Vector Architectures: Past, Present and Future

Roger Espasa, Mateo Valero*

Computer Architecture Dept.
U. Politècnica de Catalunya-Barcelona
{roger,mateo}@ac.upc.es
http://www.ac.upc.es/hpc

James E. Smith†

Dept. of Electrical & Computer Engr.
University of Wisconsin-Madison
Madison, WI 53706
jes@ece.wisc.edu

Abstract

Vector architectures have long been the of choice for building supercomputers. They first appeared in the early seventies and had a long period of unquestioned dominance from the time the CRAY-1 was introduced in 1976 until the the appearance of "killer micros", in 1991. They still have a foothold in the supercomputer marketplace, although their continued viability, in the face of micro-based parallel systems, is being seriously questioned. We present a brief history of supercomputing and discuss the merits of vector architectures. Then we relate the advantages of vector architectures with current trends in computer system and device technology. Although the viability of vector supercomputers is indeed questionable, largely because of cost issues, we argue that vector architectures have a long future ahead of them – with new applications and commodity implementations. Vector instruction sets have many fundamental advantages and deserve serious consideration for implementation in next generation computer systems, where graphics and other multimedia applications will abound.

1 A Brief History of Supercomputing

To better understand vector supercomputers, we begin with a brief history. And to provide broader perspective, we consider supercomputers in general; we don't restrict ourselves to vector machines.

1.1 The Six Hundreds: Roots of Supercomputing

Although the term wasn't coined until later, supercomputing probably began with the Control Data 6600 and 7600 [1]. The 6600 was rolled out in 1963, and the 7600 in 1969. They were scalar machines and were the first to use RISC instruction sets. James Thornton and Seymour Cray were co-developers of the 6600. Thornton went on to develop the STAR-100 [2], while Cray continued with the 7600 and the unfinished 8600 before leaving CDC to found Cray Research.

*This work was supported by the Ministry of Education of Spain under contract 0429/95 and by the CEPBA.

†This work was supported in part by NSF Grant MIP-9505853.

Although they were scalar machines, they had many characteristics of supercomputers. The 6600 and 7600 were focused on numerical processing. They implemented 60-bit floating point arithmetic and were constructed of what at the time were exotic technologies. Their clock cycles and overall performance levels were far superior to their closest competition. They had large memory capacity and very high I/O bandwidth. And their price tag was very high: many millions of dollars (in 1960s dollars).

1.2 The Rise of Vector Supercomputers

The first two vector supercomputers appeared in the early 1970s. The two machines were remarkably similar. One was developed at Texas Instruments, the TI-ASC [3]. It was Texas Instruments' only venture into large scale computers and only a handful were ever sold. The other was developed at Control Data Corporation, the STAR-100. Only a few STAR-100s sold as well, but it was the beginning of a series of vector machines developed at CDC over the years.

The ASC and the STAR-100 were centered around a very high-powered vector unit that took streams of operands from memory, operated on them, then sent the result streams back to memory, all in a single instruction. The primary job of the scalar unit was to service the vector unit – to do bookkeeping computation and do any scalar code that couldn't be done with vectors. Because they were memory-to-memory vector machines, they were built with very advanced, high bandwidth memory systems. A pipeline that stretched from memory, through the processor, and back to memory was very long and took many clock cycles to fill, but once it was filled, the throughput was tremendous.

Both machines used instruction sets that would be categorized as "CISC" today. For example, in a single vector instruction, the ASC could do a complete matrix multiplication. STAR-100 instructions operated on arrays of numerical data and on strings of characters and bits; STAR stood for STring ARray. The STAR-100 also had features to support sparse matrix operations. Many of the instructions had so much semantic content, it is doubtful that a compiler would be able to use them as targets for automatic vectorization; they could most easily be used in assembly-coded library routines.

Although they are remembered mostly for their ultimately unsuccessful memory-to-memory architectures, the ASC and STAR-100 also contained several innovations that have lived on until today. One example is the method they used for implementing conditional operations, i.e. bit masks for control. Data movement operations implemented with

stride and gather/scatter memory accesses are still used today. They could split their pipelines to double performance on 32-bit data. The ASC could be upgraded by adding additional vector pipelines, up to a total of four. And the STAR-100 also supported multiple vector pipes, although their functions were not identical.

Both machines had relatively poor scalar performance, and Amdahl's law took its toll. The CDC 7600, a scalar-only contemporary of the STAR-100, was generally much faster on all but very highly vectorized problems. In the case of the ASC, there were probably business decisions that also led to its demise; this simply wasn't TI's market.

1.3 The Golden Era of Vector Supercomputers

It wasn't until the introduction of the CRAY-1 in 1976 that vector supercomputing became successful [4]. The CRAY-1 was designed with a different philosophy than the STAR-100 and ASC. In a sense it was centered around scalar processing. The scalar unit was similar to the CDC 7600, an earlier Seymour Cray-designed machine, and was capable of much higher scalar performance than any of its contemporaries. The only substantial hardware added for supporting vector operations was the vector registers and interconnect paths for moving data between the vector registers and the functional units. The memory path was capable of only one load or store per cycle – all that was needed for scalar operation, and vector registers were added to maximize the use of this rather limited memory bandwidth. The large scalar floating point functional units were shared by the vector instructions – with slightly longer latencies for vector processing.

The CRAY-1 was the beginning of a "Golden Era" of vector supercomputers – and supercomputing in general, for that matter. This era spans approximately 15 years from 1976 until 1991. This period can be characterized as a time when a vector supercomputer had a major, unquestioned performance advantage over microprocessors in all aspects: faster clock cycles, more operations per cycle, higher memory bandwidth, and much more I/O capacity.

Cray Research, by far the most successful supercomputer vendor, continued its development of vector machines following two parallel lines. Seymour Cray went on to design the multiprocessor CRAY-2 as a follow-on to the CRAY-1. The CRAY-2 resembled the CRAY-1 in many ways. For example, it had a single memory port and one set of parallel functional units in each vector processor. Some interesting innovations appeared, as well. The CRAY-2 had 16K words (64-bit words, of course), of local memory which had to be explicitly managed by the compiler/programmer. At the same time the CRAY-2 was designed, another group at Cray Research developed the CRAY X-MP system. The emphasis in the X-MP was on multiprocessing, and the memory bandwidth per processor was expanded to four words per cycle (three for processing, and one for I/O). It was with huge success of the X-MP and the follow-on Y-MP that grew Cray Research into a large company.

Meanwhile, Control Data continued the development of vector machines based on the STAR-100 architecture. The Cyber 200 series machines continued with memory-to-memory vector operation, but had considerably improved scalar capability. Eventually, the Cyber 200 series was followed by the ETA-10 [5, 6], when ETA spun out of CDC. And the line ceased when ETA went out of business in 1989.

During this time period the Japanese manufacturers also entered the vector supercomputer market with several designs. In the early eighties, NEC, Fujitsu and Hitachi each introduced a line of parallel vector computers. Using their expertise in chip technology and emphasizing multiple vector pipelines per processor, all three manufacturers succeeded in developing very fast vector machines. As our later graphs will show, the vector processors used in these machines are the most powerful uniprocessors ever built (see the NEC SX-3 at 5.5 Gflops, for example).

Following the success of the big iron vendors, in the early 1980's several start-up companies designed and sold 'mini-supercomputers'. These were small vector machines, with slower clocks than the big supercomputers and less processing power per vector unit. The most successful example was Convex Computer Corporation. It's first machine, the C-1, was introduced in 1985 with a clock rate of 10Mhz (much slower than a contemporary 160Mhz SX-2). The selling point of these mini-supers was that they provided limited supercomputing capabilities at a much more affordable price (typically $0.5 to 1$ million dollars versus $5 to 30$ million).

1.4 Killer Micros and the Emergence of Large Parallel Systems

In the early nineties, advances in CMOS VLSI technology introduced a radical change in the computer industry. Many more transistors could be fit in a single die and, moreover, the die could be clocked faster as feature sizes shrank. No longer was it true that breaking the 100Mhz barrier required expensive ECL technology. Since then, microprocessors have continued to evolve at an impressive pace. Indeed, peak microprocessor performance has been improving at a rate of 1.6X per year, rapidly approaching the performance of vector supercomputers.

Perhaps the DEC Alpha family is the best example of the so-called "killer micros". Introduced in 1992 at an astonishing 150Mhz (reminiscent of the astonishing CDC and Cray clock cycles of their generation), successive processors in the Alpha family have seen rapidly improving clock cycles to the point that they now surpass the cycle times of the fastest supercomputers (see fig. 2). As early as 1994, the Alpha 21164 already was clocked faster (300Mhz) than the contemporary CRAY C90 (240Mhz). Since then, clock frequency has been steadily increasing and the new Alpha 21264 [7] clock will be at least 1.5 times faster than the Cray T90 (450Mhz).

The introduction of fast microprocessors substantially changed the supercomputing market. Due to their much higher volumes, microprocessors offer very low prices per processor. As soon as cheap and powerful processors appeared in the market, the idea of building supercomputers using many of these processors spread rapidly. Before long, supercomputer customers were talking in terms of Mflops per dollar, rather than absolute peak performance.

Parallel machines built of microprocessors are offered as an attractive alternative to vector supercomputers (we will call them scalar-parallel machines, to distinguish them from parallel vector supercomputers –PVPs). Smaller scalar-parallel machines can be built around a bus with uniform memory access delays to give a symmetric multiprocessor (SMP), or many more processors can be combined with direct interconnection networks to form massive parallel systems (MPP). The success of scalar-parallel machines has been based mostly on leveraging CMOS microprocessor technology and DRAM main memory systems, yielding computers that have a high performance at a low cost. Although each node in a scalar-parallel system is less powerful than a vector processor, these machines are built with the ability to scale to large numbers

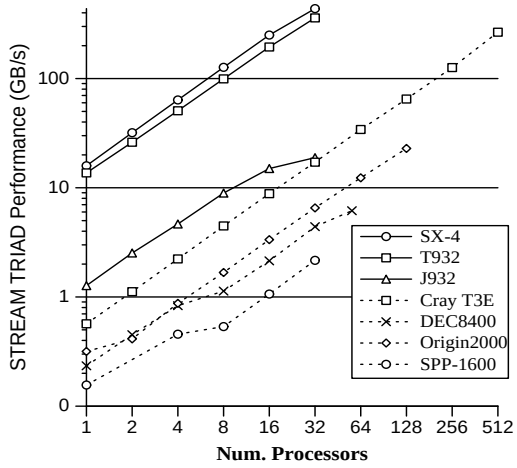

Figure 1: Performance on the TRIAD operation for several supercomputers (source: [8]).

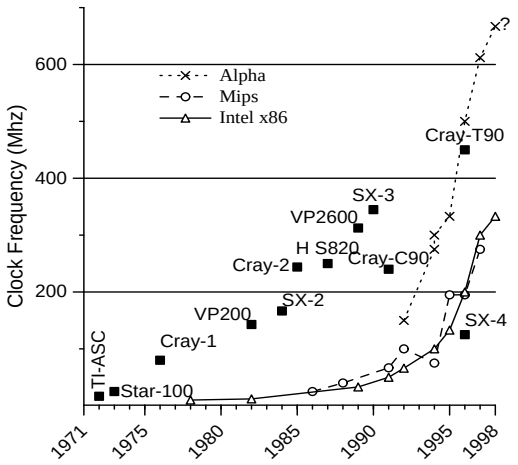

Figure 2: Evolution of clock frequency over the years for vector supercomputers and for several microprocessors (source: vendor information).

of nodes. Therefore, at their best, by using parallel programming and compilation techniques the processing power of each processor can be magnified by orders of magnitude in a large parallel system. On the other hand, some problems are difficult to program in parallel or have memory access patterns that make distributed memory parallel systems less effective than their vector counterparts.

To illustrate the importance of memory systems, figure 1 presents sustained performance of several PVPs, MPPs, and SMPs. This figure plots data gathered using the STREAM [8] benchmark. In particular, we present the performance of the TRIAD operations (performance is measured in Gigabytes per second). The core of the TRIAD benchmark consists of the following operation: $a(i) = b(i) + q \times c(i)$.

Figure 1 shows that even when using large numbers of processors, top-of-the-line parallel vector processors still outperform large MPPs like the T3E. Even the relatively modest J90 sustains much larger bandwidths than most parallel servers. The memory accessing patterns of the TRIAD benchmark are designed to have relatively low temporal locality. However, not all applications match the needs of the TRIAD operation. In fact, there is a significant number of cache-friendly scientific applications that run very well on parallel machines.

Commercial Vector Supercomputers

Table 1 summarizes the main characteristics of the major vector supercomputers built and sold from 1972 until 1996.

For each machine in the table we indicate the year of introduction and the basic cycle time of the machine (in ns units). The next column, labeled "LD/ST paths" indicates the number of ports connecting the memory system to the vector register files. There are three kinds of ports: an "L" by itself indicates a stand-alone load port. An "S" by itself indicates a stand-alone store port, while an "LS" indicates a bidirectional port that can do both loads and stores. For example, the SX-2 had 2 different ports: one port was a load-only port while the other was a bidirectional load/store port. Similarly, the Cray C90 has three ports: two load-only ports and one store-only port (this does not include the I/O port). The table also includes the number of vector registers and their sizes, as well as the number and type of functional units.

Column "Pipes" indicates the number of replicated vector pipes existing in the maximum configuration of each machine. For example, the Fujitsu VP2600 is a 4-pipe machine, indicating that each of its main vector resources is replicated four times. Finally, columns "Flops/Cycle" and "Words/Cycle" indicate the peak number of floating point operations per cycle and the maximum data transfer rate (also per cycle) between the register file and main memory. For machines having asymmetric load/store ports, the following notation is used. For the SX-2, with one load-only port and one load/store port, the maximum transfer rate is 8 words when performing loads and 4 words when doing stores. This situation is noted as "8 or 4". For machines with independent load/store ports, such as the Y-MP, the notation "2+1" indicates that it can perform 2 loads and 1 store per cycle.

Performance Trends

Figures 2 through 4 illustrate the historical trends of three key aspects of vector architectures: clock frequency (fig. 2), peak MFLOPS (fig. 3) and peak memory-to-register bandwidth (fig. 4). The three figures also include data for typical microprocessors, including the Intel x86 line, the MIPS line, and the more recent DEC Alpha family.

As shown in figure 2, beginning with the CRAY-1 in 1976 and continuing until 1991, the clock frequency of vector supercomputers was between 7 to 10 times faster than contemporary microprocessors. For example, in 1978, the clock ratio between a CRAY-1 and an x86 was about a factor of 8. Eight years later, in 1986, the CRAY-2 (245Mhz) when compared with the recently introduced R2000 (25 Mhz) yields a factor of almost 10. As late as 1990, the NEC SX-3 (340Mhz) compared very favorably against a 40-50Mhz R3000.

During the golden era, vector machines not only had much faster clocks, but also performed many more operations per clock cycle. Referring to table 1 we see that vector units can perform from 2 to 16 floating point operations per cycle. Typical micros perform either 1 or 2 floating point operations per cycle at most (the Power and R8000 chips being the most notable exceptions). Figure 3 plots the evolution of peak MFLOPS per processor from 1976 to 1998. The combination of fast clocks and multiple operations per cycle yielded machines with peak performance that was between 16 and 70 times better than contemporary microprocessors. For example, the SX-3, with its 5.5 Gflops, had around 68 times more Mflops than an 80Mflops R3000.

The third important measure for high performance computing is memory bandwidth, that is, peak number of bytes

Machine	Year intro	Cycle time	LD/ST paths	Vector Registers	Elements/ Register	Functional Units	Pipes	Flops/ cycle	Words/ cycle
TI-ASC	1972	60.0	LS	–	–	A/M	4	4	4 (32bit)
STAR-100	1973	40.0	L,L,S	–	–	A/D/L, A/M	1	2	3
CRAY-1	1976	12.5	LS	8	64	A,M,R,I,L,S	1	2	1
Fujitsu VP200	1982	7.0	LS,LS	256–8	1024–32	A/L,M,D	2	4	4
Cray X-MP	1983	9.5	L,L,S	8	64	A,M,R,I,L,L,S,P	1	2	2+1
Hitachi S810/20	1983	19.0	L,L,L,LS	32	256	A/L,A/L,M/D+A,M+A	2	12	8 or 2
NEC SX-2	1984	6.0	L,LS	8+8K	256/64–256	A,M/D,L,S	4	16	8 or 4
CRAY-2	1985	4.1	LS	8	64	A,M/R/Q,I,L	1	2	1
Hitachi S820/80	1987	4.0	L,LS	32	512	A/L,M+A,D	4	12	8 or 4
Cray Y-MP	1988	6.3	L,L,S	8	64	A,M/L,R,I,L,S,P	1	2	2+1
Fujitsu VP2600	1989	3.2	LS,LS	2048–64	64–2048	M+A/L,M+A/L,D	4	16	8
NEC SX-3	1990	2.9	L,L,S	8+16K	256/64–256	A/S,A/S,M/L,M/L	4	16	8+4
Cray C90	1992	4.0	L,L,S	8	128	A,M/L,R,I,L,S,P	2	4	4+2
NEC SX-4	1996	8.0	LS,LS	8+16K	256/64–256	A/S,M,D,L	8	16	16

Table 1: Functional unit and register file characteristics of several vector supercomputers. Legend: A=FP-Add, D=FP-Divide, I=Integer-Add, L=Logical, M=FP-Multiply, P=Population Count/Parity, R=Reciprocal Approximation, Q=Reciprocal Square Root, S=Shift. Independent functional units are separated by commas. Functional units that perform several operations are indicated using a slash mark (A/S, for example). Cascaded units, that is units that hang off other functional units, are indicated using a "+" sign (for example, "M+A").

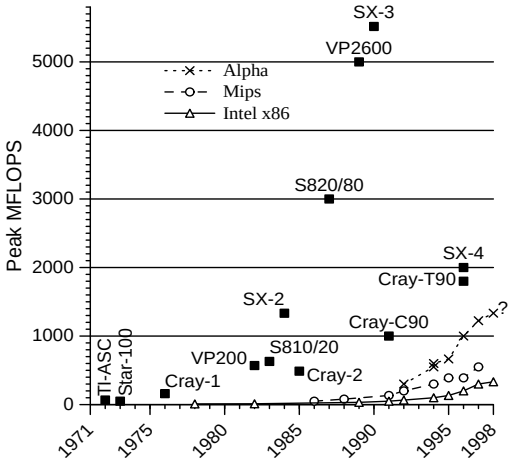

Figure 3: Evolution of peak MFLOPS over the years for vector supercomputers and for several microprocessors (source: vendor information).

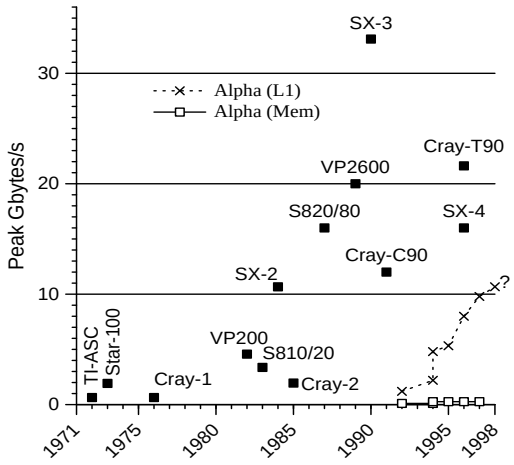

Figure 4: Evolution of peak main memory bandwidth for vector supercomputers and for an Alpha-based Workstation. The values for the Alpha main memory bandwidths are derived using the 33Mhz 21072 and 21172 chipsets [9].

transferred between main memory and the register file. Figure 4 shows the peak memory bandwidths for several vector supercomputers. Also included in the graph are the memory bandwidths for Alpha workstations. For microprocessors such as the Alpha, determining peak memory bandwidth is not easy. We have included two different measures. First, peak bandwidth between the first level cache and the register file (curve L1). Second, peak memory bandwidth between main memory and the second or third level caches (curve Mem).

Again, vector supercomputers have a very large advantage in memory bandwidth over microprocessors. Even when considering the most recent and fastest microprocessor, the Alpha 21264, peak memory bandwidth comparisons can be misleading. The 21264 can provide about 10GB/s to its L1 cache, only a factor of 1.6 smaller than a single SX-4 processor. However, the 21264's L1 cache can only hold 64KBytes of data. Meanwhile, the SX-4 offers 16GB per second to a main memory of 8 Gigabytes. Multiplying bandwidth times memory size being accessed (10GB/s times 64KBytes and 16GB/s times 8GB) we get an aggregate measure of the ability to access data. Comparing the two numbers, we see that the SX-4 has an aggregate bandwidth 200,000 times larger than the Alpha 21264. Clearly, the Alpha will only match the SX-4 performance on applications where most data fits entirely in the L1 cache.

Current status of vector supercomputers

Since the early nineties, supercomputers based on the vector paradigm have lost their dominance of the supercomputing market. Consider the "Top500 Supercomputer Sites" list [10], published every 6 months since 1993. The list includes the sites that have the world's 500 most powerful computer systems installed. In June 1993, of the top 500 computers, 310 were parallel-vector machines. All the machines included in the list at that time totaled a peak computing power of 1.8 Teraflops. The 310 vector systems represented roughly 43% of all that computing power. Four and a half years later, in November 97, the same list reports that only 108 PVP's are still in the top-500 systems. Moreover, the total peak power of all systems listed had sky-rocketed to 24.2 Teraflops, but now the vector machines only accounted for 17% of this power.

Why have vector machines declined so fast in popularity? What are the key factors that have helped in the rise of parallel microprocessor based machines?

The short answer is cost. There are many problems that are amenable to vectorization, yet very few users can afford a full blown vector supercomputer. Why are vector supercomputers so much more expensive than MPPs or SMPs ? There are several related reasons:

- Probably the most important reason is that scalar-parallel systems use commodity parts. With commodity parts, design and non-recurring manufacturing costs can be spread over a larger number of chips. If a vector machine only sells a few dozen copies, then design costs can easily be the dominant overall cost.
- The most expensive part of a computer (whether a PC, workstation, or supercomputer) is usually the memory system. Consider figure 4 again. Vector supercomputers provide high performance memory systems that sustain very large bandwidths between main memory and the vector registers. To achieve this bandwidth, vector processors rely on high-performance, highly interleaved memory systems (between 256 and 1024 memory banks depending on processor and configuration [11]). Moreover, for a high performance machine, latency also plays an important role. Therefore, vector supercomputers use the fastest memory technology available. Typically, a vector memory system will be composed of SRAM/SSRAM memory modules [12] with cycle times in the 10–20ns range; this allows keeping main memory latency around 10 to 30 processor cycles.
- Another problem is how one packages a processor with such high bandwidths. That is, consider a 20GB/s memory system and a typical CMOS package that allows its pins to operate at 133Mhz. A back-of-the-envelope calculation indicates that 1200 pins (just for data) would be needed to sustain a peak of 20GB/s second. Such numbers of pins are difficult to implement. In the past, vector manufacturers have employed multi-chip designs. These designs tend to be substantially more expensive than single-chip solutions.
- Another factor that keeps vector costs up is the base technology used in these machines. Up to very recently, most vector designs were based on ECL. While this choice was adequate in the 1976–1991 time frame, vector vendors apparently failed to realize the potential of CMOS implementations. Nor were they willing to shift from gate array to custom design in order to exploit the capabilities of CMOS. In the last 8 years, CMOS chips have outperformed ECL in numbers of transistors, speed, and reliability. Recently, most vector vendors have introduced CMOS-based vector machines (like the J90 or SX-4).
- Also important is the fact that users often have difficulty achieving peak performance on vector supercomputers [13, 14, 15, 16]. Despite high performance processors and high bandwidth memory systems, even programs that are highly vectorized fall short of theoretical peak performance [17].
- Finally, it is important to note that there have been relatively few architectural innovations since the CRAY-1. The top of the line CRAY T90 still has only 8 vector registers and has a relatively slow scalar microarchitecture when compared to current superscalar microprocessors. Meanwhile, superscalar microprocessors have adopted many architectural features to increase performance while still retaining low cost.

2 Future Vector Applications and Technologies

It is important to distinguish between vector instruction set architectures (ISAs) and vector supercomputers. The previous section established several reasons we believe that vector supercomputers are in a period of diminished popularity. However, we believe the architectural concept behind a vector supercomputer, its vector instruction set, remains very viable.

Vector ISA's have significant advantages for future technology. As feature size decreases and we rapidly approach wire delay limitations, a vector ISA has major benefits in providing ever-increasing levels of performance. In this section we present the advantages of vector ISA's and relate them to current technology constraints. It is important to stress that we are no longer talking only about supercomputers. We believe that vector ISA's have a wide range of application, from DSP's and multimedia-oriented chips to general purpose CMOS microprocessors.

Advantages of Vector ISA's

Advantages of vector ISA's over scalar or VLIW ISA's can be placed in three broad categories. First, semantic advantages; that is, vector ISA's tend to express programs in a more concise and efficient way. Second, explicit parallelism is encoded in each vector instruction, thus allowing for highly parallel implementations. Third, the combination of regularity in each vector instruction and explicit parallelism allows for very aggressive design techniques, such as heavy pipelining, functional unit replication and aggressive clocking. Let's look at each of these advantages in turn.

Number of instructions executed

The main difference between a vector and a scalar instruction is that a single vector instruction specifies a large number of operations. Thus, to perform a given task, a vector program executes far fewer instructions than a scalar program. The scalar program has to specify address computations, loop counter increments, and branch computations that are typically implicit in vector instructions.

To illustrate this point, figure 5 presents a comparison of the number of instructions executed on a vector machine (a Convex C34) and on a superscalar machine (a MIPS R10000). We selected seven programs from the Specfp92 suite and compiled them on each machine. We then ran each program on the two machines and counted the total number of instructions executed. As can be seen, the differences are huge.

This instruction count difference translates into several positive effects. First, instruction fetch bandwidth is greatly reduced. Second, execution of branches can be hidden "underneath" the execution of vector operations, thereby hiding most branch misprediction latencies. Third, even a relatively simple control unit that fetches and decodes just one instruction per cycle can be enough to sustain a very large computation rate. Overall, vector instruction sets allow implementations with simple control units. In turn, control simplicity can yield an aggressive clocking of the whole processor.

Number of operations executed

The comparison in terms of instructions executed presented in the previous section is very important for its overall effect on the fetch engine. However, to correctly gauge the overhead of a typical scalar ISA over a vector ISA we should compare the total number of operations executed. Figure 6

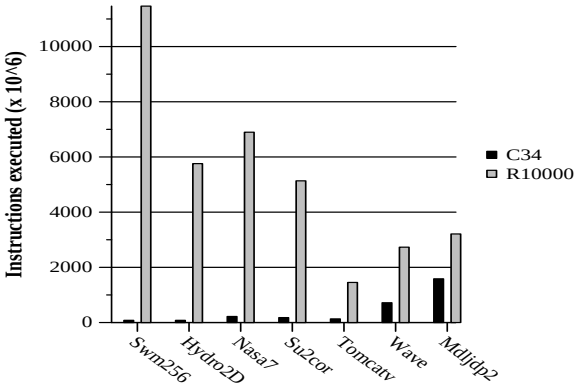

Figure 5: Number of instructions executed on the R10000 and Convex C34 machines. On the R10000 we used the hardware counters to gather the number of graduated instructions per program. On the Convex machine we used the Dixie tracing tool [18] to gather instruction traces and count total number of executed instructions.

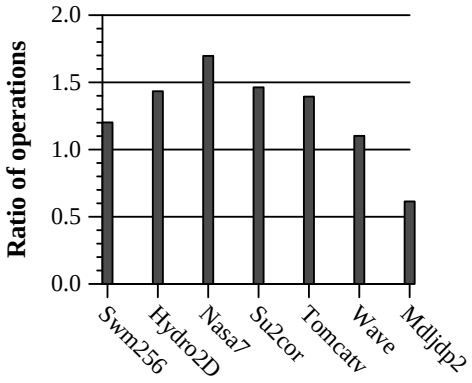

Figure 6: A comparison of the number of operations executed on the R10000 and the Convex C34. The data presented is the ratio of R10000 operations over C34 operations. Numbers above 1.0 indicate an advantage for the vector C34 machine.

presents such a comparison. For each program we compute the ratio of operations executed on the Mips R10000 machine over the number of operations executed on the Convex C34. For example, program `nasa7` executes 1.7 more operations when run on the scalar machine than when run on the vector machine.

As Figure 6 shows, for all programs but one, the vector version of each program executes many fewer operations than the scalar version. The one exception, `mdljdp2`, is due to a combination of heavily nested IF constructs in the main loop, high register pressure, and lack of support for multiple vector mask registers. We believe that, if compiled on a vector machine without these limitations, such as the SX-4, `mdljdp2` would also show fewer operations executed.

The overall conclusion is that a vector ISA expresses a given program in many fewer operations than a scalar ISA because many operations, such as address computations, are implicit in vector instructions while they must be explicitly coded in a scalar ISA.

Memory system performance

Due to the increasing gap between memory and processor speeds, current superscalar micros need increasingly larger caches to maintain performance improvements. Nonetheless, despite out-of-order execution, non-blocking caches, and prefetching, superscalar micros do not make efficient use of their memory hierarchies. The main reason for this comes from the inherently predictive model embedded in cache designs. Whenever a line is brought from the next level in the memory hierarchy, it is not known whether all data will be needed or not (and often it is not). Moreover, it is very uncommon for superscalar machines to sustain the full bandwidth that their first level caches can potentially deliver [19]. Since load/store instructions are mixed with computation and setup code, dependencies and resource constraints prevent a memory operation from being launched every cycle.

In contrast, the vector style of accessing memory has the following advantages. First, every data item requested by the processor is actually used. There is no implicit (sometimes wasted) prefetching due to cache lines. Second, information about memory access pattern is conveyed directly to the hardware through the stride value. This information can be used in a variety of ways to improve memory system performance.

When it comes to memory latency, a vector memory instruction can amortize long memory latencies over many elements. Several studies [17, 20, 21, 18] have shown that by using some superscalar-like techniques coupled with a vector engine, up to 100 cycles of main memory latency can be tolerated with a very small performance degradation.

Regarding memory bandwidth, a vector machine can make much more effective usage of whatever bandwidth it is provided. While a superscalar processor requires extra issue slots and decode hardware to exploit more ports to the first level cache, a vector machine can request several data items with a single memory address. For example, when doing a stride-1 vector memory access, a vector processor need not send every single address to the memory system. Simply sending every Nth address, a bandwidth of N words per cycle can be achieved.

Datapath Control

In order to scale current superscalar performance up to, say, 20 instructions per cycle, an inordinate amount of effort is needed. The dispatch window and reorder buffers required for such a machine are very complex. The wakeup and select logic grows quadratically with the number of entries, so the larger the window the more difficult is to build such an engine [22]. If current superscalars use 4-wide dispatch logic and barely sustain one instruction per cycle, a superscalar machine that sustains 20 operations per cycle appears infeasible.

On the other hand, a vector engine can be easily scaled to higher levels of parallelism by simply adding vector pipes and adding wider paths from the vector registers to the functional units. All this without increasing the complexity or the pressure on the decode unit. The semantic content of the vector instructions already includes the notion of parallel operations.

Low Power and Real-Time Performance

For many future applications, low power and real-time performance will be major factors. And, as odd as it may seem, vector architectures have significant advantages for both.

Vector instructions have the property of "localizing" computations. That is, once a vector instruction starts operating, only the functional unit and register busses feeding it

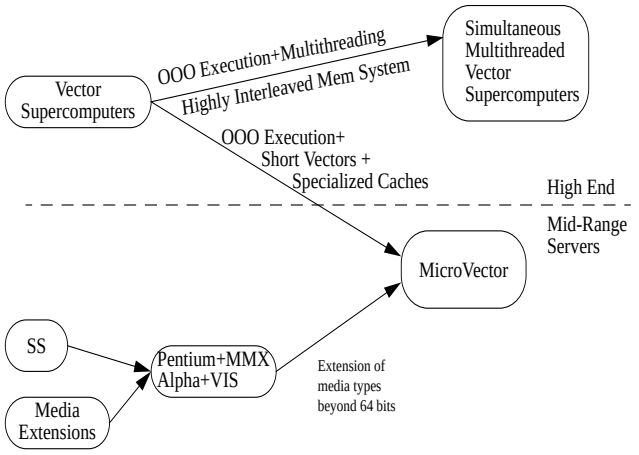

Figure 7: Possible evolution of vector architectures

need to be powered. The instruction fetch unit, the reorder buffer and other large power-hungry blocks of the processor can be powered off until the vector instruction finishes execution.

A designer using a vector ISA can easily balance performance against power consumption. For example, two vector instructions could be working in parallel if needed to meet a certain performance goal. Or, alternatively, a single vector instruction could be executed twice as fast by using a double-pipe. In both alternatives, the extra power consumption is kept at the absolute minimum needed to run the extra functional units and busses. No power is wasted due to continuously re-fetching the same loop instructions, as it would happen in a superscalar implementation.

Regarding real-time applications, vector machines can be constructed with both highly predictable and deterministic behavior. The CRAY-1 had extremely deterministic timing – probably more so than any high performance system constructed before or since. With vectors, performance features that give real-time designer headaches, like caches and branch predictors, aren't necessary for high performance. The highly structured character and natural latency-hiding features of vector architectures permit implementations that are both simple and deterministic, yet retain their potential for high performance.

3 A Typical Vector Architecture of the Future

There are a couple of directions in which vector architectures can evolve, as shown in figure 7. First, vector machines will continue to have a percentage of the high end supercomputer market, at least in the near-term. Second, superscalar architectures and vector ISA's will be merged to tackle the needs of multimedia applications.

High-End Supercomputer Market

We think vector architectures will continue in the high-end supercomputer market, to cover a certain subset of scientific applications that do not fit the SMP or MPP architecture model. However, the critical issue is whether there will be *enough* customers and enough machines sold to recover the development costs of these specialized and expensive machines. As defended in a previous paper, a possible path of evolution of these high-end machine could be the mixture of all high performance paradigms used today: vector processing, superscalar processing and multithreaded processing [23].

High performance microprocessors

The other path links the future of vector architectures to the evolution of today's commodity microprocessors. As noted elsewhere [24], in fewer than 10 years application execution times will be dominated by multimedia tasks. That is, it is very likely that video and audio processing, image rendering, etc., will be the dominant portion of future applications. As the human-computer interface improves, there will be more demand for high quality 3D graphics, regardless of the particular application at hand. Thus, processors will have to evolve to accommodate the tremendous bandwidth and computation needs of these types of applications.

Today, we already see in the major microprocessor families a set of extensions targeting the multimedia market (MMX [25], VIS [26], etc.). These multimedia extensions are simple vector-like instructions that operate on parts of a 64-bit word. Extending these limited vector instructions into more general ones, like those found in modern vector ISAs is relatively simple.

Research performed on traditional vector architectures shows a couple of promising directions for fully integrating a vector and a scalar processor. First, adding out-of-order execution to a vector processor, the ability to tolerate large memory latencies is increased and performance improves substantially [18]. Second, once the large memory latency problem is more or less solved, one can reduce the length of each vector register from 128 elements to 8 or 16 [27]. This reduction in register length leads to two very interesting consequences: first, the area occupied by the vector register file can fit into a microprocessor without compromising other critical components (such as instruction and data caches or branch predictor). Second, a single vector register looks a lot like a cache line from the second level cache. That is, today's L2 caches have line sizes in the 128 to 256 bytes range. A 16 element vector occupies $16 \times 8 = 128$ bytes. One can use this property to devise a cache hierarchy that fits these small vector registers. A possible example of such architecture is shown in figure 8. It is a narrow-width out-of-order superscalar enhanced with a powerful vector unit. The vector unit has 16 to 64 short vectors (8 or 16 elements each) and functional units that can do three types of operations: integer, floating point and multimedia. There is a wide path (1024 bits wide) between the vector-cache and the vector registers, so that a full register can be loaded with a cache line, if properly aligned, in a single cycle. The vector-cache only holds stride-1 vector memory accesses, while the scalar-cache holds scalar data and non-unit stride vector accesses. We believe this architecture can perform very well on media-intensive and floating point codes. For scalar codes, its performance would be roughly equal to a conventional superscalar machine.

4 Conclusions

When thinking about vector architectures, it is natural to envision huge liquid-cooled mainframes, built out of ECL components packed together using exotic technologies and extremely expensive memory systems. If this is all they are, they could easily vanish in the next decade. However, we believe that vector architectures when implemented with commodity components have great potential for the future. They have substantial advantages for future generation computer systems. As technology progresses and processor ar-

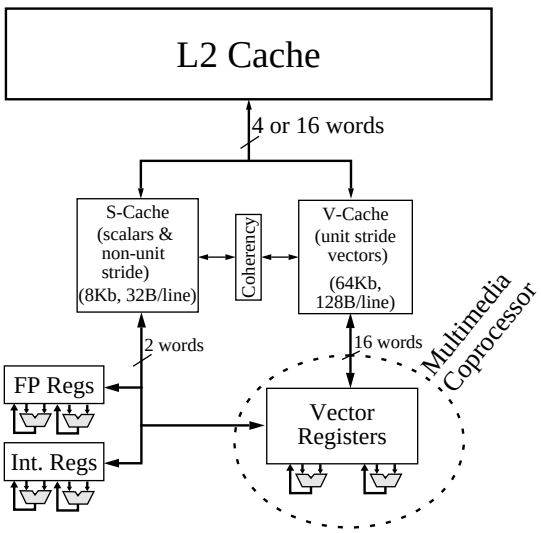

Figure 8: A possible future implementation of a micro-vector processor.

chitects are faced with the challenge of extracting more parallelism from programs, we believe that vector architectures with their many inherent advantages and suitability for future applications will come to the fore.

Vector instruction sets provide natural way to express data-level parallelism. This parallelism can be used in several ways: to improve performance by executing operations in parallel, to aggressively clock a design by deeply pipelining it, or to reduce power consumption by turning off all units not needed during the execution of a long-running vector instruction. Moreover, vector instructions express a program in a more compact way, which means fetch and decode bandwidths are more effectively utilized. Finally, accessing memory using vectors has many advantages: only useful data is requested, spatial locality can be exploited by requesting multiple data items with a single address, and stride information can be used by the hardware to optimize memory accesses.

All these features combined make vector instruction sets ideal for the next round of high performance microprocessors. They can provide high performance at a low complexity and are very well suited for tomorrow's applications: bandwidth hungry multimedia programs are the perfect fit for vector architectures.

Acknowledgments

We would like to thank Francisca Quintana for providing the data on the R10000 instruction and operation numbers.

References

[1] J. E. Thornton. *Design of a Computer–The Control Data 6600*. Scott, Foresman, Glenview, Ill., 1970.

[2] R. G. Hintz and D. P. Tate. Control data STAR-100 processor design. In *Proc. Compcon 72*, pages 1–4, New York, 1972. IEEE Computer Society Conf. 1972, IEEE.

[3] W. Watson. The TI-ASC, A highly modular and flexible super computer architecture. *Proc. AFIPS*, 41, pt. 1:221–228, 1972.

[4] R. M. Russell. The CRAY-1 computer system. *Communications of the ACM*, 21(1):63–72, January 1978.

[5] Dennis Fazio. It's really much more fun building a supercomputer than it is simply inventing one. In *SPRING COMPCON'87*, pages 102–105, San Francisco, USA, February 23-27 1987. IEEE Computer Society Press.

[6] R. A. Fatoohi. Vector performance analysis of three supercomputers: Cray 2, cray Y-MP, and ETA 10-Q. In *Proceedings of the Supercomputing 89*, pages 779–788, Reno, NV USA, 1989. ACM Press , New York, NY , USA.

[7] Jim Keller. The 21264: A Superscalar Alpha Processor with Out-of-Order Execution. In *Microprocessor Forum*, October 1996.

[8] John D. McCalpin. Memory Bandwidth and Machine Balance in Current High Performance Computers. *IEEE TCCA Newsletter*, December 1995.

[9] Digital Equipment Corporation, Maynard, Massachusetts. *DECchip 21071 and DECchip 21072 Core Logic Chipsets –Data Sheet*, EC-QAEMB-TE edition, January 1996.

[10] Jack K. Dongarra, Hans W. Meuer, and Erich Strohmaier. TOP500 Supercomputer Sites. In *IEEE SC97 Conference*, November 1997.

[11] Margaret L. Simmons, Harvey J. Wasserman, Olaf M. Lubeck, Christopher Eoyang, Raul Mendez, Hiroo Harada, and Misako Ishiguro. A performance comparison of four supercomputers. *Communications of the ACM*, 35(3):116–124, 1992.

[12] Betty Prince. *High Performance Memories*. Wiley & Sons, Ltd., 1996.

[13] Willi Schönauer and Hartmut Häfner. Supercomputers: Where are the lost cycles ? *Supercomputing*, 1991.

[14] J. J. Dongarra. Performance of various computers using standard linear equations software in a fortran environment. Technical Report CS-89-85, Univeristy of Tennesse, 1993.

[15] G. Delic. Performance analysis of a 24 code sample on Cray X/Y-MP at the Ohio Supercomputer Center. In *Proceedings of the 5th SIAM Conference on Parallel Processing for Scientific Applications*, pages 530–535, 1991.

[16] D. V. Pryor and P. J. Burns. Vectorized Monte Carlo Molecular Aerodynamics Simulation of the Reyleigh Problem. In *Proceedings of Supercomputing'88*, pages 384–391, Orlando, Florida, November 1988. IEEE Computer Society Press.

[17] Roger Espasa and Mateo Valero. Decoupled vector architectures. In *HPCA-2*, pages 281–290. IEEE Computer Society Press, Feb 1996.

[18] Roger Espasa, Mateo Valero, and James E. Smith. Out-of-order Vector Architectures. In *MICRO-30*, pages 160–170. IEEE Press, December 1997.

[19] Toni Juan, Tomas Lang, and Juan J. Navarro. The difference-bit cache. In *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, pages 114–120, Philadelphia, Pennsylvania, May 22–24, 1996.

[20] Roger Espasa, Mateo Valero, and James E. Smith. Out-of-order Vector Architectures. Technical Report UPC-DAC-1996-52, Univ. Politècnica de Catalunya–Barcelona, November 1996.

[21] Roger Espasa and Mateo Valero. Multithreaded vector architectures. In *HPCA-3*, pages 237–249. IEEE Computer Society Press, Feb 1997.

[22] Subbarao Palacharla, Norman P. Jouppi, and J. E. Smith. Complexity-effective superscalar processors. In *Proceedings of the 24rd Annual International Symposium on Computer Architecture*, pages 206–218, Denver, Colorado, June 2–4, 1997. ACM SIGARCH and IEEE Computer Society TCCA.

[23] Roger Espasa and Mateo Valero. Exploiting Instruction- and Data- Level Parallelism. *IEEE Micro*, pages 20–27, September/October 1997.

[24] William J. Dally. Tomorrow's computing engines (Keynote Speech). In *HPCA-4*, February 1998.

[25] Alex Peleg and Uri Weiser. MMX Technology Extension to the Intel Architecture. *IEEE Micro*, pages 42–50, August 1996.

[26] Marc Tremblay, J. Michael O'Connor, Venkatesh Narayanan, and Liang He. VIS Speeds New Media Processing. *IEEE Micro*, pages 10–20, August 1996.

[27] Luis Villa, Roger Espasa, and Mateo Valero. A Performance Study of Out-of-Order Vector Architectures and Short Registers. In *ICS*. ACM Press, July 1997.