 DOI 10.1109/PACT.2019.00054

POSTER: An Optimized Predication Execution for SIMD extensions

Adrián Barredo*†, Juan M. Cebrián*, Miquel Moretó*†, Marc Casas* and Mateo Valero*†
*Barcelona Supercomputing Center**, *Universitat Politècnica de Catalunya*†
firstname.lastname@bsc.es

***Abstract*—Vector processing is a widely used technique to improve performance and energy efficiency in modern processors. Most of them rely on predication to support divergence control. However, performance and energy consumption in predicated instructions are usually independent on the number of true values in a mask. This means that the efficiency of the system becomes sub-optimal as vector length increases.**

In this work we propose the Optimized Predication Execution (OPE) technique. OPE delays the execution of *sparse* masked vector instructions sharing the same PC, extracts their active elements and creates a new *dense* instruction with a higher mask density. After executing such dense instruction, results are restored to the original sparse instructions. Our approach improves performance by up to 25% and reduces dynamic energy consumption by up to 43% on real applications with predication.

I. Introduction

Data-Level Parallelism (DLP) can be exposed to the hardware by means of vector computations [1], [2], where a Single Instruction operates over Multiple Data streams (SIMD). SIMD extensions appeared to improve multimedia applications efficiency and they are dominant in current processors. However, predicated vector instructions operate independently of the active elements in the mask operands. As such, the execution time of predicated instructions depends on the architecture vector length (VL) and not on the active elements in the mask register. As a result, current SIMD implementations have VL-time performance, waste a significant energy on unnecessary computations and increase contention in the Vector Functional Unit (VFU). With the current trend of doubling the register size every four years [3], this situation will become unsustainable. Ideally, the execution time and energy consumption of predicated instructions should be proportional to the fraction of true/false values in the mask. Such an implementation would achieve *density-time* performance and energy efficiency.

In this work we propose a novel hardware mechanism, the Optimized Predication Execution (OPE) design. OPE achieves density-time performance and energy efficiency in SIMD extensions without programmer intervention.

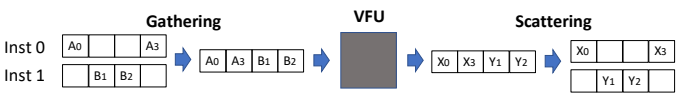

Figure 1. OPE basic functionality.

II. The OPE Mechanism

A. Overview

OPE creates a *dense* version of several dynamic predicated instructions for a certain PC. The active elements (vector elements whose corresponding mask bits are true) of these vector instructions are *gathered* into a dense instruction. In the best scenario, this dense instruction has source registers with all elements active and it is executed instead of the original instructions. As a result, the number of accesses to the VFU decreases. This is crucial for performance and energy efficiency, since the VPU can add up to 75% of the total core dissipated power [4]. Once the dense instruction is executed, results are *scattered* back to the destination registers of each original sparse instruction. OPE can be implemented in any architecture with predication support.

OPE basic functionality is shown in Figure 1. In this case, two predicated instructions with 50% mask densities, corresponding to two loop iterations for the same PC, are optimized. After gathering their active elements from the source registers into the dense, they are executed and their results are scattered into the original destination registers.

B. Hardware Components

OPE requires the following four new hardware elements:

1) The **Predication Instruction Table** (PIT) contains information about dense and sparse predicated instructions. It is needed to perform the gathering and scattering phases.

2) The **Predication Ticket Table** (PTT) keeps track of the latest created dense instruction for every PC. It facilitates the accesses to the PIT, since there can be multiple dense instructions for the same PC waiting to be executed.

3) The **Gather Unit** creates a dense version of several sparse source vector registers. The active elements of the source registers are moved into the lanes (position in a vector register that contains an element) of the dense register.

4) The **Scatter Unit** restores the results of an executed dense instruction back to the original destination registers. The dense destination register elements are moved to the corresponding active lanes of the destination registers.

C. Integration into an Out-of-Order Processor

Next, the main functional changes to incorporate OPE into a classic out-of-order processor are described.

1) **Decode**: In case a predicated instruction is found a signal is sent to Issue stage.

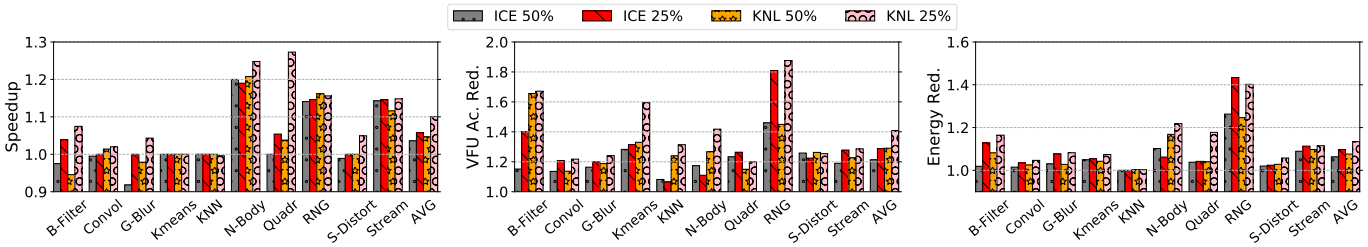

Figure 2. Performance (left), VFU access reduction (center) and dynamic energy reduction (right) results of OPE. Normalized to a non-OPE scenario.

2) **Issue**: If the signal from Decode is active and the mask register is ready, a logic decides whether the instruction is a candidate to be optimized. Then, the PTT and PIT are accessed to know if it is the first candidate for that PC. In case a new dense is required, it is created and its operands are renamed. The PTT creates and stores a new ticket, which is provided to the optimized instruction and employed to create a new PIT entry. A reservation station (RS) and a re-order buffer (ROB) entry are allocated for the dense instruction. Also, a dense destination register is reserved in the Register Alias Table (RAT) to allow operand forwarding. Candidates to be gathered into this dense instruction are given the PTT ticket after their mask operand becomes ready.

3) **Dispatch**: As candidate operands become ready, their active lanes are gathered, their RS are freed and PIT fields are updated. Once dense operands are completely populated, a timeout occurs, or a squash is triggered, the instruction becomes ready to execute.

4) **Execution**: The dense instruction is executed and optimized instructions are bypassed. If the dense destination register is used by subsequent dense instructions, it is forwarded.

5) **Writeback**: The dense instruction is written in the corresponding ROB entry. Then, the lanes are scattered into the original destination registers.

6) **Commit**: Dense and sparse instructions commit sequentially, ensuring speculation and exception handling are performed in-order.

III. Methodology

We evaluate the performance of OPE in the *gem5* [5] simulator. We have deployed OPE into an x86 processor with AVX-512 support, which contains predicated instructions. OPE functionality has been added to the simulator and the new hardware latencies are modelled. Our baseline is a processor without OPE support. We have selected ten applications which have been vectorized using Intel intrinsics [6] to extract maximum vectorization. We simulated an x86 system with a 16.04 Ubuntu and a 4.9.4 Linux kernel. Two micro architectures are modelled: a latency and a throughput-oriented implementation based on the Icelake (ICE) [7] and the Knights Landing (KNL) [8].

IV. Evaluation & Conclusions

Figure 2 shows the results of executing OPE with 25% and 50% mask densities in terms of speedup, VFU access reduction and dynamic energy reduction. Results are normalized to a regular no-OPE execution. On average, applications achieve between 3.6% and 10% performance speedups, between 21% and 41% VFU access reductions, and between 6.2% and 13.4% dynamic energy reductions.

Applications such as N-Body and RNG, contain a high percentage of long latency vector instructions per loop iteration. This situation leads to higher performance benefits, as there is more contention in the VFU.

B-Filter and S-Distort also contain long latency vector instructions. However, a higher number of instructions per loop iteration prevents an efficient population of the dense registers.

Other benchmarks, such as Convol, only contain low-latency predicated instructions. Nevertheless, an irregular memory access pattern hides OPE latencies and it is able of marginally improve performance up to 5%.

In all the experiments, the KNL configuration provides more optimization opportunities to the OPE mechanism as there is more contention in the VFU. Also, lower mask densities (i.e. 25%) lead to more optimization opportunities.

V. Acknowledgments

This work has been partially supported by the RoMoL ERC Advanced Grant (GA 321253), the European HiPEAC Network of Excellence and the Spanish Government (contract TIN2015-65316-P). A. Barredo has been supported by the Spanish Government under Formación del Personal Investigador fellowship number BES-2017-080635. M. Moretó has been partially supported by the Spanish Ministry of Economy, Industry and Competitiveness under Ramon y Cajal fellowship number RYC-2016-21104.

References

[1] K. Asanovìc, "Vector Microprocessors," Ph.D. dissertation, 1998.
[2] R. Espasa, V. M., and J. E. Smith, "Vector architectures: past, present and future," ser. ICS '98. ACM, 1998.
[3] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Sixth Edition: A Quantitative Approach*, 6th ed. Morgan Kaufmann, 2017.
[4] A. Sodani, "Race to Exascale: Opportunities and Challenges," ser. Micro '11 Keynote, 2011.
[5] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib Bin Altaf, N. Vaish, M. Hill, and D. Wood, "The gem5 simulator," vol. 39, 08 2011.
[6] Intel Corporation. Intel Intrinsics Guide. [Online]. Available: https://software.intel.com/sites/landingpage/IntrinsicsGuide/
[7] VentureBeat, "Intel confirms Ice Lake Core CPUs with 10nm+ process to followup its 8th-gen chips," 2017. [Online]. Available: https://venturebeat.com/2017/08/14/intel-confirms-ice-lake-core-cpus-with-10nm-process-to-followup-its-8th-gen-chips/
[8] A. Sodani, "Knights landing (KNL): 2nd Generation Intel Xeon Phi processor," in *Hot Chips*, 2015.