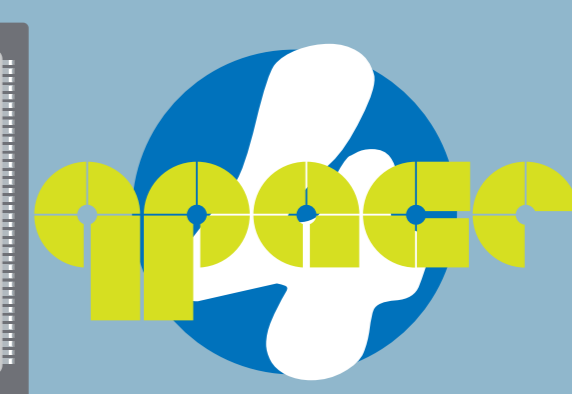
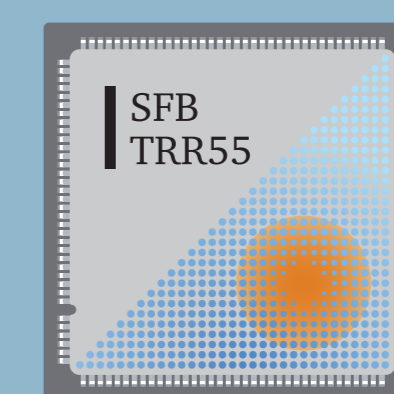


# Lattice QCD on upcoming ARM architectures

Nils Meyer\*, Dirk Pleiter\*†, Stefan Solbrig\*, Tilo Wettig\*

\*Department of Physics, University of Regensburg, Germany

†Forschungszentrum Jülich, Germany



## QPACE 4 project

- ▶ Aims to design an ARM SVE environment optimized for Lattice QCD
  - ▷ Prototype at the University of Regensburg, Germany, by 2020
- ▶ Pursues evaluation and enhancement of existing SVE software toolchain and upcoming SVE hardware technologies for Lattice QCD applications
- ▶ Enables Lattice QCD code optimized for SVE architectures, e.g., the Japanese flagship supercomputer "Post-K" announced for 2021

## ARM SVE

- ▶ The ARM Scalable Vector Extension (SVE) targets the HPC market [1]
- ▶ Key features of SVE hardware
  - ▷ Wide vector units, ranging from 128 bit to 2048 bit
  - ▷ Vectorized native 16 bit floating point operations, including arithmetics
  - ▷ Vectorized arithmetics of complex numbers
  - ▷ The silicon provider chooses the vector register length and defines the performance characteristics of the SVE hardware
  - ▷ The first SVE hardware officially announced is the Post-K, which comprises 512 bit wide vector units, 48 compute cores per node and high performance stacked memory [2]
- ▶ Key features of the SVE instruction set architecture (ISA)
  - ▷ SVE follows a vector-length agnostic (VLA) programming model that adapts itself to the available vector length (VL)
  - ▷ VLA predication allows for selection of vector elements to be used for processing, which enables, e.g., complex control flows within loops
  - ▷ Support for structure load/store, e.g., load of an array of two-element structures into two vectors, with one vector per structure element
  - ▷ The ARM C Language Extensions for SVE (ACLE) intrinsics provide convenient access to features of the SVE hardware in C/C++ [3]

## SVE code example

- ▶ daxpy in C99 ( $y_i \leftarrow y_i + a \times x_i$  with real operands)

```
void daxpy(double a, double *restrict x, double *restrict y, int n) {
    for (int i = 0; i < n; i++)
        y[i] = y[i] + a * x[i];
}
```
- ▶ Assembly output of the armclang 18.3 SVE compiler (auto-vectorization enabled)

```
whilelo p1.d, xzr, x8          // pred. for x_i, y_i; xzr=0; x8=n; z0=a
ptrue p0.d                    // pred. for result vector
.LBB0_2:
ld1d {z1.d}, p1/z, [x1, x9, lsl #3] // pred. load z1 <- y_{i..}
ld1d {z2.d}, p1/z, [x2, x9, lsl #3] // pred. load z2 <- x_{i..}
fmad z1.d, p0/m, z0.d, z2.d // pred. multiply-add
st1d {z1.d}, p1, [x2, x9, lsl #3] // pred. store z1 -> y_{i..}
incd x9 // increment x9
whilelo p1.d, x9, x8 // pred. remaining y_{i..}
b.mi .LBB0_2 // conditional branch to LBB0_2
```
- ▶ Discussion
  - ▷ armclang is unaware of the VL and optimizes for the VLA paradigm
  - ▷ The binary code executes for all VL implementations

## SVE code development tools

- ▶ Compilers
  - ▷ ARM provides us with their LLVM/clang-based armclang 18 compiler (evaluation version available [4])
  - ▷ RIKEN provides us with the Fujitsu SVE compiler
  - ▷ We test the compilers, stimulate bug fixes and propose improvements in the context of research contracts
- ▶ ARM Instruction Emulator (ArMIE)
  - ▷ ArMIE allows for functional verification of SVE binaries emulating the SVE ISA with user-defined VL (freely available [4])
  - ▷ ArMIE is designed on top of the open-source DynamoRIO toolset enabling code instrumentation at run-time [5]
  - ▷ We contribute to DynamoRIO enabling advanced SVE code analysis, e.g., instruction mixing and branching statistics ( $\rightarrow$  *Lattice 18 proceedings*)
- ▶ gem5
  - ▷ The gem5 simulator is a modular platform for computer architecture research, encompassing system-level and processor architecture [6]
  - ▷ RIKEN provides us with access to their gem5 simulator of the Post-K CPU
  - ▷ We optimize our code guided by simulations of the Post-K CPU (future work)

## "Grid" Lattice QCD framework

- ▶ Grid [7, 8] is a portable open-source Lattice QCD framework written in C++ 11, maintained by Peter Boyle (Edinburgh) and co-workers
- ▶ Targets massively parallel architectures supporting SIMD + OpenMP + MPI
- ▶ Implements more than 100 tests and benchmarks
- ▶ ISA-specific code is implemented using intrinsics and assembly

SIMD family	Register size
Intel SSE4	128 bit
Intel AVX/AVX2	256 bit
Intel ICMI, AVX512	512 bit
IBM QPX	256 bit
ARM Neon	128 bit
ARM SVE	variable
generic C	architecture independent, user-defined array size

## Enabling SVE in Grid


- ▶ We use ACLE for our SVE implementation to access the SVE features
- ▶ We minimize implications of the VLA programming model and bypass restrictions on usage of ACLE data types
  - ▷ We declare the vector length VL (in bytes) as a compile-time constant
  - ▷ Superfluous loops implied by VLA are omitted
  - ▷ The variety of predications is minimized to fit into the register file
  - ▷ The templated struct `acle<T>` (T = double, float) provides convenient access to ACLE definitions, e.g., predications and data types

- ▶ Code example: templated multiplication of two vectors of complex numbers

```
template<typename T>
struct vector {
    alignas(VL) T v[VL / sizeof(T)]; // C-array
};

struct MultComplex {
    template<typename T>
    inline vector<T> operator()(const vector<T> &a, const vector<T> &b) {
        vector<T> result; // result vector
        typename acle<T>::sve_t z_v, a_v, b_v, r_v; // ACLE data types
        svbool_t p; // predication

        p = acle<T>::activate_all_elements(); // define predication
        z_v = acle<T>::zero(); // z_v <- FP zero
        a_v = svld1(p, a.v); // load a_v <- a
        b_v = svld1(p, b.v); // load b_v <- b
        r_v = svcmla_z(p, z_v, a_v, b_v, 90); // complex multiply-add
        r_v = svcmla_z(p, r_v, a_v, b_v, 0); //
        svst1(p, result.v, r_v); // store r_v -> result
        return result; // return result
    }
};
```

- ▶ Software development, verification and optimization
  - ▷ We implement multiple coding schemes for SVE ( $\rightarrow$  *Lattice 18 proceedings*)
  - ▷ We verify selected tests and benchmarks emulating multiple VL in the ArMIE
  - ▷ At present one coding scheme fails to compile, and some tests give wrong results due to SVE toolchain immaturity
  - ▷ Run-time optimizations will be guided by gem5 CPU simulation (future work)
- ▶ Sources available at <https://github.com/nmeyer-ur/Grid/tree/feature/arm-sve> 

## Future perspectives

- ▶ Improved and more mature development environment
- ▶ Extended support of the SVE ISA by compilers, e.g., instructions for complex arithmetics and compiler-generated load/store using ACLE
- ▶ Possible endorsement of SVE data types in languages like C and C++
- ▶ We stimulate disclosure of details of ARM-based micro-architectures and ISA performance characteristics relevant for HPC code design

## References

- [1] N. Stephens et al., *IEEE Micro* 37 Issue 2 [arxiv:1803.06185]
- [2] <http://www.fujitsu.com/global/about/resources/news/press-releases/2018/0621-01.html>
- [3] "ARM C Language Extensions for SVE", ARM, *Tech. Rep.* (2017)
- [4] <https://developer.arm.com/products/software-development-tools/hpc/documentation>
- [5] <http://www.dynamorio.org/>
- [6] <http://gem5.org/>
- [7] P. Boyle et al., *Proceedings of LATTICE 15* (2016) [arxiv:1512.03487]
- [8] <https://github.com/paboyle/Grid>

