Block-Based Execution on an Integrated Vector-Scalar In-Order Core

Milan Stanic1, Oscar Palomar2
[1]Barcelona Supercomputing Center, [2] University of Manchester
milan.stanic@bcs.es, oscar.palomar@manchester.ac.uk

Abstract-*In the low-end processor mobile market, power, energy and area budgets are significantly lower than in the server/desktop/lap-top/high-end mobile markets. It has been shown that vector processors are a highly energy-efficient way to increase performance but adding support for them incurs area and power overheads that could not be acceptable for low-end mobile processors. In this work, we propose an integrated vector-scalar design that mostly reuses scalar hardware to support the execution of vector instructions. The key element of the design is our proposed block-based model of execution that groups vector instructions to execute them in a coordinated manner.*

I. INTRODUCTION

In the last 15 years, energy consumption and power dissipation have become crucial design concerns for almost all computer systems due to several reasons: for example, technology feature size scaling leads to higher power density and therefore to costly cooling. While power dissipation is critical for high-performance systems such as data centers due to large power usage, battery life is a primary concern for mobile systems.

Driven with this goal, researchers have focused on improving performance in an energy-efficient way. Vector processors [1] are energy efficient architectures that yield high performance whenever there is enough data-level parallelism (DLP) [2]. Besides the long and successful history of vector processors in supercomputers, vector units have been adopted in designs of microprocessors [3, 4, 5]. Also, SIMD multimedia extensions [6, 7] are often included in modern microprocessors. Recent research on vector processors shows that they can be a good match even for applications from domains such as column-store databases [8]. The Xeon Phi is a recent massively parallel x86 microprocessor designed by Intel and is based on the Larrabee [9] GPU, that contains a 512-bit SIMD vector processing unit in each core.

This paper contributes a method to increase the

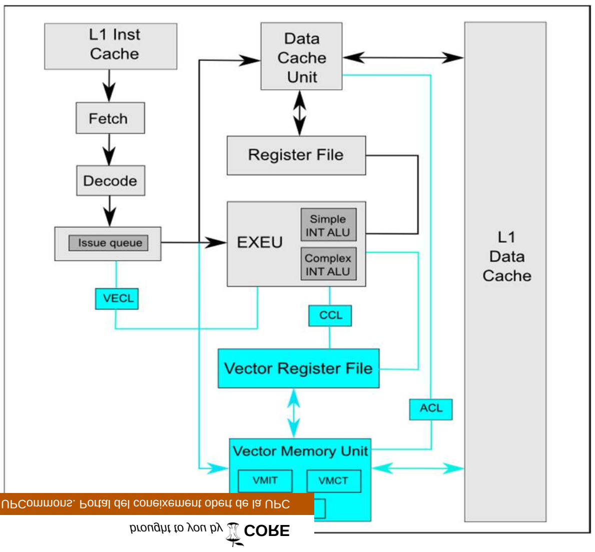
in an energy-efficient way. The energy efficiency is accomplished by modifying a scalar core to execute vector instructions on the existing scalar infrastructure. In particular, we propose an integrated vector-scalar design that combines scalar and vector processing mostly using existing resources of an energy-efficient processor (in our evaluation environment, it is based on the ARM Cortex A7). In addition to a design that uses a conventional vector execution model, we also contribute a novel block-based model of execution for vector computational instructions.

II. INTEGRATED DESIGN

As a baseline, we use a scalar core based on the highly energy-efficient ARM Cortex-A7. It is an in-order, dual-issue processor that implements the ARM v7 architecture with an 8-stage pipeline (gray blocks in Figure 1).

In our proposed integrated vector-scalar design, we attempt to maximize the reuse of resources already present in the baseline scalar core (white blocks in Figure 1) while adding support for vector instructions. While the front-end of pipeline is the same (fetch and decode stages), in the back-end we added two structures to support the execution of vector instructions on the scalar core: a vector register file, and a vector memory unit (blue blocks in Fig. 1). There is also additional logic that controls the execution of vector instructions: vector execution control logic (VECL), aliasing control logic (ACL) and chaining control logic (CCL). VECL is added in the issue stage to support the execution of computational vector instructions. ACL exchanges information between the vector memory and the data cache unit and forces scalar and vector memory instructions to be executed in-order. CCL is responsible for the execution of chained dependent computational instructions.

Fig. 1. Block diagram of the integrated design.



A. Execution of Vector Computational Instructions

We study two alternatives for executing the vector computational instructions on the existing scalar FUs: 1) the One-By-One model of execution (OBO), in essence the classic vector execution model, in which every instruction is executed to completion, i.e. for all the operations of the vector; and 2) a novel execution model called Block-Based Execution (BBE). In this model, for a block of consecutive vector computational instructions, first all operations on the first element are executed, then the operations of the second element, and so on. Fig. 2 shows an example with a sequence of vector instructions, illustrating the difference of the two execution models. For this example, we assume that vector instructions operate on floating-point data by using a single floating-point unit and a single data cache port. The first *vecload* instruction is executed in the same way and at the same time on both models, since the models refer only to computational instructions. Regarding computational vector instructions, in OBO (Fig. 2 (a)) all operations of one vector computational instruction (*vecadd*) are executed, and then we move on to the next vector instruction (*vecsub*). In BBE (Fig. 2 (b)), several consecutive vector computational instructions form a block of vector instructions, and we execute one operation from each instruction of the block and repeat this for each operation in the block of vector instructions. In the example, we execute one operation from *vecadd* and then one operation from *vecsub*. The process ends once all operations are computed. The next subsection describes the BBE model in more detail.

A. Block-Based Execution

To support this model of execution, we added a small table that keeps the information of the instructions of the block and simple control logic. In this paper, blocks of vector computational instructions are formed dynamically in a very simple way. Once a computational vector instruction is ready for execution, the control logic examines the next instructions in the issue queue and adds them to the block if they are vector computational instructions, until another instruction type (a scalar or vector memory instruction) is encountered or the block is full.

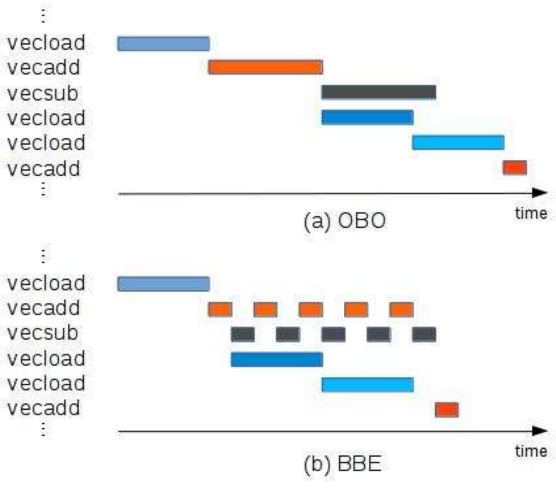


Fig. 2. An example of code with vector instructions executed with one ALU assuming (a) the one-by-one and (b) the block-based execution model.

The number of vector instructions that can be executed in parallel or with chaining using the OBO model is restricted by the number of available ALUs. BBE does not have this limitation, allowing for execution of more vector instructions in parallel. Inherently, more dependent instructions can be chained (scalar bypass logic can be reused) since one vector instruction does not occupy the ALU for all its elements in continuous cycles, and thus it can be interleaved with other instructions using the same ALU. An important advantage of BBE over OBO or a classic vector unit is the following: while a block of vector computational instructions is under execution, BBE allows for the execution of subsequent scalar or vector memory instructions if they are ready for execution and there are free functional units that can execute them. In Fig. 2 (b), the second *vecload* instruction can start execution just after the *vecsub* started with execution of the first operation.

III. CONCLUSION

Using a vector processor is one of the most energy efficient ways of achieving high performance for a wide number of applications that contain a significant degree of DLP. Power dissipation, energy consumption and area are critical concerns in processor design, especially for embedded systems in the low-end market. In this paper, we propose the integrated vector-scalar design that allows for execution of vector computational instructions mostly reusing resources of an in-order core. We also propose block-based execution model to execute vector computational instructions.

REFERENCES

[1] K. Asanovic. *Vector Microprocessors*. PhD thesis, University of California, Berkeley, May, 1998.
[2] Y. Lee et al. *Exploring the trade-offs between programmability and efficiency in data-parallel accelerators*. In ISCA 38, pages 129-140, 2011.
[3] C. Kozyrakis and D. Patterson. *Overcoming the limitations of conventional vector processors.* In Proceedings of the 30th ISCA, pages 399-409, 2003.
[4] R. Espasa et al. *Tarantula: a vector extension to the Alpha architecture*. In ISCA 29, pages 281-292, 2002.
[5] C. F. Batten. *Simplified vector-thread architectures for flexible and efficient data-parallel accelerators.* PhD thesis, Cambridge, MA, USA, 2010. AAI0822514.
[6] S. Thakkar and T. Huff. Internet streaming SIMD extensions. Computer, 32:26-34, December 1999.
[7] M. Buxton et al. Intel AVX: *New frontiers in performance improvements and energy efficiency*. White paper, 2008.
[8] T. Hayes et al. *Vector extensions for decision support dbms acceleration*. In MICRO 45, pages 166-176, 2012.
[9] L. Seiler et al. Larrabee: *a many-core x86 architecture for visual computing.* In SIGGRAPH '08, pages 18:1-18:15, 200