Towards Lattice QCD on Fugaku: SVE Compiler Studies and Micro-Benchmarks in the RIKEN Fugaku Processor Simulator

Nils Meyer
Tilo Wettig
nils.meyer@ur.de
tilo.wettig@ur.de
University of Regensburg
Regensburg, Germany

Yuetsu Kodama
Mitsuhisa Sato
yuetsu.kodama@riken.jp
msato@riken.jp
RIKEN R-CCS
Kobe, Japan

ABSTRACT

The Fugaku supercomputer, successor to the Japanese flagship K-Computer, will start operation in 2021. Fugaku incorporates the Fujitsu A64FX processor, which is the first hardware implementation supporting the Arm SVE instruction set, in this case a 512-bit version. Real hardware is not accessible today, but RIKEN has designed a simulator of the A64FX. We present micro-benchmarks relevant for Lattice QCD obtained in the RIKEN Fugaku processor simulator and compare three different SVE compilers.

KEYWORDS

Lattice QCD, SVE, Fugaku, Post-K, Fujitsu A64FX, gem5

1 INTRODUCTION

In 2016 Arm announced a novel instruction set called Scalable Vector Extension (SVE) with operands of length 128 to 2048 bits. In the QPACE 4 project we pursue evaluation and enhancement of existing SVE software and upcoming SVE hardware [3, 4], with particular emphasis on Lattice QCD.

The Fujitsu A64FX processor is the first hardware implementation to support SVE. It contains 4 Core Memory Groups (CMG) [5, 6]. Each CMG comprises 12 compute cores and 1 assistant core, 64 kiB private L1 data cache, 8 MiB shared L2 cache and 8 GiB HBM2 memory. Each core can load two 512-bit vector operands from L1 and store one 512-bit vector operand to L1 per cycle. Load and store are mutually exclusive in the same cycle. Throughput of the L2 cache is half of the throughput of the L1 cache, i.e., one 512-bit data for L1 read from L2 and one 256-bit data L1 write back to L2 per cycle. Details of the cache characteristics, as well as the clock frequency, have not been disclosed yet.

For this study we assume that one A64FX core has 2 load ports and 1 store port. The peak compute performance is 2 fused-multiply-add (fma) instructions per cycle = 32 Flops/cycle in double precision.

Real hardware is not publicly available at present. RIKEN has designed a simulator of the A64FX on the basis of the gem5 processor simulator [2]. It is capable of simulating one CMG with cycle accuracy. It was validated against a test chip of the A64FX, which we do not have access to.

2 BENCHMARK DETAILS

Micro-benchmarks are useful since they help us to identify potential bottlenecks and to guide code optimization strategies. We use the Schönauer vector triad, see Fig. 1, for 64-bit `double` and 128-bit

```
for(i=0; i<R; i++)           // R = number of repetitions
    for(j=0; j<N; j+=ILOOP) // vector triad kernel
        for(k=j; k<j+ILOOP; k++)
            A[k] = B[k] + C[k] * D[k];
```

Figure 1: Schönauer vector triad. We allocate `A`, `B`, `C` and `D` as contiguous streams in virtual memory on the heap. Each stream is aligned to 128 bytes and consists of `N` elements. Nesting of the triad kernel loop is useful to test auto-vectorization and to enable the use of SVE ACLE intrinsics.

`double _Complex` data types. The complex version is performance-relevant for Lattice QCD as it appears in the main QCD kernel, i.e., the application of the Dirac operator on a spinor. The C11 standard defines the `_Complex` data type as 2-element structures of real and imaginary part, and also defines multiplication and addition.

Each triad computes 1 addition and 1 multiplication (real numbers), or 4 additions and 4 multiplications (complex numbers). The triads are computed in parallel using vector instructions. Loading the data takes 3 load instructions, and storing the data takes 1 store instruction.

We restrict ourselves to evaluation of a single core accessing L1 and L2 caches, but not main memory. As discussed in [2], we would need at least 6 cores to accurately simulate main memory access. Also, due to differences of the underlying model and the A64FX, it is not clear how accurately the simulator models multi-core accesses to the L2 cache.

We use the closed-source compilers armclang 19.2, Arm gcc 8.2.0 and Fujitsu fccpx 4.0.0, see A.1. armclang is not aware of the hardware vector length and follows the vector-length agnostic (VLA) paradigm. Arm gcc and Fujitsu fccpx also support fixed-size 512-bit SVE. In armclang we use SVE ACLE intrinsics [1] to imitate fixed-size SVE.

3 REAL NUMBERS

We show results of the vector triad for real numbers (`double`) in Fig. 2. Eight triads are processed in parallel. armclang auto-vectorization generates a VLA loop, see A.2. The loop body contains 3 load (`ld1d`), 1 fma and 1 store (`st1d`) instructions. Assuming that the A64FX has 2 load ports and 1 store port, 3 loads and 1 store can be issued in 3 cycles. Thus, the usage of the load ports is not optimal.

Arm gcc with fixed-size 512-bit SVE results in a loop body that is twofold unrolled, see A.2. It contains 6 load, 2 fma and 2 store

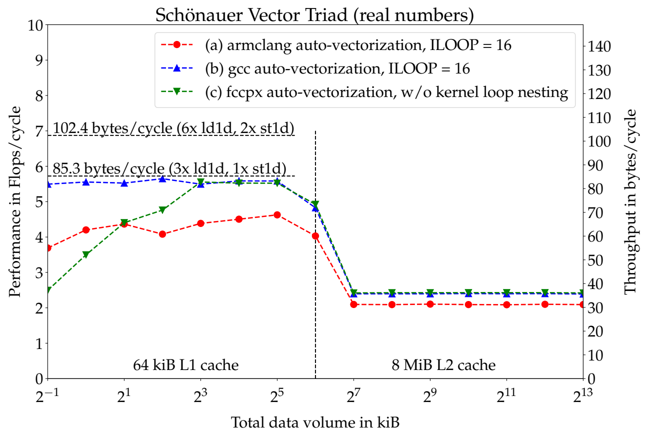

Figure 2: Vector triad using real numbers (DP).

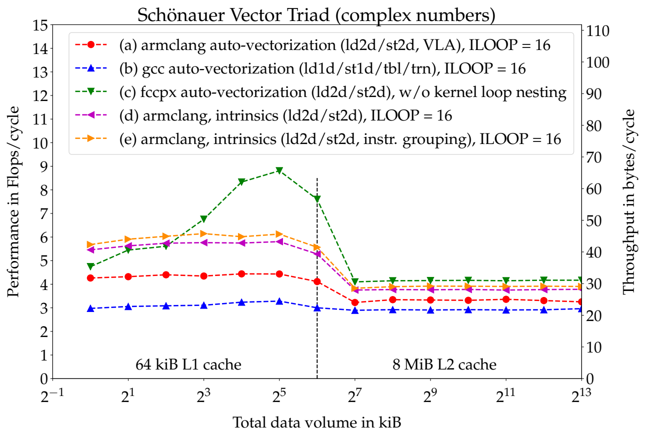

Figure 3: Vector triad using complex numbers (DP).

instructions. Again assuming that we have 2 load ports and 1 store port, 6 loads and 2 stores can be issued in 5 cycles. Thus, the usage of the load ports is optimal.

For fccpx we have to remove loop nesting of the triad kernel, otherwise fccpx generates scalar instructions. fccpx with fixed-size 512-bit SVE then generates two- and eightfold loop unrolling, see A.2. The results are comparable to gcc, but only if the dimension of the streams is sufficiently large.

None of the binaries achieves the peak throughput of the corresponding load/store scheme, which is 85.3 bytes/cycle for partial use of load/store units (armclang) and 102.4 bytes/cycle for optimal use (gcc, fccpx).

4 COMPLEX NUMBERS

For processing complex numbers 3 vectorized implementation schemes are feasible: (i) ordinary load/store (`ld1d`/`st1d`, 4 complex numbers) and permutation of real and imaginary parts prior to processing, (ii) structure load/store (`ld2d`/`st2d`, 8 complex numbers) separate/reassemble real and imaginary parts into/from separate registers, and (iii) ordinary load/store (`ld1d`/`st1d`, 4 complex numbers) and use of the `FCMLA` instructions, enabling hardware support for complex multiply-add [1, 3]. The latter does not apply, because gem5 does not support the `FCMLA` instruction.

We show results for complex numbers (`double _Complex`) in Fig. 3 and list the relevant assembly in A.3. Auto-vectorization with armclang and fccpx yields option (ii) (3a, 3c). Auto-vectorization with Arm gcc yields option (i) (3b).

Intrinsics versions with twofold loop unrolling were also tested (3d, 3e). The difference between the intrinsics versions is the instruction scheduling. In (3e) we hide the instruction latencies by grouping of loads, processing and stores, which is beneficial for performance.

If the data fit in L1, the intrinsics versions outperform auto-vectorized code of armclang and gcc. For armclang this can be explained by the missing loop unrolling, while for gcc it can be explained by the permutations generated by the compiler, see A.3. fccpx outperforms all other options due to the 6-fold loop unrolling if the data volume is not too small and due to latency hiding by instruction scheduling, see A.3.

In principle, the peak throughput of 102.4 bytes/cycle should be possible. We attribute the lower performance to the (unknown) details of the A64FX simulator.

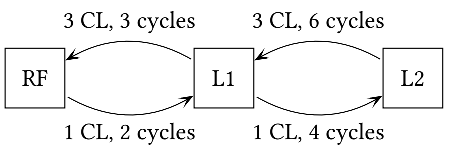

Figure 4: L2 cache access model assuming L1 cache misses for all loads and stores. Cache line (CL) transfers between the register file (RF), L1 and L2 caches proceed sequentially.

5 L2 CACHE ACCESS MODEL

For a better understanding of the performance characteristics with data in L2 we apply a single-port L1 cache model,

- load/store of the register file (RF) from/to L1 is mutually exclusive,
- read/write back of L1 from/to L2 is mutually exclusive,
- all data transfer between RF, L1 and L2 proceeds sequentially.

We restrict ourselves to evaluate access to full cache lines and assume that none of the data are available in L1, resulting in L1 cache misses for all loads and stores. Using the model, 6 ordinary loads (`ld1d`) of 64 bytes each (3 cache lines) from L2 via L1 to RF take 6 + 3 cycles, 2 ordinary stores (`st1d`, 1 cache line) from RF to L1 take 2 cycles, and eviction to L2 takes 4 cycles, see also Fig. 4. The peak throughput is $8 \cdot 64$ bytes/15 cycles = 34.1 bytes/cycle, which we also observe for `ld2d`/`st2d` schemes in good approximation.

Suboptimal usage of processor resources results in performance degradation. More information about the micro-architecture is necessary for a quantitative description.

6 FUTURE WORK

In future work we will also investigate software prefetching and multi-core benchmarks accessing the HBM2 main memory.

7 DISCLAIMER

The result of the RIKEN Post-K processor simulator [2] is just an estimated value, and it does not guarantee the performance of the supercomputer Fugaku at the start of its operation. We use the processor simulator compiled on 11th of September 2019. The Fujitsu fccpx compiler is a pre-release version under development.

8 ACKNOWLEDGMENTS

We acknowledge support from the HPC tools team at Arm.

REFERENCES

[1] Arm. 2017. ARM C Language Extension for SVE. [https://developer.arm.com/docs/100987/latest/arm-c-language-extensions-for-sve].

[2] Yuetsu Kodama, Tetsuya Odajima, Akira Asato, and Mitsuhisa Sato. 2019. Evaluation of the RIKEN Post-K Processor Simulator. (2019), 6. arXiv:1904.06451

[3] Nils Meyer, Peter Georg, Dirk Pleiter, Stefan Solbrig, and Tilo Wettig. 2018. SVE-enabling Lattice QCD Codes,. *2018 IEEE International Conference on Cluster Computing (CLUSTER)* (2018), 623. https://doi.org/10.1109/CLUSTER.2018.00079 arXiv:1901.07294

[4] Nils Meyer, Dirk Pleiter, Stefan Solbrig, and Tilo Wettig. 2019. Lattice QCD on upcoming Arm architectures. *Proceedings of LATTICE 18* (2019), 316. arXiv:1904.03927

[5] Toshiyuki Shimizu. 2018. Post-K Supercomputer with Fujitsu's Original CPU, A64FX Powered by Arm ISA. [https://www.fujitsu.com/global/Images/post-k_supercomputer_with_fujitsu%27s_original_cpu_a64fx_powered_by_arm_isa.pdf].

[6] Toshio Yoshida. 2018. Fujitsu High Performance CPU for the Post-K Computer. Hot Chips 2018 [http://www.fujitsu.com/jp/Images/20180821hotchips30.pdf].