Microelectronics Journal 46 (2015) 637–655

Contents lists available at ScienceDirect

Microelectronics Journal

journal homepage: www.elsevier.com/locate/mejo

Merging VLIW and vector processing techniques for a simple, high-performance processor architecture

Mostafa I. Soliman [a,b,*]

[a] Computer Science and Information Department, Community College, Taibah University, Al-Madinah Al-Munawwarah 2898, Saudi Arabia
[b] Computer and System Section, Electrical Engineering Department, Faculty of Engineering, Aswan University, Aswan 81542, Egypt

ARTICLE INFO

Article history:
Received 19 April 2014
Received in revised form
9 October 2014
Accepted 20 March 2015
Available online 22 April 2015

Keywords:
Data-parallel applications
VLIW
Vector processing
VHDL
Performance evaluation

ABSTRACT

This paper proposes a new processor architecture called VVSHP for accelerating data-parallel applications, which are growing in importance and demanding increased performance from hardware. VVSHP merges VLIW and vector processing techniques for a simple, high-performance processor architecture. One key point of VVSHP is the execution of multiple scalar instructions within VLIW and vector instructions on unified parallel execution datapaths. Another key point is to reduce the complexity of VVSHP by designing a two-part register file: (1) shared scalar–vector part with eight-read/four-write ports 64×32-bit registers (64 scalar or 16×4 vector registers) for storing scalar/vector data and (2) vector part with two-read/one-write ports 48 vector-registers, each stores 4×32-bit vector data. Moreover, processing vector data with lengths varying from 1 to 256 represents a key point for reducing the loop overheads. VVSHP can issue up to four scalar/vector operations in each cycle for parallel processing a set of operands and producing up to four results to be written back into VVSHP register file. However, it cannot issue more than one memory operation at a time, which loads/stores 128-bit scalar/vector data from/to data memory. The design of our proposed VVSHP processor is implemented using VHDL targeting the Xilinx FPGA Virtex-5 and its performance is evaluated.

1. Introduction

Computer architects have taken many different approaches to high-performance computing by exploiting parallelism. The pipelining technique is the simplest way to take the advantage of parallelism among instructions, see [1] for more details. It exploits instruction-level parallelism (ILP) by overlapping instruction execution to improve the performance. The use of pipelining technique can achieve an ideal performance of one operation per clock cycle. To further improve the performance, parallel processing of multiple operations per clock cycle would be used. Thus, multiple-issue scalar processors like superscalar and very-long instruction word (VLIW) processors fetch multiple scalar instructions and allow multiple operations to be issued in a clock cycle [2–4]. Superscalar and VLIW implementations of traditional scalar instruction sets are sharing in the ability to execute multiple operations simultaneously on parallel execution units. However, the parallelism is explicit in VLIW instructions and must be discovered by hardware at run time in superscalar processors. Therefore, for high performance, VLIW implementations are simpler and cheaper than superscalars because of further hardware simplifications. However, VLIW architectures require more compiler support as discussed in [5].

In addition to ILP, the exploitation of data-level parallelism (DLP) can improve the performance furthermore. DLP can be expressed using vector instruction set and processed on parallel execution units [6]. Traditionally, vector processors fetch a single vector instruction (v operations) and issue multiple operations per clock cycle. Therefore, vector instruction set architecture (ISA) reduces the semantic gap between programs and hardware [7]. Programmers can express parallelism to hardware using vector instructions, otherwise, vector compilers did ultimately get good at synthesizing vector operations even when they were not explicitly expressed. Thus, the generated code says something higher-level, and then the processor manipulates the simple operations on its own [5]. Since the use of vector ISA leads to expressing programs in a more concise and efficient way (high semantic), encoding parallelism explicitly, and using simple design techniques (heavy pipelining and functional unit replication) that achieve high performance at low cost, vector processors remain the most effective way to exploit DLP in data-parallel applications [8–10].

On multiple execution units, this paper proposes a new processor architecture called VVSHP for accelerating data-parallel applications. VVSHP merges VLIW and vector processing techniques for a simple, high-performance processor architecture. VVSHP exploits both of ILP and DLP. In addition to pipelining technique, VVSHP uses

* Correspondence address: Computer Science and Information Department, Community College, Taibah University, Al-Madinah Al-Munawwarah 2898, Saudi Arabia.
E-mail addresses: mossol@ieee.org, mossol@yahoo.com

http://dx.doi.org/10.1016/j.mejo.2015.03.012
0026-2692/© 2015 Elsevier Ltd. All rights reserved.

VLIW architecture (wide-issue static scheduling) to exploit ILP by processing multiple independent scalar instructions concurrently on parallel execution units. DLP is expressed by vector instructions, which are processed on the same parallel execution units of the VLIW architecture. Thus, on unified parallel datapaths, our proposed VVSHP processes multiple scalar instructions packed in VLIW or vector instructions by issuing up to four scalar/vector operations in each cycle. However, it cannot issue more than one memory operation at a time, which loads/stores 128-bit scalar/vector data from/to data memory. Four 32-bit results can be written back into VVSHP register file per clock cycle. To reduce the complexity of VVSHP, this paper designs a two-part register file: (1) shared scalar–vector part with eight-read/four-write ports 64×32-bit registers (64 scalar or 16×4 vector registers) for storing scalar/vector data and (2) vector part with two-read/one-write ports 48 vector-registers, each stores 4×32-bit vector data. Moreover, to reduce the loop overheads, VVSHP processes vector data with lengths varying from 1 to 256. The design of our proposed VVSHP processor is implemented using VHDL targeting the Xilinx FPGA Virtex-5, XC5VLX110T-3FF1136 device. Moreover, the performance of VVSHP is evaluated on vector/matrix kernels.

Data-parallel applications have several different portions of their runtime that can be accelerated to differing degrees [11–13]. On VVSHP, applications can be divided into scalar (unvectorizable) and vectorizable parts. Vectorizable parts can be accelerated on parallel execution units using vector ISA. On the other hand, scalar parts may be accelerated on multiple execution units using VLIW. The compiler and not the hardware is responsible for identifying groups of independent operations and packaging them together into a single VLIW instruction [14,15]. Thus, on VVSHP, not only vectorizable parts are speeded up but also scalar parts. According to Amdahl's Law [16] that governs the speedup of using parallel processing on a problem versus using sequential processing, the performance improvement to be gained from VVSHP by merging VLIW and vector processing techniques is better than either VLIW or vector processors. VVSHP increases the faster mode fraction by parallel processing VLIW/vector instructions on unified multiple execution units.

The rest of this paper is organized as follows. Some related work are discussed in Section 2. Section 3 describes the architecture of our proposed VVSHP processor. Section 4 presents the FPGA implementation of VVSHP on Xilinx Virtex-5. The performance of our proposed VVSHP is evaluated in Section 5. Finally, Section 6 concludes this paper and gives directions for future work.

2. Related work

Data-parallel applications are growing in importance and demanding increased performance from hardware [12,13]. Many architectures have been proposed in the literature to accelerate data-parallel applications. This section emphasizes on the use of vector processing to exploit DLP to accelerate data-parallel applications, even though other techniques such as the use of matrix processing can be used. Examples of using matrix processing for accelerating data-parallel applications include MOM (Matrix Oriented Multimedia) [17], MatRISC (Matrix RISC) [18], ADRES [19], Trident [20], matrix coprocessor for computing matrix product [21], SMP [22,23], and matrix unit for multi-core processors [24].

2.1. Adding vector unit to scalar processors

Asanovic [11] proposed Torrent-0 (T0), which is a single chip fixed-point vector microprocessor designed for multimedia, human-interface, neural network, and other digital signal processing tasks. T0 extends a MIPS-II core with a high performance vector coprocessor and 128-bit wide external memory interface. The vector coprocessor is structured as eight parallel lanes, where each lane contains a portion of the vector register file and one pipeline for each vector function unit.

Quintana et al. [25] proposed adding a vector unit to a superscalar core, as a way to scale superscalar processors. Their architecture has a vector register file that shares functional units both with the integer datapath and with the floating-point datapath. Moreover, it has a high performance cache interface that delivers high bandwidth to the vector unit at a low cost and low latency. The extended vector unit achieves high performance for numerical and multimedia codes with minimal impact on the cycle time of the performance of integer codes.

Espasa et al. [26] proposed Tarantula, which is an aggressive floating-point machine targeted at technical, scientific and bioinformatics workloads. Tarantula adds to the Alpha EV8 core a vector unit capable of 32 double-precision floating-point operations (FLOPs) per cycle. The vector unit fetches data directly from a 16 MByte L2 cache with a peak bandwidth of 64×64-bit per cycle. Tarantula achieves an average speedup of 5x over EV8, out of a peak speedup in terms of FLOPs of 8x.

Kozyrakis [27] proposed a scalable processor (VIRAM) based on vector architecture and IRAM technology. VIRAM has four basic components: MIPS scalar core, vector coprocessor, embedded DRAM main memory, and external IO interface. Enormous, high bandwidth memories (built using DRAM technology) are placed on the processor die to increase the main memory bandwidth. Thus, vector processors with dedicated and integrated memories are good candidates for data-parallel workloads in the embedded systems.

Gebis [28] presented the Virtual Vector Architecture (ViVA), which combines the memory semantics of vector computers with a software-controlled scratchpad memory in order to provide a more effective and practical approach to latency hiding. ViVA adds vector-style memory operations to existing microprocessors but does not include arithmetic datapaths; instead, memory instructions work with a new buffer placed between the core and second-level cache. ViVA gave significant benefit for a variety of memory access patterns like, corner turn and sparse matrix–vector multiplication kernels, without relying on a costly hardware prefetcher.

2.2. Combining vector with multithreading/VLIW/superscalar techniques

Krashinsky [29] proposed a vector-thread (VT) architecture as a performance-efficient solution for all-purpose computing. The VT architectural paradigm unifies the vector and multithreaded compute models. VT provides the programmer with a control processor and a vector of virtual processors. The control processor can use vector-fetch commands to broadcast instructions to all the vector processors or each one can use thread-fetches to direct its own control flow. VT chip, including a scalar RISC control processor and a four-lane vector-thread unit, has 7.1 million transistors, which is complex to implement on many-core technology.

Batten [30] explored a new approach to building data-parallel accelerators that is based on simplifying the instruction set, microarchitecture, and programming methodology for a VT architecture. Batten proposed Maven, which is a malleable array of VT engines that can scale from a few to hundreds of flexible and efficient VT cores tiled across a single chip.

Rivoire et al. [31] proposed vector lane threading (VLT), an architectural enhancement that allows idle vector lanes to run short-vector or scalar threads. To achieve higher performance, both data-level and thread-level parallelism can be exploited on VLT. It multithreads the vector unit to increase the utilization of vector lanes when running low-DLP code. It partitions the vector