MULI-ISSUE MULTI-THREADED STREAM PROCESSOR

Somayeh Sardashti, Hamid Reza Ghasemi, Omid Fatemi

Multimedia Processing Laboratory, Department of Electrical and Computer Engineering, University of Tehran, 14399 Tehran, Iran

ABSTRACT

The MISP Processor is a programmable media processor which supports multi-issuing, multi-threading and stream processing techniques. MISP executes applications that have been mapped to the stream programming model. The stream model decomposes applications into a set of computation kernels that operate on data streams. This mapping exposes the inherent locality and parallelisms in media applications.

MISP exploits thread level (TLP), data level (DLP), sub-word (SP) and instruction level (ILP) parallelisms inherent in media applications. Full simulator of MISP has been implemented and several media workloads composed of EEMBC benchmarks have been applied [1]. Also we applied test programs on Imagine stream processor [2]. The simulation results show MISP gains IPC of more than 2.08 times and performance of more than 1.86 times over Imagine. The synthesis results show area overhead per thread addition in MISP is about 7% without changing clock frequency.

1. INTRODUCTION

Stream processors are a kind of digital signal processors (DSPs) targeted for high-performance embedded applications [2]. They try to exploit the locality and concurrency inherent in media applications using stream programming model. Stream programming organizes data as streams and all computations as kernels. Stream processors directly execute applications mapped to this programming model. They contain clusters of functional units and provide a memory hierarchy, supporting hundreds of arithmetic units. They exploit ILP and SP within a cluster and DLP across clusters [2]. However, they can not exploit the thread level parallelism existing among kernels in the stream programs.

In this paper, we present a multi-threaded multi-issue VLIW stream processor called MISP. In MISP stream processor, multithreading technique is used to extract TLP among kernels as long as ILP, DLP, and SP which are extracted because of stream processing architecture. Also MISP supports multiple issuing of stream instructions.

The organization of the paper is as follows. Section 2 reviews related work on multithreading, stream and vector processors. Section 3 explains stream processing. Our proposed stream processor is presented in section 4 followed by the simulation results in section 5. In section 6, the hardware implementation results are presented. Finally, section 7 concludes the paper.

2. RELATED WORK

Originally, simultaneous multithreading (SMT) technique has been introduced to permit multiple independent threads to issue instructions to a superscalar's functional units in a clock cycle [3]. Later, Simultaneous multithreading method has been applied to a VLIW processor [4] which results in significant improvement of operation throughput by allowing interleaved execution of operations from multiple threads.

In another work, multithreading support has been integrated into a VLIW processor [5] to hide run-time latency effects that cannot be determined by the compiler. SYMPHONY [6], a media processor, attempts to exploit ILP, DLP, and TLP using SMT method along with SIMD structure. PLASMA, a multimedia vector processor presented in [7], exploits data, instruction, and thread level parallelisms. This processor combines multithreading and VLIW methods. These media processors can not support programmability. The demand for flexibility in media processing motivates the use of programmable processors.

Recently, a programmable stream processor named Imagine [2] has been presented which exploits data, instruction, and sub-word parallelisms inherent in media applications. However, thread level parallelism is not possible to be exploited in this processor resulting in the performance degradation and low utilization of the functional units. Furthermore, Imagine does not support multiple issuing of the stream instructions and therefore the utilization of chip modules and the performance are degraded more [8].

In this paper, we proposed a new programmable media processor, MISP, which uses streaming architecture to exploit locality and concurrency of media applications efficiently. MISP applies multithreading and multiple issuing techniques to its streaming architecture to improve performance. The streaming architecture of MISP exploits principle ideas of Imagine architecture [2].

3. STREAM PROCESSING

Stream processing is a new trend in computer architecture which fills the gap between inflexible special-purpose media architectures and programmable architectures with low computational ability for media processing. Stream processors are designed for computationally intensive media applications which have high data parallelism, producer-consumer locality, and little global data reuse [2].

With stream processing, applications are expressed as stream programs. The stream programming model extracts the inherent locality and parallelism of media applications. Media applications can be modeled as stream programs. A stream program organizes data as streams and computing operations as a sequence of kernels. Streams contain a set of elements of the same type. Stream elements can be simple, such as a single number, or complex, such as the coordinates of a triangle in 3D space. Kernels are the computational units applied upon streams which access stream elements sequentially. Figure 1 shows an example of stream programming for stereo depth extracting [2] where arrows represent streams and circles represent kernels.

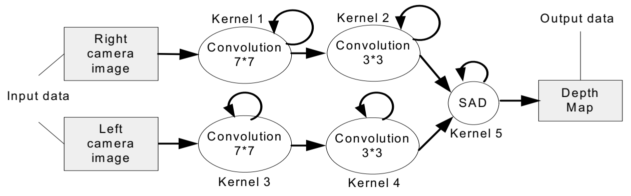

Figure 1 Stereo Depth Extraction

4. MISP PROCESSOR

MISP is a high performance programmable stream processor targeted at image and signal processing applications. MISP has a three level memory hierarchy, which causes the processor to exploit the locality and the concurrency in media applications efficiently. This memory hierarchy is partitioned into:

- the operands for arithmetic operations which are stored near the functional units in local register files (LRFs) in arithmetic clusters;
- the streams of data which are stored in a stream register file (SRF) and transfers data to and from LRFs;
- the global data which is stored in an off-chip memory which feeds data to SRF.

MISP processor is a multiple issue, multithreaded stream processor. MISP executes application mapped to streams and kernels. There are two types of instruction sets in stream processors in order to model stream programs: kernel-level and stream-level instruction sets. Stream-level instructions control the flow of data streams through the system. The main stream instructions used in MISP are:

- LOAD; transfers streams from off-chip memory to SRF.
- STORE; transfers streams from SRF to off-chip memory.
- LOAD MICROCODE; loads kernel instructions to the microcontroller microcode store.
- CLUST OP; executes a kernel in the arithmetic clusters.

Kernel-level instructions control the functional units and the register storage within the arithmetic clusters. Kernel instructions are packed as VLIW instructions. There are separate fields to control each of the cluster functional units, cluster stream buffers, and the microcontroller unit.

4.1. MISP Processor Architecture

MISP processor is designed to support stream processing as long as multi-threading and multiple issuing techniques. MISP processor extracts the existing TLP among computational kernels by supporting multiple simultaneous executing threads. Each kernel of a stream program forms a thread. Independent kernels and even sequential kernels running in a pipe manner can be scheduled as simultaneous threads in MISP. For example, in the stereo depth extraction (Figure 1), the kernel 1 is independent of the kernel 3 and the kernel 4. Also considering the program execution as a pipeline, the kernel 3 and 4 can be executed simultaneously. While the kernel 4 works on the data produced by the kernel 3, the kernel 3 produces the new data for kernel 4.

The MISP architecture is shown in Figure 2. MISP acts as a coprocessor for a host processor which controls it by accessing its control and status registers. MISP interacts with the host processor by handshaking between Host interface and Stream controller. In order to access off-chip memory, there is a memory system stream controller unit which can support up two memory instructions simultaneously. This memory system loads and stores stream elements sequentially between off-chip memory and SRF. In the following sections, MISP units are discussed in more details.

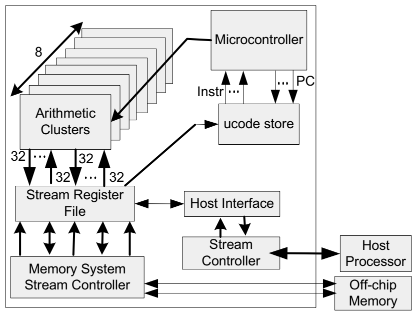

Figure 2 MISP Architecture

4.1.1. Microcontroller

The microcontroller fetches VLIW kernel instructions for multiple executing threads from a microcode store and sequences and issues a mixture of these instructions to all of the arithmetic clusters. The execution of the clusters occurs in a SIMD fashion so the same microcode instruction is issued to all the clusters simultaneously by the microcontroller.

In the microcontroller, Figure 3, there is a microcode loader unit per thread which handles the loading of kernel instructions from the SRF to the microcode storage. The microcode storage holds the microcode instructions which are to be issued by the control unit to the arithmetic clusters. The control unit is subdivided to the stream buffer controllers, the cluster controller, and the microcontroller microcode controllers. Each stream buffer controller is associated to a thread and controls its related cluster stream buffers according to the fields of VLIW instructions related to stream buffers. If there is an instruction to read or write to a stream buffer and that buffer is not ready to use, executing the related thread will be stopped.

Also there is a microcontroller microcode controller per thread which executes the fields of VLIW instructions which are related to the microcontroller. The cluster controller gets the cluster fields of the instructions from all threads, and sends a combination of the operations from active threads to the clusters.

Each thread has a priority number which is assigned statically by the programmer via the stream instructions. In order to merge operations from multiple threads, cluster controller starts with the operations of active threads with higher priorities. It assigns a needed functional unit to a thread if that unit has not been assigned to another thread with higher priority. So, it is possible that the operations of some threads not to be issued completely. In this case, the remainder of the instructions will be issued next cycles. After the issuing an instruction of a thread is completed, a new instruction will be fetched for that thread from microcode storage.

In order to protect the microcontroller to starve some of the threads, aging technique is used. If none of the needed functional units is assigned to a thread, its priority number will be incremented. So, there will be higher chance of issuing in the next cycles for that thread.

Using multithreading technique, the unused cluster functional units for an executing kernel will be used by another kernel. This results in the improvement of cluster utilization and so, total execution time of a stream program is decreased.

Also, there are one microcontroller register file (UCRF) and one microcontroller condition register file (UCONDRF) for each thread. These register files keep data that is needed during kernel execution.

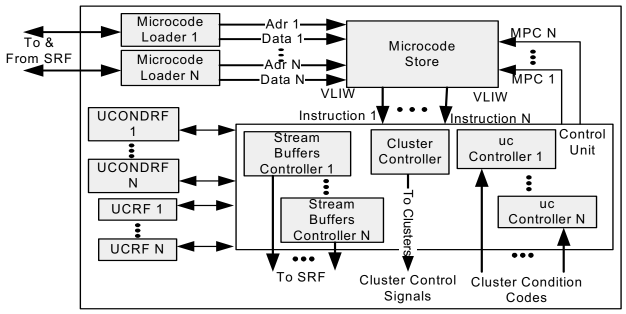

Figure 3 Microcontroller Architecture

4.1.2. Arithmetic clusters

There are 8 arithmetic clusters, Figure 4; each contains 6 functional units (3 adders, 2 multipliers, and a divide/square root module), one scratch pad unit per thread, which is a runtime addressable register file, and one communication unit per thread which handle inter-cluster communications of each thread. All of these units are fully pipelined.

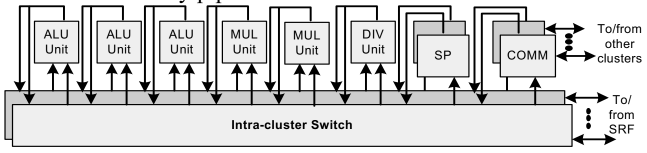

Figure 4 Arithmetic Clusters

In functional units, there are separate LRFs connected to their input lines per each executing thread to keep data during kernel execution. Also there is a separate output bus per thread, so a functional unit executing an operation from a thread, places the result on the thread associated output bus.

Data is exchanged between FUs via the intra-cluster switches. There is a switch per thread to transfer data for that thread. These switches are implemented as full crossbars where each FU broadcasts its result bus associated to a thread to the input of every LRF associated to that thread, in an arithmetic cluster.

4.1.3. Stream Register File

The stream register file (SRF), provides on-chip data storage for streams. SRF unit is partitioned into SRF storage, stream buffers, and a controller which controls transferring data between SRF storage and stream buffers. Producer-consumer locality of media application is exposed efficiently by SRF. Data is exchanged among kernels using SRF.

SRF storage consists of 128K bytes of memory. The stream buffers are employed to interface between SRF storage and SRF clients (i.e. arithmetic clusters, the memory unit, the microcontroller, and the host interface). Each stream is supported by a 64-word (each word is 4 bytes) stream buffer. There are eight stream buffers for communicating with the arithmetic clusters per each thread. Each of these stream buffers is able to transfer 8 words per cycle (1 word per cluster) in parallel to the clusters for a combined peak rate of 64 words per cycle. Also there is one buffer for transferring kernel instructions to the microcontroller per thread. There are two buffers which are used to transfer data between SRF and the memory.

4.1.4. Stream Controller

The stream controller handles the data flow between and control of all of the modules on the chip. It accomplishes this by controlling which stream instructions to issue and when they are executed.

The stream controller consists of the Scoreboard and Issue units. The Scoreboard keeps track of all operations waiting to execute and in flight. It also examines the dependencies each operation has on other operations and on hardware resources and sends ready instructions to the Issue unit. The Issue Unit is a state machine that will set all the correct control signals in order to start the execution of a given stream instruction.

In a conventional stream processor like Imagine, there is just one global control bus. So it is not possible to issue more than one command to the chip modules. This causes slow startup of the instructions and increases idle time of the modules. In MISP processor, there is a separate control bus for each chip module. This enables the stream controller to issue multiple stream instructions at every clock cycle. The Issue unit structure is a distributed state machine model that there is a separate controller for each stream instruction type. In this model, it takes only one or two clock cycles to issue a stream instruction.

5. SIMULATION RESULT

In order to evaluate the new design, we developed a cycle accurate simulator which models MISP processor. Using this simulator, we have evaluated MISP performance under several multimedia benchmarks- a subset of the EEMBC suite and four practical micro-benchmarks for different number of active threads. We compared MISP with Imagine stream processor using Imagine programming tool set. We used ISim cycle accurate simulator to evaluate Imagine [2].

Table 1 Simulated Benchmarks

Benchmark	**Comments**
RGB→ CMYK	Color-conversion
RGB→YIQ	Color-conversion
Gray Filter	3*3 Convolution
Autocorrelation	Series of dot-products
Sort	Bitonic Sort – Series of comparisons
Vector Addition	$C[i] = A[i] + B[i]$
Vector Multiplication	$C[i] = A[i] \times B[i]$
Multiply-Accumulate	$R = \sum_i A[i] \times B[i]$

Table 1 presents the simulated benchmarks. The first four kernels are taken from EEMBC [1]. Vector Addition and Vector Multiplication are commonly used micro-benchmarks characterized by low computational rate relative to memory requirements. Bitonic Sort benchmark evaluates the processor when there is much amount of inter-cluster communication.

Multiply-Accumulate benchmark is also a commonly used function in media applications.

We run different combinations of 2-thread, 4-thread, and 8-thread tests in order to show the improvements. Each combination includes various floating point, integer, memory bound and computational bound test cases. We considered that the simultaneous executing threads are independent.

In order to compare our results with a single-threaded stream processor, we applied our test programs to Imagine stream processor too. To demonstrate MISP improvements, we have measured two parameters, IPC (Instructions per Cycle) and processor performance (1/Total Executing Cycles) with different number of active threads. The overall results are shown in Figure 5. In this figure we showed the average improvements gained over Imagine which is a single-threaded stream processor. For two active threads, MISP improves IPC about 2.08 times, and total performance about 1.86 times. MISP gains higher improvements with 4 and 8 active threads.

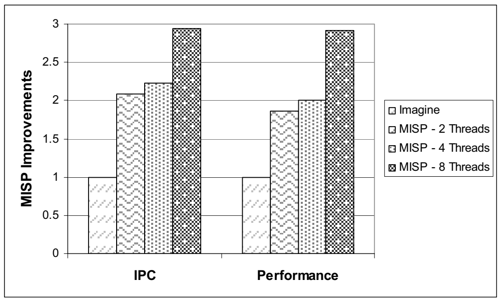

Figure 5 MISP Improvements

Figure 6 shows MISP IPC improvement for various 2-thread test combinations in comparison with Imagine in more details.

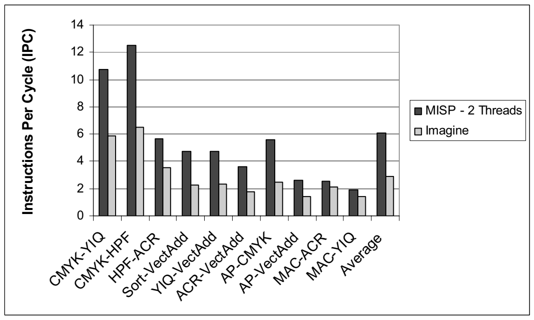

Figure 6 IPC Improvement for 2 Threads

6. HARDWARE IMPLEMENTAION

In this section, we briefly describe the overhead due to adding multi-threading and multi-issuing to a stream processor in terms of area and delay. First, we have implemented MISP without supporting multi-threading and multi-issuing techniques. In this manner, MISP acts as a single-threaded stream processor. Then, we have implemented MISP processor supporting 2 active threads and multi-issuing technique. For these two models, MISP is implemented and synthesized in a 0.25u ASIC (TSMC25) technology. The difference between the areas of these models shows how adding one thread and multi-issuing can affect area of the stream processor. Table 2 shows the result of this comparison in terms of area. The clock frequency of both single-threaded and double-threaded MISP are the same.

Table 2 Area Overhead of double-thread vs. single-threaded MISP

Units	Area Overhead (%)
SRF	1%
Memory Controller	0.1%
Ucontroller	3%
Clusters	49%
Stream Controller	37%
Host interface	0%
Total	**7%**

7. CONCLUSION

In this paper, we have proposed a new media processor named MISP. This processor is a stream processor which is featured by multithreading and multi issuing techniques.

Double-threaded MISP gains IPC of 2.08 times and performance of 1.86 times over Imagine processor. MISP gains higher improvements with 4 and 8 active threads. The area overhead per thread addition in MISP is about 7% while the clock frequency is constant.

8. REFERENCES

[1] EEMBC (EDN Embedded Microprocessor Benchmark Consortium) Benchmark Suite - www.eembc.org.

[2] Brucek Khailany, William J. Dally, Scott Rixner, Ujval J. Kapasi, Peter Mattson, Jinyung Namkoong, John D. Owens, Brian Towels, and Andrew Chang, "Imagine: media processing with streams," IEEE Micro, March/April 2001.

[3] Dean M. Tullsen, Susan J. Eggers, and Henry M. Levy, "Simultaneous Multithreading: maximizing on-chip parallelism," In Proceedings of the 22nd Annual International Symposium on Computer Architecture, Santa Margherita Ligure, Italy, June 1995.

[4] H. Pradeep Rao, S.K. Nandy, and M.N.V. Satya Kiran, "Simultaneous MultiStreaming for complexity-effective VLIW architecture," In Asia-Pacific Computer Systems Architecture Conference, Aizu-Wakamatsu, Japan, September 2003.

[5] Emre Ozer, Thomas M. Conte, and Saurabh Sharma, "Weld: a multithreading technique towards latency-tolerant VLIW processors," In Proceedings of the 8th International Conference on High Performance Computing (HiPC'01), Hyderabad, India, December 2001.

[6] S. Balakrishnan, S.K. Nandy, "Multithreaded architecture for media processing," In Proceedings of the 1st Workshop on Media Processors and DSPs (MP-DSP), Haifa, Israel, November 1999.

[7] F. Mombers and D. Mlynek, "A multithreaded multimedia processor merging on-chip multiprocessors and distributed vector pipelines," In Proceedings of the International Symposium on Circuits and Systems ISCAS 1999, Orlando, USA, June 1999.

[8] Jung Ho Ahn, William J. Dally, Brucek Khailany, Ujval J. Kapasi, and Abhishek Das, "Evaluating the Imagine stream architecture," ISCA 2004, Munich, Germany, June 2004.