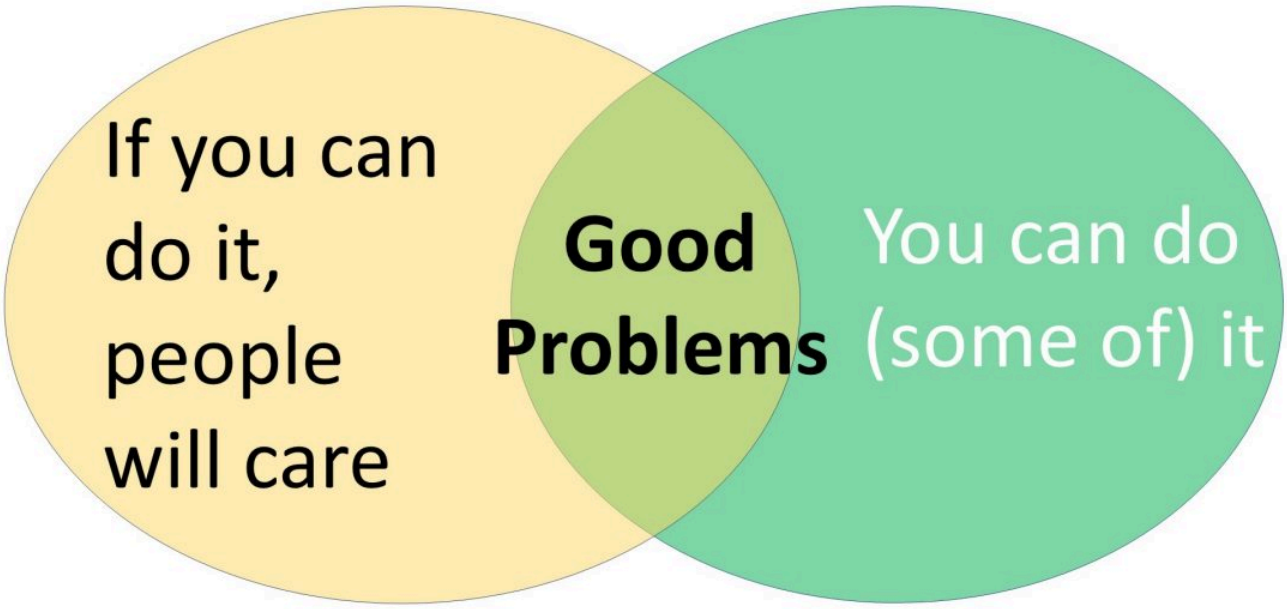
Computer Architecture Today

Informing the broad computing community about current activities, advances and future directions in computer architecture.

Increasing Your Research Impact

by Mark D. Hill on Aug 12, 2019 | Tags: Advice, Scientific Method



Many works present **their results**; this blog post seeks to aid you in developing **your own great results, especially in computer architecture and systems**. I learned these lessons over a career leading to an Eckert-Mauchly Award (acceptance speech). I structure this blog post with the scientific method in four steps:

1. Pick a good problem.
2. Develop insights and first hypotheses.
3. Test and refine hypotheses.

4. Repeat steps as needed.

Pick A Good Problem

The first step to developing creative and important influential research is picking a problem from an infinite set of possible problems. As illustrated the Vend diagram at the start of this blog, good research problems fall in the **intersection** of two criteria, "***If you can do it, people will care.***" and "***You can do (some of) it.***" Groundbreaking research rarely comes from the temptation to play Jeopardy!, where one starts with an answer–my cool mechanism–and then seeks questions it answers.

Consider, for example, the influential development of redundant arrays of inexpensive disks (RAID) that I observed in graduate school. You might be impressed with RAID 5's novel use of rotated block erasure codes, but the real genius of this research was recognizing the problem and the opportunity it presented. The sales volume of PC hard disks made them a more cost-effective way to store large data than the historical solution of using large washing-machine-sized disks. PC disks could not be used to store large data, however, because a single PC disk was only somewhat reliable and using an array of them is much worse. Therefore, the authors sought a solution to make disk arrays more reliable, ergo RAID.

As happened with RAID, I recommend seeking problems where **change** affects what is best. In computer architecture and systems, these opportunities can occur due to new software (e.g., for augmented reality), new technology (PC disks or non-volatile memory), or advances in other (sub-)fields (using ML to optimize hardware).

I also recommend seeking problems by connecting with industry through affiliate meetings and sabbaticals. As discussed later, the former led to GEMS and the latter to Gables.

Develop Insights and First Hypotheses

Related to picking a problem, good research must begin by searching for a solution with initial insights and hypotheses. While simulators dominate the methods sections of computer architecture papers, I have found this beginning step is well served by simple taxonomies, connections, and models.

Taxonomies organize a problem or existing solutions into "boxes" that can expose new opportunities. Science's most famous taxonomy is Mendeleev's **Periodic Table** that focused efforts on uncovering missing elements. Well-known in computer architecture is our **3Cs: compulsory, capacity, and conflict misses**. We developed this taxonomy by seeking simple insights from the cache simulation data deluge. As a third taxonomy

example, consider Log-based Transactional Memory (**LogTM)**, which has influenced the literature more than products (so far). We sought a TM system that could commit any transaction (improving on Herlihy & Moss) but only modestly changed conventional hardware (improving on Stanford TCC). We began by developing a two-by-two taxonomy with where new and old values get stored in one dimension and with where transactional conflicts were detected in the other. We recognized that one box corresponded to commercially-successful database management systems, so, therefore, decided to target that box, and LogTM resulted.

Connections are key to research. Steve Jobs said, “Creativity is just connecting things.” I have sought connections by working with professional colleagues and students. All my ideas have benefited from my 160 co-authors. ***Don’t unduly worry about dividing credit, but generously share credit as it often multiplies and leads to collaborations that can enable something even more worthwhile.*** Connections also occur by attending talks. At a Bart Miller talk on software datarace detection, we glimpsed a connection between dataraces and weak/relaxed memory models. With much hard thinking, we made the connection simpler and explicit with the “Sequential Consistency (SC) for Data-Race-Free (DRF) Programs” memory consistency model. **SC for DRF** has subsequently influenced CPU and GPU hardware and is the foundation of the C++ and Java memory models.

Models are also key to initial thinking. Good models focus on the essentials by discarding the less important. Amdahl’s law is the most widely-known model in computer architecture. I advocate **Bottleneck Analysis and Little’s Law** in a recent SIGARCH blog post and the **M/M/1 Queue** in another. During a recent sabbatical at Google, we developed the **Gables** model for Mobile Systems on a Chip (SoCs). Gables resulted from trying to make sense of Mobile SoC hardware with CPUs, GPUs, and dozens of accelerators having to acceptably run each of 10-20 “use cases,” such as video recording or playback. George Box said, “All models are wrong; some are useful.” Only time will tell whether Gables is useful, but it has already exposed Accelerator-level Parallelism that we conjecture will expand to many computer systems. Note that the most appropriate model varies from mental (e.g., 3Cs) through envelope (Amdahl) to computerized (Gables).

In my experience, the best research is simple. I follow Einstein: “Everything should be made as simple as possible, but not simpler.” Simplicity facilitates bolder work by eschewing the less important. Moreover, even simple computer architecture and systems ideas get more complex when actually deployed in products.

Test and Refine Hypotheses

The scientific method promotes **structured thinking**:

```
Repeat until done {
      Develop one or more new hypotheses (by thinking),
      Devise one or more experiments to test hypotheses (by thinkin
      Predict experimental outcomes (by thinking), and
      Run experiments and observe how actual outcomes different fro
}
```

Strong Inference argues for developing multiple hypotheses—to promote concurrency and mitigate becoming attached to "your" hypothesis—and devising experiments that seek to falsify hypotheses, not just gather data. Since simulation experiments are fast, computer architects are particularly prone to the pitfall of running experiments first and developing hypotheses after. This pitfall leads to scattershot experiments and no falsifying of hypotheses. On the other hand, Gregor Mendel thought carefully when founding Genetics, because his experiments of growing peas took time.

While taxonomies, connections, and models are good for initial thinking, **simulation**—and sometimes building hardware—are appropriate for the experiments that falsify or refine mature hypotheses with many known and unknown factors. Nevertheless, one should think carefully about the simulator you plan to use or build. Circa 2000, most simulators were user-level only, while feedback from industrial colleagues called for running commercial workloads on an OS including I/O. For this reason, we set about building **GEMS**. To ease work and speedup development, we were "creatively lazy" and implemented the performance part of GEMS while using initially-beta commercial SimICS to do the hard functions of booting an OS and supporting devices. This symbiosis was a great success but limited our ability to share with others. Fortunately, the (former) Michigan folks had implemented full-system functionality in **m5** and proposed that we merge with them to form **gem5**. As the right level of simulator for many tasks, gem5 is now widely used in both academia and industry.

Repeat Steps as Needed

Good research rarely takes a fast path to solutions — the path that is presented afterward in the papers. Rather, one meanders around chasing some dead ends and having to balance persistence and flexibility, as we did for LogTM variants. Making things simpler takes time, as occurred for SC for DRF. And some work fails to be publishable, e.g., my contributions to SIMD cache simulation, acoherence [sic], and probabilistic writebacks. I recommend: (a) use the scientific method to conduct searches with rapid and firm steps and (b) know that you are not alone, as all research requires setbacks. On the challenges

of developing the light bulb, Edison is alleged to have replied, “I didn’t fail 1,000 times. The light bulb was an invention with 1,000 steps.” (Some sources say 10,000 times.)

Acknowledgments

This post’s advice is channeled through me but comes from my Ph.D. co-advisors David Patterson (see “How to Have a Bad Career“) and Alan Jay Smith, my 160 co-authors, and many others that I have interacted with in academia and industry. I have been largely supported by the U.S. National Science Foundation, most recently with grants CCF-161782, CCF-1734706, and CNS-1815656.

About the Author: *Mark D. Hill is John P. Morgridge Professor and Gene M. Amdahl Professor of Computer Sciences at the University of Wisconsin-Madison. He is the 2019 Eckert-Mauchly awardee, is a fellow of IEEE and the ACM, serves as Chair of the Computer Community Consortium (2018-20) and served as Wisconsin Computer Sciences Department Chair 2014-17.*