Arm tools for SVE

Where they are and where they're going

Will Lovett

June 2019

HPC Development Solutions from Arm

Best in class commercially supported tools for Linux and high-performance computing

Code Generation

for Arm servers

arm
C/C++ & FORTRAN COMPILER

arm
PERFORMANCE LIBRARIES

+

Performance Engineering

*for **any architecture**, at any scale*

arm
FORGE

arm DDT	*Debugger*
arm MAP	*Profiler*
arm PERFORMANCE REPORTS	*Reporting*

=

arm
ALLINEA STUDIO

Commercially Supported Toolkit for High Performance Computing

- C/C++ Compiler
- Fortran Compiler
- Performance Libraries
- Performance Reports
- DDT Debugger
- MAP Profiler

SVE support in compilers & libraries

Compiler	Assembly/ disassembly	Inline assembly	ACLE	Auto-vectorization	Math Libraries
Arm Compiler for HPC					19.3 (Sept' 2019)
LLVM/Clang			LLVM 10 (~March 2020)	LLVM 11 (~Sept' 2020	
GNU			GNU 10 (~May 2020)		

Debugging SVE

- Supported in GDB today
- Supported in DDT today
- DDT inline assembly view in Forge 20.0

Arm DDT - Arm Ultimate ALL-5626-step-instruction-button

File Edit View Control Tools Window Help

Focus on current: Process Thread Step Threads Together

Threads 1 thread (1) Paused: 1 Playing: 0 Finished: 0

Show threads Currently selected: 1

Project Files

Search (Ctrl+K)

- Application Code
  - /
  - Sources
    - memalign.cpp
      - main() : int
- External Code

memalign.cpp | qt_temp.r18461

```
4   => 0x0000000000400646 <+0>:     55          push   %rbp
5      0x0000000000400647 <+1>:     48 89 e5    mov    %rsp,%rbp
6      0x000000000040064a <+4>:     48 81 ec 30 02 00 00    sub    $0x230,%rsp
7      0x0000000000400651 <+11>:    64 48 8b 04 25 28 00 00 00    mov    %fs:0x28,%rax
8      0x000000000040065a <+20>:    48 89 45 f8    mov    %rax,-0x8(%rbp)
9      0x000000000040065e <+24>:    31 c0   xor    %eax,%eax
10
11  7
12  8           void* allocationsTable[N];
13  9
14  10          for(int i=0; i<N; ++i)
15  11              allocationsTable[i] = (void*)malloc(i);
16  12
17     0x0000000000400660 <+26>:    c7 85 d8 fd ff ff 00 00 00 00    movl   $0x0,-0x228(%rbp)
18     0x000000000040066a <+36>:    83 bd d8 fd ff ff 3f    cmpl   $0x3f,-0x228(%rbp)
19     0x0000000000400671 <+43>:    0f 8f 81 00 00 00       jg     0x4006f8 <main()+178>
20
21  13          for(int i=0; i<N; ++i)
22     0x0000000000400677 <+49>:    be 40 00 00 00  mov    $0x40,%esi
23     0x000000000040067c <+54>:    bf 04 00 00 00  mov    $0x4,%edi
24     0x0000000000400681 <+59>:    e8 aa fe ff ff  callq  0x400530 <aligned_alloc@plt>
25     0x0000000000400686 <+64>:    48 89 85 e8 fd ff ff    mov    %rax,-0x218(%rbp)
26
27  14              printf("%d:\t%llu\n", i, (long long unsigned int)(allocationsTable[i]));
28  15
29  16          for(int i=0; i<N; ++i)
30     0x000000000040068d <+71>:    8b 85 d8 fd ff ff       mov    -0x228(%rbp),%eax
31     0x0000000000400693 <+77>:    48 98   cltq
32     0x0000000000400695 <+79>:    48 8b 95 e8 fd ff ff    mov    -0x218(%rbp),%rdx
33     0x000000000040069c <+86>:    48 89 94 c5 f0 fd ff ff mov    %rdx,-0x210(%rbp,%rax,8)
34
35  17              free(allocationsTable[i]);
36  18
37  19          return 0;
38     0x00000000004006a4 <+94>:    c7 85 dc fd ff ff 00 00 00 00    movl   $0x0,-0x224(%rbp)
```

Tuning SVE

- Flexible hardware counter collection in MAP in 20.0
- `arm-opt-report` available now

Reporting vectorization using arm-opt-report

arm NEOVERSE

arm-opt-report – lets start with some code

```
1   void bar();
2   void foo() { bar(); }
3
4   void Test(int *res, int *c, int *d, int *p, int n) {
5     int i;
6
7   #pragma clang loop vectorize(assume_safety)
8     for (i = 0; i < 1600; i++) {
9       res[i] = (p[i] == 0) ? res[i] : res[i] + d[i];
10    }
11
12    for (i = 0; i < 16; i++) {
13      res[i] = (p[i] == 0) ? res[i] : res[i] + d[i];
14    }
15
16    foo();
17
18    foo(); bar(); foo();
19  }
20
```

arm-opt-report – problem statement

I know this loop is interesting. What did the compiler do with it?

arm-opt-report – what did we have before?

```
$ armclang -O3 or.c -c -o or.o -Rpass=\(loop\|inline\) -march=armv8-a+sve
or.c:16:3: remark: foo inlined into Test with cost=-25 (threshold=375) [-Rpass=inline]
  foo();
  ^
or.c:18:3: remark: foo inlined into Test with cost=-25 (threshold=375) [-Rpass=inline]
  foo(); bar(); foo();
  ^
or.c:18:17: remark: foo inlined into Test with cost=-25 (threshold=375) [-Rpass=inline]
  foo(); bar(); foo();
                ^
or.c:12:3: remark: completely unrolled loop with 16 iterations [-Rpass=loop-unroll]
  for (i = 0; i < 16; i++) {
  ^
or.c:8:3: remark: vectorized loop (vectorization width: 4, interleaved count: 1) [-Rpass=sve-loop-vectorize]
  for (i = 0; i < 1600; i++) {
  ^
```

arm-opt-report – what is it?

It's **beta** quality – ***feedback is welcome!***

It shows **optimization decisions** inline with the original code

- Was a loop **unrolled**?
- If so, what was the **unroll factor**?
- Was a loop **vectorized**?
- What was the **vectorization factor**?
- What was the **interleave count**?

arm-opt-report

```
$ armclang -O3 or.c -c -o or.o -fsave-optimization-record -march=armv8-a+sve
$ arm-opt-report or.opt.yaml
< or.c
 1          | void bar();
 2          | void foo() { bar(); }
 3          |
 4          | void Test(int *res, int *c, int *d, int *p, int n) {
 5          |   int i;
 6          |
 7          | #pragma clang loop vectorize(assume_safety)
 8     V4,1 |   for (i = 0; i < 1600; i++) {
 9          |     res[i] = (p[i] == 0) ? res[i] : res[i] + d[i];
10          |   }
11          |
12     U16  |   for (i = 0; i < 16; i++) {
13          |     res[i] = (p[i] == 0) ? res[i] : res[i] + d[i];
14          |   }
15          |
16 I        |   foo();
17          |
18          |   foo(); bar(); foo();
   I        |   ^
   I        |                 ^
19          | }
```

Vectorized

4x 32-bit lanes

1-way interleaving

Fully unrolled

All 3 instances of `foo()` were inlined

arm-opt-report – what do you want to see next?

Some of the things we want to add

- Analysis information
  - why the compiler decided to optimize
  - why the compiler decided not to optimize
- Actionable advice
  - what can a user do to improve optimization
- Integration with performance analysis
- Clearer support for SVE vector widths

Known issues

- Better Fortran support
  - `armflang -g` is required
  - Fortran line numbers are out by one
  - Fails to recognize Fortran intrinsics

Thank You!

Will Lovett
Technical Product Owner, Arm Compiler for Linux

Technical Support support-hpc-sw@arm.com
Commercial Assistance HPCToolsSales@arm.com

Website https://developer.arm.com/tools-and-software/server-and-hpc