montblanc-project.eu | @MontBlanc_EU

gem5 SVE Hands-On

Richard Cooper
Arm Research

#HiPEAC21

This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement n° 779877

Overview

- Building a gem5 environment (covered this morning).
- A very simple example: Saxpy
  - Compiling the Saxpy example with SVE vectorization for gem5
  - Instrumenting the code for gem5 (m5ops)
  - Running the example in gem5 SE mode
  - Looking at the gem5 output
- A more realistic example: HACC
  - Compiling the HACC Example
  - Running HACC in gem5 SE mode

Prelude: Building a gem5 environment

- **Building gem5 was covered in this morning's session.**
  - A pre-built Docker image is also available on the gem5 website.
  - https://www.gem5.org/documentation/general_docs/building
- **Quick Reminder (for Ubuntu 20.04).**

```
$ mkdir -p /home/gem5-user/hipeac21 ; cd /home/gem5-user/hipeac21
$ sudo apt install build-essential git m4 scons zlib1g zlib1g-dev libprotobuf-dev \
                   protobuf-compiler libprotoc-dev libgoogle-perftools-dev \
                   python3-dev python3-six python-is-python3 libboost-all-dev \
                   pkg-config
$ git clone https://gem5.googlesource.com/public/gem5
$ cd gem5
$ scons -j$(nproc) build/ARM/gem5.opt
$ cd ..
$ export GEM5_PATH="/home/gem5-user/hipeac21/gem5"
```

- **We will also need to install the aarch64 GCC cross-compiler tools.**

```
$ sudo apt install gcc-aarch64-linux-gnu g++-aarch64-linux-gnu binutils-aarch64-linux-gnu
$
```

Saxpy example program

```c
// Saxpy Example for Mont-Blanc Workshop at HiPEAC 2021
// Copyright (c) 2020-2021 Arm Limited
// All rights reserved.
//

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <time.h>



void __attribute__ ((noinline))
saxpy(float * restrict x, float * restrict y, float a, size_t n)
{
   for (size_t i = 0; i < n; ++i)
   {
        y[i] = a * x[i] + y[i];
   }
}

int main(int argc, char * argv[])
{
    if ( argc != 2 ) {
        fprintf(stderr, "Usage: %s num_elements\n", argv[0]);
        exit(1);
    }
    const size_t N = (size_t)atoi(argv[1]);
    if (N == 0) {
        fprintf(stderr, "Usage: %s num_elements\n", argv[0]);
        exit(1);
    } else {
        printf("Running saxpy on %ld elements\n", N);
    }

    float * xs = (float*)malloc(N * sizeof(float));
    float * ys = (float*)malloc(N * sizeof(float));

    const float a = (float)rand() / (float)RAND_MAX;

    for (size_t i = 0; i < N; ++i) {
        xs[i] = (float)rand() / (float)RAND_MAX;
        ys[i] = (float)rand() / (float)RAND_MAX;
    }

    clock_t start = clock();

    saxpy(xs, ys, a, N);

    clock_t end = clock();

    printf("Elapsed time: %fs\n",
           ((float)(end - start)) / CLOCKS_PER_SEC);

    free(xs);
    free(ys);

    exit(0);
}
```

m5ops

- **m5ops are special opcodes that can be inserted into your workload to control the gem5 simulator (e.g. dump statistics, generate checkpoints, etc).**
  - They are encoded in the unused space of the target ISA.
  - https://www.gem5.org/documentation/general_docs/m5ops/
- **gem5 provides a C wrapper and library around these instructions for convenience:**
  - `Include: #include "gem5/m5ops.h“`
  - `Compile: -I${GEM5_PATH}/include`
  - `Link:    -L${GEM5_PATH}/util/m5/build/aarch64/out -lm5`
- **gem5 also provides the `m5` command-line tool which can be called from the console in a full-system simulation. e.g.**
  - `m5 checkpoint`
  - `m5 dump_stats`

Building the m5ops library

- The m5ops library is built separately as part of the `m5` tool.

```
$ cd ${GEM5_PATH}
$ cd util/m5
$ scons build/aarch64/out/m5
$ ls build/aarch64/out
libm5.a  m5
$
```

This can be any of the supported target ISAs:
x86, arm, thumb, sparc, aarch64

- When compiling your workload, link to `libm5.a` and include `gem5/m5ops.h` to use m5ops in your program.
  - `CCFLAGS += -I$(GEM5_PATH)/include`
  - `LDFLAGS += -L$(GEM5_PATH)/util/m5/build/aarch64/out`
  - `LDFLAGS += -lm5`

Some useful m5ops

- **Simulation control**
  - `void m5_exit(uint64_t ns_delay);`
  - `void m5_debug_break(void);`
  - `void m5_switch_cpu(void);`
- **Statistics generation**
  - `void m5_reset_stats(uint64_t ns_delay, uint64_t ns_period);`
  - `void m5_dump_stats(uint64_t ns_delay, uint64_t ns_period);`
  - `void m5_dump_reset_stats(uint64_t ns_delay, uint64_t ns_period);`
- **Checkpoint generation**
  - `void m5_checkpoint(uint64_t ns_delay, uint64_t ns_period);`
- **Workload delimiters**
  - `void m5_work_begin(uint64_t workid, uint64_t threadid);`
  - `void m5_work_end(uint64_t workid, uint64_t threadid);`
- **And many more...**

We will use `m5_reset_stats(0,0)` and `m5_dump_stats(0,0)` in this example to generate statistics for our region of interest.

MONT-BLANC 2020

Saxpy example program

```c
// Saxpy Example for Mont-Blanc Workshop at HiPEAC 2021
// Copyright (c) 2020-2021 Arm Limited
// All rights reserved.
//

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <time.h>

#include "gem5/m5ops.h"

void __attribute__ ((noinline))
saxpy(float * restrict x, float * restrict y, float a, size_t n)
{
   for (size_t i = 0; i < n; ++i)
   {
        y[i] = a * x[i] + y[i];
   }
}

int main(int argc, char * argv[])
{
    if ( argc != 2 ) {
        fprintf(stderr, "Usage: %s num_elements\n", argv[0]);
        exit(1);
    }
    const size_t N = (size_t)atoi(argv[1]);
    if (N == 0) {
        fprintf(stderr, "Usage: %s num_elements\n", argv[0]);
        exit(1);
    } else {
        printf("Running saxpy on %ld elements\n", N);
    }

    float * xs = (float*)malloc(N * sizeof(float));
    float * ys = (float*)malloc(N * sizeof(float));

    const float a = (float)rand() / (float)RAND_MAX;

    for (size_t i = 0; i < N; ++i) {
        xs[i] = (float)rand() / (float)RAND_MAX;
        ys[i] = (float)rand() / (float)RAND_MAX;
    }

    clock_t start = clock();
    m5_reset_stats(0,0);
    saxpy(xs, ys, a, N);
    m5_dump_stats(0,0);
    clock_t end = clock();

    printf("Elapsed time: %fs\n",
           ((float)(end - start)) / CLOCKS_PER_SEC);

    free(xs);
    free(ys);

    exit(0);
}
```

Include m5ops.h from the gem5 source tree.

m5ops allow you to annotate your source code with special instructions for the simulator.

Compiling the Saxpy example

- **GNU gcc can auto-vectorize C code for SVE**
- **Cross compile using the default aarch64-linux-gnu-gcc on Ubuntu 20.04 LTS**
  - On Ubuntu 18.04 LTS, install and compile with aarch64-linux-gnu-gcc-8
- **Use the following compiler switches to enable SVE auto-vectorization:**
  - `No vectorization:          CFLAGS += -march=armv8-a+nosimd+nosve -Ofast`
  - `Enable SVE vectorization:  CFLAGS += -march=armv8-a+sve -Ofast`
- **For gem5 Syscall Emulation (SE) mode, also link statically:**
  - `LDFLAGS += -static`

An example Makefile

- A simple Makefile to build unoptimized and vectorized versions of the Saxpy example:

```
# Saxpy Example Makefile for Mont-Blanc Workshop at HiPEAC 2021
# Copyright (c) 2021 Arm Limited
# All rights reserved.
#

# For Ubuntu 20.04 LTS
CC = aarch64-linux-gnu-gcc

# For Ubuntu 18.04 LTS
#CC = aarch64-linux-gnu-gcc-8

GEM5_PATH = ../gem5

CFLAGS = -I$(GEM5_PATH)/include -Wall -Werror -Ofast
CFLAGS_NOOPT = -march=armv8-a+nosimd+nosve
CFLAGS_SVE = -march=armv8-a+sve
LDFLAGS = -static -L$(GEM5_PATH)/util/m5/build/aarch64/out -lm5

.PHONY: all clean

all: saxpy-noopt saxpy-sve

saxpy-noopt: saxpy.c
	$(CC) $(CFLAGS) $(CFLAGS_NOOPT) -o $@ $< $(LDFLAGS)

saxpy-sve: saxpy.c
	$(CC) $(CFLAGS) $(CFLAGS_SVE) -o $@ $< $(LDFLAGS)

clean:
	rm -f saxpy-noopt
	rm -f saxpy-sve
```

Hands on…

```
gem5-user@wsl:~/hipeac21$
```

Looking at the generated code

- We can verify that `gcc` was able to vectorize the saxpy function by looking at the disassembly using `objdump`.

```
$ aarch64-linux-gnu-gcc \
        -march=armv8.4-a+nosimd+nosve -Ofast \
        -I../gem5/include \
        -o saxpy-noopt saxpy.c \
        -static -L../gem5/util/m5/build/aarch64/out -lm5
$ aarch64-linux-gnu-objdump -d saxpy-noopt | less
...

0000000000400810 <saxpy>:
  400810:       b4000122        cbz     x2, 400834 <saxpy+0x24>
  400814:       d2800003        mov     x3, #0x0
  400818:       bc637802        ldr     s2, [x0, x3, lsl #2]
  40081c:       bc637821        ldr     s1, [x1, x3, lsl #2]
  400820:       1f000441        fmadd   s1, s2, s0, s1
  400824:       bc237821        str     s1, [x1, x3, lsl #2]
  400828:       91000463        add     x3, x3, #0x1
  40082c:       eb03005f        cmp     x2, x3
  400830:       54ffff41        b.ne    400818 <saxpy+0x8>
  400834:       d65f03c0        ret

...
```

```
$ aarch64-linux-gnu-gcc \
        -march=armv8.4-a+sve -Ofast \
        -I../gem5/include \
        -o saxpy-sve saxpy.c \
        -static -L../gem5/util/m5/build/aarch64/out -lm5
$ aarch64-linux-gnu-objdump -d saxpy-sve | less
...

0000000000400810 <saxpy>:
  400810:       b40001a2        cbz     x2, 400844 <saxpy+0x34>
  400814:       d2800003        mov     x3, #0x0                // #0
  400818:       05242002        mov     z2.s, s0
  40081c:       25a21fe0        whilelo p0.s, xzr, x2
  400820:       2598e3e1        ptrue   p1.s
  400824:       d503201f        nop
  400828:       a5434020        ld1w    {z0.s}, p0/z, [x1, x3, lsl #2]
  40082c:       a5434001        ld1w    {z1.s}, p0/z, [x0, x3, lsl #2]
  400830:       65a10440        fmla    z0.s, p1/m, z2.s, z1.s
  400834:       e5434020        st1w    {z0.s}, p0, [x1, x3, lsl #2]
  400838:       04b0e3e3        incw    x3
  40083c:       25a21c60        whilelo p0.s, x3, x2
  400840:       54ffff41        b.ne    400828 <saxpy+0x18>   // b.any
  400844:       d65f03c0        ret

...
```

Looking at the generated code

- We can verify that `gcc` was able to vectorize the saxpy function by looking at the disassembly using `objdump`.

```
$ aarch64-linux-gnu-gcc \
        -march=armv8.4-a+nosimd
        -I../gem5/include \
        -o saxpy-noopt saxpy.c
        -static -L../gem5/util/
$ aarch64-linux-gnu-objdump -d
...

0000000000400810 <saxpy>:
  400810:       b4000122
  400814:       d2800003
  400818:       bc637802
  40081c:       bc637821
  400820:       1f000441
  400824:       bc237821
  400828:       91000463
  40082c:       eb03005f
  400830:       54ffff41
  400834:       d65f03c0

...
```

```
$ aarch64-linux-gnu-gcc \
        -march=armv8.4-a+simd+nosve -Ofast \
        -I../gem5/include \
        -o saxpy-neon saxpy.c \
        -static -L../gem5/util/m5/build/aarch64/out -lm5
$ aarch64-linux-gnu-objdump -d saxpy-sve | less
...

0000000000400810 <saxpy>:
  400810:       b4000482        cbz     x2, 4008a0 <saxpy+0x90>
  400814:       d1000443        sub     x3, x2, #0x1
  400818:       f100087f        cmp     x3, #0x2
  40081c:       54000449        b.ls    4008a4 <saxpy+0x94>
  400820:       d342fc44        lsr     x4, x2, #2
  400824:       d2800003        mov     x3, #0x0
  400828:       4e040403        dup     v3.4s, v0.s[0]
  40082c:       d37cec84        lsl     x4, x4, #4
  400830:       3ce36821        ldr     q1, [x1, x3]
  400834:       3ce36802        ldr     q2, [x0, x3]
  400838:       4e23cc41        fmla    v1.4s, v2.4s, v3.4s
  40083c:       3ca36821        str     q1, [x1, x3]
  400840:       91004063        add     x3, x3, #0x10
  400844:       eb04007f        cmp     x3, x4
  400848:       54ffff41        b.ne    400830 <saxpy+0x20>
  40084c:       f240045f        tst     x2, #0x3
  400850:       927ef443        and     x3, x2, #0xfffffffffffffffc
  400854:       54000260        b.eq    4008a0 <saxpy+0x90>
  400858:       bc637802        ldr     s2, [x0, x3, lsl #2]
  40085c:       91000464        add     x4, x3, #0x1
  400860:       bc637821        ldr     s1, [x1, x3, lsl #2]
  400864:       eb04005f        cmp     x2, x4
  400868:       1f000441        fmadd   s1, s2, s0, s1
  40086c:       bc237821        str     s1, [x1, x3, lsl #2]
  400870:       54000189        b.ls    4008a0 <saxpy+0x90>
  400874:       bc647802        ldr     s2, [x0, x4, lsl #2]
  400878:       91000863        add     x3, x3, #0x2
  40087c:       bc647821        ldr     s1, [x1, x4, lsl #2]
  400880:       eb03005f        cmp     x2, x3
  400884:       1f020401        fmadd   s1, s0, s2, s1
  400888:       bc247821        str     s1, [x1, x4, lsl #2]
  40088c:       540000a9        b.ls    4008a0 <saxpy+0x90>
  400890:       bc637802        ldr     s2, [x0, x3, lsl #2]
  400894:       bc637821        ldr     s1, [x1, x3, lsl #2]
  400898:       1f020400        fmadd   s0, s0, s2, s1
  40089c:       bc237820        str     s0, [x1, x3, lsl #2]
  4008a0:       d65f03c0        ret
  4008a4:       d2800003        mov     x3, #
  4008a8:       17ffffec        b       400858 <saxpy+0x48>
...
```

```
ldr     q1, [x1, x3]
ldr     q2, [x0, x3]
fmla    v1.4s, v2.4s, v3.4s
str     q1, [x1, x3]
add     x3, x3, #0x10
cmp     x3, x4
b.ne    400830 <saxpy+0x20>
```

```
t \
build/aarch64/out -lm5
py-sve | less

z       x2, 400844 <saxpy+0x34>
v       x3, #0x0                // #0
v       z2.s, s0
ilelo   p0.s, xzr, x2
rue     p1.s
b
lw      {z0.s}, p0/z, [x1, x3, lsl #2]
lw      {z1.s}, p0/z, [x0, x3, lsl #2]
la      z0.s, p1/m, z2.s, z1.s
lw      {z0.s}, p0, [x1, x3, lsl #2]
cw      x3
ilelo   p0.s, x3, x2
ne      400828 <saxpy+0x18>   // b.any
t
```

Running the Saxpy example in gem5 SE mode

- **To run a SVE program in gem5 SE mode, the only special thing we need to do is set the SVE vector length.**
  - Do this using the `--param` switch of se.py
  - --param sets a parameter of the SimObjects in the simulator
  - In this case we are setting the sve_vl_se parameter of all the Isa objects under all Cpu objects under the System object.
  - In gem5 the `sve_vl_se` is an integer multiple of 128-bits. In the example below, `sve_vl_se = 4` means the simulation will use an SVE vector length of 512-bits.

```
$ cd ${GEM5_PATH}
$ ./build/ARM/gem5.opt configs/example/se.py \
            ...
            --param 'system.cpu[:].isa[:].sve_vl_se = 4' \
            ...
```

This is a special syntax supported by the --param switch. It means 'all items in the isa collection'.

MONT-BLANC 2020

A simple system configuration

- A simple run-script for the Saxpy example:

```
#!/bin/bash

GEM5_PATH=../gem5

sve_vl=4

${GEM5_PATH}/build/ARM/gem5.opt \
          ${GEM5_PATH}/configs/example/se.py \
          --cpu-type MinorCPU \
          --mem-type SimpleMemory \
          --cmd saxpy-sve --options 6000 \
          --caches --l2cache \
          --l1i_size=64kB --l1i_assoc=4 \
          --l1d_size=64kB --l1d_assoc=4 \
          --l2_size=256kB --l2_assoc=4 \
          --mem-size=1GB \
          --cacheline_size=128 \
          --param "system.cpu[:].isa[:].sve_vl_se = ${sve_vl}"
```

Hands on...

```
gem5-user@wsl:~/hipeac21/saxpy$ make
aarch64-linux-gnu-gcc -I../gem5/include -Wall -Werror -Ofast -march=armv8-a+nosimd+nosve -o saxpy-noopt saxpy.c -sta
tic -L../gem5/util/m5/build/aarch64/out -lm5
aarch64-linux-gnu-gcc -I../gem5/include -Wall -Werror -Ofast -march=armv8-a+sve -o saxpy-sve saxpy.c -static -L../ge
m5/util/m5/build/aarch64/out -lm5
gem5-user@wsl:~/hipeac21/saxpy$ ll saxpy-*
-rwxrwxrwx 1 gem5-user gem5-user 605968 Jan 14 20:44 saxpy-noopt*
-rwxrwxrwx 1 gem5-user gem5-user 605968 Jan 14 20:44 saxpy-sve*
gem5-user@wsl:~/hipeac21/saxpy$
```

Looking at the gem5 Statistics

- **The gem5 output is stored in ./m5out by default, or a directory specified by the user using the --outdir/-d flag.**
  http://learning.gem5.org/book/part1/gem5_stats.html
  - config.ini, config.json: A record of the system configuration.
  - simerr, simout: The simulation output, if run with the -r and -e flags.
  - stats.txt: The output statistics of the simulation.

```
---------- Begin Simulation Statistics ----------
final_tick                                  624412500                       # Number o
(restored from checkpoints and never reset)
host_inst_rate                               17533140                       # Simulato
host_mem_usage                                 264700                       # Number o
host_op_rate                                 21349632                       # Simulato
host_seconds                                     0.05                       # Real ti
host_tick_rate                               66255762                       # Simulato
sim_freq                                1000000000000                       # Frequency of simulated ticks
sim_insts                                      842001                       # Number of instructions simulated
sim_ops                                       1035265                       # Number of ops (including micro ops) simulated
sim_seconds                                  0.000003                       # Number of seconds simulated
sim_ticks                                     3219500                       # Number of ticks simulated
....
```

- stats.txt may contain multiple statistics blocks:
- One block is generated for each call to m5_dump_stats() or m5_dump_reset_stats().
- A final block is generated when the simulation ends.

Looking at the gem5 Statistics - Simulation Time

```
---------- Begin Simulation Statistics ----------
final_tick                                   624412500                      # Number of ticks from beginning of simulation (restored from...
host_inst_rate                                17533140                      # Simulator instruction rate (inst/s)
host_mem_usage                                  264700                      # Number of bytes of host memory used
host_op_rate                                  21349632                      # Simulator op (including micro ops) rate (op/s)
host_seconds                                      0.05                      # Real time elapsed on the host
host_tick_rate                                66255762                      # Simulator tick rate (ticks/s)
sim_freq                                 1000000000000                       # Frequency of simulated ticks
sim_insts                                       842001                      # Number of instructions simulated
sim_ops                                        1035265                      # Number of ops (including micro ops) simulated
sim_seconds                                   0.000003                      # Number of seconds simulated
sim_ticks                                      3219500                      # Number of ticks simulated
system.cpu.committedInsts                         2642                      # Number of instructions committed
system.cpu.committedOps                           2642                      # Number of ops (including micro ops) committed
system.cpu.cpi                                2.437169                      # CPI: cycles per instruction
system.cpu.discardedOps                             17                      # Number of ops (including micro ops) which were discarded...
system.cpu.idleCycles                                2                      # Total number of cycles that the object has spent stopped
system.cpu.ipc                                0.410312                      # IPC: instructions per cycle
system.cpu.numCycles                              6439                      # number of cpu cycles simulated
system.cpu.numFetchSuspends                          0                      # Number of times Execute suspended instruction fetching
system.cpu.numWorkItemsCompleted                     0                      # number of work items this cpu completed
system.cpu.numWorkItemsStarted                       0                      # number of work items this cpu started
....
```

Looking at the gem5 Statistics - Instruction Counts

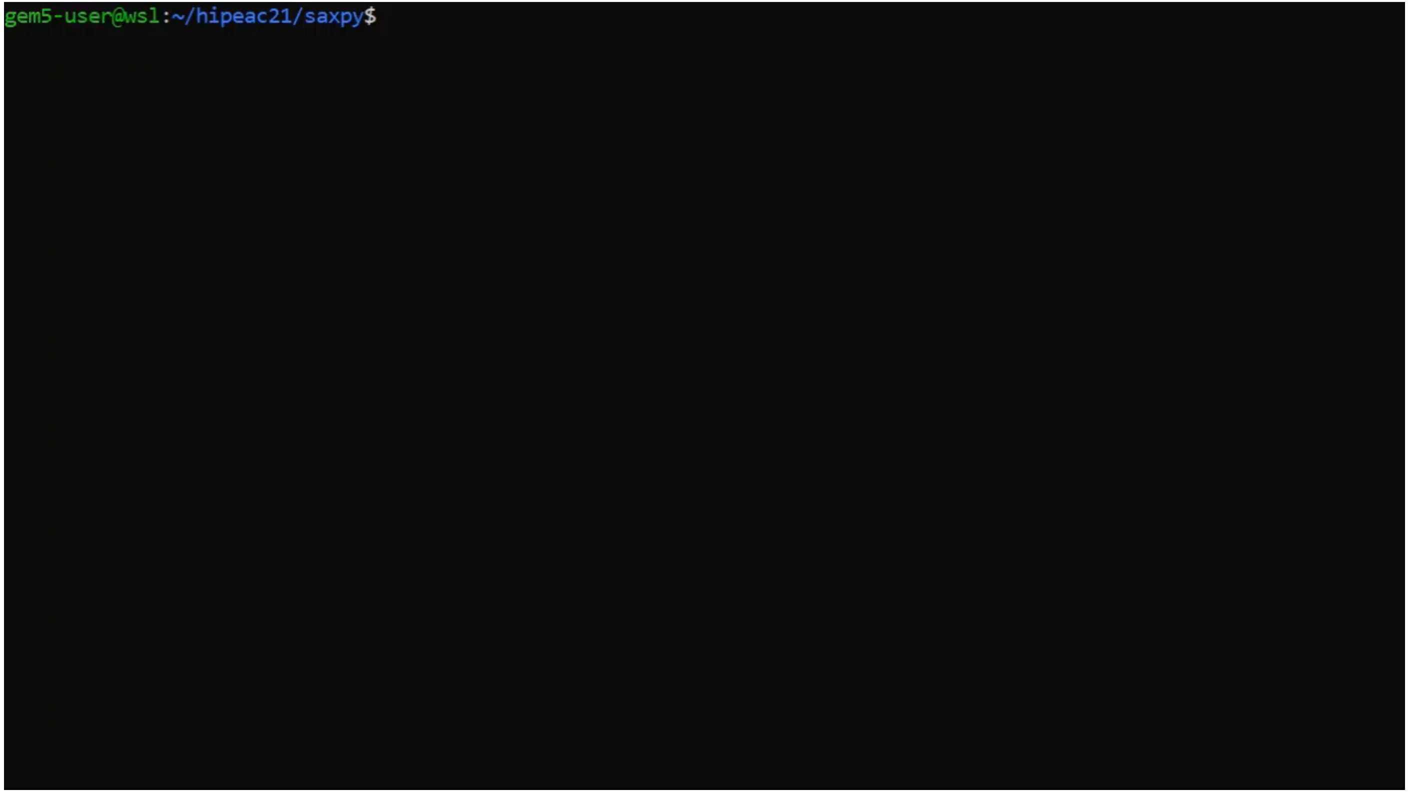
```
---------- Begin Simulation Statistics ----------

....

system.cpu.op_class_0::No_OpClass              0       0.00%       0.00% # Class of committed instruction
system.cpu.op_class_0::IntAlu                388      14.69%      14.69% # Class of committed instruction
system.cpu.op_class_0::IntMult                 0       0.00%      14.69% # Class of committed instruction
system.cpu.op_class_0::IntDiv                  0       0.00%      14.69% # Class of committed instruction
...
system.cpu.op_class_0::FloatMult               0       0.00%      14.69% # Class of committed instruction
system.cpu.op_class_0::FloatMultAcc            0       0.00%      14.69% # Class of committed instruction
system.cpu.op_class_0::FloatDiv                0       0.00%      14.69% # Class of committed instruction
system.cpu.op_class_0::FloatMisc               1       0.04%      14.72% # Class of committed instruction
...
system.cpu.op_class_0::SimdFloatMult           0       0.00%      43.19% # Class of committed instruction
system.cpu.op_class_0::SimdFloatMultAcc      375      14.19%      57.38% # Class of committed instruction
system.cpu.op_class_0::SimdFloatSqrt           0       0.00%      57.38% # Class of committed instruction
...
system.cpu.op_class_0::SimdShaSigma3           0       0.00%      57.38% # Class of committed instruction
system.cpu.op_class_0::SimdPredAlu             1       0.04%      57.42% # Class of committed instruction
system.cpu.op_class_0::MemRead               750      28.39%      85.81% # Class of committed instruction
system.cpu.op_class_0::MemWrite              375      14.19%     100.00% # Class of committed instruction
system.cpu.op_class_0::FloatMemRead            0       0.00%     100.00% # Class of committed instruction
system.cpu.op_class_0::FloatMemWrite           0       0.00%     100.00% # Class of committed instruction

...
```

Hands on…



Comparing different SVE vector lengths

- **A simple run-script for the Saxpy example:**

```
#!/bin/bash

GEM5_PATH=../gem5

for sve_vl in $(seq 1 16) ; do
    outdir=results/saxpy-sve-vl${sve_vl}
    mkdir -p ${outdir}
    ${GEM5_PATH}/build/ARM/gem5.opt \
                --outdir ${outdir} \
                ${GEM5_PATH}/configs/example/se.py \
                --cpu-type MinorCPU \
                --mem-type SimpleMemory \
                --cmd saxpy-sve --options 6000 \
                --caches --l2cache \
                --l1i_size=64kB --l1i_assoc=4 \
                --l1d_size=64kB --l1d_assoc=4 \
                --l2_size=256kB --l2_assoc=4 \
                --mem-size=1GB \
                --cacheline_size=128 \
                --param "system.cpu[:].isa[:].sve_vl_se = ${sve_vl}“
done
```

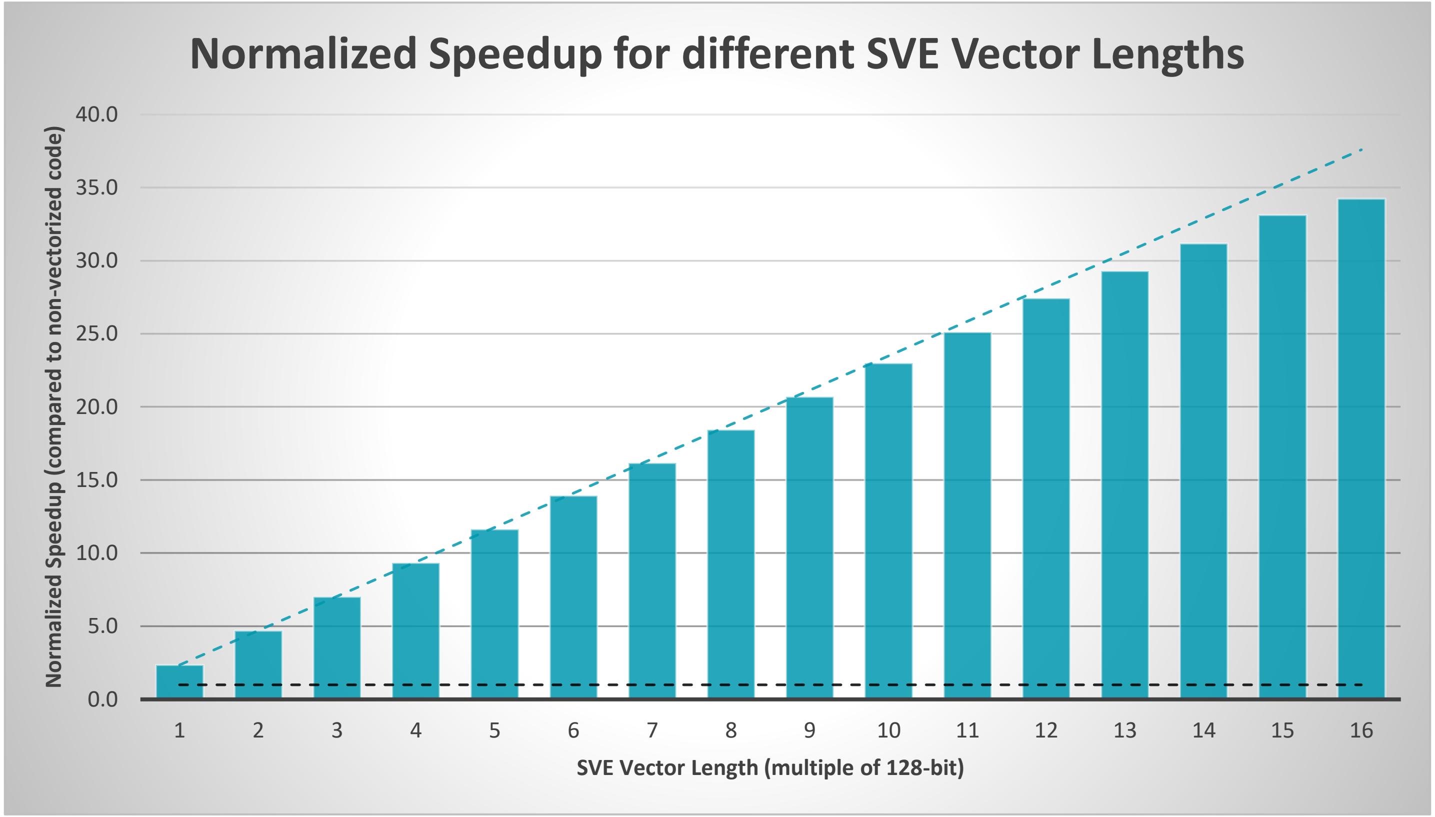
Comparing different SVE vector lengths

- **Each stats.txt file will have two statistics blocks.**
  - The first contains the statistics for the region between m5_reset_stats() and m5_dump_stats().
  - The second contains the stats at the end of the simulation.
- **Extract the relevant statistics from the first block to compare the SVE simulations for different vector lengths.**
  - sim_seconds, sim_ticks, system.cpu.numCycles
  - system.cpu.op_class_0::
    - {FloatMultAcc, SimdFloatMultAcc, MemRead, MemWrite}
- **The statistics of interest can easily be extracted manually or using your favourite tools (e.g. Python).**

Comparing different SVE vector lengths

Stats Filename	sim_seconds	sim_ticks	numCycles	FloatMultAcc	SimdFloatMultAcc	MemRead	MemWrite	$numCycles_{noopt}/numCycles$
results/saxpy-6000-noopt-vl1/stats.txt	0.0000300	30034000	60068	6000	0	12000	6000	1.00000
results/saxpy-6000-sve-vl1/stats.txt	0.0000130	12782000	25564	0	1500	3000	1500	2.34971
results/saxpy-6000-sve-vl2/stats.txt	0.0000060	6407000	12814	0	750	1500	750	4.68769
results/saxpy-6000-sve-vl3/stats.txt	0.0000040	4282000	8564	0	500	1000	500	7.01401
results/saxpy-6000-sve-vl4/stats.txt	0.0000030	3219500	6439	0	375	750	375	9.32878
results/saxpy-6000-sve-vl5/stats.txt	0.0000030	2582000	5164	0	300	600	300	11.63207
results/saxpy-6000-sve-vl6/stats.txt	0.0000020	2157000	4314	0	250	500	250	13.92397
results/saxpy-6000-sve-vl7/stats.txt	0.0000020	1859500	3719	0	215	430	215	16.15165
results/saxpy-6000-sve-vl8/stats.txt	0.0000020	1630000	3260	0	188	376	188	18.42577
results/saxpy-6000-sve-vl9/stats.txt	0.0000010	1451500	2903	0	167	334	167	20.69170
results/saxpy-6000-sve-vl10/stats.txt	0.0000010	1307000	2614	0	150	300	150	22.97934
results/saxpy-6000-sve-vl11/stats.txt	0.0000010	1196500	2393	0	137	274	137	25.10155
results/saxpy-6000-sve-vl12/stats.txt	0.0000010	1094500	2189	0	125	250	125	27.44084
results/saxpy-6000-sve-vl13/stats.txt	0.0000010	1025000	2050	0	116	232	116	29.30146
results/saxpy-6000-sve-vl14/stats.txt	0.0000010	963500	1927	0	108	216	108	31.17177
results/saxpy-6000-sve-vl15/stats.txt	0.0000010	907000	1814	0	100	200	100	33.11356
results/saxpy-6000-sve-vl16/stats.txt	0.0000010	877500	1755	0	94	188	94	34.22678

Comparing different SVE vector lengths



MONT-BLANC 2020

What is gem5 Simulating

- **These results look too good to be true...**
  - Very simple workload and region of interest.
  - Small working set.
  - I chose a slightly unrealistic system configuration so the workload would not be memory bound.
- **What does gem5 simulate out of the box?**
  - Not any existing CPU...
    - Generic in-order and out-of-order microarchitecture.
    - Operation latencies are not tuned for a specific CPU.
    - Surrounding memory system is very configurable.
- **Always know what you are simulating**
  - Sometimes the out-of-the box behaviour is sufficient - e.g. to compare relative performance.
  - For other investigations it may be necessary to tune the model:
    - Accurate custom CPU models.
    - Tune operation latencies to match a specific CPU.
    - Configure the surrounding system to match a target platform.
  - *“The gem5 simulator is a modular platform for computer-system architecture research.”*

```
saxpy(float * restrict x,
      float * restrict y,
      float a,
      size_t n)
{
   for (size_t i = 0; i < n; ++i)
   {
       y[i] = a * x[i] + y[i];
   }
}
```

```
#!/bin/bash

GEM5_PATH=../gem5

for sve_vl in $(seq 1 16) ; do
    outdir=results/saxpy-sve-vl${sve_vl}
    mkdir -p ${outdir}
    ${GEM5_PATH}/build/ARM/gem5.opt \
                 --outdir ${outdir} \
                 ${GEM5_PATH}/configs/example/se.py \
                 --cpu-type MinorCPU \
                 --mem-type SimpleMemory \
                 --cmd saxpy-sve --options 6000 \
                 --caches --l2cache \
                 --l1i_size=64kB --l1i_assoc=4 \
                 --l1d_size=64kB --l1d_assoc=4 \
                 --l2_size=256kB --l2_assoc=4 \
                 --mem-size=1GB \
                 --cacheline_size=128 \
                 --param "system.cpu[:].isa[:].sve_vl_se = ${sve_vl}“
done
```

Where to find the Op Class Names

- **The SVE instructions are defined in:**
  `gem5/src/arch/arm/isa/insts/sve.isa & sve_mem.isa`

Opcode

Op Class

```
e.g.
# FMLA (vectors)
sveTerInst('fmla', 'Fmla', 'SimdFloatMultAccOp', floatTypes, fmlaCode, PredType.MERGE)
```

- **The Op Class Latencies are defined in the CPU code for each CPU type.**
  - Tuning the CPU models' Op Class latencies is beyond the scope of this tutorial, but the basic process is to derive a CPU class from one of the base CPU classes and provide a custom Functional Unit Pool.
  - See the config scripts in `gem5/configs/common/cores/arm` for examples.

A more complex example: HACC

- Now let’s try to run the HACC example from the Arm SVE Tools Tutorial: https://gitlab.com/arm-hpc/training/arm-sve-tools
- This is the HACCKernels Benchmark for Hardware/Hybrid Accelerated Cosmology Code (HACC) - see the README for more details.
- Uses OpenMP and can be SVE vectorized.
- Requires a couple of small changes to the make configuration for cross compilation (see Hands-on).
  - arm-sve-tools/config.mk:34
    ```
    - CFLAGS_OPT    = -Ofast -mcpu=native
    + CFLAGS_OPT    = -Ofast -mcpu=generic -static
    ```
  - arm-sve-tools/ 05_Apps/01_HACC/Makefile:54
    ```
    -       $(CXX) $(CXXFLAGS_REPORT) $(CXXFLAGS_OPT) $(CXXFLAGS_OPENMP) -o $@ $^
    +       $(CXX) $(CXXFLAGS_REPORT) $(CXXFLAGS_OPT) -march=armv8-a+sve $(CXXFLAGS_OPENMP) -o $@ $^
    ```

A more complex example: HACC

Building the example:

```
$ cd /home/gem5-user/hipeac21
$ git clone https://gitlab.com/arm-hpc/training/arm-sve-tools.git
$ # Edit config.mk here...
$ cd arm-sve-tools/05_Apps/01_HACC
$ make COMPILER=gnu CC=aarch64-linux-gnu-gcc CXX=aarch64-linux-gnu-g++
..... Compiler output .....
$
```

Running the example:

```
$ ${GEM5_PATH}/build/ARM/gem5.opt \
      ${GEM5_PATH}/configs/example/se.py \
      --cpu-type DerivO3CPU --num-cpus ${NUM_CPUS} --mem-type SimpleMemory --mem-size=1GB \
      --cmd hacc_gnu_sve.exe --options 4 \
      --caches --l2cache
      --l1i_size=64kB --l1i_assoc=4 \
      --l1d_size=64kB --l1d_assoc=4
      --l2_size=256kB --l2_assoc=4 \
      --cacheline_size=128 --param "system.cpu[:].isa[:].sve_vl_se = 4
```

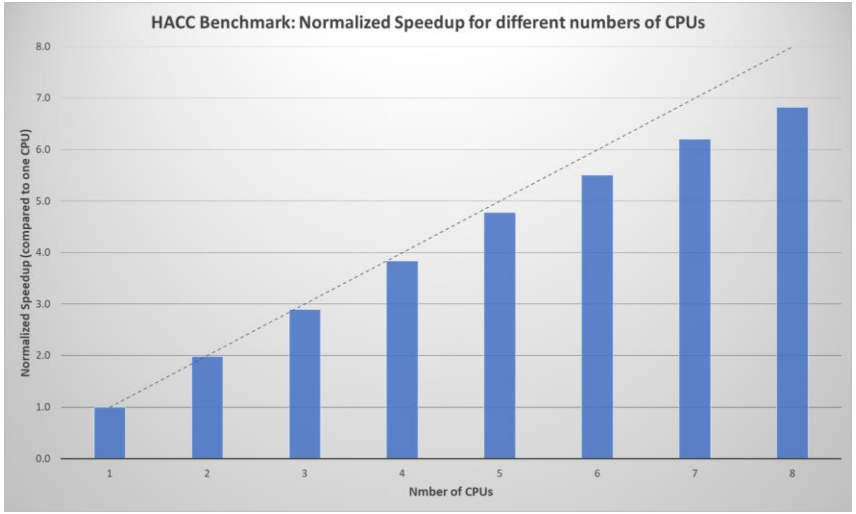
Hands on…

```
gem5-user@wsl:~/hipeac21$ git clone https://gitlab.com/arm-hpc/training/arm-sve-tools.git
Cloning into 'arm-sve-tools'...
remote: Enumerating objects: 1035, done.
remote: Counting objects: 100% (1035/1035), done.
remote: Compressing objects: 100% (513/513), done.
remote: Total 1406 (delta 580), reused 952 (delta 510), pack-reused 371
Receiving objects: 100% (1406/1406), 250.93 MiB | 5.49 MiB/s, done.
Resolving deltas: 100% (744/744), done.
Updating files: 100% (354/354), done.
gem5-user@wsl:~/hipeac21$ cp ../run-hacc-simulation.sh arm-sve-tools/05_Apps/01_HACC
gem5-user@wsl:~/hipeac21$
```

HACC Benchmark: Simulated Speedup

Number of CPUs	sim_seconds	sim_ticks	$sim\_ticks_{(1\ CPU)}/sim\_ticks$
1	0.091960	91959629000	1.0000
2	0.046387	46386878000	1.9824
3	0.031744	31743740000	2.8969
4	0.023985	23985125000	3.8340
5	0.019242	19242198000	4.7791
6	0.016710	16709786000	5.5033
7	0.014827	14826835000	6.2022
8	0.013489	13488657000	6.8176

SVE Vector Length = 4



Running SVE programs in Full-System Mode

- **SVE also works in gem5 Full System Mode.**
- **In gem5/configs/example/fs.py, configure the *maximum* SVE vector length using the `--sve-vl` command line parameter (this is different to se.py mode).**
- **Inside Linux, the default SVE Vector Length can be read or set for new processes using the `procfs` interface.**
  - Read:
    `cat /proc/sys/abi/sve_default_vector_length`
  - Set:
    `echo ${vl} > /proc/sys/abi/sve_default_vector_length`
  - Note: Linux uses the number of bytes to describe the SVE Vector Length:
    128-bits ⇔ sve_default_vector_length=16 (Linux) ⇔ sve_vl=1 (gem5)

SVE Vector Length	gem5 sve_vl	Linux
128 (1 x 128-bits)	1	16
256 (2 x 128-bits)	2	32
384 (3 x 128-bits)	3	48
…	…	…

Concluding Remarks

- **Running SVE Programs in gem5 doesn't require any special consideration, apart from configuring the SVE Vector Length.**
- **Pay attention to the system that is being simulated: is it realistic?**
  - What counts a realistic depends on your research purpose.

MONT-BLANC 2020

QUESTION TIME

More information:
montblanc-project.eu
@MontBlanc_EU