How VLIW Almost Disappeared—and Then Proliferated

Richard A. Lethin

An account of the heady early days of VLIW, its apparent demise, and its victorious resurgence.

Very long instruction word (VLIW) refers to a computer architecture and algorithms that take advantage of large amounts of instruction level parallelism (ILP) [1], [2]. Joseph A. (Josh) Fisher, a former Yale professor and a Hewlett-Packard Senior Fellow, introduced VLIW architecture in the early 1980s.

The insights underlying Fisher's invention of VLIW came to him when he was a graduate student at New York University's Courant Institute in the late 1970s. He was microcoding a clone of the Control Data Corporation (CDC) 6600 computer. To maximize the performance of the clone, called PUMA, he used a standard trick of writing microcode with many concurrent operations. In doing so, he realized he could get even more concurrency and performance by moving operations speculatively above branches. His study of this motion, and how it violated the laws of computer architecture of the day, led to his invention of the trace scheduling compiler algorithm. He wrote his Ph.D. thesis about trace scheduling. He later became a professor at Yale University, where he started the Extremely Long Instruction (ELI) project, which developed the architecture for the trace scheduling compiler.

PUMA used large-scale integration (LSI) to achieve a much smaller footprint than the 1960s-era CDC 6600, which used a much lower level of integration. PUMA needed an area of only 12 square feet versus

Digital Object Identifier 10.1109/MSSC.2009.933431

200 square feet for the 6600. But a smaller footprint was only one of the goals; Fisher's adviser, Prof. Ralph Grishman, really wanted to push architecture research at Courant by providing a means to emulate new architectures. Ironically, they didn't expect that there would be any architecture innovations from PUMA itself.

The Birth of Multiflow

In 1984, Fisher left Yale to form start-up Multiflow Computer to build VLIWs using LSI chips, and this work has had continuing influence on the field of computer architecture, though it did not produce commercial success at Multiflow. Now, three decades after its inception, the current ubiquity of VLIW demonstrates its importance. Chances are that there are VLIWs within your arm's reach right now. They are embedded in TV set-top boxes, cameras, printers, and soon, smart phones (Figure 1). VLIWs enable these devices to push the limits of the programmable computational performance, per watt and per dollar. In addition to embedded devices, VLIW technology is in places you might not expect to find it, such as the world's largest supercomputers, advanced multicore architectures, and even computers that don't employ VLIW architecture. This is a trend you should expect to see more of, because VLIW becomes even more relevant with current VLSI physical tradeoffs.

It's true that architectures that could issue multiple instructions with a software schedule preceded Fisher's architecture; wide horizontal microcode was a mainstream technique for controlling computational units. However, before Fisher's VLIW, getting performance from such hardware required hand coding to achieve the levels of concurrency that could keep the machine busy [3]. The requirement for hand coding imposed a practical limit on how wide microcode could be. Hardware-based schedulers existed, but those

FIGURE 1: The new Palm Pre phone is based on the Texas Instruments' OMAP 3 processor, which includes a VLIW-based multimedia accelerator.

had practical limits imposed by hardware costs and wiring constraints. Thus, prior to Fisher, people strongly believed that there was an upper bound on the amount of ILP arising from these practical constraints and that the upper bound was small.

Fisher's breakthrough boils down to two elements:

1) The trace scheduling compiler algorithm that looks beyond the joins and branches that delimit basic blocks to find parallelism in larger groups, or traces, of instructions. This allowed the compiler to support the "very" in VLIW.
2) Ways to improve horizontal microcode machines so that compilers could practically schedule the operations in instructions. He added features to the architecture that made the problem of software-based scheduling for very wide microcode tractable—features like wide register files, self-draining pipelines, and "dangerous load" memory operations. Dangerous loads, as we originally called them, are those that can be moved above branches without causing exceptions. We had to change the term to the more palatable "dismissible loads" to satisfy the marketing folks. Would you buy a machine with dangerous loads?

(Note that VLIW experts distinguish an *instruction*, which is a long set of bits sent to the VLIW, from an *operation*, which is an action that corresponds to an individual arithmetic logic function—an add, branch, load, or store, for example.)

The result of these elements is that the computer could find large amounts of ILP from programs expressed in ordinary high-level languages like C and FORTRAN. Software-based scheduling eliminated the gates for hardware scheduling, a big advantage in terms of cost and power savings, advantage, allowing the hardware to exploit the parallelism by scaling to wide instructions.

In 1984, I was an undergraduate at Yale College and took the computer architecture course taught there by Fisher and his VLIW co-researcher and Yale Research Scientist John O'Donnell, who was working with him on the ELI project. Many of their lectures were framed in terms of how architectures could be better implemented as VLIW. For example, when we reviewed the Xerox Dorado workstation, which allowed the user to customize its microcode for different workloads, Fisher and O'Donnell pressed us to explain why a compiler instead couldn't just directly target its microcode from the application source.

Long Days, Late Nights in the Lab

That summer, Fisher, O'Donnell, and Yale graduate student John Ruttenberg founded Multiflow to build VLIWs. They offered me a job when I graduated in 1985, and on joining the company I proceeded to design the floating-point board for their VLIW. It was truly a "Soul of a New Machine" experience, as Tracy Kidder describes in his true-story book about a computer design team racing to complete a next generation computer design under a tight schedule and great pressure [4]. We spent *continual* very late nights, and usually seven days a week, in our lab in Connecticut (Figure 2).

I built the floating-point board (Figure 3) around Weitek floating-point chips. After two years of very hard work, the resulting product hit the markets as the Multiflow Trace 200 minisupercomputer VLIW, with a trace-scheduling C and FORTRAN compiler. The machine was available in 7-, 14-, and 28-b-wide issue. The largest machine (the 28-wide version) had a 1024-b instruction word. The first machine was shipped to the Supercomputing Research Center in Lanham, Maryland.

A year later, I redesigned the floating-point board to incorporate the B21×0 floating-point chips from Bipolar Integrated Technology. This cut the latency of floating-point operations in half. Then, the summer before I left Multiflow for graduate school at MIT in 1988, I designed a modification for the board that provided, with word-level single-instruction, multiple-data (SIMD), two 32-b-wide floating point adds, multiplies, or multiply-accumulates in a single operation. This "pair mode" boosted the arithmetic performance of the Trace.

With four of these upgraded boards, the 28-wide Trace could use pair mode to execute, for each VLIW instruction, 32 floating-point arithmetic operations (as pair multiply-accumulates on the eight floating-point operations), along with 16 integer operations and four branches. This would be sustained performance, driving the Trace's then massive bandwidth from the interleaved memory system. My guess is that approximately 100 of these minisupercomputer systems were shipped to customers such as Proctor and Gamble, Grumman, China Lake Naval Weapons Center, Hewlett-Packard, Carnegie Mellon University, Sikorsky, Yale, and the University of Catania. Two papers describe the Trace architecture and its compiler [5], [6].

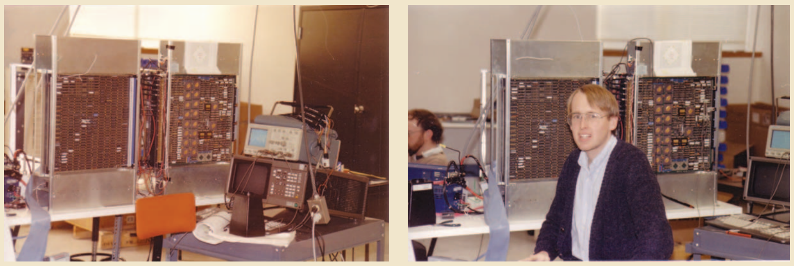
(a) (b)

FIGURE 2: (a) The Multiflow VLIW prototype in development. We did not have simulation capability for a whole board, let alone a whole system, so the boards were fabricated to a preliminary design and then made to function properly by reprogramming programmable array logic (PAL) chips and assembling with wire and solder. Extender cages on the back side of the backplane allowed some boards to be accessed by oscilloscope and logic probes. The boards measured 17 by 17 inches, close to the largest that could be fabricated at the time. (b) The author and, in the back, Shannon Hill, now of ChipWrights. (Photo by John O'Donnell.)

Spreading the VLIW Religion

In 1988, Multiflow attracted the attention of Digital Equipment Corporation (DEC) as a possible acquisition or technology supplier. Discussions between the companies began. DEC Senior Corporate Consulting Engineer Tryggve Fossum led a team in Marlboro, Massachusetts, that was finishing the VAX 9000, a mainframe-class computer implemented in emitter-coupled logic (ECL) with a high clock rate and a hardware-based instruction scheduler. The DEC team was looking for its next project, so Fossum took charge of the technical evaluation of Multiflow. Since I was by that time up in Cambridge, Massachusetts, at MIT, I traveled out to Marlboro a few days a week to assume a role as an embedded Multiflow engineer at DEC, helping to explain our VLIW technology. Another DEC corporate consulting engineer, Joel Emer, joined our discussions.

We quickly established that it would be easy to build a VLIW for a RISC instruction set architecture such as MIPS (which originally stood for microprocessor without interlocked pipeline stages). We sketched such two-wide and four-wide designs on Fossum's white boards. (Of course, this was when DEC was developing Alpha, but they couldn't disclose that to me. So our discussions were in terms of MIPS.) With Moore's law allowing more and more transistors per chip each year, the development of such machines seemed inevitable.

We created experimental models in the Multiflow compiler for the architectures and developed a means to emulate the designs and estimate their performance. We demonstrated that the Multiflow compiler could support such a

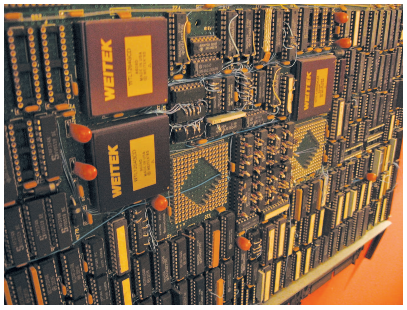

FIGURE 3: The first VLIW floating-point board, prototype for the 200 series. The bit-sliced design of the register file (large chips removed across the top) provides pins for the many ports that VLIW requires. These ports feed the Weitek floating-point arithmetic logic units (ALUs). The blue wires, bent-up legs, and other strange items on the board are bug fixes and sometimes capitulation to mysticism to fix logical and electrical problems. Of course, all of these wires and magic were integrated into the respins of the boards for production. The clock rate of this board is 16 MHz, and it could issue VLIW instructions (at least the floating-point ones) at half this rate, 8 MHz. The full 28-wide Trace had four of these boards (along with 13 other boards for integers, control, and memory).

machine and thus built the case for why the Multiflow technology would be good for DEC.

VLIW Beliefs Are Tested

The hardware-based instruction scheduler in DEC's VAX 9000, a masterful feat of engineering, could find and issue a small integer number of concurrent operations per cycle. But the large Trace could issue 28 operations per cycle, without such hardware. The Multiflow compiler was succeeding in finding and scheduling parallelism to feed the Trace. What was the trick? The trick was that scaling to a 28-wide machine really worked only for applications with lots of data parallelism, like scientific applications running on supercomputers. The Multiflow compiler exposed the data parallelism as ILP through very heavy loop unrolling. For applications that were highly dynamic, with unpredictable branches, such as system codes, the compiler couldn't statically predict control flow accurately enough to support 28-wide issue.

So this led to doubts. Fossum and Emer suggested that perhaps instruction scheduling in hardware was better. Hardware would, after all, have the benefit of runtime information. My job, of course, was to sell VLIW, and I believed in it, but with any attempt I made to assert that hardware instruction scheduler couldn't do something that the Multiflow compiler could; someone on Fossum's team would show that I was wrong. There were hardware analogs to all of the VLIW concepts. Moving instructions above branches, a neat VLIW trick, was instruction speculation. Using branch profile information to choose traces, the heart of trace scheduling, could be done with branch prediction units. [Later, when I learned about the Futamura projection, it seemed to capture this relationship between hardware (as an interpreter), and compilers, via the mechanism of partial evaluation, in a profound way.]

Even so, that hardware-based instruction scheduling seemed like a waste to me. I had a pretty vivid sense of the personal cost of hardware; those 100 or so wires in Figure 3 were all hand-soldered, and so were subsequent prototype boards. Why would anyone want to design hardware doing some function over and over again that might be done just once by a compiler?

We created experimental models in the Multiflow compiler for the architectures and developed a means to emulate the designs and estimate their performance.

The DEC guys also preferred to not have extra hardware, but they also were not so constrained by architecture religion. Emer, in fact, was already famous at that point for debunking various presumptions and marketing hype about what MIPS was and thus establishing the quantitative school of computer architecture.

Of course, whether hardware-based scheduling makes sense depends on the scale, engineering, and cost constraints of the day. The trade-offs have evolved since those discussions in the early 1990s.

Attack of the Killer Micros

In 1989, Multiflow started encountering severe competition from single-chip CPUs. Intel Corporation announced the i860, a RISC architecture that integrated floating-point unit and cache on a single chip. It could issue multiple instructions per cycle. It was a two-wide chip. Whatever the memory bandwidth advantage and multiple issue advantage of the multiboard 28-wide Trace versus the fully integrated i860, the cost disadvantage of a nonintegrated CPU like Trace was fatal.

Putting everything close together on a single chip also gave the i860 enormous speed and power advantages. The i860 could run at 25 MHz, versus 8 MHz for the Trace. The i860 used a few watts; the Trace had multiple-kilowatt power supplies, solid copper bus bars on the backplane to distribute current, and similar high-power provisions.

Multiflow had been working on a high-clock-rate ECL successor product, but the big-box approach was fundamentally based on a much higher cost technology and on a much slower performance improvement curve than the microprocessors. Multiflow and the rest of the minicomputer, minisupercomputer, and supercomputer industry were "toast."

No More Funding

In 1990, the investors in Multiflow balked on a new round of funding. While a deal to license the compiler to Intel for the i860 had gone through, the critical acquisition/license deal from DEC failed to materialize, due (I think) to DEC's own very serious business challenges from the "Killer Micros"—the highly integrated microchips like the i860. To the dismay of Fossum, Emer, and the other technical evaluation people at DEC who wanted to work with us, Multiflow went out of business. DEC immediately licensed the Multiflow hardware and compiler (on the cheap!) and DEC, Intel, and other computer companies quickly hired the Multiflow engineers and compiler developers.

I was at Intel Corporation in Santa Clara, California, that spring, working on the physical implementation of the message-driven processor for the J-Machine [7] when friends called to let me know that Multiflow had failed. I mentioned this unhappy fact to my manager, and soon Intel manager Pat Gelsinger and other Intel staff arrived at my cubicle. I wrote a list of all the Multiflow staff for their recruiting purposes.

By 1992, DEC was back on its feet and announced the EV4 Alpha 21064 microprocessor [8], which—through custom VLSI engineering methods—clocked at an amazing 200 MHz. The Killer Micros had won, as Eugene Brooks of Lawrence Livermore National Labs had predicted in the paper that coined the phrase [9].

Apostles Spread the Word

Ironically, it was by Multiflow's going out of business that Multiflow's technology lived on and permeated the industry. The Multiflow team had been pioneers in ILP, and their ideas for speculative execution, predication, and memory disambiguation soon spread.

The principles of VLIW and trace scheduling ended up influencing DEC processor designs, but not in their original form. Compiler scheduling was very important for many multiboard designs like the Trace. However, Tryggve Fossum, Joel Emer, and former Multiflow compiler engineers Bob Nix and Geoff Lowney found, with careful research, that a moderate level of superscalar issue control was indeed practical in hardware when the processor was integrated into a single chip, as they had suspected. The EV5 Alpha 21164 processor, released in 1995 [10] was the microprocessor most heavily influenced by the Multiflow collaboration. The EV5 had four-way superscalar design and in-order instruction execution. The dynamic effects of caches and branch predictors were best handled in hardware. Yet it was still important for the compiler to organize the program so that the ILP was apparent. This was done by one of DEC's optimizing compilers extended along the lines of the Multiflow compiler.

SGI (which had been acquired by MIPS Technologies) did not license the Multiflow compiler technology, but ex-Multiflow CPU designer Paul Rodman contributed to the design of the MIPS R8000, a statically scheduled multiple-issue MIPS implementation [11]. Multiflow cofounder John Ruttenberg and Multiflow engineers Woody Lichtenstein and Doug Gilmore had also gone to SGI and designed a new modulo-instruction scheduler for the R8000 in the SGI Mongoose compiler [12].

Hewlett-Packard hired Josh Fisher. HP licensed the Multiflow technology, and in 1994 Fisher established a branch of HP Labs in Cambridge, Massachusetts, to develop what became the Lx VLIW chips, in collaboration with ST Microelectronics, which focused on the embedded market [13]. Multiflow compiler engineer Stefan Freudenberger also joined the lab, and contributed to the design of the compiler for Lx.

By the mid-1990s, Equator Technologies realized that VLIW could revolutionize the digital signal processing world.

Intellectual Property

The ownership of Multiflow's intellectual property passed to a small company, Technology Licensing Inc. (TLI), created by the Multiflow's chief financial officer, but the company didn't have any practical understanding of the technology. So Multiflow cofounder John O'Donnell formed Equator Technologies to further sell (on behalf of TLI), support, and enhance the Multiflow compiler around the globe for companies such as Fujitsu and NEC. O'Donnell was joined at Equator by Multiflow software and compiler engineers Ben Cutler and Cindy Collins.

Interestingly, many of these early licensees were interested in the Multiflow compiler for non-VLIW purposes. The Multiflow compiler was fanatically engineered to use clean abstractions and intermediate representation, and it had (for the time) state-of-the-art scalar optimizations and the ability to expose lots of ILP. It was just a darned good compiler. So NEC's compiler team adopted it for compiling to its MIPS-architecture workstation line and got much better performance than MIPS' own compiler. Fujitsu licensed the compiler to use on its VPP parallel supercomputers [14]: Fujitsu bolted the back end of the Multiflow compiler onto its vectorizing compiler to trace-schedule a mix of scalar and vector operations. The scalar processor itself used VLIW techniques coming from compiler efforts to support the early QA-2 wide-word machine [15], [16]. NEC eventually dropped its MIPS product line, but Multiflow technology lives on at NEC and is the basis of the compiler for the Japanese Earth Simulator supercomputer!

VLIW for Multimedia and DSP

By the mid-1990s, Equator Technologies realized that VLIW could revolutionize the digital signal processing (DSP) world. Looking back, Ben Cutler, then Equator's vice president of corporate planning and business development, said, "At the time, DSPs were a mix of architectural features that were toxic to compilers"—a world where "Serious bodies of code existed only in object code (not even assembly!)." Equator recognized the exploding market for video processing, which uses algorithms (for example, MPEG-2 encoding) that are mainly dense linear algebra that is good for VLIW. With venture capital funding, Equator produced a multimedia VLIW DSP that could be programmed in C [17]. (The small service business of Equator separated and became the start of my company, Reservoir Labs.) Equator's advantage was having a license to the Multiflow compiler, so it was immediately available to program the chip using C when the hardware was released.

Texas Instruments embarked on a similar project, culminating in the c6x VLIW for DSP and media processing [18]. Lacking the Multiflow compiler and Multiflow people, TI took longer to get to market with a good compiler. Fujitsu also produced a multimedia DSP VLIW, the FR-V, the most recent version being a multicore of eight-way VLIWs [19]. All these multimedia VLIW chips follow the general form of executing from two to eight instructions per cycle, with one of the functional units implementing a multimedia instruction set of word-level SIMD operations. Among these VLIWs, the TI c6x (part of OMAP 3 and 4) and Fisher's Lx architecture [now embedded in systems on chips (SOCs) sold by ST Microelectronics] are the children that have prevailed and that permeate the DSP and multimedia embedded product space.

We VLIW fans comforted ourselves with the knowledge that embedded chips outsell the number of processors to the desktop/workstation/server markets many times over.

Static Compiler Versus Dynamic Hardware

Intel, a CISC (complex instruction set computer) house because of its wild success with the x86, was itself hedging on the RISC versus. CISC debate by also selling the i860 and i960, which are fine RISC (reduced instruction set) architectures. Intel's release of the Pentium Pro x86 (PPro) in 1995 settled (though perhaps did not end) the debate. PPro demonstrated that it was possible even for CISC architectures to play all of the tricks of dynamic out-of-order, speculative execution that previously were thought practical only for RISC architecture. RISC has no significant advantage, relative to code compatibility with x86, which trumps everything. Intel's i860 and i960, both good architectures, fizzled away. Eventually, the fine DEC Alpha architecture succumbed.

The Ultimate Non-VLIW

PPro is the ultimate non-VLIW; it is CISC and has hardware-based scheduling. It may seem ironic then that two of the three principal engineers of the Pentium Pro, Dave Papworth and Bob Colwell, were ex-Multiflow designers. It makes sense, though, since this meant that they were very familiar with where parallelism is in serial code. However, because absolute object code compatibility with x86 was required, they rejected VLIW for PPro.

As PPro conquered desktop, workstation, and server markets in succession, it seemed as though VLIW, for these markets, was a passing fad, like RISC.

Interestingly, the Multiflow technology did turn up to play a direct role in PPro. If you look closely, you can see settings of Multiflow compiler flags in the fine print of the first PPro SPEC benchmark results in 1995. The Multiflow compiler was a part of the Intel Fortran and C compilers! Pentium Pro (and ex-Multiflow) designer Bob Colwell reports that innovations from the Intel compiler group for PPro contributed 20–30% to performance results. The lesson is that even the most complex hardware-based instruction scheduler loves it when the compiler finds more parallelism for it. Alas, the Multiflow compiler lives on in the current Intel compilers mostly in sprit. In the 19 years since Intel acquired the Multiflow technology, the company has improved and rewritten most of the compiler several times over. The number of original lines of Multiflow compiler in the Intel compiler has diminished to just a few.

Hardware-Based ILP Encounters Wire Delay

We VLIW fans comforted ourselves, in that dark age when VLIW was in eclipse, with the knowledge that embedded chips outsell the number of processors to the desktop/workstation/server markets many times over. VLIWs were getting traction in this embedded market with, for example, the Lx and c6x architecture. We tried not to pay attention to the success of the hardware-based scheduler chips like PPro, working in environments where power (or architectural aesthetic) mattered little. Embedded chips would take over the world and they would be VLIW.

Change in Tradeoff

The recent inflection, though, at the 65-nm VLSI process node, has changed the trade-off. Chip wire delays have blocked scaling the control for hardware-scheduled pipelines that allowed for a succession of larger processors. Increasing clock frequency for increased throughput no longer pays off. Static leakage has made power the dominant consideration for desktop and mainframe processing. Complexity is no longer free. A consequence of this inflection has been the imperative to go multiprocessor or "multicore" to get performance, rather than to build larger single processors. Multicore threatens to scuttle the smooth scaling of performance on the existing software base, a foundation of the productivity improvements from Moore's law over the past decades.

Overshadowed perhaps by the multicore panic is the fact that the cores themselves have become simpler, with shorter and simpler pipelines. Doing instruction scheduling in the compiler looks increasingly attractive to allow shedding the overhead of hardware-based schedulers. For example, SiCortex achieves ultralow power in its six-core supercomputer nodes through a short, statically scheduled, multiple-issue instruction pipeline.

Tilera's 64-core chips are composed of short-pipeline, statically scheduled VLIWs. Intel had to retrench from the deep pipeline Pentium 4 to the shorter Pentium M (derived from PPro) when it hit a power wall, and possibly Intel is not done. The eagerly anticipated Intel Larrabee chip (number of cores unknown, rumored to be 24) exhibits simple, statically scheduled pipelines. VLIW is back! Compiler frameworks developed in the early 1990s but neglected until recently, such John Ruttenberg's modulo scheduler in Mongoose, are being rediscovered.

To paraphrase the Micro 2005 keynote address [20] by HP Labs researcher Norm Jouppi, known for computer architecture innovations starting with the original Stanford MIPS in the early 1980s: There are no new ideas in computer architecture, just old ideas to be rediscovered again!

The Future of VLIW Architecture

Will we be seeing pure 28-wide machines soon? I doubt it. Instead, VLIW is one axis in architectural design space. New architectures blend VLIW with the other architecture axes such as multicore, streaming, SIMD, vector, and special functional units. The ILP used to support the 28-wide Multiflow Trace was data parallelism. Such parallelism can be better exploited in combination with the other architectural techniques.

For example, see Figure 4, the organization of Streaming Processors' Storm-1 DSP. It provides stream parallelism across multiple 12-wide VLIWs, and within each VLIW there is also word-level SIMD. The VLIW contributes the ability to concurrently and efficiently utilize the different functional unit types. This could be used for data parallelism—but the majority of the data parallelism is exploited through the streaming and word-level SIMD.

Another example of VLIW blended with other architectural axes is the Tilera TILE64, shown in Figure 5. In this multicore architecture, the cores are three-way VLIWs. However, the interconnection network provides register-to-register communication with very low latency. This can be exploited to "dock" subsets of tiles. Although docked tiles have their own program counters, software can make these docked tiles function as larger combined VLIW.

For perspective, Storm-1 and TILE64 pack across their lanes or cores processors with about six times the aggregate "issue width" of the original Multiflow 28-wide Trace, running at clock rates 100 times that of the Trace. The Trace CPU required nine enormous PC boards (plus eight enormous memory boards, but the memory is off-chip in Storm-1 and TILE64). So the thumbnail-size Storm-1 and TILE64 chips are 600 times faster than the cabinet-size Trace. Hardware has come a long way.

A Slight Concession

I will concede that there occasionally might be a reason not to use any VLIW in an architecture. John Shalf and others at Lawrence Berkeley Labs are developing a massively parallel computer in which power considerations lead to the conclusion that one should not use a VLIW. The climate and weather modeling algorithms for which Shalf's design is targeted have lots of data parallelism, but not enough ILP to justify any extra circuitry that would impose area/leakage overhead.

The Future of Compilers

Have compilers come as far as hardware? Current VLIW compilers are not much better than they were back in the 1990s. That's OK, though, because VLIWs have not scaled that

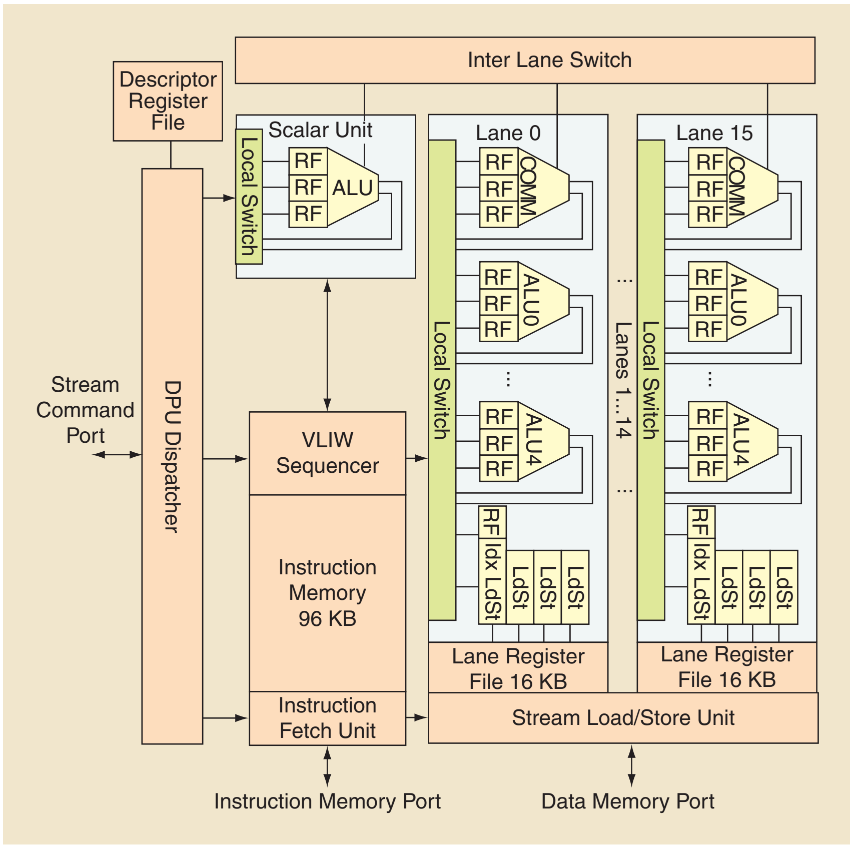

FIGURE 4: The Storm-1 architecture from Streaming Processors Inc. (SPI). The 16 lanes operate in a streaming fashion; each lane is a 12-way VLIW, and the individual operations in the lane further provide word-level SIMD. (From [21].)

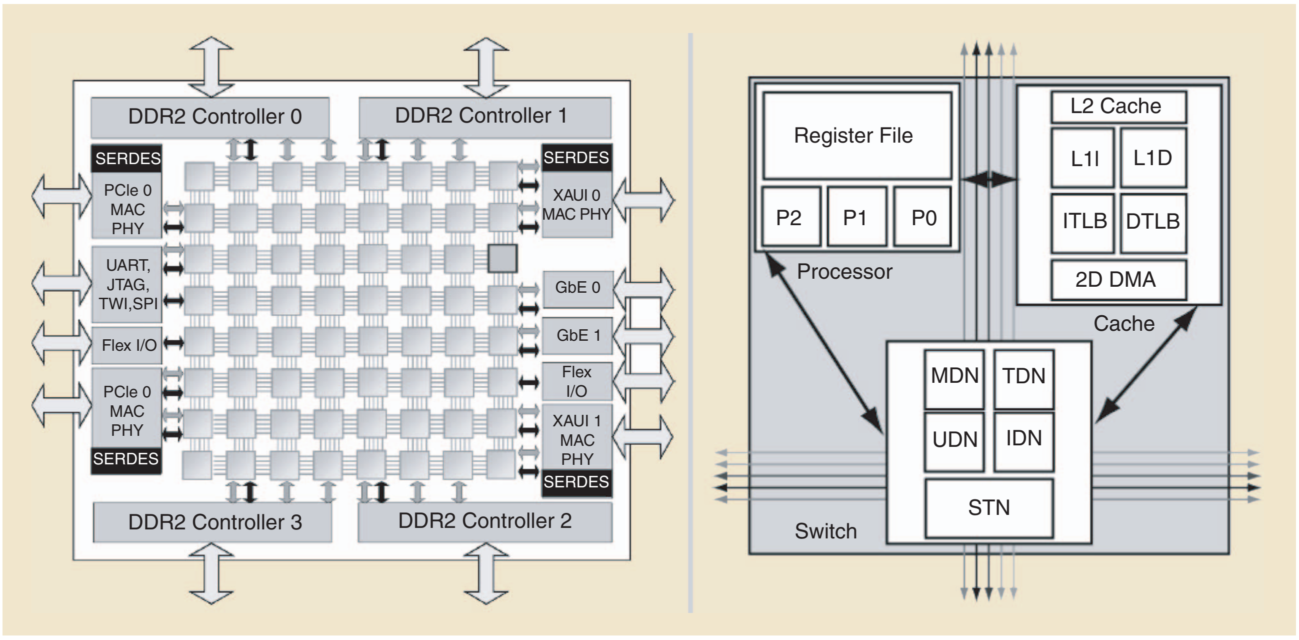

FIGURE 5: The Tilera TILE64 chip. Each processor engine is a three-way VLIW. The tiles can be docked through the communication network and the software arranged to construct a range of VLIW widths. (From [22].)

much. The problem seems to be that compilers are very weak at dealing with other non-ILP forms of parallelism. For example, to achieve the performance potential of VLIW media chips, the programmer must present "friendly" C [23], [24]. This means that the programmer must restructure code to present parallelism implicitly or explicitly in the C codes. Media codes must bring SIMD parallelism as C intrinsics that correspond directly to the word-level SIMD operations of the architecture, and they must have rearranged data layouts to align to the operations to the word-level SIMD. They must be pre-unrolled just the right amount to balance register pressure, and they need many other tweaks. This kind of media code is not assembly language, but it isn't much above it.

Incidentally, Multiflow compiler engineer Cindy Collins had just finished a module to form pairs of operations in trace formation and instruction scheduling to support word-level SIMD when Multiflow failed. This capability, available in 1990, has still not made it into most contemporary compilers.

This recalls the fresh-look approach that Josh Fisher took when he invented VLIW with his advanced compiler approach.

"Friendly" C at a higher level is needed for the "blend" architectures. For the Storm-1 streaming architecture, the programmer must write in "streaming" C dialects, choosing specific single dimensions of data-parallel algorithms to correspond to streams, indicating explicitly whether the rest of the data parallelism is inside streams or is in stripes along higher levels. For the TILE64, the programmer must manually partition the computation across the tiles for maximum performance, writing to parallel APIs. With the accelerating innovation and scale of such new architectures, the problems of programming seem to be multiplying.

A Note of Hope

But let me offer some hope. There is an emerging view in some computer architecture circles that the best efficiency for next-generation multicore chips comes with *explicit management* of the memory operations, communications operations, and synchronization across cores. For example, in the Storm-1, the stream "register files" are explicitly managed with direct memory access operations. Evidence, such as the recently published report by researchers at the Berkeley Parallel Computing Laboratory (Par Lab), shows the benefit of this approach over hardware-managed communication in conventional symmetric multiprocessing (SMP) multicore processors [25]. In an analogy to VLIW, Par Lab shows that *explicit management* of these mechanisms avoids the need for scheduling hardware and can do a better job. For scalable parallel processing (for example, aggregations of chips for terascale or exascale computing), it is implausible that hardware scheduling could practically manage these communications/memory/synchronization mechanisms at the levels of anticipated concurrency.

With this in mind, we at Reservoir Labs have been developing a "polyhedral" compiler that can generate such high-level explicit schedules automatically [26]. The various automatic parallelization, communication generation, synchronization generation, and placement compiler phases for the explicit management of the machine result in very complex problem instances that test the limit of mathematical optimization libraries. However, through new—some would say fanatical—compiler engineering, it is becoming conceivable to solve these mapping problems statically. I believe that our field is at the cusp of a new generation of processor architectures that are shaped directly by the capabilities of such new high-level compiler technology. This recalls the fresh-look approach that Josh Fisher took when he invented VLIW with his advanced compiler approach.

Acknowledgments

Thanks go to Josh Fisher, Tryggve Fossum, Ben Cutler, Bob Colwell, Norman Basch, and John Shalf for comments and suggestions for this article. I am grateful for opportunities to study and advance computer architecture in collaborations, conversations, and hard work with some of the individuals, companies, and schools building the machines mentioned here, and to *IEEE Solid-State Circuits Magazine* Editor-in-Chief Mary Lanzerotti for the opportunity to contribute to her series, which I've found to be an exciting resource to provide my VLSI students.

References

[1] J. A. Fisher, "Trace scheduling: A technique for global microcode compaction," *IEEE Trans. Comput.*, vol. 30, no. 7, pp. 478–490, 1981.

[2] J. A. Fisher, "Very long instruction word architectures and the ELI-512," in *Proc. 10th Annu. Int. Symp. Computer Architecture*, Stockholm, Sweden, June 13–17, 1983.

[3] S. Tomita, K. Shibayama, S. Oyanagi, and H. Hagiwara, "Hardware organization of a low level parallel processor," in *Proc. IFIP Congress*, 1977.

[4] T. Kidder, *Soul of a New Machine.* Little, Brown & Company, 1981.

[5] R. P. Colwell, W. E. Hall, C. S. Joshi, D. B. Papworth, P. K. Rodman, and J. E. Tornes, "Architecture and implementation of a VLIW supercomputer," *in Proc. 1990 Conf. Supercomputing*, 1990.

[6] P. G. Lowney, S. M. Freudenberger, T. J. Karzes, W. D. Lichtenstein, R. P. Nix, J. S. O'Donnell, and J. C. Ruttenberg, "The multiflow trace scheduling compiler," *J. Supercomput.*, vol. 7, 1993.

[7] W. J. Dally, A. A. Chien, S. Fiske, G. Fyler, W. Horwat, J. Keen, R. A. Lethin, M. D. Noakes, P. Nuth, and S. S. Wills, "The message driven processor: An integrated multicomputer processing element," in *Proc. 1991 IEEE Int. Conf. Computer Design on VLSI in Computer and Processors*, Oct. 1992.

[8] D. Dobberpuhl, R. Witek, R. Allmon, R. Anglin, S. Britton, L. Chao, R. Conrad, D. Dever, B. Gieseke, G. Hoeppner, J. Kowaleski, K. Kuchler, M. Ladd, M. Leary, L. Madden, E. McLellan, D. Meyer, J. Montanaro, D. Priore, V. Rajagopalan, S. Samudrala, and S. Santhanam, "A 200 MHz 64 b dual-issue CMOS microprocessor," in *Proc. IEEE Solid-State Circuits Conf.*, 1992.

[9] E. Brooks, "Attack of the killer micros," in Proc. 1989 Conf. Supercomputing, Reno, NV, 1989.

[10] W. J. Bowhill, R. L. Allmon, S. L. Bell, E. M. Cooper, D. R. Donchin, J. H. Edmondson, T. C. Fischer, P. E. Gronowski, A. K. Jain, P. L. Kroesen, B. J. Loughlin, R. P. Preston, P. I. Rubinfeld, M. J. Smith, S. C. Thierauf, and G. M. Wolrich, "A 300 MHz 64 b quad-issue CMOS RISC microprocessor," in *Proc. IEEE Solid-State Circuits Conf.*, 1995.

[11] P. Y. Hsu, "Design of the R8000 Microprocessor," IEEE Micro, Apr. 1994.

[12] J. Ruttenberg, G. R. Gao, A. Stoutchinin, and W. Lichtenstein, "Software pipelining showdown: optimal vs. heuristic methods in a production compiler," in *Proc. ACM SIGPLAN 1996 Conf. Programming Language Design and Implementation (PLDI)*, 1996.

[13] P. Faraboschi, J. Fisher, G. Brown, G. Desoli, and F. Homewood, "Lx: A technology platform for customizable VLIW embedded processing," in *Proc. 27th Int. Symp. Computer Architecture (ISCA'27)*, June 2000.

[14] T. Utsumi, M. Ikeda, and M. Takamura, "Architecture of the VPP500 parallel supercomputer," in *Proc. Supercomputing*, 1994, pp. 478–487.

[15] S. Tomita, K. Shibayama, T. Kitamura, T. Nakata, and H. Hagiwara, "A user microprogrammable, local host computer with low-level parallelism," in *Proc. 10th Int. Symp. Computer Architecture*, 1983.

[16] Y. Nakashima, T. Kitamura, H. Tamura, M. Takiuchi, and K. Miura, "Scalar processor of the VPP500 parallel supercomputer," in *Proc. 9th Int. Conf. Supercomputing*, 1995.

[17] J. S. O'Donnell, "MAP1000A: A 5W, 230MHz VLIW mediaprocessor," in *Proc. Hot Chips Workshop*, 1999.

[18] R. Simar, Jr., "Codevelopment of the TMS320C6X VelociTI architecture and compiler," in *Proc. IEEE Acoustics, Speech and Signal Processing*, 1998.

[19] T. Shiota, K. Kawasaki, Y. Kawabe, W. Shibamoto, A. Sato, T. Hashimoto, F. Hayakawa, S. Tago, H. Okano, Y. Nakamura, H. Miyake, A. Suga, and H. Takahashi, "A 51.2 GOPS 1.0 GB/s-DMA single-chip multi-processor integrating quadruple 8-way VLIW processors," in *Proc. Int. Solid State Circuits Conf. (ISSCC'05)*, Feb. 2005.

[20] N. P. Juoppi, "The future evolution of high-performance microprocessors," in *Proc. Int. Symp. Microarchitecture*, Barcelona, Spain, p. 155.

[21] B. Khailany, T. Williams, J. Lin, E. Peters Long, M. Rygh, D. W. Tovey, and W. J. Dally, "A programmable 512 GOPS stream processor for signal, image, and video processing," in *Proc. Int. Solid State Circuits Conf. (ISSCC'08)*, Jan. 2008.

[22] S. Bell, B. Edwards, J. Amann, R. Conlin, K. Joyce, V. Leung, J. MacKay, M. Reif, B. Liewei, J. Brown, M. Mattina, C. Miao, C. Ramey, D. Wentzlaff, W. Anderson, E. Berger, N. Fairbanks, D. Khan, F. Montenegro, J. Stickney, and J. Zook, "TILE64® processor: A 64-core SoC with mesh interconnect," in *Proc. Int. Solid State Circuits Conf. (ISSCC'08)*, Feb. 2008.

[23] F. Menichelli, M. Oliveieri, S. Smorfa, and I. Zaccardini, "Software optimization of the JPEG2000 algorithm on a VLIW CPU core for system-on-chip implementation," in *Proc. 3rd IASTED International Conf. Circuits, Signals, and Systems*, Oct. 2005.

[24] T-H. Tsai, Y-N. Pan, and L-T. Tsai, "DSP platform-based JPEG2000 encoder with fast EBCOT algorithm," in *Proc. SPIE Embedded Processors for Multimedia and Communications*, 2004.

[25] K. Datta, M. Murphy, V. Volkov, S. W. Williams, J. Carter, L. Oliker, D. A. Patterson, J. Shalf, and K. A. Yelick, "Stencil computation optimization and autotuning on state-of-the-art multicore architectures," in *Proc. Supercomputing (SC'08)*, Nov. 2008.

[26] B. Meister, A. Leung, N. Vasilache, D. Wohlford, C. Bastoul, and R. Lethin, "Productivity via automatic code generation for PGAS platforms with the R-stream compiler," in *Workshop on Asynchrony in the PGAS Programming Model, Colocated with the 23rd Int. Conf. Supercomputing*, June, 2009.

About the Author

Richard A. Lethin received his B.S. degree in electrical engineering from Yale College in 1985, and his M.S. and Ph.D. degrees from The Massachusetts Institute of Technology in 1992 and 1997. He was a member of the CPU development team at Multiflow Computer. After MIT, he formed Reservoir Labs into an independent research laboratory focusing generally on solving the software mapping problems for advanced computer architectures. He teaches VLSI systems design as an adjunct associate professor in the School of Engineering at Yale, and he is a Hertz Foundation Fellow. He is a Senior Member of the IEEE.