On the realizability of hardware microthreading. Revisiting the general-purpose processor interface: consequences and challenges

Poss, R.C.

Publication date
2012

Preface

Zonder verhaal zijn feiten sprakeloos.

Karel van der Toorn

When I joined the CSA group in September 2008, my supervisor Chris Jesshope tasked me thus: *"We made this novel chip called a Microgrid, and we want a C compiler to program it. Do it."*

This is most of what this book is about. You are now holding a report on four years worth of work towards compiling and running C programs on Microgrids, which are multicore, multithreaded processor chips. I am proud and content to report that a C compiler now exists for this architecture. The following pages will hopefully convince you that the corresponding software technology and example uses constitute an adequate answer to my supervisor's request.

Meanwhile, these four years have been intellectually instructive.

The first and most prominent discovery was that these Microgrid chips, that needed a compiler, lacked a specification when my work started. As I learned quickly, there are two meanings for the word "exist" in computer architecture. One definition, commonly used by the layman, suggests a working physical artifact. This definition, however, is quite uncommon in the research field: most chip architectures "exist" instead as *models* (blueprints) that can be studied analytically or via software simulations. The Microgrid architecture obviously did not exist in this first sense, as it was not expected anyway. My surprise, instead, was to realize that Microgrids only "existed" in the other sense in the *mind* of my colleagues. Their existence was vernacular, shapeshifting, and lacked committed documents that I could study and refer to. The second, most exciting discovery was that my colleagues had not invented *one* chip design but instead a *class* of chip designs, which could be instantiated over a range of design parameters. Therefore, to make a compiler, I first had to capture my colleague's minds on paper, then understand the architecture class so that a C compiler could target all its possible instances.

On this quest to gather such a *generic platform specification*, a prerequisite to implement a compiler, I made two further discoveries:

- previously written information about the new architecture was inconsistent, i.e. it expressed conflicting ideas;
- the unwritten intellectual model manipulated by my peers, while relatively consistent, was largely insufficient to implement a C compiler and accompanying software. For example, it missed the concept of external input and output.

In other words, the task initially assigned to me required that I first document the meta-architecture and then an instance thereof in sufficient detail. It also required that I complement the model with additional features required by the new software. While doing this, I had to accommodate the fact that some system features required to compile the C language are not technically part of a compiler nor the hardware architecture, but rather the "environment" provided by a software operating system. Therefore I had to also contribute some additional technology in that direction. All this happened, although as you would probably agree these steps are somewhat difficult to capture behind a simple "problem statement" such as found in most doctoral theses.

When I wrote the first version of this book, its narrative was similar to the steps outlined above and amounted to a straightforward technical report. When I initially submitted this report, my supervisor rejected it. He argued that the *scientific* community would not be interested in an engineering story; that they would rather like to understand which *new knowledge* I had created on my way. So I sat and mulled over the report. What did I learn during this work? What new knowledge did I create that was worth sharing?

At the foreground, surely there was some knowledge to share about the creation and exploitation of a C compiler for a new architecture. So I left that in, to be found in chapters 6, 8 to 11 and 13. Yet I found that sharing *only* this knowledge was oddly unsatisfying, because it depended on knowledge about said new architecture that was either unpublished or inconsistent, and somewhat incomplete. Therefore, I decided to *also* share the detailed knowledge I gathered from my colleagues about their Microgrid meta-model, complemented by details about the different concrete instances that I used in my work. And so I added that knowledge to the text, to be found in chapters 3 to 5. As a welcome side effect of this excercise, I was also able to argue towards a *renewed motivation* (chapter 2) for the architecture design, to complement the 15 years old argument [BJM96] that had so far served unchanged as justification for the line of work.

When this was done, I mulled again: although I had formulated new knowledge and made a scientific contribution to my field, I had not yet done justice to what I had found most important and fundamental in those four years. For there were two other aspects, which I dare call "knowledge" although my supervisor calls them "experience," that I had also isolated. The first is my understanding of the thrust of practitioners in my field: what *innovation* really means in computer architecture and why it is so crucially important to the future of computing. I share my views on this topic in chapters 1 and 16; this explains why you should care about my story, even though you may not be interested in Microgrids specifically. The second is my understanding of an obstacle shared between computer architects working on multi-core chip designs, that is the necessary disruption of unspoken knowledge.

Beyond the immediate need for more performance per watt, the role of a processor chip designer is to provide components to other humans to use in larger systems; as such, there is an iterated step of human-to-human communication required between component architects, system architects and software practitioners. Due to the largely technical nature of the devices considered, this communication is necessarily dense; to reduce verbosity, the people involved rely on unspoken, tacit knowledge constituted by their cultural background in computer science. *Innovation is difficult because of this reliance on tacit knowledge.* Any novel component will break some unspoken assumptions and invalidate some commonly accepted "truths." Innovation thus requires formulating a new cultural background about

hardware platforms, a step which most audiences—especially in the software world—are ill-trained to perform. In the case of Microgrids, my colleagues had already discovered that the mere prospect of additional performance is not enticing enough when its cost is a necessarily more complex mental model of the machine. What I have understood however, is that software audiences are willing to pay the price of more complex models if *they believe that they gain in flexibility*, i.e. future increases in productivity. There are two requirements to support this belief. The first is that the chip designer *does not gratuitously remove features* previously relied upon; the second is that the chip designer argues for the *generality* of any new features, to convince software audiences they can be used in ways not yet foreseen. As I found out, these two requirements were previously not very well addressed in the work around Microgrids. To my colleagues' defense, neither are they in other research projects in computer architecture. Nevertheless I believe they are still essential to the eventual fruition of innovative research in the field.

To advertise this experience, I thus decided to spend extra care to:

- detail the proposed architecture from the perspective of system software (chapters 3 and 4), highlight some of the most affected assumptions about the underlying machine model (chapter 7), and suggest how features commonly found in other platforms are also found in the proposed design (chapters 5 and 14);
- argue for generality throughout, starting with a renewed defense of general-purpose computing (chapter 1) and placing emphasis on the potential use of the platform by contemporary concurrent programming models (chapters 7, 9 and 12).

With this extra effort I want to communicate that the concepts around the advent of Microgrids are not merely the bread and butter of a small research group; that this research answers general requirements that deserve attention regardless of which specific architecture they are considered from. This is why chapters 3, 4 and 9 are longer than strictly required by a platform specification towards a C compiler, and why chapters 5, 7, 12 and 14 do not fit the purely technical line of thought of the other chapters. As I explain in section 1.7, this "meta-thesis" provides a second reading level, distinct from the technical contribution at the first level.

To keep a long story short, if you wish to restrict your reading to the joys of compiling C towards Microgrids, you can narrow down your focus to chapters 6, 8 to 11 and 13. The rest is "simply" *philosophy*.