Publication/Software Review

An Initial Evaluation of the NAG f90 Compiler

MICHAEL METCALF
CERN, 1211 Geneva 23, Switzerland
(metcalf@cernvm.cern.ch)

ABSTRACT

A few weeks before the formal publication of the ISO Fortran 90 Standard, NAG announced the world's first f90 compiler. We have evaluated the compiler by using it to assess the impact of Fortran 90 on the CERN Program Library. © 1992 John Wiley & Sons, Inc.

1 BACKGROUND

Given the long development time of the Fortran 90 standard and the gloomy predictions about the availability of compilers, the viability of the language, and even the difficulty of implementing it fully, it was with some surprise that we learned of the announcement of a full f90 compiler by NAG on June 10, 1991 (see Table 1). This followed the completion of the standard by WG5 (the ISO technical committee) and X3J3 (the ANSI technical committee) in the spring, and preceded the formal publication of the standard by ISO in August [1].

An evaluation of the compiler was carried out using 250,000 lines of code from the CERN Program Library. During the evaluation, a few errors were detected, including:

1. A segmentation fault when raising an integer to a double-precision power (but a correction to the *.h* file was supplied by return when the error was reported).
2. In a construct whereby part of an array is initialized by DATA statement operating through an EQUIVALENCE statement, the array is initialized from the first array element, even if the EQUIVALENCE statement explicitly references an element other than the first.
3. A problem with an ENTRY point.
4. Some detailed problems with some new f90 features.

All of these were promptly corrected by NAG for the subsequent release of the compiler (Release 1.1). Otherwise, the compiler was found to be quite reliable and, in addition to the conclusions listed in Table 1, we found that the evaluation was useful in gaining experience in real-world use of Fortran 90, even if mainly in a FORTRAN 77 context, and we established that it was possible, with a modest effort, to convert the code of the Library to run under f90.

As a consequence, CERN decided to purchase a site-wide license for its unix and VMS platforms.

2 FORTRAN 90

Fortran 90 contains the whole of FORTRAN 77, which greatly facilitates the change to the new

Received February 1992
Revised April 1992
Accepted April 1992

Scientific Programming, Vol. 1, pp. 91–95 (1992)
CCC 1058-9244/92/010091-5$04.00

Table 1. Information on the Fortran 90 Compiler

Product	Fortran 90 compiler.
Address	NAG Inc., 1400 Opus Place, Suite 200, Downers Grove, IL 60515-5702. Tel: (708) 971 2337 or fax (708) 971 2706. NAG Ltd., Wilkinson House, Jordan Hill Road, Oxford OX2 8DR, UK. Tel: (0865) 511245 or fax (0865) 310139.
Price	About $1,000 for a single workstation with significant discounts for site licenses.
Availability	Most major unix platforms: HP 9000/400 and 700, Sun SPARC, DEC-Station 5000, IBM RS/6000. A PC version under DOS 3.3 and above. A VMS version is also available.
Trouble-shooting	NAG provides a "Response Center" for handling problems by phone, fax, or e-mail.
Test details	Using Release 1.0a(129), a complete set of tests was carried out, as reported below. They were all performed on an HP/Apollo 9000/425 with 12 Mbytes of memory under Domain OS 10.3, and comparisons are with ftn Rev. 10.85(260) with optimization. The size of the compiler on the test platform was 1.2 Mbytes. On the Apollo, the *-frnd* option (for improved floating-point rounding) is taken by default.

Main conclusions

1. The compiler is, at least for FORTRAN 77 code, reliable.
2. It is simple to use and gives good diagnostics.
3. It runs fast enough for large-scale tests of Fortran 90 to be performed effectively.
4. It can handle large amounts of code in single runs.

Thus, it can be recommended for use by all those wishing to evaluate the impact of Fortran 90 on their existing codes and to test Fortran 90 features for future use with native compilers. It is an important tool in implementing the changeover to the new standard.

standard. In addition, it has many other important new features. These are fully described in [2], but a brief summary is provided below.

1. Array operations
2. Pointers
3. Improved facilities for numerical computation
4. Parametrization of the intrinsic data types
5. User-defined data types [3]
6. Encapsulation via modules
7. A new source form
8. New control structures—CASE and DO
9. Internal and recursive procedures
10. Optional and keyword arguments
11. Dynamic storage allocation
12. Many new intrinsic procedures

3 THE NAG COMPILER

The compiler uses C as an intermediate language, and thus relies on the native C compiler to produce assembler and then object code. There is an obvious optimization penalty involved here, and one object of the evaluation was to determine its magnitude. The compiler performs six passes:

1. Lexical and syntactic analysis, building the symbol table and an abstract syntax tree
2. Semantic analysis, annotating the parse tree and filling in the symbol table
3. Code generation by parse tree transformation
4. Output of C source code from the flattened parse tree
5. Compilation using the native C compiler
6. Linking using *ld* and the f90 run-time libraries supplied by NAG

Each pass is distinct from the others in order "to improve maintainability and to allow the reuse of components." This is an important statement by NAG: it opens up the possibility of hardware vendors attaching the front-end to a native back-end, and marketing the resulting product as a native compiler. It would be a significant simplification for the providers of libraries and fits in well with the unix concept of portability. Also, it would overcome the fact that, by its very nature, the NAG compiler can never be as highly optimized as a native one and, therefore, is not by itself suitable for large-scale production.

The compiler performs extensive checking. On the first pass it makes a complete check of the

syntax, and issues warnings and error messages as problems are detected. The second pass starts only if no errors are detected on the first. Here the semantics are checked, and the compiler is particularly good, but not perfect, at detecting such things as variables that are used but not defined, inconsistent use of actual arguments, and breaches of the typing rules. It sometimes takes a while to get through this pass successfully, but the reward is then a program with fewer errors than is usual with some compilers, and if interface blocks are also made available, then this is even more the case. The compiler performs a level of error checking often only achievable by the use of additional tools.

All diagnostics are issued with respect to the Fortran file line numbers, even if issued during the C pass. This is possible as the original names and line numbers are passed on to the C step (using the #line directive to ccp). This makes the use of a debugger possible. To test the principle, I introduced a deliberate error at line 22 of the following code, changing a .LT. to .GT., leading to the extraction of the square root of a negative number.

```
22 IF (D.GT.0.) THEN
23    COMP = CMPLX(-B/(2.*A),
        SQRT(-D) / (2.*A))
   :
27 ELSE
28    SQRTD = SQRT(D)
29    REAL1 = (-B + SQRTD) / (2.*A)
30    REAL2 = (-B - SQRTD) / (2.*A)
   :
32 ENDIF
```

I abandoned an attempt to use *dbx* after it had crashed OS on a first attempt then, after it had identified successfully a floating point exception at line 29 with ftn, finally gave a segmentation fault running the f90 test. Turning to *dde* instead, this too identified the exception at line 29 for ftn. With f90, it identified an exception in the square root function, and by invoking the *trace* option it told me the sqrt in question had been called at line 28.

For information, the C code of the snippet above looks thus:

```
# line 22 ''debug.f''
if (d_>0.) {
# line 23 ''debug.f''
tmp3.im = sqrt_r((tmp2 = -d_, &tmp2))/(2.*a_);
# line 23 ''debug.f''
tmp3.re = -b_/(2.*a_);
# line 23 ''debug.f''
comp_ = tmp3;
 . . .
# line 27 ''debug.f''
}else{
# line 28 ''debug.f''
sqrtd_ = sqrt_r(&d_);
# line 29 ''debug.f''
real1_ = (-b_ + sqrtd_)/(2.*a_);
# line 30 ''debug.f''
real2_ = (-b_ - sqrtd_)/(2.*a_);
 . . .
```

The compiler is invoked simply by typing **f90 name** for a file in the new free source form, or **f90 name.f** for an existing fixed form program. This is followed by the usual *a.out* command for execution. Thus, its use is simple and straightforward. Various options, in particular *-O* for optimization and *-c* to skip linking, are available.

4 THE EVALUATION

The evaluation was carried out using 250,000 lines of existing code from the CERN Program Library. Some of the more informative points are given here.

During the course of testing, it was established that a limited number of changes to the code were necessary. They mostly concerned features that one might describe as being on the edge or beyond the old standard. With one exception they were trivial to implement and are fully described by Metcalf [4]. The most significant changes were to nonstandard length specifications (e.g., COMPLEX*16), to Hollerith constants in DATA statements, and to replace some nonstandard double-precision complex intrinsic functions (e.g., DCMPLX).

4.1 KERNNUM

KERNNUM is a set of 210 subprograms, totalling 11,124 lines of code. It forms the kernel of the mathematical part of the Library. A test program of a further 10,400 lines exercises all the entries over legal and illegal data sets. The test was rather straightforward in that only three lines of code had to be added: one type declaration of an EXTERNAL name, and two SAVE statements. The compilation time of KERNNUM was 295 seconds, a

factor 1.14 longer with f90 than with ftn. The execution time of the test was 148 seconds, a factor 1.31 longer.

A module containing interface blocks to the whole of KERNNUM was generated automatically (using an option in a source code conversion program) and, after fixing one error, was successfully compiled and tested. Compiling a module with a name *kernnum* produces a file called *kernnum.mod*, and if the compiler encounters a USE statement as in

```
use kernnum
call ranf(x)   ! ranf is a function
```

it searches automatically in that file and, in this case, detects that the function reference is incorrect.

4.2 KERNGEN

KERNGEN contains 248 subprograms totalling 8,392 lines. It forms the general purpose kernel of the Library—basic mathematical routines, bit, byte, and character handling, and various utilities. It is exercised using a test program of 4,450 lines. Both the Library and the test program were replete with Hollerith constants in DATA statements, and these were all replaced as described [4], no mean task. In addition, the test program required interface blocks for eight generic entries (library utilities that handle different data types through a single entry), and the corresponding ENTRY points had to be added to the Library. A small number of other changes were made: a few occurrences of INTEGER*2, and some INTEGER/CHARACTER equivalences.

The code took 131 seconds to compile and 0.8 seconds to execute; the execution time is 60% longer than under ftn.

4.3 GENLIB

This is a library of 413 subprograms running to 31,342 lines of mainly mathematical code. A significant part of it is exercised by a 4,000-line test program. Apart from a small number of conversions as already described, the main change was to replace the nonstandard intrinsic functions DCMPLX, DIMAG, and DREAL by accessing Fortran 90 features, as described [4]. The Library compiled in 952 seconds (1.05 times longer than with ftn, but in less than half the real time), and the test ran for 42 seconds, a factor 1.53 longer than with ftn.

4.4 JETSET

One of the physics codes used in the evaluation was JETSET (T. Sjostrand, University of Lund). This is an event generator extensively used in simulations. It is a stand-alone program of 10,000 lines written in pure FORTRAN 77, but containing many complicated expressions. It was intended to make a comparison with ftn, but this proved impossible. Compiled under ftn, with or without optimization, JETSET produced very wrong results. It was impressive that f90 was more successful. In spite of a rather long compile time (390 seconds, 2.7 times longer than ftn), JETSET worked correctly. No changes to the code were required.

A subsequent run on a Sun, without optimization, gave a run time of 120 seconds, 20% slower than under the native Sun compiler.

4.5 GEANT

The simulation program GEANT [5] is the most widely used in high-energy physics, and its conversion was regarded as the final "challenge" to demonstrate that the compiler worked and that the move to a new standard was practical. After changing a small number of inconsistent actual arguments in subroutine calls, making a few other minor changes, and implementing parts of the HBOOK histogramming package [6], and of the ZEBRA package (this provides dynamic structuring and portable I/O facilities) [7], a program test example executed correctly. The average time per event (without the *-frnd* option) was 23 seconds, an increase of 25% compared with ftn.

4.6 New Fortran 90 Features

It was not within the scope of the initial evaluation to test f90 on a wide range of Fortran 90 features. Nevertheless, where FORTRAN 77 code had to be modified, this was often done with a new construct. In addition, a 1,500-line source code conversion program using only "modern" features, that is, none of those such as COMMON deprecated by Metcalf and Reid [2], worked correctly. However, it was a factor four slower than an

equivalent program running under ftn; NAG plans to improve the optimization of character handling in Release 2.

ACKNOWLEDGMENTS

I thank Miguel Marquina for providing me with the library code and its test programs, Malcolm Cohen (NAG) for his excellent support whilst performing the evaluation, and René Brun, Federico Carminati, and David Williams for their support and interest in this project.

REFERENCES

[1] ISO/IEC 1539 : 1991, ISO, Geneva, Switzerland.
[2] M. Metcalf and J. Reid, *Fortran 90 Explained,* Oxford and New York: Oxford University Press, 1990.
[3] M. Metcalf, "A derived-data type for data analysis," *Comput. Phys.*, vol. 5, no. 6, November/December, 1991.
[4] M. Metcalf, CERN/CN/91/11, 1991.
[5] R. Brun, F. Carminati et al., *GEANT User's Guide,* CERN Program Library, W999, 1992 (in preparation).
[6] R. Brun and M. Goossens, HBOOK Long Writeup, CERN Program Library, Y250, 1991.
[7] R. Brun and J. Zoll, *ZEBRA User's Guide,* CERN Program Library, Q100, 1987.