Three Lessons from Three Decades in Engineering

In 2020 I was invited to give an award lecture in which the awardee is asked to touch upon their career path. It was the first time I had to face myself as a writing subject and confess it was a bit uncomfortable. In the end I was able to summarize much of what I learned in conducting my career into the three lessons that make up the rest of this essay. I am sharing this in case they could serve as useful points of reflection for those earlier in their careers.

Consider the winding road

Our field, like many other technical disciplines, becomes richer and also more fragmented into sub-areas as our knowledge improves. There is always so much to learn about each of those branches of knowledge that many successful careers are built on continuing to go deeper and farther in a given area. An engineering or computer science education enables to follow that path but it also gives you a
foundation of knowledge that enables you to branch into many different kinds of work. Although there is always risk when you take on something new, the upside of being adventurous with your career can be amazingly rewarding.

I for one have never let my complete ignorance about a new field stop me from giving it a go. As a result I have worked in areas ranging from chip design to datacenter design; from writing software for web search to witnessing my team launch satellites into space; from working on Google Scholar to using ML to automatically update Google Maps; from research in compiler optimizations to helping deploy exposure notification technology to curb the spread of Covid-19[5].

It seems a bit crazy, but not going in a straight line has worked out really well for me and resulted in a rich set of professional experiences. If you choose that path, whatever the outcome, I guarantee you will be inoculated from boredom.

Develop respect for the obvious

The surest way to waste a career is to work on unimportant things. Having said that, it is far from trivial to know for certain when you begin working on a given problem whether it will turn out to have been relevant or not. I have found that many big, important problems have one feature in common: they tend to be straightforward to comprehend even if they are hard to solve. Those problems stare you right in the face. They are obvious and they deserve your attention.

Let me give you some examples by listing some of my more well cited papers next to the formulation of the problems address:

Publication	Problem addressed
ISCA'98: Memory System Characterization of Commercial Workloads With Kourosh Gharachorloo, and Edouard Bugnion	"High-end microprocessors are being sold to run commercial workloads, so why are we designing them for number crunching?"
ISCA'00: Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing With Kourosh Gharachorloo, Robert McNamara, Andreas Nowatzyk, Shaz Qadeer, Barton Sano, Scott Smith, Robert Stets, and Ben Verghese	"Thread-level parallelism is easy. Instruction level parallelism is hard. Should we bet on thread-level parallelism then?"

[5] g.co/ENS

CACM '17: The Attack of the Killer Microsecond With Mike Marty, Dave Patterson and Partha Ranganathan	"If datacenter-wide events run at microsecond speeds, why do we only optimize for millisecond and nanosecond latencies?"
CACM '13: The Tail at Scale With Jeff Dean	"Large scale services should be resilient to performance hiccups in any of their subcomponents"
IEEE Computer '07: A Case for Energy-proportional Computing With Urs Hölzle	"Shouldn't servers use little energy when they are doing little work?"

If it takes you much more than a couple of sentences to describe the problem you are trying to solve, you should seriously consider the possibility of it not being that important to be solved.

Even successes have a "sell-by" date

Some of the most intellectually stimulating moments in my career have come about when I was forced to revisit my position on technical matters that I had invested significant time and effort on, especially when the original position had a track record of success. I'll present just one illustrative example.

I joined Google after a failed multi-year chip design project and as such I immediately embraced Google's design philosophy of staying away from silicon design ourselves. Later as the technical lead of Google's datacenter infrastructure, I consistently avoided using exotic or specialized silicon even when they could demonstrate performance of efficiency improvements for some workloads, since betting on the low cost base of general purpose components consistently proved to be the winning choice. Year after year, betting on general purpose solutions proved successful.

Then Deep Learning acceleration for large ML models arose as the first compelling case for building specialized components that would have both broad applicability and dramatic efficiency advantages when compared to general purpose designs. Our estimates indicated that large fractions of Google's emerging AI workloads could be executed in these specialized accelerators with as much as a 40x cost/efficiency advantage over general purpose computing. That success had expired.

With the help of the early Brain team we let go of our past successful CPU-only strategy and began building the hardware team that was to design Seastar, our first TPU. Almost ten years later TPUs remain a key differentiator for Google in ML-based products.

Conclusion

I should disclose that some of the lessons listed above only became clear to me when looking back so it would be problematic to claim that they really guided my journey but they are consistent with career choices I have made over the years. In the end, whenever I had the luxury of choices on what to work next I almost always picked the most fun, so it is possible that there is only one lesson to be learned here after all. I will close by saying that the effect of luck and the help of others on anything I've accomplished remains unwritten here, but they had as much of an impact as any efforts of my own.

luiz@barroso.org
2020