FITS—A Light-Weight Integrated Programming Environment*

B. Chapman[1], F. Bodin[2], L. Hill[3], J. Merlin[4], G. Viland[3], and F. Wollenweber[5]

[1] Department of Electronics and Computer Science, University of Southampton, Highfield, Southampton SO17 1BJ, U.K.
bmc@ecs.soton.ac.uk
[2] IRISA, Campus Universitaire de Beaulieu, 35042 Rennes, France
[3] Simulog Sophia Antipolis, Les Taissounieres HB2, Route des Dolines, 06560 Valbonne, France
[4] VCPC, University of Vienna, Liechtensteinstr. 22, 1090 Vienna, Austria
[5] Eumetsat, Am Kavalleriesand 31, 64205 Darmstadt, Germany

Abstract. Few portable programming environments exist to support the labour-intensive process of application development for parallel systems. Popular stand-alone tools for analysis, restructuring, debugging and performance optimisation have not been successfully combined to create integrated development environments.
In the ESPRIT project FITS we have created such a toolset, based upon commercial and research tools, for parallel application development. Component tools are loosely coupled; with little modification, they may invoke functions from other components. Thus integration comes at minimal cost to the vendor, who retains vital product independence. The FITS interface is publically available and the toolset is easily extensible.

1 Introduction

Integrated software development environments are popular. It is easier to use a suite of functions which have been crafted to complement each other, than to learn how to apply a set of individual tools, possibly with different conventions and terminology and overlapping functionality; moreover, the developer may then access different kinds of information on a program concurrently.

There have been a number of efforts to build programming environments for the creation and maintenance of parallel programs. However, the only toolsets that have been truly successful are those which have been constructed by a single vendor; these tend to run on a restricted range of platforms or are limited in their scope. Whilst some individual development tools (e.g. TotalView) have been remarkably successful, efforts to provide environments based upon a collection of such tools have not been so. Indeed, the only levels of "integration" that have been adopted in practice are interaction at the operating system level, e.g. via

* The work described in this paper was supported by the European Union ESPRIT project 23502, 'FITS'.

P. Amestoy et al. (Eds.): Euro-Par'99, LNCS 1685, pp. 125–134, 1999.

Unix files, and incorporation within a simple graphical user interface. Thus a tool may benefit from the output of another tool, but no further interaction is implied.

There are sound reasons for this. The complexity of each part of the parallelisation process has led to the specialisation of tools (and their suppliers), which usually handle just one aspect of the parallelisation problem. Thus each potential component of a toolset is developed in isolation from other tools. Each tool has its own internal data structures and programming conventions. The vendors are unlikely to share details of these, let alone adapt existing structures and productisation plans to fit in with those of another product.

In the *Fortran Integrated ToolSet* ('*FITS*') project we have tried to learn from the lessons of the past regarding tool integration. This ESPRIT-funded project has created a parallel programming environment incorporating two existing, separately developed and marketed products, and includes functionality from two research prototypes. In order to permit interoperability, we have sought a light-weight solution which avoids any major re-engineering of the component tools and does not require complex interfaces. As a consequence, continuing product independence is also assured. The result is the FITS toolset, an extensible, portable programming environment for the development and maintenance of parallel applications written in Fortran.

This paper is organised as follows. First we describe the tasks involved in a typical parallelisation project. Then we introduce the components of the FITS toolset and show how they support this process. We outline the functionality of the integrated toolset and give a few examples of the benefits of the FITS tool interaction before describing the mechanisms employed to realise it. We conclude with a summary of related work and future plans.

2 The Manual Parallelisation Process

Coarse-grain parallelisation is a particularly demanding task in terms of the manual labour it involves. A suitable global parallelisation strategy must be devised and implemented, requiring a good understanding of the entire code. Tuning is an intricate process which must be performed separately for distinct architectures, implying that several different parallel program versions may be created and require maintenance. The main activities in the program parallelisation cycle are as follows:

1. **Analysis:** The code is first statically and dynamically analysed in depth in order to fully understand its data usage, to identify computationally intensive regions, and to select a parallelisation strategy.
2. **Restructuring:** The code is cleaned and restructured in preparation for parallelisation and to facilitate subsequent program maintenance. Non-standard constructs are removed to achieve a portable version.
3. **Debugging / verification:** It is debugged and its correctness is verified. The result is now suitable for use as a baseline version for parallelisation.

4. **Parallelisation:** Results of the program analysis and data locality information are used to parallelise the program, either explicitly under the message-passing paradigm or by inserting directives if a higher-level paradigm such as HPF or OpenMP is used.
5. **Debugging / verification:** The parallel program version is debugged and its correctness established.
6. **Tuning:** The performance optimisation cycle begins. The application is run several times on the target architecture. Results are visualised by a performance tool, communication hot spots identified and the corresponding source code re-examined. It may need modification to reduce or re-organise communication. Transformations are applied to optimise the single node performance, in particular to better utilise the memory hierarchy. This process terminates when results are satisfactory.
7. **Maintenance:** The verified parallel version becomes the main production version, and program maintenance becomes the dominant activity.

Many of the individual tasks in the migration process are conceptually simple, but daunting in view of the size of programs and the number of details to be considered. For example, coarse-grain parallelisation, like other large-scale program modifications, requires evaluation of the data and control flow throughout the entire program. Yet a program such as the global weather forecast model IFS [2], developed and used in production mode on a parallel system at the European Centre for Medium-Range Weather Forecasting (ECMWF), consists of over 300,000 lines of code spread throughout many files; there are some 1,400 subroutines. Instances of Fortran storage and sequence association must be identified and, if an impediment to parallelisation, removed. This involves comparing actual and formal arguments at call sites throughout the program, comparing the declarations of common blocks in the entire code, and searching for other forms of implicit and explicit equivalencing of data. But IFS has over 300 different common blocks, involving a large number of variables. Some subroutines have hundreds of arguments in their interface. Codes such as this have often been highly vectorised and may contain proprietary language extensions. Re-engineering of data structures and loops may be needed in order to achieve good performance on a parallel system.

An estimate of the effort required for each of the major steps in the parallelisation of IFS is in general agreement with an early user survey [11]. At ECMWF, about 25% of the effort went into code analysis, and another 25% into cleaning and restructuring the baseline code, to remove vendor-specific code features, reorganise control flow and data structures, and convert it to Fortran 90. A further 20% went into the initial parallelisation. Finally, debugging and performance tuning each took about 15% of the overall effort. Our estimates suggest that the fraction of this effort that is supported by a typical vendor-supplied toolset, likely to consist of a compiler, runtime system, debugger and some kind of performance tool (possibly no more than a profiler), is no more than 30%; in particular, it is unlikely to be of use in the significant initial program analysis and restructuring phases.

3 The FITS Toolset

The FITS project has created an extensible toolset facilitating the development of parallel applications. The tools it contains support almost all of the major activities in parallel application development, as outlined above.

We now describe the main components of the FITS toolset, and conclude this section with an overview of the mechanisms used to integrate them.

3.1 FITS Components

The following tools have been input to the development of the FITS environment:

FORESYS: a Fortran source code restructurer, providing source code display, checking, analysis and transformation, including conversion to Fortran 95.
TSF: a module for the user-driven application of source code transformations.
VAMPIR: a trace generator and performance analysis tool.
ANALYST: a research prototype interactive Fortran program analyser with a sophisticated graphical user interface.

In addition, ideas from another tool, **IDA**, were used to design GIM's displays.

FORESYS [13], from Simulog, is itself an integrated collection of tools to check, analyse and restructure Fortran programs. It accepts Fortran with many extensions, and can transform it into standard-conforming Fortran 77 or Fortran 95 and ensure that it meets quality assurance standards. It 'pretty prints' the source code, using colour and a variety of fonts to make the code structure clearer. It includes an editor, thus permitting code modifications from within the system. It performs a variety of interprocedural checks, and can also display data dependence graphs. This tool is thus particularly useful in steps *1*, *2*, *4* and *7* of the development process outlined in Section 2.

The TSF module [3] is an extension to FORESYS that provides a set of source code transformations for performance enhancement. Examples are loop transformations such as loop blocking, unrolling, interchanging, etc. These are applied under the user's control, usually in conjunction with steps *2* or *6* in the above description. TSF also provides a scripting mechanism that allows users to write their own transformations and to access program information stored in the FORESYS system. This allows the automation of many of the repetitive tasks that arise at various stages of code parallelisation.

VAMPIR [10], from Pallas, is a tool which supports the program optimisation step by visualising its execution behaviour. It does so by displaying information obtained by tracing a program run. It can gather and present a range of information, including the execution of message-passing constructs, collective communication and procedure calls, and it can present this as system snapshots or animated sequences. It has many measurement options and a range of graphical displays. The user may zoom in on arbitrarily small time intervals of the trace.

ANALYST [5, 6] is an interactive Fortran program analyser. It allows the user to browse through a Fortran code and request various kinds of information about

it. Information is presented in both graphical and text formats. The displays are themselves interactive: thus a user may click on an arc of a callgraph to obtain details of the subroutine invocations it represents, or may access the source text of a program unit by clicking on the corresponding node or its name in a list of subprograms. Filter options enable the extraction of data from large graphs. Displayed information is consistently related to the corresponding region of source code; program information is thus provided together with the relevant source text. ANALYST formed the basis for developing the FITS Graphical Interface Module, GIM.

IDA [8] is a command-line-driven tool for interactive program analysis. It is very easy to use and, where its functions are sufficient for the task at hand, is thus a particularly convenient tool. In particular, it can provide a quick textual display of a program's callgraph. It has inspired the provision of similar textual representations of some information in the FITS toolset.

Figure 1 shows how these components are linked together.

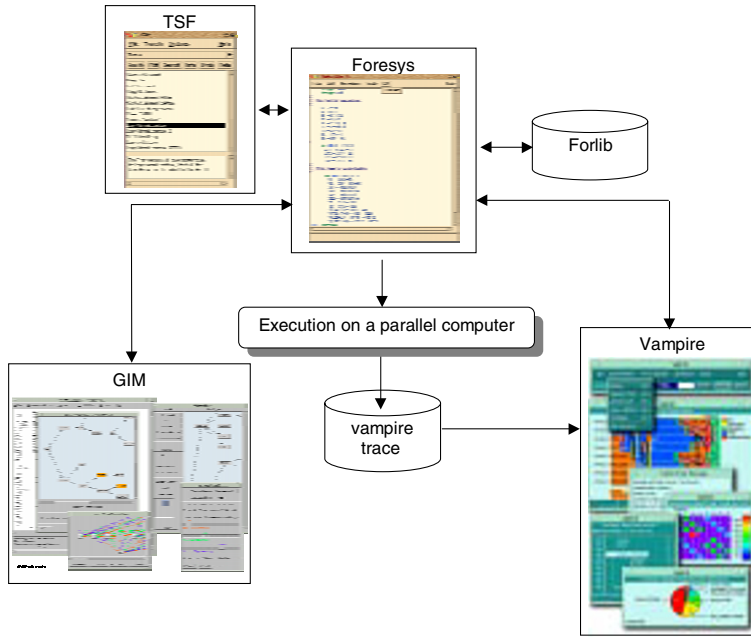

Fig. 1. Overview of the FITS toolset components.

3.2 FITS' Capablilities

As shown in Figure 2, the FITS toolset contains functions to support the majority of tasks in the parallelisation process. It is portable and extensible, permitting

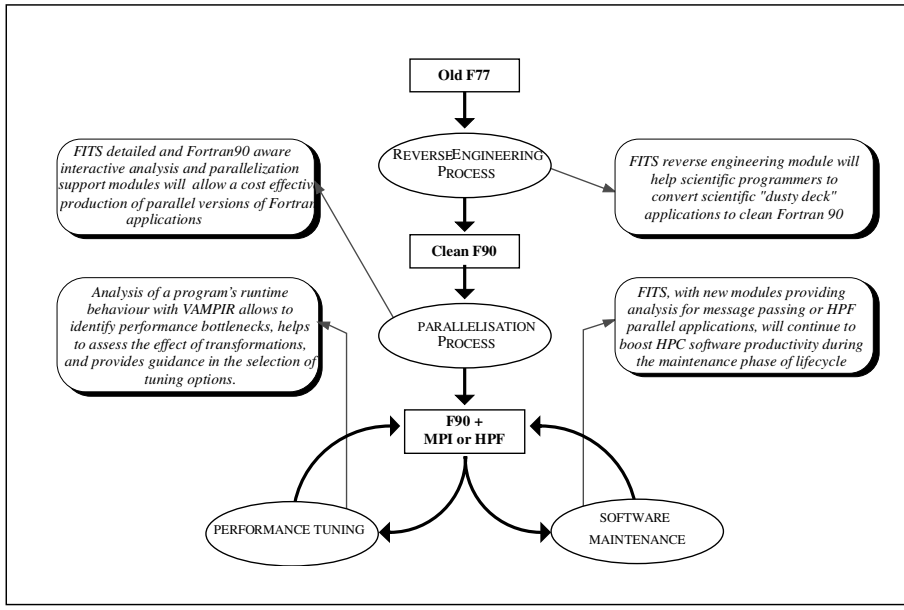

Fig. 2. Application life cycle.

the inclusion of additional tools. Although not exclusively developed for MPI [9], it provides specific functionality to support the development of MPI code.

It provides the following functions to support the parallelisation activities described in Section 2:

Static and dynamic program analysis (for steps *1* and *6*):
- Graphical displays of the source program structure, for example its call-graph, `USE`graph, and data and control flow. These displays may be manipulated to isolate features of interest, and a graphical element may be linked to a display of the corresponding source text.
- Display of the data dependence graph for a code region, and details of the dependence relationships it represents.
- 'Pretty printing' of the source program.
- Support for analysing a program's runtime behaviour, with the ability to view a source code statement or region together with a display of its execution behaviour.

Code transformations (for steps *2*, *4*, *6* and *7)*:
- Code cleaning (e.g. `GOTO` elimination, etc.) and translation to Fortran 95.
- Support for inserting MPI constructs into a code.
- Automatic insertion of instrumentation code to generate a VAMPIR tracefile.
- An *extensible* set of user-applied transformations to improve the performance of a parallel or sequential program, for example to tune it for the memory hierarchy.

Verification (for steps *3* and *5*):
- Checking the consistency of parameter passing and common block declarations.
- Static analysis of MPI message-passing in a parallel application.
- Fortran dialect checking.

The combined toolset extends the functionality of the individual tools in several ways. For example, FORESYS can automatically instrument a program so that it generates a tracefile for display by VAMPIR. VAMPIR can then not only visualise the execution behaviour, but it can also invoke FORESYS to display the corresponding location in the source text and/or GIM to display the relevant node in the callgraph, and vice versa. As another example, TSF uses FORESYS to implement program transformations, and these transformations may be invoked from a GIM display.

3.3 Tool Integration Mechanisms

At the heart of the toolset is a database created by FORESYS, called a *ForLib*. This contains detailed information about the source program, and it also keeps track of other files related to the program such as Makefiles and VAMPIR tracefiles.

Two types of integration are employed within FITS, depending on which tools are coupled.

The displays produced by GIM and the code transformations produced by TSF require a large amount of information to be exchanged with the *ForLib*. Therefore GIM and TSF are *tightly integrated* with FORESYS so that they have a fast and efficient connection to the *ForLib* database.

The coupling between FORESYS and VAMPIR is of a different nature. In this case the tools do not exchange information, but rather each tool can *remotely control* the other. For example, VAMPIR can request that FORESYS provides a source code display, and FORESYS can request that VAMPIR displays some execution events or statistics. Therefore these tools are coupled using a *light-weight integration* mechanism.

The latter mechanism has been designed for ease of use and extensibility. Although it does not provide the close interaction that is possible when a set of tools is designed and constructed by a single organisation, it does permit a variety of useful interactions between them and enhances the functionality of the individual components accordingly. This approach does not require any complex interfaces between the component systems and hence eliminates the need for close coordination of their development.

The general architecture of the FITS integration scheme is shown in Figure 3. Its major components are as follows:

The `fits_notify` utility: This utility program contains functions for remotely controlling tools in the FITS environment. Each participating tool can invoke `fits_notify`, e.g. by calling `system()`, to cause another tool to perform an

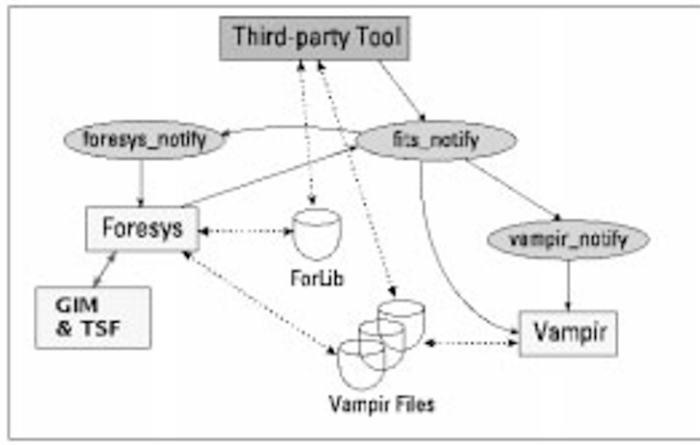

Fig. 3. FITS integration scheme.

action. The protocol is asynchronous; no reply is sent and the calling tool should not depend on the action being executed immediately. The syntax of the call permits the kind of remote action to be specified, thereby implicitly also selecting the target tool, and enables arguments to be given, including such things as a location within a program or trace file.

A set of command-line routines: These enable a tool to interact with the *ForLib*, the database containing source program information.

Interactive routines: These invoke FORESYS functions in order to obtain graphical displays of the source program.

Specified file formats: For example, a specification of the VAMPIR tracefile and configuration file formats.

The realisation depends strongly on the *ForLib* structure. Among the *ForLib*'s publically accessible attributes are: its *name*, which identifies a specific version of an application development; the list of *files* and *include files* comprising the program, with their directory paths; the list of Fortran *program units*; the Fortran *dialect* in which the program is written; a list of *program analysis options* to be applied; and the location of *files related to the program* such as Makefiles and VAMPIR tracefiles.

Operations may be run in batch mode and produce non-graphical output, mainly in text or HTML, or they may be interactive operations that provide graphical output via the displays of FORESYS and GIM. The batch commands enable a user or tool to obtain all public information on the contents of a *ForLib*, and permit its modification, e.g. by adding or removing Fortran files from it. Command-line routines are available to create a *ForLib* and to specify analysis and transformations to be applied to it. The interactive mode allows a third-party tool to request operations such as the display of a call graph, or the display

of the source code of a program unit, perhaps with highlighting of a particular location in it such as an executable statement.

Additional interaction possibilities are provided via TSF and via the VAMPIR configuration file. The latter enables a user or tool to specify options governing the creation and display of tracefiles. The VAMPIR tracefile format is also available as part of the FITS public interface, permitting a tool to read and process the trace information itself. In interactive mode, it is possible to load a tracefile and configuration file, to open certain VAMPIR displays, perhaps related to a particular module, and from there to also request source code views.

4 Related Work

In addition to vendor-supplied parallel program development toolsets, there is a range of product suites from individual vendors which combine several features of the above. These include the Kuck and Associate product line [7] for shared memory parallelisation. The FORGE parallelization tools [1] from Applied Parallel Research enable source code editing and offer control and data flow information, in combination with an HPF compiler. VAST [12] from Pacific Sierra Research helps a user migrate from Fortran 77 to parallel HPF code by automatic code cleanup and translation to Fortran 90 and HPF. It can be augmented by an HPF compiler and by a performance analysis tool, DEEP.

There have been attempts to integrate products from different vendors. The ESPRIT project PPPE (Portable Parallel Programming Environment) [4] enhanced the functionality of a range of programming tools. It began with the ambitious goal of integrating them within the PCTE framework. However, despite the undoubted power of PCTE and its benefits in terms of opportunities for closer tool interaction, the process of integration proved to be too complex and time-consuming for both users and vendors, and this approach was abandoned.

5 Conclusions and Future Work

In this paper we have described the FITS programming environment for the development of parallel Fortran applications. The environment is both portable and extensible. The method for tool interaction was chosen to satisfy the need of software vendors for independence, and to permit the inclusion of additional tools with relative ease, possibly to create customised environments at individual user sites. The realisation has required a central repository of information on a specific application development, the *ForLib*, and a set of functions which permit the asynchronous, remote invocation of functionality provided by component tools. The result is a working environment which will soon be available on the market.

During its development, the toolset was used by two end-user companies in the FITS project, Battelle and QSW, to parallelise two large industrial application codes, namely a CFD code and an electromagnetic scattering simulation. Their feedback provided an important input to the design and development of FITS.

The capabilities of the resulting toolset offer the opportunity to explore new approaches to some parallelisation problems, including introducing interactive guidance for the process of program transformation.

Acknowledgments

We thank our colleagues in the FITS project, especially Markus Egg (VCPC), Yann Mevel (IRISA), Hans-Christian Hoppe and Karl Solchenbach (Pallas), Wolfgang Nagel and Holger Brunst (TU Dresden), Aron Kneer (Battelle), and Ornella Fasolo, Agostino Longo and Carlo Nardone (QSW). We also thank Jerry Yan (NASA Ames) for his valuable comments and suggestions about FITS.

References

[1] Applied Parallel Research: *APR's FORGE 90 parallelization tools for High Performance Fortran*. APR, June 1993

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

[3] F. Bodin, Y. Mevel, and R. Quiniou, *A user level program transformation tool*, Proc. Int. Conf. on Supercomputing, 1998

[4] B. Chapman, T. Brandes, J. Cownie, M. Delves, A. Dunlop, H.-C. Hoppe and D. Pritchard. *The ESPRIT PPPE project: a final report.* Technical report VCPC 05-96, University of Vienna, 1996.
URL: `www.vcpc.univie.ac.at/activities/reports/doc-files/tr_05-96.ps`.

[5] B. Chapman, M. Egg and F. Wollenweber, *ANALYST Version 1.0 User's Guide*, VCPC, April 1996, rev. February 1997

[6] B. Chapman, M. Egg, *ANALYST: Tool Support for the Migration of Fortran Applications to Parallel Systems*, Proc. PDPTA' 97, Las Vegas, June 30–July 3, 1997

[7] Kuck and Associates. *KAP/Pro Toolset for OpenMP*. See `www.kai.com/kpts/`

[8] J.H. Merlin and J.S. Reeve. *IDA—an aid to the parallelisation of Fortran codes.* Technical report, Dept. of Electronics and Computer Science, University of Southampton, Sept. 1995. Software and documentation are available from `www.vcpc.univie.ac.at/information/software/ida/`.

[9] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard.* Int. Journal Supercomputing Applications, Vol 8 3/4, 1994.

[10] W. Nagel, A. Arnold, M. Weber, H.-C. Hoppe and K Solchenbach. *VAMPIR: Visualisation and analysis of MPI Resources.* Supercomputer 63, 12(1), 69-80, 1996

[11] C.M. Pancake and C. Cook, *What users need in parallel tool support: survey results and analysis*, Proc. Scalable High Performance Computing Conference, 1994

[12] C. Rodden and B. Brode. *VAST/Parallel: automatic parallelisation for SMP systems.* Pacific Sierra Research Corp., 1998.

[13] Simulog SA, *FORESYS, FORtran Engineering SYStem, Reference Manual Version 1.5*, Simulog, Guyancourt, France, 1996