GSoC21 LFortran and Computational Chemistry

Rohit Goswami · 2021-08-20 · gsoc21 · fortran · ramblings

> *Directed recollections of the GSoC21 timeline*

Background

If the last ten weeks of weekly posts have not made it clear; as a student developer in the Google Summer of Code in the year 2021; I was privileged to work on the LFortran compiler alongside Gagandeep Singh and Thirumalai Shaktivel under the fantastic mentorship of Ondřej Čertík under the Fortran-Lang organization.

Pencil Pushing

This section describes the overall work done in a manner designed to delight minds enamoured by brevity and numbers. It is by no means a replacement for going over the series.

Project Overview

Given that the LFortran compiler is implemented in a manner reminiscent of both `CPython` and `clang`, the main goal of the project was to leverage the compiler to parse, compile, link and run a production code base. For this, the code base chosen was Čertík, Pask, and Vackář (2013), which was further trimmed down to a miniature version here. Practically speaking, this involved tackling the Fortran standard itself, segregating the functionality at various stages in the “automatic coding” pipeline; and implementing standardese. Also, a lot of ****debugging**** naturally.

Statistics

For a single overview of the work done; the following table of contributions sums things up nicely.

Table 1: Code contributions to LFortran repository as of 2021-08-19T23:16:22+00:00

Contribution	Open	Closed/Merged	Total
Merge	10	22	36
Requests			
Issues	20	11	31

This is an ever evolving set of metrics, and the finer details of these contributions can be found by following the weekly series above. There were some extra contributions across the other repositories, involving minor fixes and some work done towards integrating with Meson.

MR Overview

These are better described in the weeks they were made; and only the highlights of the 22 will be briefly alluded to here.

Implement more kind() [997]

This was the first compile time enhancement / bugfix

Restructures and Grammar changes [1024, 1037]

Code quality enhancement for more readable ASR values

Binary Operations [1045, 1072, 1080]

Together these MRs implemented the population of value members for integer and real binary operations

Split ast_to_asr [1096]

A major code refactor for improved readablity and rapid development

tiny implementations [1107, 1112]

Together these implement the `tiny` intrinsic at compile time

LLVM values [1131]

This enhancement patchset furthered the population of value members at the backend

Compile time functions (5)

`real` [1114], `selected_kinds` [1154], `floor` [1176], `int` [1169], `Integer2Real` cast [1162]

Runtime sin [1167, 1173]

Together these show the hopes and dreams of the current runtime library implementation

Testing framework for runtime benchmarks [1194]

Jinja based rapid templating setup for benchmarking the runtime functions

Bugfixes [1015, 1087]

These span various growing pains, and were debugged mostly via the stacktrace

Reflections

So, after weeks of thousands of words of (semi-)popular writing, countless lines of code and hours of fantastic meetings; where are we and where is `LFortran` and where do we go from here?

- This is a rather vague point, given that we have a fantastic set of issues
- Also, the Zulip instance is great for getting in touch with the community
  - One of the nicer parts of being associated with the project in such a visible manner was that I was able to funnel new talent as well

Target Practice

In terms of finally compiling our minimum viable computational chemistry project, minidftatom; described in the Week 5 post, we are tantalizingly close; so much so that is a mere bugfix or two away; as can be seen from the state of this MR. Naturally my choice of issues are biased towards my own interests:

- Better benchmarking of the pure Fortran intrinsics
- Finish improving the CI (`ninja`)
- Meson integration
- More work on the runtime library
  - We plan to have multiple backends for the runtime (the relevant wiki)
  - We need also to implement the compile time intrinsics in a more streamlined manner (a RFC issue for the same)
- Document all things!

- API documentation and other bugaboos are pet interest of mine

Major Accomplishments

Perhaps more than anything else; the design strategies worked out over the twelve week span will have the most lasting impact. In particular:

Compile Time Intrinsics

> These are the C++ equivalent of the Fortran standard, and are evaluated when possible at compile time (the backend then just uses the value)

Runtime Library Design

> Inspite of strong community opposition to implementing elementary functions in pure Fortran; we ended up with a very well designed elegant runtime design I am immensely satisfied with. Broadly speaking (detailed in the
>
> $$Week10post$$
>
> ()), the design encompasses `impure` parts of the library, which cannot be implemented in pure Fortran, and `pure` sections which can be implemented without any external calls

Both of these intrinsic designs were accompanied by practical implementations and a template evolved over time to aid future contributions. One of the most interesting parts of the runtime design was using certifiably correct polynomial approximations generated with Sollya.

Implementing features at compile time required an intimate understanding of the ASR, along with the way it is generated and consumed. This took weeks of effort at first; especially given the generated nature of much of the underlying code.

Learning

Compiler design touches almost every aspect of the compuation; from hardware to linkers, and I managed to expand my understanding of several concepts early on with Cooper and Torczon (2011), Muller (2016), Levine (2000) and a host of other texts referenced in the early weeks.

A special mention must be made of "J3/18-007r1 (F2018 Interpretation Document)" (n.d.), which happens to be the first full language standard I have read almost cover to cover.

Misc Logs

Like every week; there are some notes of the implementations done this week as well, focused around the ****runtime intrinsic**** functions.

Exp

The method to calculate the exponent is largely taken from Detrey and de Dinechin (2005). Basically this needs a reduction to $-\frac{log2}{2}, \frac{log2}{2}$ but just to get the ball rolling an initial terrible approximation is essentially:

1. `floor(x)`, and compute `e` to the integral power in a loop
2. For the remaining part, use a 16th order Remez between `[0,1]`, something like `p=remez(exp(x),16, [0,1],default,1e-16,1e-30);`

This is clearly a terrible idea, and there are far better approaches, however, it will suffice for now.

Better Trignometric Functions

Visual inspection and basic trignometry (along with series expansions) brings about the understanding that our polynomial approximation should only contain odd terms. This is easily accomplished in Sollya with fpminimax.

```
P = fpminimax(sin(x),[|1,3,5,7,9,11,13,15,17,19,21|],[|1,D...|],[-pi/2;pi/2], fixed, relative);
```

We can also compute the supnorm of the approximation over the interval:

```
supnorm(P,sin(x),[-pi/2;pi/2],relative,2^(-40));
```

Which for the seventh order polynomial above turns out to be around 2E-17 throughout. The double word (DD) variant is not worth the extra effort, since the real issue with the approximation here is still the range reduction.

Conclusions

I like writing; but somehow this was one of the most difficult things to write about; mostly because I'm still itching to get back to the codebase. I believe this is probably a good sign. Surprisingly, GSoC21 turned out to be one of my most enriching experiences as a developer till date. Unsurprising given the fantastic people involved, actually. There is very much left to be said about `LFortran`, some of which will be said by me at the upcoming FortranCon 2021, and in other outlets.

References

Čertík, Ondřej, John E. Pask, and Jiří Vackář. 2013. "Dftatom: A Robust and General Schrödinger and Dirac Solver for Atomic Structure Calculations." *Computer Physics Communications* 184 (7): 1777--91. <https://doi.org/10.1016/j.cpc.2013.02.014>.

Cooper, Keith, and Linda Torczon. 2011. *Engineering a Compiler*. Elsevier. <https://books.google.com?id=_tgh4bgQ6PAC>.

Detrey, J., and F. de Dinechin. 2005. "A Parameterized Floating-Point Exponential Function for FPGAs." In *Proceedings. 2005 IEEE International Conference on Field-Programmable Technology, 2005.*, 27--34. Singapore, China: IEEE. <https://doi.org/10.1109/FPT.2005.1568520>.

"J3/18-007r1 (F2018 Interpretation Document)." n.d. Accessed June 28, 2021. <https://j3-fortran.org/doc/year/18/18-007r1.pdf>.

Levine, John R. 2000. *Linkers and Loaders*. San Francisco: Morgan Kaufmann.

Muller, Jean-Michel. 2016. *Elementary Functions*. Boston, MA: Birkhäuser Boston. <https://doi.org/10.1007/978-1-4899-7983-4>.