Overview of ARM-SVE gem5 simulator with "Open parameters"

Yuetsu Kodama, Tetsuya Odajima and Mitsuhisa Sato

RIKEN R-CCS

{yuetsu.kodama, tetsuya.odajima, msato}@riken.jp

Open gem5-Arm-SVE

- **RIKEN developed a processor simulator based on gem5 for supercomputer Fugaku to evaluate its kernel performance.**
  - Since RIKEN simulator is based on parameters of A64FX, that is a processor for Fugaku, under NDA with Fujitsu, we cannot share it with the community.
- **Gem5-riken-open is an "open" version of RIKEN simulator for Arm-SVE by excluding the detailed parameters of A64FX.**
  - By co-operations with Prof. Simon in University of Bristol, we set the "open" parameters and re-build the gem5 simulator.
  - **Note: "Some of parameters in this gem5 are taken from public information about Cavium ThunderX2, but all parameters including these parameters are not relevant to any specific hardware."**
  - The gem5 simulator provided supports the simulation of SVE instructions on an Arm based architectural mode.
  - Based on an out-of-order processor core, similar to that in a current generation Arm HPC processor, we also implement a SVE unit, whose default set is 512-bit.

Set up on CEA (1)

```
# Login to our R&D cluster (inti)
$ ssh inti.ocre.cea.fr
# Setup public key authentication
$ ssh-keygen # Press enter at every prompt
$ cp .ssh/id_rsa.pub .ssh/authorized_keys
# Allocate a VM with 8 cores within the "sandy" partition and the "workshop"
# reservation for 1 day. The "dockerenv" image has been setup with docker
# and your home directory from the physical cluster will be mounted inside.
# The VM disk itself is ephemeral so any data stored outside of your home will
# be lost once the VM allocation is terminated.
$ pcocc alloc -p sandy -t 1-0 -c 8 --reservation=workshop dockerenv
# Wait at least 30s for the VM to boot
# Access the VM via ssh
$ pcocc ssh vm0
# Add yourself to the docker group to allow using docker without "sudo"
$ sudo usermod -a -G docker $USER
# Logout and log back in to update your groups
$ exit
```

Set up on CEA (2)

```
$ pcocc ssh vm0

# Use the gem5 docker image
$ docker run -ti linaro/gem5-riken-open


# Your home is shared with the physical cluster, including the ARM nodes
# Assuming you have compiled ARM binaries in $HOME/arm you could use the
# following command to access these binaries from /arm in your container
$ docker run  -v $HOME/arm:/arm -ti linaro/gem5-riken-open
```

In following sample, gem5-riken-open should be replaced to linaro/gem5-riken-open.

Linaro/gem5-riken-open may be still slightly old version where some tool's option is not implemented. We are now request new image to linaro, but you can copy new image from USB stick to your terminal, and copy to the training environment.

Sample Run

```
// [copy gem5-riken-open.tar.gz from USB memory]  if you run docker on your PC
// # docker load –i gem5-riken-open.tar.gz
# docker image ls
REPOSITORY          TAG          IMAGE ID         CREATED          SIZE
gem5-riken-open      latest       c4b8dacb38b5       2 hours ago      3.07 GB
# docker run -it -u r-sim -w /home/r-sim gem5-riken-open
$ cp /opt/samples/daxpy_C.c daxpy.c
$ make -f /opt/samples/Makefile_C daxpy.axf
$ time gem5-o3 -c daxpy.axf -n 1
gem5 Simulator System.  http://gem5.org
…
command line: /opt/gem5/fs_sim_riken/build/ARM/gem5.opt
/opt/gem5/fs_sim_riken/configs/example/se.py --cpu-type=O3_ARM_riken_3 --
caches --l2cache -c daxpy.axf -n 1 -e omp1.txt
…
I'm 0 of 1.
time = 0.028237 ms, 7.252895 GFLOPS, dummy = 306900.000000
Exiting @ tick 71442840 because exiting with last active thread context
real    0m3.614s
$ exit
```

This time is measured in kernel part between get_dtime() in program.

In this case, total time is 0.071 ms on simulator (tick is ps), so gem5 is 50K times slower than real chip.

Compare SVE vs NOSVE

```
$ cp /opt/samples/Makefile_C Makefile.nosve
[ edit Makefile.nosve ]
$ diff Makefile.nosve /opt/samples/Makefile_C
4c4
< CFLAGS        = -O3 -march=armv8-a -fopenmp -static --sysroot=$(ARM_SYSROOT)
---
> CFLAGS        = -O3 -march=armv8-a+sve -fopenmp -static --sysroot=$(ARM_SYSROOT)
$ ln –s daxpy.c daxpy-nosve.c
$ make –f Makefile.nosve daxpy-nosve.axf
$ time gem5-o3 -c daxpy-nosve.axf -n 1
…
I'm 0 of 1.
time = 0.088619 ms, 2.311017 GFLOPS, dummy = 306900.000000
Exiting @ tick 133136724 because exiting with last active thread context
real    0m5.309s
$ exit
```

Using SVE is 3.2 time faster than nosve.

Compare SVE between vector length

```
$ gem5-o3 -c daxpy.axf -n 1 –v 128
…
I'm 0 of 1.
time = 0.064803 ms, 3.160348 GFLOPS, dummy = 306900.000000
Exiting @ tick 10972038 because exiting with last active thread context
$ exit
```

-v [128 | 256 | 512 | 1024 | 2048] // set vector length

SVE supports Vector Length Agnostic programming, so same binaries are available for different vector length.

Vector Length	GFLOPS	Ratio
No sve	2.31	0.73
-v 128	3.16	1.00
-v 256	5.08	1.61
-v 512	7.25	2.29
-v 1024	11.38	3.60
-v 2048	13.72	4.34

Stats.txt

```
$ gem5-o3 –c daxpy.axf –n 1
…
time = 0.028237 ms, 7.252895 GFLOPS, dummy = 306900.000000
Exiting @ tick 71442840 because exiting with last active thread context
$ less m5out/stats.txt
---------- Begin Simulation Statistics ----------
sim_seconds                    0.000071               # Number of seconds simulated
sim_ticks                     71441840                # Number of ticks simulated
…
$ cat daxpy.c
…
 time = get_dtime();
…
   for (i = 0; i < iteration; i++) {
   #pragma omp for nowait
     for (j = 0; j < N; j++) {
       y[j] += alpha * x[j];
     }
   }
  time = get_dtime() - time;
```

Stats.txt incudes simulation statistics from start to end of simulation, but we want to know statistics about kernel region.

In program, the kernel part is enclosed by two get_dtime () to measure kernel execution time.

Run-pa

Run-pa is a script to get simulation statistics in the region specified by two get_dtime().

```
$ run-pa daxpy.axf T1
Running 1pass
Running 2pass
run-pa completed
stats-T1-all.txt: PA information in all section
stats-T1.txt: PA information between 27856472 and 55993308 ticks
$ run-pa --help
usage: run-pa {binary} {tag} [-p|--print-stdout]
positional arguments:
  binary                Your binary file
  tag                   New stats name
optional arguments:
  -h, --help            show this help message and exit
  -n NUM_THREADS, --num_threads NUM_THREADS
                        Number of OpenMP threads
  -p, --print-stdout    Show all stdout/stderr
```

Gem5-pa

```
$ gem5-pa stats-T1.txt
===Use double precision. Vector length :512bit===
***Whole information***
Execution Time : 2.8e-05 [sec]
FLOP : 204800
GFLOPS : 7.31
Number of instructions : 93062
***cpu information***
Number of committed instructions on cpu : 93062 ( 100.0 % )
Number of L1D misses : 154
L1D Miss rate : 0.2 [%]
committed_insts_per_cycle::0 27.50 [%]
committed_insts_per_cycle::1 2.68 [%]
committed_insts_per_cycle::2 2.27 [%]
committed_insts_per_cycle::3 32.10 [%]
committed_insts_per_cycle::4 35.45 [%]
Waiting memory : 12.02 [%]
Waiting decode : 10.23 [%]
Waiting microOp : 0.39 [%]
Waiting squash : 0.0 [%]
Waiting calculation : 4.86 [%]
```

Gem5-pa is a script to display the compatible profile data of Fujitsu system from the stats.txt file.

-a option shows information of all CPUS,
For other options, please run gem5-pa --help.

Waiting ** shows the reason why 0 instruction committed. Waiting memory is a case that the top of the reorder buffer is a memory instruction.

Look an assembler list

```
$ diff Makefile /opt/sample/Makefile_C
16,18d15
< %.s: %.c
<       $(CC) $(CFLAGS) -S -o $@ $<
<
$ make daxpy.s
$ less daxpy.s
…
.L5:
        ld1d    z0.d, p0/z, [x1, x0, lsl 3]
        ld1d    z1.d, p0/z, [x3, x0, lsl 3]
        fmla    z0.d, p1/m, z1.d, z2.d
        st1d    z0.d, p0, [x1, x0, lsl 3]
        incd    x0
        whilelo p0.d, x0, x2
        bne     .L5
…
```

Search ld1, then this field are shown.
This is an assembler list of kernel part of daxpy.
These are vector length agnostic coding.

Multithread execution

```
$ gem5-o3 –c daxpy.axf –n 2
I'm 0 of 2.
I'm 1 of 2.
time = 0.029529 ms, 6.935555 GFLOPS, dummy = 306900.000000
Exiting @ tick 72668540 because exiting with last active thread context
$ gem5-o3 –c daxpy.axf –n 1 –o 1000
...
time = 0.144212 ms, 14.201315 GFLOPS, dummy = 3069000.000000
$ gem5-o3 –c daxpy.axf –n 2 –o 1000
...
time = 0.105838 ms, 19.350328 GFLOPS, dummy = 3069000.000000
```

-n 2 specifies the two thread execution.
But in this case, performance does not increased because of too small execution time.

-o specifies the arguments to the program.
In the case that argument is 1000, two thread execution is 1.36 times faster than 1 thread.

Current parameter overview

type	list	value
SVE		512bit
Pipeline stage width	Decode	4
	Dispatch	6
	issue	8
	commit	4
Exec units	Int complex/ simple	1/2
	Float	2
	Load	2
	Store	1

type	list	value
Cache	L1D/core	32KB 8way
	L2/shared	8MB/bank 8way/bank 4bank
	Bus width	32B
	Cache line	64B
Memory	DDR4-2400	128MB(*)/ch 19.2GB/ch 8ch

(*) memory model is 1GB/ch but for docker with small memory we limited the total memory to 1GB.