Microprocessors and Microsystems 39 (2015) 237–249

Contents lists available at ScienceDirect

Microprocessors and Microsystems

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

# Modular vector processor architecture targeting at data-level parallelism

Seyed A. Rooholamin, Sotirios G. Ziavras *

*Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ 07102, USA*

ARTICLE INFO

*Article history:*
Available online 6 May 2015

*Keywords:*
Parallelism
Vector processor
Performance
Speedup
Benchmarking

ABSTRACT

Taking advantage of DLP (Data-Level Parallelism) is indispensable in most data streaming and multimedia applications. Several architectures have been proposed to improve both the performance and energy consumption for such applications. Superscalar and VLIW (Very Long Instruction Word) processors along with SIMD (Single-Instruction Multiple-Data) and vector processor (VP) accelerators, are among the available options for designers to accomplish their desired requirements. We present an innovative architecture for a VP which separates the path for performing data shuffle and memory-indexed accesses from the data path for executing other vector instructions that access the memory. This separation speeds up the most common memory access operations by avoiding extra delays and unnecessary stalls. In our lane-based VP design, each vector lane uses its own private memory to avoid any stalls during memory access instructions. The proposed VP, which is developed in VHDL and prototyped on an FPGA, serves as a coprocessor for one or more scalar cores. Benchmarking shows that our VP can achieve very high performance. For example, it achieves a larger than 1500-fold speedup in the color space converting benchmark compared to running the code on a scalar core. The inclusion of distributed data shuffle engines across vector lanes has a spectacular impact on the execution time, primarily for applications like FFT (Fast-Fourier Transform) that require large amounts of data shuffling. Compared to running the benchmark on a VP without the shuffle engines, the speedup is 5.92 and 7.33 for the 64-point FFT without and with compiler optimization, respectively. Compared to runs on the scalar core, the achieved speedups for this benchmark are 52.07 and 110.45 without and with compiler optimization, respectively.



## 1. Introduction

High-performance computing processors often have a superscalar or VLIW architecture that focuses mostly on exploiting ILP (Instruction-Level Parallelism). Roger et al. [1] show that ILP and DLP can be merged in a single simultaneous vector multithreaded architecture for higher performance. VIRAM's [2] basic multi-lane architecture can be used to build VPs that exploit DLP through SIMD processing. Each lane contains similar pipelined execution and load-store units. Each vector register is uniformly distributed among the lanes. All the elements from a vector in a lane are processed sequentially in its pipelined units while corresponding elements from different lanes are processed simultaneously. Using EEMBC benchmarks, it was demonstrated that a cache-less VIRAM is much faster than a superscalar RISC or a cache-based VLIW processor [3].

The SODA VP has a fully programmable architecture for software defined radio [4]. Using SIMD parallelism and being optimized for 16-bit computations, it supports the W-CDMA and IEEE802.11a protocols. Embedded systems using a soft core or hard core processor for the main execution unit also have the option to attach a hardware accelerator to increase their performance for specialized tasks. Sometimes these accelerators are realized using FPGA (Field-Programmable Gate Array) resources to speed up applications with high computational cost. Designing a custom hardware accelerator that will yield outstanding performance needs good knowledge of HDL (Hardware Description Language) programming. Another SIMD, FPGA-based processor uses a 16-way data path and 17 memory blocks as the vector memory in order to perform data alignment and avoid bank conflicts [5]. VESPA [6] is a portable, scalable and flexible soft VP which uses the same instruction set as VIRAM but the coprocessor architecture was hand-written in Verilog with built-in parameterization. It can be scaled with regards to the number of lanes and yields 6.3× improvement with 16 lanes for EEMBC benchmarks compared to a one-lane VP. It is flexible as the size of the vector length and its width, as well as the memory crossbar, can vary according to the target application. The VIPERS soft VP is a general-purpose accelerator that can achieve a 44× speedup compared to the Nios II scalar processor [7]; it increase

* Corresponding author.
*E-mail address:* Ziavras@njit.edu (S.G. Ziavras).

http://dx.doi.org/10.1016/j.micpro.2015.04.007
0141-9331/

the area requirements 26-fold. It supports specific instructions for the applications, such as motion estimation and median filters, and can be parameterized in terms of number of lanes, maximum vector length and processor data width. VEGAS [8] is a soft VP with cache-less scratchpad memory instead of a vector register file. It achieves 1.7–3.1× improvements in the area-delay product compared to VESPA and VIPERS. With the integration of a streaming pipeline in the data path of a soft VP, a 7000× times speedup results for the N-body problem [9].

An application-specific floating-point accelerator is built using a fully automated tool chain, co-synthesis and co-optimization for SIMD extension with a parameterizable number of vector elements [10]. An application-specific VP for performing sparse matrix multiplication was presented in [11]. IBM's PowerEN processor integrates five hardware application specific accelerators in a heterogeneous architecture to perform key functions such as compression, encryption, authentication, intrusion detection and XML processing for big workload network applications. Hardware acceleration facilitates energy-proportional performance scaling [12]. An innovative lane-based VP which can be shared among multiple cores in a multicore processor was proposed in [13]; it improves performance while maintaining low energy cost compared to a system with exclusive per-core VPs. Three shared-vector working policies were introduced for coarse-grain, fine-grain and exclusive vector-lane sharing, respectively. Benchmarking showed that these policies yield 1.2–2× speedups compared to a similar cost system where each core contains its own dedicated VP.

A major challenge with these VPs is slow memory accesses. Comprehensive explorations of MIMD, vector SIMD and vector thread architectures in handling regular and irregular DLP efficiently confirm that vector-based microarchitectures are more area and energy efficient compared to their scalar counterparts even for irregular DLP [14]. Lo et al. [15] introduced an improved SIMD architecture targeted at video processing. It has a parallel memory structure composed of various block sizes and word lengths as well as a configurable SIMD architecture. This structure can perform random register file accesses to realize complex operations, such as shuffling, which is quite common in video coding kernel functions. A crossbar is located between the ALU (Arithmetic Logic Unit) and register file.

In a VIRAM-like architecture, a memory crossbar often connects the lanes to the memory banks to facilitate index memory addressing and data shuffling. This crossbar adds extra delay when not actually needed, such as for stride loads and stores. Moreover, it increases the energy consumption. Adding a cache to each lane may solve this problem to some extent but the cache coherence problem will require an expensive solution, often prohibitive for embedded systems. Since in practical applications stride addressing is more common than other types of addressing [16], we introduce here a VP model that does not sacrifice performance for less likely memory access instructions. We develop a VIRAM-based, floating-point VP embedded in an FPGA that connects to a scalar processor. This VP comprises four vector lanes, and provides two separate data paths for each lane to process and execute load and store operations in the LDST (Load–Store) unit in parallel with floating-point operations in the ALU. Each cache-less lane is directly attached to its own local memory. Data shuffle instructions are supported by a shuffle engine in each lane which is placed after the lane's local memory and connects to other lanes via a combinational crossbar. All the local memories connect to the shared bus which is used to exchange data between these memories and the global memory. The prototyping of a system with four lanes shows substantial increases in performance for a set of benchmarks compared to similar systems that do not contain the shuffle engines.

Previously proposed VPs are not versatile enough in multithreading environments. They were mostly capable of handling simultaneously multiple threads using the same vector length in predefined contexts. However, this approach is not often efficient for real applications since a VP is a rather high-cost, high-performance accelerator that consumes considerable area and energy in multicore processors. A more flexible VP that can be shared dynamically by multiple cores results in better resource utilization, higher performance and lower energy power dissipation. Our proposed solution supports the simultaneous processing of multiple threads having diverse vector lengths. In fact, the vector lengths used by any given thread are allowed to change during execution. The following sections show the detailed architecture of our VP, benchmarking results on an FPGA prototype, and performance analysis.

## 2. Methodology and realized architecture

### 2.1. System architecture prototyping on FPGA

Fig. 1 depicts the basic architecture of the FPGA-prototyped VP introduced in this paper. For the sake of simplicity we show a single scalar core, Xilinx's soft core MicroBlaze (MB) that fetches instructions from its instruction memory (not shown in the figure) and issues them to appropriate execution units. The MB is in charge of executing all scalar and control instructions while vector instructions are sent to the VP. The shuffle engine, which is distributed along the lanes, is activated only to realize vector data shuffling with multiple vector lanes. Our design introduces two innovative concepts. First, it removes the competition of lanes to access memory banks, which is the case for earlier works, by employing cache-less private memories for the lanes; the private memories form a low-order interleaved space that resides between the lanes and the global memory. Second, the vector length can vary even between instructions in the same thread. In all previously introduced VPs, the vector length was defined for each working context, program or thread. It was usually a fixed number for each thread and was set in advance by the scheduler. In contrast, our model allows us to define the vector length for each individual instruction. As a result, the vector length can vary widely, even for instructions in the same loop. Although usage of a mask register could potentially have the same effect, the performance can degrade.

Data needed by applications running on the VP should be preferably stored in the private memories of lanes. Since these private memories connect to the AXI (Advanced eXtensible Interface)

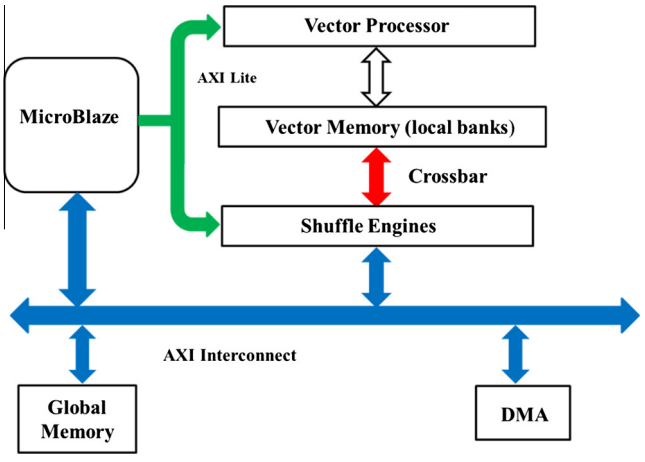


**Fig. 1.** High-level architecture of the multi-lane VP prototyped on a Xilinx FPGA. The vector memory is low-order interleaved. Each vector lane is attached to a private memory.