Vector Architectures

Lecture #11: Thursday, 9 May 2002
Lecturer: Prof. Bill Dally
Scribe: James Bonanno
Reviewer: Mattan Erez

Logistics :

- Project Update: presentations next Tuesday (May 14)
- Read all project proposals
- People not in project groups should form them soon

The topic of this lecture was vector architectures. After discussing a definition of vector architectures, similarities and difference between the stream architecture (as idealized in Imagine) were discussed, as were the corresponding advantages and disadvantages. The discussion concluded by focusing on reasons for the apparent fall in popularity of vector machines compared to the "Killer Micros", and how this relates to stream processor architecture. The two papers read in preparation for this discussion were:

- *Roger Espasa, Mateo Valero, and James E. Smith, "Vector Architectures: Past, Present, and Future," ICS '98, Proceedings of the 1998 International Conference on Supercomputing, July 13-17, 1998, Melbourne, Australia. ACM, 1998.*
- *John Wawrzynek, Krste Asanovic, and Brian Kinksbury, "Spert-II: A vector Microprocessor System," IEEE Computer, March 1996.*

1 What is a Vector Processor?

It was suggested that a key aspect of vector architecture is the single-instruction-multiple-data (SIMD) execution model. SIMD support results from the type of data supported by the instruction set, and how instructions operate on that data.

In a traditional scalar processor, the basic data type is an n-bit word. The architecture often exposes a register file of words, and the instruction set is composed of instructions that operate on individual words.

In a vector architecture, there is support of a vector datatype, where a vector is a collection of VL n-bit words (VL is the vector length). There may also be a vector register file, which was a key innovation of the Cray architecture. Previously, vector machines operated on vectors stored in main memory. Figures 1 and 2 illustrate the difference between vector and scalar data types, and the operations that can be performed on them.

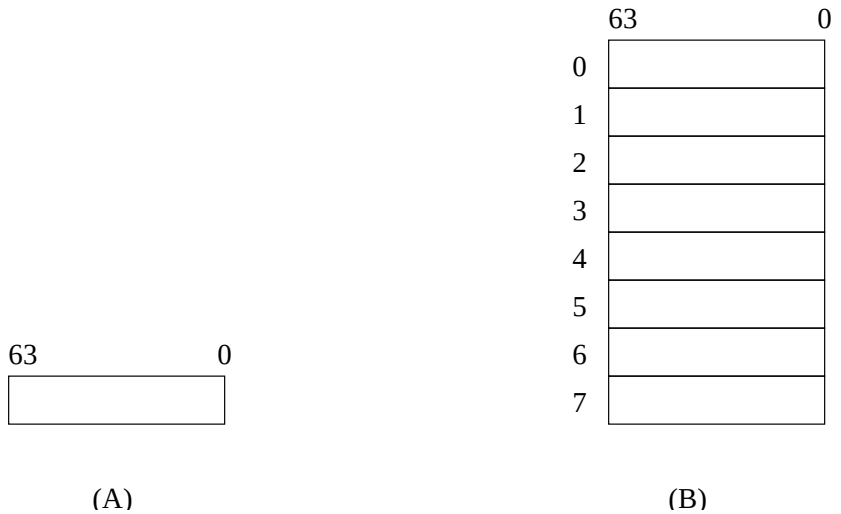

Figure 1: (A): a 64-bit word, and (B): a vector of 8 64-bit words

2 Comparing Vector and Stream Architectures

The following table lists similarities and differences between vector and stream architectures that were discussed in class. These aspects are described below in more detail.

Similarities	Differences
SIMD	cross-pipe communication
vector Load/Store	record vs. operation order
pipes (DLP support)	local register files (LRF) and microcode

2.1 Similarities

2.1.1 SIMD

Vector architectures exhibit SIMD behaviour by having vector operations that are applied to all elements in a vector. Likewise, in stream architectures, the same sequence computation (microprogram) is performed on every stream element. SIMD has three main advantages:

1. Requires lower instruction bandwidth.
2. Allows for cheap synchronization.
3. Allows easier addressing.

2.1.2 Vector Load/Store

Vector load/store instructions provide the ability to do strided and scatter/gather memory accesses, which take data elements distributed throughout memory and pack them

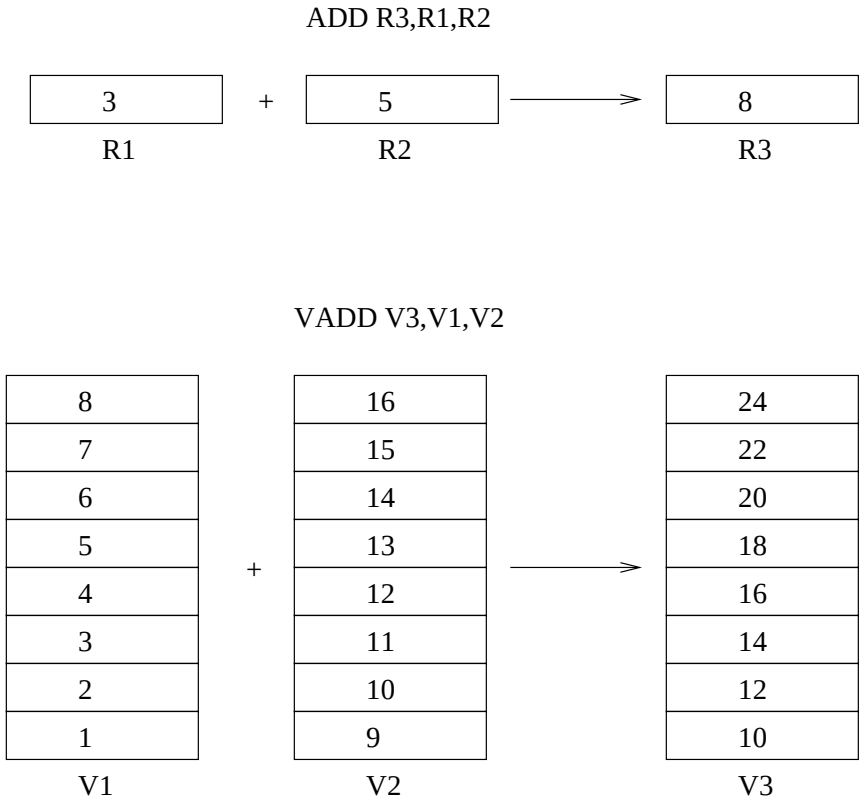

Figure 2: Difference between scalar and vector add instructions

into sequential vectors/streams placed in vector/stream registers. This promotes data locality. It results in less data pollution, since only useful data is loaded from the memory system. It provides latency tolerance because there can be many simultaneous outstanding memory accesses. Vector instructions such as VLD and VST provide this capability. As illustrated in figure 3 VLD V0,(V1) loads memory elements into V0 specified by the memory addresses (indices) stored in V1. VST (V1), V0 works in a similar manner. VST (S1), 4, V0, stores the contents of V0 starting at the address contained in S1 with stride of 4.

2.1.3 Pipes

Having multiple pipes allows easy exploitation of DLP. A vector architecture specifies that the same operation is performed on every element in a vector. It does not specify how this is implemented in the microarchitecture. For example, the T0 processor has 8 pipes, thereby allowing a vector operation to be performed in parallel on 8 elements of the vector. Figure 4 shows how the T0 processor structures its vectors. In that architecture, the vector length is 32. Having 8 pipes therefore results in an arithmetic operation latency of 4 cycles.

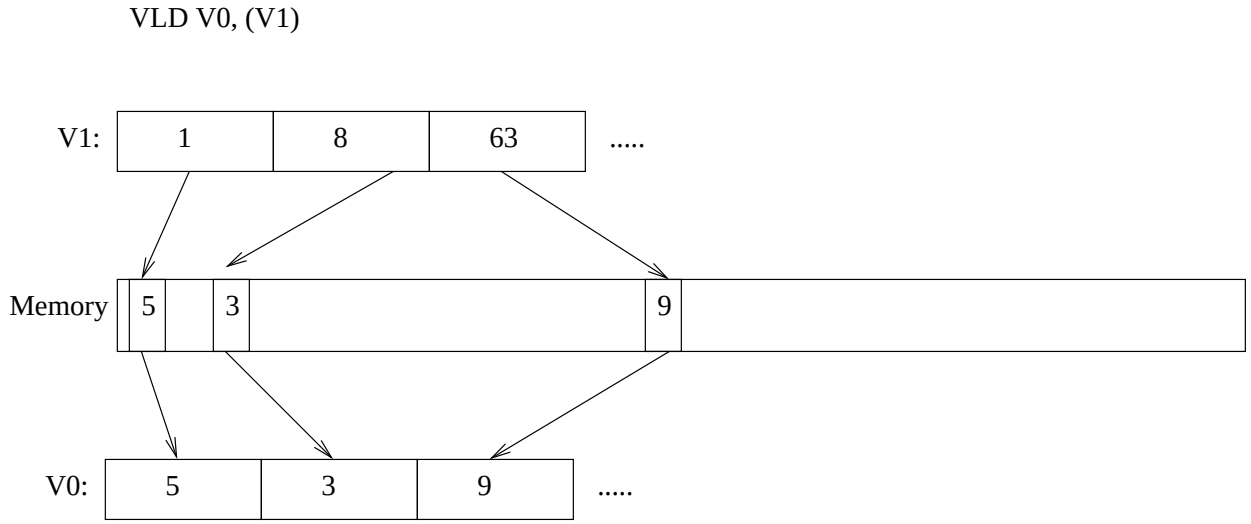

Figure 3: Indexed Vector Load Instruction

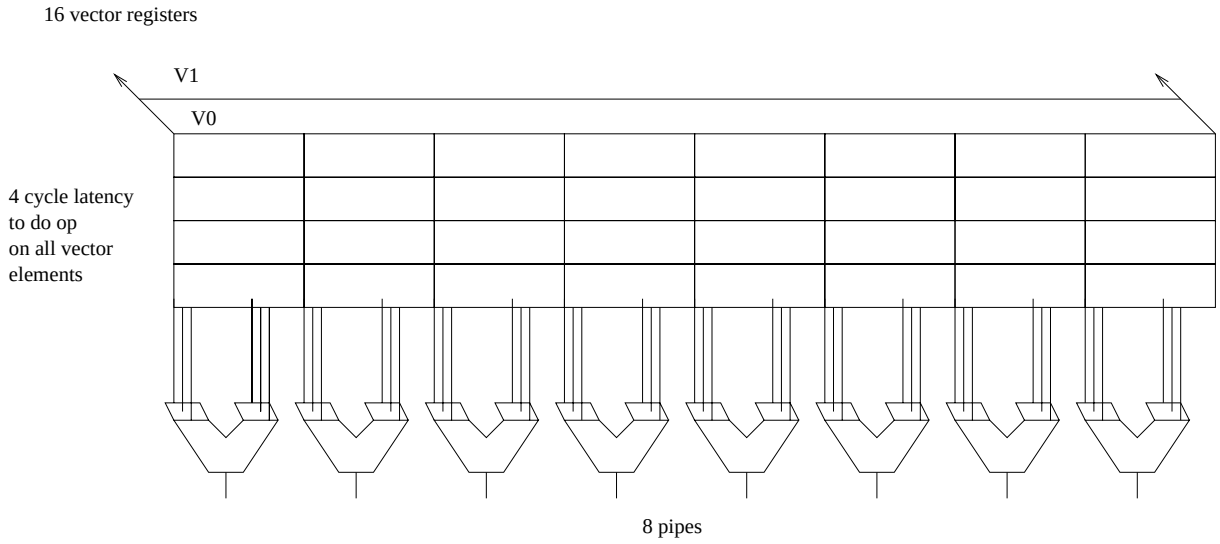

Figure 4: T0 vector organization

2.2 Differences

2.2.1 Cross-Pipe Communication

A stream architecture may allow cross-pipe communication (In Imagine, this is inter-cluster communication), while such communication in vector processors is only possible by rearranging the data ordering with load/store instructions to/from the memory system.

2.2.2 Record vs. operation order, LRF and microcode

Vector processors perform single operations on entire vectors, while stream processors like Imagine execute entire micro-programs on each data element of a stream. The effect of this difference is that in vector architectures, the intermediate vectors produced by each instruction are stored in the vector register file, while in a stream processor, intermediate values are consumed locally. They are stored in local register files, and are accessed at a higher bandwidth. A possible disadvantage is the apparent increase in code complexity in such a stream processor.

3 Memory Bandwidth

It was suggested that the true reason why vector supercomputers are fast is because of the memory bandwidth that they provide. They do so by supporting banked memory and allowing several simultaneous outstanding memory requests. Such architectures do not rely on cache techniques to achieve reasonable performance. They therefore often outperform cache-reliant architectures in applications that make "random" memory accesses.

4 Killer Micros

Key reasons why today's microprocessors have displaced widespread use of vector architectures include:

1. CMOS became a universal technology, while many vector supercomputers continued to use Bipolar where the yield was poor on dense chips.

2. A processor could be implemented on a single chip in CMOS, while older vector machines involved multi-chip implementations.

3. Several applications have very good performance on non-vector machines.

4. Caches exploited locality to improve memory system performance without performing the expensive optimizations often found in vector machines.

Figure 5 shows the relative performance growth curves of CMOS and Bipolar technology, and also memory performance.

4.1 Relevance to Stream Processors

It seems prudent to use CMOS technology for the foreseeable future, to optimize for high bandwidth, and to provide latency tolerance and SIMD execution. Finally the question was posed as to why there have been so few architectural innovations to vector processors?

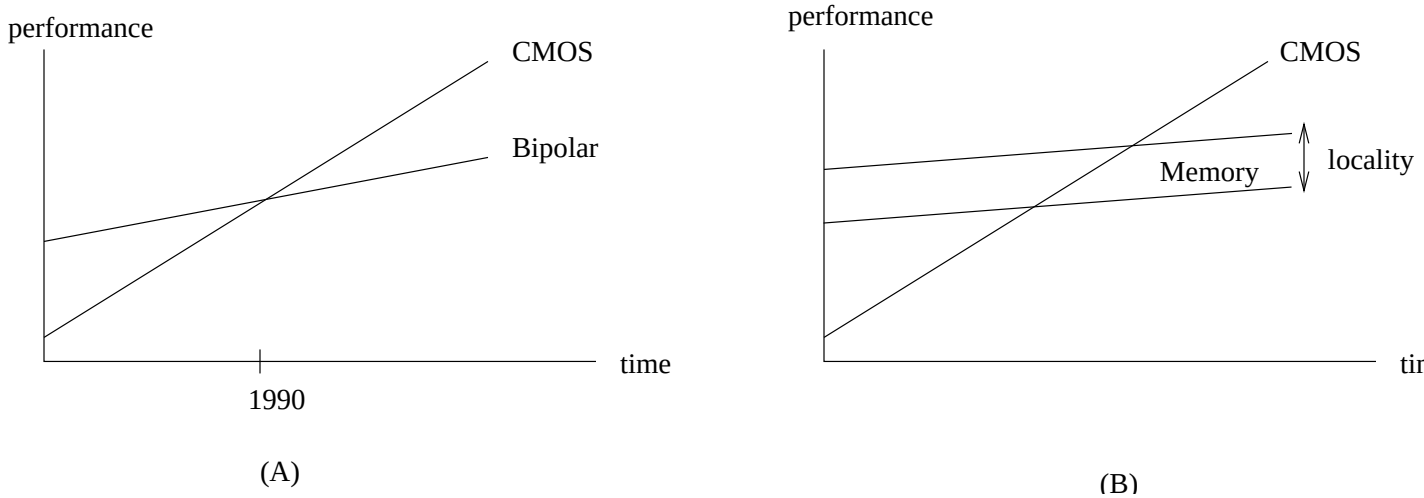

Figure 5: Performance Growth of (A) CMOS vs Bipolar, (B) Memory Performance

Perhaps this is due to the fact that key innovations are often made during the earlier generations of an architecture.