizer implementing them. The main outputs will thus be a source optimizer for HPF, and feedback to task 3.2 in the form of a list of requirements concerning the kind of pattern-matching capabilities and program transformations needed by such an optimizer.

We will investigate all program transformations that can be applied to HPF programs—not only those specific to the extensions of HPF over Fortran 90.

We will also investigate complicated optimizations that cannot usually be found by an automatic tool, but which are often found by users. A common example is the replacement of some typical small program kernels by optimized (probably very different) equivalent program kernels. The characteristics of the generalized source-to-source transformation system developed in task 3.2 allows one to implement these kind of transformations easily and incrementally, and so makes them practical.

The work will consist of two parts:

- To investigate source-to-source transformations that can optimize HPF code.
- To implement those transformations in the data-base, and to test and benchmark them.

The output source-to-source optimizer *could be used in any HPF programming environment.*

Strong points of such a system are that it could be easily expanded as new optimizing transformations are found, and that the user will be able to build his own data-base of optimizing transformations.

Task 3.4: HPF intermediate code optimiser

Several of the HPF compilers currently being implemented work by translating the HPF source into Fortran 90 plus calls to a standard message-passing library. PVM and MPI are those most commonly proposed.

The code produced by these compilers is unlikely to be optimal. That applies especially to initial implementations, but is likely to remain the case. Even for sequential languages, sophisticated users can write assembler code that is more efficient than that produced by the compiler.

This opens the door to a new method of optimizing HPF programs: modification of the intermediate code produced by the translator.

In this task, this scheme will be implemented within the HPF programming environment currently being developed by BES. This programming environment consists of a translator from HPF to Fortran 90 plus PVM or MPI, a source-level debugger and a profiler.

The following work will be performed:

1. The existing source-level debugger and the profiler will be extended and integrated to produce a new tool. This tool will allow the user to examine the original HPF program along with the code produced by the translator, and the profiling information produced by the profiling system.

2. The translator will be modified to allow the user to replace or modify the code that it generates.

The tool will, for example, allow the user to step with the debugger through the program. In addition to seeing the original source code of the program being executed, the user will also be able to see the translated code with message-passing calls, what communications are taking place, and the cost of them. To optimize the program the user might then:

1. Modify the original code so as to remove some of the communications. (A problem encountered by users of Connection Machine Fortran, for example, is that seemingly innocent statements have a lot of communication associated with them.)

2. Provide alternative code to that created by the translator. (The translator will be modified to allow the user to do this). This is similar to the technique of inserting assembler code in programs written in a high-level language. This technique should generally be used with care as it impairs portability, but it is sometimes very useful, as it allows dramatic optimizations of some program kernels.

This work will be done in a way that makes it as portable as possible to other HPF compilers that are implemented as translators to Fortran 90 with message-passing. The tool will also be integrated with the common tool interface developed in tasks 1.5 and 1.6.

The HPF programming environment used as a baseline for this task is currently being implemented in another (Eureka) project. It will not be completed until the second half of 1994, so this task will therefore start in the second year of this project.

7.4 Workpackage 4: Validation and evaluation

While tool and library developers will obviously perform their own validation and evaluation as an integral part of the development, it is useful to have a task to evaluate the tools independently. This role will be undertaken by Telmat, who are 'independent' in the sense of not being involved themselves in tool development in the project, but who have a strong interest in using the the tools both as software support for their hardware products and for 'in house' porting of customers' applications. They are also well suited to this role because of their wide experience and constant tracking of commercially available programming environments and support tools, and their knowledge of users' demands and expectations.

The evaluation will be performed using a variety of source codes, including:

- real application codes, including large geoscience applications (e.g. seismic modelling),
- 'Genesis' benchmarks,
- 'EuroBen' benchmarks,

on variety of hardware platforms, namely:

- CS-2,
- 88000 based workstation cluster,
- Future TELMAT MPP machines.

The workpackage comprises two tasks.

Task 4.1: Ports to HPF

The first tools developed in the project will not be available for evaluation until after the first year of the project. However, this time will be usefully occupied by (i) testing the ideas for code restructuring and 'improvement' upon which the tools are based, by hand transforming codes and measuring the effects on performance, and (ii) evaluating existing and forthcoming HPF technology and support tools for comparison with the planned project developments. This will provide valuable feedback to the tool development. The task will use currently available tools (e.g. Foresys, Forest and Partita from Connexité) and 'HPF' compiler/ translator technology (e.g. XHPF77 & HPF77) together with others that may become available.

The output of this task will be a deliverable report on:

- the performance, using current HPF compilers / translators, of codes 'naively' ported to HPF;
- the performance gains achieved by various code modifications;
- guidelines for an efficient HPF programming style.

This will provide feedback to the tool development.

Task 4.2: Evaluation of tools

In this task the project tools will be evaluated as they become available, and the comparative evaluation of HPF support software from other sources will also be continued.

Activities will include:

- Using the tools within Telmat's activities in other European projects, notably to port industrial application codes to MPP systems, where the tools will provide MPP programming support.
- Promoting good HPF programming styles through training using the tools, e.g. via the COMETT ACT project.
- Releasing early versions of selected project software (under non-disclosure agreements) to selected customers for independent feedback.

Telmat have a strong interest in exploiting the tools for use on the hardware platforms they market.

7.5 Workpackage 5: Project management

The project management details are described in the next section.

7.6 Risks

Part of the work on the HPF programming environment that will be an input to task 3.4 ('HPF intermediate code optimiser') depends on funding from a proposed Eureka project, whose funding status will be resolved on June 93. In the (unlikely) event that the Eureka funding fails the work for task 3.4 may have to be delayed or cancelled. Should this happen, it should not have serious repercussions for the rest of the project, as this task does not produce outputs to any other task; it only has inputs from them.

8 Project Management

The University of Southampton will be the Coordinating Contractor for the project, and will manage the project via its Parallel Applications Centre (PAC).

The PAC have considerable expertise in managing major industrial staff and defense-related R & D projects. The PAC is currently responsible for managing collaborative HPC applications projects with numerous companies, including BP International, British Gas, Cambridge Control, Ordnance Survey, Rover and Scott Wilson Kirkpatrick. Since this project brings together several commercial organisations, PAC's neutrality is an advantage in the role of project manager. It also has considerable expertise and background in the field of parallel high performance computing.

We will appoint a Project Manager to be responsible for management and coordination of the project. He will be supported by an administrative assistant.

The responsibilities of the Project Manager and administrator are:

1. to coordinate all activities within the project, and to ensure that any deviations are resolved (involving the project's Management Board as necessary);
2. to liaise between the project and the Commission on all matters concerning the project;
3. to organise regular technical and administrative meetings, presentations, exhibitions etc. as appropriate;
4. to coordinate the preparation and distribution of all reports;
5. to maintain accurate records of cost and effort reports;
6. to act as project representative for liaison with other projects.

The project's Management Board will consist of one person appointed by each of the partners, under the chairmanship of the Project Manager. This appointee will have authority to make both technical and administrative decisions on behalf of the respective partner. A deputy of similar responsibility may replace an appointee unable to attend a meeting, in order to ensure all contractors can be represented. Each appointee will also represent the interests of the Associated partners and sub-contractors who are directly or indirectly connected with that partner. A partner's representative may be replaced at any time by written notice to all other partners. A partner's representative on the Management Board will hold one vote when decisions are made.

Meetings of the Management Board will be held either immediately before or immediately after other project meetings, and will be called at the request of the Project Manager or any of the partner's representatives. There will be at least seven days notice of such meetings, with an agenda to be circulated by the project administrative assistant. Meetings can be called at shorter notice if the Project Manager deems the need to be extremely urgent.

The Management Board shall be responsible for:

1. Decisions on all project management issues. A simple majority decision will carry a motion. A tied vote will result in a motion not being carried.
2. Amendments to the workplan as defined in the Technical Annex, and resolution of disputes between partners that do not affect their legal rights and obligations. A simple majority decision will carry a motion. A tied vote will result in a motion not being carried.

3. The terms of the CEC contract, the costs or time schedules described in that contract, the termination date of the contract and the procedure for publications and press releases. Decisions on all of these matters must be decided by unanimous vote of all partners.

The main responsibility of the Management Board is to monitor the progress of each task with respect to the overall workplan to ensure timely achievement of deliverables and to identify all problems at an early stage.

Each workpackage has a lead contractor, who is responsible for coordinating the collaboration within the workpackage.

The quality control will be ensured by the Management Board. In particular all deliverables within the project must be agreed by the Management Board before presented to the CEC.

9 Workpackages

9.1 Workpackage 1: HPF program development tools

Leader:	Connexité
Other partners:	BES SA, INRIA, University of Southampton
Duration:	3 years
Manpower:	
Univ. Southampton	3 MY
task 1.1: A tool for improving 'dusty deck' Fortran codes: 1 MY $(1+0+0)$	
task 1.2: Data mapping 'assistant': 2 MY $(0+1+1)$	
BES SA	1.5 MY
task 1.3: Intrinsics and library call generator: 1.5 MY $(.5+.5+.5)$	
Connexité	4.5 MY
task 1.4: Adapting Forsys tools for HPF: 1.5 MY $(1+.5+0)$	
task 1.5: Building modern interfaces for tools: 1.5 MY $(0+.5+1)$	
task 1.6: Inter-tool protocol: 1.5 MY $(0+1+.5)$	
INRIA	3 MY
task 1.5: Building modern interfaces for tools: 1.5 MY $(0+.5+1)$	
task 1.6: Inter-tool protocol: 1.5 MY $(1+.5+0)$	
Total	12 MY
Other major resources required:	None

9.2 Workpackage 2: HPF Visualisation and Graphics

Leader:	Siemens
Other partners:	IBM Bergen, University of Southampton, TUM
Duration:	3 years
Manpower:	
TUM	3 MY
task 2.1: Trace and cross-reference data: 3 MY $(1+1+1)$	
Univ. Southampton	3 MY
task 2.2: Array-level visualisation tool: 3 MY $(1+1+1)$	
Siemens	9 MY
task 2.3: Message-passing level visualisation tool: 9 MY $(3+3+3)$	
IBM Bergen	6 MY
task 2.4: HPF Graphics Library: 6 MY $(2+2+2)$	
Total	21 MY
Other major resources required:	None

9.3 Workpackage 3: HPF optimisation

Leader:	BES SA
Other partners:	University of Southampton
Duration:	3 years
Manpower:	
Univ. Southampton	3 MY
task 3.1: HPF translation schemes: 3 MY $(1+1+1)$	
BES SA	6.5 MY
task 3.2: Source-to-source transformation system: 3 MY $(1+1+1)$	
task 3.3: Source-to-source optimiser: 1.5 MY $(0.5+0.5+0.5)$	
task 3.4: HPF intermediate code optimiser: 2 MY $(0+1+1)$	
Total	9.5 MY
Other major resources required:	None

9.4 Workpackage 4: Validation and evaluation

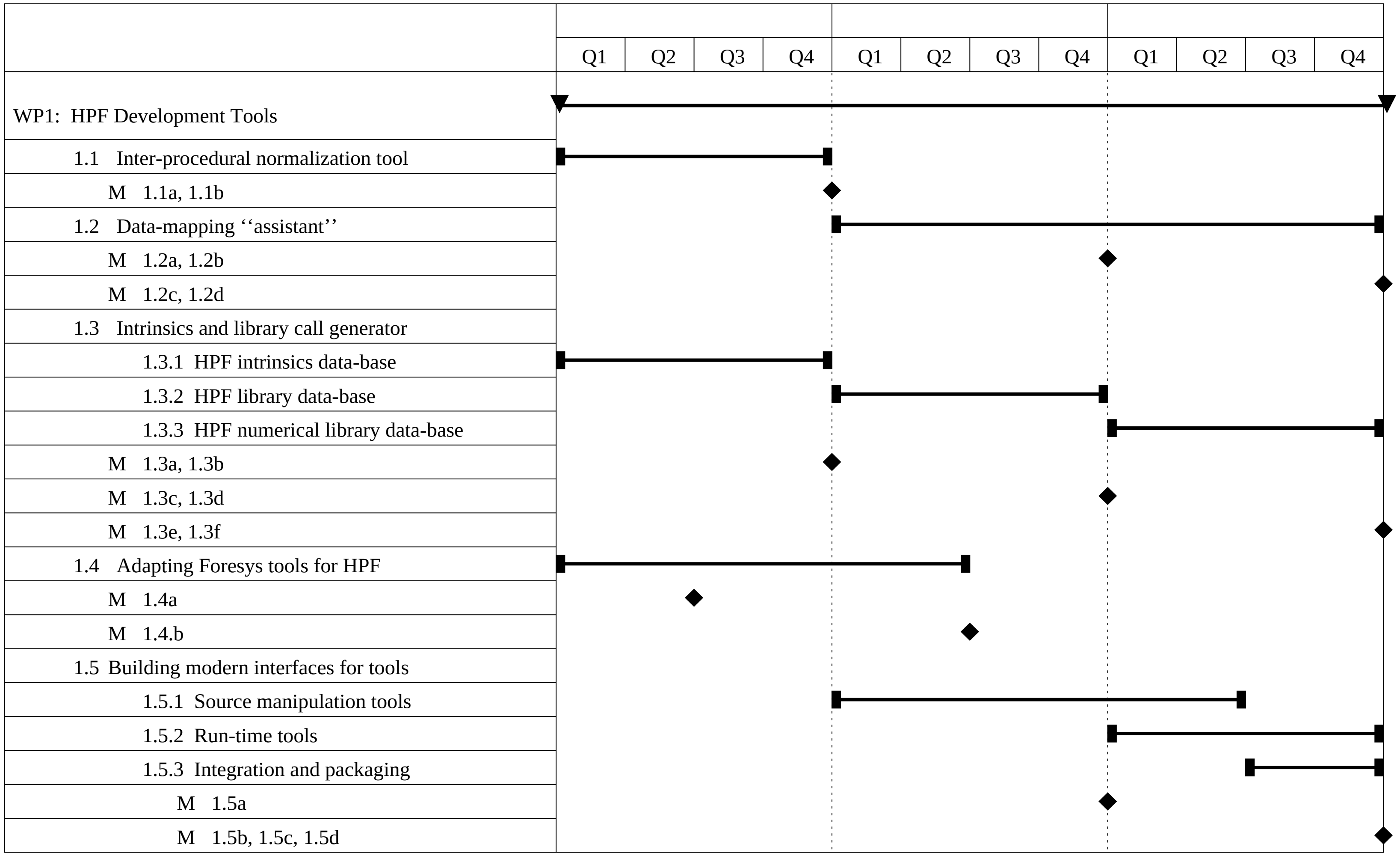
Leader:	Telmat Informatique
Other partners:	-
Duration:	3 years
Manpower:	
Telmat	3 MY
task 4.1: Evaluation of project tools: 3 MY $(1+1+1)$	
Total	3 MY
Other major resources required:	None

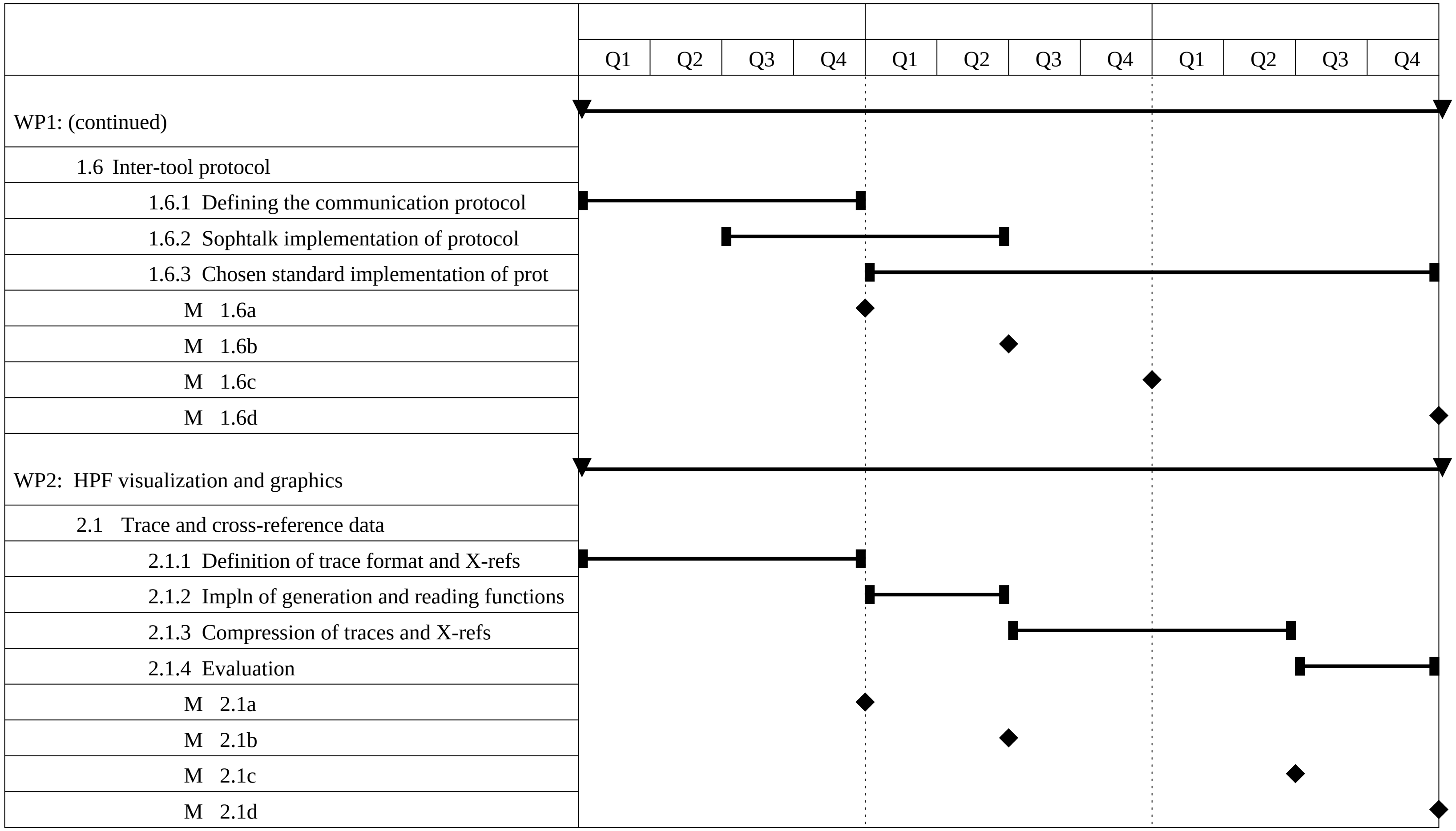
9.5 Workpackage 5: Project Management

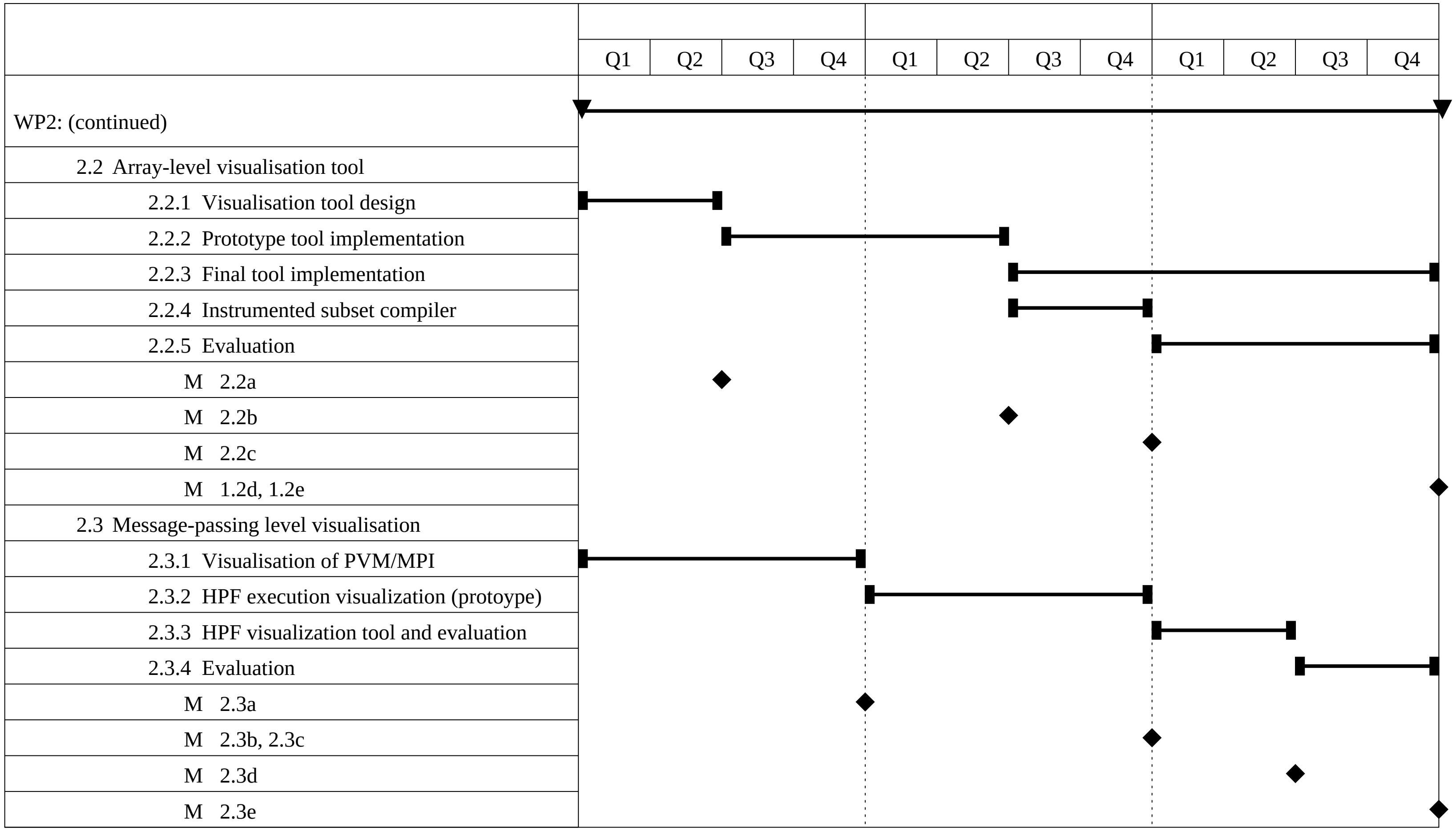
Leader:	University of Southampton
Other partners:	-
Duration:	3 years
Manpower:	
Univ. Southampton	3 MY
task 5.1: Project Management: 3 MY $(1+1+1)$	
Total	3 MY
Other major resources required:	None

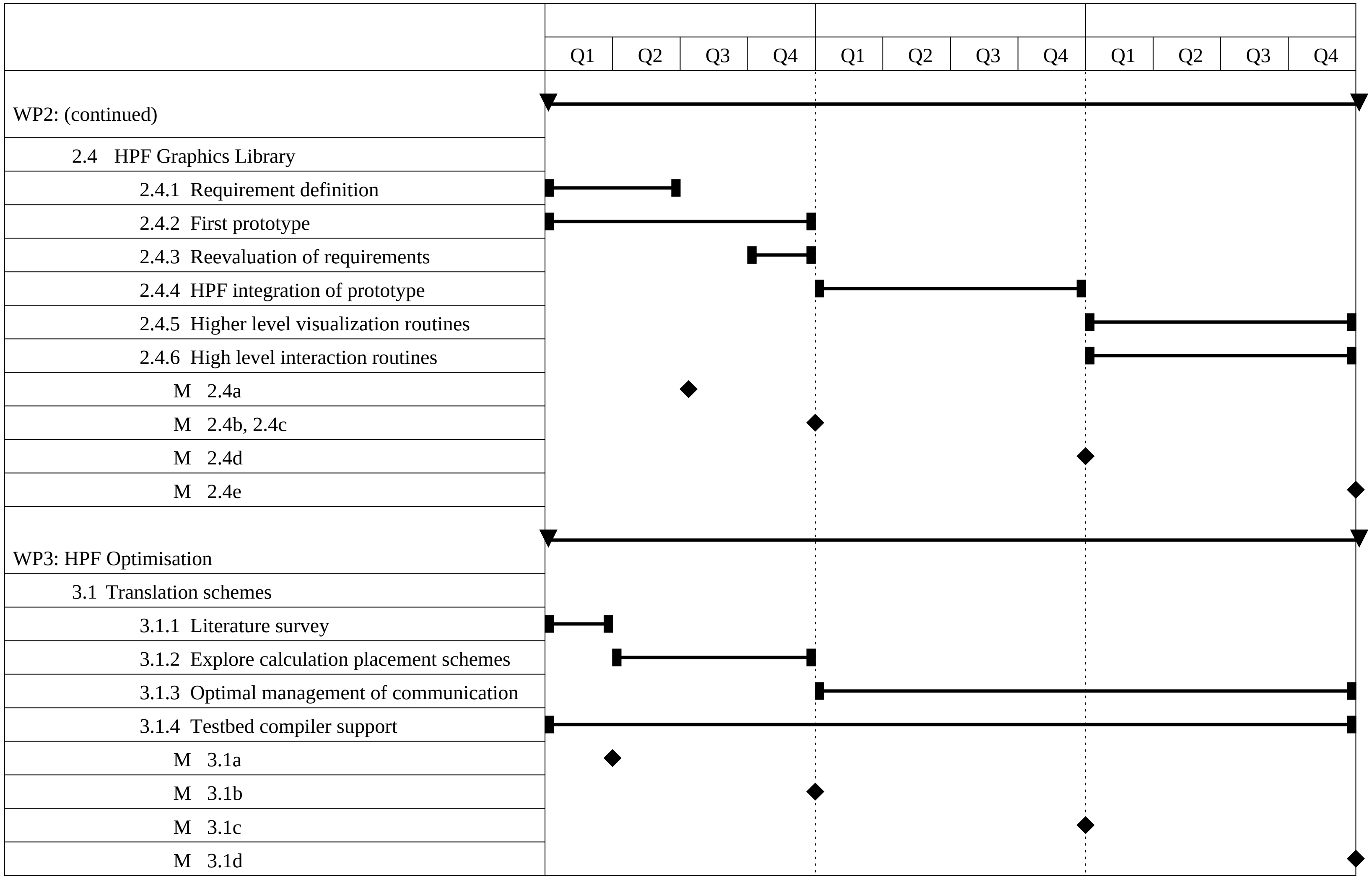
10 Workpackage interdependencies and milestones

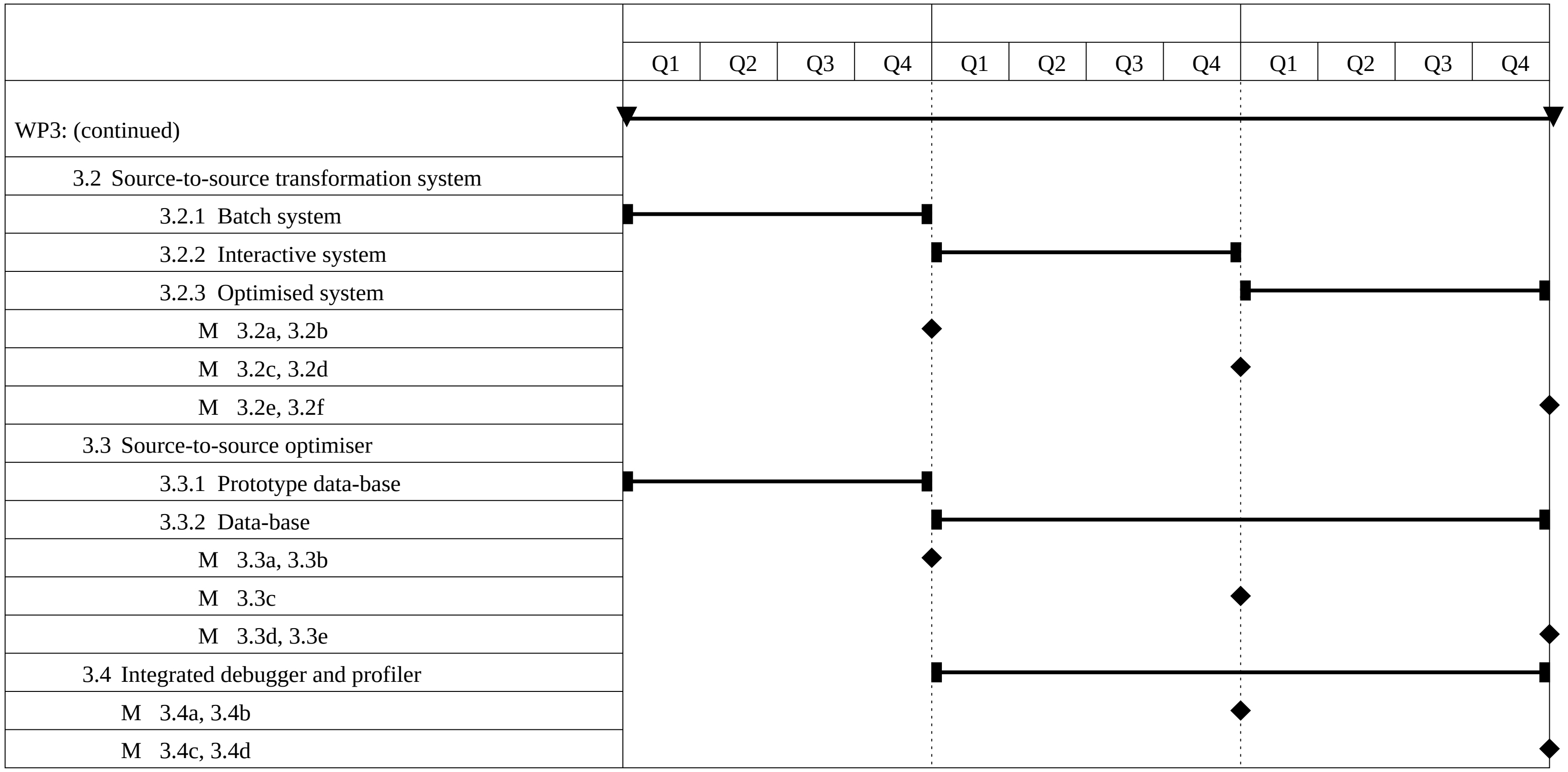
See attached Pert and Gantt charts.

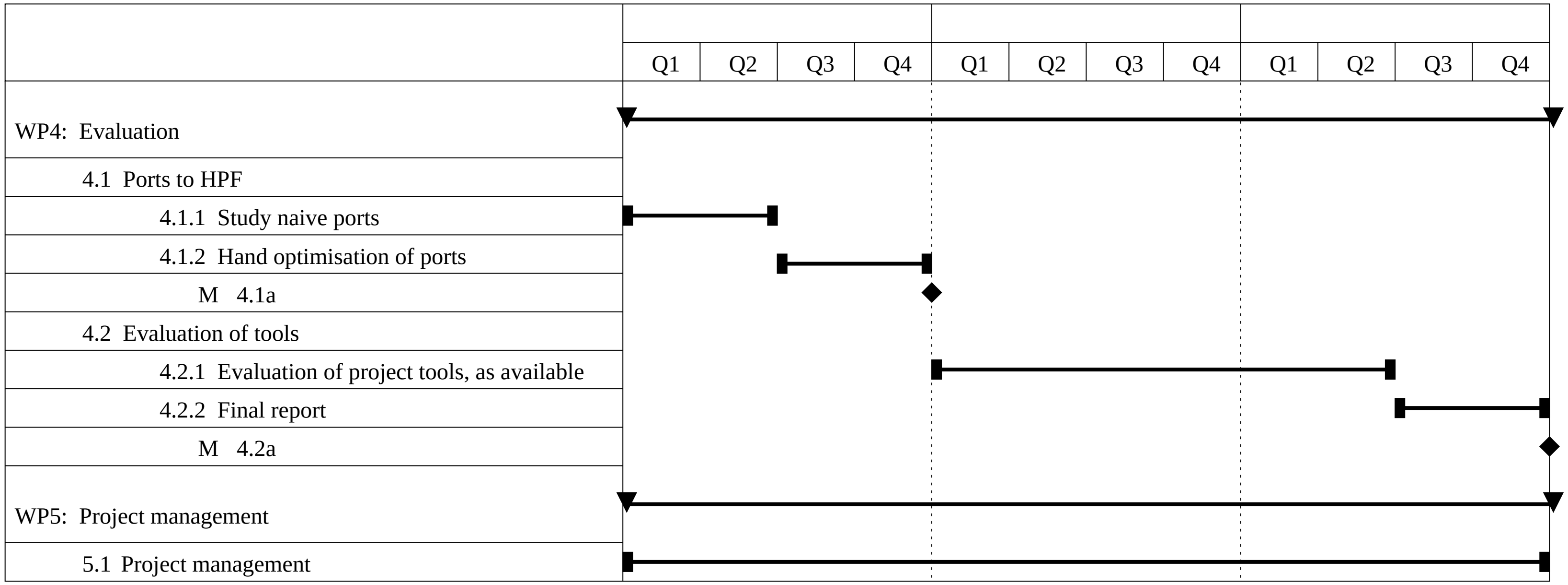

Q1 Q2 Q3 Q4 Q1 Q2 Q3 Q4 Q1 Q2 Q3 Q4
WP1: HPF Development Tools
1.1 Inter-procedural normalization tool
M 1.1a, 1.1b
1.2 Data-mapping ‘‘assistant’’
M 1.2a, 1.2b
M 1.2c, 1.2d
1.3 Intrinsics and library call generator
1.3.1 HPF intrinsics data-base
1.3.2 HPF library data-base
1.3.3 HPF numerical library data-base
M 1.3a, 1.3b
M 1.3c, 1.3d
M 1.3e, 1.3f
1.4 Adapting Foresys tools for HPF
M 1.4a
M 1.4.b
1.5 Building modern interfaces for tools
1.5.1 Source manipulation tools
1.5.2 Run-time tools
1.5.3 Integration and packaging
M 1.5a
M 1.5b, 1.5c, 1.5d


Q1
Q2
Q3
Q4
Q1
Q2
Q3
Q4
Q1
Q2
Q3
Q4
WP1: (continued)
1.6 Inter-tool protocol
1.6.1 Defining the communication protocol
1.6.2 Sophtalk implementation of protocol
1.6.3 Chosen standard implementation of prot
M 1.6a
M 1.6b
M 1.6c
M 1.6d
WP2: HPF visualization and graphics
2.1 Trace and cross-reference data
2.1.1 Definition of trace format and X-refs
2.1.2 Impln of generation and reading functions
2.1.3 Compression of traces and X-refs
2.1.4 Evaluation
M 2.1a
M 2.1b
M 2.1c
M 2.1d

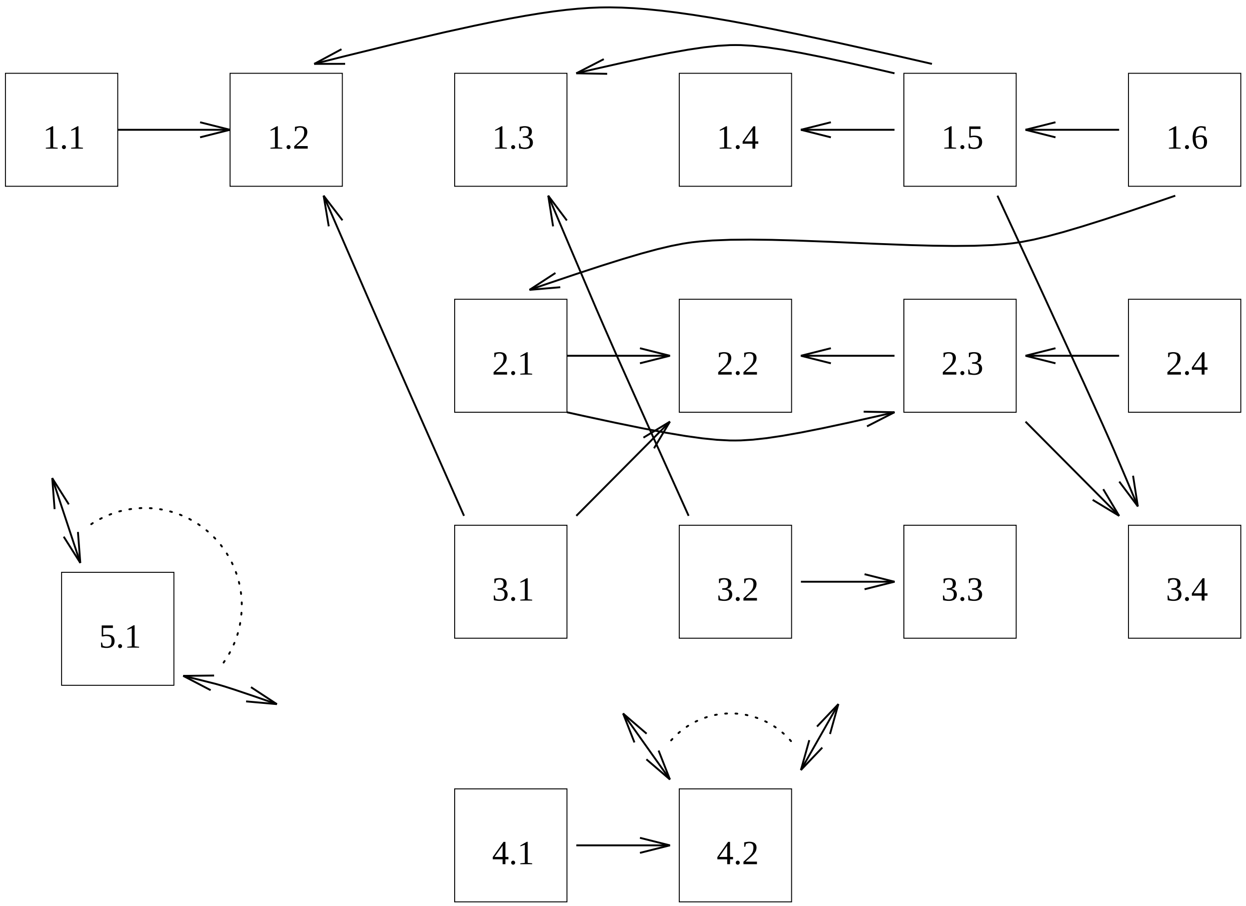

Q1
Q2
Q3
Q4
Q1
Q2
Q3
Q4
Q1
Q2
Q3
Q4
WP2: (continued)
2.2 Array-level visualisation tool
2.2.1 Visualisation tool design
2.2.2 Prototype tool implementation
2.2.3 Final tool implementation
2.2.4 Instrumented subset compiler
2.2.5 Evaluation
M 2.2a
M 2.2b
M 2.2c
M 1.2d, 1.2e
2.3 Message-passing level visualisation
2.3.1 Visualisation of PVM/MPI
2.3.2 HPF execution visualization (protoype)
2.3.3 HPF visualization tool and evaluation
2.3.4 Evaluation
M 2.3a
M 2.3b, 2.3c
M 2.3d
M 2.3e


Q1
Q2
Q3
Q4
Q1
Q2
Q3
Q4
Q1
Q2
Q3
Q4
WP2: (continued)
2.4 HPF Graphics Library
2.4.1 Requirement definition
2.4.2 First prototype
2.4.3 Reevaluation of requirements
2.4.4 HPF integration of prototype
2.4.5 Higher level visualization routines
2.4.6 High level interaction routines
M 2.4a
M 2.4b, 2.4c
M 2.4d
M 2.4e
WP3: HPF Optimisation
3.1 Translation schemes
3.1.1 Literature survey
3.1.2 Explore calculation placement schemes
3.1.3 Optimal management of communication
3.1.4 Testbed compiler support
M 3.1a
M 3.1b
M 3.1c
M 3.1d


Q1
Q2
Q3
Q4
Q1
Q2
Q3
Q4
Q1
Q2
Q3
Q4
WP3: (continued)
3.2 Source-to-source transformation system
3.2.1 Batch system
3.2.2 Interactive system
3.2.3 Optimised system
M 3.2a, 3.2b
M 3.2c, 3.2d
M 3.2e, 3.2f
3.3 Source-to-source optimiser
3.3.1 Prototype data-base
3.3.2 Data-base
M 3.3a, 3.3b
M 3.3c
M 3.3d, 3.3e
3.4 Integrated debugger and profiler
M 3.4a, 3.4b
M 3.4c, 3.4d


Q1
Q2
Q3
Q4
Q1
Q2
Q3
Q4
Q1
Q2
Q3
Q4
WP4: Evaluation
4.1 Ports to HPF
4.1.1 Study naive ports
4.1.2 Hand optimisation of ports
M 4.1a
4.2 Evaluation of tools
4.2.1 Evaluation of project tools, as available
4.2.2 Final report
M 4.2a
WP5: Project management
5.1 Project management

HPF-Tools: Major inter-task dependencies



11 Major Deliverables and Review Points

11.1 Workpackage 1

Task	Deliverable	Id	type	avail-ability	Months from project start
1.1: Interprocedural normalisation					
	Interprocedural normalisation tool	1.1a	report	P	12
		1.1b	tool	R	12
1.2: Data mapping 'assistant'					
	Alignment generator	1.2a	report	P	24
		1.2b	tool	R	24
	Data mapping 'assistant'	1.2c	report	P	36
		1.2d	tool	R	36
1.3: Intrinsics / library call generator (ILG)					
	ILG progress report	1.3a	report	P	12
	HPF intrinsics data-base	1.3b	tool	C	12
	ILG progress report	1.3c	report	P	24
	HPF library data-base	1.3d	tool	C	24
	ILG final report	1.3e	report	P	36
	HPF num-library data-base	1.3f	tool	C	36
1.4: Adapting Foresys Tools for HPF					
	adapting parser, visualizer, Forest	1.4a	code	C	6
	product Foresys/HPF	1.4b	code	C	18
1.5 Building Modern Interfaces for Tools					
1.5.1 Source Manipulation Tools					
	prototype environment w/src tools	1.5a	code	R	24
	documentation source tools interface	1.5b	report	P	36
1.5.2 Run Time Tools					
1.5.3 Integration and Packaging					
	doc final HPF environment	1.5c	report	P	36
	product final HPF environment	1.5d	code	C	36
1.6 Inter-tool Protocol					
1.6.1 Defining the communication protocol					
	protocol specification	1.6a	report	P	12
1.6.2 Sophtalk implementation of the protocol					
	product w/existing tools	1.6b	code	R	18
1.6.3 Chosen standard implementation of the protocol					
	doc standard version	1.6c	report	P	24
	product using standard	1.6d	code	C	36

11.2 Workpackage 2

Task	Deliverable	Id	type	availability	Months from project start
2.1: Trace and cross reference data					
	Defn: trace format, X-ref, interfaces	2.1a	report	P	12
	Impln: generation & reading funcs	2.1b	code	P	18
	Compression of traces and X-refs	2.1c	report	P	24
		2.1d	code	P	30
	Evaluation	2.1e	report	P	36
2.2: Array-level visualisation tool					
	Visualisation tool design	2.2a	report	P	6
	Prototype implementation	2.2b	code	R	18
	Instrumented subset compiler	2.2c	code	R	24
	Final implementation	2.2d	code	R	36
	Evaluation	2.2e	report	R	36
2.3: Message-passing level visualisation tool					
	Visualization of PVM/MPI level	2.3a	tool	R	12
	HPF execution visualization	2.3b	report	P	24
	Prototype	2.3c	tool	R	24
	HPF execution visualization tool	2.3d	tool	R	30
	Evaluation	2.3e	report	P	36
2.4: HPF Graphics Library					
	Requirements definition document	2.4a	report	P	6
	First prototype	2.4b	code	P	12
	User documentation	2.4c	report	P	12
	Progress report	2.4d	report	P	24
	Final report	2.4e	report	P	36

11.3 Workpackage 3

Task	Deliverable	Id	type	availability	Months from project start
3.1: Translation Schemes					
	Literature survey	3.1a	report	P	3
	Calculation placement strategies	3.1b	report	P	12
	Progress report	3.1c	report	P	24
	Testbed implementation and final report	3.1d	tool	R	36
3.2: Source-to-source transformation system					
	progress report	3.2a	report	P	12
	batch system	3.2b	tool	C	12
	progress report	3.2c	report	P	24
	interactive	3.2d	tool	C	24
	final report	3.2e	report	P	36
	optimised	3.2f	tool	C	36
3.3: Source-to-source optimiser					
	progress report	3.3a	report	P	12
	prototype data-base	3.3b	tool	C	12
	progress report	3.3c	report	P	24
	final report	3.3d	report	P	36
	data-base	3.3e	tool	C	36
3.4: HPF intermediate code optimiser					
	progress report	3.4a	report	P	24
	prototype	3.4b	tool	C	24
	final report	3.4c	report	P	36
	tool	3.4d	tool	C	36

11.4 Workpackage 4

Task	Deliverable	Id	type	availability	Months from project start
4.1: Ports to HPF					
	Progress report	4.1a	report	P	12
4.2: Evaluation of project tools					
	Final report	4.2a	report	P	36

11.5 Workpackage 5

Task	Deliverable	Id	type	availability	Months from project start
5.1: Project Management					

12 Other dependencies

Part of the work by BES on the HPF programming environment that will be an input to task 3.4 ('HPF intermediate code optimiser') depends on funding from a proposed Eureka project, whose funding status will be resolved on June 93. In the (unlikely) event that the Eureka funding fails the work for task 3.4 may have to be delayed or cancelled. Should this happen, it should not have serious repercussions for the rest of the project, as this task does not produce outputs to any other task; it only has inputs from them.

References

[EA88] S. Eichholz and F. Abstreiter. Runtime observation of parallel programs. In *Proceedings of the Conference of Parallel Processing (CONPAR'88)*, Manchester (UK), September 1988. Cambridge University Press.

[Eic87] S. Eichholz. Parallel programming with ParMod. In S. K. Sahni, editor, *Proceedings of the 1987 International Conference on Parallel Processing*, pages 377–380, University Park, August 1987. The Pennsylvania State University Press.

[Fri92] M. Friedrich. *Sprachmittel und Werkzeuge zur Unterstützung paralleler und verteilter Programme.* Dissertation, SFB–Bericht, Institut für Informatik, TU München, March 1992.

[IW93] H. Ilmberger, C.P. Wiedemann. A Visualization and Control Environment for Parallel Program Debugging. HICSS-26 (Hawaii Int. Conf. on System Sciences, Jan. 93, Hawaii), Vol. II, 387-396.

[WSF91] A. Weininger, T. Schnekenburger, and M. Friedrich. Parallel and distributed programming with ParMod-C. In *First International Conference of the Austrian Center for Parallel Computation (ACPC'91)*, Salzburg, Austria, 1991.

[Mer91] J.H. Merlin. ADAPTing Fortran 90 array programs for distributed memory architectures. *Proc. 1st Int. Conf. of the ACPC*, Salzburg, Sept. 1991.

[Mer92] J.H. Merlin. Techniques for the Automatic Parallelisation of 'Distributed Fortran 90'. Southampton University technical report, SNARC 92-02, 1992.