Tutorial

Beginning with gem5

Khazaal Farah , COUTURIER Ulysse

Contents

1 Installation of gem5

Gem5 uses Git for version control, which is a distributed version control system. Git is commonly pre-installed on most platforms, including Ubuntu. However, if Git is not installed on Ubuntu, you can easily install it using the command below :

— sudo apt install git

Gem5 relies on SCons as its build environment. SCons can be seen as an enhanced version of make, utilizing Python scripts for all build-related tasks. This design choice provides a highly adaptable build system. To get SCons on Ubuntu use

— sudo apt install scons

Gem5 relies on the Python development libraries. To install these on Ubuntu use

— sudo apt install python3-dev

To clone the repository, use the git clone command.

— sudo git clone https://gem5.googlesource.com/public/gem5

To build a basic x86 system , use the command below.

— python3 ‘which scons‘ build/X86/gem5.opt -j9

2 Debug Files

First, all debug flags are listed when typing the command "./build/ISA/gem5.opt –debug-help. But the ones used in this report are listed bellow.

Moreover, all flags were separately generated, but it is possible to merge them all in one file using a comma between the flags in the gem5 command.

2.1 Config.txt

2.1.1 Generation

This file is automatically created by launching a simulation.
It is called **"Config.ini"** or **"Config.json"**.

2.1.2 Content of the Configuration file

The configuration file provides all the information about the system's components. It is advisable to check it for each simulation to ensure that the device you are simulating is the correct one. This file can also provide information about the management policies of the CPU or cache memories in relation to other components. Additionally, it includes all the timings added by the simulator to create a simulation that closely resembles real-world conditions.

2.1.3 Screens

To illustrate the file, here is a screenshot bellow of the Data Cache configuration. The file shows for example the type of replacement policy,

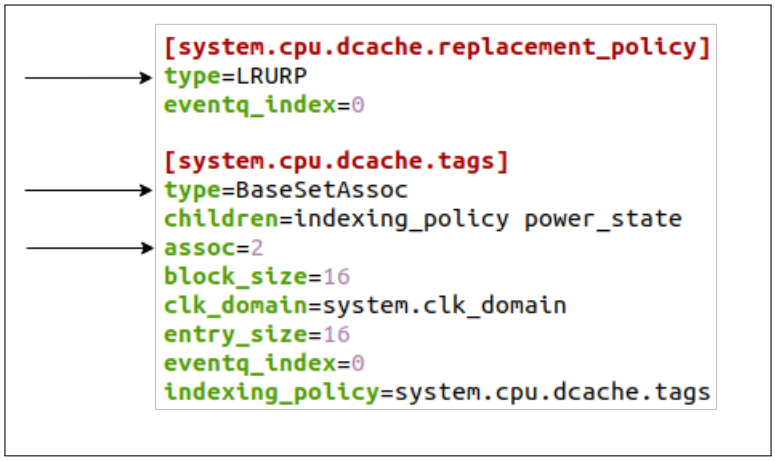

Figure 1 – Screen of the config file

2.2 Statistics

2.2.1 Generation

This file is automatically created by launching a simulation. It is called **"Stats.txt"**.

2.2.2 Contents of the Statistics File

Like the previous one, this file is useful to have a backup of global information on the simulation. It can be use to check the coherence between the desired system and the simulated one.

Details about Level 2 Cache

The Stats.txt file contains a wealth of information about every component. In the case of the L2 shared Cache, there is a significant amount of information that could potentially lead to the discovery of new ways to prevent vulnerabilities.

By analyzing the behavior of the Level 2 cache and how it is shared among CPUs, and by regularly reviewing these statistics, it may be possible to identify new security metrics. A few examples of the provided information are listed in Table below. You can locate them in the file by using "Ctrl+F" and searching for the text in the "Information given" column.

Information given	Description
l2cache.replacements	Number of replacements (Count)
l2cache.overallMisses::total	Number of overall misses
l2cache.overallMissRate::total	Miss rate for overall accesses (Ratio)
l2cache.overallHits::total	Number of overall hits (Count)
l2cache.WriteClean.hits::total	Number of WriteClean/Flush hits
l2cache.tags.avgOccs::total	Average percentage of cache occupancy
l2cache.tags.occupancies::cpu0.inst	Average occupied blocks per tick
l2cache.tags.dataAccesses	Number of data accesses

Table 1 – Example of l2 cache information given by stats

DRAM given information

Stats.txt also provides information about the DRAM, as mentioned earlier. This information has been summarized in Table below.

Information given	Description
dram.bytesRead::cpu0.inst	Number of Inst read from this memory (Byte)
dram.bytesRead::cpu0.data	Number of data read from this memory
dram.bytesWritten::writebacks	Number of bytes wrote after flush
dram.numReads::cpu0.inst	Number of read req responded to by this memory
dram.bwInstRead::cpu0.inst	Inst Read bandwidth from this memory (Byte/Sec)
dram.busUtil	Data bus utilization in percentage (Ratio)

Table 2 – Example of DRAM information given by stats

2.2.3 More details

Moreover, it can offer precise details regarding the number of iterations for a particular function. For instance, in the screenshot below, the file presents information such as the count of simulated instructions or the total number of hits and misses in the Data Cache.

```
simInsts            5908       # Number of instructions simulated (Count)

system.cpu.cpi      11.627116  # CPI: cycles per instruction ((Cycle/Count))

system.cpu.dcache.overallHits::cpu.data          1302     # number of overall hits (Count)
system.cpu.dcache.overallMisses::cpu.data        810      # number of overall misses (Count)
```

Figure 2 – Screen of the stat file

2.3 Execution behavior

2.3.1 Generation

This file is not generated automatically, and it is essential to enter the indicated line below in order to obtain this file.

— ./build/{ISA}/gem5.opt –debug-flag=Exec
–debug-file={NAME_OF_FILE}.out {PYTHON SCRIPT TO SIMULATE}

2.3.2 Contents of the Execution file

This file represents the Execution phase of the pipeline, displaying all the assembly code executed by the CPU, along with additional information such as the tick, the value of the PC (program counter), and more.

2.3.3 Analyzing the Execution File for Assembly Code Insights

To view the assembly code related to your executed binary, use Ctrl+F and type "@main" to quickly navigate to your desired location. In fact, this file (when you don't specify a starting point with "–debug-start") also provides information about assembly code for setting up the system and its registers. Once you're in the main section, you can visualize all instructions along with additional information on the right, including memory writes, memory reads, and interactions with the ALU (Arithmetic and Logic Units).

2.3.4 Screens

screenshot of a single line from this file can be seen below. A description of each piece of information is also provided. These details are generic and always arranged in the same order for all lines. The screenshot, labeled as 3, has been split into two parts for improved readability. The left part is displayed in the first line, while the right part is shown in the second line.

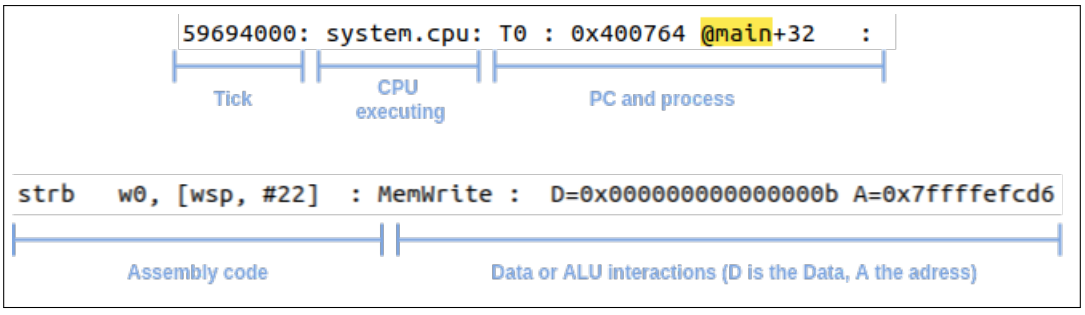

Figure 3 – Screen of the exec file

2.4 Memory Accesses

2.4.1 Generation

This file is not generated automatically, and it is essential to enter the indicated line below to obtain this file.

— ./build/{ISA}/gem5.opt –debug-flag=MemoryAccess –debug-file={NAME_OF_FILE}.out {PYTHON SCRIPT TO SIMULATE}

2.4.2 Content of the Memory Access File

This file displays all read and write accesses to memory, including the access timestamp (tick), the size of the requested data, and its address.

2.4.3 Screens

In the screenshot labeled as 4 below, you can see an example of a request found in this file. It begins with the timestamp (tick) on the left, followed by the purpose of the request, whether it is a read or a write, and lastly, it provides additional information regarding the manipulated data.

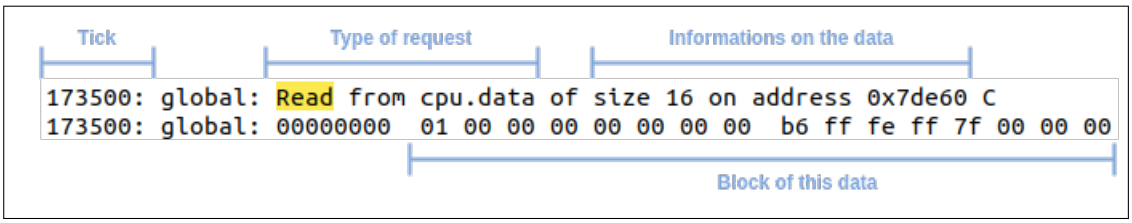

Figure 4 – Screen of the Memory Access file

2.5 MinorMem

2.5.1 Generation

This file is not generated automatically, and it is necessary to input the indicated command line below in order to obtain this file.

— ./build/{ISA}/gem5.opt –debug-flag=MinorMem –debug-file={NAME_OF_FILE}.out {PYTHON SCRIPT TO SIMULATE}

2.5.2 Compatibility

This flag (along with all flags starting with "Minor") is reserved for the Minor CPU because it provides the functionalities required to observe this type of information.

2.5.3 Content of the MinorMem file

MinorMem is similar to MemoryAccess but offers more detailed information. Instead of merely indicating read or write access to memory, it also displays requests made to the TLB (Translation Lookaside Buffer) for translating virtual addresses from the process's stack to physical addresses within the memory system.

2.5.4 Screens

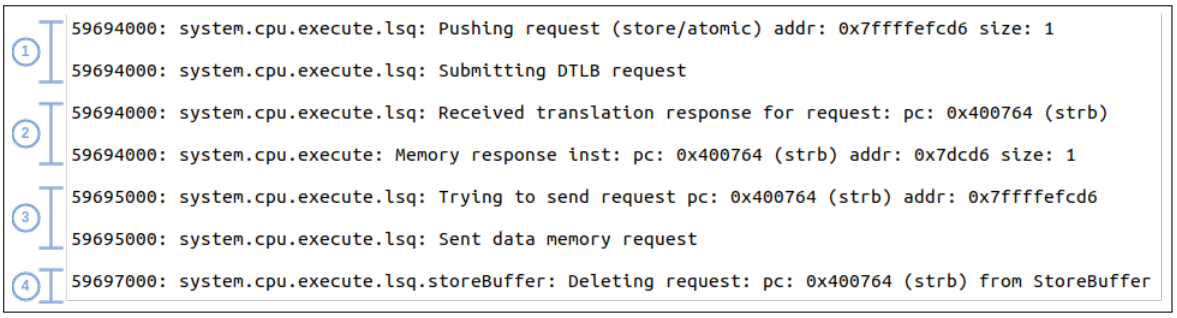

Figure 5 – Screen of the MinorMem file

In this example screenshot from the MinorMem debug file, there are four visible parts indicated by the blue line on the left. The first part involves pushing a request into the Store buffer at the virtual address 0x7ffffefcd6 for a data of size 1 byte, along with a translation request for this address to the TLB. The second part includes the reception of this request and the response address (e.g., the physical address of the data), which is 0x7dcd6. The third part is the transmission of this store request to the memory system, and finally, the last part involves removing this request from the storeBuffer.

2.6 Cache debug

2.6.1 Generation

This file is not generated automatically, and it is essential to enter the line indicated below in order to obtain this file.

— ./build/{ISA}/gem5.opt –debug-flag=Cache –debug-file={NAME_OF_FILE}.out {PYTHON SCRIPT TO SIMULATE}

2.6.2 Compatibility

This flag will only function as intended in a system that includes a cache. Misusing it in a basic system without cache memories will result in an empty file.

2.6.3 Content of the Cache File

As implied by its name, the Cache flag records requests for reading and writing data along with their respective addresses. It also includes information about whether these requests resulted in a hit or miss and provides additional details about the written data, such as its readability, writability, dirtiness, etc.

2.6.4 Screens

The following screenshot displays various write requests, including hits, misses, and a flush. The second part shows a "WriteClean" operation associated with the flush. Additionally, it is evident that the flush affects the entire cache line, from cd0 to cdf (where cache lines consist of 16 bytes). The misses and hits are indicated on the right.

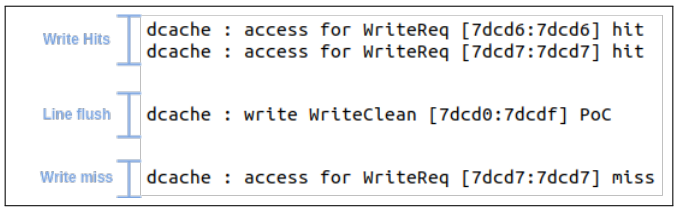

Figure 6 – Screen of the Cache file

The second screenshot labeled as 7 below displays the information located to the right of the previously mentioned "miss." You can also observe the tag and the

set associated with this data. The screenshot has been divided into two parts, with the first part representing the beginning of the line and the second part depicting the end of the line.

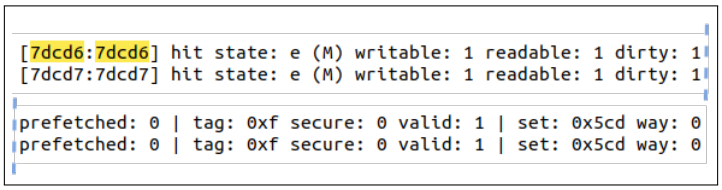

Figure 7 – Second screen of the Cache file