A Performance Benchmark Study of Fortran 90 Compilers

John K. Prentice
Quetzal Computational Associates
quetzal@aip.org

To appear in the *Fortran Journal*, vol 5, no 3, May/June 1993

Introduction

Fortran 90[1] represents a significant improvement over Fortran 77[2] and there is considerable interest in the language in the scientific programming community. There are currently three F90 compilers commercially available for a variety of computers. These products are the ParaSoft Corporation F90 compiler, the Numerical Algorithms Group Ltd[3] F90 compiler, and Pacific-Sierra Research Corporation VAST-90. The ParaSoft, NAG, and Pacific-Sierra Research F90 compilers translate F90 codes into an intermediate language and then compile that intermediate language using an optimizing compiler. This

[1] Fortran 90 will henceforth be abbreviated as F90.

[2] Fortran 77 will henceforth be abbreviated as F77.

[3] Numerical Algorithms Group Ltd will henceforth be abbreviated as NAG.

use of an intermediate language in the compilation process has led to concerns about whether these F90 compilers generate sufficiently efficient code to make them viable for either development or production work. To assess this performance question, Quetzal Computational Associates assembled a suite of F90 benchmark codes which exercise a significant cross section of new F90 features. This suite was then used to compare the execution speed of the code generated by each of these three F90 compilers, and when possible, to make comparisons between the execution speed of F90 and equivalent F77 codes. This article discusses the codes in this benchmark suite and reports the findings of this study.

The F90 compilers

The three F90 compilers used in this study are all available commercially and fully implement the F90 language, with the single exception that neither the ParaSoft or Pacific-Sierra Research compilers currently support F90 non-advancing I/O. The NAG F90 compiler, on the other hand, does support this feature. During the course of this project, I encountered a few bugs in all of the compilers. However, without exception, I found the user support for these compilers to be excellent and I received fixes to all my problems in an extremely timely manner.

The compilers all translate F90 into an intermediate language. The NAG F90 compiler translates F90 into C and then compiles the resulting code using an optimizing C compiler. The ParaSoft and Pacific-Sierra Research F90 compilers translate F90 into F77 and then compile the resulting code using an optimizing F77 compiler. For F90 features that cannot be emulated in either C or F77, a relocatable library is used that provides the necessary functionality. In addition to translating F90 to F77, the Pacific-Sierra Research product, VAST-90, has an extremely sophisticated capability to translate F77 codes to F90 and to tidy up F77 programs. The ParaSoft F90 and Pacific-Sierra Research VAST-90 also have the capability of compiling Connection Machine Fortran.

The addresses, telephone numbers, and e-mail addresses for these companies are given below.

- **ParaSoft F90:**
  ParaSoft Corporation
  2500 E. Foothill Blvd, Suite 104
  Pasadena, CA 91107
  USA

  phone: (818) 792-9941
  FAX: (818) 792-0819
  e-mail: info@parasoft.com

- **NAG F90:**
  NAG Ltd
  Wilkinson House
  Jordan Hill Road
  OXFORD
  United Kingdom OX2 8DR

  phone: +44 865 511245
  FAX: +44 865 310139
  Telex: 83354 NAG UK G
  e-mail: infodesk@vax.nag.co.uk

  -or-

  NAG Inc
  1400 Opus Place, suite 200
  Downers Grove, IL 60515-5702

USA

phone: (708) 971-2337
fax: (708) 971-2706
e-mail: infodesk@nag.com

- **Pacific-Sierra Research VAST-90:**
Pacific-Sierra Research Corporation
Computer Products Group
2901 28th Street, Suite 300
Santa Monica, CA 90405
USA

phone: (310) 314-2300
FAX: (310) 314-2323
e-mail: dave@psrv.com

The benchmark suite

The benchmark suite developed for this study currently contains eight codes. There are several other codes under development for the suite and they will be included in future reports of the benchmark results. The benchmark suite includes codes meant to test specific features of the language as well as full production level scientific computer codes. Each of the benchmark codes is briefly described below.

1. `kepler`: This code numerically solves the equations of motion for a planet orbiting the Sun. The equations of motion are solved numerically using a 12th order explicit symmetric multistep method. Both F77 and F90 versions of this code were developed. The F90 version differs

from the F77 version in the use of a significant amount of array syntax and the dot_product intrinsic.

2. pochhammer: This benchmark code calculates the values of Pochhammer's symbol

$$(z)_0 = 1, \tag{1}$$

$$(z)_n = z(z+1)(z+2)\dots(z+n-1), \tag{2}$$

for various values of z and n. The F77 version solves for the values using a do-loop:

```
      do 10 k=1,n-1
          poch=poch*(z+dble(k))
10    continue
```

The F90 version, on the other hand, uses recursion to solve for the values. The recursion is done using a F90 recursive function:

```
recursive function pochhammer (z,n) result(poch_value)
integer, parameter :: LONGreal = selected_real_kind(15,90)
real (kind=LONGreal), parameter :: one = 1.0
real (kind = LONGreal) :: poch_value,z
integer :: n
!
if (n == 0) then
    poch_value = one
else
    poch_value = pochhammer(z,n-1) *        &
&                              (z + real(n-1,LONGreal))
endif
!
end function pochhammer
```

3. gas dynamics: This benchmark code is a one and a half dimensional explicit finite difference code for modeling the fluid dynamics of a gas

jetting out of a combustion chamber into an evacuated cavity with an expanding radius. At the far end of the cavity are screens which impede the flow. A real gas equation of state is used and the drag caused by the screens is explicitly calculated. The expanding gas front is moved by free molecular flow, whereas the flow behind the front is calculated by numerically solving the nonlinear partial differential equations for conservation of mass, momentum, and energy using a HULL finite difference scheme. The F77 version of this code was a production code and was ported to F90 for this project. The conversion of the F77 code to F90 was done using Pacific-Sierra Research's VAST-90 and then some additional modifications were made by hand. The F90 version uses allocatable memory, array syntax, recursion, internal procedures, modules, optional argument passing, automatic arrays, and several new F90 intrinsics.

4. `overlap`: This benchmark code was derived from a production code used to generate mesh mappings in a solid dynamics domain decomposition code. It takes a collection of two dimensional finite difference meshes and computes how these meshes overlap each other. The output is a series of mappings which allow the domain decomposition code to communicate boundary information between two or more meshes. Both F77 and F90 versions of this code were developed. The F90 code was generated by running the F77 version through VAST-90. A few changes were then made to the VAST-90 generated code, in particular several external subroutines were changed to internal subroutines.

The major differences between the F77 and F90 versions is the use of internal procedures and CASE statements in the F90 version.

5. `static`: This code was contributed by Yuemin Qian of the Department of Computer Science at the University of Waterloo[4]. It tests the use of derived data types in F90 by defining a derived type, declaring an array using this derived type, and then performing some floating point arithmetic on this array. The essential part of the code is given below:

```
      integer :: i,j,n
      parameter( n = 2000 )
      type test
        double precision :: list(n)
      end type test
      type(test) :: rows(n)
!
      do i = 1,  n
         do j = 1, n
            rows(i)%list(j) = 123.0
         end do
      end do
!
      do i = 1,  n
         do j = 1, n
            rows(i)%list(j) = rows(i)%list(j)**3 +         &
&                                      10.0*rows(i)%list(j)
         end do
      end do
```

   Note that the arrays in this code are statically dimensioned.

6. `dynamic`: This code was also contributed by Yuemin Qian. This code again tests derived data types in F90 and is essentially identical to `static` except that now the arrays are allocatable. The essential part of the code is given below:

[4] yqian@neumann.waterloo.edu

```
      integer :: i,j,n
      type test
        integer, pointer :: list(:)
      end type test
      type(test) :: rows
      allocatable :: rows(:)
!
      n = 2000
      allocate(rows(n))
      do i = 1,  n
         allocate(rows(i)%list(n))
         do j = 1, n
            rows(i)%list(j) = 123.0
         end do
      end do
!
      do i = 1,  n
         do j = 1, n
            rows(i)%list(j) = rows(i)%list(j)**3 +        &
&                                    10.0*rows(i)%list(j)
         end do
      end do
```

7. `lu`: This benchmark code was contributed by Steve Moulton[5] of the Department of Computer Science at the University of Tennessee. It is a F90 version of a code to perform LU decompositions and is under development for use in the F90 version of LAPACK. The benchmark calculations were for an order 2000 dense matrix which was filled with random numbers. The code is a mixture of F77 and F90. In particular, many of the BLAS are in F77, whereas the drivers and most of the significant numerical routines are in F90. The F90 routines use numerous F90 features, including array syntax, allocatable memory, and modules. No equivalent F77 code is currently included in the benchmark suite.

[5]moulton@cs.utk.edu

8. `lfk`: This code contains a version of the Livermore Fortran Kernel (Livermore Loops). There are both F77 and F90 versions. The F77 version was obtained from NETLIB and modified slightly to correct some non-ANSI 77 problems. It has not been changed in any way that affects the results. The F90 conversion was done by VAST-90 and has not been significantly changed. The original F77 version of this code, obtained from NETLIB, was rather messy and no effort was made to revise it. Therefore, the F90 version is also rather messy and is a good example of a 'dusty deck' Fortran code. It should also be noted that, unlike the other codes in this benchmark suite, this one does significant amounts of I/O. The most significant difference between the F77 and F90 versions is the use of array syntax and some F90 intrinsics in the F90 version.

Benchmark results

The benchmark codes were compiled and run on a Sun Sparc 10 with 64 Mbytes of memory. The operating system was Solaris version 1.1 (also known as SunOS 4.1.3). The compilers used were Sun Fortran 77 version 1.4, Sun C version 1.1, NAG F90 version 2.0(78), ParaSoft F90 version 2.04, and Pacific-Sierra Research VAST-90 version 1.02E. One notable exception was the `lfk` benchmark where NAG F90 version 1.2(542) was used because of problems compiling the code with version 2.0(78)[6]. The C generated by the NAG F90

[6]NAG F90 version 2.0(78) is a beta version of the new version 2.0 compiler (due out in May 1993) and the compilation problem I encountered is being fixed in the final version

compiler was compiled using the -O optimization flag. The F77 generated by the ParaSoft and Pacific-Sierra Research F90 compilers was compiled using the -fast optimization flag. Each code was executed six times in a row on an otherwise idle system with no logged in users. The execution time for each code was determined using the UNIX `time` utility. Both CPU and system time were recorded. Tables **??** and **??** show the average CPU and system time for each benchmark.

The three F90 compilers used in this benchmark study are all fairly new products and development work on them is still being done. They were all able to correctly compile the codes in this benchmark suite, many of which are several thousand lines long and often involve sophisticated programming techniques. The code generated by these compilers was also correct, generating the same answers as the F77 versions and the same answers as each other. The performance of the code generated by the compilers varied, but for every benchmark where a comparison was made against a F77 code at least one F90 compiler generated code which compared favorably in speed against the F77 code. This is a good indication that the use of an intermediate language in the compilation process does not necessarily lead to a degradation in performance and in some cases may even led to an improvement in performance over equivalent F77 codes.

2.0 release. A major difference between version 2.0 and earlier versions is that version 2.0 does considerably more optimization of the generated C code. In all cases I have looked at, the code generated with version 2.0 runs considerably faster than the code generated with version 1.2. This should be remembered when looking at the **`lfk`** results in this report since it is likely that version 2.0 will run these benchmarks considerably faster.

It is reasonable to expect that some performance will be sacrificed when employing some of the newer F90 features, but this is an acceptable price if the F90 code is easier to maintain or allows for more sophisticated programming constructs, as is the case for other languages such as C and C++. What is significant in these benchmark results is that these F90 codes did employ a considerable number of the new F90 features and in general did not experience dramatic performance degradations as compared to F77. Further, the companies developing these compilers are all actively working on improved optimization and the performance of the code they generate will most certainly continue to improve.

Conclusion

In the course of this benchmark study, all three of these F90 compilers have proven themselves to be solid products capable of generating correct and efficient executable code for everything from simple test programs to sophisticated physics codes. They also have proven themselves to be capable of generating code whose execution speed is comparable to F77 codes. The only exception to this was the recursive benchmark, where a loss of performance is virtually inevitable. Even in this benchmark, however, one of the compilers was able to come surprisingly close to the speed of the nonrecursive F77 program. It is thus clear that products are on the market which now make it possible to employ Fortran 90 as a programming language for scientific computing.

It is my intention to continue adding codes to this F90 benchmark suite and to release periodic performance updates as new versions of these compilers become available and other companies develop F90 compilers. I would therefore welcome contributions of F90 codes for inclusion in this benchmark suite. It is essential that such codes adhere to the ISO/ANSI F90 standard so that they can be reliably compiled by all of these compilers. The codes in the benchmark suite are available free to anyone interested, so any codes contributed to the suite must be donated with that understanding. If anyone is interested in donating codes to the benchmark suite, please contact me at quetzal@aip.org. If anyone is interested in receiving a copy of the benchmark suite, it can be obtained via anonymous ftp from ftp.cs.unm.edu. Log in as anonymous and give a valid e-mail address as your password. The benchmark suite is located in the directory pub/smith-quetzal. That directory also has a copy of the benchmark results. The results will be updated periodically as new versions of the compilers are released or new codes are added to the benchmark suite.

Benchmark	*Sun F77*	*PS F90*	*NAG F90*	*PSR VAST-90*
kepler	47.0 ± 0.05	57.3 ± 0.0	74.9 ± 0.0	57.3 ± 0.05
pochhammer	14.7 ± 0.05	336.3 ± 0.0	340.2 ± 0.06	35.0 ± 0.0
gas dynamics	72.6 ± 0.1	72.1 ± 0.1	100.2 ± 0.04	69.5 ± 0.2
overlap	18.3 ± 0.1	27.5 ± 0.1	21.3 ± 0.1	18.5 ± 0.1
static	N/A	8.0 ± 0.1	7.1 ± 0.1	8.2 ± 0.1
dynamic	N/A	20.8 ± 0.1	31.4 ± 0.2	22.3 ± 0.1
lu	N/A	2589.8 ± 0.2	2345.6 ± 2.2	2959.5 ± 0.1
lfk	405.7 ± 24.5	387.5 ± 44.2	1939.1 ± 32.6	367.7 ± 31.7

Table 1: Average CPU time required for the benchmark codes. Each code was run six times in a row on an otherwise idle system with no logged in users. The error shown after the average time is the standard deviation for the six runs. N/A in the Sun F77 column means that no equivalent F77 code existed for that benchmark. Note that for the Pochhammer benchmark, the F90 codes use recursion, while the F77 code did not. Also note that the NAG F90 time shown for the lfk benchmark was with version 1.2 of the compiler, while the other NAG times were with version 2.0.

Benchmark	*Sun F77*	*PS F90*	*NAG F90*	*PSR VAST-90*
kepler	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0	0.0 ± 0.0
pochhammer	0.0 ± 0.0	0.1 ± 0.04	0.0 ± 0.0	0.0 ± 0.0
gas dynamics	1.5 ± 0.2	1.6 ± 0.1	0.2 ± 0.08	1.8 ± 0.2
overlap	0.2 ± 0.04	0.2 ± 0.1	0.2 ± 0.1	0.2 ± 0.1
static	N/A	2.4 ± 0.1	2.4 ± 0.1	2.4 ± 0.1
dynamic	N/A	1.7 ± 0.1	2.2 ± 0.3	1.8 ± 0.1
lu	N/A	2.4 ± 0.1	2.2 ± 0.2	1.5 ± 0.1
lfk	185.4 ± 37.9	161.4 ± 68.1	212.0 ± 58.2	112.6 ± 46.7

Table 2: Average system time required for the benchmark codes. Each code was run six times in a row on an otherwise idle system with no logged in users. The error shown after the average time is the standard deviation for the six runs. N/A in the Sun F77 column means that no equivalent F77 code existed for that benchmark. Note that for the Pochhammer benchmark, the F90 codes use recursion, while the F77 code did not. Also note that the NAG F90 time shown for the lfk benchmark was with version 1.2 of the compiler, while the other NAG times were with version 2.0.