CLRC
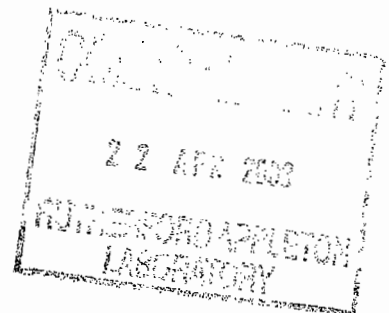
Software Tools for HEC Applications

R F Fowler   C Greenough

16th April 2003

COUNCIL FOR THE CENTRAL LABORATORY OF THE RESEARCH COUNCILS

Software Tools for HEC Applications

R.F. Fowler and C. Greenough

Computational Science and Engineering Department
CLRC Rutherford Appleton Laboratory,
Chilton, Didcot OX11 0QX, UK

Email: r.f.fowler@rl.ac.uk or c.greenough@rl.ac.uk

This report is available from http://www.cse.clrc.ac.uk/Activity/HPCI

March 4, 2003

Abstract

This report provides an overview of some of the currently available software tools for use in High End Computing Applications (HEC). In particular we look at tools that help improve software quality, tools for program development and debugging and tools for software version control. These are vary broad areas and even in the limited context of HEC requirements we can only hope to give a brief summary of some of the main options.

This review mainly concentrates on tools that are appropriate for languages such as Fortran, C, C++ and Java which are of major interest to the HEC community. This report builds on previous HEC reports on Software quality for Fortran 90 and Software engineering and code development.

Keywords: software quality, Software tools, Fortran 90, Fortran 95, High Performance Fortran C, C++, Java.

Contents

1 Introduction

This report gives a survey of some of the tools that may be useful in developing software to exploit High End Computing (HEC) resources. It extends and updates information in a previous report on *Software Quality Assurance for Fortran 90[1] - A Survey of Available Tools* [1], and so some extent the information in *Software Engineering and Code Development for HPC Applications* [2]. As the title indicates, the previous report focused on tools for use with Fortran 90, and in particular tools related to Quality Assurance (QA). While this language is still widely used in scientific and engineering research, others, such as C, C++ and Java, are also becoming increasingly popular. In addition there are many other tools and environments that are useful to the HEC software developer, even though they are not directly related to quality assurance. These include software for version control and the increasing trend towards integrated development environments (IDEs) which are becoming more common.

As a simple illustration of the increasing importance of C and C++ in some areas of numerical computing, one can look at the algorithms published in ACM-TOMS[4] over the last few years. While Fortran is still just in the majority, C and C++ account for a third of the codes in this

Language	Number of Algorithms
Fortran (77 and 90)	11
C++	4
C	3
Matlab	2
Awk	1

Table 1: Number of algorithms published on ACM-TOMS web site that make use of a given language. Some algorithms use more than one language. Data is for algorithms published from January 2000 to November 2002.

small sample. Within the Fortran coded algorithms only 2 make use of the current Fortran 90/95 standard, while 9 are written in Fortran 77. This may be in part due to the time taken to get a new algorithm accepted for publication, though it is a little disappointing.

Another example of the increasing importance of C++ can be seen in a report on Numerical Libraries[12]. In this survey of numerical software, Fortran interfaces exist for almost 75% of the packages discussed, C interfaces for 53% and C++ interfaces for 22%. While C++ can probably be made to call Fortran packages, via C like calls, it is unlikely that Fortran code could easily access libraries written in C++.

A lot of work is also been carried out to develop practical numerical methods and libraries in Java. There were problems in doing this originally since neither the Java standard nor the interpreted nature of the Java virtual machine were suited to high performance numerical work. Recent developments[11] have shown that some Java compilers can now achieve up to 80% of the performance of Fortran code.

In addition, Java programs can of course make calls to C, C++ and Fortran libraries. Hence it is quite possible to use Java to develop a portable graphical interface to software , while the high performance sections of the code are written in one of the more traditional languages.

Clearly there is a need to provide software tools that will help in the development of accurate and efficient software for HEC scientific and engineering requirements. In this report we survey some of the tools that are currently on the market. As this is a very broad area, particular with the number of products available for C and C++ development, we cannot hope to provide an exhaustive list. Instead we aim to give details of tools in three main areas which are:

- QA and source code analysis tools.

- Version control tools

- Software development environments.

The first of these areas is mainly the subject of the previous report, though we now include some tools related to C and C++. Software version control, using tools such as RCS and CVS is highly important in any environment where more than one programmer is developing a given program. Integrated development environments are becoming more common in both the Unix and the Windows worlds. Where in the past a vendor would typically provide just a compiler and a command line debugger, it is possible now to get a whole suite of tools to analyse the structure and performance of a code.

2 Software development tools for Quality Assurance

2.1 Software Quality

Software quality is not a term that has a precise, "scientific" definition. Nevertheless it is vital that software developed for research into scientific and engineering problems should strive to be of high quality for several reasons:

- If the quality of the software used to produce a new computational result is questionable, then the result must also be treated with caution. Just as results from a poorly designed experiment are of little use, computational results need to be based on sound theory coupled with quality software;

- A research code may be developed by several different students or staff at different times. Poor application of software quality control techniques can lead to programming errors, wasted development time and dubious results;

- High quality software can be made available to other research groups, or included in commercial products.

Production of high quality software requires systematic design and development coupled with comprehensive testing. Clearly this can be at odds with scientific investigations which often require rapid results to prove the usefulness, or otherwise, of a new method. However it is vital to strike the right balance between the need for rapid development of a prototype code and the requirement of trustworthy results and re-usable code. Improving the quality of poorly written code can be a very time consuming process, and it may prove easier to start again from scratch. It is much more efficient if software quality control is applied from the beginning, rather than trying to retro fit it to existing code.

The need for quality assurance is underlined by research such as that due to Les Hatton [3]. Hatton has analysed a large number of commercial scientific software packages written in C and Fortran 77. The Fortran 77 codes had an average of 12 statically detectable faults per 1000 executable statements (these were found using the QA Fortran static analysis program). In run time tests of nine software packages for seismic data analysis, which nominally implement exactly the same mathematical algorithms, substantial variations in the results were found. These differences are sufficient to cast doubt on some new analysis methods which require high accuracy in the processed data.

Another investigation of software quality in Fortran codes has been made by Hopkins [9]. He has investigated how the quality metrics of some well known public domain packages and published algorithms have changed with time. The most worrying aspect of this work is that there seems to be little improvement in the quality measures of many published Fortran algorithms over the past twenty years.

Software tools can be useful is the assessment of a software product in several ways:

- Detection of software defects that are not usually reported by compilers, such as argument mismatch between program units and use of unassigned variables;

- Objective measures of programming style and complexity, e.g. a program unit has an excessive number of source code lines or has so many independent paths through it to making testing difficult;

- Instrumentation of the code to identify parts that have not been exercised by a test set;

- Restructuring to improve readability and maintainability;

- Extraction of software design and data flow from the source code to aid understanding and re-documentation.

To be really effective they should be used frequently through the development process so that errors and deviations from good programming practise can be identified early on. Application to existing software can be helpful in identifying problems and hidden faults. However, they cannot be expected to transform poorly-designed software into good software. Careful design and testing are still vital in the development process.

In another study by Hopkins [10], various Fortran QA tools and high quality debugging compilers have been used to examine the existing Fortran codes in the ACM-TOMS CALGO collection of algorithms [4]. This was an exercise to bring the older algorithms in the library up to date, but it did reveal a number of cases where common faults, such as array bound errors, use of unassigned variables and similar mistakes existed in the codes. These could give erroneous results on some machines. Hopkins also looked at ways that the test data sets could be arranged to ensure that almost all the basic blocks were executed at least once, through the use of coverage analysis. While use of QA software and maximisation of test coverage do not guarantee the correctness of a code, it does improve the level of confidence in it.

A useful guide to coding and documenting Fortran 90 software has been placed on the Web by the UK Meteorological Office, at:
    `http://www.meto.gov.uk/research/nwp/numerical/fortran90/f90\_standards.html`
While not all the rules and suggestions in this document will be applicable to every software developer, it could be useful as the basis of a local coding standard.

Another useful website for Fortran related software is the Fortran Market Place:
    `http://www.fortran.com/fortran`
This has many links to information on Fortran 90 compilers and some public domain software such as Ftnchek (error checking for Fortran 77) and convert.f90 (converts fixed format to Fortran 90 free format).

2.2 Software Tool Capabilities

In this section we outline some of the capabilities that are available in software tools that can be used for quality assurance purposes.

2.3   Static Error Detection

Static errors may be detected by programs which provide *lint*-like checking for Fortran 77. Possible tests include:

- Argument checking - are the number and type of arguments consistent between the declaration of a subprogram and each call;

- Common block checking - warnings can be generated for named common blocks of different lengths, and related potential problems;

- Use of implicit typing - though standard conforming, implicit typing often leads to errors;

- Unused items - variables and sections of code that are never used are reported as these are at least a source of unnecessary clutter and may indicate programming errors;

- Implicit type truncations, such as copying of real to integer, may lead to unwanted truncation.

In Fortran 90 all the above programming errors can be made since Fortran 77 is a subset of the language. However careful use of new features, such as explicit interfaces to all subprogram interfaces and replacing common blocks with modules, should enable the compiler to detect some of the above problems. On the down side Fortran 90 does allow new errors to occur such as those associated with memory management. Static analysis should be able to detect some of these problems, for example allocatable and pointer variables that are used before allocation. Other errors, like references through a pointer to data that has been deallocated (a "dangling" pointer), may only be caught by detailed run time checks.

A system such as ftnlint, provided with SGI and Cray Fortran 90 compilers, is able to generate warnings for many of the common static errors listed for Fortran 77 style codes. It can also make global checks on module variables and parameters, argument INTENT statements, subroutine calls where explicit interfaces are required, and related problems.

2.4   Dynamic Error Detection

Many software errors can only be detected at run time, as was mentioned in the previous section. Some software tools have been developed which will instrument the source code to provide extra run time checks. Array bound errors and variables used before set could be detected in this way. However, these checks can often be more conveniently built into the compiler and/or its run time support system. The majority of Fortran 90 compilers now include array bound checking as an option. Some also have options to detect variables used before set and memory leaks. In some cases this additional support requires the use of an interactive debugging tool, which runs the executable program.

2.5   Code Restructuring and Transformation

A lot of old Fortran software is very poorly structured due to it having been developed originally as Fortran 66. A number of tools exist to transform and simplify such spaghetti code to Fortran 77 style code. Some tools are now becoming available to further transform Fortran 66 and 77 to take advantage of the new Fortran 90 syntax. Possible transformations include:

- Source code transformation from fixed format to the new free format;

- Reformatting and "pretty printing";

- Re-structuring to remove spaghetti code and use new constructs such SELECT, EXIT, CYCLE and array notation;

- Transformation of common blocks into modules, possibly with USE and ONLY statements to minimise potential name clashes;

- Generation of interface blocks for subroutines to allow better compile-time argument checking;

- Addition of appropriate INTENT statements to subroutine arguments.

An example of how one particular tool, Vast90, can translate code is shown below. The original Fortran 77 code is listed first, followed by the Fortran 90. Note that this example was produced using an old version of Vast90 with the default options, so it may not show the full capabilities of the current version, in which the transformation tool is now known as Vast77to90.

```
C f77
      SUBROUTINE MATMUL(A,A1,A2,L,N,M,NB)
C.........................................
      INTEGER NB,L,M,N
      DOUBLE PRECISION A(NB,NB),A1(NB,NB),A2(NB,NB)
C     Local
      INTEGER I,J,K
C                                 Aik = A1ij . A2jk
      DO 10 I=1,L
        DO 20 K=1,N
          A(I,K)=0.0
          DO 30 J=1,M
            A(I,K)=A(I,K)+A1(I,J)*A2(J,K)
30        CONTINUE
20      CONTINUE
10    CONTINUE
      RETURN
      END
```

```
! f90
      SUBROUTINE MATMUL(A, A1, A2, L, N, M, NB)
!...Translated by Pacific-Sierra Research VAST-90 1.02X
      IMPLICIT NONE
!-----------------------------------------------
!   D u m m y   A r g u m e n t s
!-----------------------------------------------
      INTEGER L, N, M, NB
      DOUBLE PRECISION, DIMENSION(NB,NB) :: A, A1, A2
!-----------------------------------------------
!   L o c a l   V a r i a b l e s
!-----------------------------------------------
      INTEGER :: I, J, K
!-----------------------------------------------
!                                 Aik = A1ij . A2jk
      DO I = 1, L
        DO K = 1, N
          A(I,K) = 0.0
          A(I,K) = A(I,K) + SUM(A1(I,:M)*A2(:M,K))
        END DO
      END DO
```

```
        RETURN
        END SUBROUTINE MATMUL
```

In this case the array notation has been used to remove the inner loop, and the outer two loops left in place. In this particular case the whole subroutine could better be replaced with the intrinsic routine MATMUL.

2.6   Software Structure Analysis

To aid understanding and maintenance of software it is useful to be able to extract call graphs and other diagrams of the structure. Variable cross referencing tables are also a useful tool for software maintenance.

A typical cross reference map produced by the Sun Fortran 90 compiler is shown below for one subroutine.

```
Cross-reference map:   13 Identifiers (74 References)
```

Type Kind	Usage	EqDSv	Offset	Home	Name	References							
Char	1-D Arr		1	Arg	ARGNAM	2	8	17-	29=				
Int	4 Scalar	-	0	Stack	IND1	14	20=	21+	22	23=	24	25	26
Int	4 Scalar	-	1	Stack	IND2	14	24=	25+	26=	29	30	32+	33
Int	4			Intrin	INDEX()	20	22	24	25				
Int	4 1-D Arr		2	Arg	LENARG	2	10	17=	30=	36=			
Int	4 Scalar		4	Arg	LENST	2	12	20	22	24	25		
Int	4			Intrin	LEN_TRIM()	17							
Int	4			Intrin	MIN()	36							
Int	4 Scalar	-	2	Stack	NEWIND	14	22=	23+					
Int	4 Scalar		5	Arg	NOARG	2	11	17	28=	29	30	36+	
Char	Scalar		3	Arg	STAMNT	2	9	20-	21	22-	24-	25-	27

Such listings can be used to understand where variables are modified and used and can also form part of the supporting documentation of the code. Similarly call graphs help to understand and document the control flow.

2.7   Software Metrics

Some software analysis tools go beyond checking the correctness and overall structure of a program and look at the complexity of the code within subprograms. Many different metrics have been defined which can be used to measure the quality of programs. These include things such as the static path count, the number of ways through a given subprogram (related to the Npath metric [5]) and the cyclomatic complexity metric due to McCabe [6]. Based on programming experience, there are recommended maximum values for these measures, which indicate when a subprogram is poorly structured or is too convoluted to understand or test.

Using such metrics during the course of software development is important in highlighting excessively complex subprograms as they are being developed. The programmer can then restructure the code, perhaps splitting it into more modules. Application of a metrics analysis to existing software can also identify poorly designed routines, but correcting them is then more difficult. Some examples of how quality measures can identify excessively complex routines in Fortran software are given in [9].

3 Software Analysis Tools for Fortran

This section lists a number of software tools, most of which are commercial, which can be used to detect faults, report on standard conformance, measure software complexity and/or restructure Fortran software. Some of these are actually only applicable to Fortran 77 software. 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.

3.1 Ftnchek

Author

Dr. Robert Moniot,
Fordham University,
New York, NY 10023 USA
Website: `http://www.dsm.fordham.edu/~ftnchek/`

Description

Ftnchek is a freely available tool that provides many checks for Fortran 77 software. It also supports a few Fortran 90 extensions at present, but still lacks too many features (for example the USE statement is not recognised) to be of much use with new code. It is however an extremely useful program to check Fortran 77 codes. Its static testing features include:

- Standard conformance test, to Fortran 77 standard.

- Subroutine arguments checking.

- Identification of unused variables, used before set values and type mismatch cases.

- Provide a call tree of the program.

This package is freely available and development is being continued with the goal of eventually providing full support for Fortran 90/95.

3.2 Floppy and Flow

Author

Julian J. Bunn. Website: `http://www.netlib.org/floppy/`

Description

Floppy and Flow are two related tools for analysis of Fortran 77. Features include:

FLOPPY is a software tool that takes as input a file of FORTRAN 77 code and checks it according to various "coding conventions". Floppy can "tidy" the source FORTRAN, producing a new file with indented DO-loops, block IF-s, and so on. Floppy can be used to generate HTML from the Fortran program. In this case, a new file is written where each module and include file name in the source Fortran is replaced by an HTML Anchor. The format of the source is preserved when the document is browsed by an HTML browser. FLOPPY can also be used to write a binary summary file which is then used as input to the FLOW program. The FLOW program takes the binary summary file produced by FLOPPY, and can produce various reports on the structure of the original FORTRAN program.

The source of these two tools is available on netlib. It seems that these programs are not under active development, so a Fortran 90 version seems unlikely.

3.3 Polyhedron plusFORT

Vendor

Polyhedron Software
Linden House
93 High Street
Standlake
Witney, OX29 7RH, UK
Website: `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 6.50 are:

plusFORT SPAG. As well as F66 to F77 conversions, this tool is able to convert F77 code to use F90/95 constructs. Fortran 90 modules are now understood and common blocks can be converted to these if required. It does still have a few limitations, for example it does not recognise overloaded operators and procedures. Also, converting a code that used common blocks to modules required some hand editing of block data initialisations to make it work.

SPAG can also remove dead code and unused variables as well as code "polishing" and standardisation of declarations. The intent of all arguments can also be added to the reformatted code.

Global symbol tables can be produced for further analysis, using GXCHK (see next section). Complexity metrics can be determined for each subprogram. The measures reported are: (1) logical content (number of decision points, comparable to McCabe), (2) all control jumps, where control is transfered to another point, and (3) unstructured jumps. The restructuring of SPAG can often reduce the last two measures. The metrics can be calculated for both Fortran 77 and Fortran 90 code.

plusFORT Static Analysis. A full global static analysis can be performed using GXCHK. This uses the symbol tables which are generated by SPAG. It provides checking for variables used before set, argument mismatch, common block problems, etc. It can also generate

interface specifications for each subprogram. Call trees and charts showing where variables are used may be produced.

plusFORT Dynamic Analysis Dynamic analysis is made possible by using SPAG to add run time checking statements to the code. Used before set errors, including tests on array elements, are checked in the modified code. The instrumented code is run in the normal way and produces a log file of any faults.

plusFORT Coverage Analysis. The program CVRANAL can be used to instrument Fortran software for coverage analysis. A report is produced as to how often each block has been executed in the test runs. A cumulative count of the times each block is executed can be made over several runs. CPU time profiling can also be performed.

plusFORT Automake. This automatically checks for dependencies via INCLUDE and USE statements to rebuild an executable file and can be used in a similar way to the Unix command "make", with a configuration file. It is Fortran 90 aware in that it will determine the necessary compilation order to make sure that required modules have been built before the routines that use them.

Cost

The plusFORT set of tools costs between £695 and £5995, depending on platform and number of users. A 40% educational or non-commercial discount is available. A restricted version, which is mainly limited to Fortran 77 code analysis is available free of charge, for non-commercial use. This version only runs on Linux systems.

Fortran 90 Features

Most Fortran 90/95 features are supported by SPAG, with the exception of declarations relating to overloaded operators and procedures.

3.4   Leiden University FORCHECK

Vendor

Forcheck Group
Leiden University
P.O. Box 9604
2300 RC Leiden
Netherlands
Website: `http://www.forcheck.nl/forchek`

Description

FORCHECK supports Fortran 77, Fortran 90/95 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;

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 NT/95. A graphical user interface is available on some systems and a command line interface on all systems. An evaluation license can be obtained to try out the software before purchase.

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. A 40% academic discount is available.

3.5   Cleanscape FortranLint

Vendor

Cleanscape Software International,
445 Sherman Ave, Ste. Q
Website: `http://www.cleanscape.net/products/fortranlint/index.html`

Description

Details of Cleanscape FortranLint can be obtained from their 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.

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 to Fortran 77 and 90. The tool can also check OpenMP constructs for various errors.

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

3.6 NAG Fortran Tools

Vendor

NAG Ltd,
Wilkinson House,
Jordan Hill Road,
Oxford, OX2 8DR, UK.
Website: `http://www.nag.co.uk/nagware/NQ.asp`

Description

In the past NAG has provided software tools for analysis and transformation of Fortran 77 software. They now offer a similar set of tools for Fortran 90. These can be run through either a GUI, which allows easy selection of the source files and operations to perform on them, or a command line interface.

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 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 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 90 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.

Code metrics – this tool is currently only available for Fortran 77. It reports a comprehensive set of code metrics for each source file.

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. A tool to convert common blocks to modules is provided, along with restructuring and cross reference tools for Fortran 90. The module tool correctly deals with common data defined in block data statements.

These tools provide a path for transforming old Fortran software to Fortran 90/95 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. The addition of restructuring and common conversion tools should enhance the usefulness of this product.

3.7   Cobalt Blue - FOR_STRUCT, FOR_STUDY

Vendor

Cobalt Blue Inc,
11585 Jones Bridge Road
Suite 420-306
Alpharetta,
Georgia 30005, USA
Website: `http://www.cobalt-blue.com`

Description

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

FOR_STRUCT. This product is mainly described in terms of it 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 output, 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 does not extend to many import new features such as modules. This makes it of limited use in new Fortran 90 development work.

FOR_C This tool converts Fortran 77 code into structured ANSI C.

3.8 Veridian VAST and DEEP

Vendor

Veridian,
1200 South Hayes Street, Suite 1000,
Arlington VA22202, USA.
Website: `http://www.veridian.com`

VAST tools

Veridian sell several software tools under the title VAST. These include translators such as VAST/77to90 and VAST/77toHPF, compiler systems such as VAST/f90 and VAST-HPF and the code optimiser and parallelisation systems VAST/Parallel, VAST/Superscalar, etc. While much of the emphasis of these products is on extraction of the maximum performance of existing serial codes on parallel and vector architectures, some can also be useful from the point of view of software quality. The translation tools like VAST/77to90 allow old software to be transformed into the newer format with enhanced readability and maintainability. The optimisation tools could also be useful in transforming existing serial codes of known quality into higher performance versions without the usual level of risks associated with extensive "hand coded" optimisation.

VAST/77to90 Capabilities

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.

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.

Using VAST/77to90

An older version of this software tool (V1.02X) was available to the authors on an IBM AIX system. It has been successfully used to translate a semiconductor device modelling program from Fortran 77 to Fortran 90. The conversion process was quite straightforward.

A simple example of the translated code was shown in Section 2.5. The ability of this version of Vast to restructure code to remove unnecessary GOTOs was not as good as that of the plusFORT version of SPAG. On two complex Fortran 66 style subroutines, SPAG reduced the GOTO counts from 149 and 67 to 74 and 30 respectively. Vast V1.02X could only restructure these to use 133 and 53 GOTOs. The current version of VAST/77to90 may well be improved in this area.

The Fortran 77 semiconductor code has been developed over a number of years by different people, but has been ported to a range of systems. In the past FTNCHEK and various compiler cross checking tools have been used on it so it was not expected that any significant errors should be caught. A fault that was found was one where a variable used as an array subscript had been declared double precision instead of integer. In these tests interface blocks were not generated for the code. These would have allowed fuller linkage checking, but may lead to problems where array sections have been passed as arguments in non-standard ways.

The transformed code was tested using the Sun f90 (V1.2) compiler and the DEC OSF/1 f90 compiler (V1.3). Both compiled and ran the transformed code without problems, giving the same answers as the original Fortran 77 version. Performance was unchanged on the DEC machine, not unexpectedly as aggressive optimisation (at the expensive of readability) was not requested. The performance of the Sun f90 version was inferior to the original f77 version, but this is due to the poor performance of this compiler version which has been replaced by V2.0.

DEEP

Pacific-Sierra Research 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 2500 lines the US cost is \$125 (2002). A 10% academic discount is available.

3.9   Simulog FORESYS

Vendor

Simulog,
1 rue James Joule,
78286 Guyancourt Cedex,
France.
Website: `http://www.simulog.fr/`

Description

The FORESYS product (FORtran Engineering SYStem) provides a set of modules that cover analysis, inspection, restructuring, parallelisation and interfacing to CASE tools. Features include:

Supports Fortran 90/95 input and conversion of Fortran77 to the new standard. The generation of modules for common blocks and code restructuring is also implemented.

Global analysis to detect defects in subroutine arguments, including pointers and equivalence variables.

Ability to check coding rules for Fortran, including a range of optional rules, such as requiring all variables to be declared.

Reports on 35 different software metrics for each program unit along with global statistics.

Supported systems include Sun Solaris, SGI IRIX, IBM AIX, HP-UX and Digital Unix.

3.10   LDRA - Testbed

Vendor

LDRA Ltd,
131 Mount Pleasant,
Liverpool, L3 5TF, UK.
Website: `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. Testbed does not currently support Fortran 90, though it does deal with Fortran 77 and a wide range of other languages including C, C++ and Ada. Java is not supported.

Static analysis

The functions provided by the static analysis tool include:

- Adherence to language and user defined programming standard;

- Level of structures 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.

3.11   Programming Research Ltd – QA Fortran

Vendor

Programming Research Ltd,
Mark House
9/11 Queens Road
Hersham
Surrey KT12 5LU, UK
Website: http://www.programmingresearch.com

Description

QA Fortran provides a deep flow static analyser to detect errors and inconsistencies. It also provides many measures which can be used to monitor software quality. The software checks 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.

3.12   Other Fortran QA tools

There are a number of QA tools providing similar functionality to QA Fortran and LDRA Testbed for Fortran 77 software, such as *McCabe QA* (from McCabe and Associates, `www.mccabe.com`), and *MALPAS* (from TACS, UK).

4   Software Analysis Tools for C and C++

The C language has been used in many scientific and engineering applications, and is still very popular. It is also used to develop many underlining libraries used in HEC systems, such MPI and PVM. Most compilers now implement what is commonly referred to as ANSI-C, based on the 1990 ISO standard. Some compilers are now adding the new extension defined in the latest standard, C99. Some of the C99 extensions address problems in using C for scientific and numerical computing.

Many new software developments are making use of the object orientated (OO) approach to programming. While OO methods can be implemented, with some significant limitations within Fortran 90/95, it will not be fully supported until the release of the next standard (Fortran 2000). There is some debate as to how useful OO methods will be for numerical computations, but to have proper support for this programming technique should enhance the popularity of Fortran.

The C++ has emerged as the most widely used OO language to date. There is a great potential for more efficient and safer code construction within the OO framework, though C++ is a large and complex language which includes almost all of the procedural language C within it. Hence it is easy to develop "C++" programs that actual make use of the many dangerous and hard to maintain features of C. To avoid such pitfalls a number of authors recommend the use of a set of programming guidelines for C++ software. One such set has recently been published by CERN for their own software development staff and is available on their web site[13]. Several other sets of guidelines and style recommendations for both C and C++ are available on the Internet. Other web pages that gather links to sites offering advice on coding standard include:
`http://www.chris-lott.org/resources/cstyle/`
`http://new-brunswick.net/workshop/c+/faq/coding-standards.html+`

The CERN C++ guide runs to 62 pages of rules and style suggestions, so to check these manually for all new software can be tedious. Hence they are looking at software tools that will test conformance to the guidelines automatically, such as some of the tools discussed in the following sections.

The program "lint" was originally developed to find errors in C programs and was probably one inspiration for many of the checking tools that now exist for C, C++, Fortran and other languages. Most Unix vendors provide a version of lint with their C compilers, often along with other tools such as cflow (call graph information) and cxref (a C cross reference tool). Such tools are not generally provided for C++, perhaps because that language requires more explicit interfaces and such checking is built into the compiler. C++ does suffer from portability problems, since it is a large and complex language and though an ISO standard is now available, both compilers and the existing code base are only slowly converging on this. Hence there is the need for programming guide lines, as mentioned above, and for tools to check the use of these. A report comparing C++ program analysers[14] looked at some of the features of available C++ and concluded that none could check all the rules the authors considered desirable. This report is now several years old, and vendor claims suggest that the checking ability of many tools has improved in this time.

In the following sections we list a number of commercial and open source tools which can help in checking C, C++ and Java code for errors and/or conformance to coding rules. This list is in no way comprehensive, and no attempt has been made to verify the claims made for the tools. It should however illustrate the range of software that is available. Some indication of pricing is given for those tools which openly publish this information.

4.1   CodeCheck

Vendor

Abraxas Software,
4726 SE Division St.,
Portland OR,
97206 USA
Website: `http://www.abxsoft.com/codchk.htm`

Description

The CodeCheck package is described as:

> CodeCheck Version 10.0 is a programmable tool for managing all C and C++ source code on a file or project basis. CodeCheck is input compatible with all variants of K&R, ANSI C and C++. CodeCheck is designed to solve all of your Portability, Maintainability, Complexity, Reusability, Quality Assurance, Style Analysis, Library/Class Management, Code Review, Software Metric, Standards Adherence, and Corporate Compliance Problems. We support Embedded Systems C++ Specification.
>
> - Maintainability-CODECHECK identifies and measures complex, sloppy, and hard to maintain code.
> - Portability-CODECHECK identifies code that will not port between DOS, OS/2, Unix, VMS, Microsoft Windows and the Macintosh, and porting to 64-bit machines. Validate Embedded Systems targets.
> - Complexity - Measures program size (Halstead) Object Oriented Metrics, Program Cost, and Program Complexity (McCabe).
> - Compliance - CodeCheck allows your corporate C/C++ coding and project specification standards to be completely automated for compliance validation.
> - Flexible Licensing, Floating Licenses for ALL operating systems;
>
> CodeCheck includes pre-written expert system Rule Files that can be applied to any C or C++ project for: POSIX, SVID, X3J11, Ellemtel, Meyers, Plum-Hall, and ANSI C/C++ Source Code Conformance Validation. Program Complexity (McCabe), Program Size (Halstead), General Maintainability, C++ Style Checking, Hungarian prefix Checking, Corporate Compliance Analysis, Comeau's Style Guidelines, and porting to 64-bit machines.

Cost

Prices quoted on the website start at $995 for a single user floating license, with a non-floating Linux license for $495.

4.2   CodeCenter, ObjectCenter, TestCenter

Vendor

CenterLine Software C++Expert (not called this now) Unix http://www.centerline.com/

CenterLine Development Systems Inc.
145 Rosemary Street,
Suite H Needham,
MA 02494 USA
tel: 781-444-8000 fax: 781-444-1146
Website: `http://www.centerline.com/`

Description

CenterLine provide a set of tools for testing and developing C and C++ software. Their products mainly run on Unix platforms (including Sun, HP, and IBM). Some tools, such as TestCenter, can be used in conjunction with other development environments while others such as ObjectCenter and CodeCenter are complete IDEs.

Their products include:

- TestCenter helps programmers thoroughly test C and C++ code. It is tightly integrated with programming environments such as ObjectCenter, CodeCenter, SPARCworks, and SoftBench but remains a non-intrusive way of testing code while it is being written.

- ObjectCenter is a C and C++ programming environment supporting multiple UNIX platforms. It provides a component-based approach to developing C++ code as well as run-time checking to enable the rapid development of quality software.

  Some key features are listed as:

  - An interpreted, Interactive Workspace for component development, debugging, and rapid prototyping

  - Process-and-component-level debugging modes for power and scalability.

  - Automatic static and dynamic runtime detection of over 250 errors for optimal code quality

  - Integrated graphical browsing tools for visualizing class hierarchies, finding and reusing classes, and understanding control flow

  - CenterLine-C++ compiler and incremental linking for rapid edit-to-execute time

  - Support for multiple UNIX workstations, compilers, tools, GUIs, and frameworks

- CodeCenter is a C programming environment for prototyping, building, testing, debugging, enhancing and maintaining C programs on UNIX systems. It provides similar features to the ObjectCentre product in terms of rapid development and testing of C code, though without the ability to deal with C++.

4.3   FlexeLint, PC-Lint

Vendor

Gimpel Software,
3207 Hogarth Lane,
Collegeville,
PA 19426 USA
Website: `http://www.gimpel.com/`

Description

This commercial version of Lint has been extended to include C++ as well as C. It is described as:

> PC-lint and FlexeLint will check C/C++ source code and find bugs, glitches, inconsistencies, non-portable constructs, etc. PC-lint for C/C++ runs on Microsoft Windows NT/95/98/2000/ME/XP, DOS, and OS/2. FlexeLint for C/C++ runs on most Unix platforms including Linux.
>
> Features include:
>
> - value tracking to detect subtle initialisation and value misuse problems
>
> - Inter-function Value Tracking – The powerful inter-statement value tracking has been extended to cross function boundaries. Functions called with specific values are later processed, with these values used to initialise parameters. To take full advantage of inter-function tracking, a multi-pass operation has been introduced.
>
> - with value tracking as an enabling technology, we support semantics checking for almost 100 library functions, this checking can be extended to user functions.
>
> - optional strong type checking (typedef-based) with a rich option set to detect nominal type differences. You can even form a fully checked type hierarchy of scalar types using only typedef
>
> - user-defined semantic checking for function arguments and return values
>
> - find unused macros, typedef's, classes, members, declarations, etc. across the entire project.
>
> - checks flow of control for possibly uninitialised variables.
>
> - explicit support for a subset of the MISRA (Motor Industry Software Reliability Association) standard

Cost

Prices for pclint start at $239, and FlexeLint at $998.

4.4 Code Advisor

Vendor

Hewlett-Packard Company.
Website: `http://www.hp.com/`

Description

If you purchase HP's C++ SoftBench, you can use SoftBench CodeAdvisor to check your code for critical coding violations beyond compiler errors. SoftBench CodeAdvisor allows you to check the entire project, selected targets, or selected files.

Softbench's CodeAdvisor tool provides a sophisticated rules-based source code checker that scans source code for these hard to find errors. Critical defects can be found quickly and efficiently, resulting in higher quality code, shorter cycle time, and greater portability for code written by both novice and experienced C++ developers.

4.5 CodeWizard, Insure++

Vendor

Parasoft Headquarters
2031 S. Myrtle Ave.
Monrovia, CA 91016 USA
Website: `http://www.parasoft.com/`

Description

CodeWizard is available under Windows and most Unix systems and is a tool that checks C and C++ source code for conformance to a set of style and programming rules. The pre-defined rules are selected to automatically identify dangerous coding constructs that compilers do not detect. CodeWizard allows the user to specify additional rules to conform to a given policy.

The features of the tool are listed as:

- Reports sophisticated violations that compilers don't catch

- Organises coding guidelines by subject and severity, and selectively implements guidelines by means of a multi-level suppression/ unsuppression feature

- Guidelines can be implemented at the company, group, and individual level

- Pinpoints the exact location of programming and design violations

- Provides a full, detailed explanation of each violation

- Lets you customise built-in coding guidelines or create new ones using a graphical point-and-click interface

- Creates custom guidelines quickly with new Auto-Create feature

- Identifies potential errors when porting from 32-bit to 64-bit architectures

Insure++ provides runtime testing for C and C++ codes. It replaces the normal compiler so that monitoring can be included around all code statements. The package is available on Windows and most Unix systems. Insure++ detects many memory errors that are common in C and C++ such as allocation errors, leaks and corruption. It can also identify variable initialisation errors and definition conflicts.

4.6   QA C, QA C++

Vendor

See QA Fortran entry.

Description

QA C and QA C++ provide static analysis for C and C++. The software is available under Windows and most Unix systems including Linux. A version for Java, QA J, is also available on Windows, Sun Solaris and Linux systems. Some common features of the products are listed as:

- Rigorous enforcement of ISO/IEC C Standard 9899:1990 or ISO/IEC C++ Standard 14882.

- Identification of undefined or unspecified behaviour and problematic object-oriented implementations.

- Detection of potential portability problems

- Flexible configuration to support compiler language extensions

- Detection of bugs and dangerous code

- Enforcement of coding standards and style conventions

- Powerful GUI environment

- Optional command line interface

- Integration into leading IDEs and programmer's editors

- Industry standard metrics

- Function structure diagrams

- Code and data relationship diagrams

- Quality profiling using demographic analysis

- Best practise advice on your code

4.7 Rational Purify, PurifyPlus, PureCoverage

Vendor

Rational Software Corporation
18880 Homestead Rd.
Cupertino, CA 95014 USA
Website: `http://www.http://www.rational.com/`

Description

Rational Software provide a range of tools for C, C++ and Java which run on Windows and most Unix systems. Products include:

- Purify: Identifies execution errors and memory leaks within applications. No recompilation is required to instrument the code and the source need not be available.

- PurifyPlus: Includes the memory leak checking of Purify with details of code coverage and run time calling sequences.

- PureCoverage: Gives reports on code coverage and untested parts of the software.

4.8 Valgrind

Vendor

Julian Seward, Nick Nethercote.
http://developer.kde.org/ sewardj/+

Description

Valgrind is a GPL tool for finding memory management problems. It is currently only available on Linux systems. Checks include:

- Use of uninitialised memory

- Reading/writing memory after it has been free'd

- Reading/writing off the end of malloc'd blocks

- Reading/writing inappropriate areas on the stack

- Memory leaks – where pointers to malloc'd blocks are lost forever

- Passing of uninitialised and/or unaddressable memory to system calls

- Mismatched use of malloc/new/new vs free/delete/delete

- Some misuses of the POSIX pthreads API

The tool can be used directly on a compiled executable without recompilation and tracks all memory access operations.

4.9   Other Open Source Tools

There are a number of GPL or similarly licenced tools that can be used principally to identify errors in C and C++ code. These include:

- ElectricFence. This detects errors in memory obtained via malloc and similar system calls. It is available for most Unix systems and replaces the normal malloc library at link time. It uses virtual memory hardware to detect "out of bounds" memory errors.

- Splint (previously lclint). This is a static analysis tool for C (ISO C99). It does not deal with C++. It is particular aimed at finding possible security holes in C code. The user is required to add certain annotations about argument use to the code via comments. Splint then checks for possible coding errors using these conditions, as well as the more usual lint style checks.

- dmalloc. This is another memory fault debugging library for C programs, though it may also be used with C++.

- YMAD. Also a library for C and C++ memory error detection, this one comes as a dynamic library to avoid the need to relink files.

- Clint. This is a static checking tool for C++ that provides link like warnings. Additional programming rules can be added written in python.

- Gcc. This compiler provides options to check many of the errors that were traditionally reported by lint. The suite supports C, C++, Objective-C, Fortran 77, Java and Ada.

- Jlint.A version of lint for java code.

- JCSC. Also inspired by lint, this is a static coding checker for Java. It can check software against a defined coding standard.

5   Case Study: Applying some QA tools to an existing application

5.1   The Test Code: A Fortran CFD Application

To illustrate the usefulness of one of the commercial tools we have applied plusFORT SPAG and some of the NAG tools to an existing Fortran application code. A CFD (Computational Fluid Dynamics) program was selected to run the tool on. The code consists of about 23,500 lines of Fortran, which does include some Fortran90 features, such as allocatable arrays and modules, but is really based on an earlier Fortran77 version of the software. This is likely to be a common situation for academic codes where the basic core of the program has remained unchanged while later additions exploit newer features of the available compilers.

SPAG's ability to understand most Fortran90 constructs and to be able to restructure them is particularly useful in this case. Detailed options files are provided with control which structures should be replaced. For example, labelled DO loops can be replaced with the DO/ENDDO version, multi-way IF/THEN/ELSE blocks can be converted to SELECT/CASE statements, etc. The appearance of the final code can be controlled in terms of indentation and appearance of keywords and variable names.

5.2   Code metrics

SPAG can provide three basic metrics for each subroutine in the program. These are described in the manual as:

1. LOGICAL CONTENT. This is a measure of the number of decision points (such as IF or ELSEIF statements) in the subprogram. The target value is 50 (corresponding to a McCabe Cyclomatic Complexity of around 10). This metric can be reduced by splitting up large subprograms. Restructuring has little effect on it.

2. ALL CONTROL JUMPS. This is a measure of how discontinuous control flow is in the subprogram. Any GOTO or other command which transfers control to a different point in the source code contributes to this metric. The target value is 20. Restructuring often reduces this metric appreciably.

3. UNSTRUCTURED JUMPS. Similar to ALL CONTROL JUMPS except that jumps from within a DO loop to or just past the terminal statement (using GOTO, CYCLE or EXIT), and jumps to a RETURN, STOP or END statement are excluded. The target value is 0. Restructuring often reduces this metric appreciably.

For the 94 subroutines of the test code, all 7 already had the metrics of zero for metrics (2) and (3). This indicates that the code mainly avoids unstructured loops and GOTOs. In the cases where the All Control Loops metric was non-zero, it was always much less than the suggested guide line of 20. In the 4 subroutines with unstructured jumps, the metric was always had the value 3, and could often be traced to GOTO's used to terminate a loop early. SPAG was able to convert these to block IF/ENDIF statements and reduce these metrics to zero. One other case existed where a WHILE loop had been implemented using a GOTO. This case could also be eliminated by allowing SPAG to convert it to a proper WHILE loop.

Metric (1) is a measure of complexity and as noted in the manual, restructuring does not reduce it. For the example code 49 out of the 94 subroutines had this metric above the recommended value of 50 and 40 were above 74. This indicates that many subroutines have become too large, and this is supported by inspection of the worst cases. The routine with the highest metric value (90), has 590 lines of code with 90 separate IF statements. If seems likely that this could be greatly reduced by a more systematic approach to the way each Cartesian axis is treated, rather than replicating large amounts of code, as is done at present. The usefulness of SPAG metrics here is in highlighting those routines which are excessively complex. It is up to the programmer to decide if it is possible to rewrite the subroutines or break it into smaller units.

5.3   Conversion to Fortran 90 Form

As the code already made use of some Fortran 90 constructs, it would seem sensible to use SPAG to more fully exploit the new format. As noted previously, SPAG is able to perform an number of transformations including adding INTENT statements to arguments, conversion of common blocks to modules, and other more cosmetic changes.

Overall the conversion was done efficiently, though as the code already made use of modules for most of its data, the main benefits were the uniform indentation of the code, the removal of unused variables and unnecessary labels plus the added INTENT statements. One problem was seen with block data statements for the one remaining common block in the original code. This was not correctly converted to a data statement in the new module.

5.4   Compiler testing options

While the primary goal of a compiler is often seen as generation of efficient code from any valid source code presented to it, the ability to detect programming errors is more important in the development of quality software. While all compilers can be expected to reject obvious syntax errors, good ones will provide additional warnings about possible mistakes, such as variables that are used before being set. The explicit interfaces and intent statements which are available in Fortran 90 means that many compilers are now able to make checks which previously were only possible using tools such as ftnchek.

A good comparison of many Linux and Windows Fortran compilers is given on the Polyhedron web site, www.polyhedron.co.uk. They include a set of diagnostic tests to compare how effective each compiler is at trapping common errors. The results show a very wide spread in capabilities with best results been given by the Salford FTN95 compiler on Windows and the Lahey LF95 compiler on Linux. The Portland Group compiler, pgf90, was one of the poorer compilers in terms of diagnostics and error detection.

The Lahey compiler was used to build the CFD application with all diagnostics enabled. The compiler was able to identify a few cases where variables had been used before set. It can sometimes do this at compile time for obvious cases, but it can also do much more comprehensive run time checks. These slow down the code considerably, but are very effective. While most of the errors identified did not actually alter the output of the code, at least one case would have done so. An incorrect variable had been used in one expression. Since the incorrect name actually existed, the mistake would not be caught by use of IMPLICIT(NONE). Since the variable had not been initialised at the point it was first used, the Lahey compiler reported the error at runtime.

5.5   Conclusions

The CFD application we looked at was still in a state of rapid development. Software tools such as SPAG and the NAGTOOLS provide a good way to analyse and tidy up the existing code. To be useful, this requires the developer to accept the new version of the code as the working standard.

The use of metrics helps to identify routines that are too complex, and to some extent the restructuring tools can help in this area. However, in several areas of the code it was felt that logical structures were unnecessarily complicated and the only way to fix this would be a complete re-design of the routines in question.

Run time checks implemented in compilers are no substitute for using a full set of test cases on the code. They can be useful in catching some errors which otherwise might prove difficult to track down.

6   Software development tools for Source Code Control

6.1   The need for version control

Consistent version control is essential in any non-trivial software development. As changes are made to the software which may introduce new bugs it is extremely useful to know which parts of the code have been altered and for what reason. When there is to be more than one programmer working on the software it is also essential that they can take locks on the parts that each is

modifying to prevent the conflicting changes been to a single file. Some tools, such as CVS, also allow two programmers to work on the same source file, if required. Changes can then be merged back together, with human intervention only where necessary.

There are many different version control tools available, both commercial and free, offering differing levels of sophistication. A brief introduction to the use of the three most well know tools, SCCS, RCS and CVS, is given in [2].

6.2 SCCS

SCCS is the Source Code Control System, and is the "standard" source control system provided with most commercial version of Unix. The standard interface to SCCS is via the command line, and the user can check-in the current version of a program. When editing is required on one or more routines, these can be checked-out of the repository, edited and then returned, along with a log message to describe the changes made.

As the original source of this package is not in the public domain, the version supplied is dependent on the software vendor. Some vendors have integrated SCCS into the GUI based IDEs. For example, the Sun WorkShop environment includes a graphical tool for managing files stored under SCCS.

A Linux "How-to" on RCS and CVS makes the following observation:

> SCCS (Source Code Control System) is no longer being enhanced or improved. The general consensus has been that this tool is clumsy and not suited to large numbers of users working on one project. Actually, SCCS interleaves all the versions, but it can make new development get progressively slower. Hence, SCCS is NOT recommended for new projects; however, it is still there to support old code base in SCCS.

A public domain clone of SCCS is available, called GNU CSSC ("Compatibly Stupid Source Control") This aims to be a relatively unembellished copy of the traditional SCCS. More details are available on the web at: `cssc.sourceforge.net/index.shtml`

6.3 RCS

RCS is the Revision Control System and is a more recent development that tries to avoid the problems found with SCCS. The source code of this package is freely available under the GPL (Gnu General Public License). Versions are available for Windows systems as well as for Unix.

RCS is used as the base file storage mechanism by some other version control systems, such as CVS. These systems provide additional facilities on top of the basic RCS interface. RCS on its own provides a complete set of command line options to check in new versions of files and to lock ones that are currently been edited. This simple form of locking prevents conflicting changes been made to one routine, in a similar manner to that used in SCCS. It does, however, prevent any parallel development on a single file.

RCS commands to extract the most recent version of set of files can be included in Unix make-files. Some IDEs also include RCS to automatically keep track of software versions for the user. Examples include the Elixir IDE from Elixir Technology, for Java, and the Moonshine Professional IDE, for C/C++.

6.4 CVS

CVS (Concurrent Versions System) is another public domain package for version control that is
very widely used in large open source projects. It uses RCS but extends it to support things such
as the integration of multiple changes made independently to one file. CVS is another command
line based system, though several GUIs have been developed which can be used with it. These
include LinCVS, based on the Qt toolkit and tkCVS which is a popular interface based on the
tk toolkit. A version of tkCVS is also available under Windows.

From the web site `freshmeat.net` it can be seen that CVS, and a host of related graphical and
web based front ends to it, are by far the most active of the open source version control systems.
The main features which have helped it become so popular are:

- A client-server access model allowing people to easily access the repository from anywhere
  on the web,

- The ability to allow unreserved check-out of files which avoids artificial conflicts common
  with the exclusive check-out model.

- The client tools are available on most platforms.

6.5   Other version control systems

While CVS is the dominant version control system at present, some other open source systems
are also under active development. These include:

Subversion . This tool is still at an earlier stage of development but aims to be a replacement
for CVS. It is intended to reproduce all the good features of CVS while making improve-
ments in many areas such as speed and features of the code that can be stored in the
repository.

Aegis Aegis is a transaction-based software configuration management system. It provides a
framework within which a team of developers may work on many changes to a program in-
dependently, and Aegis coordinates integrating these changes back into the master source
of the program, with as little disruption as possible. Aegis supports geographically dis-
tributed development. A new baseline is only accepted one user defined tests are passed.

Anthill This is another tool to provide code check out. It can also run nightly regression tests
and report code metrics on each build.

Katie Katie is a revision control system, somewhat like a cross between CVS and NFS, that was
inspired by Rational ClearCase. The three most interesting features of Katie are that the
repository is mounted as a file system (rather than being copied to a local workspace), that
all versions of all files (even deleted ones) are accessible through this file system (so there is,
for example, no need for a Katie-specific diff command like CVS has), and that directories
are versioned (just like files are). This package is at an early stage of development.

7   Software development toolkits and IDEs

Software Development Toolkits (SDK) and Integrated Development Environments (IDEs) gen-
erally provide a more user friendly and productive way to program than traditional techniques.

In the past it was usual have the standalone compiler and perhaps a command line debugger under which the executable could be run. While this approach is still possible, and preferred by some, it is now common to find packages that combine graphical language sensitive editors with compilers, make tools, graphical debuggers and source code analysis tools.

To take one example, the Sun ONE Studio 7 Enterprise IDE for Solaris provides compilers for C, C++, Fortran 95 and Java. These are integrated with a range of graphical editors, debuggers, performance analysis tools, make tools and user interface development tools. A single common graphical user interface makes it easy to move through the various tools in the development and debugging process. Similar environments are offered on most Unix systems and are the norm for many Windows products such as Visual Fortran and Microsoft visual C++. The common Microsoft Visual development environment provides an integrated text editor, a resource editor for Win32 user interfaces, a graphical debugger, a language aware source browser, project manager for automating builds, a profiler and Microsoft's visual Source Safe for version control.

Sun's Java package is also available separately as a Software Development Kit. Though this does include a range of tools for compiling, interpreting and debugging Java, the emphasis is more the wide range of support libraries that the package offers for things such as web based transaction, user interface design and graphical output.

Clearly there are very many tools that provide IDEs and SDK for Fortran and other languages and it would be impossible to list them all, let alone make a qualitative comparison between them. Instead we list a sample of the available products which are in more common use on current systems.

7.1 Sun Studio One

Sun. Website: `www.sun.com`

Description

As mentioned above the Sun Studio One product provides a single development environment for Fortran, C, C++ and Java. While the individual tools can still be used from a command line interface, the graphical interface offers a much easier way to debug and update code. For example, changes made in the debugger can be instantly added to the executable with a full recompile and tested without having to leave the debug window. Access is also available to source browsing tools that can display the call tree structure and do some QA checks on the software.

7.2 SGI ProDev WorkShop

SGI. Website: `www.sgi.com`

Description

SGI provides the ProDev WorkShop tool as IDE for software development under IRIX. Its four main components are:

- Visual Debugger, which allows changes to be made to the code at run time, support for very large and distributed programs.

- SpeedShop - Powerful Performance Analyzer providing a graphical presentation of CPU and memory requirements to identify bottlenecks.

- Integrated Build Manager to recompile software from within the IDE.

- Graphical Code Analyzer to provide detailed understanding of the code. As well as call trees this provides a number ways of viewing C++ class relationships.

7.3   N.A. Software FortranPlus

N.A. Software Ltd
Roscoe House
62 Roscoe Street
Liverpool L1 9DW
Website: 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:

FortranPlus This is a Fortran 95 compiler for Sparc, Intel and Power PC 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.

7.4   Fujitsu Visual Analyzer

Vendor

Lahey Computer Systems,
865 Tahoe Blvd.
Box 6091
Incline Village
NV 89450 USA
Website: http://www.lahey.com

Description

Lahey have recently released 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 is $395 in the USA. The product is included with LF95/Pro compiler.

7.5 Totalview debugger

Vendor

Etnus
Website: http://www.etnus.com

Description

Totalview is not really an IDE, but is included here as an advanced debugging environment, which can be used for C, C++ and Fortran development on a wide range of systems. It deals with parallelism from MPI and OpenMP codes and is able to deal with large amounts of data.

7.6 Other Open Source IDEs/SDKs

Other IDEs available as Open Source projects include:

Anjunta IDE; Anjuta is a versatile Integrated Development Environment (IDE) for C and C++ in GNU/Linux. It has been written for GTK/GNOME, and features a number of advanced programming facilities. These include project management, application wizards, an onboard interactive debugger, and a powerful source editor with source browsing and syntax highlighting.

KDevelop: This is an integrated development environment which makes the creation and development of GNU Standard Applications an easy task even for beginners. Highlights of the current release are: an application wizard for easy creation of KDE, Qt, GNOME, and terminal C/C++ projects, full project management, a syntax-highlighting editor, an integrated dialogeditor for the Qt/KDE GUI libraries, an internal debugger, a full-featured classbrowser with classtools, CVS support, an integrated HTML-based help system offering manuals and class-references, and extensive search mechanisms to browse sources and documentation.

Motor: This is described as: Motor is a text-mode integrated programming environment for Linux. It consists of an editor with syntax highlighting, a project manager, a makefile generator, gcc, ctags, gdb, autoconf/automake and grep front-ends. CVS integration is also provided. It allows one to edit, compile, and debug programs without a need to leave the IDE, automatically check in/out files from a CVS repository and import projects into CVS, and generate distribution packages (tar.gz and RPM).

JDEE Java Development Environment for Emacs (JDEE) is an Emacs-based integrated development environment (IDE) for developing Java applications and applets. Features include multiple code browsers, a JPDA-based debugger, method and field completion, template-based and procedure-based code generation, Java source code interpreter, context-sensitive help, and more.

8   Summary

This document has given a brief listing of a range of available software tools that can be utilised to improve software development for the major languages, Fortran, C, C++ and Java. Clearly there is a very wide choice of both commercial and open source products around and this survey is by no means comprehensive. However it does give a indication of what is on offer. It is up to the user to select what is most appropriate for any given project and to find the tools that work for him.

With so many tools available for free on the Internet, it makes sense to collect ones that are applicable the programming work been undertaken. Even if your software currently contains no bugs, it would be useful to have the reports from software tools to confirm this.

References

[1] R.F.Fowler and C.Greenough, *Software quality assurance for Fortran 90 : a survey of available tools*, RALTR 1999046, CLRC Rutherford Appleton Laboratory (1999). Also available as a UKHEC technical report from www.ukhec.ac.uk.

[2] M.Antonioletti, E.Breitmoser and R.J.Allan, *Software engineering and Code Development for HPC Applications*, UKHEC Technical Report, (2001). Available from www.ukhec.ac.uk.

[3] L. Hatton, *The T Experiments: Errors in scientific software*, IEEE Comp. Sci. & Eng., **4**, No. 2, (April-June 1997), 27.

[4] The Journal: ACM Transactions On Mathematical Software publishes algorithms which are available on line from: http://www.acm.org/calgo/contents/.

[5] B.A. Nejmeh, *NPATH: A measure of execution path complexity and its applications*, Comm. ACM., **31**, No.2, (1988), 188-200.

[6] T.J. McCabe *A complexity measure*, IEEE Trans. Soft. Eng., **SE-2**, No. 4, (1976), 308-320.

[7] R.J. Allan & C.J. Müller, *SMP Programming*, Technical Report, (CLRC Daresbury Laboratory, 1999).

[8] *Re-engineering Technical Report, Volume 2*, US Air Force Software Technology Support Center (1995) http://stsc.hil.af.mil/RENG/REENGv2.html.

[9] T.R. Hopkins, *Is the quality of numerical subroutine code improving?* in "Modern Software Tools for Scientific Computing", E. Arge, A.M. Bruaset and H.P. Langtangen (eds.) (Birkhäuser, 1997) ISBN 0-8176-3974-8.

[10] T.R. Hopkins, *Renovating the collected algorithms from ACM* ACM Trans. on Math. Software, **28**, No. 1, (2002) p.59.

[11] J.E. Moreira, S.P. Midkiff and M. Gupta, *From flop to megaflops: Java for technical computing*, ACM Transactions on Programming Languages and Systems, **22**, No. 2, (2000).

[12] M. Antonioletti, G. Darling, J.V. Ashby and R.J. Allan, *Applied Numerical Libraries for Parallel Software*, UKHEC Report, To be published.

[13] ATLAS Quality Assurance Group, *ATLAS C++ Coding Standard*, ATL-SOFT-2002-001 (2002). Available at:
http://atlas.web.cern.ch/Atlas/GROUPS/SOFTWARE/OO/qa/General/

[14] S. Meyers and M. Klaus, *A First Look at C++ Program Analyzers*, Dr. Dobb's Journal, Feb. Issue, (1997). Available at: `http://www.aristeia.com/ddjpaper1_frames.html`.