Vector Processors

Topics:

- Introduction
- Some Vector Processors
- New Terms
- Basic Vector Processor Architecture
- Problems: Vector length and Stride
- The Effect of cache design into vector computers
- References

Introduction

Vector processors are special purpose computers that match a range of (scientific) computing tasks. These tasks usually consist of large active data sets, often poor locality, and long run times. In addition, vector processors provide vector instructions.

These instructions operate in a pipeline (sequentially on all elements of vector registers), and in current machines. Some properties of vector instructions are

- The computation of each result is independent of the computation of previous results, allowing a very deep pipeline without any data hazards.
- A single vector instruction specifies a tremendous amount of work – it is the same as executing an entire loop. Thus, the instruction bandwidth requirement is reduced.
- Vector instructions that access memory have a known access pattern. If the vector elements are all adjacent, then fetching the vector from a set of heavily interleaved memory banks works very well. Because a single access is initiated for the entire vector rather than to a single word, the high latency of initiating a main memory access versus accessing a cache is amortized. Thus, the cost of the latency to main memory is seen only once for the entire vector, rather than once for each word of the vector.
- Control hazards are non-existent because an entire loop is replaced by a vector instruction whose behavior is predetermined.

Typical vector operations include (integer and floating point:

- Add two vectors to produce a third.
- Subtract two vectors to produce a third
- Multiply two vectors to produce a third

- Divide two vectors to produce a third
- Load a vector from memory
- Store a vector to memory.

These instructions could be augmented to do typical array operations:

- Inner product of two vectors (multiply and accumulate sums)
- Outer product of two vectors (produce an array from vectors)
- Product of (small) arrays (this would match the programming language APL which uses vectors and
- Arrays as primitive data elements)

Some Vector Processors

PROCESSOR	YEAR	CLOCK(MHZ)	REGISTER ELEMENT	FUCTIONAL (PER REGISTER)	UNITS
CRAY-1	1976	80	8	64	6
CRAY-XMP	1983	120	8	64	8
CRAY-YMP	1988	166	8	64	8
NEC SX/2	1984	160	8+8192	256 variable	16
CRAY C-90	1991	240	8	128	8
NEC SX/4	1995	400	8+8192	256 variable	16
CRAY J-90	1995	100	8	64	8
CRAY T-90	1996	500	8	128	8
NEC SX/5	1999				

New Terms:

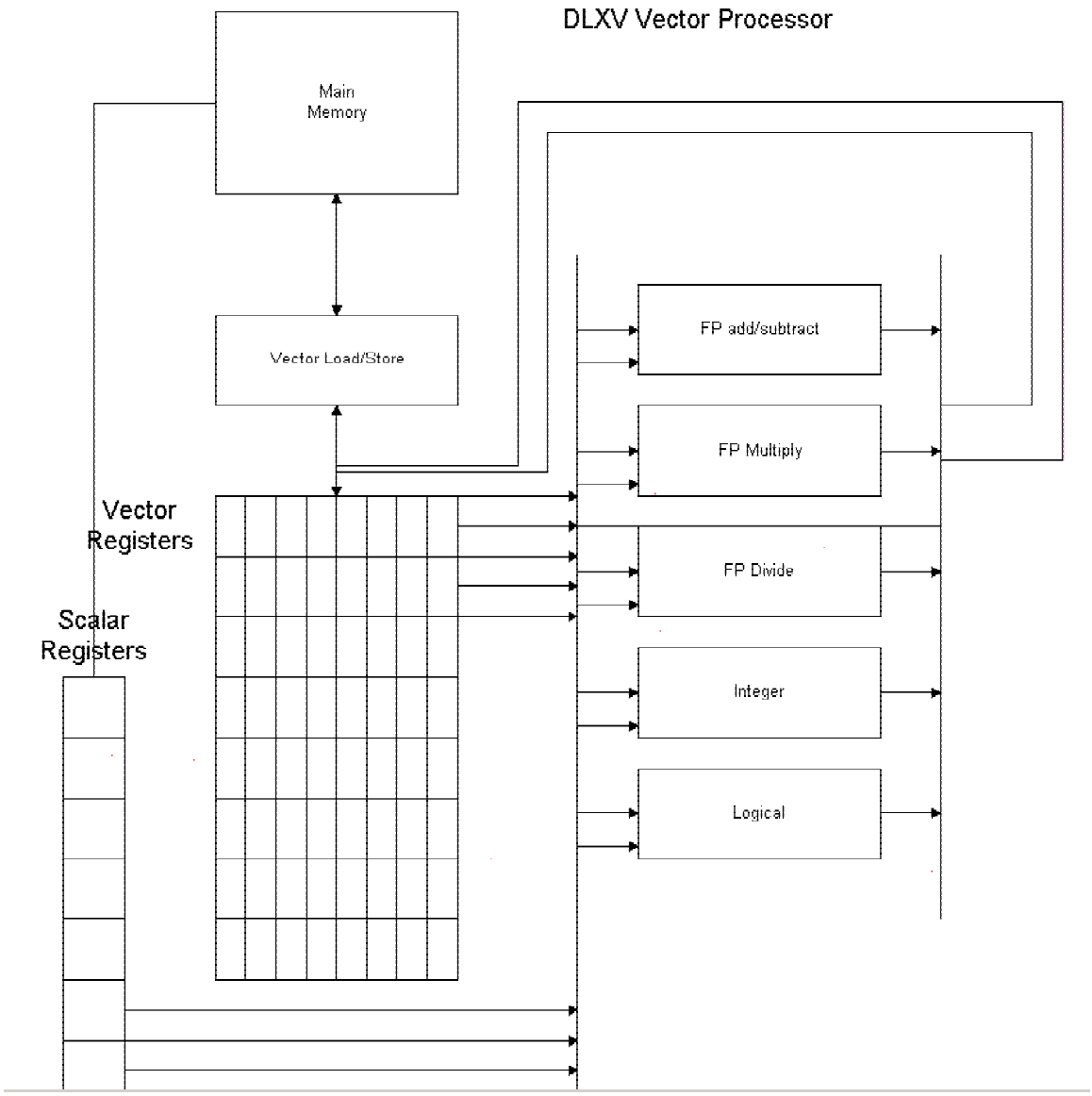
- *initiation rate* is the rate of consuming operands and producing new results. (ie. one per clock cycle for individual instruction, more for parallel operations)

- *convoy* is the set of vector instructions that could potentially begin execution together in one clock period. A convoy must complete before new instructions can begin.

- *chime* is a timing measure for the time for a vector sequence. A vector sequence of m convoys (executes in m chimes), with a vector length of n elements executes in roughly m x n clock cycles. A chime ignores the startup overhead for a vector operation.

- *vector start-up time* is the overhead to start execution. It is related to the pipeline depth, and is due to the time to clear out existing vector operations from the unit.

Basic Vector Architecture



Problems: Vector length and stride

Vector lengths do not often correspond to the length of the vector registers - except by plan:

- For shorter vectors, we can use a vector length register applied to each vector operation
- For longer vectors, we can split the long vector into multiple vectors (of equal, or of maximum plus smaller lengths). The process is called *strip-mining*. The strip-mined loop consists of a sequence of convoys.

Stride is the distance separating elements in memory that will be adjacent in a vector register. The unit stride is easiest to handle.

- Non-unit strides can cause major problems for the memory system, which is based on unit stride (ie. all the elements are one after another
- In different interleaved memory banks). Caches deal with unit stride, and behave badly for non-unit stride.
- To account for non-unit stride, most systems have a stride register that the memory system can use for loading elements of a vector register. However, the memory interleaving may not support rapid loading. In 1995 the vector computers had from 64 to 1024 banks of memory to overcome some of these problems - and to allow fast vector memory load/stores.

The Effect of cache design into vector computers

While cache memories have been successfully used in general purpose computers to boost system performance, their effectiveness for vector processing has not been established. Most of existing supercomputer vector processors typically do not have a cache memories because of perceived from these observations:

- Numerical programs generally have data sets that are too large for the current cache sizes. Sweep accesses of a large vector may result in complete reloading of the cache before the processor reuses them.
- Address sequentially which has been an important assumption in the conventional caches may not be as good in vectorized numerical algorithms that usually access data with certain stride which is the difference between addresses associated with consecutive vector elements.
- Register files and highly interleaved memories have been commonly used to achieve high memory bandwidth required by vector processing.

It is on clear whether cache memories can significantly improve the performance of such systems.

Although cache memories have potential for improving the performance of future vector processors, there are practical reasons why such vector caches have not yet been satisfactorily efficient. A single miss in the vector cache results in a number of processor

stall cycles equal to the entire memory access time, while the memory accesses of a vector processor without cache are fully pipelined. In order to benefit from a vector cache, the miss ratio must be kept extremely small. In general, cache misses can be classified into these categories (Refer to part I for more details from website):

- Compulsory miss
- Capacity miss
- Conflict miss

The compulsory misses are the misses in the initial loading of data, which can be properly pipelined in a vector computer. The capacity misses are due to the size limitation of a cache to hold data between references. If application algorithms are properly blocked as mentioned above, the capacity misses can be attributed to the compulsory misses for the initial loading of each block of data provided that the block size is less than cache size. The last category, conflict misses, plays a key role in the vector processing environment. Conflicts can occur when two or more elements of the same vector are mapped to the same cache line or elements from two different vectors compete for the same cache line. Since conflict misses that significantly degrade vector cache performance have a lot to do with vector access stride, one may wish to adjust the size of an application problem to make a good access stride for a given machine. However, not only does this approach give a programmer a burden of knowing architecture details of a machine but also infeasible for many applications.

Proposals such as prime-mapped cache schemes have been proposed and studied. The new cache organization minimizes cache misses caused by cache line interferences that have been shown to be critical in numerical applications. The cache lookup time of the new mapping scheme keeps the same as conventional caches. Generation of cache addresses for accessing the prime-mapped cache can be done in parallel with normal address calculations. This address generation takes shorter time than the normal address calculation due to the special properties of the Mersenne prime. Therefore, the new mapping scheme does not result in any performance penalty as far as the cache access time is concerned. With this new mapping scheme, the cache memory can provide significant performance improvement, which will become larger as the speed gap between processor and memory increases.

References:

- J.L. Hennessy and D.A. Patterson, *Computer Architecture, A Quantitative Approach*. Morgan Kaufmann, 1990.
- http://csep1.phy.ornl.gov/ca/node24.html
- http://www.comp.nus.edu.sg/~johnm/cs3220/l21.htm
- http://penta-performance.com/sager/vector/Default_vector2.htm
- http://www-icpc.doc.ic.ac.uk/facilities/vx/arch.html