

Is Fortran easier to optimize than C for heavy calculations?

Asked 16 years, 2 months ago Modified 2 years, 7 months ago

Viewed 189k times



477



From time to time I read that Fortran is or can be faster than C for heavy calculations. Is that really true? I must admit that I hardly know Fortran, but the Fortran code I have seen so far did not show that the language has features that C doesn't have.



If it is true, please tell me why. Please don't tell me what languages or libs are good for number crunching, I don't intend to write an app or lib to do that, I'm just curious.

c

performance

fortran

Share

Improve this question

Follow

edited Jan 28, 2018 at 8:40



user325117

asked Sep 28, 2008 at 16:02



quinmars

11.6k ● 8 ● 34 ● 41


67 Notreally subjective from the answers given below. The correct title is "Are there any fundemental architectural

reasons why a Fortran compiler MIGHT produce better optimised code than a C compiler" but that's just being nit-picking. – [Martin Beckett](#) Sep 28, 2008 at 16:34

The question is too broad (Fortran is faster than C in which problem domain?), but I don't think the following discussion is subjective. We can produce a small code that proves/disproves any claim, as long as the problem is specific enough – [Paulus](#) Oct 6, 2008 at 5:10

1 Note that you don't need to write your program in Fortran if all you want to do is call some Fortran libraries. One can easily call Fortran code from C, all you need to remember about is name mangling, passing every variable by reference and different matrix ordering. – [quant_dev](#) Jan 31, 2009 at 19:30

1 Why do noobies always ask is this faster than that type questions ? Half the time they worry about nonsense functions that take a few cycles every other hour. – [mP.](#) Aug 25, 2009 at 3:31

3 @sixlettervariables although you and I already know the answer, it is a question that occurs to most people early in their careers and its important to understand the answer. Rather than post a dismissive comment why not find an answer you agree with and give +1 – [MarkJ](#) Mar 12, 2012 at 12:42 

23 Answers

Sorted by:

Highest score (default)



520

The languages have similar feature-sets. The performance difference comes from the fact that Fortran says aliasing is not allowed, unless an EQUIVALENCE statement is used. Any code that has aliasing is not valid Fortran, but it is up to the programmer and not the



compiler to detect these errors. Thus Fortran compilers ignore possible aliasing of memory pointers and allow them to generate more efficient code. Take a look at this little example in C:

```
void transform (float *output, float const * input, fl
*n)
{
    int i;
    for (i=0; i<*n; i++)
    {
        float x = input[i*2+0];
        float y = input[i*2+1];
        output[i*2+0] = matrix[0] * x + matrix[1] * y;
        output[i*2+1] = matrix[2] * x + matrix[3] * y;
    }
}
```

This function would run slower than the Fortran counterpart after optimization. Why so? If you write values into the output array, you may change the values of matrix. After all, the pointers could overlap and point to the same chunk of memory (including the `int` pointer!). The C compiler is forced to reload the four matrix values from memory for all computations.

In Fortran the compiler can load the matrix values once and store them in registers. It can do so because the Fortran compiler assumes pointers/arrays do not overlap in memory.

Fortunately, the `restrict` keyword and strict-aliasing have been introduced to the C99 standard to address this problem. It's well supported in most C++ compilers these

days as well. The keyword allows you to give the compiler a hint that the programmer promises that a pointer does not alias with any other pointer. The strict-aliasing means that the programmer promises that pointers of different type will never overlap, for example a `double*` will not overlap with an `int*` (with the specific exception that `char*` and `void*` can overlap with anything).

If you use them you will get the same speed from C and Fortran. However, the ability to use the `restrict` keyword only with performance critical functions means that C (and C++) programs are much safer and easier to write. For example, consider the invalid Fortran code:

```
CALL TRANSFORM(A(1, 30), A(2, 31), A(3, 32), 30),
```

which most Fortran compilers will happily compile without any warning but introduces a bug that only shows up on some compilers, on some hardware and with some optimization options.

Share Improve this answer

Follow

edited Mar 7, 2019 at 21:34



[user823738](#)

17.5k ● 8 ● 54 ● 78

answered Sep 28, 2008 at 16:14



[Nils Pipenbrinck](#)

86.2k ● 33 ● 155 ● 223

30 All true and valid, jeff. However, I don't consider the "assume no aliasing"-switch safe. It can break code inherited from other projects in so subtle ways that I'd rather not use it. I've become a restrict-nazi for that reason :-)) – [Nils Pipenbrinck](#)
Sep 28, 2008 at 16:27

- 39 A good example is the mere existence of `memcpy()` vs. `memmove()`. Unlike `memcpy()`, `memmove()` copes with overlapping areas, therefore `memcpy()` can be faster than `memmove()`. This issue was sufficient reason for somebody to include two functions instead of one into the standard library. – [jfs](#) Sep 28, 2008 at 17:24
-
- 13 I think that was Sebastian's point - because of the complexity/flexibility of C memory handling, even something as simple as moving memory is tricky. – [Martin Beckett](#) Sep 29, 2008 at 1:24
-
- 5 Your `call transform` example doesn't have much sense. – [Vladimir F Героям слава](#) May 13, 2014 at 19:14
-
- 5 Fortran has a similar feature set as C? Really? First, which Fortran? The current popular Fortran standards (2003/2008) = super flexible module standard, OO (for those who care), native parallel programming support (`do parallel`), native distributed computing support (`co-array fortran`). If you start your program from scratch, it's so much easier to write a parallel numerical application. Also, interfacing with C libs is easy via the `iso_c_binding` module. – [Mali Remorker](#) Jun 20, 2018 at 7:54
-



Yes, in 1980; in 2008? depends

182



When I started programming professionally the speed dominance of Fortran was just being challenged. I remember [reading about it in Dr. Dobbs](#) and telling the older programmers about the article--they laughed.



So I have two views about this, theoretical and practical. *In theory* Fortran today has no intrinsic advantage to C/C++ or even any language that allows assembly code.

In practice Fortran today still enjoys the benefits of legacy of a history and culture built around optimization of numerical code.

Up until and including Fortran 77, language design considerations had optimization as a main focus. Due to the state of compiler theory and technology, this often meant *restricting* features and capability in order to give the compiler the best shot at optimizing the code. A good analogy is to think of Fortran 77 as a professional race car that sacrifices features for speed. These days compilers have gotten better across all languages and features for programmer productivity are more valued. However, there are still places where the people are mainly concerned with speed in scientific computing; these people most likely have inherited code, training and culture from people who themselves were Fortran programmers.

When one starts talking about optimization of code there are many issues and the best way to get a feel for this is [to lurk where people are whose job it is to have fast numerical code](#). But keep in mind that such critically sensitive code is usually a small fraction of the overall lines of code and very specialized: A lot of Fortran code is just as "inefficient" as a lot of other code in other languages and [optimization should not even be a primary concern of such code](#).

A wonderful place to start in learning about the history and culture of Fortran is wikipedia. [The Fortran Wikipedia](#)

[entry](#) is superb and I very much appreciate those who have taken the time and effort to make it of value for the Fortran community.

(A shortened version of this answer would have been a comment in the excellent thread started by **Nils** but I don't have the karma to do that. Actually, I probably wouldn't have written anything at all but for that this thread has actual information content and sharing as opposed to flame wars and language bigotry, which is my main experience with this subject. I was overwhelmed and had to share the love.)

Share Improve this answer

Follow

edited Oct 17, 2012 at 20:46



Russ

11.3k ● 12 ● 45 ● 57

answered Oct 5, 2008 at 23:23



jaredor

2,284 ● 1 ● 14 ● 9

2 the link:

web.archive.org/web/20090401205830/http://ubiety.uwaterloo.ca/... no longer works. Is there an alternative link?

– [nathanielng](#) Feb 5, 2015 at 9:07



70



To some extent Fortran has been designed keeping compiler optimization in mind. The language supports whole array operations where compilers can exploit parallelism (specially on multi-core processors). For example,

Dense matrix multiplication is simply:

```
matmul(a,b)
```

L2 norm of a vector x is:

```
sqrt(sum(x**2))
```

Moreover statements such as `FORALL` , `PURE` & `ELEMENTAL` procedures etc. further help to optimize code. Even pointers in Fortran arent as flexible as C because of this simple reason.

The upcoming Fortran standard (2008) has co-arrays which allows you to easily write parallel code. G95 (open source) and compilers from CRAY already support it.

So yes Fortran can be fast simply because compilers can optimize/parallelize it better than C/C++. But again like everything else in life there are good compilers and bad compilers.

Share Improve this answer

edited Feb 10, 2016 at 20:20

Follow



CT Zhu

54.3k ● 18 ● 124 ● 135

answered Dec 29, 2008 at 4:56



user49734

717 ● 5 ● 2

You may wish to read

pimiddy.wordpress.com/2012/04/20/pure-functions-in-cc and

open-std.org/jtc1/sc22/wg21/docs/papers/2015/p0078r0.pdf

as well. – Jeff Hammond Dec 23, 2016 at 0:51

-
- 1 The `forall` construct is deprecated because compilers couldn't optimize the code well. The replacement is `do concurrent`. Also, the code `sqrt(sum(x**2))` looks inefficient, because the compiler probably constructs the whole vector `x**2`. I would guess that a loop is better, but it's undoubtedly best to call the intrinsic `norm2` function.
- knia Aug 9, 2017 at 11:34 ✎
-



42



It is funny that a lot of answers here from not knowing the languages. This is especially true for C/C++ programmers who have opened and old FORTRAN 77 code and discuss the weaknesses.

I suppose that the speed issue is mostly a question between C/C++ and Fortran. In a Huge code, it always depends on the programmer. There are some features of the language that Fortran outperforms and some features which C does. So, in 2011, no one can really say which one is faster.

About the language itself, Fortran nowadays supports Full OOP features and it is fully backward compatible. I have used the Fortran 2003 thoroughly and I would say it was just delightful to use it. In some aspects, Fortran 2003 is still behind C++ but let's look at the usage. Fortran is mostly used for Numerical Computation, and nobody uses fancy C++ OOP features because of speed reasons. In high performance computing, C++ has almost no place to go(have a look at the MPI standard and you'll see that C++ has been deprecated!).

Nowadays, you can simply do mixed language programming with Fortran and C/C++. There are even interfaces for GTK+ in Fortran. There are free compilers (gfortran, g95) and many excellent commercial ones.

Share Improve this answer

Follow

edited Sep 7, 2017 at 7:18



user2974951

65 ● 13

answered Oct 30, 2011 at 8:42



Hossein Talebi

445 ● 4 ● 2

13 Could you please add a source, specific experience, or scientific institute /high performance computing agency that is either moving away from C++ for future projects or rewriting c++ projects to another language. I am only asking because I know at least two scientific institutions, Sandia and CERN, that heavily use c++ for their high performance modeling. Additionally Sandia converted one of their modeling software (LAMMPS) over from fortran to c++ adding in a number of awesome enhancements.

– [Zachary Kraus](#) Sep 9, 2015 at 3:46

11 LAMMPS is written in extremely simple C++ that does not take advantage of most of the modern features of the language. It represents the C++ that Fortran programmers who know C write after learning C++98. This is not to say LAMMPS is poorly written, just that it is not the example you want to cite when advocating for C++ in HPC.

– [Jeff Hammond](#) Dec 23, 2016 at 1:01



33



There are several reasons why Fortran could be faster. However the amount they matter is so inconsequential or can be worked around anyways, that it shouldn't matter. The main reason to use Fortran nowadays is maintaining or extending legacy applications.



- PURE and ELEMENTAL keywords on functions. These are functions that have no side effects. This allows optimizations in certain cases where the compiler knows the same function will be called with the same values. *Note: GCC implements "pure" as an extension to the language. Other compilers may*

as well. Inter-module analysis can also perform this optimization but it is difficult.

- standard set of functions that deal with arrays, not individual elements. Stuff like `sin()`, `log()`, `sqrt()` take arrays instead of scalars. This makes it easier to optimize the routine. *Auto-vectorization gives the same benefits in most cases if these functions are inline or builtins*
- Builtin complex type. In theory this could allow the compiler to reorder or eliminate certain instructions in certain cases, but likely you'd see the same benefit with the `struct { double re; double im; };` idiom used in C. It makes for faster development though as operators work on complex types in Fortran.

Share Improve this answer

Follow

edited Jan 28, 2021 at 12:09



klutt

31.2k ● 19 ● 61 ● 105

answered Sep 28, 2008 at 16:41



Greg Rogers

36.4k ● 17 ● 69 ● 94

-
- 2 "but likely you'd see the same benefit with the struct `{ double re, im; };` idiom used in C". C compilers will most likely return that struct in sret form with the caller stack allocating space, passing a pointer to the callee that fills it in. That is several times slower than returning multiple values in registers as a Fortran compiler would. Note that C99 fixed this in the special case of complex. – J D Jan 29, 2012 at 19:48
-

I am not sure I agree with your main reason. I personally like to use fortran for math heavy code where arrays and math functions are the most important part of the code. But I know for a fact in a lot of government institutions and banks they continue to use fortran to maintain or extend legacy code. I personally used fortran to extend code a professor on my Ph.D. committee wrote. – [Zachary Kraus](#) Sep 9, 2015 at 3:36

- 1 Your statement, "the main reason to use Fortran nowadays is maintaining or extending legacy applications", is completely wrong. I have yet to see any other programming language even remotely close to the features Fortran provides, especially for Mathematical applications. You already named a few of them, such as superb array support, built-in complex arithmetic, pure or elemental functions - and I could name more as well. That alone is enough to prove your statement above is wrong. – [Pap](#) Jul 14, 2017 at 17:58
-



32



I think the key point in favor of Fortran is that it is a language slightly more suited for expressing vector- and array-based math. The pointer analysis issue pointed out above is real in practice, since portable code cannot really assume that you can tell a compiler something. There is ALWAYS an advantage to expression computations in a manner closer to how the domain looks. C does not really have arrays at all, if you look closely, just something that kind of behaves like it. Fortran has real arrays. Which makes it easier to compile for certain types of algorithms especially for parallel machines.

Deep down in things like run-time system and calling conventions, C and modern Fortran are sufficiently similar

that it is hard to see what would make a difference. Note that C here is really base C: C++ is a totally different issue with very different performance characteristics.

Share Improve this answer

edited Mar 6, 2010 at 10:58

Follow



F'x

12.3k ● 8 ● 73 ● 125

answered Sep 28, 2008 at 18:13



jakobengblom2

5,843 ● 2 ● 28 ● 34



There is no such thing as one language being faster than another, so the proper answer is **no**.

29



What you really have to ask is "is code compiled with Fortran compiler X faster than equivalent code compiled with C compiler Y?" The answer to that question of course depends on which two compilers you pick.



Another question one could ask would be along the lines of "Given the same amount of effort put into optimizing in their compilers, which compiler would produce faster code?" The answer to this would in fact be *Fortran*.

Fortran compilers have certian advantages:

- Fortran had to compete with Assembly back in the day when some vowed never to use compilers, so it was designed for speed. C was designed to be flexible.

- Fortran's niche has been number crunching. In this domain code is *never* fast enough. So there's always been a lot of pressure to keep the language efficient.
- Most of the research in compiler optimizations is done by people interested in speeding up Fortran number crunching code, so optimizing Fortran code is a much better known problem than optimizing any other compiled language, and new innovations show up in Fortran compilers first.
- **Biggie:** C encourages much more pointer use than Fortran. This drastically increases the potential scope of any data item in a C program, which makes them far harder to optimize. Note that Ada is also way better than C in this realm, and is a much more modern OO Language than the commonly found Fortran77. If you want an OO language that can generate faster code than C, this is an option for you.
- Due again to its number-crunching niche, the customers of Fortran compilers tend to care more about optimization than the customers of C compilers.

However, there is nothing stopping someone from putting a ton of effort into their C compiler's optimization, and making it generate better code than their platform's Fortran compiler. In fact, the larger sales generated by C compilers makes this scenario quite feasible

Share Improve this answer

edited May 12, 2011 at 13:33

Follow

answered Oct 30, 2008 at 13:37



[T.E.D.](#)

44.7k ● 10 ● 76 ● 138

-
- 6 I agree. And may I add that when Fortran was introduced (it was the first high level language) in the late 50's early 60's many were skeptical about how efficient it could be. Therefore its developers had to prove that Fortran could be efficient and useful and set about to "optimize it to death" just to prove their point. C came much later (early to mid 70's) and had nothing to prove, so to speak. But by this time a lot of Fortran code had been written so the scientific community stuck to it and still does. I don't program Fortran but I've learned to link to call Fortran subroutines from C++.
- [Olumide](#) May 12, 2011 at 2:08
-

- 5 The research on compiler optimization has been more diverse than you seem to think. The history of LISP implementations, for example, is full of successes at doing number crunching faster than Fortran (which remains the default contender to challenge). Also, a huge part of compiler optimization has targeted intermediate representations of the compiler, which means that, differences in semantics aside (like aliasing), they apply to any programming language of a given class. – [Pierre Thierry](#) Aug 2, 2013 at 1:46
-

- 4 The idea is repeated a lot, but it's a bit disingenuous to say that there's no consequences for efficiency inherent to a programming language design. Some programming language features necessarily lead to inefficiencies because they limit the information available at compile time. – [Praxeolitic](#) Aug 26, 2015 at 23:24
-

@Praxeolitic - That is quite correct (which is why I am pleased that I said no such thing). – [T.E.D.](#) Nov 16, 2015 at 19:35 ✎

@T.E.D. Honestly, coming back here a few months later I have no idea why I left that comment on your answer. Maybe I meant to leave it somewhere else? X-P – [Praxeolitic](#) Nov 16, 2015 at 21:43



21



There is another item where Fortran is different than C - and potentially faster. Fortran has better optimization rules than C. In Fortran, the evaluation order of an expressions is not defined, which allows the compiler to optimize it - if one wants to force a certain order, one has to use parentheses. In C the order is much stricter, but with "-fast" options, they are more relaxed and "(...)" are also ignored. I think Fortran has a way which lies nicely in the middle. (Well, IEEE makes the live more difficult as certain evaluation-order changes require that no overflows occur, which either has to be ignored or hampers the evaluation).

Another area of smarter rules are complex numbers. Not only that it took until C 99 that C had them, also the rules govern them is better in Fortran; since the Fortran library of gfortran is partially written in C but implements the Fortran semantics, GCC gained the option (which can also be used with "normal" C programs):

-fcx-fortran-rules Complex multiplication and division follow Fortran rules. Range reduction is done as part of complex division, but there is no checking whether the result of a complex

multiplication or division is "NaN + I*NaN", with an attempt to rescue the situation in that case.

The alias rules mentioned above is another bonus and also - at least in principle - the whole-array operations, which if taken properly into account by the optimizer of the compiler, can lead faster code. On the contra side are that certain operation take more time, e.g. if one does an assignment to an allocatable array, there are lots of checks necessary (reallocate? [Fortran 2003 feature], has the array strides, etc.), which make the simple operation more complex behind the scenes - and thus slower, but makes the language more powerful. On the other hand, the array operations with flexible bounds and strides makes it easier to write code - and the compiler is usually better optimizing code than a user.

In total, I think both C and Fortran are about equally fast; the choice should be more which language does one like more or whether using the whole-array operations of Fortran and its better portability are more useful -- or the better interfacing to system and graphical-user-interface libraries in C.

Share Improve this answer

answered Oct 30, 2008 at 13:04

Follow



T B

What is there to meaningfully "rescue" in the Nan+INan case? What makes infinities different from NaN is that infinities are signed. Operations involving non-complex infinities that would yield a muddled sign yield NaN and I see

no reason complex numbers should do otherwise. If one multiples (DBL_MAX,DBL_MAX) by (2,2), squares the result, and then squares that result, what should be the sign of the result in the absence of overflow? What is one instead multiples it by (1.001,DBL_MAX)? I'd regard (NaN,NaN) as the correct answer, and any combination of infinity and NaN as nonsensical. – [supercat](#) Jun 5, 2018 at 6:23



15



There is nothing about the *languages* Fortran and C which makes one faster than the other for specific purposes. There are things about specific *compilers* for each of these languages which make some favorable for certain tasks more than others.



For many years, Fortran compilers existed which could do black magic to your numeric routines, making many important computations insanely fast. The contemporary C compilers couldn't do it as well. As a result, a number of great libraries of code grew in Fortran. If you want to use these well tested, mature, wonderful libraries, you break out the Fortran compiler.

My informal observations show that these days people code their heavy computational stuff in any old language, and if it takes a while they find time on some cheap compute cluster. Moore's Law makes fools of us all.

Share Improve this answer

answered Sep 28, 2008 at 16:35

Follow



[jfm3](#)

37.7k ● 10 ● 34 ● 35

14 *Almost upmodded this. The problem is that Fortran *does* have some inherent advantages. However, you are quite corrent that the important thing to look at is the compier, not the language.* – [T.E.D.](#) Oct 30, 2008 at 13:46



15



Any speed differences between Fortran and C will be more a function of compiler optimizations and the underlying math library used by the particular compiler. There is nothing intrinsic to Fortran that would make it faster than C.



Anyway, a good programmer can write Fortran in any language.



Share Improve this answer

edited Sep 28, 2008 at 23:23

Follow

answered Sep 28, 2008 at 16:09



Kluge

3,707 ● 3 ● 27 ● 22

1 @Scott Clawson: you got -1'd and I don't know why. +1'd to remedy this. However, something to take into account is Fortran has been around longer than a lot of our parents have been. Lots of time spent optimizing compiler output :D – [user7116](#) Sep 28, 2008 at 16:12

I agree. I had just posted a very similar answer in parallel.

– [Tall Jeff](#) Sep 28, 2008 at 16:14

Pointer alias issue in C has been raised by others, but there are several methods the programmer can use on modern

compilers to deal with that, so I still agree. – Tall Jeff Sep 28, 2008 at 16:37

- 4 @Kluge: "There is nothing intrinsic to Fortran that would make it faster than C". Pointer aliasing, returning compound values in registers, built-in higher-level numerical constructs... – J D Jan 29, 2012 at 19:57
-



15



I compare speed of Fortran, C, and C++ with the classic Levine-Callahan-Dongarra benchmark from netlib. The multiple language version, with OpenMP, is

<http://sites.google.com/site/tprincesite/levine-callahan-dongarra-vectors>

The C is uglier, as it began with automatic translation, plus insertion of restrict and pragmas for certain compilers. C++ is just C with STL templates where applicable. To my view, the STL is a mixed bag as to whether it improves maintainability.

There is only minimal exercise of automatic function inlining to see to what extent it improves optimization, since the examples are based on traditional Fortran practice where little reliance is place on in-lining.

The C/C++ compiler which has by far the most widespread usage lacks auto-vectorization, on which these benchmarks rely heavily.

Re the post which came just before this: there are a couple of examples where parentheses are used in Fortran to dictate the faster or more accurate order of evaluation. Known C compilers don't have options to

observe the parentheses without disabling more important optimizations.

Share Improve this answer

edited Oct 30, 2008 at 13:18

Follow

answered Oct 30, 2008 at 13:11



The ugly translation is required to overcome the aliasing problem. You have to give the compiler mock variables to tell it that byref variables implemented as pointers can be register-optimised. – [david](#) Feb 20, 2020 at 4:32

So which one is faster? – [Nike](#) Dec 28, 2022 at 23:52



15



I'm a hobbyist programmer and i'm "average" at both language. I find it easier to write fast Fortran code than C (or C++) code. Both Fortran and C are "historic" languages (by today standard), are heavily used, and have well supported free and commercial compiler.



I don't know if it's an historic fact but Fortran feel like it's built to be paralleled/distributed/vectorized/whatever-many-cores-ized. And today it's pretty much the "standard metric" when we're talking about speed : "does it scale ?"

For pure cpu crunching i love Fortran. For anything IO related i find it easier to work with C. (it's difficult in both case anyway).

Now of course, for parallel math intensive code you probably want to use your GPU. Both C and Fortran have a lot of more or less well integrated CUDA/OpenCL interface (and now OpenACC).

My moderately objective answer is : If you know both language equally well/poorly then i think Fortran is faster because i find it easier to write parallel/distributed code in Fortran than C. (once you understood that you can write "freeform" fortran and not just strict F77 code)

Here is a 2nd answer for those willing to downvote me because they don't like the 1st answer : Both language have the features required to write high-performance code. So it's dependent of the algorithm you're implementing (cpu intensive ? io intensive ? memory intensive?), the hardware (single cpu ? multi-core ? distribute supercomputer ? GPGPU ? FPGA ?), your skill and ultimately the compiler itself. Both C and Fortran have awesome compiler. (i'm seriously amazed by how advanced Fortran compilers are but so are C compilers).

PS : i'm glad you specifically excluded libs because i have a great deal of bad stuff to say about Fortran GUI libs. :)

Share Improve this answer

Follow

answered Nov 6, 2015 at 11:17



ker2x

450 ● 3 ● 12



13



Quick and simple: **Both are equally fast, but Fortran is simpler.** Whats really faster in the end depends on the algorithm, but there is considerable no speed difference anyway. This is what I learned in a Fortran workshop at high performance computing center Stuttgart, Germany in 2015. I work both with Fortran and C and share this opinion.

Explanation:

C was designed to write operating systems. Hence it has more freedom than needed to write high performance code. In general this is no problem, but if one does not programm carefully, one can easily slow the code down.

Fortran was designed for scientific programming. For this reason, it supports writing fast code syntax-wise, as this is the main purpose of Fortran. In contrast to the public opinion, Fortran is not an outdated programming language. Its latest standard is 2010 and new compilers are published on a regular basis, as most high performance code is written in Fortran. [Fortran further supports modern features as compiler directives \(in C pragmas\).](#)

Example: *We want to give a large struct as an input argument to a function (fortran: subroutine). Within the function the argument is not altered.*

C supports both, call by reference and call by value, which is a handy feature. In our case, the programmer

might by accident use call by value. This slows down things considerably, as the struct needs to be copied in within memory first.

Fortran works with call by reference only, which forces the programmer to copy the struct by hand, if he really wants a call by value operation. In our case fortran will be automatically as fast as the C version with call by reference.

Share Improve this answer

answered Nov 10, 2016 at 10:16

Follow



[Markus Dutschke](#)

10.5k ● 4 ● 72 ● 64

Can you write a native parallel application in C (meaning, without calling any libraries?). No. Can you write a native parallel application in Fortran? Yes, in a few different ways. Ergo, Fortran is "faster". ALthough, to be fair, in 2010 when you wrote your comment, support for parallel do and co-array features of Fortran were probably not as widespread as they are now. – [Mali Remorker](#) Jun 20, 2018 at 8:02

-
- 1 @MaliRemorker He wrote this in 2016, not in 2010.
– [Ekrem Dinçel](#) Mar 28, 2020 at 12:41

The overhead of creating parallel processes is in my opinion not the most relevant factor for a programming language being called 'fast'. More relevant are practical considerations, like a non-performant peace of code. So it does not help anything if you save time once (when creating the parallel process) and waste it on multiple cores later. The term 'fast' should also consider non-parallel codes. For this reason, I can not see an argument against my point. Maybe your comment was ment as an independent answer?
– [Markus Dutschke](#) Mar 28, 2020 at 16:11



12



I was doing some extensive mathematics with FORTRAN and C for a couple of years. From my own experience I can tell that FORTRAN is sometimes really better than C but not for its speed (one can make C perform as fast as FORTRAN by using appropriate coding style) but rather because of very well optimized libraries like LAPACK (which can, however, be called from C code as well, either linking against LAPACK directly or using the LAPACKE interface for C), and because of great parallelization. On my opinion, FORTRAN is really awkward to work with, and its advantages are not good enough to cancel that drawback, so now I am using C+GSL to do calculations.

Share Improve this answer

Follow

edited May 2, 2022 at 14:01



texnokrates

3 ● 2

answered Jul 29, 2010 at 20:01



grzkv

2,619 ● 3 ● 28 ● 37



10



I haven't heard that Fortan is significantly faster than C, but it might be conceivable tht in certain cases it would be faster. And the key is not in the language features that are present, but in those that (usually) absent.

An example are C pointers. C pointers are used pretty much everywhere, but the problem with pointers is that



the compiler usually can't tell if they're pointing to the different parts of the same array.

For example if you wrote a strcpy routine that looked like this:

```
strcpy(char *d, const char* s)
{
    while(*d++ = *s++);
}
```

The compiler has to work under the assumption that the d and s might be overlapping arrays. So it can't perform an optimization that would produce different results when the arrays overlap. As you'd expect, this considerably restricts the kind of optimizations that can be performed.

[I should note that C99 has a "restrict" keyword that explicitly tells the compilers that the pointers don't overlap. Also note that the Fortran too has pointers, with semantics different from those of C, but the pointers aren't ubiquitous as in C.]

But coming back to the C vs. Fortran issue, it is conceivable that a Fortran compiler is able to perform some optimizations that might not be possible for a (straightforwardly written) C program. So I wouldn't be too surprised by the claim. However, I do expect that the performance difference wouldn't be all that much. [~5-10%]



9



Generally FORTRAN is slower than C. C can use hardware level pointers allowing the programmer to hand-optimize. FORTRAN (in most cases) doesn't have access to hardware memory addressing hacks. (VAX FORTRAN is another story.) I've used FORTRAN on and off since the '70's. (Really.)

However, starting in the 90's FORTRAN has evolved to include specific language constructs that can be optimized into inherently parallel algorithms that *can* really scream on a multi-core processor. For example, automatic Vectorizing allows multiple processors to handle each element in a vector of data concurrently. 16 processors -- 16 element vector -- processing takes 1/16th the time.

In C, you have to manage your own threads and design your algorithm carefully for multi-processing, and then use a bunch of API calls to make sure that the parallelism happens properly.

In FORTRAN, you only have to design your algorithm carefully for multi-processing. The compiler and run-time can handle the rest for you.

You can read a little about [High Performance Fortran](#), but you find a lot of dead links. You're better off reading about

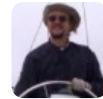
Parallel Programming (like OpenMP.org) and how FORTRAN supports that.

Share Improve this answer

edited Dec 5, 2011 at 10:51

Follow

answered Sep 28, 2008 at 16:28



[S.Lott](#)

391k ● 82 ● 517 ● 788

10 @S.Lott: I couldn't imagine how awful C code would have to look to do as good as simply written Fortran for most of the codes we have here...and I'm a C programmer. You'll get better performance out of simpler code in Fortran. Not that you or I couldn't find a counterexample. :D – [user7116](#) Sep 28, 2008 at 16:35

3 No compiler is going to spread a computation on 16 elements of a vector to 16 different CPUs. That'd be hundreds of times slower... – [Greg Rogers](#) Sep 28, 2008 at 16:57

5 -1 // "Generally FORTRAN is slower than C. This is true for almost everything." why? // an argument based on the ease of using multi-threading in Fortran vs C isn't saying something about the performance. – [steabert](#) Dec 5, 2011 at 9:28

5 @S.Lott: "multi-threading only exists for performance". Err, no. – [J D](#) Jan 29, 2012 at 20:02

6 @S.Lott: "C's use of pointers almost at the hardware level allows C to be faster than Fortran". Err, no – [J D](#) Jan 29, 2012 at 20:04



5



The faster code is not really up to the language, is the compiler so you can see the ms-vb "compiler" that generates bloated, slower and redundant object code that is tied together inside an ".exe", but powerBasic generates too way better code. Object code made by a C and C++ compilers is generated in some phases (at least 2) but by design most Fortran compilers have at least 5 phases including high-level optimizations so by design Fortran will always have the capability to generate highly optimized code. So at the end is the compiler not the language you should ask for, the best compiler i know is the Intel Fortran Compiler because you can get it on LINUX and Windows and you can use VS as the IDE, if you're looking for a cheap tigh compiler you can always relay on OpenWatcom.

More info about this: <http://ed-thelen.org/1401Project/1401-IBM-Systems-Journal-FORTRAN.html>

Share Improve this answer

answered Jul 17, 2010 at 3:26

Follow



JPerez45

67 ● 1 ● 1

"So at the end is the compiler not the language you should ask for, the best compiler i know is the Intel Fortran Compiler" I guess you could ask though "which language has better compilers" or "which language is better suited to writing high-performance compilers". – [Nike](#) Dec 29, 2022 at 0:04



Fortran has better I/O routines, e.g. the implied do facility gives flexibility that C's standard library can't match.

3



The Fortran compiler directly handles the more complex syntax involved, and as such syntax can't be easily reduced to argument passing form, C can't implement it efficiently.



Share Improve this answer

answered Apr 5, 2015 at 18:17

Follow



Zeus

1,525 ● 1 ● 18 ● 35

2 Do you have benchmarks that show Fortran beating C for I/O? – [Jeff Hammond](#) Dec 23, 2016 at 0:53



3



Fortran can handle array, especially multidimensional arrays, very conveniently. Slicing elements of multidimensional array in Fortran can be much easier than that in C/C++. C++ now has libraries can do the job, such as Boost or Eigen, but they are after all external libraries. In Fortran these functions are intrinsic.



Whether Fortran is faster or more convenient for developing mostly depends on the job you need to finish. As a scientific computation person for geophysics, I did most of computation in Fortran (I mean modern Fortran, >=F90).

answered Jul 2, 2018 at 18:31

Share Improve this answer

Follow



Kai

81 ● 1 ● 7

-
- 1 And also whether Fortran is faster also depends on: the compiler you use (it matters!), for computation jobs whether you apply appropriate parallelization to the code, and how your code is written. – Kai Jul 2, 2018 at 18:34
-

Kai I think it would be better if you put that comment into your answer. I can't do it because "too many edits are pending on Stack Overflow". – Nike Dec 28, 2022 at 23:56



2



Using modern standards and compiler, no!

Some of the folks here have suggested that FORTRAN is faster because the compiler doesn't need to worry about aliasing (and hence can make more assumptions during optimisation). However, this has been dealt with in C since the C99 (I think) standard with the inclusion of the restrict keyword. Which basically tells the compiler, that within a give scope, the pointer is not aliased.

Furthermore C enables proper pointer arithmetic, where things like aliasing can be very useful in terms of performance and resource allocation. Although I think more recent version of FORTRAN enable the use of "proper" pointers.

For modern implementations C general outperforms FORTRAN (although it is very fast too).

<http://benchmarksgame.alioth.debian.org/u64q/fortran.html>

EDIT:

A fair criticism of this seems to be that the benchmarking may be biased. Here is another source (relative to C) that puts result in more context:

<http://julialang.org/benchmarks/>

You can see that C typically outperforms Fortran in most instances (again see criticisms below that apply here too); as others have stated, benchmarking is an inexact science that can be easily loaded to favour one language over others. But it does put in context how Fortran and C have similar performance.

Share Improve this answer

edited Dec 19, 2016 at 23:21

Follow

answered Dec 19, 2016 at 17:00



cdcddcd

569 ● 1 ● 6 ● 15

-
- 4 One would be a fool to believe anything about Fortran from a post that assumes it is still FORTRAN. That changed more than 25 years ago. Those tests cannot compare languages, because they use compilers from different vendors although both Intel and GCC have compilers for both C and Fortran. Therefore those comparisons are worthless.

– Vladimir F Героям слава Jan 28, 2018 at 10:58 



0



This is more than somewhat subjective, because it gets into the quality of compilers and such more than anything else. However, to more directly answer your question, speaking from a language/compiler standpoint there is nothing about Fortran over C that is going to make it inherently faster or better than C. If you are doing heavy math operations, it will come down to the quality of the compiler, the skill of the programmer in each language and the intrinsic math support libraries that support those operations to ultimately determine which is going to be faster for a given implementation.

EDIT: Other people such as @Nils have raised the good point about the difference in the use of pointers in C and the possibility for aliasing that perhaps makes the most naive implementations slower in C. However, there are ways to deal with that in C99, via compiler optimization flags and/or in how the C is actually written. This is well covered in @Nils answer and the subsequent comments that follow on his answer.

Share Improve this answer

edited Sep 28, 2008 at 16:47

Follow

answered Sep 28, 2008 at 16:11



Tall Jeff

9,984 ● 7 ● 46 ● 61

It sounds like a benchmark test of an algorithm. Which takes less time, FORTRAN or C? Doesn't sound subjective to me.

Perhaps I'm missing something. – [S.Lott](#) Sep 28, 2008 at 16:30

- 1 Disagree. You are comparing the compilers, not the languages. I think the original question is if there is anything about the LANGUAGE that makes it inherently better. Other answers here are getting into some of the subtle questionable differences, but I think we most agree they are in the noise. – [Tall Jeff](#) Sep 28, 2008 at 16:35
-

This isn't $O(n)$ analysis of algorithms. It's Performance. Don't see how performance can be a hypothetical implementation-independent concept. Guess I'm missing something. – [S.Lott](#) Sep 28, 2008 at 17:24

- 1 -1: "there is nothing about Fortran over C that is going to make it inherently faster or better than C". Err, no. – [J D](#) Jan 29, 2012 at 20:06
-



-1

Most of the posts already present compelling arguments, so I will just add the proverbial 2 cents to a different aspect.



Being fortran faster or slower in terms of processing power in the end can have its importance, but if it takes 5 times more time to develop something in Fortran because:



- it lacks any good library for tasks different from pure number crunching
- it lack any decent tool for documentation and unit testing

- it's a language with very low expressivity, skyrocketing the number of lines of code.
- it has a very poor handling of strings
- it has an inane amount of issues among different compilers and architectures driving you crazy.
- it has a very poor IO strategy (READ/WRITE of sequential files. Yes, random access files exist but did you ever see them used?)
- it does not encourage good development practices, modularization.
- effective lack of a fully standard, fully compliant opensource compiler (both gfortran and g95 do not support everything)
- very poor interoperability with C (mangling: one underscore, two underscores, no underscore, in general one underscore but two if there's another underscore. and just let not delve into COMMON blocks...)

Then the issue is irrelevant. If something is slow, most of the time you cannot improve it beyond a given limit. If you want something faster, change the algorithm. In the end, computer time is cheap. Human time is not. Value the choice that reduces human time. If it increases computer time, it's cost effective anyway.

Share Improve this answer

answered Jul 20, 2009 at 3:00

Follow



Stefano Borini

143k ● 100 ● 308 ● 446


4 downvoted because although you raise interesting points which add to a discussion of fortrons benefits/drawbacks versus other languages (which I don't completely agree with), this isn't really an answer to the question... – [steabert](#) Dec 5, 2011 at 9:14

1 +1 from me for some not-nitpicky answer. As you said, Fortran might be faster in some rare tasks (have not seen any personally). But the amount of time you waste for maintaining an unmaintainable language ruins any possible advantage. – [André Bergner](#) Jul 10, 2012 at 8:02

11 -1. You seem to be thinking of Fortran in terms of F77. That was superseded by F90, F95, F03, **and** F08. – [Kyle Kanos](#) May 27, 2013 at 14:39

4 Downvoted because it talks exclusively about one side of a trade-off. Development speed may matter for most general programming, but that doesn't make it the only valid trade-off. Fortran programmers are often scientists/engineers who value the simplicity of the language (FORMula TRANslation is extremely easy to learn and master. C/C++ is *not*), the excellent libraries (often used from other languages), and the speed (e.g. weather simulations that take days in Fortran, but months if written entirely in other languages). – [BraveNewCurrency](#) Oct 1, 2013 at 3:21

1 Fortran supports many types of I/O. I don't get your point about people not using features, that is their problem not Fortran's. It does encourage modularization with "modules" of all things. Very good interoperability with C using `iso_c_binding`, which is native Fortran. Gfortran free compiler is pretty darn good. It will never have every feature because

new ones keep getting added. – [Mark S](#) May 27, 2016 at 0:07 



-3



Fortran traditionally doesn't set options such as `-fp:strict` (which `ifort` requires to enable some of the features in `USE IEEE_arithmetic`, a part of `f2003` standard). Intel C++ also doesn't set `-fp:strict` as a default, but that is required for `ERRNO` handling, for example, and other C++ compilers don't make it convenient to turn off `ERRNO` or gain optimizations such as `simd reduction`. `gcc` and `g++` have required me to set up `Makefile` to avoid using the dangerous combination `-O3 -ffast-math -fopenmp -march=native`. Other than these issues, this question about relative performance gets more nit-picky and dependent on local rules about choice of compilers and options.

Share Improve this answer

answered May 26, 2015 at 13:14

Follow



[tim18](#)

620 ● 1 ● 5 ● 8

Recent `gcc` updates corrected the problem with combination of `Openmp` and `fast-math`. – [tim18](#) Sep 23, 2016 at 11:51

This answer is really about users not understanding the default flags for their compilers, not the languages themselves. – [Jeff Hammond](#) Dec 23, 2016 at 0:52



Highly active question. Earn 10 reputation (not counting the [association bonus](#)) in order to answer this question. The reputation

requirement helps protect this question from spam and non-answer activity.