University of BRISTOL

Growing Up as a (Software) Engineer

Jim Cownie

July 2020 for UCL Knowledge Quarter Social

bristol.ac.uk

Objectives

1. Have a fun discussion
2. Give you some questions to think about
3. Tell you some illustrative stories (that do have a point besides being amusing)
4. Tell you not to worry
    1. You don’t need to plan what you’ll be doing in ten or twenty years
    2. You do need to enjoy what you’re doing
    3. You don’t need to become a manager

Introduction

I am going to give you questions to think about

And some answers

My answers may be wrong

My answers are based on experience in industry (not academia)

- I hope the experience (and questions) remain relevant even if my examples are based on my experiences

I am “male, pale and stale” (Sorry, but there’s not much I can do about that).

My Experience is Old

I first wrote code for an IBM 1130 16b minicomputer (FORTRAN IV, punch cards, mid 1970s)

I started full time work in 1979 at Inmos

Much has changed in 40 years

But, no-one (without a time-machine) can have had the experience you will have, so we have to look backwards!

What Have I Done?

Inmos (1979-1984)
- Worked on CAD tools for the Transputer (“Fat Freddy”)
- Wrote the microcode assembler, bits of compilers, OS, …

Meiko (1984-1995)
- Wrote a code generator in BCPL for a Fortran compiler targeting the Transputer
- Represented Meiko on the HPF forum and the MPI-1 forum
- Caused profiling interfaces to be in MPI (and owned that chapter)

BBN/Etnus (1995-2005) (from Bristol reporting to Cambridge,MA)
- Worked on parallel debugging (“Totalview”)

Intel (2005-2019) (from Bristol reporting to US Central Time)
- OpenMP® runtimes
- Transactional memory
- Etc…

What Did I Achieve?

MPI profiling interface

- A hack, but a good one
- I reckon it has paid my CO_2 dues by enabling tuning of MPI codes

OpenMP stuff

- Speculative locks
- `monotonic/nonmonotonic` schedule distinction

Made some hardware architects understand shared-memory parallel processing a bit better

Intel Senior Principle Engineer; ACM Distinguished Engineer

Worked on fun things, had fun, helped people (I hope!)

What Changes and What Doesn't

Hardware Changes

Then:

- Memory was near (1 cycle away)
  - No need for caches
- Cores were simple
- Dennard scaling applied

Now:

- Memory is a l..o..n..g way away (and not all equally far)
- Caches matter hugely
- Cores are big, complicated, and trying to guess what you want to do
- Dennard scaling is dead

Dennard Scaling

The good part of Moore's Law (the physics!)

For MOS **when there is no leakage:**

When the transistor density doubles, clock rate can go up 1.4x, while power consumption stays the same (despite having twice the number of transistors running faster).

Ceased to apply ~2005 (because leakage became significant)

$\Rightarrow$ Clock speed bounded

$\Rightarrow$ Energy consumption becomes critical

$\Rightarrow$ Drive to many core machines

Hardware Constants

The speed of light (1ft ~= 1ns)

The performance limits are still often not the ones people expect

- To first approximation, “It’s always data movement”

Some hardware folk still say

- “It’s only software” or “That’s a SMOP (Simple Matter Of Programming”)

And we now have more history to prove that they’re still wrong!

Software Changes

Then:

- The BCPL compiler could bootstrap itself in 64KiB
- Linux did not exist
- C (derived indirectly from BCPL) was a new language
- Register allocation was an unsolved problem
- Source code control was a new idea (SCCS)

Now:

- LLVM is > 5.5M lines of code
- Compilers are smarter
- We have 100s of programming languages (Julia, Chapel, …)
- We have `github`
- Python is way better than BASIC

Software Constants

Fortran still matters (more later)

Some of the same code is still running in important applications

Compilers and runtime systems remain important

Software is still fun to work on and develop

Things Get Forgotten/Ignored

We had Occam (based on CSP) in 1984

- Parallel language
- Deterministic (until you use ALT, so you can see the non-determinism)
- Naturally supports clusters

Now we have

- OpenMP: subject to races, doesn't support clusters
- MPI: a library rather than a language
- But Golang and Rust may be moving in the right direction ☺

Useful Questions to Ask Yourself

Questions:

Why am I doing this?
What should I be doing?
Who is the customer (who is paying)?
Which metrics matter?
What do I want to do? (Do I want to manage people?)
How do I become visible?
How much work do I have to do?
How do I plan my career?

Example:

Why am I optimising this code?

Possible answers:

- My manager told me to. (But why did she do that?)
- We need it to run fast. (Who is “we”?)
- It’s an interesting problem I’ve snuck the time to look at.
- A manager’s bonus five levels above me depends on it.
- We’ll sell more compilers if this works better.
- More customers will buy our machines if this benchmark runs well (according to a salesperson…)

What is the Real Problem?

Do we need to make this function run faster?

Is there a better way to solve the real problem?

Has someone else already solved the real problem?

E.g.

Don’t waste time improving a hand-written linear algebra routine; instead use an optimized BLAS

Who is Paying (and for What)?

Follow the money!

In industry this is the ultimate answer…

- What you are doing must add value that causes someone to pay more for something your company is selling
- That may not directly be your code, though!

Who is Really the Customer (1) ?

Q: Why are supercomputers like dog-food?

A: Because in neither case does the ultimate consumer make the purchasing decision.

- User is programmer/scientist
- Purchaser is lab-director/university/funding agency/politician who wants a braggable entry on the Top500 list
- This is why we have perverse hardware with too much FP and not enough memory BW

Metrics matter and have unforeseen consequences.

Who is Really the Customer (2) ?

Consider the compiler team in a hardware vendor

- The compiler is given away for free (or close to free)
- How is working on it justified?
- Who is paying for all the people working on it?

Enemies and Opponents

- “Your enemies are on your side; those are your opponents”

Compiler team funding depends on internal company politics; they must sell their work to management.

We're Getting to "The Five Whys"

Developed by Toyota to root cause problems

Normally asking "Why?" five times takes you high enough…

Even once is useful, e.g. when answering StackOverflow questions:

- "My OpenMP code fails like this…"
- The problem is very often
  - "I have written bad code to do something that OpenMP or a library could do much better."
  - e.g. reductions, linear algebra operations, …

The right answer is "Don't do that."

One Advantage of Interpreted Languages

No sane person will write their own BLAS routines in Python!

Better integration of libraries into the language

Expectation that there will be a library to do (at least some of) what you want

"The best code is the code you do not have to write."
"A couple of months in the laboratory can frequently save a couple of hours in the library."

Which Metrics Matter?

This is a general question, we'll look at programming languages as an interesting example…

Possible metrics:

- (Most obvious, easy to find): Number of programmers/fashionability (e.g. Tiobe Index, Github Language Popularity Index)
- (Less obvious, harder to find): Amount of CPU resources consumed by code written in the language

The answer is that for all HPC questions: "It depends…"

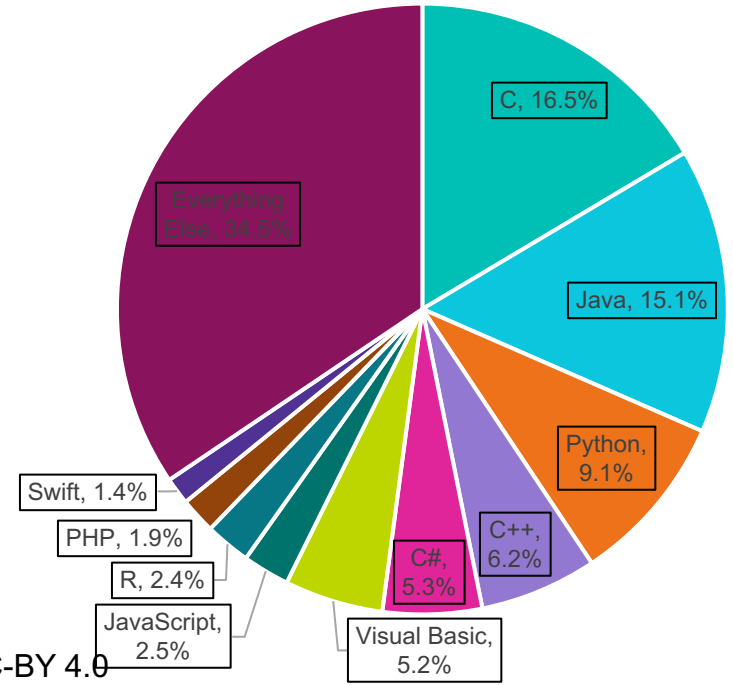
Programming Language Metrics

Tiobe Programming Community Index
Q2/2020

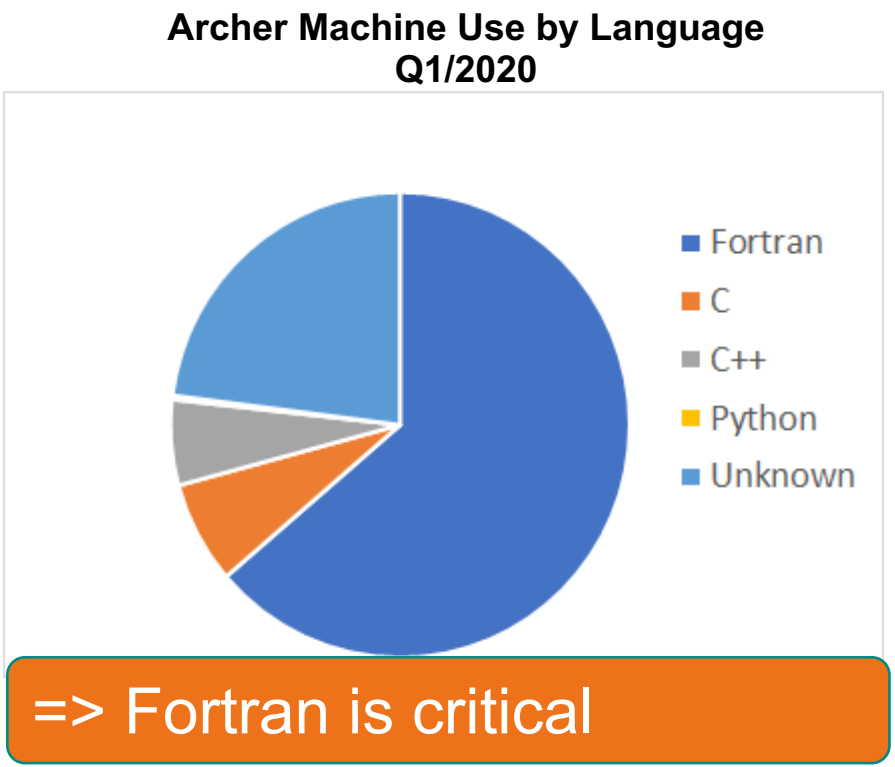
C is first,
C++ is fourth,

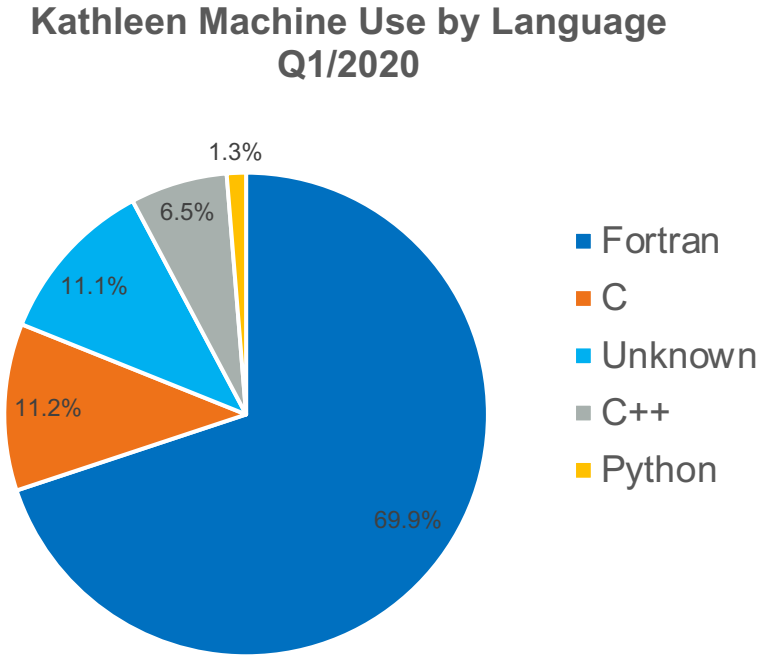
Fortran is … fiftieth
(below Ada, Haskell, Lua,
“Classic Visual Basic”, …)

=> Fortran is irrelevant, right?



UK HPC Machine Use





=> Fortran is critical

Where do HPC Hardware Vendors need to Focus?

Archer has 4920x2socket nodes => 9840 sockets

- 60% of that => 5904 sockets running Fortran!

If Archer2 use is the same, then 60%x5848x2 ~=7000 => ~$35M

Kathleen has 192x2socket nodes => 384 sockets

- 70% of that => ~270 sockets running Fortran!

So Fortran isn't dead, but instead it's important.

=> Fortran programmers and compiler/tools writers will be needed (and probably in short supply…)

Who is Your Enemy?

The person who keeps submitting annoying bugs in your code?

- Absolutely not!
- They are using your code and care enough to help you to improve it

That team at *other university* who are working on the same problem and might publish first?

- What would the funding agency who is funding you both like?
- How can we solve the science problem sooner?

Respect your opponents, try not to have enemies.

Open Source or Not?

In industry

Advantages:

- Open sourcing code provides leverage and a shared codebase that can be jointly maintained (no sane company wants to maintain 5M lines of code!)
- Easy to find people to employ who already know the codebase
- Can use other people's code (e.g. ANL developing Clang code)

Disadvantages:

- License must be appropriate (BSD-style, not GNU style)

Open Source in Research

Advantages:

Reproducibility

- Becoming more and more important. Conferences/journals require reproduction “artefacts” to enable others to verify results.

More eyes on the code

Allows co-operation and avoids time spent re-inventing a square wheel

Eases commercial spin outs (=> use a BSD-style license)

Reduces funding cost if multiple teams need not write equivalent code

Open Source in Research

Disadvantages:

I don't want people to see my horrid code…

- Well, don't write horrid code!

That group at *other university* might use it, and they **are** my enemy because they're competing for the same funding.

- Who is paying you both? What would they prefer?
- Where might you want to work next?

I'm hoping to start a spin-out and license the code from the university so I don't want everyone to have my code

- Wouldn't you rather have the code for free and have others help you to develop it? (A BSD-style license lets you keep some code secret).

How Can I Grow?

Become more visible

- Decide that you are not as introverted as you thought you were
- Publish your code
- Help people to use it

If you're using open-source code

- Report bugs
- Help to fix things
- Push your fixes/enhancements upstream

Think about how to present results at the right level

Contribute to "industry" (OpenMP, MPI, SYCL,…) and "real" standards (C++, Fortran, …)

Join professional organisations (ACM, BCS, Society of Research Software Engineering)

Standards

Sound frightening; lots of weird experts debating intricate details of OpenMP or C++ or …

But

- They need contributions
- You don't have to speak out until you are ready
- It's valuable, visible, work
- They recognise that they are also too "pale, stale and male"

You can (and should) sell this to your organisation:

- "We rely on this technology so need to know what is coming and influence it"
- "We need open standards to avoid vendors getting their hand in our wallet!"

You do **NOT** Need a Plan

I never had one

I followed my nose and took interesting jobs when people approached me (or asked people I was already working with on one occasion)

Do stuff which interests you (you do not have to be a manager)

Get involved in standards

- The community needs input
- It will make you friends for life (that's how I know Jack Dongarra, Bill Gropp, Rusty Lusk, Torsten Hoefler, Michael Klemm, …)

You Should **NOT** Need to Work All the Time

Keep work on work machines

Don’t install work email/slack/… on **your** phone

Keep the work machine in the office (try to have a separate office even if you are working from home)

Don’t turn the work machine on other than during office hours

Don’t agree to meetings at stupid times even if you work for a US company

Unless you really want to, don’t work on SW in your own time

Conclusions

Be helpful

Don’t be tribal

Respect your opponents

Don’t have enemies

Do interesting things

Decide you can speak out in public

Follow your nose

Try to Follow Dr Phil’s **CLANGERS**

Connect

Learn

(be) **A**ctive

Notice

Give back

Eat well

Relax

Sleep

Have Fun, do What You Enjoy. Good Luck!

Questions, Discussion, Why I am an idiot…

Backup

Interesting Languages (in HPC)

- One new language: Julia
  - Can run interactively (“Jupyter”)
  - Can get compiled-language performance
  - Less fashionable than Python/R but on the rise
  - Could be a language to get in on early
- One old language: Fortran
  - Still hugely important, as we saw:
    - VASP is #1 consumer of cycles on Archer
    - UK Met Office code is all Fortran
  - Not fashionable, but that’s potentially good:
    - If you are one of 10 experts that’s better than being one of 1000
    - There will be high demand for Fortran compiler experts

Other Random Things to Think About

- Work out how to present your results in a way which the target audience will understand
  - Not: “I halved the time spent in OpenMP dynamic scheduling by 50%.”
  - But: “*Important code* now runs 5% faster than before.”
- Read Tufte (“The Visual Display of Quantitative Information”) and follow his rules
- Never use `omp_set_num_threads(`*`constant`*`)` that implies
  - You don’t expect the code to run on anyone else’s machine
  - You expect to throw the code away before you replace your machine

References

Dennard Scaling: https://en.wikipedia.org/wiki/Dennard_scaling

Tiobe index: https://www.tiobe.com/tiobe-index/

Github language popularity: https://madnight.github.io/githut/#/pull_requests/2020/2

LLVM size: David Wheeler's `sloccount` (https://dwheeler.com/sloccount/) on llvmorg-9.0.0.

```
Total Physical Source Lines of Code (SLOC) = 5,657,032
```

Library/Lab quotation: Frank Westheimer

Archer usage data: Adrian Jackson (Thanks!)
https://twitter.com/adrianjhpc/status/1281314767563558912

Archer2 spec: https://www.archer2.ac.uk/about/hardware.html

References (2)

AMD Rome 64C price approximation: https://www.tomshardware.com/uk/news/amd-epyc-rome-2-cpu-price-specs,40119.html (I used $5000/socket, despite “As for the flagship 64-core models, Newegg listed them between $6,650 and $7,220 .”)

Kathleen usage data from Owain Kenway (Thanks!)

CLANGERS: www.drphilhammond.com

ACM: https://www.acm.org

Society of Research Software Engineering: https://society-rse.org/

Tufte: “The Visual Display of Quantitative Information” https://www.edwardtufte.com/tufte/books_vdqi