Grid Lattice QCD framework on A64FX

R-CCS seminar

Nils Meyer

June 30, 2021

University of Regensburg (Regensburg, Germany)

Outline

- QPACE 4
- Grid and SVE compilers
- Benchmarks on QPACE 4
- Alternative layout of complex numbers
- Summary and outlook

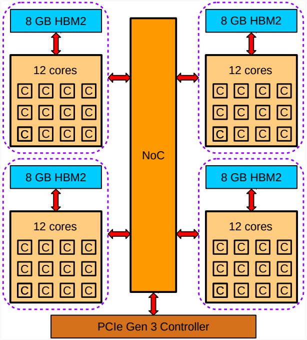
QPACE 4

- Latest member of QCD PArallel Compute Engine (QPACE) series
- 64 Fujitsu A64FX CPUs
  - Fujitsu PRIMEHPC FX700 [PrimeHPC]
  - Deployed June 2020 at Regensburg University
  - InfiniBand EDR interconnect (100 Gbit/s)
    - 2 partitions, 32 nodes each
- Open-source software stack
  - CentOS Stream 8, GCC 10.1, OpenMPI 4.0
  - GlusterFS parallel filesystem
  - Grid Lattice QCD framework [Boyle et al.:Latt15], [Boyle et al.:Grid]
  - Grid Python Toolkit (GPT) [Lehner et al.:21]

A64FX on QPACE 4

- Nominal frequency 1.8 GHz
- 4 Core Memory Groups
  - 12 cores each, no assistant core
  - 8 MB shared L2 cache
  - 8 GB HBM2 memory
- 1 PCIe Gen3 x16 interface (no Tofu)

- Features available on Fugaku, but currently not on QPACE 4
  - Sector cache support
  - Hardware barrier support
  - Energy savings options (eco modes)
  - Energy consumption measurement



SVE Programming Models

- Arm C Language Extensions (ACLE) for SVE (SVE ACLE) [Arm:ACLE]
  - `arm_sve.h` header file enables SVE ACLE in SVE compilers
- Vector-length agnostic (VLA) programming model
  - Official SVE programming model
  - Size of vector registers unknown at compile time
  - Restrictions on usage of SVE ACLE vectors include, but are not limited to*
    - `sizeof()` not applicable
    - Not allowed as member of unions, structures and classes
    - Not allowed as types of array elements and C++ STL containers like std::vector
- Vector-length specific (VLS) programming model
  - Not an official SVE programming model
  - Sized GNU vectors derived from sizeless SVE ACLE vector types (`typedef` declaration)
    - Restrictions (*) do not apply to sized vector
    - Compatible with SVE ACLE functions
  - First supported by GCC 10 (released May 2020)
  - Announced for LLVM 11 at ISC 2020, but not delivered
  - Supported as of clang/LLVM 12 (released April 2021)

Grid Lattice QCD Framework

- C++11, supports OpenMP and MPI
- Supports all x86 SIMD extensions, Arm NEONv8, A64FX, GPGPU
- Performance portability: C++ Templates + intrinsics (if available)
- Core of architecture abstraction layer is a template class with architecture-specific member data types
- Set of architecture-specific lower-level functions define actions at vector register level, $O(100)$ lines of code
  - Arithmetic of real and complex numbers
  - Intra- and inter-register permutations
  - Conversion of floating-point precision
- 100% SIMD efficiency
- Interleaved re/im layout of complex numbers (RIRI)

Grid Lattice QCD Framework

- Port to A64FX / 512-bit SVE (N. Meyer)
  - Lower-level functions implement Arm C Language Extensions (ACLE) for SVE
  - Hardware support processing complex numbers in RIRI layout (`svcmla`, `svcadd`)
  - Vector-length agnostic implementation (2018) [Meyer et al.:Latt18], [Meyer et al.:Cluster18]
  - Fixed vector length implementation (2020) [Meyer et al.:APLAT20]
  - Hand-optimized Wilson Dslash / Domain Wall kernels
- Progress since APLAT 2020 (N. Meyer et al.)
  - Performance improvement of Wilson Dslash and Domain Wall kernels (in upstream Grid develop since Dec. 2020)
  - Collaboration with University of Erlangen-Nürnberg (FAU), Germany
    - Domain Wall kernel performance study using alternative layout of complex numbers
  - Investigation of MPI performance issues started

SVE Compilers

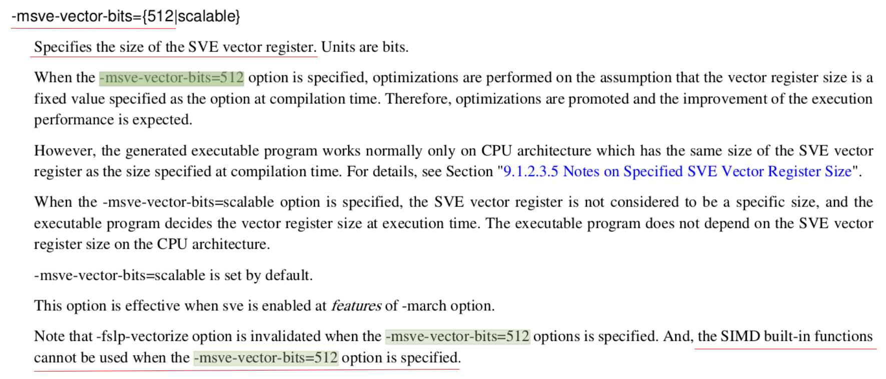
- SVE compiler capabilities (? = unknown/not tested)

Compiler	Target	VLA	VLS	Compiles Grid	Comment
	A64FX			VLA / VLS	
armclang 20.x	✓	✓	✗	✓/ –	Poor performance
armclang 21.0	✓	✓	?	✓/ ?	Poor performance
clang 11.0	✓	✓	✗	✓/ –	Poor performance
clang 12.0	✓	✓	✓	✓/ ✗	Poor performance, VLS compiles partially (linker error)
GCC 10.1	✗	✓	✓	✓/ ✓	**Good performance, default on QPACE 4**
GCC 10.2	✗	✓	✓	✓/ ✓	**Same performance as GCC 10.1**
GCC 11.1	✓	✓	✓	? / ✓	Poor performance
FCC 4.3.1 (clang mode)	✓	✓	✗	✓/ –	Poor performance, not available on QPACE 4
FCC 4.3.1 (trad mode)	✓	✓	✓	✗/ ✗	FCC 4.3.1 outdated
FCC 4.5.0 (clang mode)	✓	✓	✓✗	?	Not tested

- GCC 10.1 / 10.2
  - Deliver best performance
  - Lack of proper SVE instruction scheduler for A64FX
- clang/LLVM compilers
  - Performance issues resolving Grid's template constructions (root cause: overloaded assignment operator?)
  - VLS does not solve this issue
  - Remove clang/LLVM support from Grid's configure file and restrict to GCC 10.1 / 10.2?

FCC 4.5.0

- Technical Computing Suite V4.0L20, C User's Guide, p. 184



-msve-vector-bits={512|scalable}

Specifies the size of the SVE vector register. Units are bits.

When the -msve-vector-bits=512 option is specified, optimizations are performed on the assumption that the vector register size is a fixed value specified as the option at compilation time. Therefore, optimizations are promoted and the improvement of the execution performance is expected.

However, the generated executable program works normally only on CPU architecture which has the same size of the SVE vector register as the size specified at compilation time. For details, see Section "9.1.2.3.5 Notes on Specified SVE Vector Register Size".

When the -msve-vector-bits=scalable option is specified, the SVE vector register is not considered to be a specific size, and the executable program decides the vector register size at execution time. The executable program does not depend on the SVE vector register size on the CPU architecture.

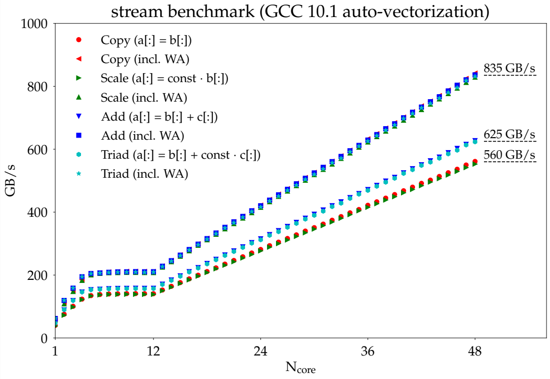
-msve-vector-bits=scalable is set by default.

This option is effective when sve is enabled at *features* of -march option.

Note that -fslp-vectorize option is invalidated when the -msve-vector-bits=512 options is specified. And, the SIMD built-in functions cannot be used when the -msve-vector-bits=512 option is specified.

- Need `-msve-vector-bits=512` *and* SIMD built-in functions (intrinsics)

Main Memory Throughput

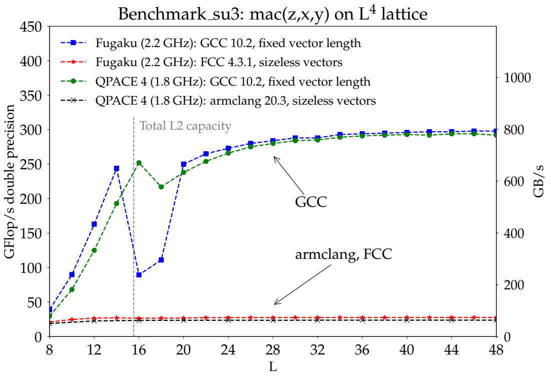


- $N_{\text{core}} > 12$: data throughput scales with number of cores in use
  - Triad: 841 GB/s on Fugaku [Alappat et al.:PMBS21]
- Caches implement write-back policy
  - Write Allocation (WA): cache block (256 bytes) load from memory on write miss
    - Reduces effective memory throughput
  - WA can be mitigated by `DC ZVA` instruction (not beneficial in Grid → not used in Grid)

SU(3) Matrix Multiplication

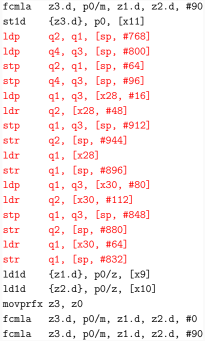
- Independent SU(3) matrix multiplication $z = x \times y$ on each lattice site in 4d volume
- Templates + intrinsics, `svcmla` for complex multiply-add

GCC	throughput comparable to stream triad for large volume
armclang & FCC	excessive copy operations limit performance



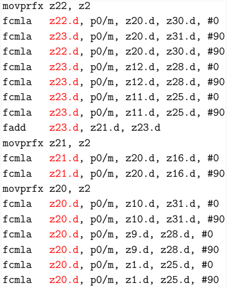
SU(3) Matrix Multiplication Assembly

- clang/LLVM (also FCC 4.3.1)
  - Excessive copy operations, not in high-level code
  - Every call to Grid's lower-level functions is affected
  - Example: armclang 20.3, VLA (clang 12, VLS even worse)



```
fcmla   z3.d, p0/m, z1.d, z2.d, #90
st1d    {z3.d}, p0, [x11]
ldp     q2, q1, [sp, #768]
ldp     q4, q3, [sp, #800]
stp     q2, q1, [sp, #64]
stp     q4, q3, [sp, #96]
ldp     q1, q3, [x28, #16]
ldr     q2, [x28, #48]
stp     q1, q3, [sp, #912]
str     q2, [sp, #944]
ldr     q1, [x28]
str     q1, [sp, #896]
ldp     q1, q3, [x30, #80]
ldr     q2, [x30, #112]
stp     q1, q3, [sp, #848]
str     q2, [sp, #880]
ldr     q1, [x30, #64]
str     q1, [sp, #832]
ld1d    {z1.d}, p0/z, [x9]
ld1d    {z2.d}, p0/z, [x10]
movprfx z3, z0
fcmla   z3.d, p0/m, z1.d, z2.d, #0
fcmla   z3.d, p0/m, z1.d, z2.d, #90
```

- GCC 10.1 / 10.2
  - Tendency to maximize dependencies, irrespective of implementation in high-level code
  - Potentially harmful on A64FX due to high instruction latencies and limited out-of-order execution capabilities
  - Example: GCC 10.1, VLS



```
movprfx z22, z2
fcmla   z22.d, p0/m, z20.d, z30.d, #0
fcmla   z23.d, p0/m, z20.d, z31.d, #90
fcmla   z22.d, p0/m, z20.d, z30.d, #90
fcmla   z23.d, p0/m, z12.d, z28.d, #0
fcmla   z23.d, p0/m, z12.d, z28.d, #90
fcmla   z23.d, p0/m, z11.d, z25.d, #0
fcmla   z23.d, p0/m, z11.d, z25.d, #90
fadd    z23.d, z21.d, z23.d
movprfx z21, z2
fcmla   z21.d, p0/m, z20.d, z16.d, #0
fcmla   z21.d, p0/m, z20.d, z16.d, #90
movprfx z20, z2
fcmla   z20.d, p0/m, z10.d, z31.d, #0
fcmla   z20.d, p0/m, z10.d, z31.d, #90
fcmla   z20.d, p0/m, z9.d, z28.d, #0
fcmla   z20.d, p0/m, z9.d, z28.d, #90
fcmla   z20.d, p0/m, z1.d, z25.d, #0
fcmla   z20.d, p0/m, z1.d, z25.d, #90
```

Wilson Dslash Kernel

- 9-point stencil in 4d spacetime, complex internal structure
- Specialization for A64FX (`--dslash-asm`): manual scheduling and prefetching using ACLE
- Single-node performance in GFlop/s (normalization to 1320 Flop per site)[1]

```
$ Benchmark_wilson --grid *.*.*.* --mpi 1.1.1.1|4 --comms-sequential --dslash-asm
```

	GCC 10.1				GCC 10.1			
	APLAT 2020 presentation				Upstream Grid			
Volume	1 MPI rank		4 MPI ranks		1 MPI rank		4 MPI ranks	
	DP	SP	DP	SP	DP	SP	DP	SP
$16^3 \times 32$	336	635	275	490	360	893	321	619
$24^3 \times 48$	351	711	312	620	350	990	383	827
$32^3 \times 64$	344	694	317	633	339	1023	383	898

- Upstream Grid
  - Performance improvement up to $\sim$ 47% compared to APLAT 2020 presentation
  - Performance gain or penalty using 4 MPI ranks
  - GCC 10.1 outperforms other compilers, SP breaks TFlop/s barrier

[1] DP = double precision, SP = single precision

Wilson Dslash Kernel

- Single-node performance comparison: KNL 7230 vs. A64FX / QPACE 4[2]

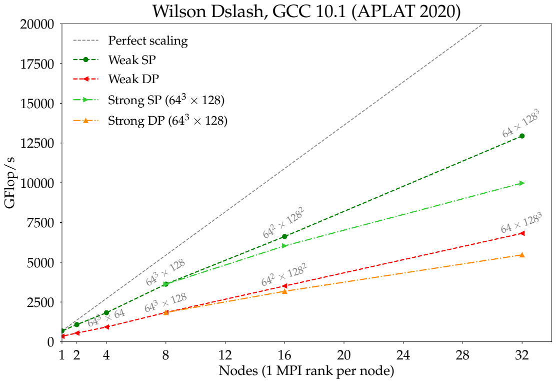
```
$ Benchmark_wilson --grid 32.32.32.32 --mpi *.*.*.* --comms-sequential --dslash-asm
```

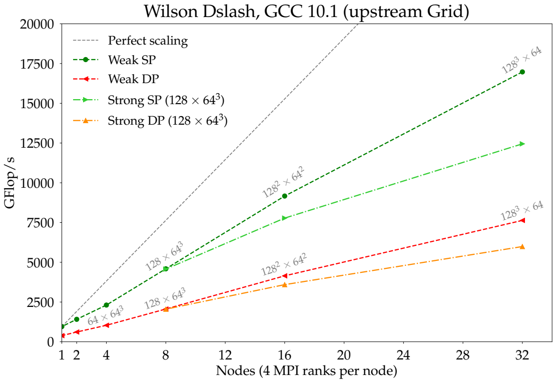
MPI ranks	--mpi	KNL GFlop/s DP	KNL GFlop/s SP	A64FX GFlop/s DP	A64FX GFlop/s SP
1	1.1.1.1	**170**	300	322	**1015**
4	4.1.1.1	162	**509**	345	882
4	1.4.1.1	164	402	**376**	821
4	1.1.4.1	153	387	344	824
4	1.1.1.4	149	339	360	826
4	2.2.1.1	162	445	356	839
4	2.1.2.1	152	442	342	843
4	2.1.1.2	151	403	352	847
4	1.2.2.1	155	389	357	820
4	1.2.1.2	147	349	368	821
4	1.1.2.2	145	347	349	823

- KNL
  - DP: 1 MPI rank performs better than 4 ranks
  - SP: 4 MPI ranks performs better than 1 rank
- A64FX
  - DP: 4 MPI ranks perform better than 1 rank
  - SP: 1 MPI rank performs better than 4 ranks

[2] clang 9.0.1 on KNL, GCC 10.1 on QPACE 4

Wilson Dslash Kernel MPI Scaling





- Marginal difference between 1 and 4 MPI ranks per node ($\sim$ 3 %, not shown here)[3]
- 1 node → 2 nodes performance increase: up to $\sim$ 48% SP ($\sim$ 59% DP)
- Wider performance gap between weak and strong scaling compared to APLAT 2020 presentation

[3] 4 MPI ranks per node motivated by GPT

Domain Wall Kernel

- Performance-relevant part of Domain Wall Dirac operator
  - Wilson Dslash kernel (slide 13), 4d gauge and 5d fermion fields
  - Gauge field reuse traversing 5-direction in innermost loop
- Single-node performance in GFlop/s (normalization to 1320 Flop per site)

```
$ Benchmark_dwf --grid *.*.*.* -Ls * --mpi 1.1.1.1|4 --comms-sequential --dslash-asm
```

	GCC 10.1				GCC 10.1			
	APLAT 2020 presentation				Upstream Grid			
Volume	1 MPI rank		4 MPI ranks		1 MPI rank		4 MPI ranks	
	DP	SP	DP	SP	DP	SP	DP	SP
$16^3 \times 32 \times 8$	362	712	320	627	484	949	408	803
$16^3 \times 32 \times 16$	363	724	323	642	477	958	409	815
$24^3 \times 48 \times 8$	359	723	314	673	476	960	432	875

- Upstream Grid
  - Performance improvement of up to $\sim 33\%$ compared to APLAT 2020 presentation
  - Performance penalty using 4 MPI ranks
  - GCC 10.1 outperforms outperforms other compilers, SP almost breaks TFlop/s barrier

Domain Wall Kernel

- Single-node performance comparison: KNL 7230 vs. A64FX / QPACE 4[4]

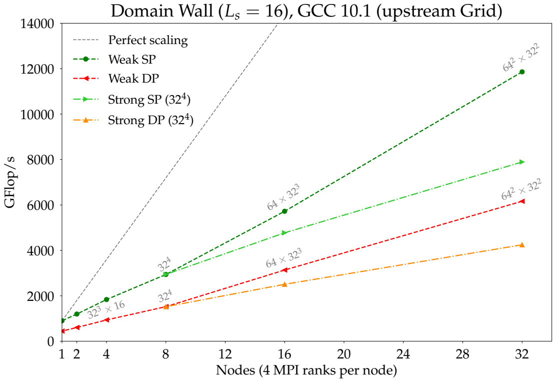
```
$ Benchmark_dwf --grid 16.16.16.16 -Ls 8 --mpi *.*.*.* --comms-sequential --dslash-asm
```

MPI ranks	`--mpi`	KNL GFlop/s DP	KNL GFlop/s SP	A64FX GFlop/s DP	A64FX GFlop/s SP
1	1.1.1.1	151	**324**	**475**	**913**
4	4.1.1.1	**184**	336	417	752
4	1.4.1.1	173	284	429	674
4	1.1.4.1	159	314	353	681
4	1.1.1.4	162	307	349	677
4	2.2.1.1	183	315	419	717
4	2.1.2.1	173	331	378	718
4	2.1.1.2	176	315	377	719
4	1.2.2.1	162	290	383	662
4	1.2.1.2	160	278	384	664
4	1.1.2.2	158	298	345	663

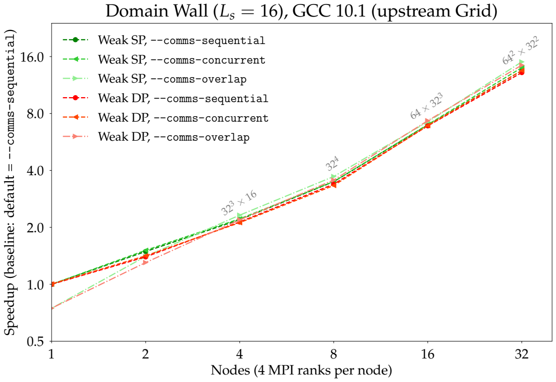
- KNL
  - DP: 4 MPI ranks perform better than 1 rank
  - SP: 1 MPI rank performs better than 4 ranks (exceptions: `--mpi 4.1.1.1` and `2.1.2.1`)
- A64FX
  - DP: 1 MPI rank performs better than 4 ranks
  - SP: 1 MPI rank performs better than 4 ranks

[4] clang 9.0.1 on KNL, GCC 10.1 on QPACE 4

Domain Wall Kernel MPI Scaling



Domain Wall ($L_s = 16$), GCC 10.1 (upstream Grid)



Domain Wall ($L_s = 16$), GCC 10.1 (upstream Grid)

- Worse scaling than Wilson Dslash
- Marginal difference between 1 and 4 MPI ranks per node ($\sim$ 3 %, not shown here)[5]
- 1 node → 2 nodes performance increase: only up to $\sim$ 36% SP ($\sim$ 33% DP)
- `--comms-overlap` under- and slightly outperforms other schemes

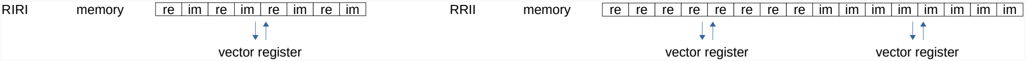
[5] 4 MPI ranks per node motivated by GPT

MPI Performance Issues

- Non-trivial code path
  - 3 communication schemes (`--comms-{sequential, concurrent, overlap}`)
  - Partitioning (`--mpi`)
  - “Random” memory access
  - Multiple template function calls
  - Many branches
  - On-chip communication
    1. Inter-process communication using shared memory (this presentation: `shmget` implementation)
    2. Message passing using MPI (not shown here)
  - Off-chip communication: MPI
- Early stage of investigation
  - Optimization attempts so far
    - Pragma-based loop unrolling → GCC crash
    - Manual loop unrolling → no improvement or performance penalty
    - Software prefetching → no improvement or performance penalty
    - Gather loads in selected lower-level functions → performance penalty
  - Code instrumentation needed (future work)

Domain Wall Kernel Performance Study

- Collaboration with Erlangen-Nürnberg University (FAU), Germany
  - Performance study of Domain Wall kernel on Fugaku
    - Interleaved re/im layout (RIRI): 1 vector register holds multiple re/im *pairs*
    - Split re/im layout (RRII): 1 vector register holds multiple re *or* im
  - To be published in *Concurrency and Computation: Practice and Experience* (PMBS special edition)
  - Preprint available [Alappat et al.:PMBS21]
  - GridBench (P. Boyle et al., [Boyle et al.:GridBench])
    - Features subset of Grid
    - Domain Wall kernel structure similar to Grid (but no MPI), supports RIRI and RRII layout
    - Data sets generated by Grid (RIRI), re-arranged to RRII by GridBench
    - Specialization for A64FX (RIRI + RRII) available (N. Meyer and C. Alappat, [Meyer et al.:GridBench])
    - Only DP available



RIRI
memory
re | im | re | im | re | im | re | im
vector register
RRII
memory
re | re | re | re | re | re | re | re | im | im | im | im | im | im | im | im
vector register
vector register

Domain Wall Kernel

- Performance-relevant part of Domain Wall Dirac operator

$$\psi'(n,s)_{\alpha a} = (D\psi)(n,s)_{\alpha a} = \sum_{\mu=1}^{4}\sum_{\beta=1}^{4}\sum_{b=1}^{3}\left\{U_\mu(n)_{ab}(1+\gamma_\mu)_{\alpha\beta}\psi(n+\hat{\mu},s)_{\beta b} + U_\mu^\dagger(n-\hat{\mu})_{ab}(1-\gamma_\mu)_{\alpha\beta}\psi(n-\hat{\mu},s)_{\beta b}\right\}$$

- Roofline model: arithmetic intensity AI($V = 24^4 \times 8$) = 0.88 Flop/byte (DP, memory traffic measurement)
  - Estimated kernel peak performance = min(peak FP (DP), AI · saturated memory throughput) ≈ 753 GFlop/s
- Simplified implementation scheme neglecting, e.g., periodic boundary conditions (applies to RIRI + RRII)

```
#define x_p 1 // x−plus  direction
#define x_m 2 // x−minus direction
#define y_p 3 // y−plus  direction
...
#pragma omp parallel for schedule(static)
for {t,z,y,x} = 1:{L_t−2,L_z−2,L_y−2,L_x−2} // collapsed loop over 4d space−time
{
    for(int s=0; s<L_s; ++s) // loop over 5th dimension
    {
        O[t][z][y][x][s] = R(x_p) · U[x_p][t][z][y][x] · P(x_p) · I[t][z][y][x+1][s] +
                           R(x_m) · U[x_m][t][z][y][x] · P(x_m) · I[t][z][y][x−1][s] +
                           R(y_p) · U[y_p][t][z][y][x] · P(y_p) · I[t][z][y+1][x][s] +
                           R(y_m) · U[y_m][t][z][y][x] · P(y_m) · I[t][z][y−1][x][s] +
                           R(z_p) · U[z_p][t][z][y][x] · P(z_p) · I[t][z+1][y][x][s] +
                           R(z_m) · U[z_m][t][z][y][x] · P(z_m) · I[t][z−1][y][x][s] +
                           R(t_p) · U[t_p][t][z][y][x] · P(t_p) · I[t+1][z][y][x][s] +
                           R(t_m) · U[t_m][t][z][y][x] · P(t_m) · I[t−1][z][y][x][s];
    }
}
```

Domain Wall Kernel

- Data layout characteristics

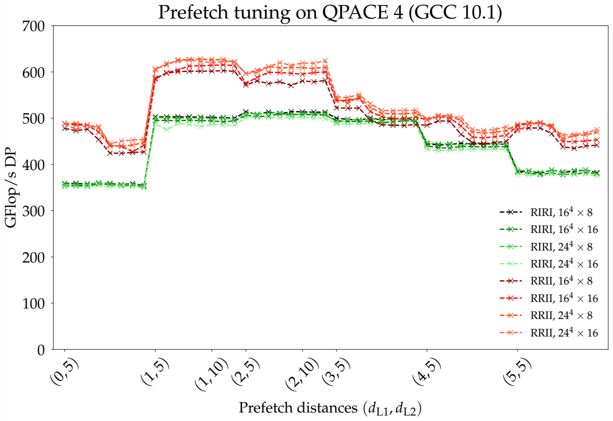
	RIRI	RRII
Parallel lattice site updates	4 DP, 8 SP	8 DP, 16 SP
Architecture vector registers	32 registers sufficient	32 registers not sufficient → spilling
Projection (P) to half spinor	svcadd ($z_1 \pm iz_2$)	Resolved in arithmetics (for free)
Reconstruction (R) of full spinor	svcadd ($z_1 \pm iz_2$)	Resolved in arithmetics (for free)
SU(3) × half spinor	svcmla ($z_1 + z_2 \times z_3$)	Real arithmetics incl. FMA
Real Flop per lattice site update	1440 (need $0 + z_2 \times z_3$)	1320
FLA / FLB pipeline usage	Imbalanced	Balanced

- Simplified implementation scheme neglecting, e.g., periodic boundary conditions (applies to RIRI + RRII)

```
#define x_p 1 // x-plus  direction
#define x_m 2 // x-minus direction
#define y_p 3 // y-plus  direction
...
#pragma omp parallel for schedule(static)
for {t,z,y,x} = 1:{L_t-2,L_z-2,L_y-2,L_x-2} // collapsed loop over 4d space-time
{
    for(int s=0; s<L_s; ++s) // loop over 5th dimension
    {
        O[t][z][y][x][s] = R(x_p) · U[x_p][t][z][y][x] · P(x_p) · I[t][z][y][x+1][s] +
                           R(x_m) · U[x_m][t][z][y][x] · P(x_m) · I[t][z][y][x-1][s] +
                           R(y_p) · U[y_p][t][z][y][x] · P(y_p) · I[t][z][y+1][x][s] +
                           R(y_m) · U[y_m][t][z][y][x] · P(y_m) · I[t][z][y-1][x][s] +
                           R(z_p) · U[z_p][t][z][y][x] · P(z_p) · I[t][z+1][y][x][s] +
                           R(z_m) · U[z_m][t][z][y][x] · P(z_m) · I[t][z-1][y][x][s] +
                           R(t_p) · U[t_p][t][z][y][x] · P(t_p) · I[t+1][z][y][x][s] +
                           R(t_m) · U[t_m][t][z][y][x] · P(t_m) · I[t-1][z][y][x][s];
    }
}
```

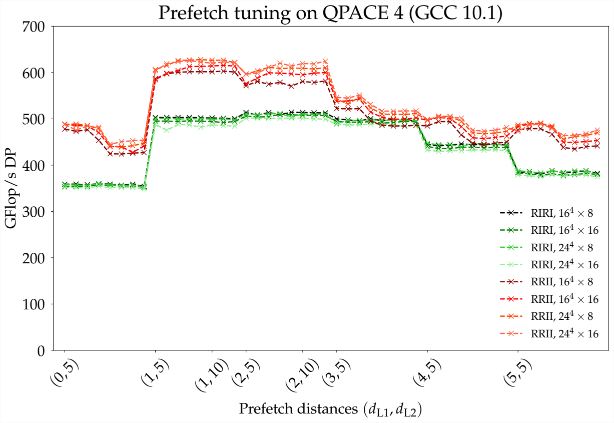
Software Prefetching

- No benefit of software prefetching of gauge field (not shown here)
  → hardware prefetching works correctly
- 2-pass software prefetching of fermion field beneficial
  - Prefetch L2 ← memory
    (d_{L2} directions ahead, $d_{L2} \in [5, 12]$ shown)
  - Prefetch L1 ← L2
    (d_{L1} directions ahead, $d_{L1} \in [0, 5]$ shown)
- Strong impact of L1 ← L2 prefetching
  - First plateau corresponds to $d_{L1} = 0$
  - Second plateau corresponds to $d_{L1} = 1$
  - etc.



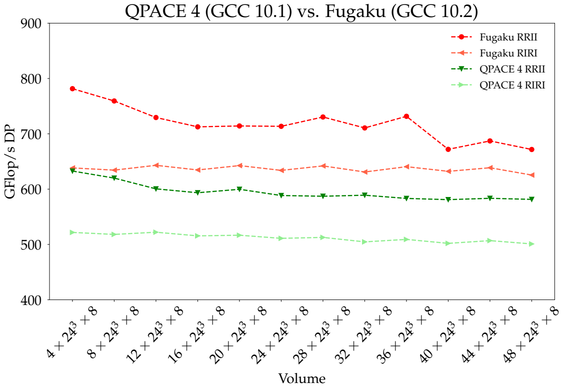
Software Prefetching

- RIRI layout
  - L2 ← memory prefetching performance mostly stable over range of d_{L2} (intra-plateau)
  - Sweet spot: $d_{\text{L1}} = 2$, $d_{\text{L2}} \approx 10$
  - Performance: $\sim$ 500 GFlop/s ($\sim$ 620 GB/s)
- RRII layout
  - Wavy patterns: likely due to unfavorable combination of d_{L1} and d_{L2}
  - Sweet spot: $d_{\text{L1}} = 1$, $d_{\text{L2}} \approx 10$
  - Performance: $\sim$ 600 GFlop/s ($\sim$ 740 GB/s)
- RRII outperforms RIRI by $\sim$ 20%
- GCC 11.1 and clang/LLVM compilers underperform irrespective of layout (not shown here)



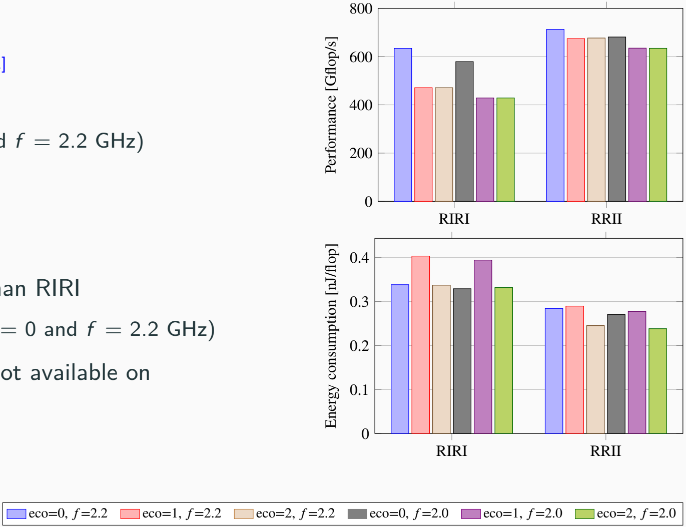
Architecture Comparison

- Fugaku data from [Alappat et al.:PMBS21]
- Roofline model: good agreement on Fugaku
- Fugaku (2.2 GHz) outperforms QPACE 4 (1.8 GHz) by up to ∼ 25%
- Clock frequency ratio: 2.2 / 1.8 = 1.22



Performance and Energy Consumption on Fugaku

- $V = 24^4 \times 8$, data from [Alappat et al.:PMBS21]
- RRII consistently outperforms RIRI
  - Performance benefit $\sim 12\%$ (Eco $= 0$ and $f = 2.2$ GHz)
- Eco modes 1, 2: FLA pipeline only
  - Significant performance drop of RIRI
  - Mild performance drop of RRII
- RRII consistently more energy efficient than RIRI
  - RIRI consumes $\sim 20\%$ more energy (Eco $= 0$ and $f = 2.2$ GHz)
- Tuning knobs and energy measurement not available on QPACE 4



Towards RRII in Grid

- Full integration of RRII in Grid reasonable
  - Enable RRII everywhere, incl. communication
  - No data layout transformation RRII ⇆ RIRI necessary
- Upstream Grid is limited to RIRI layout and 128-, 256- and 512-bit vectors on CPUs
  - For RRII need to separate re and im → RRII implementation of Grid's lower-level functions
  - RRII must respect Grid's internal mechanics
- RRII also likely to be beneficial on other architectures, e.g.,
  - any Armv8 architecture (Arm NEONv8)
  - AMD Epyc Rome (Intel AVX2)

Summary and Outlook

- No relevant progress in SVE compiler performance in one year
  - GCC 10.1 / 10.2 remain best options for Grid
  - GCC 11.1 and clang/LLVM compilers support A64FX as target, but deliver poor performance
- MPI performance issues (future work)
- Domain Wall kernel: RRII layout outperforms RIRI layout in terms of GFlop/s and energy efficiency
  - RRII layout currently not supported by Grid (future work)
- Poster presentation “Grid on QPACE 4” at Lattice 21 (Monday 26/7, 3pm US/Eastern timezone)

This work used computational resources of the supercomputer Fugaku provided by RIKEN through the HPCI System Research Project (Project ID: hp200261). This work was supported in part by KONWIHR, by DFG in the framework of SFB/TRR 55 and by MEXT as “Program for Promoting Researches on the Supercomputer Fugaku” (Simulation for basic science: from fundamental laws of particles to creation of nuclei).

We thank the HPC tools team at Arm UK for support.

References

Fujitsu. FUJITSU Supercomputer PRIMEHPC. 2021 [https://www.fujitsu.com/global/products/computing/servers/supercomputer/index.html].

Peter Boyle, Azusa Yamaguchi, Guido Cossu, and Antonin Portelli. Grid: A next generation data parallel C++ QCD library. *Proceedings of LATTICE 15* (2015), 023 [arXiv:1512.03487].

Christoph Lehner et al. GPT - Grid Python Toolkit. 2021 [https://github.com/lehner/gpt].

Arm. ARM C Language Extensions for SVE. 2020 [https://developer.arm.com/documentation/100987/latest].

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

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

Nils Meyer, Peter Georg, Dirk Pleiter, Stefan Solbrig, and Tilo Wettig. Lattice QCD on QPACE 4. *Asia-Pacific Symposium for Lattice Field Theory (APLAT 2020)* (2020) [https://conference-indico.kek.jp/event/113/contributions/2139/].

Christie Alappat, Thomas Gruber, Georg Hager, Julian Hammer, Jan Laukeman, Nils Meyer, Gerhard Wellein, and Tilo Wettig. ECM modeling and performance tuning of SpMV and Lattice QCD on A64FX. To be published in *Concurrency and Computation: Practice and Experience (PMBS Special Issue)*. 2021 [https://arxiv.org/abs/2103.03013]

Peter Boyle et al. GridBench. 2020 [https://github.com/paboyle/GridBench].

Nils Meyer and Christie Alappat. GridBench with A64FX support. 2021 [https://github.com/nmeyer-ur/GridBench/tree/intrinsics].