Learning FORTRAN In the Modern Era

I've recently come to maintain a large amount of scientific calculation-intensive FORTRAN code. I'm having difficulties getting a handle on all of the, say, nuances, of a forty year old language, despite google & two introductory level books. The code is rife with "performance enhancing improvements". Does anyone have any guides or practical advice for **de**-optimizing FORTRAN into CS 101 levels? Does anyone have knowledge of how FORTRAN code optimization operated? Are there any typical FORTRAN 'gotchas' that might not occur to a Java/C++/.NET raised developer taking over a FORTRAN 77/90 codebase?

fortran

asked Aug 28 '08 at 4:36

David J. Sokol
1,755 1 18 25

2 Would this text perhaps be interesting to you? fortranrefactoring.com.ar/papers/… – Rook Sep 27 '11 at 17:54

1 @DavidSokol I enjoyed listening to you on the dinosaur TDL podcast, especially with this question for context :) Note, the podcast doesn't mention this, I just remembered seeing it after listening. – Tim Post ♦ Mar 9 '12 at 8:14

10 Answers

You kind of have to get a "feel" for what programmers had to do back in the day. The vast majority of the code I work with is older than I am and ran on machines that were "new" when my parents were in high school.

Common FORTRAN-isms I deal with, that hurt readability are:

- Common blocks
- Implicit variables
- Two or three DO loops with shared CONTINUE statements
- GOTO's in place of DO loops
- Arithmetic IF statements
- Computed GOTO's
- Equivalence REAL/INTEGER/other in some common block

Strategies for solving these involve:

1. Get Spag / plusFORT, worth the money, it solves a lot of them automatically and Bug Free(tm)
2. Move to Fortran 90 if at all possible, if not move to free-format Fortran 77
3. Add IMPLICIT NONE to each subroutine and then fix every compile error, time consuming but ultimately necessary, some programs can do this for you automatically (or you can script it)
4. Moving all COMMON blocks to MODULEs, low hanging fruit, worth it
5. Convert arithmetic IF statements to IF..ELSEIF..ELSE blocks
6. Convert computed GOTOs to SELECT CASE blocks
7. Convert all DO loops to the newer F90 syntax

```
myloop: do ii = 1, nloops
    ! do something
enddo myloop
```

8. Convert equivalenced common block members to either ALLOCATABLE memory allocated in a module, or to their true character routines if it is Hollerith being stored in a REAL

If you had more specific questions as to how to accomplish some readability tasks, I can give advice. I have a code base of a few hundred thousand lines of Fortran which was written over the span of 40 years that I am in some way responsible for, so I've probably run across any "problems" you may have found.

answered Sep 15 '08 at 22:29

user7116
42.6k 11 87 130

1 Also, a nice text regarding fortran refactoring. fortranrefactoring.com.ar/papers/… – Rook Sep 27 '11 at 18:00

It is very difficult if the code is not organised very well and pretty much obfuscated by optimisation details. Migrating to fortran 90 style would help you. In addition the later versions of fortran 2008 provide functionality for optimisation. – Zeus yesterday

Legacy Fortran Soapbox

I helped maintain/improve a legacy Fortran code base for quite a while and for the most part think **sixlettervariables** is on the money. That advice though, tends to the technical; a tougher row to hoe is in implementing "good practices".

- Establish a required coding style and coding guidelines.
- Require a code review (of more than just the coder!) for anything submitted to the code base. (Version control should be tied to this process.)
- Start building and running unit tests; ditto benchmark or regression tests.

These might sound like obvious things these days, but at the risk of over-generalizing, I claim that most Fortran code shops have an entrenched culture, some started before the term "software engineering" even existed, and that over time what comes to dominate is "Get it done now". (This is not unique to Fortran shops by any means.)

Embracing Gotchas

But what to do with an already existing, grotty old legacy code base? I agree with Joel Spolsky on rewriting, don't. However, in my opinion **sixlettervariables** does point to the allowable exception: *Use software tools to transition to better Fortran constructs.* A lot can be caught/corrected by code analyzers (FORCHECK) and code rewriters (plusFORT). If you have to do it by hand, make sure you have a pressing reason. (I wish I had on hand a reference to the number of software bugs that came from fixing software bugs, it is humbling. I think some such statistic is in Expert C Programming.)

Probably the best offense in winning the game of Fortran gotchas is having the best defense: Knowing the language fairly well. To further that end, I recommend ... books!

Fortran Dead Tree Library

I have had only modest success as a "QA nag" over the years, but I have found that education does work, some times inadvertently, and that one of the most influential things is a reference book that someone has on hand. I love and highly recommend

Fortran 90/95 for Scientists and Engineers, by Stephen J. Chapman

The book is even good with Fortran 77 in that it specifically identifies the constructs that shouldn't be used and gives the better alternatives. However, it is actually a textbook and can run out of steam when you really want to know the nitty-gritty of Fortran 95, which is why I recommend

Fortran 90/95 Explained, by Michael Metcalf & John K. Reid

as your go-to reference (sic) for Fortran 95. Be warned that it is not the most lucid writing, but the veil will lift when you really want to get the most out of a new Fortran 95 feature.

For focusing on the issues of going from Fortran 77 to Fortran 90, I enjoyed

Migrating to Fortran 90, by Jim Kerrigan

but the book is now out-of-print. (I just don't understand O'Reilly's use of Safari, why isn't every one of their out-of-print books available?)

Lastly, as to the heir to the wonderful, wonderful classic, Software Tools, I nominate

Classical FORTRAN, by Michael Kupferschmid

This book not only shows what one can do with "only" Fortran 77, but it also talks about some of the more subtle issues that arise (e.g., should or should not one use the EXTERNAL

declaration). This book doesn't exactly cover the same space as "Software Tools" but they are two of the three Fortran programming books that I would tag as "fun".... (here's the third).

Miscellaneous Advice that applies to *almost* every Fortran compiler

- There is a compiler option to enforce IMPLICIT NONE behavior, which you can use to identify problem routines without modifying them with the IMPLICIT NONE declaration first. This piece of advice won't seem meaningful until after the first time a build bombs because of an IMPLICIT NONE command inserted into a legacy routine. (What? Your code review didn't catch this? ;-)
- There is a compiler option for array bounds checking, which can be useful when debugging Fortran 77 code.
- Fortran 90 compilers should be able to compile almost all Fortran 77 code and even older Fortran code. Turn on the reporting options on your Fortran 90 compiler, run your legacy code through it and you will have a decent start on syntax checking. Some commercial Fortran 77 compilers are actually Fortran 90 compilers that are running in Fortran 77 mode, so this might be relatively trivial option twiddling for whatever build scripts you have.

edited Oct 3 '08 at 22:07

answered Oct 3 '08 at 19:43

jaredor

1,375 1 8 8

There's something in the original question that I would caution about. You say the code is rife with "performance enhancing improvements". Since Fortran problems are generally of a scientific and mathematical nature, do not assume these performance tricks are there to improve the compilation. It's probably not about the language. In Fortran, the solution is seldom about efficiency of the code itself but of the underlying mathematics to solve the end problem. The tricks may make the compilation slower, may even make the logic appear messy, but the intent is to make the solution faster. Unless you know exactly what it is doing and why, leave it alone.

Even simple refactoring, like changing dumb looking variable names can be a big pitfall. Historically standard mathematical equations in a given field of science will have used a particular shorthand since the days of Maxwell. So to see an array named B(:) in electromagnetics tells all Emag engineers exactly what is being solved for. Change that at your peril. Moral, get to know the standard nomenclature of the science before renaming too.

answered Sep 22 '08 at 18:46

SumoRunner

366 1 4

As someone with experience in both FORTRAN (77 flavor although it has been a while since I used it seriously) and C/C++ the item to watch out for that immediately jumps to mind are arrays. FORTRAN arrays start with an index of 1 instead of 0 as they do in C/C++/Java. Also, memory arrangement is reversed. So incrementing the first index gives you sequential memory locations.

My wife still uses FORTRAN regularly and has some C++ code she needs to work with now that I'm about to start helping her with. As issues come up during her conversion I'll try to point them out. Maybe they will help.

answered Aug 28 '08 at 13:14

dagorym

2,212 2 15 20

8 Fortran arrays start with index 1 by default but can be declared to start with any value. – M. S. B. Jun 7 '10 at 21:33

I have used Fortran starting with the '66 version since 1967 (on an IBM 7090 with 32k words of memory). I then used PL/1 for some time, but later went back to Fortran 95 because it is ideally suited for the matrix/complex-number problems we have. I would like to add to the cnsoderations that much of the convoluted structure of old codes is simply due to the small amount of memory available, forcing such thing like reusing a few lines of code via computed or assigned GOTOs. Another problem is optimization by defining auxiliary variables for every

repeated subexpression - compilers simply did not optimize for that. In addition, it was not allowed to write DO i=1,n+1 ; you had to write n1=n+1; DO i=1,n1. In consequence old codes are overwhelmed with superfluous variables. When I rewrote a code in Fortran 95, only 10% of the variables survived. If you want to make the code more legible, I highly recommend looking for variables that can easily be eliminated.

Another thing I might mention is that for many years complex arithmetic and multidimensional arrays were highly inefficient. That is why you often find code rewritten to do complex caculations using only real variables, and matrices addressed with a single linear index.

answered Jan 11 '12 at 11:16

J . A. Maruhn
41 1

Well, in one sense, you're lucky, 'cause Fortran doesn't have much in the way of subtle flow-of-control constructs or inheritance or the like. On the other, it's got some truly amazing gotchas, like the arithmetically-calculated branch-to-numeric-label stuff, the implicitly-typed variables which don't require declaration, the lack of true keywords.

I don't know about the "performance enhancing improvements". I'd guess most of them are probably ineffective, as a couple of decades of compiler technology have made most hinting unnecessary. Unfortunately, you'll probably have to leave things the way they are, unless you're planning to do a massive rewrite.

Anyway, the core scientific calculation code should be fairly readable. Any programming language using infix arithmetic would be good preparation for reading Fortran's arithmetic and assignment code.

answered Aug 28 '08 at 5:01

tuxedo
279 1 7

Could you explain what you have to do in maintaining the code? Do you really have to modify the code? If you can get away by modifying just the interface to that code instead of the code itself, that would be the best.

The inherent problem when dealing with a large scientific code (not just FORTRAN) is that the underlying mathematics and the implementation are both complex. Almost by default, the implementation *has to* include code optimization, in order to run within reasonable time frame. This is compounded by the fact that a lot of code in this field is created by scientists / engineers that are expert in their field, but not in software development. Let's just say that "easy to understand" is not the first priority to them (I was one of them, still learning to be a better software developer).

Due to the nature of the problem, I don't think a general question and answer is enough to be helpful. I suggest you post a series of specific questions with code snippet attached. Perhaps starting with the one that gives you the most headache?

answered Sep 17 '08 at 7:58

Lahur
151 2

I loved FORTRAN, I used to teach and code in it. Just wanted to throw that in. Haven't touched it in years.
I started out in COBOL, when I moved to FORTRAN I felt I was freed. Everything is relative, yeah? I'd second what has been said above - recognise that this is a PROCEDURAL language - no subtelties - so take it as you see it.
Probably frustrate you to start with.

answered Nov 21 '08 at 15:01

jsfain
29 2 4

I went through that phase too. In fact I remember "best practices" that minimized the gotchas. But then I went on to Lisp, Pascal, C, C++. I still have to work with some Fortran. The real problem is that most of it is written with very little programmer discipline. People are still teaching Fortran, but they're not teaching discipline. – Mike Dunlavey Sep 23 '09 at 20:44

I started on Fortran IV (WATFIV) on punch cards, and my early working years were VS FORTRAN v1 (IBM, Fortran 77 level). Lots of good advice in this thread.

I would add that you have to distinguish between things done to get the beast to run at all, versus things that "optimize" the code, versus things that are more readable and maintainable. I can remember dealing with VAX overlays in trying to get DOE simulation code to run on IBM with virtual memory (they had to be removed and the whole thing turned into one address space).

I would certainly start by carefully restructuring FORTRAN IV control structures to at least FORTRAN 77 level, with proper indentation and commenting. Try to get rid of primitive control structures like ASSIGN and COMPUTED GOTO and arithmetic IF, and of course, as many GOTOs as you can (using IF-THEN-ELSE-ENDIF). Definitely use IMPLICIT NONE in every routine, to force you to properly declare all variables (you wouldn't believe how many bugs I caught in other people's code -- typos in variable names). Watch out for "premature optimizations" that you're better off letting the compiler handle by itself.

If this code is to continue to live and be maintainable, you owe it to yourself and your successors to make it readable and understandable. *Just be certain of what you are doing as you change the code!* FORTRAN has lots of peculiar constructs that can easily trip up someone coming from the C side of the programming world. Remember than FORTRAN dates back to the mid-late '50s, when there was no such thing as a science of language and compiler design, just *ad hoc* hacking together of something (sorry, Dr. B!).

answered May 9 '14 at 15:21

Phil Perry
1,705 3 17

Here's another one that has bit me from time to time. When you are working on FORTRAN code make sure you skip all six initial columns. Every once and a while, I'll only get the code indented five spaces and nothing works. At first glance everything seems okay and then I finally realize that all the lines are starting in column 6 instead of column 7.

For anyone not familiar with FORTRAN, the first 5 columns are for line numbers (=labels), the 6th column is for a continuation character in case you have a line longer than 80 characters (just put something here and the compiler knows that this line is actually part of the one before it) and code always starts in column 7.

answered Sep 10 '08 at 14:12

dagorym
2,212 2 15 20

4 This comment is true of FORTRAN 77 and earlier, but not for Fortran 90 and later, which uses free-form source layout. – M. S. B. Jun 7 '10 at 21:36

1 Also, the lines are supposed to finish in the 72th column, not 80th. – Rook Sep 27 '11 at 17:57