JÜLICH
FORSCHUNGSZENTRUM

Exploring SVE for scientific applications

Experiments with ARM SVE

Stepan Nassyr, Forschungszentrum Jülich, 24 January 2018

Member of the Helmholtz Association

Outline

JÜLICH
FORSCHUNGSZENTRUM

- Hardware at JSC
- SVE
  - SVE Overview
  - Applications of interest
  - SVE ZGEMM kernel
  - Compiling for SVE
  - Running SVE applications with ARMIE
- Performance Analysis
  - Challenges
  - Critical path analysis
  - asmcp - dependency graph tool
  - CP Analysis example
- Conclusions

Member of the Helmholtz Association

Hardware at JSC

Member of the Helmholtz Association

JSC Supercomputers

JÜLICH FORSCHUNGSZENTRUM

- JUQUEEN
  - 28672 nodes IBM PowerPC®A2 16 CPU cores per Node
- JURECA
  - 1872 x 2 Xeon E5-2680 v3 CPUs
  - 75 nodes with 2x NVIDIA K80 GPU
  - Booster 1640 x Xeon Phi 7250-F
- QPACE 3
  - 672 Nodes with Intel Xeon Phi 7210
- DEEP
  - Cluster: 256 x Xeon E5-2680
  - Booster: 12 x 16 x 2 x Xeon Phi 7120X

Member of the Helmholtz Association

JSC ARM prototypes

JÜLICH
FORSCHUNGSZENTRUM

- 4x Huawei Taishan 2160
  - Hi1610ES 32xARM Cortex-A57 @ 2.1 Ghz
  - All work done here
- 12x Huawei Taishan 2180
  - Hi1612 32xARM Cortex-A57 @ 2.1 Ghz
  - Not completely configured
- 12x Huawei Taishan 2280
  - Hi1616 32xARM Cortex-A72 @ >2.1 Ghz
  - No OS yet

Member of the Helmholtz Association

SVE

Member of the Helmholtz Association

SVE Overview

JÜLICH
FORSCHUNGSZENTRUM

- SIMD Addition the the ARMv8-A ISA
- Vector Width not known at compile time
- Utilizes predicates for most operations
- Introduces instructions for predicates and loop control

Member of the Helmholtz Association

SVE Challenges

JÜLICH
FORSCHUNGSZENTRUM

- Lot of SIMD Code assumes fixed Vector size (example: BQCD)
- if V times an operation is required, this has to be a loop (V = elements in vector)
- Internal implementations with fixed-size kernels (i.e. in OpenBLAS)

Possible strategies:

- Full vector-size agnostic implementations
- ”Fixed”-size (vector register count) - kernels, actual size known at runtime (i.e. 2x2 full vector kernels instead of 8x8 element kernels)
- Fixed-element-count kernels, select valid kernel at runtime

Member of the Helmholtz Association

Applications of interest

JÜLICH
FORSCHUNGSZENTRUM

- Materials Science:
  - KKRnano
  - Quantum ESPRESSO
- Lattice QCD:
  - BQCD
  - grid
- All these applications use complex arithmetic
- Dominant BLAS routines:
  - Lattice QCD: ZGEMV
  - KKRnano/QE: ZGEMM
- Example Walltime QE (Data: Ausurf112)
  - ZGEMM: 77.8%
  - ZGEMV: 9.7%
  - ZTRMV: 8.3%

Member of the Helmholtz Association

Remarks on ARM/SVE

JÜLICH
FORSCHUNGSZENTRUM

- Structured loads/stores `ld2, st2` avoid explicit permutations (complex numbers)
- SVE and ARMv8.3-A have complex arithmetic instructions (fcmla, fcadd)
  - Currently not auto-generated
  - available through ACLE
- Promise of higher GFLOP/s efficiency
- Current MiniKKR (simplified KKRnano, same geometry) benchmarks achieved 30-50 % of max GFLOP/s on Taishan 2160. Similar figure to Intel X86

Member of the Helmholtz Association

ZGEMM

JÜLICH
FORSCHUNGSZENTRUM

Calculates $C = \beta \times C + \alpha \times A \times B$ (A,B,C complex matrices, α,β complex scalars)

Pseudocode (naive):

```
For i = 1..N
    For k = 1..M
        C(k,i) = beta*C(k,i)
        For j = 1..K
            C(k,i) += A(k,j)*B(j,i)*alpha
```

Member of the Helmholtz Association

ZGEMM

JÜLICH
FORSCHUNGSZENTRUM

Pseudocode (possible vectorization):

```
For i = 1..N
    For j = 1..K (vectorized)
        temp(j:j+l) = B(j:j+l,i)*alpha
    For k = 1..M (vectorized)
        d = vector(l)
        For j = 1..K
            d += mul(vector(temp(j)),A(k:k+l,j))
        C(k:k+l,i)=beta*C(k:k+l,i)+d
```

l is the SIMD size in elements

Member of the Helmholtz Association

SVE ZGEMM

JÜLICH
FORSCHUNGSZENTRUM

Calculating $\beta \times C$-loop:

```
svbool_t prk = svwhilelt_b64(0UL,M*2);
for(std::size_t k = 0; svptest_any(svptrue_b64(),prk);)
{
    svfloat64_t tmp = svld1(prk,reinterpret_cast<double*>(C+i*M+k));
    svfloat64_t c   = svdup_f64(0.0);

    // innermost loop goes here

    c = svcmla_m(prk, c, betavec, tmp, 0);
    c = svcmla_m(prk, c, betavec, tmp, 90);

    svst1(prk, reinterpret_cast<double*>(C+i*M+k),c);

    k+= svcntp_b64(prk,prk)/2;
    prk = svwhilelt_b64(k*2,M*2);
}
```

$\alpha \times B$ loop looks pretty much the same, except we store into a cache

Member of the Helmholtz Association

SVE ZGEMM

JÜLICH FORSCHUNGSZENTRUM

Calculating innermost loop ($A \times B \times \alpha$, $B \times \alpha$ precalculated):

```
for(std::size_t j = 0; j < K; j++)
{

    auto ptr = A+j*M+k;
    svfloat64_t akvec = svld1(prk,reinterpret_cast<const double*>(ptr));
    svfloat64_t ab_ji = svld1rq(prk,reinterpret_cast<double*>(abcache_ptr+j));

    c = svcmla_m(prk, c, ab_ji, akvec, 0);
    c = svcmla_m(prk, c, ab_ji, akvec, 90);

}
```

Member of the Helmholtz Association

Compiling for SVE

JÜLICH
FORSCHUNGSZENTRUM

- ARM HPC compiler suite (version 18.1 out)
- includes armclang, armflang and SVE enabled GCC
- Enable `-march=armv8-a+sve` compiler switch
- SVE ACLE only supported by armclang (AFAIK)

Member of the Helmholtz Association

ARMIE

JÜLICH
FORSCHUNGSZENTRUM

- ARM Instruction Emulator can run SVE-enabled binaries
- Vector bit size can be specified
- DynamoRIO instrumentation support
- Example:

```
armie -c memtrace -msve-vector-bits=256 ./sve_binary
```

- Validity testing possible, even for complex applications

Member of the Helmholtz Association

DynamoRIO

JÜLICH
FORSCHUNGSZENTRUM

- Dynamic binary instrumentation framework
- Memory tracing, instruction counting, basic block analysis
- ARMIE has built-in memtrace and inscount clients:

```
armie -c memtrace -msve-vector-bits=256 ./sve_binary
armie -c inscount -e -msve-vector-bits=256 ./sve_binary
```

- Possible to write own clients

Member of the Helmholtz Association

DynamoRIO Instruction Count

JÜLICH FORSCHUNGSZENTRUM

- Counts AArch64 Instructions and specifically SVE instructions
- SVE instructions are recorded in binary.sve.instrs
- decode utility can be used to make the traces human-readable

```
armie -c inscount -e -msve-vector-bits=256 ./sve_binary
cat sve_binary.sve.instrs \
| awk '{printf("%s %d\n",$2,$3)}'\
| while read x c; do decode $x $c; done
```

Member of the Helmholtz Association

Instruction Count - ZGEMM example

JÜLICH
FORSCHUNGSZENTRUM

```
1       mov     z18.d, d11                               |1       ptrue   p0.d
1       ptrue   p1.d                                     |1       ld1rqd  {z0.d}, p0/z, [x24]
1       str     z17, [x8]                                |1       ld1rqd  {z1.d}, p0/z, [x25]
1       str     z18, [x8, #1, mul vl]                    |1       whilelo p1.d, xzr, x9
16      whilelo p0.d, xzr, x22                           |1       whilelo p2.d, xzr, x10
32      ld2d    {z0.d, z1.d}, p0/z, [x20, x9, lsl #3]    |1       mov     z2.d, #0                // =0x0
32      incd    x8                                       |16      mov     p3.b, p1.b
32      fmul    z3.d, z1.d, z18.d                        |64      ld1d    {z4.d}, p3/z, [x14, x15, lsl #3]
32      fmul    z2.d, z0.d, z18.d                        |64      mov     z3.d, z2.d
32      fnmls   z3.d, p1/m, z0.d, z17.d                  |64      cntp    x17, p3, p3.d
32      movprfx z4, z2                                   |64      fcmla   z3.d, p3/m, z0.d, z4.d, #0
32      fmla    z4.d, p1/m, z1.d, z17.d                  |64      fcmla   z3.d, p3/m, z0.d, z4.d, #90
32      st2d    {z3.d, z4.d}, p0, [x20, x9, lsl #3]      |64      st1d    {z3.d}, p3, [x16, x15, lsl #3]
32      whilelo p0.d, x8, x22                            |64      whilelo p3.d, x17, x9
256     whilelo p0.d, xzr, x22                           |16      mov     p3.b, p2.b
256     mov     z0.d, d0                                 |64      ld1d    {z4.d}, p3/z, [x13, x16, lsl #3]
256     mov     z1.d, d1                                 |64      mov     z3.d, z2.d
512     ld2d    {z2.d, z3.d}, p0/z, [x20, x16, lsl #3]   |64      fcmla   z3.d, p3/m, z4.d, z1.d, #0
512     ld2d    {z4.d, z5.d}, p0/z, [x15, x16, lsl #3]   |64      fcmla   z3.d, p3/m, z4.d, z1.d, #90
512     incd    x14                                      |1024    ld1d    {z4.d}, p3/z, [x17]
512     movprfx z6, z2                                   |1024    ld1rqd  {z5.d}, p0/z, [x18]
512     fmla    z6.d, p1/m, z4.d, z0.d                   |1024    fcmla   z3.d, p3/m, z5.d, z4.d, #0
512     fmls    z6.d, p1/m, z5.d, z1.d                   |1024    fcmla   z3.d, p3/m, z5.d, z4.d, #90
512     movprfx z2, z3                                   |64      st1d    {z3.d}, p3, [x16]
512     fmla    z2.d, p1/m, z4.d, z1.d                   |64      cntp    x16, p3, p3.d
512     movprfx z7, z2                                   |64      whilelo p3.d, x16, x10
512     fmla    z7.d, p1/m, z5.d, z0.d                   |~
512     st2d    {z6.d, z7.d}, p0, [x20, x16, lsl #3]     |~
512     whilelo p0.d, x14, x22                           |~
```

Left: reference FORTRAN implementation, right: custom kernel.

Member of the Helmholtz Association

Instruction Count - ZGEMM example

JÜLICH FORSCHUNGSZENTRUM

- armclang/armflang v. 18.0 beta
- $M = N = K = 16, randomdata$
- Compiled with `-march=armv8-a+simd+sve -Ofast`
- Run with `-msve-vector-bits=512`
- Reference implementation: 23078 AArch64 (7229 SVE) instructions
- Custom kernel: 22970 AArch64 (5030 SVE) instructions
- Reference implementation:
  - Auto-vectorization for 1 of the 2 dept=2 loops fails
  - no fcmla (structured loads + normal fma instead)
  - Stores in innermost loop

Member of the Helmholtz Association

JÜLICH
FORSCHUNGSZENTRUM

Performance Analysis

Member of the Helmholtz Association

Performance Analysis challenges

JÜLICH
FORSCHUNGSZENTRUM

- No SVE-enabled hardware available (to us)
- No cycle-accurate simulator (GEM5/QEMU State?)
- Possible ways to analyze performance:
  - Dynamical analysis through instruction/branch counting
  - Statical analysis by looking at the critical path

Member of the Helmholtz Association

Method

- Attempt to establish reference point (In-Order Processor, no speculative execution, no stalls)
- Analyse assembly instructions for dependencies
- Find the longest dependency chain (Critical Path)
- Find instruction latencies (or assume)
- Calculate loop latency estimate

JÜLICH
FORSCHUNGSZENTRUM

Member of the Helmholtz Association

asmcp - Overview

JÜLICH
FORSCHUNGSZENTRUM

- Custom static analysis tool for generated assembly.
- Builds dependency graphs
- Detects shared registers
- Not bound to an architecture (configurable with a ”definition file”)
- No hard dependencies (dot required to render graph)
- Very WIP

Member of the Helmholtz Association

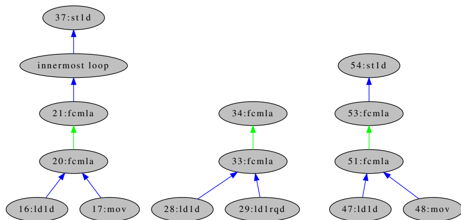
Example of definition file

JÜLICH FORSCHUNGSZENTRUM

```
registers:
  vector [0:64]: d\d+
  vector [0:128]: v\d+
  vector [0:512]: z\d+
  vector [0:512]: z\d+\.d

register sharing:
  d(\d+) [0:64] v\1
  v(\d+) [0:128] z\1
  z(\d+).d[0] [0:512] z\1.d
  z(\d+).d[1] [0:512] z\1.d
  z(\d+).d [0:512] z\1

rules:
  label: \.([A-Za-z0-9_]+):
  comment: //

instructions:
  ld1d        w:r, r:r, r:m           ; lat:?, thr:?
  ld2d        w:r, w:r, r:r, r:m      ; lat:?, thr:?
  ...
  fmla        rw:r, r:r, r:r          ; lat:?, thr:?
  fmls        rw:r, r:r, r:r          ; lat:?, thr:?
  fmul        w:r, r:r, r:r           ; lat:?, thr:?
  ...

jump instructions:
  bl
  b
  b.ne
```

Member of the Helmholtz Association

Example Graph for sve ZGEMM



From left to right: $\beta \times C$-loop, innermost loop, $\alpha \times B$-loop

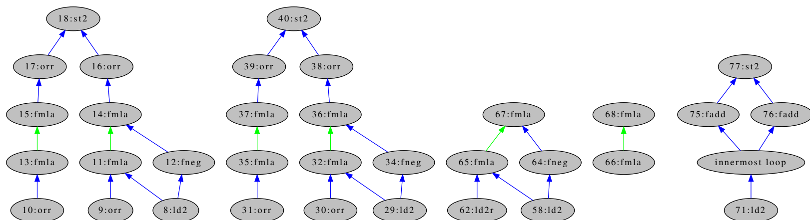
Member of the Helmholtz Association

JÜLICH
FORSCHUNGSZENTRUM

Analysis ot simple hand-written NEON ZGEMM kernel

Member of the Helmholtz Association

Dependency Graph for neon ZGEMM



From left to right: $\beta \times C$-loop, $\alpha \times B$-loop , innermost loop, adding and storing C values

Member of the Helmholtz Association

Instruction Information

JÜLICH FORSCHUNGSZENTRUM

- From Cortex-A57 Optimization Manual:
  - fneg: 3 clock cycles
  - fadd: 5 clock cycles
  - fmla, fmls: 9 clock cycles
  - ld2: 6 clock cycles
  - ld2r: 5 clock cycles
  - st2: 4 clock cycles
  - orr: 3 clock cycles
- Assuming loads don't stall
- With pipeline conflicts $\Delta t(n, k) = n + (k - 1)$
- Without pipeline conflicts $\Delta t(n, k) = n$

Member of the Helmholtz Association

Loop cost (with conflicts)

JÜLICH
FORSCHUNGSZENTRUM

- $\beta \times C$-loop and $\alpha \times B$-loop

$$\Delta t = (6+3-1)+(9+3-1)+(9+2-1)+(3+2-1)+(4+1-1) = 37$$

- innermost loop

$$\Delta t = (6+3-1)+(9+3-1)+(9+2-1) = 29$$

- add and store

$$\Delta t = (6+1-1)+(5+2-1)+(4+1-1) = 16$$

Member of the Helmholtz Association

Loop cost (without conflicts)

JÜLICH FORSCHUNGSZENTRUM

- $\beta \times$ *C*-loop and $\alpha \times$ *B*-loop

$$\Delta t = 6 + 9 + 9 + 3 + 4 = 31$$

- innermost loop

$$\Delta t = 6 + 9 + 9 = 24$$

- add and store

$$\Delta t = 6 + 5 + 4 = 15$$

Loop cost (total)

JÜLICH FORSCHUNGSZENTRUM

- Total loop cost can be calculated with

$$N \times \left(\frac{M}{2} \times \Delta t_{\beta \times C} + \frac{K}{2} \times \Delta t_{\alpha \times B} + \frac{M}{2} \times \left(K \times \Delta t_{inner} + \Delta t_{add,store} \right) \right)$$

- Example $M = N = K = 16$:
  - Loop cost without conflicts: 59008 cycles
  - Loop cost with conflicts: 70912 cycles
- Example $M = N = K = 8$:
  - Loop cost without conflicts: 8608 cycles
  - Loop cost with conflicts: 10304 cycles

Member of the Helmholtz Association

Loop cost (measurement)

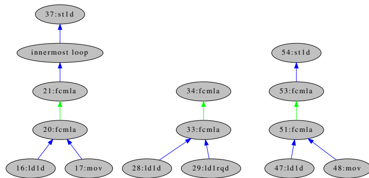
JÜLICH FORSCHUNGSZENTRUM

- Used PAPI to measure performance for mentioned cases
- 10000 inner iterations (iterations between measurements)
- 100 outer iterations (number of measurements)
- $M = N = K = 16$ case:
  - Avg. cycles: 26641
  - Estimate 2.215 to 2.66 times higher than measurement
- $M = N = K = 8$ case:
  - Avg. cycles: 3915
  - Estimate 2.2 to 2.63 times higher than measurement

Member of the Helmholtz Association

SVE predictions

JÜLICH FORSCHUNGSZENTRUM



- Applying the method on a custom zgemm kernel
- Assuming instruction latency like ASIMD/NEON
- Loop costs in cycles:
  - $\beta \times C$-loop and $\alpha \times B$-loop: 27/26
  - innermost loop: 25/24
- Total cycle cost:

$$N \times \left(\frac{K}{V} \times \Delta t_{\alpha \times B} + \frac{M}{V} \times (\Delta t_{\beta \times C} + K \times \Delta t_{inner}) \right)$$

With $V = \frac{vectorbits}{128}$

Member of the Helmholtz Association

SVE predictions

JÜLICH FORSCHUNGSZENTRUM

Vector bits	no conflicts	with conflicts
128	116224	111616
256	58112	55808
512	29056	27904
1024	14528	13952
2048	7264	6972

- Cycle count $\propto \frac{1}{V}$
- Prediction for 128 bits roughly $2\times$ NEON (same length of dependency chain, but double the iterations)

Member of the Helmholtz Association

Conclusions

Conclusions

JÜLICH
FORSCHUNGSZENTRUM

- Scientific Applications can benefit from SVE
- Compilers are at an advanced stage (auto-vectorization with structured loads)
- Correct execution of complex SVE-enabled applications possible with ARMIE
- Performance Analysis remains a challenge
- Our tool asmcp is an attempt to address the challenge, further improvements needed:
  - Analysis assumes pipeline flush at end of basic blocks
  - Further considerations required for more accurate estimates (i.e. pipeline modeling)
  - FMA µop Late-forwarding (accumulate only 4 cycles)

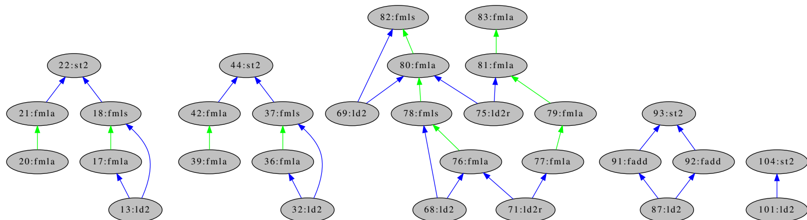
Member of the Helmholtz Association

JÜLICH
FORSCHUNGSZENTRUM

Thank you for your attention!

Member of the Helmholtz Association

Unrolling innermost loop 2x

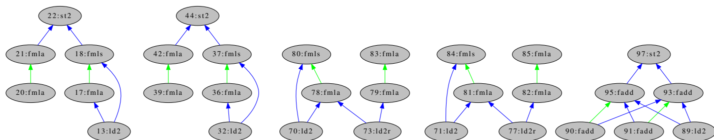


From left to right: $\beta \times C$-loop, $\alpha \times B$-loop , innermost loop, adding and storing C values

- Inner loop cost changes: 49/42 cycles
- Less iterations in inner loop: 61696/52864 cycles estimate
- Measured Cycles: 25619 ($M = N = K = 16$)
- Estimate 2.06 to 2.41 times higher

Member of the Helmholtz Association

Unrolling innermost loop 2x (v2)



From left to right: $\beta \times$ *C*-loop, $\alpha \times$ *B*-loop , innermost loop (2 graphs), adding and storing *C* values

- Same critical path as non-unrolled (more instructions in parallel), but less iterations
- Estimate: 47744/34432 cycles
- Measured Cycles: 23287 ($M = N = K = 16$)
- Estimate 1.48 to 2.05 times higher

Member of the Helmholtz Association

NEON predictions

JÜLICH
FORSCHUNGSZENTRUM

FMA late-forwarding (4 cycles in chain, 9 cycles last) + chained loop

- 1st version: 47744/37888 cycles
- 2x unrolled: 39232/30464 cycles
- 2x unrolled v2: 36288/22272 cycles

Member of the Helmholtz Association

SVE predictions

JÜLICH FORSCHUNGSZENTRUM

Vector bits	no conflicts	with conflicts
128	93184	88576
256	46592	44288
512	23296	22144
1024	11648	11072
2048	5824	5536

FMA late-forwarding (4 cycles in chain, 9 cycles last)

Member of the Helmholtz Association

SVE predictions

JÜLICH
FORSCHUNGSZENTRUM

Vector bits	no conflicts	with conflicts
128	73984	69376
256	36992	34688
512	18496	17344
1024	9248	8672
2048	4624	4336

FMA late-forwarding (4 cycles in chain, 9 cycles last), inner loop chain

Member of the Helmholtz Association