IJCSC International Journal of Computer Science & Communication Vol. 1, No. 2, July-December 2010, pp. 395-397

Recent Trends in Vector Architecture: Survey

Ritu Baniwal & Kumar Sambhav Pandey

Computer Science and Engineering Department, NIT, Hamirpur (H.P.)
Email: baniwalritu@gmail.com

ABSTRACT

A Vector processor is a processor that operates on an entire vector with a single instruction. Vector processor fetches less number of instructions which in turn reduces the fetch and decode bandwidth. Vector processors are more efficient in terms of power usage. In this paper different optimization techniques which improve the performance of vector processor have been discussed.

Keywords: Vector Processor

1. INTRODUCTION

Vector processing increases the computer performance. A vector is a set of numbers like an array. Scalar processors operate only on one element at a time. These types of computers are single instruction/single data (SISD). A vector processor is a computer that operates on an entire vector with a single vector instruction. These types of computers are known as single-instruction/ multiple-data (SIMD) computers because a single vector instruction can process a set of data.

Many performance optimization schemes are used in vector processors [6]. 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 [6].

2. METHODS TO IMPROVE EFFICIENCY

Some of the techniques available to increase the efficiency of vector instruction include overlapping and chaining.

2.1. Instruction Overlap

Overlapping instructions is combining the two or more instructions to overlap their execution. A vector processor can perform different operations on different operands simultaneously if it has independent functional units. Overlapping improves the performance.

2.2. Chaining

Chaining is a special form of instruction overlap. It is possible with multiple functional units. Chaining is passing the result of one operation in one functional unit to another functional unit. For example, an add instruction followed by a store instruction can combine so each element of the vector is stored as soon as the result is obtained. The processor does not have to wait for the add instruction to finish before storing the data. Chaining is similar to data forwarding in scalar processor.

3. IMPROVEMENT METHODS

3.1. CODE

Vector processors have three limitations: First, centralized vector register file limits the number of functional units because of its size and complexity. Second, precise exceptions for vector instructions are difficult to implement [1]. Third, vector processors require an expensive on-chip memory system that supports high bandwidth at low access latency [1]. CODE (Clustered Organization for Decoupled Execution) architecture addresses these limitations. It is designed around a clustered vector register file that separates the task of staging the operands for a single functional unit from the task of communicating data among functional units [1]. It uses a separate network to transfer operand across functional units. CODE scales efficiently to 8 functional units.

Communication Network

CODE requires a separate communication network for transferring vector registers between clusters. If the source operand is not available in the cluster, it is moved across clusters. Communication takes place by executing a move-to instruction in the output interface of the sending cluster and a move-from instruction in the input interface of the receiving cluster respectively [1]. All move

instructions are generated by hardware in the vector issue logic.

Vector Issue Logic

There may be more than 32 vector registers across all clusters. Each vector functional unit (VFU) can directly access only a subset of the registers. The issue logic selects the appropriate cluster to execute each instruction.

3.2. Decoupled Vector Architecture

There are two ways of dynamically scheduling: First, decoupled architectures, in which there is limited dynamic scheduling. In this, address generation from data computation tasks are separated. Second, out-of-order vector architecture with vector register renaming.

In decoupled vector architecture, the vector unit is split into address generation and data computation components. All vector arithmetic instructions are sent in-order to the computation instruction queue [14]. When all the operands are ready, they are dispatched to an arithmetic functional unit and the execution on data starts.

In decoupled vector architecture, the instruction stream is split into three different streams [2]: One has the vector computation instructions only which is executed by the vector processor (VP). The second contains all the memory accessing instructions which are handled by the address processor (AP). The third one has the scalar computation instructions executed which is executed by the scalar processor (SP). A fourth processor called the fetch processor (FP) controls all these three processors. The FP is responsible for fetching all instructions and distributing them among the AP, SP and VP.

The advantage of a decoupled pipelined architecture is that vector arithmetic instructions do not block the issue stage while waiting for the vector memory operands. The vector arithmetic instruction is sent to an instruction queue, freeing the issue stage to run ahead to find more vector memory instructions later in the instruction stream; these can begin address generation before earlier vector arithmetic instructions have begun execution [14].

3.3. Out-Of-Order Vector Architecture

Register renaming and out-of-order issue in a vector processor improves the performance. The performance can be improved by introducing extra physical registers for renaming. Renaming enables the precise exceptions to be easily implemented due to which virtual memory can be easily introduced [3].

3.4. Software Restart Markers to Implement Virtual Memory in a Vector Processor

Traditional vector architectures lack to support virtual memory because it is difficult to implement precise exceptions. A new exception handling model has been proposed for vector architectures which is based on software restart markers [4]. The program is divided into different regions of code. The processor can commit instruction results within a region in any order. The machine jumps immediately to the exception handler whenever an exception occurs and kills ongoing instructions [4]. To restart execution, the operating system has just to begin execution at the start of the region. This approach avoids the area and energy overhead to buffer uncommitted vector unit state [4].

3.5. Multithreaded Vector Architecture

Multithreading techniques are used to increase the throughput and maximize resource utilization in a vector processor. In multithreaded architecture, there are several copies of all three set of registers (A, S and V**)** where A, S are scalar registers and V is vector register [9]. These are needed to support multiple hardware contexts (up to a maximum of 4 contexts). The fetch and decode units are time multiplexed between the N contexts in the machine [9]. The decode unit considers only one thread in each cycle. In this scheme only 1 instruction can be dispatched per cycle. The instruction from a particular thread is sent to the functional unit after it is dispatched. After this, the fetch unit starts fetching the consecutive instructions from that thread. If the instruction cannot be dispatched, then this decoding cycle has been lost and the switch logic will select some other thread to attempt decoding in the following cycle [9].

In multithreaded architecture, there is no out-of-order or simultaneous issue. Only one instruction is fetched and sent to the functional units per cycle. The instructions execute in-order within a single thread. The scheduling is done among different threads. Each thread is allowed to be run until it blocks when some data dependency or some resource conflict occurs. Otherwise some other thread is chosen. If numbers of threads exist, the lowest numbered thread is chosen. The reason to allow a thread to proceed as long as it does not block **(**instead of switching threads every cycle) is to favor the amount of chaining happening between vector instructions [9].

3.6. Simultaneous Multithreaded Architecture

In this architecture, the DLP paradigm is merged with the ILP paradigm: out-of-order execution, register renaming and multithreading. Multithreading and out-of-order execution collectively yield simultaneous

multithreaded vector architecture. Multithreading is implemented only at the fetch, decode and commit stages.

The fetch unit selects one out of T threads and fetches several instructions (4 or 8) on behalf of it [11]. The instructions are renamed at the decode stage using a per-thread rename table. After being renamed, instructions go into several common execution queues. Independent threads use independent renames tables. So there are no false dependences or conflicts. Once renamed, each type of instruction goes to its corresponding queue. These instructions can be any mixture of scalar, memory or vector instructions. From the instruction queues any combination of instructions that are independent and that can belong or not to the same thread can be dispatched to execute onto the functional units [11]. Out-of-order execution happens between all types of instructions, scalar, memory and vector. On the other hand, inside vector instructions the individual operations are executed in-order. It means once a vector instruction gets a resource, it will use it until the whole operations are completed. It does not allow other vector instructions to share that resource. The joint use of ILP and DLP techniques on highly regular code (vectorizable code) yields a much larger performance [11] than the existing architectures.

3.7. Vector Lane Threading

Vector lane threading (VLT) allows short-vector or scalar threads to be run on idle vector lanes. VLT can exploit both data-level and thread-level parallelism. This achieves higher performance. Multithreading scheme has been proposed to increase the utilization of vector lanes. In this the vector lanes are partitioned across several threads. These can execute in parallel. The number of lanes assigned to each thread corresponds to its amount of data-level parallelism [5].

4. CONCLUSION

Vector processors fetch less number of instructions so reduces the fetch and decode bandwidth. They exploit data parallelism as well as instruction level parallelism. Different optimization techniques have been introduced to improve the performance of vector processors. Multithreading techniques increase the throughput and maximize resource utilization in a vector processor. Simultaneous multithreading has been introduced in vector processors to improve the performance to a large extent. It does so by merging the DLP paradigm with the ILP paradigm.

REFERENCES

[1] C. Kozyrakis and D. Patterson, "Overcoming the Limitations of Conventional Vector Processors", *In Proceedings of the 30th International Symposium on Computer Architecture, San Diego, California, June 2003, pp. 399–409.*

[2] R. Espasa and M. Valero, "Decoupled Vector Architecture", *In the Proceedings of the 2nd Intl. Symp. on High-Performance Computer Architecture*, Pages 281–90, San Jose, CA, Feb. 1996.

[3] R. Espasa, M. Valero, and J. Smith, "Out-of-order Vector Architectures", *In the Proceedings of the 30th Intl. Symp. on Microarchitecture*, Pages 160–70, Research Triangle Park, NC, Dec. 1997.

[4] Mark Hampton, Krste Asanovic, "Implementing Virtual Memory in a Vector Processor with Software Restart Markers", *20th ACM International Conference on Supercomputing (ICS06), Cairns, Australia, June 2006.*

[5] Suzanne Rivoire, Rebecca Schultz, Tomofumi Okuda, Christos Kozyrakis, "Vector Lane Threading", *Proceedings of the 2006 International Conference on Parallel Processing (ICPP'06).*

[6] Vector Processors, Pratyusa Manadhata, Vyas Sekar

[7] R. Espasa et al, "Vector Architectures: Past, Present and Future", In *the Proceedings of the 2th Intl. Conf. on Supercomputing*, Pages 425–432, Melbourne, Australia, July 1998.

[8] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann, San Francisco, CA, Fourth Edition, 2002.

[9] R. Espasa and M. Valero, "Multithreaded Vector Architectures", *In HPCA-3,* Pages 37-249. *IEEE Computer Society Press*, Feb 1997.

[10] F. Quintana, J. Corbal, R. Espasa, and M. Valero, "Adding a Vector Unit to a Superscalar Processor", *In ICS-13, June 1999.*

[11] Simultaneous Multithreaded Vector Architecture: Merging ILP and DLP for High Performance, Roger Espasa Mateo Valero, *In Proceedings of the Fourth International Conference on High-Performance Computing, 1997.*

[12] R. Espasa and M. Valero, "Decoupled Vector Architectures", *In HPCA-2,* Pages 281-290, IEEE Computer Society Press, Feb 1996.

[13] R. Krashinsky, C. Batten, M. Hampton, S. Gerding, B. Pharris, J. Casper, and K. Asanovic, "The Vector-Thread Architecture", *In Proceedings of the 31st International Symposium on Computer Architecture, Munich, Germany, June 2004, pp. 52–63.*

[14] K. Asanovic, "Vector Microprocessors", PhD Thesis, University of California at Berkeley, 1998.

[15] K. C. Yager, "The Mips RlOOOO Superscalar Microprocessor", *IEEE Mzcro,* Pages *28-40,* April 1996.