FVLLMONTI

Call: **H2020-FETPROACT-2020-01**

Grant Agreement no. **101016776**

Deliverable D5.7 – Open source release: parameterizable simulator with application examples - V1

Start date of the project: 1st January 2021
Duration: 56 months
Project Coordinator: Cristell MANEUX - University of Bordeaux
Contact: Cristell MANEUX - cristell.maneux@ims-bordeaux.fr

DOCUMENT CLASSIFICATION

Title	Open source release: parameterizable simulator with application examples - V1
Deliverable	D5.7
Estimated Delivery	31/12/2023 (M36)
Date of Delivery Foreseen	31/12/2023 (M36)
Actual Date of Delivery	31/12/2023 (M36)
Authors	Giovanni Ansaloni – P5 – EPFL, Alireza Amishahi – P5 – EPFL
Approver	Cristell Maneux (UBx)
Internal reviewers	Alberto Bosio (ECL), Jean-Luc Rouas (UBx)
Work package	WP5
Dissemination	PU
Version	V1.0
Doc ID Code	D5.7_FVLLMONTI_P5-EPFL-20231231
Keywords	Code release, computer architectures, Full system simulation, Domain-specific acceleration, open-source.

DOCUMENT HISTORY

VERSION	PUBLICATION DATE	CHANGE
0.1	28.11.2023	Initial version from EPFL (G. Ansaloni, A. Amirshahi)
0.2	05.12.2023	First content-complete version (G. Ansaloni, A. Amirshahi)
0.3	06.12.2023	Version sent for internal review (G. Ansaloni, A. Amirshahi)
1.0	14.12.2023	Submitted version (G. Ansaloni, A. Amirshahi, Jean-Luc Rouas)

DOCUMENT ABSTRACT

This deliverable D5.7 illustrates the initial version of the open-source source system simulation framework, developed to support the project activities in FVLLMONTI WP5.

The deliverable covers the following topics:
1 – It describes the simulator components, in particular regarding the N2C2 implementation model, abstracted from the architectural design in WP4.
2 – It illustrates the structure of the repository itself, which comprises the simulator source code, as well as companion elements such as an example application and the technical manual. Furthermore, the deliverable provides information on how to obtain the Linux disk image required for full system simulations.
3 – It presents an overview on the intended framework usage, including information on how to configure virtual systems, compile and run applications, and collect execution statistics.

The system simulation infrastructure is available at https://github.com/gem5-X/TiC-SAT, where it is licensed under the BSD 3-Clause license agreement.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 101016776.

TABLE OF CONTENT

DOCUMENT CLASSIFICATION 2
DOCUMENT HISTORY 2
DOCUMENT ABSTRACT 2
TABLE OF CONTENT 3
LIST OF FIGURES AND TABLES 4
LIST OF ACRONYMS / GLOSSARY 5

1. Introduction 6

2. Repository structure 7

3. Integration of N2C2 accelerators in gem5-X-TiC-SAT 8
_I. Configuring N2C2 latency 9
_II. Configuring N2C2 array size and input/output bitwidths 9
_III. Example transformer Application 10
_IV. Accelerator library 10

4. Exploration outcomes 11

5. Conclusions and Future work 13

6. Annex I: System simulation with gem5-X-TiC-SAT 14
_I. Registration 14
_II. Compiling gem5-X-TiC-SAT 14
_III. Launching a gem5-X-TiC-SAT full system simulation 15

7. Annex II: Support Enhancements in gem5-X 16
_I. Enhanced checkpointing 16
_II. gperf profiler 17
_III. File sharing between gem5-X and host system using 9P over Virtio 17
_IV. Modifying disk image using QEMU 18

LIST OF FIGURES AND TABLES

Figure 1 : gem5-X-TiC-SAT github repository.....8

Figure 2 : Structure of a transformer block (left), with a detailed view of the multi-heads attention layer (right). Layers composed for the most part by GEMM operations are marked by an asterisk (*). 10

Figure 3 : Run-time required to execute a BERT-Large transformer block, considering a non-optimized implementation (N.O.), a software-optimized solution (C-Opt.) and system including N2C2 accelerators of sizes ranging from 4*4 to 16*16. 11

Figure 4: Memory accesses across levels of cache and memory for different system configurations.....12

Table 1 : Transformer benchmarks characteristics and speed-up of 16*16 N2C2 systems with respect to the N.O. baseline 13

LIST OF ACRONYMS / GLOSSARY

AI:	Artificial Intelligence
ASR:	Automatic Speech Recognition
BERT:	Bidirectional Encoder Representations from Transformers
D:	Deliverable
FF:	Feed-Forward
FU:	Functional Unit
GEMM:	GEneral Matrix-to-matrix Multiplication
ISA:	Instruction Set Architecture
M:	Month of the project
MHA:	Multi-Head Attention
MT:	Machine Translation
N2C2:	Neural Network Compute Cube
PU:	Public
RoI:	Region of Interest
ST:	Speech Translation
TiC-SAT:	Tightly-Coupled Systolic array Accelerators for Transformers
ViT:	VisionTransformer
VNWFET:	Vertical Nanowire Field Effect Transistors
WA:	Workload automation
WP:	Work Package

1. Introduction

Full system (FS) simulation allows to model the run-time characteristics of entire hardware/software stacks, comprising memory hierarchies, interconnects and computing resources, as well as operating systems and user-level applications executing on such resources. FS simulations are performed by defining virtual systems via configuration scripts, and profiling their run-time behaviour when executing a target application. Gem5 is the most commonly-used FS simulator, both in academia and industry [1]. The project partner EPFL has enriched it with key extensions to develop the gem5-X environment [2], providing out-of-the-box simulation of ARM-based systems with a full Linux stack, along with several architectural enhancements including ISA extensions, advanced check-pointing, Workload Automation (WA), and gperf profiler support.

Full system simulation activities in WP5 adopted gem5-X as a starting point, developing a further extension, named gem5-X-TiC-SAT[1] [3], which allows to instantiate behavioural models of N2C2 accelerators as custom functional units. Employing this approach, the benefit of FVLLMONTI technological breakthroughs (explored in WP1-2-3), as embodied in the accelerator architecture devised in WP4, can be measured when accelerating the optimized transforms DNN models from T5.3/T5.5. The framework described herein, focus of T5.2 (*"Architecture design"*), hence enables to bridge the innovations pursued by the FVLLMONTI consortium across the hardware/software divide, while also being instrumental for the *"Run-time optimization strategies"* pursued in T5.4.

Furthermore, beyond enabling the exploration of the benefit of FVLLMONTI technologies and architectures from a system level perspective, the system simulator is an important accomplishment in its own right, as it allows the evaluation of dedicated tightly coupled accelerator models when executing key AI applications.

This D5.7 deliverable details the first version of the gem5-X-TiC-SAT framework implementation[2]. It provides a description of the repository structure in Section 2. Its various aspects are discussed in further sections: Section 3 illustrates how N2C2 accelerators can be configured and interfaced using gem5-X-TiC-SAT, and how applications (including the provided transformer example) can be simulated. Section 4 showcases the outcomes of the performed system-level explorations, elaborating on the initial ones reported in D5.3. These results are also reported in a peer-reviewed scientific paper presented at the ACM/IEEE ASP-DAC conference [3]. Section 5 provides concluding remarks and perspectives for ongoing and future works. Finally, two annexes are included, detailing how the gem5-X-TiC-SAT can be set-up, and the main architectural enhancement brought by gem5-X.

[1] TiC-SAT is named after *"Tightly-Coupled Systolic array Accelerators for Transformers"*, which is the N2C2 embodiment in our work *Amirshahi et al., 2023* [3].

[2] A further deliverable at month M56 will report on the gem5-X-TiC-SAT capabilities at the end of the project.

2. Repository structure

The full system simulation source code is available as a public repository on github: https://github.com/gem5-X/TiC-SAT. It is released under the BSD-3clause license [4], which allows re-distribution of the code, with or without modifications, as soon as the original authorship by the involved consortium partners is acknowledged.

Included in the repository are:

- A README file providing quick-start instructions on using gem5-TiC-SAT, including the link to the main gem5-X website [5], where users can register to download the gem5-X disk image (itself containing a Ubuntu16.04 Linux operating system).
- A complete release of the gem5-X-TIC-SAT simulator. This comprises utilities for the full system simulation of single and multi-core systems, provided by gem5. In addition, it offers the enhanced checkpointing, profiling and file-sharing capabilities supported by gem5-X, described in Section 7. Finally, it supports a novel system component developed in T5.2, which models the execution of N2C2 arrays as tightly coupled functional units, governed by extensions to the ARM ISA, as detailed in Sections 3.I and 3.II.
- The parametric implementation of a Transformer block, including Projection, Normalization, FeedForward and Attention layers. A software library is also provided to map the application GEMM operations either as tiled or non-tiled loop in software, or as N2C2-accelerated primitives. The Transformer and the library code are described in 3.III and 3.IV.
- A technical manual comprehensively detailing the code release.

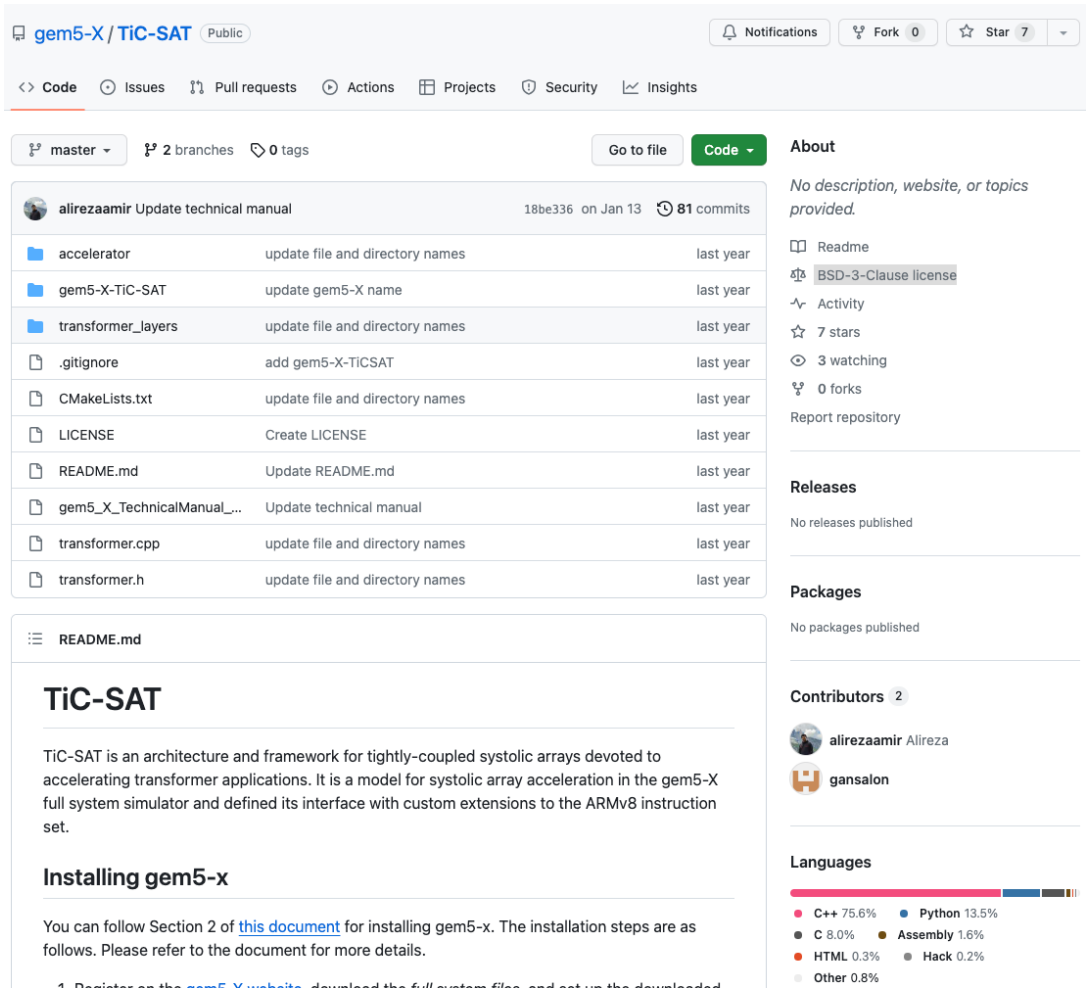

Figure 1 : gem5-X-TiC-SAT github repository.

Not part of the repository are the Ubuntu Linux disk image, the Linux kernel and the bootloader required for system simulation. These are obtained by registering as a gem5-X user at the main gem5-X website [5]. Instructions about the registration process are provided in the repository README file.

3. Integration of N2C2 accelerators in gem5-X-TiC-SAT

gem5-X-TiC-SAT supports the instantiation of system modules modelling N2C2 accelerators. These are integrated in a tightly-coupled way, that is, as dedicated functional units governed by custom instructions extending the ARMv8 ISA.

The structure of the N2C2 component, and the format of the ISA extensions, has been illustrated in D5.3. Herein, we instead detailed how N2C2 components are configured and instantiated.

I. CONFIGURING N2C2 LATENCY

The latencies associated with the N2C2 custom instructions are defined in the processor configuration file, as the N2C2 itself is connected as a functional unit in the processor pipeline. The latency is indicated in terms of CPU clock cycles. An excerpt of the configuration file, reporting the N2C2 ISA extensions, is presented below:

```
class MinorDefaultCusProcessFU(MinorFU):
  <...>
    opLat = 1

class MinorDefaultCusQueueFU(MinorFU):
    <...>
    opLat = 1

class MinorDefaultCusParamWriteFU(MinorFU):
    <...>
    opLat = 1
```

The three operations (`Process`, `Queue` and `ParamWrite`) map to the *SA_IOC*, *SA_IO* and *SA_LD* ISA extension defined in D5.3 as custom instructions, extending the capability of the simulated processor core(s). The instructions allow to activate the N2C2 computation while performing an IO transfer (*SA_IOC*), to perform an IO without activating the N2C2 (*SA_IO*) and to program the N2C2 weight values (*SA_LD*).

In the example above, all operations related to the N2C2 custom Functional Unit (FU) have a latency of 1 CPU clock cycle. These values can nonetheless be easily changed to abide to the architectural characterization effort in WP4 and/or technology considerations from WP1/2/3.

II. CONFIGURING N2C2 ARRAY SIZE AND INPUT/OUTPUT BITWIDTHS

The systolic array size can be configured parametrically in the `src/dev/arm/systolic_m2m.hh` file by editing the related pragmas:

```
#define KERNEL_DIM 8

#define W_DATA 4
```

In this example, the systolic array size (kernel size) is assigned to 8, creating a model for an 8*8 N2C2 accelerator.

In the default N2C2 configuration, it is assumed that all the data are represented in 8 bits. This default can be modified in the modify `src/dev/arm/systolic_m2m.hh` and the `src/dev/arm/systolic_m2m.cc` files. In the first file, the parameter `W_DATA` (see the code snippet above) should be changed as desired. This parameter indicates how many values can fit in the 32-bit bus width. Note that, depending on the chosen bitwidth, variables declared in this file should also be changed accordingly, e.g. from `int8_t` to `int16_t`.

Finally, the shifting index in `src/dev/arm/systolic_m2m.cc` (lines 54, 62, 71, 84, 115, 122) should be adjusted to abide to the data width, in order to correctly pack values on the 32-bit system bus. For instance, to process 16-bits inputs and compute 16-bits outputs, line 54 must be as follows:

```
auto currVal = (int16_t)((val >> (16 * (W_DATA -i-1))) & 0xffff);
```

III. EXAMPLE TRANSFORMER APPLICATION

The repository provides the C++ code implementing a transformer block as an example application. The main files are in the repository roots (`transformer.cpp` and `transformer.h`). The latter allows to define the transformer block characteristics, such as the number of attention heads, their size, and the size of the feed-forward layers.

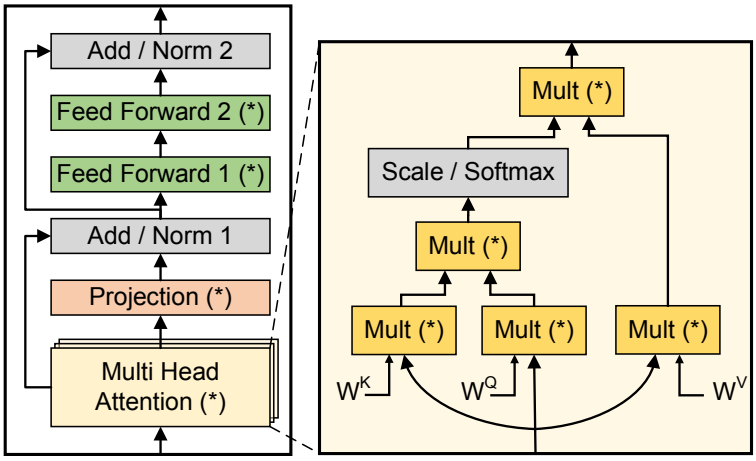

Figure 2 : Structure of a transformer block (left), with a detailed view of the multi-heads attention layer (right). Layers composed for the most part by GEMM operations are marked by an asterisk (*).

The provided code can be cross-compiled in the host system:

```
arm-linux-gnueabi-g++ transformer.cpp -o transformer.o
```

Afterwards the compiled executable file can be moved to the shared host/guest folder (see Section 7) and launched as a standard Linux application:

```
./transformer.o
```

gem5-X system calls are used to extract timing and memory statistics from a section of the application, they can be added them to the application code in order to delimit a Region of Interest (RoI):

```
<...other code...>
system("m5 resetstats");
   < ... Region of Interest ... >
system("m5 dumpresetstats");
<...other code...>
```

Multiple RoIs can be defined, resulting in the generation of multiple statistic files for every simulation run.

IV. ACCELERATOR LIBRARY

The provided application code relies on library functions to implement transform layers either in software, or leveraging N2C2 acceleration. These functions are declared in the repository file `accelerator/smm_gemm.h` and implemented in `accelerator/smm_gemm.cpp`. They are listed below:

- `conventionalCompute()` performs standard GEMM operations using triple-nested loops.
- `tiledCompute()` also provides a software-only implementation, but implements tiling to increase data locality.
- `smmCompute()` instead performs calls to the modelled N2C2 hardware accelerator to accelerate the computation of GEMMs.

By default, the provided transformer application employs the naïve `conventionalCompute()` strategy. More advanced ones can be derived by modifying calls to `conventionalCompute()` to that of a different library function.

4. Exploration outcomes

In the context of the activities in Task 5.4, we performed a detailed exploration of system performance metrics, under different configurations, when executing a block of the BERT-large transformer model [6]. Experiments are carried out on a system with in-order CPU core clocked at 1 GHz, 32 KB of L1 data and instruction cache and 1MB of L2 cache size. The main memory is a 4GB DDR4.

Results are shown for the naïve and tiled GEMM software-only implementations described in Section 3.IV, as well as for systems implementing N2C2 accelerators of different sizes ranging from 4*4 to 16*16. In each case, stacked bars detail the execution time for every layer inside the transformer block. The same figure also displays the corresponding speed-ups with respect to the non-optimized baseline.

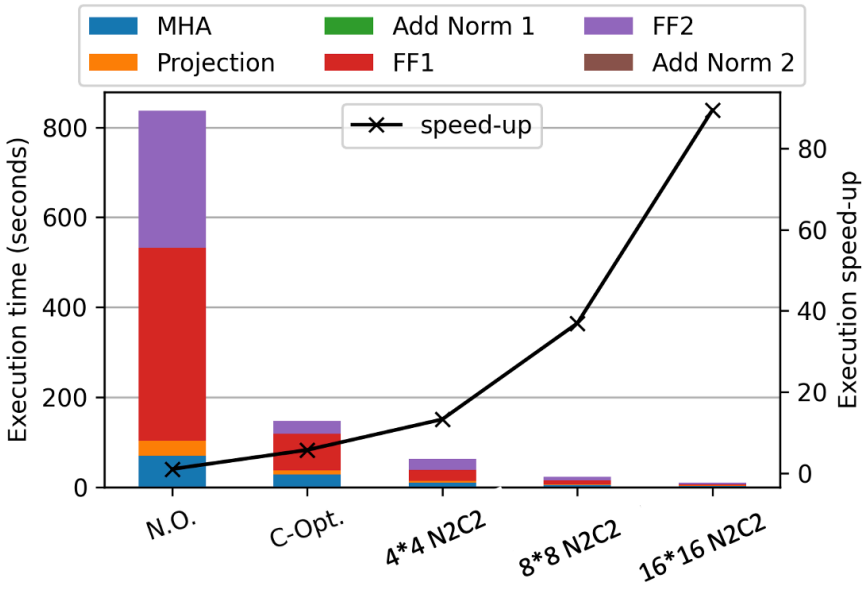

Figure 3 : Run-time required to execute a BERT-Large transformer block, considering a non-optimized implementation (N.O.), a software-optimized solution (C-Opt.) and system including N2C2 accelerators of sizes ranging from 4*4 to 16*16.

The graphs highlight that N2C2 accelerators can effectively speed-up even highly optimized software baselines. When considering a small 4*4 N2C2, an additional speed-up exceeding 2X (with respect to the optimally-tiled C-Opt. implementation) is obtained. Further reductions in run-time are achieved when increasing the array size.

Figure 3 also shows that the most time-consuming layers in a transformer block are the feed-forward (FF) ones. These layers almost entirely consist of GEMM operations, and hence can be accelerated by employing N2C2s. GEMMs are not nonetheless exclusive to feed-forward layers. Indeed, the only layers that do not comprise GEMMs are scaling and normalization layers. Non-GEMM layers are not, however, the computational bottlenecks of transformers not exceeding 3.1% of the execution time across all configurations.

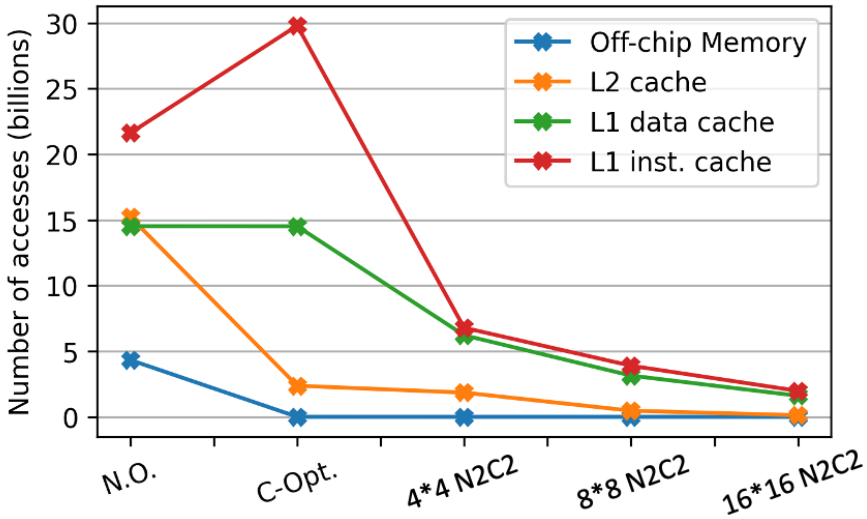

Figure 4: Memory accesses across levels of cache and memory for different system configurations.

N2C2s increase performance both by parallelizing computation, and by increasing the locality of memory accesses, hence lowering the burden on the cache hierarchy. Focusing on this latter aspect, Figure 4 shows the number of accesses in different memory hierarchy levels for different baselines and N2C2-based instances, again for the BERT-Large case. As shown in this figure, L2 cache (and main memory) accesses are reduced by applying software optimization. Such benefit comes at the cost of an increase in L1 instruction accesses, as tiling introduces more complex loop structures, hence more instructions are executed.

When N2C2 acceleration is used to execute the transformer block, accesses across the memory hierarchy are drastically decreased. The reason for this behavior is two-fold: first, a single N2C2 invocation via the SA_IOC custom instruction triggers k*k MAC operations, hence lowering the number of required instructions. Second, k*k parameters are resident during computation in the SA, and must not be re-read at each use, thus reducing the size of the working set. Both effects become more prominent for larger N2C2 sizes.

Finally, Table 1 reports the benchmark characteristics and the speed-up in ten different transformer applications. BERT transformers are sized according to [7], while VisionTransformer (ViT) size are taken from [8]. The speed-up refers to the executions of the entire block with respect to non-optimized baselines. Results show that double-digit (up to 89.5X) speed-ups are achieved in all applications when a 16*16 N2C2 accelerator is employed. Such outcomes originate from a combination of software optimization and hardware acceleration. They highlight the benefit of employing tightly-coupled acceleration, as it can

effectively leverage data reuse in the memory hierarchy, in the presence of both accelerated as well as non-accelerated layers.

Table 1 : Transformer benchmarks characteristics and speed-up of 16*16 N2C2 systems with respect to the N.O. baseline.

Model	Number of parameters	Speedup
ViT-base/16	$86 * 10^6$	69.4
ViT-base/32	$86 * 10^6$	48.8
ViT-large/16	$307 * 10^6$	82.5
ViT-base/32	$307 * 10^6$	57.2
ViT-huge/14	$632 * 10^6$	82.7
BERT-tiny	$4* 10^6$	20.3
BERT-mini	$11 * 10^6$	38.2
BERT-medium	$41 * 10^6$	58.3
BERT-base	$110 * 10^6$	69.3
BERT-large	$340 * 10^6$	89.5

5. Conclusions and Future works

This deliverable presented the system structure and capabilities of the system simulator developed and employed to perform architectural simulation in WP5. In particular, it illustrates how it enables the assessment of the performance of accelerator solutions developed in WP4, and algorithmic optimization strategies for the automated speech recognition and machine translation pursued in T5.3/T5.5.

Detailed exploration results confirm the preliminary data published in D5.3: N2C2 accelerators modelled by the gem5-X-TiC-SAT simulator induce double-digit speedups (up to 89.5X with respect to a baseline implementation of the BERT-Large transformer) by accelerating the critical GEMM computational kernels.

Such result will offer a springboard towards further exploration, which will be at the core of WP5 activities for the remainder of the project. On one side, we plan to incorporate the characterization (performed in WP4) of the VNWFET vertical nanowire transistors in the design of the N2C2 accelerator, modelling the ensuing performance/efficiency benefits from a system level perspective. On the other, we will leverage algorithmic-level opportunity (such as sparsity), in order to reduce the computational cost of transformers dedicated to ASR and MT.

6. Annex I: System simulation with gem5-X-TiC-SAT

I. REGISTRATION

gem5-X users register at https://esl.epfl.ch/gem5-x. Upon registering, they receive a mail with a link to download an archive containing:

• A system bootloader

• A kernel binary (vmlinux)

• A disk image (including the Ubuntu16.04 operating system)

The archive is decompressed with the following shell command:

```
tar –zxvf fullsystemimages.tar.gz
```

The archive structure is as follows:

• The bootloader and kernel are in the `/binaries/` directory

• The disk image (`gem5_ubuntu16.img`) can be found at the `/disks/` directory

The following sets up the path to full system images, so that the files under it can be used and recognized by gem5-X-TiC-SAT during FS simulation.

```
cd <path_to_gem5–X–TiC–SAT>
./apply–patch . sh <path_to_full_system_images>
```

The full system files are now set up and ready to be used in FS mode.

II. COMPILING GEM5-X-TIC-SAT

After cloning the gem5-X-TiC-SAT repository from `https://github.com/gem5–X/TiC–SAT` , the device tree can be generated. Source files are in the following directory:

```
<path_to_gem5–X–TiC–SAT>/system/arm/dt
```

If running on an Ubuntu-based host system, prerequisites must be installed before generating the device tree binaries:

```
sudo apt–get install gcc–arm–linux–gnueabihf gcc–aarch64–linux–gnu
sudo apt–get install device–tree–compiler
```

The device tree binary files are then generated as follows:

```
cd <path_to_gem5–X–TiC–SAT>
make –C system/arm/dt
```

Once the device tree is generated, the simulation environment must be set up in order to compile and run the simulator executable i.e., the virtual machine simulating the configured architecture. To this end, a SCons (SConstruct) builder is provided. It is run as follows:

```
sudo apt install build-essential git m4 scons zlib1g \
zlib1g-dev libprotobuf-dev protobuf-compiler libprotoc-dev \
libgoogle-perftools-dev python-dev python-six python \
libboost-all-dev swig
```

Once the above is done, the ARM gem5 binary can be built. Multiple builds can be generated, including `.fast`, `.opt`, and `.debug`. `gem5.fast` is the preferred choice if the environment is only used to run experiments. However, if debug capabilities or trace generation is needed, these can be accessed in `gem5.debug` or `gem5.opt`. The corresponding commands are as follows:

```
cd <path_to_gem5-X-TiC-SAT>
scons build/ARM/gem5.{fast, opt, debug}
```

Additionally, the compilation process can be sped up by using the option `-jN` on the Scons build line, where `N` is the number of threads assigned for compilation.

III. LAUNCHING A GEM5-X-TIC-SAT FULL SYSTEM SIMULATION

Once the build process is complete, simulations can be launched with the following commands:

```
cd <path_to_gem5-X-TiC-SAT>

./build/ARM/gem5.{fast, opt, debug} \
--remote-gdb-port=0 \
-d /path/to/your/output/directory \
configs/example/fs.py \
--cpu-clock=1.6GHz \
--kernel=vmlinux \
--machine-type=VExpress_GEM5_V1 \
--dtb-file=<full_path_to_gem5-X-TiC-SAT>/system/arm/dt/armv8_gem5_v1_1cpu.dtb \
-n 1  \
--disk-image=gem5_ubuntu16.img  \
--caches  \
--l2cache  \
--l1i_size=32kB \
--l1d_size=32kB  \
--l2_size=1MB  \
--l2_assoc=2  \
--mem-type=DDR4_2400_4x16  \
--mem-ranks=4  \
--mem-size=4GB  \
--sys-clock=1000MHz \
--cpu-type=MinorCPU
```

In the example above, a single-core system (indicated by the option "`-n 1`") with a clock CPU clock frequency of 1.6GHz and a system clock frequency of 1 GHz is generated. It presents separate instructions and data caches of 32kB each, and a unified L2 cache of 1MB, 2-ways set associative. Mem memory is a DDR4 DRAM of 4GB. After bootloading is complete, the system starts the "`gem5_ubuntu16.img`" disk image.

The running gem5 instance can be interfaced by telnet using another terminal, as-if connecting to a remote machine. The port “3456” is used by default.

```
telnet localhost 3456
```

Alternatively, the terminal program provided with gem5-X can be used (after building it).

```
cd <path_to_gem5-X-TiC-SAT>/util/term/
make
m5term 127.0.0.1 3456
```

Upon connecting to your gem5 instance, users are able to see the startup messages from the virtual machine boot, followed by the Ubuntu login request.

7. Annex II: Support Enhancements in gem5-X

gem5-X (and, in turn, gem5-X-TiC-SAT), provide additional features with respect to the standard gem5 distribution. They are described in this section.

I. ENHANCED CHECKPOINTING

The boot process during the FS simulation in gem5-X automatically takes a checkpoint when the boot and login is complete. Usually, booting is not part of the region of interest of a simulation. It can hence be performed with a faster, but performance-inaccurate, CPU model such as SimpleAtomicCPU.

After the region of interest is reached more complex CPU models can be adopted, such as the built-in gem5 in-order or our-of-order (OoO) models. To do so, the simulation should be exited first with the following command from the terminal:

```
m5 exit
```

Then, the automatically-taken checkpoint can be used, while switching the CPU model as needed:

```
./build/ARM/gem5.{fast, opt, debug} \
--remote-gdb-port=0 \
-d /path/to/your/output/directory \
configs/example/fs.py \
--cpu-clock=1GHz \
--kernel=vmlinux \
--machine-type=VExpress_GEM5_V1 \
--dtb-file=<full_path_to_gem5-X-TiC-SAT>/system/arm/dt/armv8_gem5_v1_1cpu.dtb \
-n 1  \
--disk-image=gem5_ubuntu16.img  \
--caches  \
--l2cache  \
--l1i_size=32kB \
--l1d_size=32kB  \
--l2_size=1MB  \
--l2_assoc=2  \
```

```
--mem-type=DDR4_2400_4x16  \
--mem-ranks=4  \
--mem-size=4GB  \
--sys-clock=1600MHz \
-r 1 \
--cpu-type={MinorCPU, DerivO3CPU}
```

In the command above, the number after "–r" is the checkpoint number. In this case we are resuming from the first checkpoint. The CPU type can be MinorCPU for in-order core or DerivO3CPU for OoO cores.

Further checkpoints can be taken, e.g. just before reaching a Region Of Interest (ROI) to be profiled in an application. This can be done from the command line or in a script, using the following command:

```
m5 checkpoint
```

Or, inside C/C++ program, by invoking the "m5 checkpoint" system call:

```
system("m5 checkpoint");
```

II. GPERF PROFILER

gem5-x supports the gperf profiling tool [9] within simulated systems. The gperf statistical profiler induces minimal overhead, enabling the identification of application bottlenecks and exploration of the effectiveness of architectural modifications and extensions.

The following commands:

```
LD_PRELOAD=/usr/lib/libprofiler.so.0 /
CPUPROFILE=<FILE_TO_SAVE_PROFILING> CPUFREQUENCY=1000 <program>
```

Launches the "<program>" to be profiled with profiling data being saved to file mentioned in the "CPUPROFILE" parameter.

III. FILE SHARING BETWEEN GEM5-X AND HOST SYSTEM USING 9P OVER VIRTIO

Gem5-X utilize the 9P protocol developed by Bell Labs [10] over a Virtio [11] device driver to allow fast modification of files without modifying the root file system. Once Linux is booted, a folder on the host machine can be mounted within the virtual machine to access files on the host machine file system.

9P over virtio is enabled as follows:

a) The Diod library must be installed

```
sudo apt-get install diod
```

b) Then, gem-X should be compiled as usual, with the following commands:

```
cd <path_to_gem5-X-TiC-SAT>/
scons build/ARM/gem5.{fast, opt, debug}
```

c) The Linux kernel "vmlinux_wa" should be used for simulations. This file is provided with the gem5-X archive including, which include the disk image, under "full_system_images/binaries".

d) An additional parameter must be used when launching simulations

```
workload-automation-vio=<FULL_PATH_TO_SHARED_FOLDER_ON_HOST_SYSTEM>
```

e) Once the system is booted, the following must be executed in the gem5 terminal

```
mount.sh <FULL_PATH_TO_SHARED_FOLDER_ON_HOST_SYSTEM>
```

After these steps, a shared directory is created on the host ad guest (gem5 virtual machine) system. Any file under the "SHARED_FOLDER_ON_HOST_SYSTEM" in the host will also appear in the "/mnt" directory in the gem5 simulation, and vice versa.

IV. MODIFYING DISK IMAGE USING QEMU

gem5-X allows to run disk images using QEMU [12], a lightweight functional simulator. While QEMU simulations do not provide reliable performance exploration, they are much faster than gem5/gem5-X ones. Hence, they can be used to modify a disk image as needed, for example for installing additional libraries etc., in advance of detailed performance exploration.

The modified disk image can then be booted in gem5-X as usual. To this end, if running on an Ubuntu-based host system, the following prerequisites need to be installed.

```
sudo apt-get install qemu qemu-user qemu-system qemu-user-static
```

Then, the image should be mounted

```
cd <PATH_TO_FULL_SYSTEM_IMAGES>/disks/
mkdir local_mnt
sudo mount -o loop,offset=$((2048*512)) gem5_ubuntu16.img local_mnt
sudo mount -o bind /proc local_mnt/proc
sudo mount -o bind /dev local_mnt/dev
sudo mount -o bind /dev/pts local_mnt/dev/pts
sudo mount -o bind /sys local_mnt/sys
```

Users can than chroot into the image emulating it using QEMU

```
cd local_mnt/
sudo chroot ./
```

These commands allow to access the ARMv8 disk image as a simulated environment, e.g. to compile or download applications within the image. When the disk image has been updated as desired, the QEMU simulation can be exited and the image can be unmounted as follows:

```
exit
cd ..
sudo umount local_mnt/proc
sudo umount local_mnt/dev/pts
sudo umount local_mnt/dev
sudo umount local_mnt/sys
sudo umount local_mnt
```

REFERENCES

[1] Butko, Anastasiia, et al. "Accuracy evaluation of gem5 simulator system." *7th International workshop on reconfigurable and communication-centric systems-on-chip (ReCoSoC)*. IEEE, 2012.

[2] Qureshi, Yasir Mahmood, et al. "Gem5-X: A Gem5-based system level simulation framework to optimize many-core platforms." *2019 Spring Simulation Conference (SpringSim)*. IEEE, 2019.

[4] The 3-Clause BSD License : https://opensource.org/license/bsd-3-clause/.

[5] gem5-X: gem5 made easy. Available online: https://www.epfl.ch/labs/esl/research/full-system-simulation-and-design/gem5-x/ .

[6] Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." *arXiv preprint arXiv:1810.04805* (2018).

[7] Turc, Iulia, et al. "Well-read students learn better: On the importance of pre-training compact models." *arXiv preprint arXiv:1908.08962* (2019).

[8] Dosovitskiy, Alexey, et al. "An image is worth 16x16 words: Transformers for image recognition at scale." *arXiv preprint arXiv:2010.11929* (2020).

[9] gperf profiler. Available online : https://gperftools.github.io/gperftools/cpuprofile.html .

[10] Plan 9 from Bell labs. Available Online: https://9p.io/plan9/about.html .

[11] Virtio. Available online: https://wiki.libvirt.org/Virtio.html#virtio .

[12] Bellard, Fabrice. "QEMU, a fast and portable dynamic translator." *USENIX annual technical conference, FREENIX Track*. Vol. 41. 2005.