

MICROPROCESSOR PERFORMANCE, PHASE 2 HARNESSING THE TRANSFORMATION HIERARCHY

Yale N. Patt

Department of Electrical and Computer Engineering
The University of Texas at Austin
Email: patt@ece.utexas.edu

ABSTRACT

We are witnessing a substantial shift in the way we pursue performance improvement in the world of microprocessors. The past has been characterized by performance improvement mostly below the hardware/software interface, transparent to the executing software. The future will be characterized mostly by breaking through that interface and harnessing all levels of the transformation hierarchy.

1. PRE-INTRODUCTION

I have been asked to write a paper expressing my views on the future of Computing for this event commemorating and celebrating the life of Stamatis Vassiliadis. This paper will discuss my views of the Future within the context of the high performance microprocessor, something I have worked on and thought about for a good number of years.

But, I am unable to start this paper without first reflecting on Stamatis, since indeed, his influence permeates much of what we do, think, and dream as computer architects. I remember our first interaction, at ISCA 1995 in Santa Margherita Ligore, where at a tutorial I was giving, two young engineers from Company X were quarreling with one of my definitions during the coffee break, clearly spouting the liturgy of the Marketing Department of their company. Stamatis, who I did not know at the time, stood close by and listened patiently while I tried to explain to them what was true, independent of what their Marketing Department wished was true. Finally, Stamatis politely interrupted, and with that soft voice and gentle but firm manner told them they would do well to listen “because Professor Patt was trying to teach them something fundamental” [1].

My fondest recollection of our interactions came in Delft three years later after I agreed, at his insistence, to spend a week at TU Delft. In Delft, every day we argued computer architecture, drank beer in the town beer garden, and ate Greek food in this Dutch city. Through it all, I appreciated more and more the cornerstones that guided Stamatis’ professional life [2]:

- Do not hide your lack of understanding. Only by bringing it out in the open does one have a chance of correcting ignorance.
- Be scientific, testing fundamental concepts, and never blur the experiment for the short-term goal of carrying the day.
- Think beyond the horizon, while acting within what you understand.
- Enjoy the miracles and wonders of life, never forgetting that intense argument should be followed always with a joke and a pint of good beer.

In hoping to follow Stamatis’ lead, I have been torn between making this paper about (1) the lack of science in our discipline and the willingness of too many to “cook the books” to produce an accepted paper, and (2) the future of the microprocessor ten years from now, and the changing paradigm afoot. In the spirit of Stamatis, I have chosen the latter. The lack of science I will get to in some other venue. It is serious, it is problematic, and Stamatis would not have me ignore it. But for this volume, I would rather think positively and cherishing his always upbeat spirit, I prefer to discuss where we are heading and what is possible, based on our ability to understand where we are.

2. PHASE I: STAYING WITHIN EACH LEVEL OF THE TRANSFORMATION HIERARCHY

Figure 1 shows the transformation hierarchy. It captures what has been inherent in computer processing for a very long time. In an effort to manage complexity, we have created walls between the layers in the hierarchy. So, for example, someone working in microarchitecture understands the circuits available as resources, and understands the need to implement the specification of the ISA. The job of the microarchitect is to stay within the barriers above and below, and use the circuits available to produce a microarchitecture that implements the specified ISA.

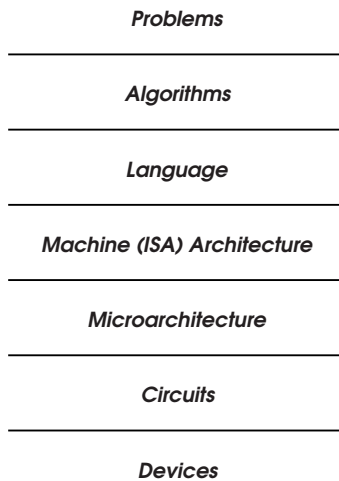


Fig. 1. Levels of Transformation

Similarly, someone working at the mechanical language level understands the constructs available in the mechanical language, and understands the need to implement the algorithm specified. The job of the programmer is to stay within the barriers above and below, and convert the algorithm to a mechanical language implementation.

Since Intel first produced the 4004 in 1971 to the present day, the number of transistors on the chip and the operating frequency of the chip have both grown exponentially. The Intel 4004 had 2300 transistors and ran at 105 KHz. The Intel Pentium in 1992 had 3.1 million transistors and ran at 66 MHz. Today's chips contain hundreds of millions of transistors and run at frequencies in excess of 2 GHz.

This increase in resource capability, along with increased knowledge of the benefits of microarchitecture, have resulted in exponential performance benefit in the performance of the chip. Most importantly, this increase in performance has been accomplished mostly transparent to the software. That is, the chip has been able to do better, but the software has not had to pay attention to why that has been happening. That is, the barrier between what is done at the software level and above (in Figure 1) has, for the most part, not had to be broken to achieve this performance benefit.

In an earlier paper [3], I identified a number of what I called Agents of Evolution which I insisted were responsible for the performance benefit. Usually that agent involved some existing bottleneck and the fix that eliminated it. In that light we added pipelining to the chip, which enabled the need for branch prediction, which in turn enabled the need for speculative execution. We added a small instruction cache, followed by separate instruction and data caches, then a second level unified cache to further mollify the discontinuity between L1 access time and off-chip access time.

In some cases, the Agent was simply good fortune caused by the availability of available chip area due to a shrink in feature size. This was responsible for the inclusion of the floating point unit on the Intel 486 chip and the addition of MMX to the Pentium chip. More recently we have seen wide issue, out-of-order execution, and trace caches. In almost all cases, these features enhance the microarchitecture with no requirement that the software even know about it.

3. PHASE II: BREAKING THE BARRIERS OF THE TRANSFORMATION HIERARCHY

Phase II of microprocessor performance improvement has arrived, I would argue, due to the enormous richness given us by process technology and our inability to harness this richness in the ways we have done in the past. For example, we will soon have ten billion transistors on a chip, with operating frequencies of more than 10 GHz.

If we follow our past approaches, we will need power consumption closer to a jet engine than to what can reasonably be expected from a laptop's power supply. Furthermore, the operating frequency of the chip will be so much higher than that of memory that the memory system will be unable to supply the instructions and data needed to keep the processor working at its rated capability. Finally, utilizing this transistor count on behalf of one or even a few cores would mean a very complicated design, probably not one that the designer will get right in the first place, and if he/she did, verifying it would be a nightmare.

I would argue that this is not the end of the road, that there is much we can do, and it starts with breaking through the artificial barriers that separate the layers of the transformation hierarchy. Those barriers were useful in the past when we needed to manage the complexity of the entire task of transforming a problem in natural language to the electrical circuits that did the work with the level of understanding we had then. Today, I believe those barriers get in the way, and breaking through them can create opportunities. Some examples follow.

3.1. Holistic Design

The Intel 486 included the floating point unit on-chip because the 486 had one million transistors available, and the floating point unit could easily be accommodated within that budget. With ten billion transistors on a chip, there are ten thousand times as many transistors available as was the case with the 486. I would argue that this suggests lots of special purpose functionality that is available when needed to perform specific time-consuming tasks. The find-first instruction in the AMD 29000 is an early example (~1985) of this. It identified the first bit set in a bit vector. To determine this in software would have taken tens of cycles. To do it in logic required a very simple priority encoder.

A key requirement of this concept is that the functionality remains powered off when not needed. The idea is that the algorithm person knows that the functionality is available on the chip, and specifies when to use it. The compiler generates code to power up the function unit, use it, and then power it down. To do this requires the algorithm, the compiler, and the microarchitecture all being aware of the functionality of the data path.

3.2. Compiler-Managed Caches

Cache misses, given the huge disparity between on-chip frequency and off-chip access times, is catastrophic with respect to performance. What if the compiler managed the cache hierarchy, and in particular used prefetch and poststore instructions to move instructions and data around in the on-chip cache hierarchy.

With the compiler in charge, prefetching should be far more effective, with the result that most of the time what is needed is in the L1 cache where its access is effectively immediate. To do this means breaking the barrier between the compiler and the microarchitecture. It also could mean making the cache implementation visible to the algorithm, which could then organize data and code in units more conducive to the compiler managed cache hierarchy.

3.3. Helper Threads (SSMT)

We introduced the notion of Subordinate Simultaneous Microthreading (SSMT) in 1999 [4] to take advantage of spare capacity in an SMT machine if the application admitted only one thread. Helper threads, as they are more commonly called, is a useful construct in its own right. Compiling has historically been associated with the period of time before the program starts execution. What if a helper thread could be used to recompile while the algorithm is running and thereby take advantage of run-time information not previously available to the compiler.

3.4. NiagaraX/PentiumY

Conventional wisdom suggests that the best use of the expected large increase in transistor count on a chip is multiple simple cores, rather than continuing to make more and more sophisticated cores. Certainly the former

approach saves power, and if the simpler cores are able to share the work to the point we can run these cores at lower frequency, they save even more power, given that power consumption is a function of frequency cubed.

The problem is that very few algorithms are parallelizable to the extent needed for this approach to win. On the contrary, every algorithm has its sequential part (Amdahl's bottleneck), and unless there is some core that can handle that part of the algorithm, the speedup is adversely affected, regardless how many cores are on the chip. If the compiler understands the organization of the heterogeneous cores, or better yet, if the algorithm can be written to take advantage of this organization, the Amdahl's bottleneck can be mollified.

I call this approach NiagaraX/PentiumY, a chip consisting of as many simple cores as will fit on the chip after allocating enough real estate to the heavy duty core (wide issue, serious branch prediction, out-of-order execution, trace cache, etc.) and an interconnection structure that allows the processor to move code and data among the cores as needed.

3.5. More Microcode

More cores mean more bandwidth required, exacerbating the already tenuous memory wall problem. One solution could be encoding instructions and data in as few bits as necessary for transfer on/off chip, but when on-chip, expanding them as needed. This would save off-chip bandwidth, increase cache performance, but at the expense of chip area, which does not seem to be at a premium.

3.6. Verification Hooks

The chip suggested by the elements described in this paper is by no means simpler than what is being produced today. And, verification of even today's chips is a very hard problem. However, if we break the barrier between microarchitecture and verification, perhaps we can promote verification to first class status and design into the microarchitecture verification hooks to improve our ability to verify that a chip does what it has been specified to do.

3.7. Fault-Tolerant Design

Frequencies are sufficiently high that we can no longer trust a bit to behave properly as it travels down a wire on the chip. An AND gate producing a 1 might find that value arriving at the input to the next gate as a 0. If we are to operate at our current frequencies and beyond, we must treat logic design as a fault-tolerant discipline.

4. CONCLUSIONS

What I have tried to do in this paper is point out some ways we can continue to ride the microprocessor performance improvement curves if we are willing to break the artificial barriers that have kept algorithms from providing hints to compilers, and both from knowing exactly what the implementation of the microarchitecture consisted of.

I am not saying that any of the above suggestions comes without cost, or that some of them might involve costs greater than their benefit. I am saying that process technology continues to give us more capability and it is clear that we can not handle that capability in the ways we have done in the past. I am suggesting that one way out of the fog could be to expose the various levels of the transformation hierarchy to each other and let them cooperate toward global solutions.

5. REFERENCES

- [1] Stamatis Vassiliadis, personal interaction, ISCA Santa Margharetta Ligure, June 1995.
- [2] Stamatis Vassiliadis, personal interaction, Delft, June 1998.
- [3] Y. N. Patt, "Requirements, bottlenecks, and good fortune: Agents for microprocessor evolution," *Proceedings of the IEEE*, vol. 89, pp. 1553–1559, November 2001.
- [4] R. S. Chappell, J. Stark, S. P. Kim, S. K. Reinhardt, and Y. N. Patt, "Simultaneous subordinate microthreading (SSMT)," in *ISCA '99: Proceedings of the 26th annual international symposium on Computer architecture*, Atlanta, Georgia, United States, 1999, pp. 186–195.