A Survey of Available Tools for Developing Quality Software Using Fortran 95

D.J. Worth and C. Greenough
April 2005

Abstract

This report provides a survey of some of the software tools currently available to assist in the development of Fortran programs using the 1990 and 1995 standards. A few tools also address the development of High Performance Fortran (HPF). This report covers the whole scope of the software development life cycle with the exception of requirements gathering and design tools though the importance of these stages is acknowledged. Using the ‘waterfall’ or ‘cascade’ model of the software development process there are sections covering coding (including source code quality assurance), source code control, the build environment, compiling, debugging, testing and source code documentation.

Keywords: software quality, development tools, Fortran 90, Fortran 95, High Performance Fortran

Email: d.j.worth@rl.ac.uk or c.greenough@rl.ac.uk
Reports can be obtained from www.softeng.cse.clrc.ac.uk

Software Engineering Group
Computational Science & Engineering Department
Rutherford Appleton Laboratory
Chilton, Didcot
Oxfordshire OX11 0QX
6 Integration and System Testing

6.1 Validation ................................................................. 30
6.2 Debugging ................................................................. 30
6.2.1 The GNU Debugger (GDB) .......................................... 30
6.2.2 Data Display Debugger (DDD) ..................................... 31
6.2.3 Insight ................................................................. 31
6.2.4 KDbg ................................................................. 31
6.2.5 Absoft Fx2 Debugger ................................................ 32
6.2.6 Intel Debugger ........................................................ 32
6.2.7 Portland Linux Cluster Debugger (PGDBG) ..................... 33
6.2.8 Allinea Distributed Debugging Tool (DDT) ....................... 34
6.2.9 Etnus TotalView ..................................................... 34
6.3 Code Coverage Analysis ................................................ 35
6.3.1 Unix command t cov ................................................ 35
6.3.2 Polyhedron plusFort – CVRANAL .................................. 35
6.3.3 FACT ............................................................... 35
6.3.4 Cleanscape Grayboxx ............................................... 36
6.4 Performance Analysis .................................................. 36
6.4.1 Unix commands prof and gprof .................................... 36
6.4.2 Intel VTune ........................................................... 36
6.4.3 Intel Cluster Tools .................................................... 37
6.4.4 Allinea Optimization and Profiling Tool (OPT) .................. 38
6.4.5 Portland Linux Cluster Profiler (PGPROF) ....................... 39
6.4.6 Multi-processing Environment (MPE) ............................ 39
6.5 Source Code Documentation .......................................... 40
6.5.1 Understand for Fortran ............................................ 40
6.5.2 f90tohtml ............................................................ 40
6.5.3 f90doc ............................................................... 41
6.5.4 ROBOdoc ........................................................... 41
1 Introduction

This report is aimed at scientific software developers, using or looking to use Fortran 90 and 95 and examines the typical software development lifecycle suggesting tools that could be used at each stage. We have not looked at tools which focus on requirements gathering or high level design, since these are often common to many languages and most subject to personal preference on how (or even if) they are carried out.

We will use the ‘waterfall’ or ‘cascade’ software development cycle as a framework for the report. Briefly the parts of the software development cycle we shall consider are:

- design implementation - including coding and code quality assurance,
- source code control,
- compiling - including makefiles,
- testing,
- profiling.

These areas have been covered in previous reports [4, 5, 7] but it is time to bring these reports up-to-date and include new material and tools available to developers since their publication.

Since there is such a large number of tools, this review cannot include them all or even provide comprehensive details of those that we do discuss. However, it should give an idea of the range and capabilities of tools available at the time of writing.

This report is one of the outputs of the Software Engineering Support Programme (SESP). Some more detail on SESP is given in section 3.

2 The Fortran Language

Fortran continues to be the mainstay for implementing large scale computational modelling packages, despite the growth in the use of C and C++ within the scientific community. In this section we reflect on the history of Fortran and the features of the 2003 standard.

Fortran began life in the 1950s as a high-level language primarily aimed at mathematical formulae. The first standard for the language was brought out in 1966 to overcome a proliferation of dialects of the language that had built up in these early years. Even with the standard many dialects remained as well as extensions to the standard that provided essential features for large-scale programming. To overcome these difficulties a new standard known as Fortran 77 was published in 1978.

Fortran was aimed at scientific and numerical computing but other languages were arising that were more suitable for other application areas and they were being adopted in preference to Fortran. To maintain the superiority in scientific computing a new standard was developed to develop the language building on the experience of other languages. This new standard was Fortran 90. It introduced dynamic storage, whole array operations, modules, derived data types, pointers and new control structures, as well as allowing free format source code.

Following publication of the Fortran 90 standard there were two driving forces for change. The first was minor changes requested by users and the second was the development of High Performance Fortran (HPF). HPF was a set of extensions of Fortran making it possible to write portable code for use on parallel computers when handling problems involving large datasets. These extensions were mainly in the form of directives (in the form of Fortran comment lines) recognised only be a HPF processor. Not all of the desired features could be implemented using these directives so new syntax became necessary. These two drivers led to the creation of Fortran 95 a minor upgrade from Fortran 90.

Since the introduction of Fortran 95 there have been two official extensions ([1], [2]) that have been published as Type 2 Technical Reports (TRs). These provide features for
• Allocatable dummy arguments and type components, and

• Support for the five exceptions of the IEEE Floating Point Standard (IEEE 1989) and for other features of this Standard.

These are both included in Fortran 2003 as well as changes regarding object orientation and interfacing with C and other minor changes. The move towards object orientation in Fortran will be extended with type extension and inheritance, polymorphism, dynamic type allocation, and type-bound procedures allowing more class-like structures to be implemented. Interfacing with C is achieved by requiring types, variables and procedures to be ‘interoperable’. These are requirements of syntax so that the compiler knows at compile time whether an entity is interoperable.

Other changes and additions in the 2003 standard include:

• Derived type enhancements: parameterized derived types, improved control of accessibility, improved structure constructors, and finalizers.

• Data manipulation enhancements: allocatable components, deferred type parameters, VOLATILE attribute, explicit type specification in array constructors and allocate statements, pointer enhancements, extended initialization expressions, and enhanced intrinsic procedures.

• Input/output enhancements: asynchronous transfer, stream access, user specified transfer operations for derived types, user specified control of rounding during format conversions, named constants for preconnected units, the flush statement, regularization of keywords, and access to error messages.

• Support for international usage: access to ISO 10646 4-byte characters and choice of decimal or comma in numeric formatted input/output.

• Enhanced integration with the host operating system: access to command line arguments, environment variables, and processor error messages.

A full description of the new features of Fortran 2003 can be found in [3].

3 Software Engineering Support Programme

The Software Engineering Support Programme (SESP) (http://www.sesp.cse.clrc.ac.uk/) is an EPSRC support activity to provide and encourage the use of up-to-date software engineering techniques and tools in software development within computational science and engineering.

The main goals of this SES Programme are:

• accelerate the introduction and widespread use of high-payoff software engineering practices and technology by identifying, evaluating, and maturing promising or underused technology and practices;

• maintain a long-term competency in software engineering and technology transition;

• enable the UK academic community to make measured improvements in their software engineering practices by working with them directly;

• encourage the adoption and sustained use of standards of excellence for software engineering practice;

• foster collaborations with other groups, in the UK, Europe and the US, that have an interest in the applications of advanced software engineering techniques in computational science.
These goals will improve the level of software engineering practice within UK computational science research groups. As a result, the software they develop will be of a higher quality; more easily developed and maintained; more easily re-used within the community and be computationally more efficient.

The main thrust of the programme is to gather together processes and tools that will help improve software engineering in computational science. This can be characterised by the *Technology Watch, Assessment* and *Evaluations* process. Although the software engineering community has various very formally defined processes of software assessment and evaluation a rather more pragmatic approach has been defined for SESP.

**Technology Watch** – In each elements of the SESP information is gathered on a regular basis and a rolling update made to a Technology Report that would be made available to the community through the SESP Web site.

**Assessments** – The starting point of selecting a tool for use in anger is through paper assessment using a basic requirements document. The detail of the assessment would clearly depend on the area being addressed but a there will always be a collection of fundamental requirements such as operating systems, supported languages etc. These paper assessments can identify tools for practical evaluation and much of the material developed in the paper assessments added to the technology watch reports.

**Evaluations** – Through the assessment, various tools will be selected for more direct evaluation. They would be used in a realistic context either by SESP staff or those involved in the CCP and HEC programmes and their usefulness and effectiveness documented. Although in general the evaluations would not be placed on critical paths within the CCP or HEC activities, these programmes provide a considerable number of representative software packages that can be made the subject of an evaluation. The evaluations would lead to detailed reports and if successful the deployment of the tool or practice within the main stream.

At present the two major foci for the programme are on software quality assurance and transformation of legacy software.

### 4 Software Development

There are many possible definitions and descriptions of software development but a useful introduction can be found at [http://www.startvb.com/](http://www.startvb.com/) and the material we present in this section is based on this introduction.

‘Software development is the process of developing software through successive phases in an orderly way’. The obvious stage in the process is the actual writing of code but before this there are stages that include preparation of requirements and objectives, and the design of what is to be coded. Then following the coding stage the software must be tested to confirm that what has been developed has met the requirements identified at the outset. Before software development methods were recognised the development of new software often had very little a-priori requirements specification or design thought code did get written and tested to ensure it ‘worked’. Often the trouble would then be the lack of documentation of the methods used or choices made during implementation leaving those charged with maintaining such software with a learning curve steeper than it might otherwise have been! The complexity of modern systems and computer products long ago made the need clear for some kind of orderly development process.

The different phases involved in software development are as follows:

1. Identification of required software
2. Analysis of the software requirements
3. Detailed specification of the software requirements
4. Software design
5. Programming
6. Testing
7. Maintenance

The software development process is guided by some systematic software development method (SDM), also referred to by a number of terms, like process models, development guidelines, and systems development life cycle models (SDLC). There are many approaches to development taken by SDLC models. Some of these models include:

**The Waterfall Model** – A classic SDLC model, with a linear and sequential method that has goals for each development phase. The waterfall model simplifies task scheduling, because there are no iterative or overlapping steps. One drawback of the waterfall model is that it does not allow for much revision.

**Rapid Application Development (RAD)** – This model is based on the concept that better products can be developed more quickly by using workshops or focus groups to gather system requirements.

**Joint application development (JAD)** – This model involves the client or end user in the design and development of an application, through a series of collaborative workshops called JAD sessions.

**Prototyping Model** – In this model, a prototype (an early approximation of a final system or product) is built, tested, and then reworked as necessary until an acceptable prototype is finally achieved from which the complete system or product can now be developed.

**Synchronize-and-Stabilize** – This model involves teams working in parallel on individual application modules, frequently synchronizing their code with that of other teams and stabilizing code frequently throughout the development process.

**Spiral Model** – This model of development combines the features of the prototyping model and the waterfall model. The spiral model is favored for large, expensive, and complicated projects. *(where the costs and end date spiral out of control!)*

We shall consider the ‘waterfall’ or ‘cascade’ lifecycle model for software development illustrated in Figure 1 and give a brief description of the stages below.

**Feasibility** The feasibility study determines whether a particular development project should go ahead. If the project is to proceed then a project plan and budget estimate for the other stages of development will be produced.

**Requirements analysis** The requirements for the new or modified system are gathered at this stage. They should be recorded so that at the end of the project the software can be tested to ensure it fulfills the requirements.

**Design specification** Design focuses on:

- high level design, e.g. what programs are needed, what are their inputs and outputs, what are their interactions with other software or the operating system.
• low level design, e.g. how will the program work, what models or algorithms will be used, what libraries are required.
• data design, e.g. for input and output, data structures in the software.

The level of detail in the design may be a matter of personal choice or may be specified by particular development procedures. Having a detailed design will make generating the code easier but will make changing things difficult whereas a more broad brush design will leave more work in the implementation phase but allows room for the details to come out as the development progresses. Above all the design should be documented including reasons for making particular choices if a number of options were available. This makes it much easier for new developers to join a project and helps when new features are required.

**Coding** In this phase the designs are translated into code. Programming tools such as compilers and quality assurance tools are used to generate good quality source code and the software application. Testing of small self-contained parts (modules) of the overall application may take place depending on the modularity of the code.

**Testing** The overall system is tested to ensure that it works on the intended platform(s), giving correct results or showing the required behaviour defined in the requirements document. The use of debuggers and profiling tools will be useful at this stage to identify errors in the code and get the best possible performance from the code. Optimum performance is especially important in scientific computing applications.

**Maintenance** Once the system is delivered to users it will inevitably need maintenance. There may be bugs caused by unexpected input values (add them to a test suite) or by unexpected (inappropriate) use of the software (tighten up the documentation). In addition users will want more or different functionality and will definitely want it to run faster or address bigger problems! The software development process should be able to accommodate changes at this stage through a well thought out design and any changes should have their own requirements, design, coding and testing stages.

We are interested in tools aimed at the following stages of the cycle:
• Coding and unit testing
• Integration and system testing
• Delivery and maintenance (especially of legacy code).

For the remainder of this report we will take each of these stages and consider the tools that may help a developer produce high quality software.

5 Coding and Unit Testing

This stage can be broken down into five further stages:

• coding
• source code control
• constructing a build environment
• compiling
• testing

These sub-stages are addressed in the remainder of this section with descriptions of tools relevant to that section given as they arise.

5.1 Coding

This section addresses the issue of checking the quality of the source code.

The authors are not aware of any tool that can write source code for you (in the scientific software development domain) so typing the code is up to the developer!. The environment in which it is done is left to personal preference but various editors offer syntax highlighting which can help to flag typos in language reserved words. The use of the UNIX/Linux tool ctags can help minimise the risk of getting the wrong routine name or arguments. This tool builds a database of routines from the source files and editors such as (g)vim and (x)emacs can query the database to find the signature of a required routine.

When a set of routines has been written the natural thing to do is to try to compile it. However some compilers may miss certain errors, e.g. argument type mismatch between subroutine declaration and call that could seriously reduce the quality of the software. Further, in the rush to complete the coding, routines may have become overly complex and this will make the code difficult to maintain. Now is the time to make use of the tools listed below which can be used to detect faults, report on standard conformance, measure software complexity and/or restructure Fortran software.

A few of these are actually only applicable to Fortran 77 software. They are included because they offer some features which are not common in the Fortran 90 tools and may be useful in analysis of legacy code. In some cases an indication of cost is given, though many vendors do not put this information on their websites.

Also included here are some tools for the conversion of Fortran 77 to Fortran 90. These can help improve software quality through increased readability and maintainability.

5.1.1 Polyhedron plusFORT

Vendor
Polyhedron Software http://www.polyhedron.co.uk
Description

The plusFORT product consists of a set of tools for Fortran analysis checking. The tools currently available in Version 6 are:

**plusFORT SPAG.** SPAG, the plusFORT restructuring tool, can unscramble spaghetti Fortran 66 code, and convert it to structured Fortran 77. It also converts back and forth between standard Fortran 77, and code with VAX and Fortran 90 extensions such as **DO WHILE, ENDDO, CYCLE, EXIT and SELECT CASE.**

SPAG does not change the meaning of a program, or even the order in which statements are executed; it does change the way the program logic is written down, making it much easier to understand and maintain. Blocks of code are reordered so that logically related sections are physically close, and jumps in control flow are minimised. SPAG may also replicate small code fragments where this improves the re-structured code. SPAG computes complexity metrics before and after restructuring.

SPAG contains a powerful code beautifier, with dozens of options controlling spacing, case, labels, indentation, use of **CONTINUE** etc. You can use SPAG to switch back and forth between the F77 and F90 source forms. But SPAG goes much further:

- SPAG identifies, and optionally removes dead code (statements which could never be executed) and clutter (variables or **PARAMETERS** which are declared but never used).
- SPAG allows you to switch programs to explicit typing by adding declarations for implicitly defined variables. Explicit typing, which in Fortran is normally enforced using **IMPLICIT NONE**, allows your compiler to detect errors which might otherwise remain undetected for years.
- SPAG can rewrite your declarations from scratch using either Fortran 77 or Fortran 90 notation.
- SPAG provides a simple and safe method for systematically changing the names of symbols within a program.
- SPAG allows you to specify how upper and lower case should be used to distinguish different types of symbol. For example, local variables may be lower case, **PARAMETERs** upper case, dummy arguments capitalised etc. (e.g. local, PARAM, Dummyarg, **COM**monvar).
- SPAG generates symbol table files which are used by GXCHK (see next section), and may be used by user-written applications. For example it would be a simple matter to write a program to check conformance to local variable naming conventions. A detailed description of the symbol table format is supplied.

**plusFORT Static Analysis.** GXCHK is the plusFORT global cross check tool. GXCHK views data usage from a global perspective, and detects errors and anomalies that compilers and other tools miss. Examples are:

- Subprogram argument mismatch or misuse (e.g. constant actual argument is illegally modified by subprogram).
- **COMMON** variables assigned a value but never used, or used but never assigned a value.
- Globally unused **COMMON** variables, **COMMON** blocks, **PARAMETERs** and **INCLUDE** files.
- Inconsistent **COMMON** block definitions.
- **COMMON** variable name appears in other contexts (e.g. as a local variable, or in a different **COMMON**).
An unusual feature of GXCHK is its ability to analyse local and global data together. For example, if a local variable in subprogram A has the same name as a COMMON variable in subprogram B, it could be that the programmer has omitted the relevant COMMON statement from subprogram A. GXCHK, unlike other static analysers spots this problem. GXCHK also produces a call tree, and concise but comprehensive charts showing where and how each symbol is used. Handy aggregated reports are produced for COMMON blocks and INCLUDE files.

Interface specifications for every subprogram, showing calls in and out, dummy arguments, COMMON variables usage etc., can also be produced.

GXCHK operates in a ‘compile and link’ style which allows reports to be updated with minimal source code analysis. AUTOMAKE can be used to automate the process.

The GXCHK database can be queried interactively. This allows users instant access to information which is vital to program development and maintenance. For example, a programmer might set up a static analysis query window in a corner of the screen, and use it to find the consequences of a change to a COMMON block.

**plusFORT Dynamic Analysis** The Dynamic Analysis option of plusFORT is a tool for diagnosing these errors at run-time. Calls to probe routines are inserted in the source code before any operation which depends on the value of a data item, and the program is compiled and linked in the normal way. The executable code appears to the user to operate in exactly the same way as the original, but if a probe detects an undefined data item, it writes details to a log-file for later analysis. Source code for the probe routines is supplied.

**plusFORT Coverage Analysis.** The plusFORT coverage analysis facility places probes into Fortran source code which allow users to monitor the effectiveness of testing. At the end of each run, the probes update the coverage statistics for each source file. This data may be analysed at any time using the CVRANAL tool. CVRANAL identifies untested code blocks, and execution hot-spots.

In addition, CVRANAL can annotate your source code as shown below. The annotations are comments and do not affect the validity of the source code.

plusFort also includes the AUTOMAKE tool, a fully automatic alternative to the UNIX style MAKE utility, see 5.3.5.

**Cost**

The plusFORT set of tools costs between £695 and £5995, (excluding VAT) depending on platform and number of users. A 50% educational or non-commercial discount is available.

**Fortran 95 Features**

In addition to translating source form, declaration style, and control constructs to Fortran 95, SPAG can also translate COMMON blocks and INCLUDE files to Fortran 95 modules. SPAG can also create a module containing an INTERFACE block for each subprogram, and insert appropriate USE statements in calling subprograms. Fortran 95 compilers can use these constructs to perform argument checking.

5.1.2 Leiden University FORCHECK

**Vendor**

Forcheck Group, Leiden University [http://www.forcheck.nl/](http://www.forcheck.nl/)
Description

FORCHECK supports Fortran 77, Fortran 90, Fortran 95 and High Performance Fortran. It also supports many popular extensions. The tool provides:

- Call tree generation;
- Cross reference tables for variables, subprograms, COMMON blocks, etc.;
- Source code listing with error and warning messages inserted.
- Inter-procedure argument checks;
- Static checks for variables used before set, unused variables, etc.;
- A summary report of the code analysis;
- Consistency of COMMON blocks and usage of INCLUDE files;
- Verification of public module variables.

A wide range of compilers (and most of their extensions) are supported. These include Cray, IBM, DEC, SGI and SUN Fortran 90 compilers. Several Fortran 95 compilers, such as those from Lahey and Salford, are also supported. The software runs on many different Unix systems as well as DOS and Windows. A graphical user interface is available on some systems and a command line interface on all systems.

Cost

The software runs on many systems and prices range from 425 Euros on a DOS-based PC to 4750 Euros for a site license (excluding VAT). A 40% academic discount is available.

5.1.3 Cleanscape FortranLint

Vendor

Cleanscape Software International

http://www.cleanscape.net/products/fortranlint/index.html

Description

Full details of Cleanscape FortranLint can be obtained from the web site. FortranLint includes a source code analyser that can detect a wide range of potential problems, including:

- Inappropriate arguments passed to functions;
- Inconsistencies in common block declarations;
- Non-portable code;
- Type usage conflicts between different subprograms;
- Unused functions, subroutines, and variables;
- Variables which are referenced but not set;
- OpenMP support.

FortranLint can be used to:

- Check source files before they are compiled;
• Map out unfamiliar programs. This would include call graph generation and cross-referencing;

• Enforce programming standards.

As well as standard Fortran 90, the tool also supports many of the vendor extensions. In particular, there is support for D.O.D Mil-Std 1753, VAX/VMS Fortran, Cray, Microsoft, Lahey compiler extensions. The tool can also check details of HPF ‘comments’, if requested to do so.

To make identification of important errors easier, selected warnings can be suppressed. A graphical interface is also included (under X-windows and MS Windows) which can be used to browse the source along with the warning messages and call tree.

5.1.4 NAG Fortran 90 Tools

Vendor
The Numerical Algorithms Group Ltd http://www.nag.co.uk

Description
In the past NAG has provided software tools for analysis and transformation of Fortran 77 software. They offer a similar set of tools for Fortran 95. These can be run through a GUI which allows easy selection of the source files and operations to perform on them.

The individual tools available include the following.

Polish – for ‘pretty printing’ source code. This includes consistent indentation of loops and conditional statements. Keywords and variables can be individually mapped to upper or lower case to enhance readability. A graphical interface allows simple editing of the formatting options.

Declaration standardisation – this splits up the declarations into groups for easier understanding. Clear comment sections separate scalar arguments from array arguments, local scalars, etc.

Name changer – to alter variable and subprogram names. This allows variable names to be changed to more meaningful strings. It offers a safer conversion process than global edits which might change unintended substrings.

Dependency analysis – generates makefiles for a set of Fortran 95 files. With the addition of modules to the language, combined with the existing problem of “include” files, it is a complex task to include all file dependencies in a makefile. This can lead to difficult-to-find problems when out of date modules are used by mistake. This tool generates a full set of dependency rules from the selected Fortran files.

Call graph generator – this generates a traditional call tree for a given set of Fortran 95 files, as an aid to documentation and understanding of the software.

Interface builder – this creates explicit interface blocks for Fortran routines which currently only use implicit ones. This enables the Fortran 95 compiler to do much more argument checking of procedure calls. A new module file containing the interface blocks can be generated and appropriate USE statements are inserted in the calling routines. The ONLY qualifier is added to the interface block USE so that it is clear which routine(s) have been defined. In the case of calls to library routines, which are not defined in the code, interface blocks can still be generated, though the user must check that these are indeed correct.
**USE statement annotator** – this can add appropriate ONLY clauses to USE statements. This makes the way data is imported more explicit to anyone reading the code. It also reduces the chances of subsequent updates to the software using module variables by mistake.

**Common Block to Module Converter** – this removes common blocks and replaces them with USE statements referencing a module generated by the tool containing the global variables previously declared in COMMON statements.

A similar set of NAG tools is available for Fortran 77 software. These include a restructuring tool, similar in operation to SPAG, which attempts to replace old style coding with more readable structures such IF-THEN-ELSE blocks. With the two sets of Fortran tools comes a converter from Fortran 77 to Fortran 95. This includes a graphical user interface and takes advantage of the new free format and declaration style of Fortran 95.

These tools should provide a good path for transforming legacy Fortran software to Fortran 95 using a well structured and easy to read format. The addition of explicit interfaces to all subprogram calls should significantly increase the confidence in the quality of the software, as it allows immediate detection of linkage errors.

### 5.1.5 Cobalt Blue - FOR_STRUCT, FOR_STUDY

**Vendor**
Cobalt Blue Inc [http://www.cobalt-blue.com](http://www.cobalt-blue.com)

**Description**
Cobalt Blue have two products which can be used for Fortran software analysis. There is some support for Fortran 90, but most of the documentation refers to Fortran 77. The tools available are:

**FOR_STRUCT.** This product is mainly described in terms of its capabilities to restructure old Fortran 66 code, removing spaghetti code and replacing VAX extensions with standard Fortran 77. However, it also has some support for Fortran 90, including SELECT CASE and relational operators. A demo version is available, limited to 100 lines of code.

The extent of Fortran 90 support in the demo version is very limited. While it accepts SELECT statements and also generates ENDDO for loops, it rejects all MODULE statements.

**FOR_STUDY.** This tool provides analysis of Fortran software. It includes global argument checking, common block checks and various other potential problems. Call graphs may also be generated. The output reports of FOR_STUDY can be generated in HTML format.

Though there is explicit mention of Fortran 90 feature support, it seems that these do not extend to many important new features such as modules. This makes it of limited use in new Fortran 90 development work.

### 5.1.6 N.A. Software Loft90, FortranPlus

**Vendor**
N.A. Software Ltd [http://www.nasoftware.co.uk](http://www.nasoftware.co.uk)

**Description**
N.A. Software have in the past marketed a conversion tool (F77 to F90), and currently sell a Fortran 95 compiler which includes some global analysis and error checking. These tools are:
Loft90. This product was used to convert Fortran 77 code to Fortran 90 format and to take advantage of the new and safer features of the more recent standard. However, N.A. Software have stopped development of this product and no longer actively sell it.

FortranPlus This is a Fortran 95 compiler for Windows 9x and NT and Linux systems. It supports a number of useful features including interface checking, even when implicit interfaces have been used, and generation of cross reference listings of all variables. A graphical debugger is also part of the system and a program development system keeps track of which files have been recompiled and the options used. New features such as IEEE arithmetic and exceptions and the ISO varying string module are supported.

5.1.7 Crescent Bay Software VAST and DEEP

Vendor
Crescent Bay Software http://www.crescentbaysoftware.com/

Description
The VAST system uses very high-level analysis and optimization to go beyond what normal compiling systems can accomplish. It examines entire loop nests in great detail, and can restructure them to suit the target architectures of many kinds of computer systems. VAST contains sophisticated data dependency analysis techniques to detect when optimized execution will be safe, has very advanced inlining abilities, and uses interprocedural analysis to optimize across procedure boundaries.

VAST versions exist for optimizing C, C++ and Fortran. Target platforms include SMP systems, distributed memory parallel systems, superscalar systems, vector and parallel-vector systems and many others.

This software tool provides translation of Fortran 77 software to Fortran 90. In addition to the relatively simple conversion of fixed form to the new free form, a number of more complex operations can be performed including:

- Addition of INTENT statements to subprogram arguments;
- Removal of obsolete features such as arithmetic IF, etc.;
- Elimination or reduction in the number of GOTO’s. In the same way that SPAG restructures Fortran 77, VAST/77to90 attempts to simplify code. This is useful in understanding old software and making future maintenance easier. Unnecessary labels are also removed from DO loops;
- Use of vector notation where possible. Many Fortran 77 loops over vector and matrix subscripts can be reduced to the new and compact array notation available in Fortran 90. As well as often making the code easier to read, this can help optimising compilers identify potentially parallel operations. The tool has options for conservative or aggressive transformation to array notation. In the latter case the code may be harder to read but offer more potential opportunities for parallelism;
- Creation of modules to replace common blocks. Modules offer a safer mechanism for sharing data and VAST/77to90 can automatically generate separate module files. Commons that are in INCLUDE files are also correctly converted;
- Automatic generation of interface blocks. Using explicit interfaces for all subroutine and function calls allows the Fortran compiler to catch errors in argument lists. As well as creating the necessary interface details in a module, appropriate USE statements can be inserted into each calling subprogram;
• Fortran ‘lint’ type diagnostics are generated during the conversion process. This includes warnings about variables used but never set, unused variables and unused code.

VAST/77to90 is available on many systems, including Mac OS X, Linux (x86 and PowerPC), SUN/Sparc Solaris, IBM RS/6000 and SP2 (AIX), HP 9000/700 series (HPUX), DEC Alpha (Unix), SGI (IRIX, Altix), Windows/NT.

Under Windows NT, VAST/77to90 includes a graphical user interface which makes all the processing options easily available for user customisation. In other versions all the options are set via command line switches.

Crescent Bay Software also market DEEP (Development Environment for Parallel Programming). This is a type of Integrated Development Environment (IDE) for parallel programs in Fortran (77, 90, 95) and C. It includes various analysis and debugging tools to help understand the behaviour of parallel codes. While this tool is mainly aimed at parallel optimisation, it also includes call graph display, source code browsing and debugging features useful in software quality assurance.

Cost

VAST/77to90 now comes with a metered license, with cost based on the number of lines converted. For up to 5000 lines the US cost is $450 (January 2005) rising to 500,000 lines at $10,000 and unlimited use at $16,000. A 10% academic discount is available.

5.1.8 Fujitsu Visual Analyzer

Vendor


Description

Lahey provide a sophisticated Fortran 95 compiler that, in the Enterprise and Professional versions, includes the Fujitsu Visual Analyzer. This product provides flow graph analysis for Fortran 95 (and C) software. It can also provide global tracking of common and module usage. An error checking option allows global analysis to detect static errors. Conformance to the Fortran 95 standard is also checked.

This product is only available for Windows NT/95, though it does include a graphical user interface. The single user license for the compiler is $1095 for the Enterprise version and $795 for the Professional edition in the USA.

5.1.9 Simulog FORESYS

Vendor

Simulog, http://www.simulog.fr/is/fore1.htm

Description

The FORESYS product (FORtran Engineering SYStem) is a set of tools that cover analysis, inspection, restructuring, parallelisation and interfacing to CASE tools. It provides the following features:

Code restructuring –

• Converts old code to Fortran 90 or portable extended F77 (with WHILE and INCLUDE)
• Regeneration of declarations. Declaration with attributes are used in Fortran 90
• Control structures like: goto and arithmetic if are updated to: IF...THEN...ELSE, DO...WHILE, SELECT...CASE
• Unused code is removed while combined logical tests are streamlined
• Variables rename feature, alteration of variables types or sizes
• Creation of Fortran 90 modules containing the interface to each program unit called within the code and COMMON blocks extracted from the source code. COMMON blocks can also be extracted from source code and saved as include files in Fortran 77
• Replacement of non-standard intrinsic functions by their generic equivalent whenever possible

Advanced interprocedural analysis –

• Tracks all inconsistencies between sub-program definition and call: missing arguments, wrong types, array shape mismatch, passing constants to output parameters. COMMON blocks consistency checks: name, type, size and order mismatch in common block variable are detected. Global data-flow analysis: tracks what arguments or global variables are input or output to any given sub-program. Works for F77 as well as for Fortran 90: intent attribute is checked but not required
• Alias and pointer analysis integrated with data-flow analysis: works with pointers and equivalence

Software metrics – Including:

• Correctness metrics: number of error and warnings, diagnostic density
• Textual complexity metrics: SLOC, Halstead
• Documentation metrics: comment number, density and distance
• Structural complexity: decisions, loops, McCabe cyclomatic complexity
• Interprocedural complexity: callers, callee

All these tools can make use of a graphical user interface. Supported systems include Digital Alpha/Digital Unix, HP PA/HP-UX, IBM RS6000/AIX, SGI/IRIX, SUN Sparc/Solaris.

5.1.10 Salford – CHECKMATE

Vendor
Silverfrost Limited
http://www.silverfrost.com/15/ftn95/checkmate_worlds_best_runtime_checking.asp

Description
CHECKMATE is used through the Salford Fortran compiler FTN95. Switches set on the compiler make it produce an executable that performs checks at runtime. The following is a list of some of the things that are checked:

• Check that subroutines/functions are called with the right number of arguments of the right type. This check works even in traditional Fortran programs which have not used the INTERFACE statement or in programs constructed from several files with inconsistent INTERFACE statements
• Checks that array subscripts do not go out of bounds. FTN95 performs this check even for arrays declared with bounds of (*) or assumed-size or ALLOCATE’d arrays. The only requirement is that in the case of argument arrays the caller must also have been compiled in CHECK mode (normally entire programs are compiled in CHECK mode anyway).
• Checks that integer and floating point calculations do not overflow

• Using the /UNDEF option you can check that a program does not use variables before they have been defined. Although the compiler can warn in simple cases, often it is only at run-time that this problem can be detected. The undefined check works with everything - common data, local variables, allocated data, even individual elements of an array (which can be partially undefined)

• Checks that arguments passed to routines as constants are not treated as variables and altered by the callee (or anything that it calls). The effect of doing this is normally very insidious, usually the constant is overwritten, corrupting the program at a later point

• Checks that whole array operations on arrays where conformability cannot be checked at compile time are in fact conformable

• Checks that unavailable objects - such as arguments that are not available through the ENTRY point in use, or absent optional arguments, or unallocated ALLOCATABLE objects - are not in fact used

• Checks that references are not made through 'dangling' or unassigned pointers

5.1.11 LDRA - Testbed

Vendor

LDRA Software Technology http://www.ldra.com

Description

The LDRA software tool Testbed is available on a wide range of platforms and for many programming languages including Fortran. The two main areas of quality that Testbed addresses are static analysis and coverage analysis. However, LDRA have no plans for a Fortran 90 version, so it is included here only for comparison with Fortran 90 tools.

Static analysis The functions provided by the static analysis tool include:

• Adherence to language and user defined programming standard;
• Level of structured programming used;
• Complexity metrics such as McCabe;
• Static data flow analysis and procedure interface analysis;
• Code re-formatting and variable cross referencing.

Coverage analysis After static analysis, Testbed can be used to insert instrumenting statements into the software. The modified version of the software can then be run on standard test cases to find which paths through the software have not been tried out. Detailed reports on how often each block of code is executed can be produced.

5.1.12 Programming Research Ltd – QA Fortran

Vendor

Programming Research Ltd http://www.programmingresearch.com
Description

QA Fortran is an X Windows based toolset, which provides extensive quality assurance testing for Fortran applications. Examining source code in greater depth than any compiler, it detects programmer errors, identifies unreliable practices and objectively measures software quality to aid management of development and maintenance projects. Software characteristics (metrics) are extracted by QA Fortran and presented in a graphical, easy to understand manner. The software can check for adherence to local programming standards.

Fortran 90 is not supported by QA Fortran, and they do not have plans to add it at the present time.

5.1.13 Other QA tools

There are a number of QA tools providing similar functionality to QA Fortran and LDRA Testbed for Fortran 77 software, such as Logiscope (from Verilog, France), Analysis of Complexity Tool (from McCabe and Associates, USA), and MALPAS (from TACS, UK).

5.1.14 MPI Check

Vendor

High Performance Computing Group, Iowa State University
http://andrew.ait.iastate.edu/HPC/MPI-CHECK.htm

Description

MPI-CHECK is a tool developed to aid in the debugging of MPI programs that are written in free or fixed format Fortran 90 and Fortran 77. MPI-CHECK provides automatic compile-time and run-time checking of MPI programs. MPI-CHECK automatically detects the following problems in the use of MPI routines:

- mismatch in argument type, kind, rank or number;
- messages which exceed the bounds of the source/destination array;
- negative message lengths;
- illegal MPI calls before MPI\_INITIALIZE or after MPI\_FINALIZE;
- inconsistencies between the declared type of a message and its associated DATA\_TYPE argument;
- actual arguments which violate the INTENT attribute.

Full details may be found in [6].

5.1.15 MARMOT

Vendor

MARMOT is being developed within the European CrossGrid project. Its website is http://www.hlrs.de/organization/tsc/projects/marmot/
Description

MARMOT surveys the MPI-calls made and automatically checks the correct usage of these calls and their arguments during runtime. It does not replace classical debuggers, but can be used in addition to them.

Currently MARMOT supports the C and Fortran language binding of the MPI-standard 1.2. MARMOT is a library that is linked to the MPI-application in addition to the existing MPI-library and that allows a detailed analysis of this application at runtime. It generates a human-readable log file that includes:

- Violations of the MPI-standard are reported as error.
- Unusual behaviour or possible problems are reported as warnings.
- Notes are displayed when harmless but remarkable behaviour occurs.
- The MPI-calls are traced on each node throughout the whole application.
- When detecting a deadlock the last few calls (as configured by the user) can be traced back on each node.

5.2 Source Code Control

The most important information in a software development project is the source code. It takes a great deal of effort to create (Even from the best of designs) and it must be kept secure and incorruptible as far as possible. The major potential problem with source code is that it can be changed with no record kept or note made about why the change was necessary. Source code control cannot stop this occurring but does prompt developers to record what they did and why and automatically documents exactly what was changed. This gives developers an audit trail on the changes, enables rollback of source if a bug is found and even the ability to re-create old version of the software.

The long established technologies in this field are SCCS, RCS and CVS and there are brief descriptions of how to use SCCS, RVS and CVS in [7]. Newer systems are described below, including Subversion and Perforce. A comparison of source code management (SCM) tools can be found at http://better-scm.berlios.de/comparison/comparison.html.

5.2.1 SCCS

The Source Code Control System was originally developed at the Bell Telephone Laboratories. It is a standard part of Unix System V and comes with every flavour of Linux.

5.2.2 RVS

Revision Control System was developed after SCCS in the early 1980s. It is generally easier to use than SCCS and more powerful. It may also be faster at retrieving a file from the repository though mostly the time difference will not be noticable.

5.2.3 CVS

The Concurrent Versions System is built on top of RCS. It is very versatile and complex, providing not only an audit trail of changes made to the code but allowing collaborative development not dependent on file locking as in RCS.

There are many GUIs for CVS clients for both Unix/Linux and Windows and it comes built in to Java IDEs such as NetBeans and Eclipse.


TkCVS – http://tkcvs.sourceforge.net/ – Client written in Tcl/Tk


TortoiseCVS – http://www.tortoisecvs.org/ – TortoiseCVS lets you work with files under CVS version control directly from Windows Explorer. It’s freely available under the GPL.

5.2.4 Subversion

An open source project with website http://subversion.tigris.org/.

Description

The goal of the Subversion project is to build a version control system that is a compelling replacement for CVS in the open source community. Subversion features:

- Most current CVS features.
- Directories, renames, and file meta-data are versioned.
- Commits are truly atomic.
- Branching and tagging are cheap (constant time) operations.
- Choice of database or plain-file repository implementations.
- Versioning of symbolic links.
- Efficient handling of binary files.

There are a many clients for Subversion. Some to note are

RapidSVN – http://rapidsvn.tigris.org/ – A cross-platform GUI front-end for the Subversion revision system written in C++ using the wxWidgets framework. This project also includes a Subversion client C++ API.


5.2.5 Perforce

Vendor

Description

The Perforce SCM System features comprehensive software configuration management capabilities built around a scalable client/server architecture. Requiring only TCP/IP, developers can access the Perforce Server through a variety of Perforce clients (GUIs for several platforms, Web, or Command-Line). Perforce can be deployed quickly and easily, and requires minimal administration, even for large or distributed sites.

Available on over 50 operating systems, Perforce includes version control, workspace management, atomic change transactions and a powerful branching model to develop and maintain multiple codelines.

All Perforce software is free and fully functional, with the exception of the Perforce Server, which allows only two users and two client workspaces when used without a license. A Perforce license enables the Perforce Server to support more users and an unlimited number of workspaces. It also entitles you to Perforce Technical Support. Prices are available from the Perforce web site along with details of educational discounts and free licensing for open source development.

5.3 Build Environment

In this section we will describe tools to help compile source code into an application. The obvious choice is to use make that comes with all flavours of UNIX and Linux but there are other technologies and tools that could be used. One in particular ‘ant’ has gained popularity, in part through its transparent use in Java IDEs such as Sun Java Studio, NetBeans and Eclipse.

5.3.1 Make and Associated Tools

Make and makefiles

Every programmer knows make and how useful it is to automate the build process for large projects. There are many tutorials and books on how to use it and it has its own section in [7]. Make uses makefiles to control the compilation and building of executables and libraries from source files. These makefiles can be made portable in many ways. See [8] for a procedure that has been used to provide portable compilation for a parallel subroutine library.

Checking source dependencies

There are a number of tools to automatically check dependencies of Fortran source files. They provide the dependency information either as part of a makefile that is written to compile the source or as a separate file which must be copied into an existing makefile. A list of some of these tools in given below.

makemake – http://www.fortran.com/fortran/makemake.perl – A Perl script which attempts to automatically generate a makefile from a given set of C, Fortran 77 or Fortran 90 source files. Generates a dependency tree for Fortran modules and include files that is copied into a makefile.


fmkmf – http://www.geos.ed.ac.uk/abs/homes/hcp/fmkmf/index.html – A Perl script of shell script that constructs a makefile for a Fortran 90 (or 95) program that uses modules.

fimake – http://marine.rutgers.edu/po/tools/perl.html – Contains configuration files for a large number of compilers.

**makedepf90** – [http://www.helsinki.fi/~eedelman/makedepf90.html](http://www.helsinki.fi/~eedelman/makedepf90.html) – A C program for automatic creation of Makefile dependency lists for Fortran source code. Makedepf90 supports **MODULEs**, **INCLUDEs**, cpp `#includes`, f90ppr `$includes` and coco `??includes` and set-files.

**Maketool**


**Description**

Maketool is a simple GTK+ based GUI front end for GNU make and other make programs. Maketool's features include:

- Figures out what targets are available and presents them in a menu.
- Runs make and detects compiler errors and warnings in the output.
- Double-clicking on errors starts an editor with that file and line.
- Works with any makefile system that uses GNU make.
- Automatically updates Makefile if project uses automake, autoconf or imake.
- GUI for selecting configure options for automake and autoconf.
- Save and reload logs of make runs.
- Make in series or in parallel, (using make's `-j` and `-l` flags).
- Now also supports Solaris make, IRIX `smake`, and BSD `pmake`.

**5.3.2 Apache Ant**

**Description**

**Apache Ant** – [http://ant.apache.org/](http://ant.apache.org/) – A Java-based build tool. In theory, it is kind of like Make, but without Make's wrinkles. Ant is different. Instead of a model where it is extended with shell-based commands, Ant is extended using Java classes. Instead of writing shell commands, the configuration files are XML-based, calling out a target tree where various tasks get executed. Each task is run by an object that implements a particular Task interface. This removes some of the expressive power that is inherent by being able to construct a shell command, but it gives you the ability to be cross platform – to work anywhere and everywhere. If you really need to execute a shell command, Ant has an `<exec>` task that allows different commands to be executed based on the OS that it is executing on.

**Antelope** – [http://antelope.tigris.org/](http://antelope.tigris.org/) – A graphical user interface for running Ant. While new to tigris.org, Antelope is a mature application that is in wide use as a front end to Apache Ant. Antelope provides a set of buttons, one per target, which makes it easy to start a specific Ant operation. Basically, an Ant build file is opened with Antelope and the targets defined in the build file can be executed by clicking a button. Antelope is much more than a thin wrapper around Ant, however. Antelope can run in two different environments. It can run as a stand-alone GUI application and as a plugin to jEdit ([www.jedit.org](http://www.jedit.org)), an excellent open source editor for programmers.

As a stand-alone application, Antelope provides the ability to create and edit Ant build files, the ability to run build file targets, and can trace target and task execution in a
quasi-debug mode. Output from the build process can be captured to a file and/or to screen and performance statistics can be gathered to guide optimization efforts. Several very useful tasks to extend Ant are included.

As a jEdit plugin, Antelope expands the features of the stand-alone application to nicely integrate with the text editor and with the Console and ErrorList plugins. Antelope has an open API that allows other plugins to utilize Antelope’s capabilities.

In either mode, Antelope provides property and reference inspection and the ability to set properties that would be passed as parameters to Ant on the command line. The Antelope Project also provides a set of additional tasks that provide functionality not found in the standard tasks distributed with Ant.

5.3.3 Jam

http://www.perforce.com/jam/jam.html

Description

Jam is a software build tool that makes building simple things simple and building complicated things manageable. It has been freely available as C source for many years from the Perforce Public Depot and is widely used to build commercial and academic software. Jam is a very good solution for conventional C/C++ compile-and-link builds.

Because Jam understands C/C++ dependencies, there is no need to declare header or object files. The built-in Jam rule ‘Main’ handles header file dependencies and object files both automatically and on-the-fly.

Before any targets are updated, Jam gathers complete dependency information for C/C++ source files, allowing Jam to:

- Build as much as possible, instead of halting on the first build error.
- Avoid building targets if targets on which they depend fail to build.
- Build across parallel paths with multiple, concurrent processes.

From the information on Jam it is clear that it is aimed only at C and C++ and will not understand dependencies in Fortran files.

5.3.4 SCons

http://www.scons.org/

Description

SCons is an Open Source software construction tool, that is, a next-generation build tool. Think of SCons as an improved, cross-platform substitute for the classic Make utility with integrated functionality similar to autoconf/automake and compiler caches such as ccache. In short, SCons is an easier, more reliable and faster way to build software.

Features of SCons include:

- Configuration files are Python scripts – use the power of a real programming language to solve build problems.
- Reliable, automatic dependency analysis built-in for C, C++ and Fortran – no more ‘make depend’ or ‘make clean’ to get all of the dependencies. Dependency analysis is easily extensible through user-defined dependency Scanners for other languages or file types.
- Built-in support for C, C++, D, Java, Fortran, Yacc, Lex, Qt and SWIG, and building TeX and LaTeX documents. Easily extensible through user-defined Builders for other languages or file types.
- Building from central repositories of source code and/or pre-built targets.
- Built-in support for fetching source files from SCCS, RCS, CVS, BitKeeper and Perforce.
- Improved support for parallel builds – like `make -j` but keeps N jobs running simultaneously regardless of directory hierarchy.
- Integrated Autoconf-like support for finding `#include` files, libraries, functions and type-defs.
- Global view of all dependencies – no more multiple build passes or reordering targets to build everything.
- Ability to share built files in a cache to speed up multiple builds – like ccache but for any type of target file, not just C/C++ compilation.
- Designed from the ground up for cross-platform builds, and known to work on Linux, other POSIX systems (including AIX, *BSD systems, HP/UX, IRIX and Solaris), Windows NT, Mac OS X, and OS/2.

### 5.3.5 Polyhedron Software AUTOMAKE

**Vendor**

Polyhedron Software [http://www.polyhedron.co.uk/pf/pfother.html#automake](http://www.polyhedron.co.uk/pf/pfother.html#automake)

**Description**

AUTOMAKE is part of Polyhedron’s PlusFORT tool and can be regarded as a fully automatic alternative to the UNIX style MAKE utility. Unlike MAKE, AUTOMAKE builds and maintains the dependency database (or ‘make-file’) itself. This removes a major source of error: users of a conventional MAKE may omit a dependency or specify it wrongly. If that happens (and anyone can make a mistake!), MAKE may appear to update a program correctly, while in fact building a corrupt hybrid of new and old code. The consequences may not be immediately apparent, and can be extremely hard to trace. This is precisely the sort of problem that MAKE is supposed to avoid.

AUTOMAKE users do not have that problem because dependency data is maintained automatically by AUTOMAKE. AUTOMAKE checks for new or deleted source files, and scans source code as necessary to update `INCLUDE` file dependencies.

AUTOMAKE is Fortran 95 aware, recompiling modules from the bottom up, taking full account of module dependencies.

### 5.3.6 Checking platform dependencies

**Automake and Autoconf**

**Description**

When the correct compilation of software depends on libraries and header files not provided by the software then it is a good idea to be able to automate checks for the existence (and correct versions) of these files. Many source distributions of linux tools come with a `configure` script to check the platform dependencies and create a makefile that will compile the program on that
platform. This script is built using autoconf and the makefile is created from a template that is made using automake.

The use of these tools is beyond the scope of this report but a web search on autoconf will provide many pages of introductory and tutorial material.

**Buildtool**

http://buildtool.sourceforge.net/

**Description**

This is another tool that performs the same function as automake and autoconf and has the following features:

- **Automatic configuration scripts**: Buildtool lets you write simple, but yet powerful, automatic configuration scripts. These are used to check for features of the build system and adapt the software package to build properly in it.

- **Build scripts**: With Buildtool you can also write build scripts. These are used to build and install a software package in the build system. They let you compile any kind of package, ranging from standalone utilities to shared libraries, and in a very simple way, which hides all details of the build host.

**Cleanscape qef**

**Vendor**

Cleanscape Software International http://www.cleanscape.net/stdprod/qef/qef.html

**Description**

qef from Cleanscape is a high-level software construction system. qef provides a complete program development environment which operates consistently across UNIX and Windows NT platforms. qef can be used to manage, control, and configure the software construction process. qef includes tools that replace make, Imake, source code management packages, dependency generators, etc.

### 5.4 Compiling

There is an excellent introduction to compilers and compiling in [7] from which the main message is “get to know each compiler well and play with the options to get the most optimised code”

A good comparison of many Unix and Windows Fortran compilers is given on the Polyhedron web site: http://www.polyhedron.co.uk. They include a set of diagnostic tests to compare how effective each compiler is at trapping common errors.

We cannot provide an exhaustive list of Fortran compilers here but we do list below some of the more well known ones.

#### 5.4.1 Absoft Pro Fortran

**Vendor**

Absoft Corporation http://www.absoft.com/Products/Compilers/compilers.html
Description

Windows –

- Highly optimized ANSI/ISO F95 and F77 compilers for 32-bit x86-compatible processors with specific optimizations for Intel/P4 and AMD/Athlon processors yielding competitive performance
- C/C++ compiler is included for mixed language development
- Compilers support all popular extensions and are link/debug compatible with Microsoft C
- Complete Integrated Development Environment (IDE) with color programmer’s editor, debugger, and ability to execute select third party programs as plug-ins. All Pro Fortran compilers can be executed equally well from the IDE or command line
- Pre-configured libraries for linear algebra (BLAS, LAPACK95, ATLAS, HDF), VAX/Unix-compatibility and 2D/3D graphics are all included on the distribution CD at no additional charge
- Optional tools are available for automatically converting serial code to code designed for dual processors
- Numerical libraries with special functions for mathematics and statistics; and Visualization libraries with special numerical and visualization functions are available
- Source compatible versions available for Macintosh, Windows and Linux.

Linux – 32-bit and 64-bit versions available

- Industry leading performance and optimizations
- Easy installation for any Linux distribution. ‘Just Installs’ without modification on the latest 32-bit or x86_64 Linux implementations from Red Hat, SuSE, Mandrake, and other major builds
- Conforms to the Linux Standard Base specification
- Compatible with glibc, gcc, g77, f2c, the GNU linker, debugger, assembler and make utility
- Fortran compiler operates in F95, F90 or F77 modes
- Includes Fx2 multi-language debugger for Fortran, C, C++, Assembler
- Multi-processing options including automatic parallelization and OpenMP tools
- Compiler generates 32-bit executables
- 64-bit version compliant with the x86-64 ABI
- 64-bit compiler takes advantage of full 64-bit memory model
- 64-bit compiler generates 32-bit and 64-bit executables (Note: the compiler itself runs only on systems using Intel EM64T 64-bit, AMD Opteron 64-bit and AMD Athlon 64-bit processors)

5.4.2 Compaq Visual Fortran

Vendor

HP http://www.hp.com/go/fortran/
Compaq Visual Fortran for Windows 2000, NT, ME, 98 and 95 systems is a complete development system that includes Compaq’s industry-leading native Fortran 95 compiler, the latest visual development environment from Microsoft, full access to the Win32 Application Programmer Interfaces (APIs), Microsoft Fortran PowerStation compatibility, and support for numerous de facto industry-standard Fortran language extensions.

Compaq Visual Fortran shares a common code base with Compaq’s Fortran 95 compilers for Tru64 UNIX, Linux Alpha and OpenVMS Alpha systems, giving broad source compatibility and rapid advancement in features and performance. The use of other languages in technical development such as C++ demands that Fortran supports mixed-language application development. Because Compaq Visual Fortran includes Microsoft’s Visual C++ 6.0 development environment and ‘Developer Studio’ tool set, Compaq has the unique advantage among PC Fortran vendors that it supports mixed-language development with Microsoft Visual C++ 6.0 within a common development environment.

Other compilers in this family are available from HP for Tru64 UNIX Alpha, Linux Alpha, and OpenVMS (Integrity / Alpha / VAX).

5.4.3 Intel

Vendor


Description


Linux – Capitalize on the Compaq Visual Fortran front-end language features and Intel back-end code-generation and optimization features of Intel Fortran Compiler for Linux, including support for Intel Extended Memory 64 Technology (Intel EM64T). Take maximum advantage of legacy code and software investments through compatibility with commonly used GNU development tools such as linkers and debuggers; mixed language support for Fortran and C; and adherence to the latest ISO Fortran 95 standard. Perform threaded application development and optimization with support for the OpenMP 2.0 standard and the auto-parallelism feature.

5.4.4 Lahey/Fujitsu

Vendor

Lahey Computer Systems Inc. [http://www.lahey.com/]

Description

Windows – Lahey/Fujitsu Fortran v7.1 –

- LF Fortran v7.1 *Enterprise edition* integrates the Fortran for .NET language system and the Lahey/Fujitsu Fortran 95 language system into the latest Microsoft Visual Studio development environment.
- LF Fortran v7.1 *Express edition* includes the command-line Lahey/Fujitsu Fortran 95 optimizing compiler, linker, command-line debugger, and librarian.
- LF Fortran v7.1 *Academic edition* is a plug-in to Visual Studio .NET 2003 and includes the Lahey/Fujitsu Fortran for .NET language system. The Academic edition requires a membership in Microsoft’s Academic Alliance.

Linux – Lahey/Fujitsu Fortran 95 v6.2 –

- LF95 *Express* includes the powerful Lahey/Fujitsu Fortran 95 optimizing compiler, command line debugger, link-compatibility with g77 and egcs, online documentation, and free e-mail support.
- LF95 *PRO* adds auto-parallelization, OpenMP compatibility, the Winteracter Starter Kit, WiSK, for creating Windows GUIs and displaying graphics, thread-safe BLASs and LAPACK, Polyhedron’s Automake utility, the Fujitsu SSL2 math library (thread-safe for parallel applications), and free phone support.

5.4.5 NAGWare f95 Compiler

**Vendor**
The Numerical Algorithms Group Ltd [http://www.nag.co.uk/nagware/NP.asp](http://www.nag.co.uk/nagware/NP.asp)

**Description**
The NAGWare f95 Compiler is a full Fortran 95 compiler for a range of Unix platforms. Derived from the NAGWare f90 Compiler, the world’s first Fortran 90 compiler, f95 provides robust, well tested, widely ported compiler capabilities for the Fortran 95 language. The f95 Compiler accepts fixed or free format Fortran 95 input and many common Fortran 77 extensions are allowed. HPF code is also compiled and checked though only single processor output is generated. New features of Fortran 2000 have been added, including the two J3 Technical Reports.

  - Allocatable Components
  - IEEE Floating Point Exception Handling

The latest update, Release 5.0, brings further innovations and refinements in the areas of error detection, improved performance and Fortran 2003 features.

5.4.6 The Portland Group Compilers

**Vendor**

**Description**
The Portland Group develops and sells high performance, production quality compilers and software development tools to the high performance parallel computing market.

**PGI Workstation** – PGI Workstation-class compilers and tools are for 32-bit x86 (AMD* AthlonXP*, Intel* Pentium* 4 and Xeon*), 64-bit AMD64 technology (AMD* Opteron*) and 64-bit IA32 EM64T (Intel Xeon-64) processor-based workstations or servers with up to 4 SMP CPU running Linux or 32-bit Windows operating systems.
**PGI Server** – PGI Server-class compilers and tools are for 32-bit x86 (AMD* AthlonXP*, Intel* Pentium* 4 and Xeon*), 64-bit AMD64 technology (AMD* Opteron*) and 64-bit IA32 EM64T (Intel Xeon-64) processor-based workstations or servers with up to 16 SMP CPU running the Linux operating system.

### 5.4.7 Salford FTN95

**Vendor**


**Description**

Salford FTN95 is a full Fortran 95 standards compliant compiler, capable of producing fast executables for Win32 and for Microsoft .NET. FTN95 ships with CHECKMATE (see Section 5.1.10) and a great range of supporting software. All standard and many vendor-specific legacy language features are supported, so that Fortran projects may be any combination of Fortran 77, Fortran 90 and Fortran 95.

Features include:

- Full Fortran 95 language compatibility
- Comprehensive integration with Microsoft Visual Studio .NET
- Standalone editor - Plato3
- Runtime checking with CHECKMATE
- Very fast compilation speed
- Excellent compile-time diagnostics
- Full source level debugging (Visual Studio .NET integrated debugger)
- Builds console, Windows and Microsoft .NET applications
- Microsoft Visual C++ Interoperability
- Salford C/C++ Interoperability
- GUI building with Salford Clearwin+

### 5.4.8 g95

[http://www.g95.org](http://www.g95.org)

**Description**

This is a free Fortran 95 compiler that has versions for the following platforms:

- Linux on x86, powerpc, 64-bit Opteron, 64-bit Itanium
- OSX on Power Mac G4
- FreeBSD on x86
- Cygwin
- HP-UX 11
- Solaris
- Windows (without Cygwin)

It is interesting to see that amongst the codes to have been successfully compiled with g95 are CASTEP, GALAHAD and MA41 (unsymmetric) and MA57 (symmetric) sparse direct solvers from the HSL2002 library.

5.4.9 GNU Fortran 95 (gFortran)

http://gcc.gnu.org/fortran/

Description

Another free Fortran-95 compiler under active development. A quote from the website

Gfortran is not complete, but it should already be usable for some user code. The implementations of COMMON and EQUIVALENCE are fairly new and thus not well tested, so give them a try. We provide details on the status of the run time library. Gfortran is regularly compiled on i386 GNU/Linux, PowerPC GNU/Linux, Cygwin and FreeBSD/i386, and can thus be built out of the box most of the time there. We are interested in your experiences on other platforms.

See the website for more details.

5.5 Unit Testing

In general unit testing of large scale scientific software is impracticable and what is required above all else is correct results in as little time as possible. May scientific applications use software libraries, e.g. for numerical linear algebra or solving ODEs. This speeds up development as a programmer is free to concentrate on novel aspects of the code rather than having to re-invent well established methods. Numerical libraries from established vendors, e.g. HSL and NAG bring with them extensive testing, again allowing a programmer to concentrate on testing other parts of the code if necessary.

There is little software available for unit testing Fortran code though we did find the following offerings.

5.5.1 Cleanscape Grayboxx

Vendor

Cleanscape Software International

http://www.cleanscape.net/products/grayboxx/index.html

Description

A complete software life-cycle testing toolset developed for software written in C, Fortran, Ada, and Assembly. Grayboxx provides a complete software testing solution that verifies functional and structural performance requirements for mission critical applications. Grayboxx automatically conducts the following test methodologies: Blackbox Testing, Whitebox Testing, Regression Testing, Assertion Testing, and Mutation Testing.

Grayboxx speeds the development process by allowing developers and test engineers to automatically:

- Generate test cases
- Conduct coverage analysis with complexity metrics
• Conduct unit performance testing with no probe insertions
• Generate test stubs
• Generate test harnesses
• Execute tests
• Prepare modules
• Verify results

Grayboxxx also allows for both full and partial regression testing, allowing the tester to run
the same test more than once or to name the test titles to run with a subset of test cases.

5.5.2 DejaGNU - The GNU Testing Framework
http://www.gnu.org/software/dejagnu

Description

DejaGnu is a framework for testing other programs. Its purpose is to provide a single front
end for all tests. Think of it as a custom library of Tcl procedures crafted to support writing a
test harness. A Test Harness is the testing infrastructure that is created to support a specific
program or tool. Each program can have multiple testsuites, all supported by a single test
harness. DejaGnu is written in Expect, which in turn uses Tcl – Tool command language.

DejaGnu offers several advantages for testing:

• The flexibility and consistency of the DejaGnu framework make it easy to write tests for
  any program, with either batch oriented, or interactive programs.

• DejaGnu provides a layer of abstraction which allows you to write tests that are portable
to any host or target where a program must be tested. For instance, a test for GDB can
run from any supported host system on any supported target system.

• All tests have the same output format. This makes it easy to integrate testing into other
  software development processes. DejaGnu's output is designed to be parsed by other
  filtering script and it is also human readable.

• Using Tcl and Expect, it’s easy to create wrappers for existing testsuites. By incorporating
  existing tests under DejaGnu, it’s easier to have a single set of report analyse programs..

Running tests requires two things: the testing framework and the testsuites themselves. Tests
are usually written in Expect using Tcl, but you can also use a Tcl script to run a testsuite that
is not based on Expect.

6 Integration and System Testing

We noted above that the testing of scientific applications is done at the system level (a) because
the focus is on results and speed and (b) because the software will in all probability use well
maintained libraries for the major computational work.

At the system level testing can be broken down into four parts which we will describe in the
remainder of this section.

Validation  Do the results tally with experiments or expert user expectations?
Debugging  What to do if the software doesn’t work or the results are not as expected.
Code coverage analysis Are all parts of the software exercised by test cases?

Performance analysis Where does the code spend its time, where should it be changed to speed things up?

6.1 Validation

Validation requires the development of test cases reflecting experiments for which quantitative results are available or reflecting scenarios where the qualitative response of the system can be tested. The development of such test cases must be left to an expert in the application field but once they are set up newer versions of the software can be run using the same test cases to ensure that the results are repeatable. This re-running of new versions can be automated and is known as regression testing. There are tools to help a developer here but the most likely way of implementing regression testing is via scripts written to run each test case and check the results against the ‘right’ answers.

Comparing numerical files can be a difficult and tedious job especially if the results differ only in the last column because of some change in algorithm or postprocessing. One tool that may help is ndiff (http://www.math.utah.edu/~beebe/software/ndiff/) that assumes you have two text files containing numerical values, and the two files are expected to be identical, or at least numerically similar. ndiff allows you to specify absolute and/or relative error tolerances for differences between numerical values in the two files, and then reports only the lines with values exceeding those tolerances. It also tells you by how much they differ.

6.2 Debugging

If the software fails to get to the end of a run or the results are not those expected then the first tool a developer could use is a debugger. By stepping through the code as it runs and examining the values of the variables bugs in the program may be spotted and corrected. There are many debuggers ranging from console tools to sophisticated graphical interfaces but all have the same purpose – to allow access to variables and their data in order to find mistakes in the program logic or model implementation.

The following is a selection of debuggers for various platforms covering the whole spectrum of sophistication. The Lahey and Salford Fortran compilers have built in debuggers but no information on them is available to add to this section.

6.2.1 The GNU Debugger (GDB)

http://www.gnu.org/software/gdb/gdb.html

Description

GDB, the GNU Project debugger, allows you to see what is going on ‘inside’ another program while it executes – or what another program was doing at the moment it crashed.

GDB can do four main kinds of things (plus other things in support of these) to help you catch bugs in the act:

- Start your program, specifying anything that might affect its behavior.
- Make your program stop on specified conditions.
- Examine what has happened, when your program has stopped.
- Change things in your program, so you can experiment with correcting the effects of one bug and go on to learn about another.
The program being debugged can be written in C, C++, Pascal, Objective-C (and many other languages [including Fortran]). Those programs might be executing on the same machine as GDB (native) or on another machine (remote). GDB can run on most popular UNIX and Microsoft Windows variants.

GDB is a console based debugger.

6.2.2 Data Display Debugger (DDD)
http://www.gnu.org/software/ddd/

Description
GNU DDD is a graphical front-end for command-line debuggers such as GDB, DBX, WDB, Ladebug, JDB, XDB, the Perl debugger, the bash debugger, or the Python debugger. Besides ‘usual’ front-end features such as viewing source texts, DDD has become famous through its interactive graphical data display, where data structures are displayed as graphs.

This is found on many Linux systems.

6.2.3 Insight
http://sources.redhat.com/insight/

Description
Insight is a graphical user interface to GDB, the GNU Debugger written in Tcl/Tk by people working at Red Hat, Inc. and Cygnus Solutions. It will run in GUI or console mode - it will even show a console window in the GUI.

6.2.4 KDbg
http://members.nextra.at/johsixt/kdbg.html

Description
KDbg is a graphical user interface to gdb, the GNU debugger. It provides an intuitive interface for setting breakpoints, inspecting variables, and stepping through code. KDbg requires KDE, the K Desktop Environment.

Features

• Inspection of variable values in a tree structure.

• Direct member: For certain compound data types the most important member values are displayed next to the variable name, so that it is not necessary to expand the subtree of that variable in order to see the member value. For example, you don’t need to go into a variable of type QString if you want to see the string that the variable holds. (BTW, this is of course not hardcoded, but can be extended to new types.) KDbg can also display Qt 3’s QString values, which are Unicode strings.

• Debugger at your finger tips: The basic debugger functions (step, next, run, finish, until, set/clear/enable/disable breakpoint) are bound to function keys F5 through F10. Quick and easy.

• Of course, lots of other basic functions: View source code, search text, set program arguments and environment variables, display arbitrary expressions. Everything you need to debug a program, eh!
- Debugging of core dumps, attaching to running processes is possible.
- Conditional breakpoints.

6.2.5 Absoft Fx2 Debugger

Vendor

Absoft Corporation http://www.absoft.com/Products/Debuggers/fx2/fx2_debugger.html

Description

Fx2 is a powerful, fully-featured source-level debugging solution. The standard Fx2 debugging solution is configured as a scalar (single process) debugger and is compatible with 32-bit, 64-bit Linux distributions, Windows and Macintosh OS X. Fx2’s intuitive graphical interface is designed to meet the requirements of novice and experienced programmers. Fx2 is included at no charge with all Absoft products for 32-bit Linux, 64-bit Linux, Windows and Macintosh OS X.

Features include

- Debugging tool for combined Fortran, C, C++ and assembler development, featuring:
  - Adjustable arrays
  - Complex variables
  - Command playback
  - Software watch-points, hardware breakpoints, etc.
  - Support for multi-threaded code (MPI)
  - Support for DWARF, DWARF2, stabs, and xstabs debug formats
  - Support for ELF, Mach-O, Coff, xCoff, PEF object file formats

- Supports AMD/IA32/Linux, AMD64/Linux, Windows, Macintosh OS X (includes support of Pro Fortran for OS X, XL Fortran and XL C/C++), other platforms coming soon

- Compatible with other compilers from Absoft, GNU, IBM, Intel, PathScale, and PGI

- Supports IBM XL Fortran on 64-bit Power/Linux systems

- Redirect Input/Output to/from any file

- Perform proper array indexing for Fortran and C/C++

- Handle Fortran and C quoted strings

A single user license costs $300.

6.2.6 Intel Debugger

Vendor

Description

The Intel Debugger is a component of the Intel compilers. It is a full-featured symbolic source code application debugger that helps programmers locate run-time programming errors (i.e., bugs) in their code. It provides extensive debugging support for the following languages: C, C++ and Fortran (including Fortran 90). It also provides a choice of command-line or graphical user interface (GUI). The command-line interface provides DBX-like (default) or GDB (GNU Debugger)-like (optional) command syntax. The GUI supports all Intel Debugger command-line functionality. The main window includes the basic debugging and convenience features, optional windows, various pop-up menus and dialog boxes, and a command-entry prompt. Intel Debugger works with the following compilers:

- Intel C/C++ and Fortran compilers
- GCC, G++ and G77 compilers
- Microsoft C/C++ compilers

Intel Debugger works on the following desktop and server platforms:

- IA-32 running Linux* and Windows*
- Systems using Intel Extended Memory 64 Technology, running Linux and Windows
- Intel Itanium 2-based systems running Linux and Windows

6.2.7 Portland Linux Cluster Debugger (PGDBG)

Vendor


Description

You can use PGDBG to debug MPI-parallel and OpenMP thread-parallel programs on your Linux cluster. It helps you debug programs on SMP Servers, distributed-memory clusters and even hybrid clusters where each node contains multiple SMP 32-bit or 64-bit processors.

PGDBG complements the powerful MPI and OpenMP parallel performance analysis profiler, PGPROF (see Section 6.4.5).

Some features of PGDBG:

Control Threads and Processes Separately – The PGDBG cluster debugger provides the ability to separately debug and control OpenMP threads, pthreads, and MPI processes on your Linux cluster. You can Step, Next, Break, Halt, Wait or Continue on threads or processes individually or collectively as a group.

Powerful GUI Windows – PGDBG provides a comprehensive set of GUI windows at your disposal to assist you in this difficult task. Using a single window, you now have precise control over each node of your cluster or each thread of your multiple-CPU SMP node. The Main GUI window displays source code for your F77, F95, C or C++ program and it has buttons for one-touch commands such as Run, Break, Quit, Print, or Stack traceback. Buttons on the Main window also allow you to select and control individual or collective threads and processes as well as the commands to access them. Input and output is displayed in the Main window or in a separate I/O Window. A Disassembly window shows you how the currently executing high-level source code has been compiled into assembly language. The assembly is interleaved with the source code as well. Assembly language Stepping is enabled as well as breakpoint indicators. Separate Active Process
and Active Thread summary windows can be spawned that give you a current status on each active thread or process. A Register window can be used to display register values in a variety of formats – integer, hex, floating point, etc. Custom Display windows can be used to follow values for important variables or memory regions in your executing program. A separate I/O Window can receive the stdout and stderr from your debug session.

Easy-to-use Features – PGDBG handles F77, F95, C and C++ programs and it is DBX-compatible with an extended command language for setting of breakpoints and watchpoints, and evaluation of expressions. Using PGDBG, you can control execution and examine the state of a program either symbolically using source code or at the assembly level. PGDBG allows you to switch contexts between threads in a parallel region, and step or examine the state of any executing thread. PGDBG is also interoperable with the GNU gcc and g77 compilers when the -g dwarf compilation option is used.

6.2.8 Allinea Distributed Debugging Tool (DDT)

Vendor

Allinea Software http://www.allinea.com/?page=48

Description

The Distributed Debugging Tool is a comprehensive graphical debugger for scalar, multi-threaded and large-scale parallel applications that are written in C, C++ and Fortran. DDT is available on IBM AIX and Linux on Power, HP-UX, Linux (most variants), SGI Altix, SGI IRIX and Sun Solaris. It is compatible with compilers from Absoft, IBM, Intel, Pathscale, PGI, Sun Microsystems and the GNU compiler suite. DDT supports all known MPI distributions.

DDT offers enhanced Fortran 95 capabilities, including support for Fortran 90/95 modules, derived data types, pointers, allocatable arrays and more. DDT now has automatic detection for source files, selecting between 32 and 64 bit versions of the debugger as appropriate.

6.2.9 Etnus TotalView

Vendor

Etnus LLC http://www.etnus.com/TotalView/index.html

Description

TotalView is a debugger aimed at the high performance and parallel computing markets. It supports threads, MPI, OpenMP, C/C++, and Fortran, plus mixed-language codes. Advanced features like on-demand memory leak detection, other heap allocation debugging features, a wide variety of breakpoints, the Message Queue Graph/Visualizer, powerful data analysis, and control at the thread level.

6.3 Code Coverage Analysis

Analysis of code coverage (or test coverage analysis) is a technique whereby test cases are created to exercise all parts of the software. By doing this a developer can find:

cold spots – Parts of the code that are never used, or just not used by the test cases.

hot spots – Parts of the code that are used frequently.

new test cases – To exercise a part of the code not already tested.
This kind of testing is really a test of the completeness of the test cases, i.e. do they exercise all parts of the code but it also gives indirect testing of the code itself. If all cases had the same cold spot(s) then maybe that code can be removed, or if there is a common hot spot then this is an area to study in detail to find ways of making it more efficient.

There are many measures that can be analysed in coverage analysis, among the most important are:

- Statement coverage, does each executable statement get executed?
- Decision coverage, are all clauses in if...then...else statements encountered?
- Condition coverage, are all logical expressions in if, do while etc. statements tested?
- Path coverage, have all possible paths through a function been tested?
- Function coverage, have all functions been invoked?
- Call coverage, have all calls to subroutines and functions been executed?

6.3.1 Unix command tcov

The command tcov gives line-by-line information on how a program executes. It produces a copy of the source file, annotated to show which lines are used and how often. It also summarizes information about basic blocks. tcov does not produce any time-based data.

Using tcov involves three basic steps:

1. Compiling the program to produce a tcov experiment – use the -a compiler flag
2. Running the experiment
3. Using tcov to create summaries of execution counts for each statement in the program

With some compilers the option -ax will provide extra information that can be analysed by tcov.

6.3.2 Polyhedron plusFort – CVRANAL

Vendor

Polyhedron Software http://www.polyhedron.co.uk

Description

The plusFort package includes CVRANAL, a coverage analysis facility that places probes into Fortran source code allowing users to monitor the effectiveness of testing. At the end of each run, the probes update the coverage statistics for each source file. This data may be analysed at any time using the CVRANAL tool. CVRANAL identifies untested code blocks, and execution hot-spots.

In addition, CVRANAL can annotate source code with the coverage statistics for each executable line. The annotations are comments and do not affect the validity of the source code.

6.3.3 FACT

http://www.d1.ac.uk/TCSC/UKHEC/FCAT/
Description

FACT is similar to CRVANAL in that it reports the execution count for each line of executable source code.

FCAT offers some facility for the coverage analysis of parallel codes. It treats a line as being executed if at least one processor has executed it. The counter for the line is taken as the maximum of the number of times this line has been executed over all processors. For example, if the code has been run on two processors and a line was executed 5 times on processor one and 0 times on processor two, then the counter for this line is 5.

WARNING: because the output of the instrumented code of all processors is written to the same standard output, on some parallel computers an output string may be half-written, then followed by another string. This is can cause problem to FCAT. In such event you may have to run the code again (no need to recompile) and hopefully the problem will not repeat itself.

6.3.4 Cleanscape Grayboxx

Vendor

Cleanscape Software International
http://www.cleanscape.net/products/grayboxx/index.html

Description

A complete software life-cycle testing toolset that includes coverage analysis with complexity metrics. It can perform the following coverage functions

- Measure test effectiveness and reliability of testing by analyzing application source code
- Set up test cases and measures their efficiency
- Consolidate results of test coverage measurements for several scenarios or during a test campaign
- Enable effective visualization of covered and uncovered source code

6.4 Performance Analysis

Another branch of software testing is performance analysis, sometimes called profiling. Performance analysis is looking at program execution to pinpoint where bottlenecks or other performance problems such as memory leaks might occur. Once you know where potential trouble spots are, you can change your code to remove or reduce their impact. As with debuggers there is a range of tools to do this from Linux/Unix standard command line tools such as prof and gprof to sophisticated GUI tools such as VTune from Intel.

6.4.1 Unix commands prof and gprof

The commands prof and gprof work in similar ways. They require the program to be compiled with a certain flag (-p or -pg) that causes the compiler to insert a call to the mcount subroutine into the object code generated for each recompiled function of the program. During program execution, each time a parent calls a child function the child calls the mcount subroutine to increment a distinct counter for that parent-child pair. When the program is run it produces a call graph profile file (mon.out or gmon.out) that is interpreted by prof or gprof.

6.4.2 Intel VTune

Vendor

Intel http://www.intel.com/software/products/vtune/vpa/
Description

Provides performance analysis on Linux and Windows systems with the following features.

- Use sampling to gain an accurate representation of your software’s actual performance, with negligible overhead. Gather CPU snapshots to identify problems such as cache misses. No special builds or instrumentation are required.

- View results of time and event sampling on multiple levels with Hotspot Analysis, drilling down to the exact operating system process, thread, module executable, function/method, individual line of source code, or individual machine/assembly language instruction to identify specific bottlenecks.

- Produce a picture of program flow to quickly identify critical functions and call sequences using Call Graph Profiling. Gain a high-level, algorithmic view of program execution.

- Track system activity and resource consumption during runtime with the Counter Monitor to help identify system level performance issues.

- Collect performance data on a remote Linux system while maintaining the Windows GUI ease-of-use for data analysis and interpretation.

- Gather performance data for Intel PXA2xx processors using a very lightweight and low-overhead remote data collector. Use the host system to easily control (configure, start, and stop) the target development system.

- Tune code more efficiently using Intel Tuning Assistant to suggest code improvements based on an extensive knowledge base.

There is a Windows GUI for Windows and the Linux version can be run through Eclipse (http://www.eclipse.org/).

6.4.3 Intel Cluster Tools

Vendor


Description

Two related tools Intel Trace Analyser and Intel Trace Collector are available as part of the suite of Intel cluster tools. They replaced Vampire and Vampirtrace from Pallas GmbH (www.pallas.com) when Intel took over the Pallas HPC team.

With Intel Trace Collector you can:

- Automatically instrument MPI and routine entry/exit points. The traditional application programming interface (API) is also available to instrument application source code.

- Trace concurrent behavior of parallel applications and calculate statistics on demand for specific time intervals and processes.

- Reduce runtime overhead through scalability and memory handling. Trace data is cached in memory.

- Monitor the application runtime with the Intel Trace Collector or use it in a purely statistical mode with counters. It calculates statistics regarding function calls, messages, and collective operations at runtime and stores them without event data.
• Capture application statistics displays with trace data.

• Be assured of completely thread-safe tracing of multi-threaded MPI applications, event-based tracing to non-MPI applications supporting Java processes, and multi-threaded processes without MPI.

• Automatically monitor function entry/exit points through binary instrumentation of the executables. This leads to a detailed program analysis of your code and application runtime. In addition, with instrumentation capabilities you can trace C++ and Java class hierarchies.

• Use the extended APIs to trace other programming models and libraries.

• Record event trace data in highly-scalable Structured Trace Format (STF).

Intel Trace Analyzer offers:

• Timeline Views and Parallelism Display: Visualize the concurrent behavior of parallel applications and calculate statistics on demand for specific time intervals and processes

• Communication Statistics: View communication metrics for an arbitrary time interval and message-length distribution

• Execution Statistics: Study subroutine execution metrics for an arbitrary time interval and call-tree comparisons between different program runs

• Variant Displays: View by global or per-process variants. Restrict statistics to arbitrary parts of the trace, and compare multiple traces to assess optimizations. A fast GUI allows you to control displays and statistics on large data sets using object point-and-zoom for enhanced detail, context-sensitive sub-menus, and coupled displays with automatic updates of statistics recomputation.

6.4.4 Allinea Optimization and Profiling Tool (OPT)

Vendor

Allinea Software http://www.allinea.com/?page=49

Description

The Optimization and Profiling Tool (OPT) is a companion product to DDT (see Section 6.2.8), and provides powerful profiling capabilities for both scalar and large-scale parallel applications. Features include

• Grid-enabled with a client/server architecture that works transparently on local or remote compute resources

• Cross-platform, providing the same easy-to-use interface whatever O/S and processor type

• Highlights and analyzes the bottlenecks in software

• Timeline display with filtering and stack information

• Displays history of MPI calls with information on timing, message sizes and other key parameters

• Access to CPU and PAPI performance statistics, with cache profiling
• In depth call-graph analysis highlights communication and computation imbalance as well as waiting processes

• Interoperable with the Distributed Debugging Tool (DDT)

• Supports all major MPIs, OpenMP and scalar programming

• C, C++, Fortran and Fortran 95 support

• Compatible with compilers from all major vendors including Absoft, Intel, Pathscale, the Portland Group - and the GNU compiler suite

6.4.5 Portland Linux Cluster Profiler (PGPROF)

Vendor

Description
PGPROF is an interactive and powerful, postmortem statistical analysis tool. You can use PGPROF to analyze MPI-parallel and OpenMP thread-parallel programs on your Linux cluster. It helps you analyze programs on SMP Servers, distributed-memory clusters and even hybrid clusters where each node contains multiple SMP 32-bit or 64-bit processors. You can profile on the function level and on the source code level for F77, F95, HPF, C and C++.

PGPROF provides a comprehensive set of GUI windows to assist you when analyzing a cluster application. With PGPROF you can quickly determine where execution time is spent and see which functions were called and how often. The PGPROF profiler supports function-level and line-level analysis. You can even effectively profile optimize code at the block level using PGI’s unique instrumentation or a sample-based/gprof style methodology.

The PGPROF GUI displays information in intuitive easy-to-use formats such as bar-charts, percentages, counts or seconds. New capabilities allow you to quickly analyze MPI Sends, MPI Receives and other MPI communication. Information on where time is being spent in thread-parallel regions is also available.

6.4.6 Multi-processing Environment (MPE)
http://www-unix.mcs.anl.gov/perfvis/download/index.htm#MPE

Description
The MPE (Multi-Processing Environment) library exploits the features of MPI to provide a number of useful facilities, including performance and correctness debugging, graphics, and some common utility routines. The MPE library was developed for the MPICH implementation of MPI (and is included with the MPICH distribution), but can and has been used with any MPI implementation.

Currently the main components of the MPE are

• A set of routines for creating logfiles for examination by the graphical visualization tool Jumpshot-4.

• A shared-display parallel X graphics library.

• Routines for sequentializing a section of code being executed in parallel.

• Debugger setup routines.
MPE is easier to use with MPICH than with other implementations of MPI, particularly since getting profiling results with the minimum of work is straightforward. It is simply adding a flag to the compile and link commands (e.g. `mpif77`, `mpif90`) as follows:

- `mpitrace` to compile and link with tracing library.
- `mpianim` to compile and link with animation libraries.
- `mpilog` to compile and link with logging libraries.

For other MPI implementations, the user needs to compile and link their application with MPE profiling libraries explicitly as shown in an example makefile.

### 6.5 Source Code Documentation

Tools like `javadoc` for Java that take specifically formatted comments in Java source code and create HTML files that help to document the code are available for Fortran. We will describe some of these tools in this section.

#### 6.5.1 Understand for Fortran

**Vendor**


**Description**

Understand for FORTRAN is an interactive development environment tool providing reverse engineering, automatic documentation, metrics and cross referencing of FORTRAN source code. It supports FORTRAN 77 (F77) and FORTRAN 90 (F9X) language standards, with common VAX, Cray and Salford extensions.

Understand for FORTRAN helps you reverse engineer, understand and maintain large amounts of legacy FORTRAN source code. The tool includes a syntax colorizing ‘smart’ editor that can tell you about what you are editing. It also offers in-depth graphical reverse engineering, fast code navigation, detailed cross referencing, interactive source browsing, automatic documentation via HTML output reports, various complexity and volume metrics, as well as easy integration with outside editors. A PERL API and C/C++ API are included so that you can write scripts and programs generating automatic documentation directly from your source code.

The current (February 2005) cost is $695 for a single license, $595 each for a 5 license pack, and $395 each for a 10 license pack.

#### 6.5.2 f90tohtml

[http://mensch.org/f90tohtml/](http://mensch.org/f90tohtml/)

**Description**

`f90tohtml` is a PERL script that converts FORTRAN source code into HTML. The HTML contains syntax highlighted source code within which all subprogram calls are linked, both forward and backwards. A clickable calling tree is constructed. A subject index can be made from a user-supplied hash. A search engine, based on regular expressions, searches the code.

There are 2 files that must be updated to make `f90tohtml` work with your source code, one is a script that prepares the way by making a list of all the source files and the other is an input file to `f90tohtml` itself that tells it where the source file lists are and where to put the HTML files.
6.5.3 f90doc

http://theory.lcs.mit.edu/~edemaine/f90doc/

Description

f90doc is an easy-to-learn documentation tool for Fortran 90. It generates pages of HTML documentation from Fortran 90 source. The documentation is derived from comments prefixed with !! written (if desired) in a simple and intuitive documentation language whose source is easily readable in the code itself, while formatting nicely in HTML. f90doc is similar to javadoc, but does not require comments to be written in HTML.

A list of main features for the latest version:

- A useful (but simple) documentation language that is easily readable with a text editor and convertable into HTML, supporting lists, bold, italic, and verbatim text.
- Module, subroutine, function, type, interface, and variable declarations, each of which can be commented.
- A table of contents for each top-level block (module, program, subroutine, or function), with links to the full description of each portion.
- Use statements in modules, which generate links.
- Call statements in programs, subroutines, and functions, which generate a list and optionally links.
- Module procedure statements in interfaces, which generate links.
- Detection of public/privateness of module objects.

A couple of limitations: only free-form input files are supported, and only objects in modules are documented. This latter means f90doc will be most useful to those who are re-engineering their existing code into a more object oriented form and can add comments as they go.

It is still useful on existing source as it does give the subroutine name, the variables and the called routines.

6.5.4 ROBOdoc

http://www.xs4all.nl/~rfsber/Robo/robodoc.html

Description

ROBOdoc allows specially formatted documentation headers to be extracted from source files, reformatted in HTML, XML DocBook, TROFF, ASCII, LaTeX, PDF, or RTF, and stored in a different file. This way ROBODoc allows you to include your documentation in the source code. You can edit the source code as well as the documentation in a single file with your favourite editor. This makes it easier to keep your documentation in sync with your source code.

ROBODoc can be used to document functions, methods, classes, variables, makefile entries, system tests, and anything else you can think of.

ROBODoc works with C, C++, Fortran, Perl, shell scripts, Assembler, DCL, DB/C, Tcl/Tk, Forth, Lisp, COBOL, Occam, Basic, HTML, and any other language that supports remarks.

Without the specially formatted headers this tool produces no useful output. However it comes with a companion tool robohdrs. This helper program can take clean source file and insert ROBODoc headers to functions, global variables, and macros. There are issues with this tool but it saves lots of cumbersome typing when starting on documenting an existing code-base with ROBODoc.
References


