Vector Processors

Pratyusa Manadhata, Vyas Sekar
{pratyus,vyass}@cs.cmu.edu

1 Introduction

A Vector processor is a processor that can operate on an entire vector in one instruction. The operand to the instructions are complete vectors instead of one element.

Vector processors reduce the fetch and decode bandwidth as the number of instructions fetched are less. They also exploit data parallelism in large scientific and multimedia applications. Based on how the operands are fetched, vector processors can be divided into two categories - in memory-memory architecture operands are directly streamed to the functional units from the memory and results are written back to memory as the vector operation proceeds. In vector-register architecture, operands are read into vector registers from which they are fed to the functional units and results of operations are written to vector registers.

Many performance optimization schemes are used in vector processors. Memory banks are used to reduce load/store latency. Strip mining is used to generate code so that vector operation is possible for vector operands whose size is less than or greater than the size of vector registers. Vector chaining - the equivalent of forwarding in vector processors - is used in case of data dependency among vector instructions. Special scatter and gather instructions are provided ed to efficiently operate on sparse matrices.

Instruction set has been designed with the property that all vector arithmetic instructions only allow element N of one vector register to take part in operations with element N from other vector registers. This dramatically simplifies the construction of a highly parallel vector unit, which can be structured as multiple parallel lanes. As with a traffic highway, we can increase the peak throughput of a vector unit by adding more lanes. Adding multiple lanes is a popular technique to improve vector performance as it requires little increase in control complexity and does not require changes to existing machine code.

2 Why are they expensive

The reason behind the declining popularity of vector processors are their cost as compared to multiprocessors and superscalar processors. The reasons behind high cost of vector processors are

- Vector processors do not use commodity parts. Since they sell very few copies, design cost dominates overall cost.
- Vector processors need high speed on-chip memory which are expensive.
- It is difficult to package the processors with such high speed. In the past, vector manufactures have employed expensive designs for this.
- There have been few architectural innovations compared to superscalar processors to improve performance keeping the cost low.

3 Semantic advantage

Vector processing has the following semantic advantages.

- Programs size is small as it requires less number of instructions. Vector instructions also hide many branches by executing a loop in one instruction.
- Vector memory access has no wastage like cache access. Every data item requested by the processor is actually used.
- Once a vector instruction starts operating, only the functional unit(FU) and the register buses feeding it need to be powered. Fetch unit, decode unit, ROB etc can be powered off. This reduces the power usage.

4 CODE

CODE (Clustered Organization for Decoupled Execution) is a proposed vector architecture which claims to overcome the following limitations of conventional vector processors.

- Complexity of central vector register files(VRF) - In a processor with N vector functional units(VFU), the register file needs approximately 3N access ports. VRF area, power consumption and latency are proportional to O(N*N), O(log N) and O(N) respectively.
- Difficult to implement precise implementation - In order to implement in-order commit, a large ROB is needed with at least one vector register per VFU. In order to support virtual memory, large TLB is needed so that TLB has enough entries to translate all virtual addresses generated by a vector instruction.
- Vector processors need expensive on-chip memory for low latency.

Vector registres are organised in the form of clusters in CODE architecture. Each cluster consists of 4-8 registers and one VFU (arithmetic, load/store or floating point). Number of access ports needed by each cluster is 5 and is independent of number of clusters. Area, power and latency in each cluster is constant. As we add more clusters, it increases both number of VFUs and number of registers. So area, power and latency per register remains constant. The clusters use a separate communication network for transferring vector registers. This allows separate design decision for the communication network.

Issue logic selects the appropriate cluster to execute an instruction and generates inter-cluster transfers of operands. It tracks the mapping of architectural registers to cluster registers using a renaming table. Various possible schemes for cluster selection are - random, minimizing the number of transfers and load balancing among clusters.

CODE supports precise exception using a history buffer. On issuing an instruction, unallocated registers are assigned for its destination and any source registers that needs to be moved from other clusters. The physical registers with old values are not released until the exception behavior is known. The history buffer keeps track of changes to renaming table. If the instruction at the head of buffer doesn't cause an exception, the registers with old values are released. In case the instruction caused an exception, the history buffer is scanned to restore the old mapping in renaming table. Since the history buffer stores only mapping of registers and not the actual value, it's size is small.

In order to reduce the size of TLB, CODE proposes an ISA level change.

CODE allows partial completion of an instruction in case of an exception. If an exception is caused between elements 10 and 15, the result of first 9 elements are committed. When exception is resumed, the instruction restarts from 10th element. With this semantic, a TLB of size one will work, higher size of TLB is needed for performance reasons.

Instructions in different clusters can execute in decoupled manner. Need of synchronization arises when one of the instruction queues is full or inter-cluster transfer of registers is needed. CODE can hide communication latency by forcing the output interface to look ahead into the instruction queue and start executing register move instructions. It also simplifies chaining - output of one instruction is chained to the network and dependent instruction's input is chained to the network. In that case, the dependent instruction gets the output elements as soon as they arrive. CODE is also compatible with multi-lane architecture, i.e. each lane can have multiple clusters.

5 Conclusion

Vector supercomputers are not viable due to cost reason, but vector instruction set architecture is still useful. Vector supercomputers are adapting commodity technology like SMT to improve their price-performance. Superscalar microprocessor designs have begun to absorb some of the techniques made popular in earlier vector computer systems (Ex - Intel MMX extension). Vector processors are useful for embedded and multimedia applications which require low power, small code size and high performance.

References

[1] Roger Espasa, Mateo Valero, James E. Smith, " Vector Architectures: Past, Present and Future", in Supercomputing, 1998.

[2] C. Kozyrakis, D. Patterson, " Vector vs. Superscalar and VLIW Architectures for Embedded Multimedia Benchmarks", in MICRO, 2002.

[3] C. Kozyrakis, D. Patterson, " Overcoming the Limitations of Conventional Vector Processors", in ISCA, 2003.

[4] Hennessy/Patterson Appendix G: Vector Processing Appendix G