Survey of FORTRAN Compiler Options and Their Impact on COSY INFINITY

Ravi Jagasia and Alexander Wittig

MSUHEP - 090422

Abstract

Compiler options are different not only for different compilers, but even among versions of a given compiler. Several options control the mechanism used in generating the code, including optimizations for speed, size and standard compliance. Many of those options affect the speed, accuracy and correctness of the resulting executable.

In this survey we present the options relevant for building COSY INFINITY with the GNU and Intel Fortran compilers. We build executables with several options on different hardware, comparing the accuracy of the result as well as the runtime using three different test cases. As a result of our testing, we suggest a set of command line options to use in order to get correct results, while minimizing the impact on the speed of the resulting executable.

1 Compilers

Compiler options are different not only for different compilers, but even among versions of a given compiler. To this extent, we outline the following options that we either require or would like to have working from a performance perspective.

1.1 Intel Fortran Compiler Version 9.2 and 11.0

This compiler is invoked by the command:

```
ifort
```

Detailed documentation for the options of this compiler can be found in [2] for Intel Fortran 9.2 and for Intel Fortran 11. A quick overview of the optimization options of Intel Fortran 11 can be found in [3].

The optimization flags used in our tests are (descriptions taken from `ifort --help`):

`-O0`

Disable optimizations.

```
-O1
```

Optimize for maximum speed, but disable some optimizations which increase code size for a small speed benefit.

```
-O
-O2
```

Enable optimizations (default).

```
-O3
```

Enable -O2 plus more aggressive optimizations that may not improve performance for all programs.

```
-fp-model strict
```

Controls the semantics of floating-point calculations.

- strict - enables -fp-model precise -fp-model except, disables contractions, enables property to allow for modification of the floating point environment
- except - enable floating point semantics
- precise - allows value-safe optimizations

The following option only works on Intel Fortran version 11:

```
-m32
```

Tells the compiler to generate code for IA-32 architecture.

1.2 GNU Fortran Compiler, GCC Version 4.3.2

This compiler is invoked by the command:

```
gfortran
```

A listing of the primary flag optimizations to be tested will be given below. The options that these flags invoke can be found on the online documents for GCC. However, options particularly important to the compilation of COSY will be given with descriptions.

The optimization flags used in our tests are (taken from [4] and [5]):

```
-O0
```

Reduce compilation time and make debugging produce the expected results. This is the default.

```
-O
-O1
```

Optimize. Optimizing compilation takes somewhat more time, and a lot more memory for a large function. With this, the compiler tries to reduce code size and execution time, without performing any optimizations that take a great deal of compilation time.

`-O2`

Optimize even more. GCC performs nearly all supported optimizations that do not involve a space-speed tradeoff. As compared to `-O`, this option increases both compilation time and the performance of the generated code.

`-O3`

Optimize yet more. `-O3` turns on all optimizations specified by `-O2` and also turns on the `-finline-functions`, `-funswitch-loops`, `-fpredictive-commoning`, `-fgcse-after-reload` and `-ftree-vectorize` options.

`-fno-range-check`

Disable range checking on results of simplification of constant expressions during compilation. For example, GNU Fortran will give an error at compile time when simplifing "a = 1. / 0". With this option, no error will be given and "a" will be assigned the value "+Infinity". If an expression evaluates to a value outside of the relevant range of ["-HUGE()":"HUGE()"], then the expression will be replaced by "-Inf" or "+Inf" as appropraite. Similarly, "DATA i/Z'FFFFFFFF'/" will result in an integer overflow on most systems, but with `-fno-range-check` the value will "wrap around" and "i" will be initialized to -1 instead.

`-ffloat-store`

Do not store floating point variables in registers, and inhibit other options that might change whether a floating point value is taken from a register or memory.

This option prevents undesirable excess precision on machines such as the 68000 where the floating registers (of the 68881) keep more precision than a double is supposed to have. Similarly for the x86 architecture. For most programs, the excess precision does only good, but a few programs rely on the precise definition of IEEE floating point. Use -ffloat-store for such programs, after modifying them to store all pertinent intermediate computations into variables.

2 Testing

In the following we describe the test that were run, the machines and the environments those tests were run in, as well as the results of the tests.

2.1 Machines

We ran these tests on several different machines in order to cover a broad spectrum of different processors. Also, not all compilers are available on all machines.

Vivaldi Pentium 4 2GHz, 512 MB RAM
Ubuntu Linux 8.10
Intel Fortran 11, gfortran 4.3.2

bt.pa.msu.edu Pentium 4 2GHz, 512 MB RAM
FreeBSD 7.2
gfortran 4.3.4 (prerelease)

MacMini Core Duo 1.66GHz, 2 GB RAM
Mac OS X 10.4.11
Intel Fortran 9.2, gfortran 4.2.3

MacBook Core 2 Duo 2GHz, 2GB RAM
Mac OS X 10.5.6
Intel Fortran 11

Due to limitations of the FreeBSD operating system, bt.pa.msu.edu is not able to run executables with the default memory size of COSY. In order to create working executables, we have scaled down the stack size by a factor of 5 on this machine. Also note that, since bt.pa.msu.edu is our web server, it is running several other processes in parallel to COSY as we did not want to disrupt service for our speed tests. This may impact the results.

From here on we will reference the machines by the names given in the above list.

2.2 Compiler Options

We selected several compiler options based on the manual for each compiler. We tried to select the default options, options that seem to produce IEEE 754 compliant output, and options that turn off all optimization.

2.2.1 Intel Fortran compiler options

We compiled the sources with the following 6 different settings, which work on Intel Fortran 9.2 as well as Intel Fortran 11.

```
-O0

-O1

-O2

-O3
```

`-O0 -fp-model strict`

`-O2 -fp-model strict`

Specifically on the 64bit capable MacBook, we also ran each of the above command lines with an added `-m32`. This is because we found odd behavior on 64bit machines in conjuction with the `-O0` optimization. Also in some cases there are noticeable differences in the execution speed between 64bit and 32bit executables.

2.2.2 GNU Fortran compiler options

We compiled the sources with the following 6 different settings.

`-O0 [-fno-range-check]`

`-O1 [-fno-range-check]`

`-O2 [-fno-range-check]`

`-O3 [-fno-range-check]`

`-O0 -ffloat-store [-fno-range-check]`

`-O2 -ffloat-store [-fno-range-check]`

In the case of the high precision COSY code, the additional flag `-fno-range-check` is required to compile the code for the built in tests. We did not use that flag for the standard COSY sources in the demo and VI tests.

2.3 Consistency

In this test we wanted to determine the compliance with IEEE 754. This testing is done using the Alex's hpCOSY code, compiled without graphics support and MPFR/MPFI bindings. This code relies heavily on strict IEEE 754 adherence. Since in the long run this code will be included in the main distribution version of COSY, it will be useful to have that information available.

Of course passing in this test also should ensure that the currently used built in intervals work.

2.3.1 Test Case

The test code for the high precision intervals is derived from the code in [1]. It is an analytic expression for the transfer map of x and a in a homogenous magnetic segment. The problem calculates 12 consecutive 30 degree segments. This, of course, adds up to the identity.

To speed test the intervals, the above computation is performed 5000 times. All computations are done with 44 valid digits, that is 3 limbs.

The code for this problem is in the file hptest.fox of the supporting material (see Appendix).

2.3.2 Test Results

We documented the results of our tests in table 1. If the compiled version ran and reproduced the reference results bit by bit (see Appendix), we put down the time for the timing test in seconds. Note that the times are only comparable between different compilers on the same machine. If the compiled binary refused to run due to failure in the internal testing mechanism, we put don "SF" (soft failure). If the compiled binary did run, but produced output different from the reference output, we put down "HF" (hard failure). If a compiler or test was not available on a given machine, we put down "-". We only ran the special cases of generating 32bit executables by explicitly specifying `-m32` on the 64bit machines. On purely 32bit architectures, this switch has no effect.

	Vivaldi	bt.pa.msu.edu	MacMini	MacBook
`gfortran -fno-range-check ...`				
`-O0`	HF	113	118	-
`-O1`	SF	113	89.5	-
`-O2`	SF	105	86.6	-
`-O3`	SF	109	86.5	-
`-O0 -ffloat-store`	HF (177)	394	139	-
`-O2 -ffloat-store`	HF (131)	283	107	-
`ifort ...` (Version 9.2)				
`-O0`	-	-	68.76	-
`-O1`	-	-	SF	-
`-O2`	-	-	SF	-
`-O3`	-	-	SF	-
`-O0 -fp-model strict`	-	-	68.57	-
`-O2 -fp-model strict`	-	-	40.35	-
`ifort ...` (Version 11.0)				
`-O0`	90.3	-	-	HF
`-O0 -m32`	-	-	-	59.28
`-O1`	SF	-	-	SF
`-O1 -m32`	-	-	-	SF
`-O2`	SF	-	-	SF
`-O2 -m32`	-	-	-	SF
`-O3`	SF	-	-	SF
`-O3 -m32`	-	-	-	SF
`-O0 -fp-model strict`	87.8	-	-	HF (54.55)
`-O0 -m32 -fp-model strict`	-	-	-	58.71
`-O2 -fp-model strict`	50.2	-	-	22.66
`-O2 -m32 -fp-model strict`	-	-	-	25.31

Table 1: Results of the consistency tests

Remarks Code compiled for 64bit machines with Intel Fortran and `-O0` on Mac OS X (the MacBook in our test) produces wrong results. Note that even with `-fp-model strict` the results did not match the reference results. In the case of only `-O0`, the results were not rigorous (i.e. the intervals did not include the correct result). In the case of `-O0 -fp-model strict`, the results did include the mathematically correct result, but had smaller error terms than the reference output. In both cases the errors were not detected by the built in tests.
Turning off the 64bit support, and producing 32bit executables resulted in the same behavior as on all other tested platforms.

Running gfortran on Vivaldi produced no correct output at all. In most cases, the internal tests caught errors before the test was run. In the case of `-O0`, however, the tests passed, but the resulting output was mathematically incorrect. With `-O0 -ffloat-store`, the result only differed in the size of the error terms from the reference output. The results produced were mathematically valid. Using `-O2 -ffloat-store`, produced the exact same results.

2.4 Speed

In this test we want to measure the speed impact of the different compiler options. The testing is done using the stock COSY sources given on the web, compiled with PGPLOT on Vivaldi, and without graphics support on other machines.

2.4.1 Test Case

In this test case we test the old COSY Verified Integrator as it applies to an object moving in the gravitational field of 10 bodies. The position of the bodies themselves are calculated via conversion of ellipse orbital parameters. The object is far from the asymptotic regions of the pointwise gravitational bodies and the integration time is cut down to make the test run in a manageable time frame. Regardless of this, it still requires many Taylor Model operations.

2.4.2 Test Results

We documented the results of our tests in the following table. If the compiled version ran, we put down the time for the timing test in seconds. Note that the times are only comparable between different compilers on the same machine. If a compiler was not available on a given machine, we put down "-". For the integrator test, the error in TMNOT, Variable exhausted, occurs sometimes and will be marked as an "Err".

	Vivaldi	bt.pa.msu.edu	MacMini	MacBook
`gfortran ...`				
`-O0`	162.13	248.78	107.43	-
`-O1`	60.12	61.71	55.33	-
`-O2`	Err	Err	50.80	-
`-O3`	Err	Err	50.48	-
`-O0 -ffloat-store`	202.16	509.36	118.93	-
`-O2 -ffloat-store`	111.73	319.81	70.61	-
`ifort ...` (Version 9.2)				
`-O0`	-	-	108.38	-
`-O1`	-	-	39.59	-
`-O2`	-	-	39.84	-
`-O3`	-	-	40.81	-
`-O0 -fp-model strict`	-	-	108.37	-
`-O2 -fp-model strict`	-	-	47.98	-
`ifort ...` (Version 11.0)				
`-O0`	109.85	-	-	78.36
`-O0 -m32`	-	-	-	80.00
`-O1`	43.81	-	-	29.41
`-O1 -m32`	-	-	-	23.08
`-O2`	43.97	-	-	30.38
`-O2 -m32`	-	-	-	24.40
`-O3`	43.87	-	-	28.82
`-O3 -m32`	-	-	-	24.98
`-O0 -fp-model strict`	211.99	-	-	77.41
`-O0 -fp-model strict -m32`	-	-	-	81.19
`-O2 -fp-model strict`	56.02	-	-	36.06
`-O2 -fp-model strict -m32`	-	-	-	31.02

Table 2: Results of the speed tests

2.5 Consistency of Non-High Precision Code

2.5.1 Test Case

In this test we examine the consistency of output being created from the standard package of COSY with it's included demo.fox. The file demo.fox has been modified in order to do one runthrough of all demos in ASCII output which is then stored in a file to be cross checked with other compiler options using the `diff` program. The modifications were primarily done to remove any user interactivity and for the program to terminate after completing all the demos.

2.5.2 Test Results

We make a table corresponding to identical program output. It should be noted that the only differences in output observed are numerical in nature, rather than anything structural regarding output such as new lines or spaces. Two entries in the table have the same letter assigned in the case that they have identical output and different letters otherwise. In the case that nothing matches, a slash / is given. The letters are given in order of their occurance.

For purposes of completeness, we again note the speed of completion of all the demos in `demo.fox` (in seconds). However, since the test produces close to ten thousand lines of text and examines another five thousand lines of data from `SYSCA.DAT`, it is likely that the speed reported here also includes some dependence of read write speeds, which is hardware dependent.

Primary differences occur in demo files where fit loops are being employed. e.g. the value of the objective function as it is being fitted for the design of a storage ring, or in 'Fitting a Four Cell Third Order Achromat'.

	Vivaldi	bt.pa.msu.edu	MacMini	MacBook
`gfortran ...`				
`-O0`	/	D	C	-
`-O1`	/	/	C	-
`-O2`	H	D	C	-
`-O3`	H	D	C	-
`-O0 -ffloat-store`	E	G	C	-
`-O2 -ffloat-store`	E	G	C	-
`ifort ...` (Version 9.2)				
`-O0`	-	-	A	-
`-O1`	-	-	F	-
`-O2`	-	-	B	-
`-O3`	-	-	B	-
`-O0 -fp-model strict`	-	-	A	-
`-O2 -fp-model strict`	-	-	A	-
`ifort ...` (Version 11.0)				
`-O0`	A	-	-	/
`-O0 -m32`	-	-	-	A
`-O1`	A	-	-	A
`-O1 -m32`	-	-	-	A
`-O2`	B	-	-	B
`-O2 -m32`	-	-	-	B
`-O3`	F	-	-	B
`-O3 -m32`	-	-	-	B
`-O0 -fp-model strict`	A	-	-	A
`-O0 -fp-model strict -m32`	-	-	-	A
`-O2 -fp-model strict`	A	-	-	A
`-O2 -fp-model strict -m32`	-	-	-	A

Table 3: Results of the demo consistency tests

	Vivaldi	bt.pa.msu.edu	MacMini	MacBook
`gfortran ...`				
`-O0`	13.90	23.06	10.99	-
`-O1`	9.04	13.69	5.86	-
`-O2`	9.14	13.58	5.84	-
`-O3`	8.41	13.14	5.70	-
`-O0 -ffloat-store`	16.02	33.92	11.76	-
`-O2 -ffloat-store`	11.26	27.36	6.76	-
`ifort ...` (Version 9.2)				
`-O0`	-	-	11.60	-
`-O1`	-	-	5.28	-
`-O2`	-	-	4.51	-
`-O3`	-	-	4.66	-
`-O0 -fp-model strict`	-	-	11.60	-
`-O2 -fp-model strict`	-	-	5.50	-
`ifort ...` (Version 11.0)				
`-O0`	14.25	-	-	8.14
`-O0 -m32`	-	-	-	8.68
`-O1`	7.24	-	-	3.58
`-O1 -m32`	-	-	-	3.77
`-O2`	6.78	-	-	3.19
`-O2 -m32`	-	-	-	3.14
`-O3`	6.79	-	-	3.11
`-O3 -m32`	-	-	-	3.40
`-O0 -fp-model strict`	13.53	-	-	8.18
`-O0 -fp-model strict -m32`	-	-	-	8.71
`-O2 -fp-model strict`	8.34	-	-	3.68
`-O2 -fp-model strict -m32`	-	-	-	4.00

Table 4: Results of the demo speed tests

Appendix

2.6 Reference Results

These reference results for the consistency test were generated using ifort 9.2 with all optimizations turned off ("`ifort -O0`") on the MacMini computer.

```
Testing precalculated constants:
IDELN2:
  0.6931471805599453      |    6243314768165359b-53
  0.2319046813846300E-16  |    7525737178955839b-108
  0.5707708438416198E-33  |    1668365045630537b-161
+-0.1202192769595098E-45  | +- 6181917753585427b-205

IDEPI:
   3.141592653589793      |     884279719003555b-48
  0.1224646799147353E-15  |    4967757600021511b-105
  -.2994769809718339E-32  |   -8753721960665019b-161
+-0.2949765617711364E-46  | +- 6067315958716815b-207



Calculating map of a 90 degree homogenous dipole (r0=1m)
xi:
  0.2500000000000000      |                   1b-2
+- 0.000000000000000      | +-                      0

ai:
  0.5000000000000000      |                   1b-1
+- 0.000000000000000      | +-                      0



Calculating map of 12 30 degree homogenous dipoles (r0=1m)
xf:
  0.2500000000000000      |                   1b-2
+-0.4078579247830823E-29  | +-  727642852623119b-147

af:
  0.5000000000000000      |                   1b-1
+-0.2595024536356237E-29  | +- 7407485564059585b-151


Difference xf:
   0.000000000000000      |                         0
+-0.4078579247830829E-29  | +-   45477678288945b-143
```

```
Difference af:
   0.000000000000000      |                          0
+-0.2595024536356240E-29  | +- 7407485564059595b-151
```

2.7 Source Code

The complete source code, makefiles, and test cases used in these test is temporarily available at `http://bt.pa.msu.edu/~alex/tests.zip`. It includes the source code and Makefile to build hpCOSY, the official sources downloaded from the COSY website, and the fox files including the test code.

References

[1] A. Wittig and M. Berz. Computation of high-order maps to multiple machine precision. *International Journal Modern Physics A*, in print, 2008.

[2] Intel Fortran Compiler User and Reference Guides, `http://www.intel.com/software/products/compilers/docs/flin/main_for/index.htm`.

[3] Quick-Reference Guide to Optimization with Intel Compilers version 11, `http://www.intel.com/software/products/compilers/docs/qr_guide.htm`.

[4] Using the GNU Compiler Collection, `http://gcc.gnu.org/onlinedocs/gcc.pdf`.

[5] gfortran(1): GNU Fortran 95 compiler - Linux man page, `http://linux.die.net/man/1/gfortran`