Tensor Processing Primitives in the Computational Sciences: Earthquake Simulations

Alexander Breuer, Antonio Noack
Faculty of Mathematics and Computer Science
Friedrich Schiller University Jena
Jena, Germany
{alex.breuer, antonio.noack}@uni-jena.de

Evangelos Georganas, Alexander Heinecke, Kirill Voronin
Intel Labs
Intel Corporation
Santa Clara, USA
{evangelos.georganas, alexander.heinecke, kirill.voronin}@intel.com

***Abstract*—Many HPC and certainly Artificial Intelligence (AI) or Deep Learning (DL) applications are comprised in their core of small linear algebra operations which are then used to compose large and more complicated tensor operations. Especially in the field of AI / DL portability among different hardware platforms is essential due to an extensive reliance on Python and the high-level nature of many frontends. However, scientists are often faced with the challenge to run their codes in vastly different environments. They therefore have to restrict themselves to high-level languages and hope for good compiler optimizations. Especially for complicated linear algebra operators, as they arise in high-order methods in the computational sciences, this is huge leap of faith. In this work we demonstrate how Tensor Processing Primitives (TPPs), a low-dimensional SIMD abstraction for various CPU architectures, can be used to obtain very high fractions of floating point peak on seven different CPU micro-architectures offering four different ISAs.**

***Index Terms*—small linear algebra, runtime code generation, SIMD architectures, seismic simulations**

I. Introduction

We present Tensor Processing Primitives (TPPs), [1], in the context of earthquake simulations, a computational sciences workload. TPPs are a small set of about 50 versatile low-dimensional tensor operators which allow for the formulation of higher-dimensional tensor operations. We use the library LIBXSMM as our backend. LIBXSMM just-in-time generates efficient machine code by hardwiring kernel- and hardware-specific optimizations.

In previous work the library fully supported the TPP concept for x86 vector extensions, i.e., Intel AVX2 and Intel AVX512, and ARM AArch64's ASIMD. Until this point LIBXSMM's support for the ARM AArch64 Scalable Vector Extension (SVE) was solely limited to dense matrix-matrix multiplications. We make the following contributions:

- Full SVE support for unary, binary and ternary TPPs in LIBXSMM. Support and optimization for processors with 256-bit and 512-bit SVE.
- Added SVE support for (sparse matrix) $\times$ (3D tensor) and (3D tensor) $\times$ (sparse matrix) TPPs.
- Integration of unary and binary TPPs into the Extreme-Scale Discontinuous Galerkin Environment (EDGE) for demanding seismic setups.
- Support for TPP-equation-based time and volume integrators in EDGE.
- Thorough benchmarking, analyses and discussion of key standalone primitives on seven recent microarchitectures.
- Full application benchmarking on eight recent processors. Discussion of EDGE's performance results when running a single seismic forward simulation and when running fused forward simulations.
- Two large-scale simulations of the 2014 Mw 5.1 La Habra, California Earthquake on up to 2,048 nodes of the Frontera supercomputer.

II. Performance Portability

- 2nd Generation Intel(R) Xeon(R) Scalable Processor (Plat. 8280), abb.: CLX, #Cores: 28, TFLOPS: 4.28
- 3rd Generation Intel(R) Xeon(R) Scalable Processor (Plat. 8380), abb.: ICX, #Cores: 40, TFLOPS: 6.36
- 4th Generation Intel(R) Xeon(R) Scalable Processor, abb.: SPR, #Cores: hcc, TFLOPS $\geq$8.5
- AMD EPYC 7742 (Rome), #Cores: 64, TFLOPS: 5.29
- AMD EPYC 7J13 (Milan), #Cores 64, TFLOPS: $\geq$5.9
- Amazon Graviton2 (GRT2), #Cores: 64, TFLOPS: 2.56
- Fujitsu A64FX (A64FX), #Cores: 48+4, TFLOPS: 5.45
- Amazon Graviton3 (GRT3), #Cores: 64, TFLOPS: 3.42

The list above shows the different testbeds used to study the performance portability of the TPP approach in scientific applications. We also depict the FP32 Fused-Multiply-Add (FMA) throughput out of registers as we later correlate realized application performance with these peak numbers. In our case we define performance portability as achieving the same fraction of FMA peak performance across different architectures for the same application.

Tab. I covers the achieved fraction of peak FMAs for the ternary GEMM TPP as a key kernel for single seismic forward simulations using EDGE in the column %peak dense. Examples for a fifth order instantiation of the solver with viscoelastic attenuation are presented below. All kernels were benchmarked out of a hot L1 cache on a single core.

Tab. I also covers %peak for fused-simulation kernels in EDGE. In this case we run several forward simulations data-parallel in the SIMD units of the CPU. The execution mode can be regarded as vectorization across multiple right-hand-sides. The per-tetrahedral-element degrees of freedom matrix now becomes a 3D tensor in which the fastest running

dimension is the number of concurrent forward simulations. The second dimension holds the the quantities and the outer most dimension contains the number of modes of the higher order element. Therefore, in fused mode, EDGE's performance heavily depends on (sparse matrix) x (3D tensor) and (3D tensor) x (sparse matrix) TPPs, where the sparse matrices are, e.g. stiffness matrices. The approach enables exploitation of zeros appearing in the occurring matrix structures. Example matrix patterns are shown on the poster, respective performance measurements are provided in Tab. I. In case of CLX that means a single kernel processes 16 forward simulations, without doing any matrix multiplications involving zero entries. The occurring sparsity patterns are "hardwired" into the instruction stream, allowing for perfect utilization (unmasked/non-predicated) of SIMD units and without any scalar control-flow analyzing the sparsity pattern at runtime. However, due to the unstructured nature of the sparsity patterns we are not always able to hide all FMA latencies, thus the efficiency of the sparse kernels is lower than their dense small GEMM counterparts. Yet, the sparse kernels typically offer much higher overall throughput. This can be seen in the last column of Tab. I for the kernels running out of hot local caches.

TABLE I
ACHIEVED FRACTION OF PEAK FOR DENSE MATRIX-MATRIX & SPARSE MATRIX-TENSOR PRODUCTS

Processor	*Matrix*	*%Peak dense*	*%Peak sparse*	*Speed -Up*
CLX	tet4_5_stiffV0	58.2	29.7	3.40
	tet4_4_stiffV0	37.1	32.3	5.64
	tet4_starMatrix (O6)	54.9	13.8	0.85
	tet4_starMatrix (O5)	30.5	15.8	1.75
ICX	tet4_5_stiffV0	71,5	32.9	3.07
	tet4_4_stiffV0	48.8	25.9	3.44
	tet4_starMatrix (O6)	56.9	19.9	1.18
	tet4_starMatrix (O5)	40.4	27.1	2.26
SPR	tet4_5_stiffV0	82.0	65.3	5.31
	tet4_4_stiffV0	63.9	53.1	5.38
	tet4_starMatrix (O6)	77.6	30.5	1.33
	tet4_starMatrix (O5)	65.9	51.1	2.62
Rome/ Milan	tet4_5_stiffV0	92.9	61.8	4.43
	tet4_4_stiffV0	51.2	56.4	7.14
	tet4_starMatrix (O6)	93.6	62.1	2.24
	tet4_starMatrix (O5)	42.1	62.3	5.00
GRT2	tet4_5_stiffV0	93.5	45.0	3.21
	tet4_4_stiffV0	77.4	39.0	3.26
	tet4_starMatrix (O6)	79.6	36.0	1.53
	tet4_starMatrix (O5)	68.4	38.2	1.89
A64FX	tet4_5_stiffV0	60.8	24.2	2.65
	tet4_4_stiffV0	42.0	22.5	3.47
	tet4_starMatrix (O6)	47.3	22.3	1.59
	tet4_starMatrix (O5)	34.3	24.3	2.39
GRT3	tet4_5_stiffV0	94.3	55.5	3.92
	tet4_4_stiffV0	69.6	45.6	4.25
	tet4_starMatrix (O6)	81.2	43.6	1.81
	tet4_starMatrix (O5)	68.6	41.4	2.04

III. APPLICATION PERFORMANCE AND SCALABILITY

In order to demonstrate end-to-end performance of our solver EDGE, we used the following seismic setup: layer over halfspace 3 benchmark using order five in space and time [2]. We used three relaxation mechanisms (viscoelasticity) and local time stepping. The problem-adapted unstructured mesh has a total of 743,066 tetrahedrons.

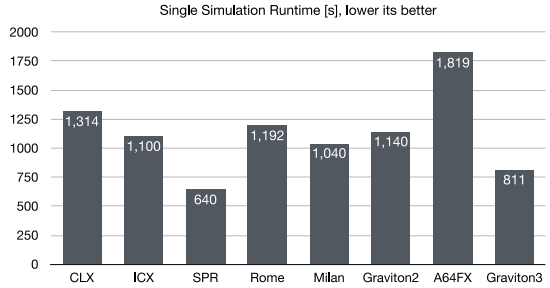

Fig. 1. Application performance of single forward simulations.

In Fig. 1 we observe that the solver's simulation runtime correlates mostly with the peak performance numbers discussed above, validating the performance portability of our approach. In this case performance portability means that we achieve the same fraction of theoretical peak performance when only executing FMAs out of registers, except for A64FX. When running fused simulations (Fig. 2) we obtain an additional per-simulation speed-up of 1.6x - 2.2x, while maintaining the same performance characteristics with respect to floating point peak. Compared to Tab. I we see a slight drop in speed-ups offered through the exploitation of sparsity on hot caches. Our end-to-end runs also employ kernels with other matrix patterns and we also stream through memory instead of staying local in caches.

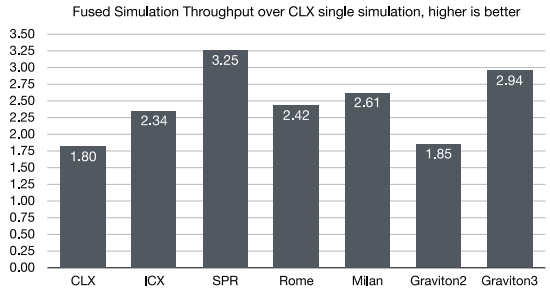

Fig. 2. Application performance of fused forward simulations.

IV. CONCLUSIONS

In this poster we demonstrated that Tensor Processing Primitives (TPP) enable highest performance in the computational sciences independent of the used instruction set architecture or processor. We showcased performance portability (what means in this context achieving a similar fraction of peak computational throughput in terms of FP32 FMAs) of LIBXSMM's TPP backend for key kernels of our solver EDGE on recent microarchitectures: Intel Cascade Lake, Intel Ice Lake, Intel Sapphire Rapids, AMD Zen 3, A64FX, Neoverse N1, Neoverse V1. Additionally, excellent scaling to thousands of nodes is not compromised when ensuring a fast single node execution (this data is only shown on the poster, but not in this abstract). For workloads and configurations visit https://short.dial3343.org/sc22. Results may vary.

REFERENCES

[1] Georganas, Evangelos and Kalamkar, Dhiraj and Avancha, Sasikanth and Adelman, Menachem and Anderson, Cristina and Breuer, Alexander and Bruestle, Jeremy and Chaudhary, Narendra and Kundu, Abhisek and Kutnick, Denise and Laub, Frank and Md, Vasimuddin and Misra, Sanchit and Mohanty, Ramanarayan and Pabst, Hans and Ziv, Barukh and Heinecke, Alexander, "Tensor Processing Primitives: A Programming Abstraction for Efficiency and Portability in Deep Learning Workloads" Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2021.

[2] Breuer, Alexander and Heinecke, Alexander, "Next-Generation Local Time Stepping for the ADER-DG Finite Element Method", url = https://arxiv.org/abs/2202.10313