Scalable Softcore Vector Processor for Biosequence Applications

Arpith C. Jacob, Brandon Harris, and Young H. Cho
{jarpith,bbh2}@cse.wustl.edu,young@arl.wustl.edu
Department of Computer Science and Engineering, Washington University in St. Louis
St. Louis, Missouri 63130–4899

1 Introduction

Currently available genome databases are growing exponentially in size[1], making it difficult for software analysis tools to keep up. A number of hardware accelerators utilizing special purpose VLSI [1] or reconfigurable hardware [2] have been proposed. However, they are inflexible; support for new applications usually requires a laborious redesign. None of these accelerators can be easily adapted to other applications that require differing hardware resources.

The design philosophy of the *Softcore Vector Processor* is based on two important goals: adaptability and performance. Instruction based execution allows programmable support for a large number of algorithms. The fact that different classes of applications require different subsets of hardware resources, argues for a customizable hardware design built from primitives. The second goal was to achieve programmability without sacrificing performance. The SVP was designed to perform competitively with full custom solutions available in the market.

2 System Architecture

The SVP is designed to be part of a pipelined, high throughput computation architecture. The system architecture consists of one or more chained *Functional Units* (FU) that performs a specific transformation (algorithm) on the data stream.

Functional Unit: (Figure 1) A user programmable *Instruction Controller* (IC) broadcasts instructions to a linear array of *Processing Elements* (PEs) which support the Single Instruction stream Multiple Data stream (SIMD) paradigm. The *Memory Controller* (MC) provides temporary storage for the PEs and the IC, as well as playing a vital role in providing access to data streams.

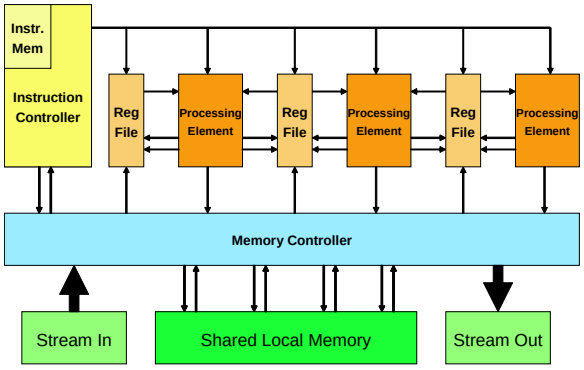

Figure 1. The Softcore Vector Processor Functional Unit

Instruction Controller: Instructions are loaded from local, programmable instruction memory. There are two classes of instructions: those broadcast to PEs, and those executed in the IC and meant primarily to affect program flow (such as an atomic loop instruction). An observation made during the analysis of common sequence analysis algorithms was that a large number of registers in the PEs were used to store constants or base addresses of tables in memory. These are instead moved to IC registers and broadcast as an immediate operand to all PEs. On our target application, instruction register broadcast, reduced the number of registers required on PEs by 40%.

Processing Element: Processing Elements are resource efficient, independently operating core computational units, that implement common operations (such as MAX/MIN). Register files on the

[1] http://www.ncbi.nlm.nih.gov/Genbank/genbankstats.html

Table 1. Comparative Performance

System	Freq (MHz)	Mcups/ PE	PEs/ chip	Mcups/ chip
SVP128	150	2.95	128	378
SVP32	150	3.22	32	103
Kestrel	20	0.78	64	50
Fuzion150	200	1.63	1536	2500
GeneM.2	192	5.21	192	1000

SVP are shared by two processing elements placed adjacent to it. In addition to providing storage space, they facilitate communication between PEs. Conditional commands *bmset* and *bmend* delimit blocks of code that are to be executed on a subset of PEs, by operating on a special 8-bit mask register in each PE. *bmset* compares two operand registers and performs a bit left shift and set operation on the mask register if the condition is satisfied. The logical *and* of all bits in the register is used to switch it on or off.

Memory Controller: The memory controller provides concurrent read/write access to temporary storage for the PEs and the IC. Memory may be organized in distributed or shared fashion. We make use of replicated block RAM storage for this purpose. An important function of the memory controller is to provide stream processing capabilities to the PEs. Input and output stream FIFOs are memory mapped in the address space of the memory controller. This provides the programmer access to streams using the same load/store instruction used to access on chip memory.

3 Performance

The Smith-Waterman algorithm [3] is used to compare two DNA or protein sequences for similarity, in the hope of detecting common features. We have implemented a fine-grained parallel version of the Smith-Waterman algorithm on the SVP16 simulation on Modelsim. We extrapolate performance numbers based on this to larger configurations. To quantify its performance we define the measure Millions of Cell Updates per Second (MCUPS), which represents the number of matrix cells that can be computed per second. We compare the performance of the SVP with varying numbers of 16-bit wide PEs: 32 and 128, estimated to run at 150 MHz on a single Xilinx Vertex 4 VLX200 part.

Table 1 compares the performance of the SVP against other systems that run Smith-Waterman. The Kestrel and Fuzion 150 are software programmable (but not hardware customizable) parallel processors built on custom ASICs. The SVP provides the best performance per PE compared to the other programmable architectures. The per chip performance numbers are between two and seven times better than the Kestrel system. The Paracel GeneMatcher2 is an FPGA based custom solution to accelerating Smith-Waterman. Although GeneMatcher2 out-performs all of the previously mentioned designs, its architecture lacks the flexibility of a programmable processor. Since the Kestrel uses 8-bit data widths, we synthesized an 8-bit variant using 256 processing elements running at an estimated 120 MHz. This implies that the SVP can potentially double its performance by simply reducing the result precision to match the Kestrel.

4 Conclusion

The SVP was designed to be a high-performance programmable system. We believe its stream processing capabilities and its ease of programming will allow us to run a large class of applications. The design choice of targeting FPGAs and the corresponding hardware customizability afforded, will enable us to incrementally add to the compute capabilities, while still maintaining comparable performance with the state of the art.

References

[1] A. D. Blas, D. Dahle, M. Diekhans, L. Grate, J. Hirschberg, K. Karplus, H. Keller, M. Kendrick, F. J. Mesa-Martinez, D. Pease, E. Rice, A. Schultz, D. Speck, and R. Hughey. The kestrel parallel processor. *IEEE Trans. Parallel Distrib. Syst.*, 16(1), 2005.

[2] D. T. Hoang. Searching genetic databases on splash 2. In D. A. Buell and K. L. Pocek, editors, *IEEE Workshop on FPGAs for Custom Computing Machines*, pages 185–191, Los Alamitos, CA, April 1993. IEEE Computer Society Press.

[3] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.