2. J. Hennessy et al., "MIPS: A Microprocessor Architecture," *Proc. 15th Ann. Microprogramming Workshop*, 1982, pp. 17–22.
3. C. Rowen et al., "MIPS: A High Performance 32-bit NMOS Microprocessor," *Proc. Int'l Solid-State Circuits Conf.*, 1984, pp. 180–181.
4. T.R. Gross et al., "Measurement and Evaluation of the MIPS Architecture and Processor," *ACM Trans. Computer Systems*, Aug. 1988, pp. 229–258.
5. S. Przybylski, "The Design Verification and Testing of MIPS," *Proc. Conf. Advanced Research in VLSI*, 1984, pp. 100–109.
6. N. Jouppi, "TV: An NMOS Timing Analyzer," *Proc. 3rd CalTech Conf. VLSI*, 1983, pp. 71–85.
7. S. Przybylski et al., "Organization and VLSI Implementation of MIPS," *J. VLSI and Computer Systems*, vol. 1, no. 2, 1984, pp. 170–208.
8. D.A. Patterson and C.H. Sequin, "RISC-I: A Reduced Instruction Set VLSI Computer," *Proc. 8th Ann. Symp. Computer Architecture*, 1981, pp. 443–457.
9. S. Krishnamurthi, "Artifact Evaluation for Software Conferences," *SIGPLAN Notices*, vol. 48, no. 4S, 2013, pp. 17–21.

Thomas R. Gross is a faculty member in the Computer Science Department at ETH Zurich. Contact him at thomas.gross@inf.ethz.ch.

Norman P. Jouppi is a distinguished hardware engineer at Google. Contact him at jouppi@acm.org.

John L. Hennessy is president emeritus of Stanford University. Contact him at hennessy@stanford.edu.

Steven Przybylski is the president and principal consultant of the Verdande Group. Contact him at sp@verdande.com.

Chris Rowen is the CTO of the IP Group at Cadence Design Systems. Contact him at rowenchris@gmail.com.

HPS Papers: A Retrospective

Yale N. Patt
University of Texas at Austin

Wen-mei W. Hwu
University of Illinois at Urbana–Champaign

Stephen W. Melvin

Michael C. Shebanow
Samsung

HPS happened at a time (1984) when the computer architecture community was being inundated with the promises of RISC technology. Dave Patterson's RISC at Berkeley and John Hennessy's MIPS at Stanford were visible university research projects. Hewlett Packard had abandoned its previous instruction set architecture (ISA) in favor of the HP Precision Architecture (HP/PA) and had attracted a number of people from IBM to join their workforce. Sun Microsystems had moved from the Motorola 68020 to the Sparc ISA. Motorola itself was focusing on their 78K, later renumbered 88K. Even Intel was hedging its bets, pursuing the development of the i860 along with continued activity on x86.

The RISC phenomenon rejected the VAX and x86 architectures as far too complex. Both architectures had variable-length instruction sets, often with multiple operations in each instruction. The VAX Index instruction, for example, required six operands to assist in computing the memory location of a desired element in a multidimensional subscripted array. If you wanted the location of A[I,J,K], you could obtain it with three instantiations of the Index instruction. The Index instruction took one of the subscripted variables—say I—and checked the upper and lower bounds; if they both passed, it used the size information of each element in the array and the array's dimensions to compute part of the location of A[I,J,K]. More than a half-dozen operations were performed in carrying out the work of this instruction. Intel's x86's variable-length instruction had prefixes to override an instruction's normal activity, and several bytes when necessary to locate the location of an operand.

RISC advocates argued that with simpler instructions requiring in general a single operation, wherein the signals needed to control the datapath were contained within the instruction itself, one would require no microcode and could make the hardware much simpler, resulting in a program executing in less time. Not everyone followed the mantra exactly; for example, Hennessy's MIPS used a pipeline reorganizer to package two operations in a single instruction. But for the most part, RISC meant single-cycle execution of individual instructions, statically scheduled and fetched one at a time. The instructions were essentially micro-ops.

We marched to a different drum. "We" were Yale Patt, a visiting professor in the fifth year of his nine-year visit at UC Berkeley, and Wen-mei Hwu, Michael Shebanow, and Steve Melvin, all PhD students at Berkeley who completed their PhDs there with Yale over the next several years. We dubbed our project the High Performance Substrate (HPS). We were all part of the Aquarius project, led by Al Despain. Al concentrated on designing a Prolog engine, and we focused on high-performance microarchitecture.

Berkeley was a special place to do research in the 1980s. Among the work going on, Velvel Kahan invented and refined his specification of IEEE floating-point arithmetic; Michael Stonebraker invented Ingres; Domenico Ferrari produced distributed UNIX; Manuel Blum, Dick Karp, and Mike Harrison pursued exciting theoretical issues; Lotfi Zadeh refined his fuzziness concepts; and Sue Graham did important work in compiler technology.

Unlike the RISC projects going on at the time, we felt it was unnecessary to meddle with the ISA; rather, we would require the microarchitecture to break instructions into micro-operations (AMD later called them Rops, for RISC ops) and allow the micro-ops to be scheduled to the function units dynamically (that is, at runtime). We insisted on operating below the ISA's level, allowing the ISA to remain inviolable and thereby retaining all the work of the previous 50 years. We were sensitive to not introduce features that might seem attractive at the moment but could turn out to be costly downstream.

Several pieces of previous work provided insights. Twenty years earlier, in the mid-1960s, the IBM 360/91 introduced the Tomasulo algorithm, a dynamic scheduling (aka out-of-order execution) mechanism, in their FPU.[1] Instructions were allowed to schedule themselves out of program order when all flow dependencies (RAW hazards) were satisfied. Unfortunately, instructions were allowed to retire as soon as they completed execution, breaking the ISA and preventing precise exceptions. In fact, the 360/91 design team had to get special permission from IBM to produce the 360/91 because it did not obey the requirements of the System 360 ISA.[2] IBM did not produce a follow-on product embracing the ideas of the 360/91 FPU.

Ten years earlier, in 1975, Arvind and Kim Gostelow[3] and Jack Dennis and David Misunas[4] introduced the important notion of representing programs as dataflow graphs, and Robert Keller[5] introduced the notion of a window of instructions—that is, instructions that had been fetched and decoded but not yet executed. When we studied these papers in 1984, several things became clear to us.

First, with respect to the Tomasulo algorithm, we would have to do something about the lack of precise exceptions. We correctly decided that this problem was the deal breaker that prevented IBM from producing follow-on designs. We introduced the Result Buffer, which stored the results of instructions executed out of order, allowing them to be retired in order. Although we did not know it at the time, Jim Smith and Andy Pleszkun were concurrently investigating in-order retirement of out-of-order execution machines, and came up with the name Reorder Buffer (ROB), which is a much better descriptor for our Result Buffer.[6]

Second, we recognized that Dennis and Arvind's dataflow graph concept could allow huge gains in performance by allowing many instructions to execute concurrently—and that although it would be near-impossible for a global scheduler to see the available parallelism in an irregularly parallel program, it did not matter. It only mattered that the operations themselves knew when they were ready to execute. By allowing the operations themselves to determine when they were ready to execute, we could exploit irregular parallelism. Allowing the nodes to determine when they can fire is the hallmark of dataflow.

Third, we recognized that, unlike Tomasulo, the mechanism need not be restricted to the FPU—that indeed other arithmetic logic unit operations, and more importantly, loads and stores, could also be allowed to execute in parallel. This let us build the dataflow graph from all instructions in the program, not just the floating-point instructions.

Fourth, we recognized that we did not have to change the ISA and in fact not doing so provided two benefits:

- Our HPS microarchitecture could be used with any ISA. That is, instructions in any ISA could be decoded (that is, converted) to a dataflow graph and merged into the dataflow graph of the previously decoded instructions. The dataflow graph's elements were single operations: micro-ops. Decoding could produce multiple operations each cycle, that is, wide issue.
- Importantly, we could maintain the ISA's inviolability.

Fifth, we recognized that to make this work, we needed to be able to continuously supply a lot of micro-operations into the execution core. We incorporated a post-decode window of instructions, but knew we could not, as Keller advocated, stop supplying micro-ops when we encountered an unresolved conditional branch. For this we would need an aggressive branch predictor, and beyond that, speculative execution. We did not have a suitable branch predictor at the time, but we knew one would be necessary for HPS to be viable. It would also require a fast mechanism for recovering the state of the machine if a branch prediction took us down the wrong path.

Finally, we recognized that unlike previously constructed dataflow machines, wherein the compiler constructed a generally unwieldly dataflow graph, our dataflow graph would be constructed dynamically. This allowed nodes of the dataflow graph to be created in the front end of the pipeline when the instructions were fetched, decoded, and renamed, and removed from the dataflow graph at the back end of the pipeline when the instructions were retired. The result: at any point in time, only a small subset of the instruction stream, the instructions in flight, are in the dataflow graph. We dubbed this concept "restricted dataflow." We saw that although pure dataflow was problematic for many reasons (such as saving state on an interrupt or exception, debugging programs, and verifying hardware), restricted dataflow was viable.

These revelations did not come all at once. We had the luxury at the time of meeting almost daily in Yale's office as we argued and struggled over the concepts. Sometimes one or more of us were ready to give up, but we didn't.

At the time, many of our colleagues were skeptical of our work and made it clear they thought we were barking up the wrong tree. We were advocating future chips that would use the HPS microarchitecture to fetch and decode four instructions each cycle. Our critics argued that there was not enough parallel work available to support the notion of four-wide issue, and besides, there were not enough transistors on the chip to handle the job. To the first criticism, we pointed out that although we could not see the parallelism, it did not matter. The parallelism was there, and the dataflow nodes would know when all dependencies had been satisfied and they were ready to fire. To the second argument, we begged for patience. Moore's law was alive and well, and although at the time there were fewer than 1 million transistors on each chip, we had faith that Moore's law would satisfy that one for us.

We started publishing the HPS microarchitecture with two papers at MICRO 18. The first paper provided an introduction to the HPS microarchitecture.[7] We described the rationale for a restricted dataflow machine that schedules operations out of order within an active window of the dynamic instruction stream. We showed the need for an aggressive branch predictor that allowed speculative execution, and we provided a mechanism for quickly repairing the state of the machine when a branch misprediction occurs. We introduced the notion of decoding complex instructions into micro-ops, and the use of the result buffer to retire instructions in program order. The second paper at MICRO 18, the "Critical Issues" paper, pointed out some of the problems that would have to be solved to make HPS viable.[8] We discussed various issues, including caching and control flow, and discussed what we called the "unknown address problem." We presented an algorithm for determining whether a memory read operation is ready to be executed in the presence of older memory writes, including those with unknown addresses.

We continued the work over the next several years, publishing results as we went along. In one of those subsequent papers, published at MICRO 21, we introduced the concept of a fill unit and the treatment of large units of work as atomic units.[9] In all, we published more than a dozen papers extending the ideas first put forward in 1985. More importantly, starting with Intel's Pentium Pro, announced in 1995, the microprocessor industry embraced the fundamental concepts of HPS: aggressive branch prediction leading to speculative execution; wide-issue, dynamic scheduling of micro-ops via a restricted dataflow graph; and in-order retirement enabling precise exceptions.

Yale Patt continued as a visiting professor at Berkeley for four more years, then moved on to the University of Michigan and eventually to the University of Texas at Austin. Wen-mei Hwu finished his PhD in 1987 and took a faculty position at the University of Illinois at Urbana–Champaign. Steve Melvin completed his PhD in 1990 and has been a successful independent consultant in the computer architecture industry for more than 25 years. Michael Shebanow completed his PhD and went on to become a lead architect on many microprocessors over the years, including the M88120 at Motorola, the R1 and Denali at HAL, the M3 at Cyrix, and the Fermi GPU shader core complex at Nvidia. He is now a vice president at Samsung, leading a team focused on advanced graphics IP.

References

1. R.M. Tomasulo, "An Efficient Algorithm for Exploiting Multiple Arithmetic Units," *IBM J. Research and Development*, vol. 11, no. 1, 1967, pp. 25–33.
2. R.M. Tomasulo, lecture at University of Michigan, Ann Arbor, Jan. 30, 2008; www.youtube.com/watch?v=S6weTM1tNzQ.
3. Arvind and K.P. Gostelow, *A New Interpreter for Dataflow and Its Implications for Computer Architecture*, tech. report 72, Dept. of Information and Computer Science, Univ. of California, Irvine, 1975.
4. J.B. Dennis and D.P. Misunas, "A Preliminary Architecture for a Basic Data Flow Processor," *Proc. 2nd Int'l Symp. Computer Architecture*, 1975, pp. 126–132.
5. R.M. Keller, "Look Ahead Processors," *ACM Computing Surveys*, vol. 7, no. 4, 1975, pp. 177–195.
6. J.E. Smith and A.R. Pleszkun, "Implementation of Precise Interrupts in Pipelined Processors," *Proc. 12th Ann. IEEE/ACM Int'l Symp. Computer Architecture*, 1985, pp. 36–44.
7. Y.N. Patt, W.M. Hwu, and M. Shebanow, "HPS, A New Microarchitecture: Rationale and Introduction," *Proc. 18th Ann. Workshop Microprogramming*, 1985, pp. 103–108.
8. Y.N. Patt et al., "Critical Issues Regarding HPS, A High Performance

Microarchitecture," *Proc. 18th Ann. Workshop Microprogramming*, 1985, pp. 109–116.

9. S.W. Melvin, M.C. Shebanow, and Y.N. Patt, "Hardware Support for Large Atomic Units in Dynamically Scheduled Machines," *Proc. 21st Ann. Workshop Microprogramming and Microarchitecture*, 1988, pp. 60–63.

Yale N. Patt is the Ernest Cockrell, Jr. Centennial Chair in Engineering at the University of Texas at Austin. Contact him at pattyn@austin.utexas.edu.

Wen-mei W. Hwu is the AMD Jerry Sanders Endowed Chair in Electrical Engineering at the University of Illinois at Urbana–Champaign. Contact him at w-hwu@uiuc.edu.

Stephen W. Melvin is an independent consultant in the computer architecture industry. Contact him at melvin@zytek.com.

Michael C. Shebanow is a vice president at Samsung. Contact him at shebanow@gmail.com.

Retrospective on "Two-Level Adaptive Training Branch Prediction"

Tse-Yu Yeh
Apple

Yale N. Patt
University of Texas at Austin

Looking back at the world of computer architecture in 1991, when we wrote our MICRO paper,[1] we believe our motivation for addressing branch prediction is best illustrated by our earlier 1991 ISCA paper,[2] which showed that instructions per cycle can be greater than two. Conventional wisdom at the time had argued otherwise. We demonstrated in that paper that there was enough instruction-level parallelism in nonscientific workloads to support our contention. We further showed that the HPS microarchitecture (a superscalar out-of-order execution engine) could improve the execution rate greatly by exploiting instruction-level parallelism, provided the penalty caused by incorrect branch predictions was minimized.

In fact, the lack of an aggressive, highly accurate branch predictor was a major stumbling block in enabling our high-performance microarchitecture to achieve its potential. The HPS microarchitecture required the execution core to be kept fully supplied with micro-ops, organized as a dataflow graph that allowed lots of concurrent execution. This meant a powerful and effective branch predictor that could fetch, decode, and execute instructions speculatively, and only very infrequently have to throw away the speculative work because of a branch misprediction. The computer architecture community had pretty well given up on branch prediction providing any major benefits. Most accepted as fact that Jim Smith's 2-bit saturating counter was as good as it was going to get.[3] His branch predictor was published in ISCA in 1981 and was still the best-performing branch predictor 10 years later. It was the branch predictor in Pentium, Intel's "brainiac" microprocessor, the top of the x86 line at the time, released in 1991.

Tse-Yu spent the summer of 1990 at Motorola, working for Michael Shebanow, one of the original inventors of the HPS microarchitecture. Michael was interested in branch prediction, and in fact almost did his PhD dissertation at Berkeley on branch prediction. Tse-Yu and Michael had many conversations about microarchitecture tradeoffs that summer, and often the discussions would gravitate to branch prediction. When Tse-Yu returned to Ann Arbor for the fall semester, he was excited about the possibility of a branch predictor that could support the HPS microarchitecture.

After many after-midnight meetings over several months in Yale's office, combined with a whole lot of simulation, the two-level adaptive branch predictor was born. We knew the branch predictor would have to be dynamic; that was a no-brainer. The branch predictor would have to use historical information based on the input data that the program was processing, and it would have to accommodate phase changes. But that was not enough. What historical information would yield good branch prediction? Somehow we stumbled on the answer: it was not the direct record of a branch's behavior that was needed, but rather whether the branch was taken at previous instances of time having the same history. That is, if the