Virtual Parallel Platform for performance Estimation

VIPPE vs GEM5 Comparison

September, 2020

Micro-electronic Engineering Group

TEISA Department

University of Cantabria

http://vippe.teisa.unican.es

VIPPE is a complex tool infrastructure which integrates several techniques and parts. The integration of these techniques and development of these parts has been development across several projects by the Spanish Government and the EU Commission through several research programs.

IPT-2012-0847-430000

ART-010000-2012-5

HARP

Index

1 Introduction

1.1 What's Vippe?

VIPPE is a tool infrastructure for simulation and performance estimation of complex, heterogeneous MPSoC. VIPPE allows the designer to simulate and application composed of different pieces of functionality (written in C or C++) according to a model of the target platform and of how the application functionality is mapped on such platform.

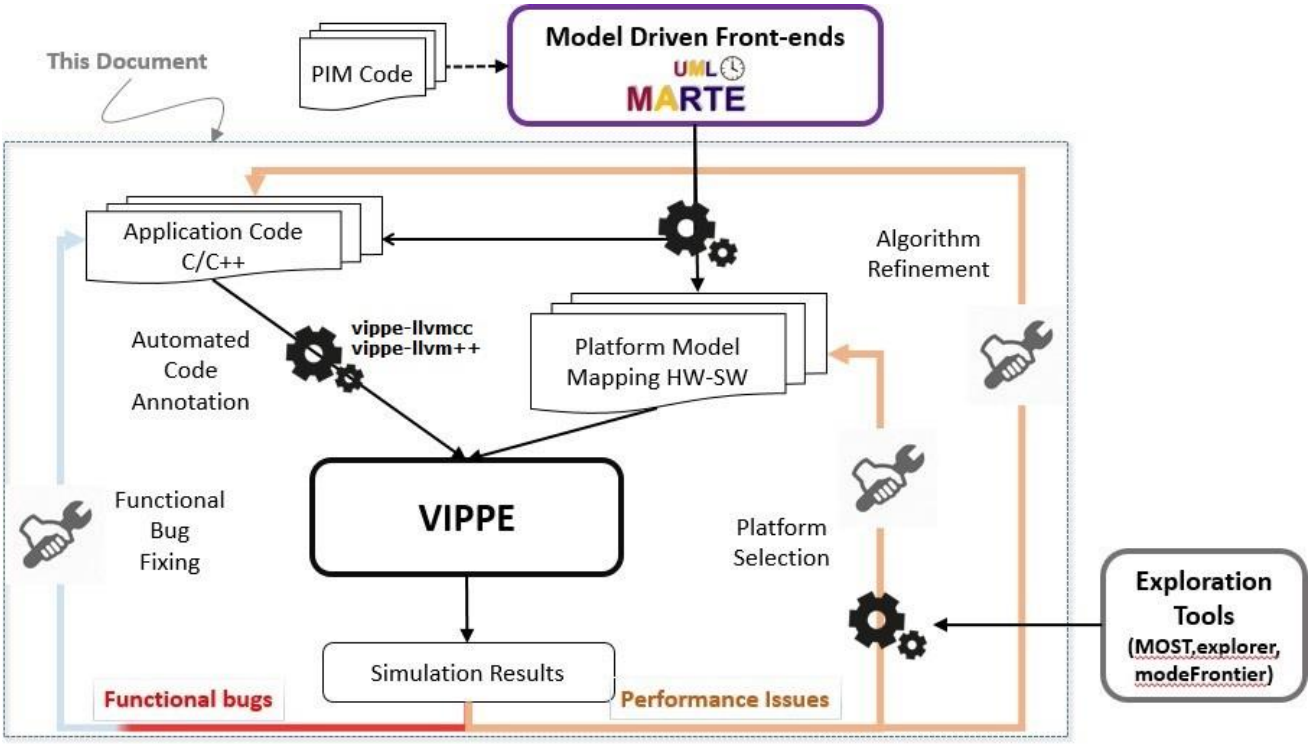


Figure 1. VIPPE enables fast functional and performance assessment of the application targeted to a specific platform.

1.2 What's Gem5?

GEM5 is a modular discrete event driven computer system simulator platform. That means:

1. gem5's components can be rearranged, parameterized, extended or replaced to suit our needs.
2. It simulates the passing of time as a series of discrete events.
3. Its intended use is to simulate one or more computer systems in various ways.
4. It's more than just a simulator; it's a simulator platform that lets you use as many of its premade components as you want to build up your own simulation system.

GEM5 is written primarily in C++ and python and most components are provided under a BSD style license. It can simulate a complete system with devices and an operating system in full system mode (FS mode), or user space only programs where system services are provided directly by the simulator in syscall emulation mode (SE mode). There are varying levels of support for executing Alpha, ARM, MIPS, Power, SPARC, RISC-V, and 64 bit x86 binaries on CPU models including two simple single CPI models, an out of order model, and an in order pipelined model. A memory system can be flexibly built out of caches and crossbars or the Ruby simulator which provides even more flexible memory system modelling.

There are many components and features not mentioned here, but from just this partial list it should be obvious that gem5 is a sophisticated and capable simulation platform. Even with all gem5 can do today, active development continues through the support of individuals and some companies, and new features are added and existing features improved on a regular basis.

1.3 Benchmarks Tests

We use two classes of most common sorting algorithms to compare benchmarks results. The O(n2) complexity Bubble, Insertion, Selection sorts, and the O(n log n) complexity Heap, Merge and Quick sorts. Also, we test some non-sorting algorithms like prime numbers finder, matrix power normalization, eight-queens chess problem, duplicated random numbers puzzle and a recursive iteration problem solver. All this tests are stored in *$VIPPE_HOME/examples/benchmarks* directory.

Simulations were executed on a virtual machine running Ubuntu 18.04. The host machine runs on an Intel Core i5-8500 3GHz with 16GB RAM.

2 ARM Benchmarks

2.1 ARM Simulated Model

Parameter	Simulated System
ISA	ARM v7-a
CPU	Cortex - a8
CPU frequency	1 GHz
Num. of cores	1
Main memory size	500 MB
L1 data cache size	32 kB
L1 instruction cache size	32 kB

Table 1. Simulated system parameters

2.2 About Gem5 Test for ARM

In order to use Gem5 in terms of performance assessment, we use the OPT binary provided. This binary is built with most optimizations on (e.g., -O2), but with debug symbols included. This binary is much faster than debug, but still contains enough debug information to be able to debug most problems. As we test a binary file, gem5 will simulate user space only program where system services are provided directly by the simulator in *syscall emulation mode* (SE mode).

The test will be performance under ARM architecture simulation. Overtime ARM has become the industry standard for low-power processing and driven innovation throughout the industry. The result today is easily seen in the billions of ARM-powered products that service nearly every facet of the industry.

The binary file we will simulate is previously cross compiled with Clang 3.5 with target **armv7-linux-gnueabihf** and statically linked as Gem5 simulator pre-requisite.

For instance, in bubbleSort example, we get gem5 input binary typing in terminal:

- clang++ -Wall --target=arm-arm-none-eabi -march=armv7-a -m32 -fPIC -O2 -static bubbleSort.cpp -o bubbleSort.arm

A rapid sight of *BubbleSort.arm*, typing < *file Bubble.arm* > in linux console:

- *file Bubble.arm*

ELF 32-bit LSB executable, ARM, EABI5 ver.1 (GNU/Linux), statically linked, for GNU/Linux 3.2.0

VIPPE vs GEM5

From <*GEM5 INSTALL DIR*>*/examples*, we can run gem5 simulation with this:

- *../build/ARM/gem5.opt ../configs/example/se.py --cpu-type=O3_ARM_v7a_3 --num-cpus=1 --cpu-clock=1GHz --mem-type=SimpleMemory --mem-size=512MB --caches --cacheline_size=32 --l1d_size=32kB --l1i_size=32kB --l1i_assoc=1 --l1d_assoc=1 -c bubbleSort.arm*

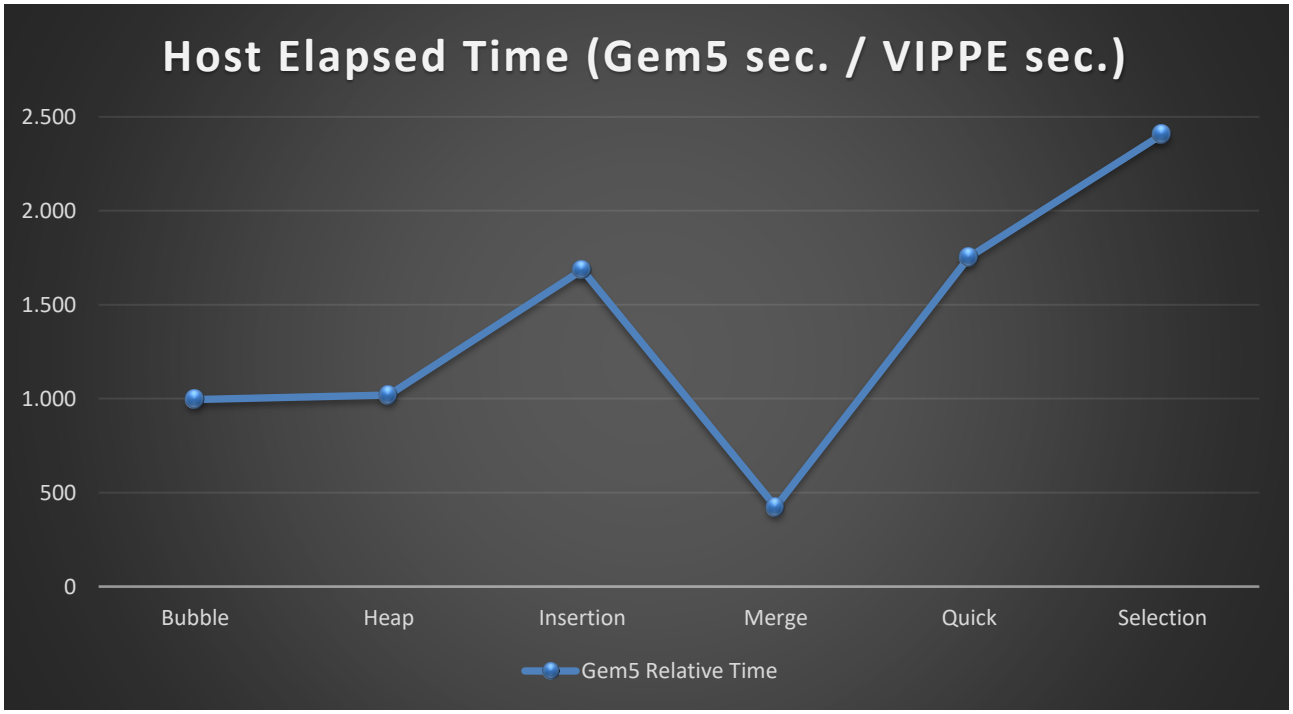
2.3 About VIPPE Test for ARM

To see a detailed explanation of VIPPE usage, please read the VIPPE manual. For this benchmarks, we'll just use a simple usage way:

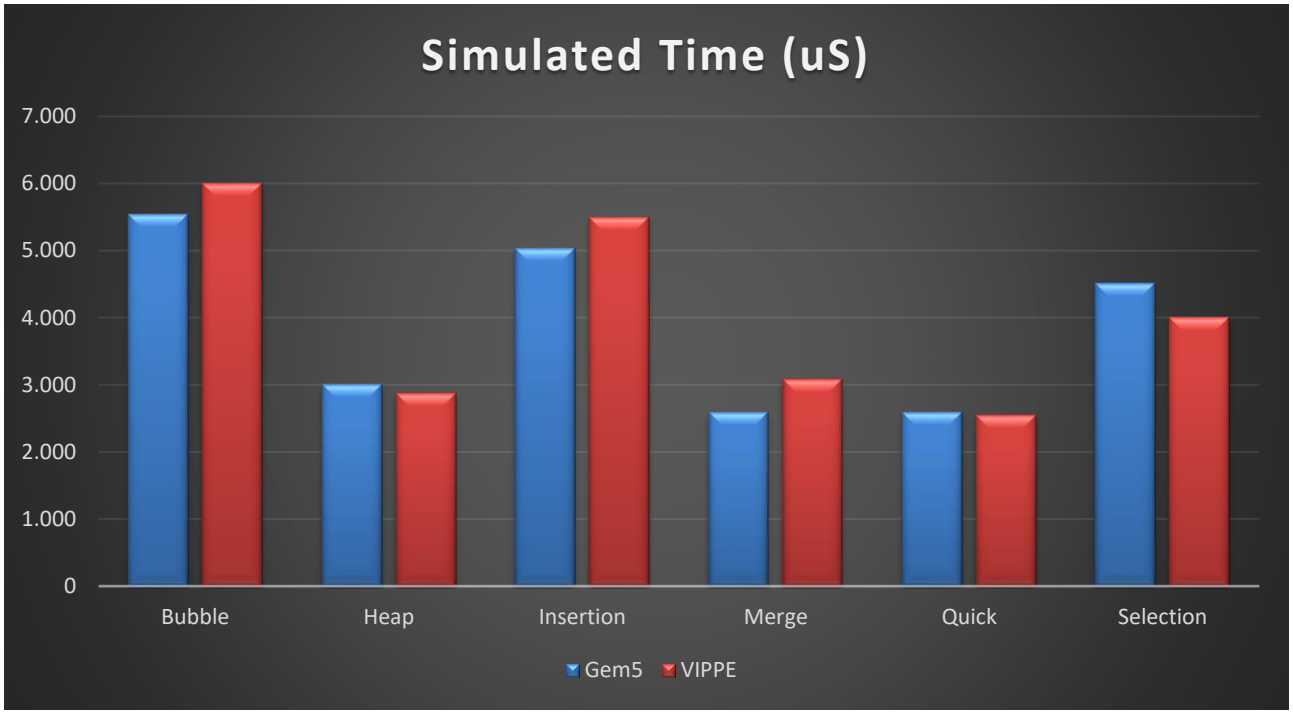
- vippe-llvm++ [-O2] --vippe-verbose --vippe-asm --vippe-cpu=armv7a example.cpp -o example.o
- clang++ [LINKER_FLAGS] example.o -o example.x [INCLUDES] [LIBS]
- ./example.x

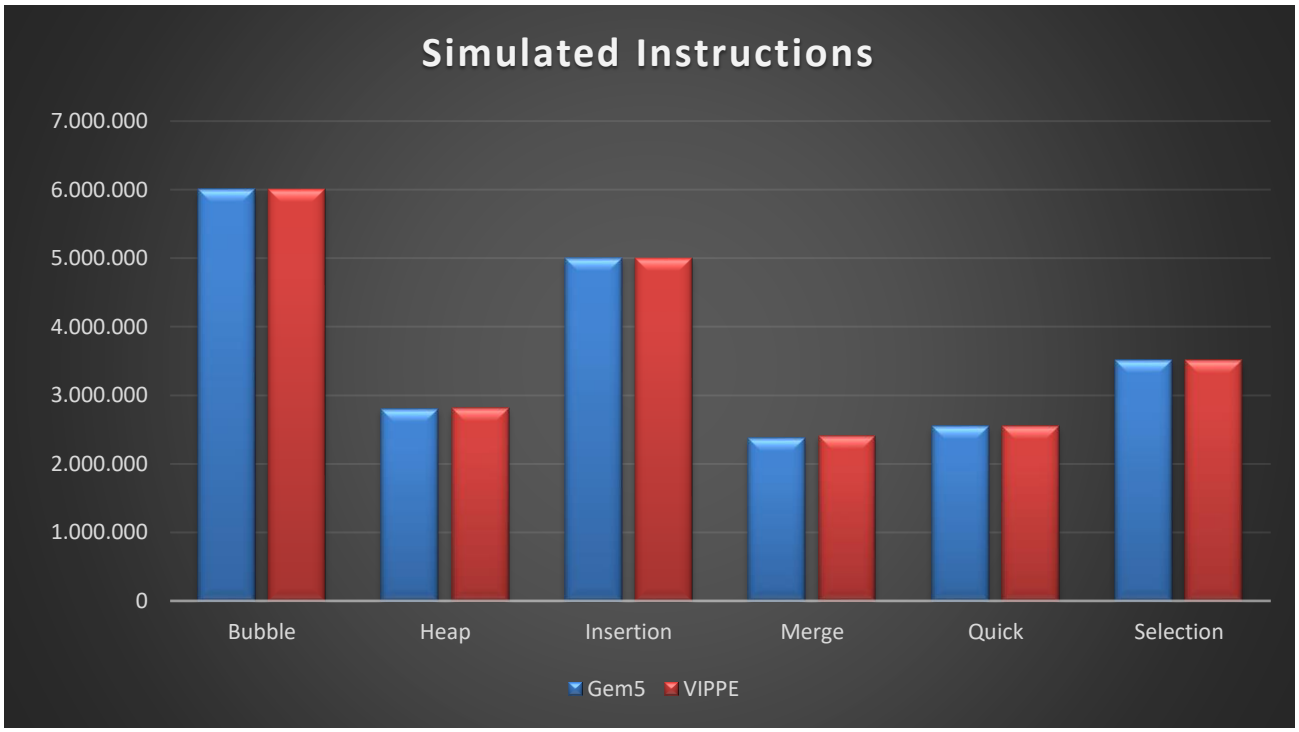
2.4 Sorting Benchmarks

Here we use two classes of most common sorting algorithms to compare benchmarks results. The O(n2) complexity Bubble, Insertion, Selection sorts, and the O(n log n) complexity Heap, Merge and Quick sorts.

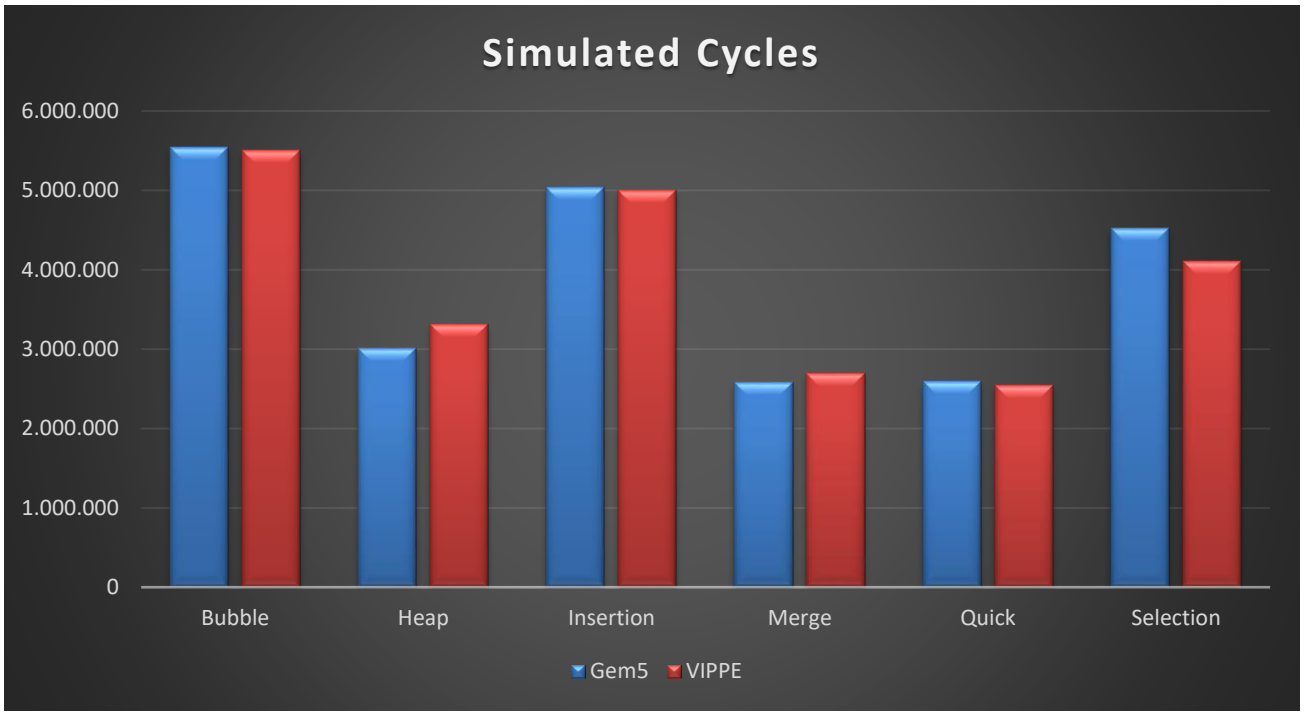


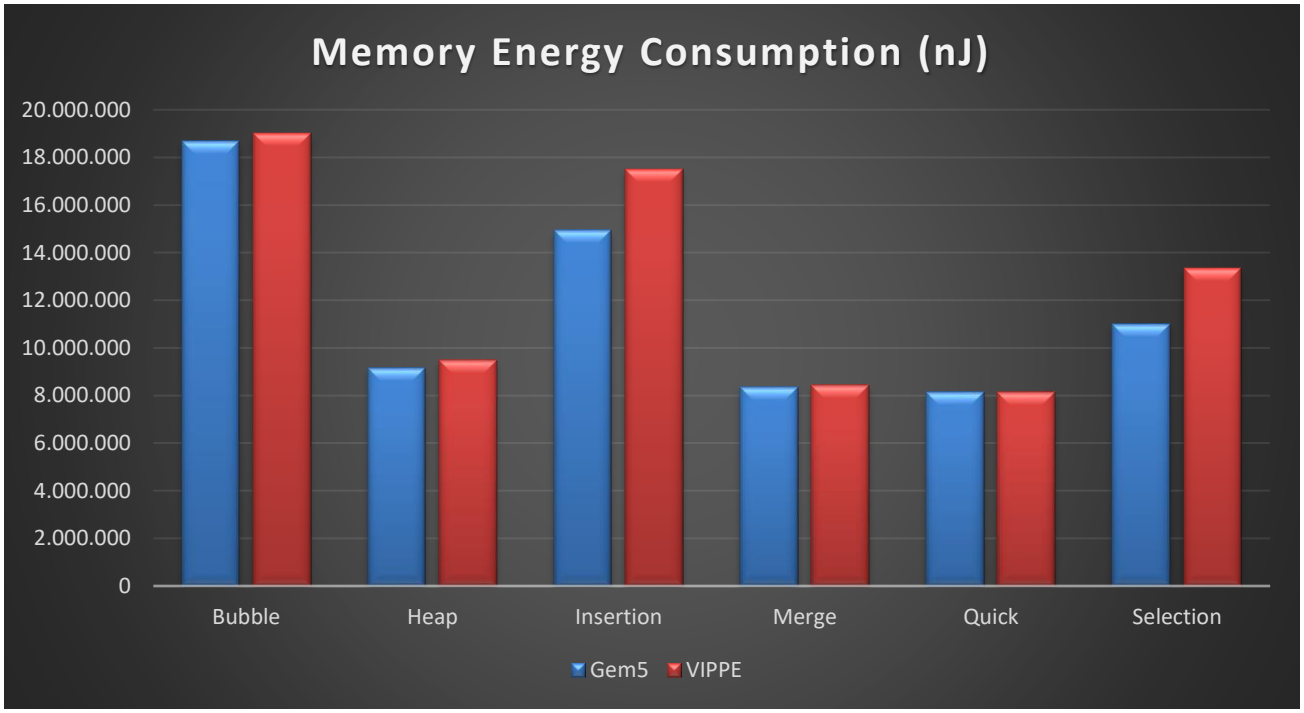
VIPPE vs GEM5



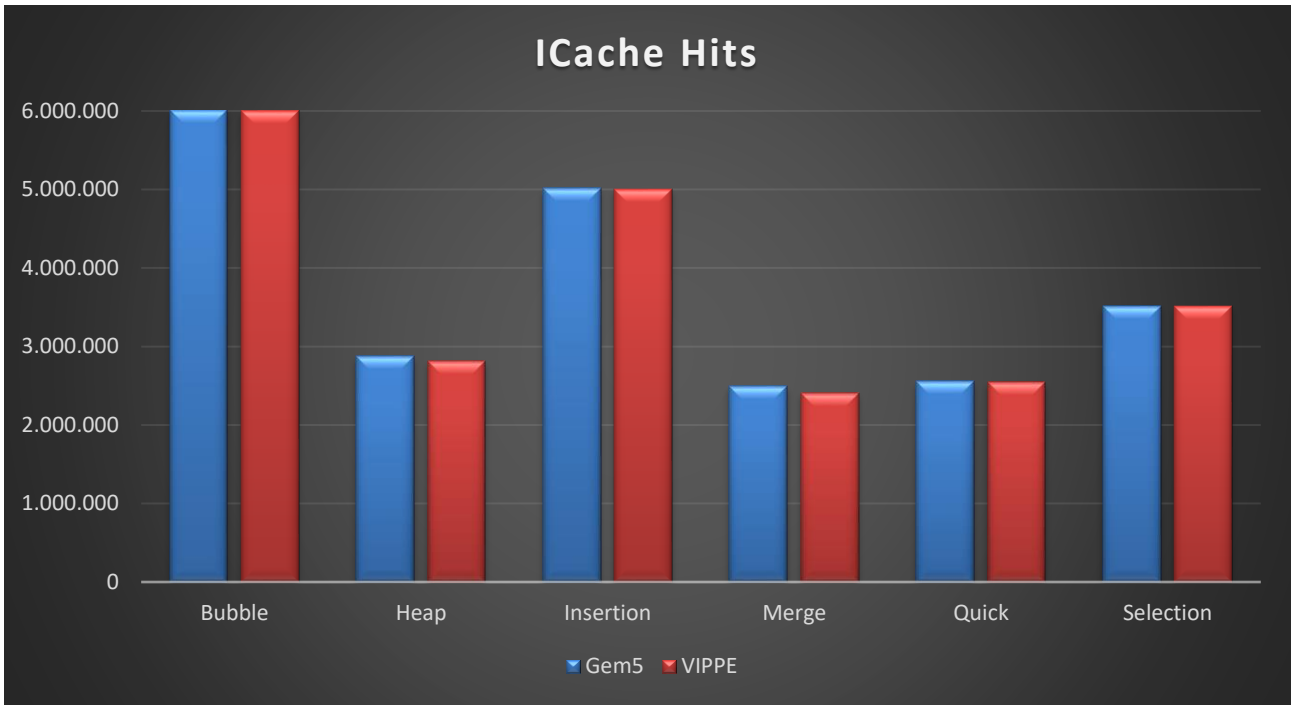


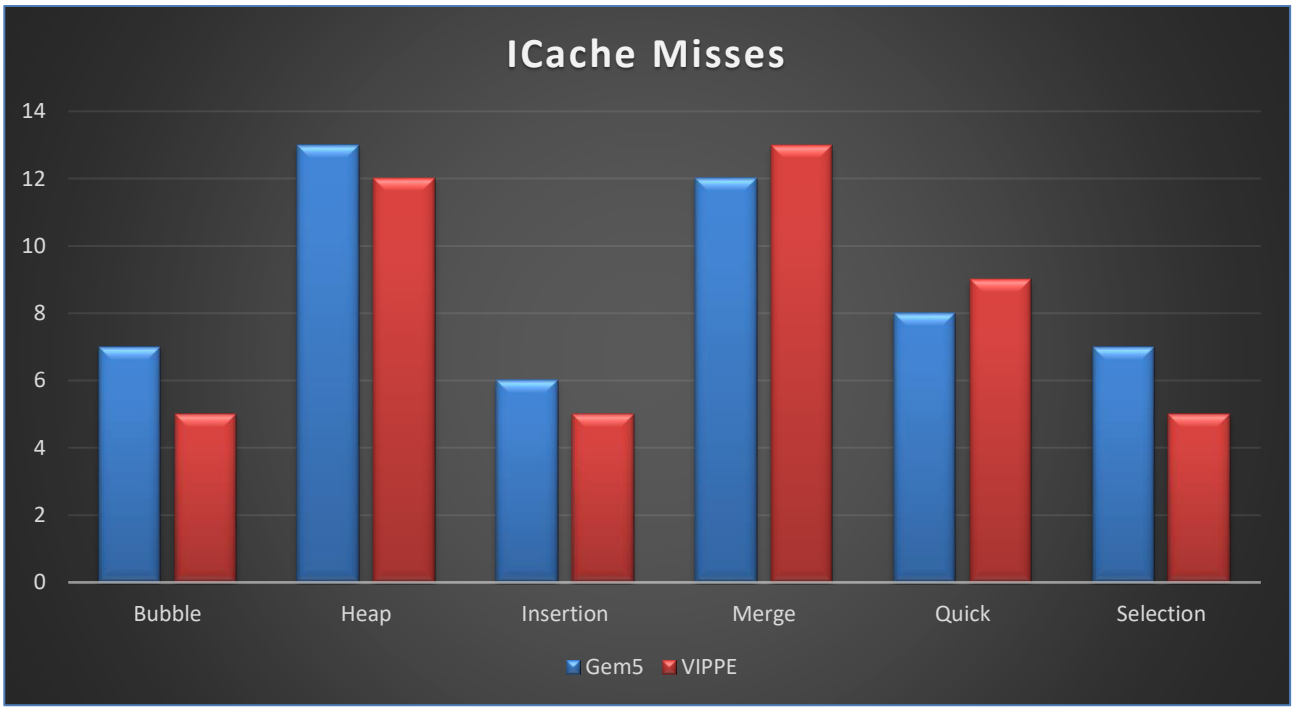
VIPPE vs GEM5



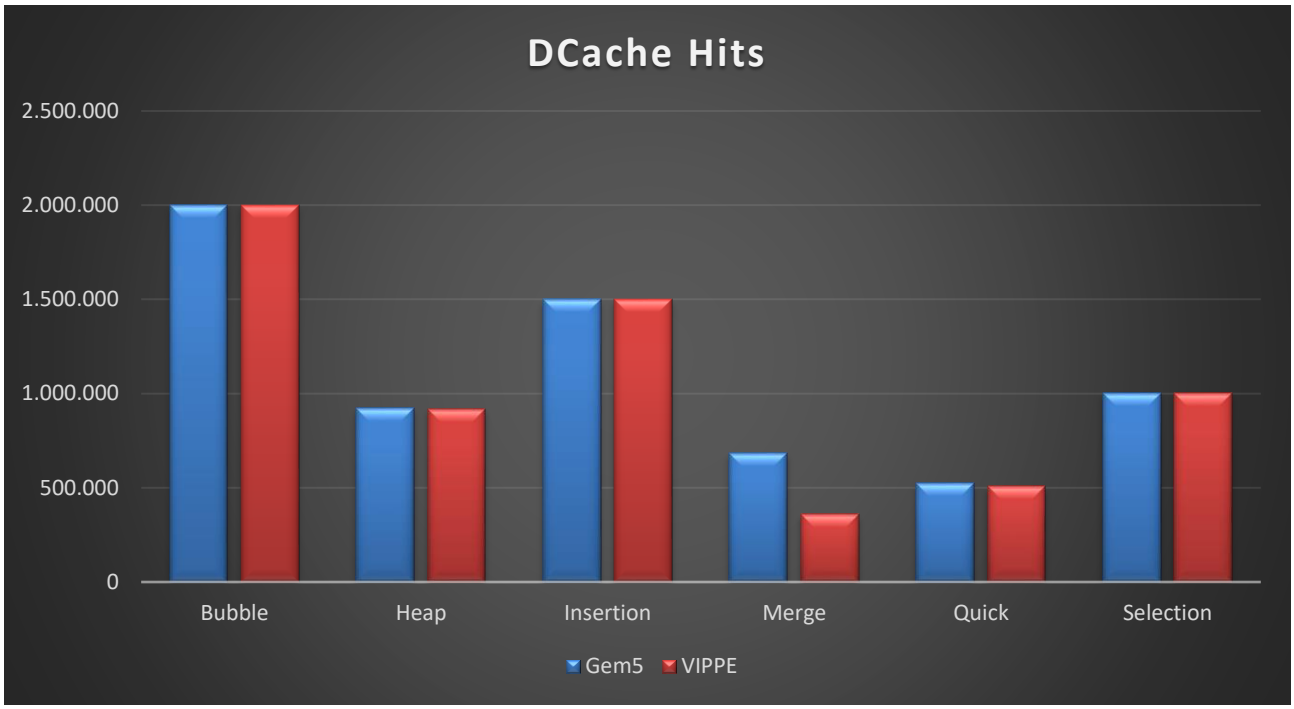


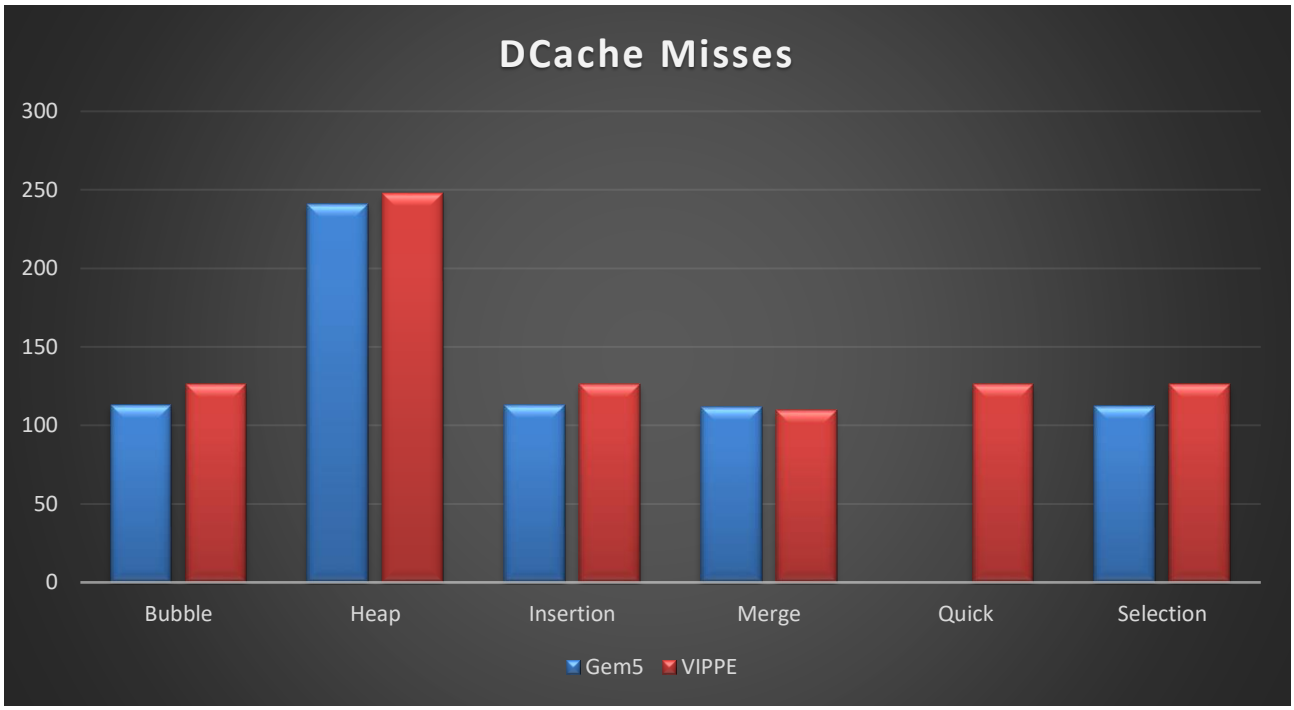
VIPPE vs GEM5





VIPPE vs GEM5





2.5 Complex Algorithms Benchmarks

For benchmarking we use SingleSource workloads from the LLVM test-suite and Computer Language Benchmark Game Project. The algorithms used are described as follows.

nSieveBits – Fast prime number generator with segmented Sieve of Eratosthenes. The Sieve of Eratosthenes is a standard benchmark used to determine the relative speed of different computers or, the efficiency of the code generated for the same computer by different compilers. Find primes numbers from 0 to 409600 in this test.

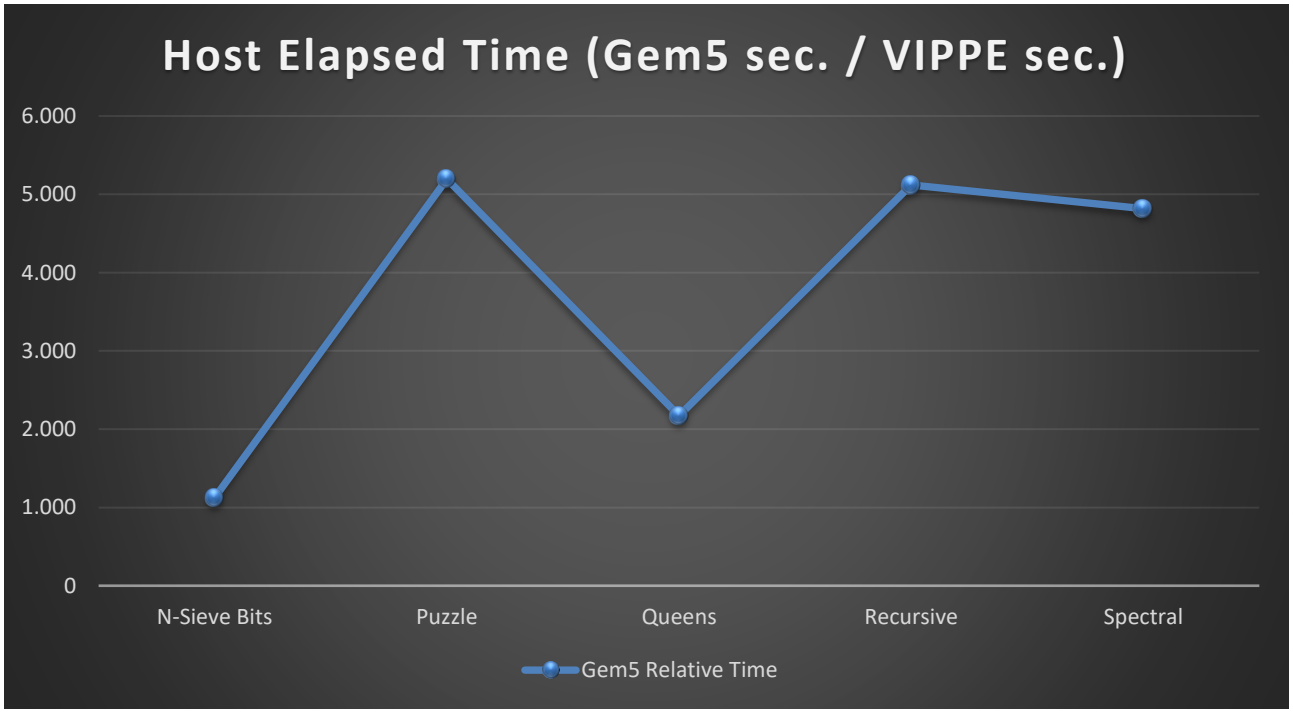
Spectral Normalization – The Von Mises iteration, is a very simple algorithm, but it may converge slowly. The most time-consuming operation of the algorithm is the multiplication of 150 ranged matrix A by a vector, so it is effective for a very large sparse matrix with appropriate implementation.

$$\|A\|_2 = \sqrt{\lambda_{\max}(A^* A)} = \sigma_{\max}(A)$$

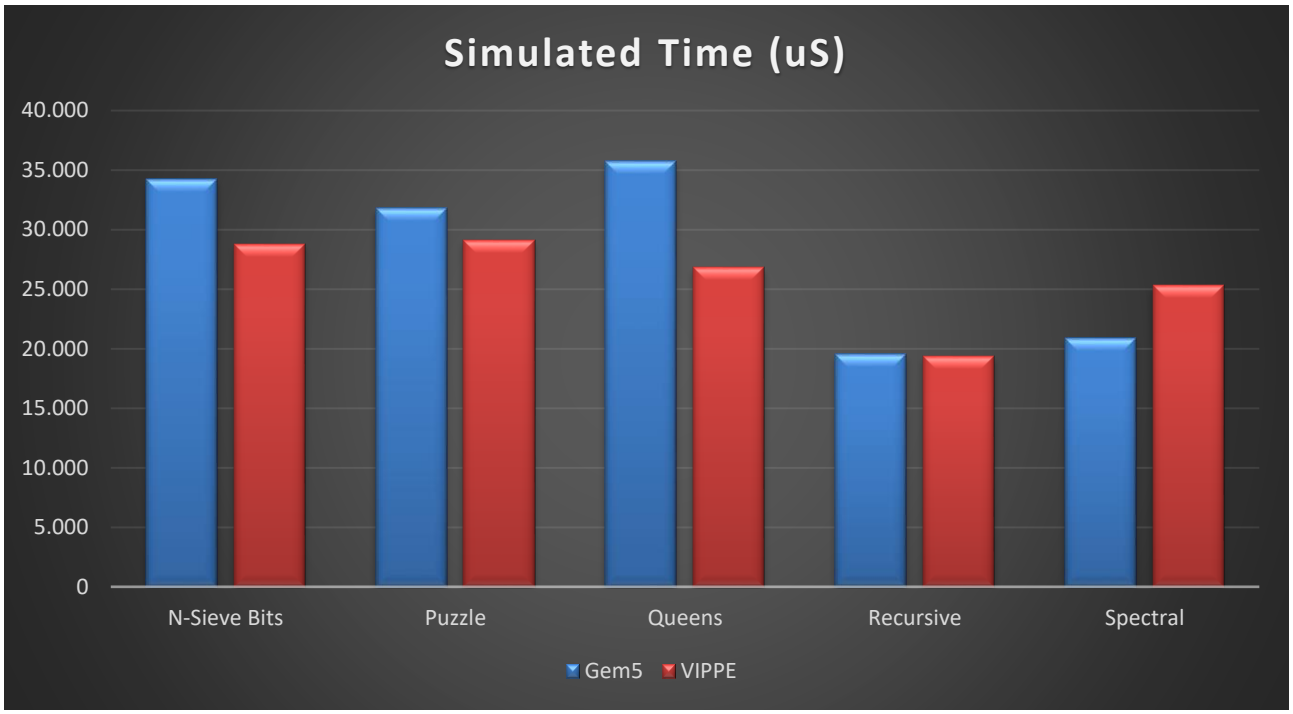
Queens – Find solutions to the Eight-Queens chess problem. The program finds all the possible ways that N (11 for this test) queens can be placed on an NxN chessboard so that the queens cannot capture one another, so no rank, file or diagonal is occupied by more than one queen.

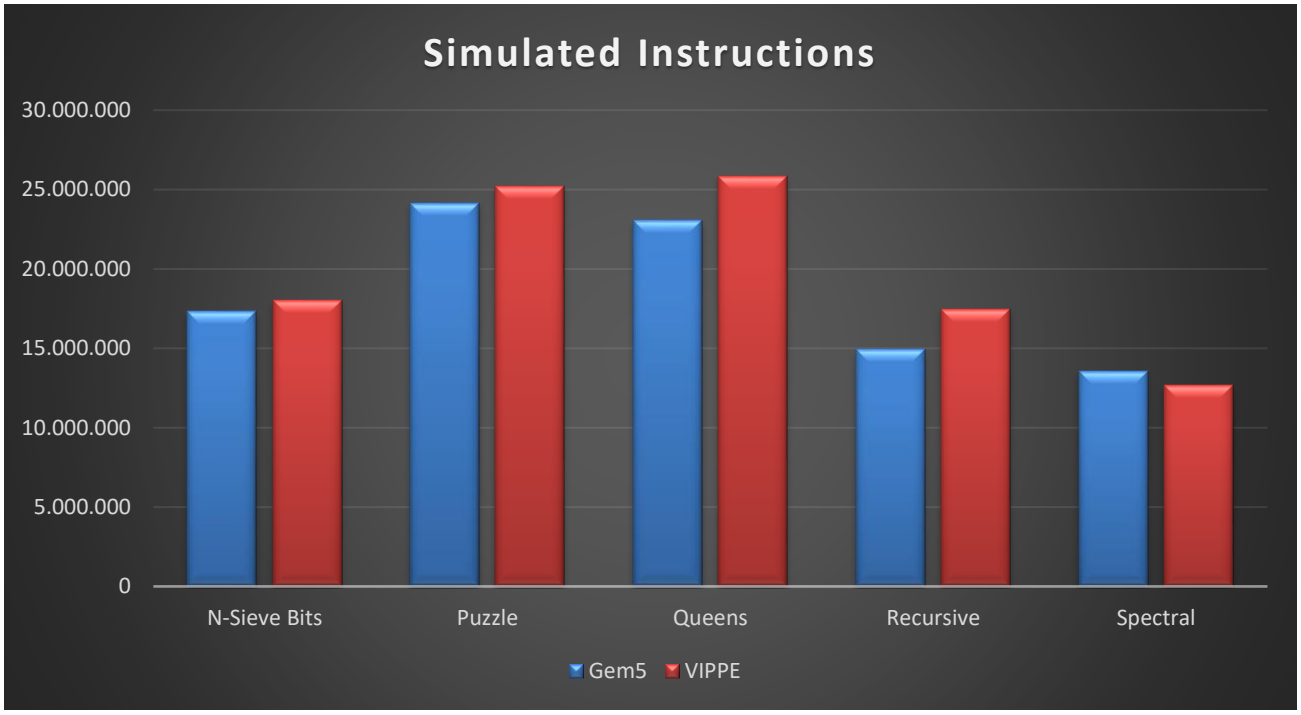
Puzzle – Find duplicate numbers stored in an array, previously filled with runtime generated random numbers. For this test we use an array with length 5000.

Recursive – Recursive algorithm to calculate 3 simple numeric functions: Ackermann, Fibonacci and Tak. For this benchmark, the Fibonacci and Tak implementations provide separate functions, one for integer calculation and one for double calculation. So, there is a function that uses integer calculation with integer parameters and other function uses double calculation with double parameters.

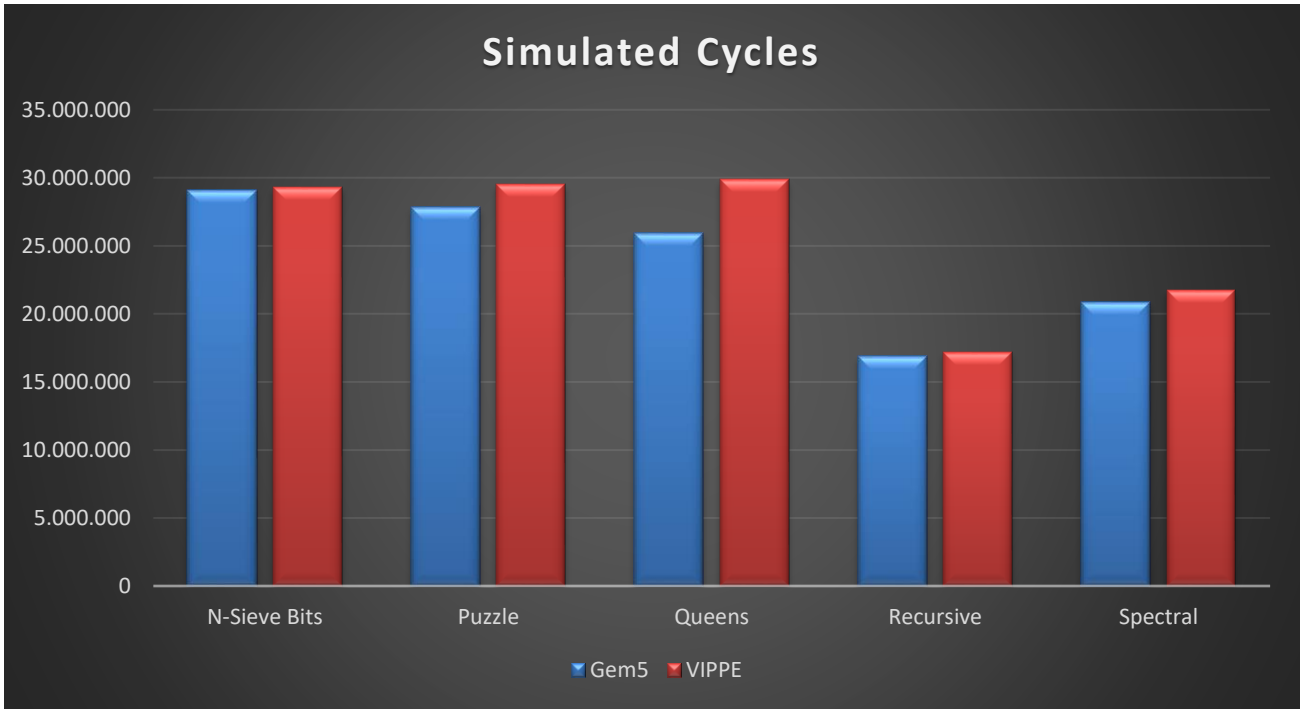


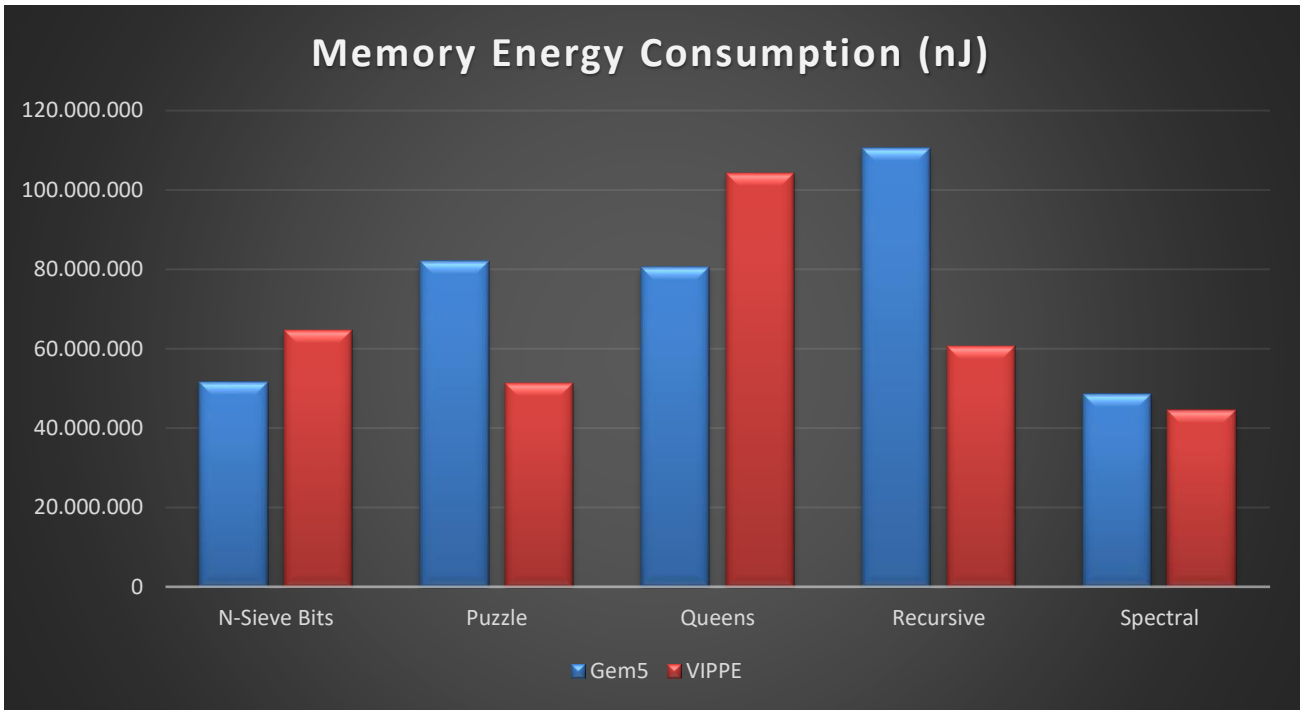
VIPPE vs GEM5



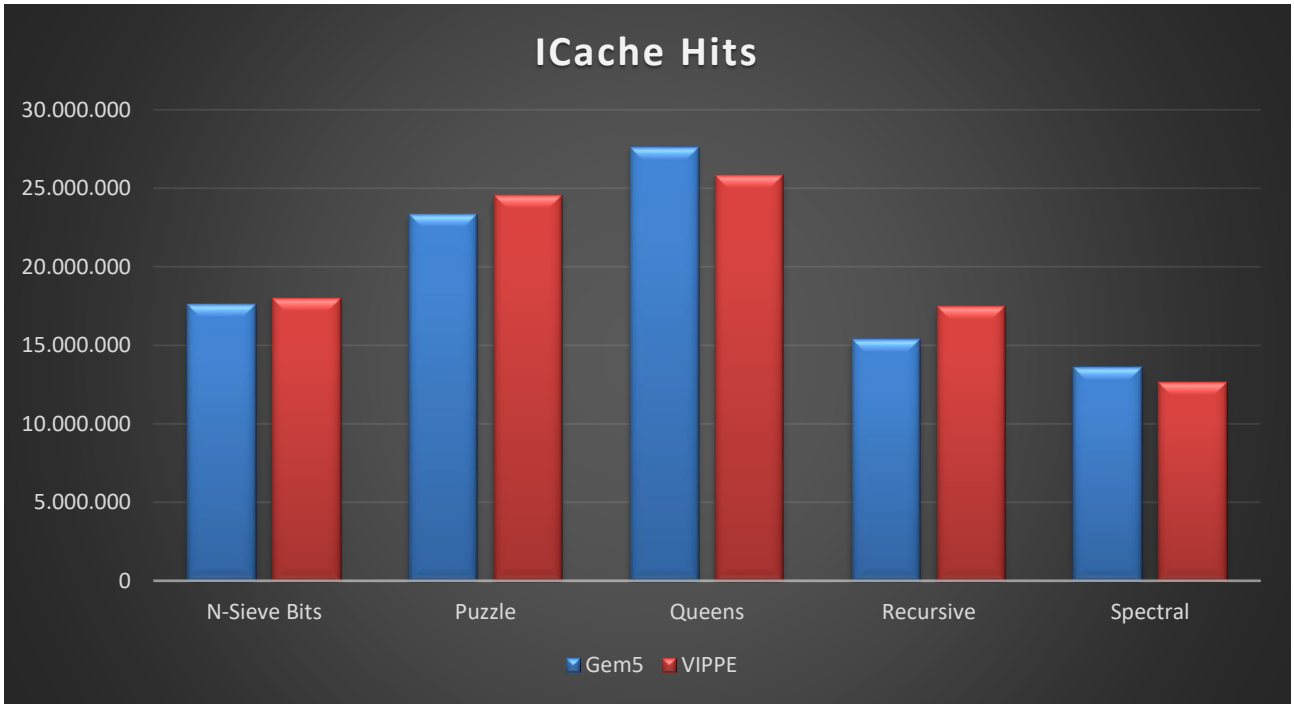


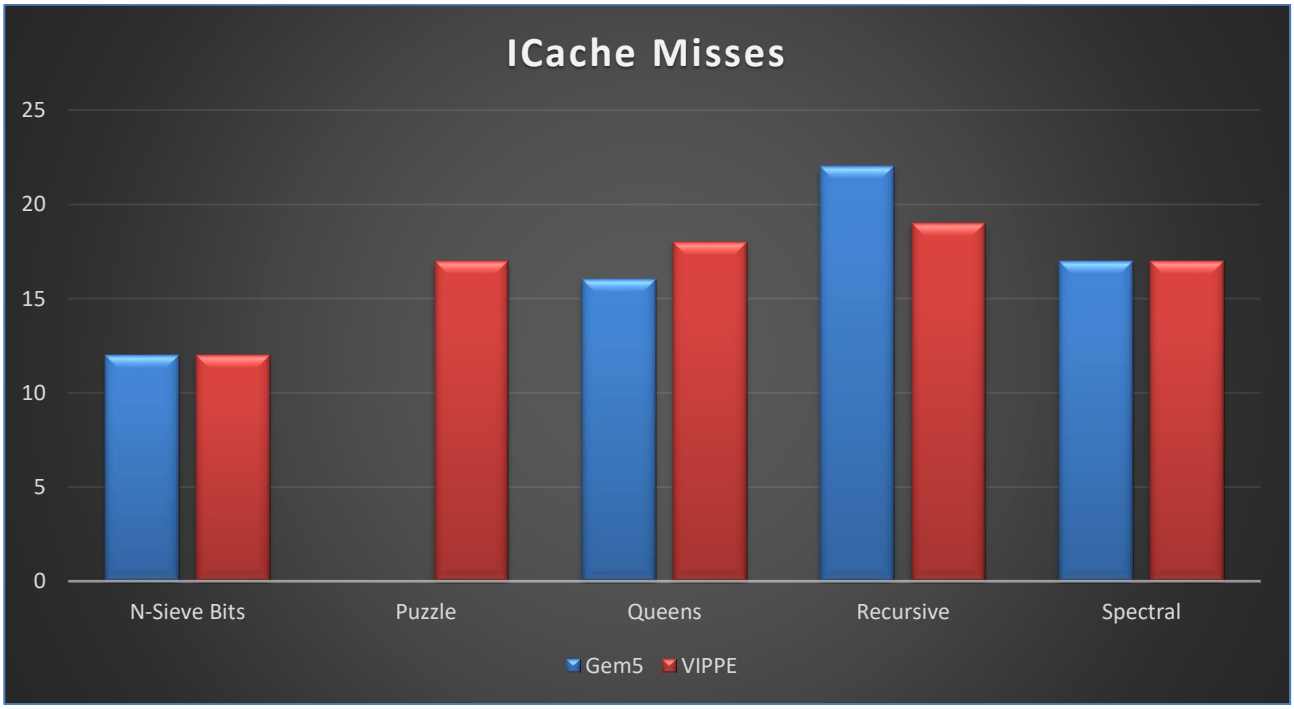
VIPPE vs GEM5



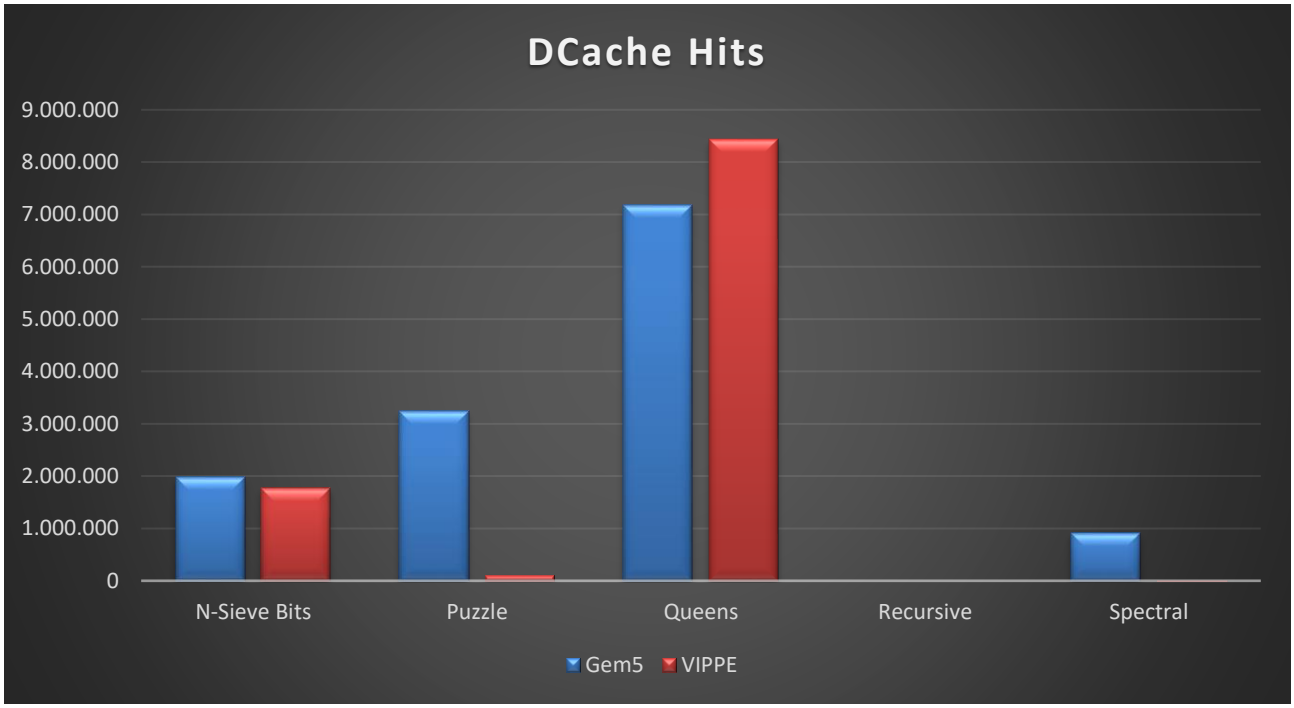


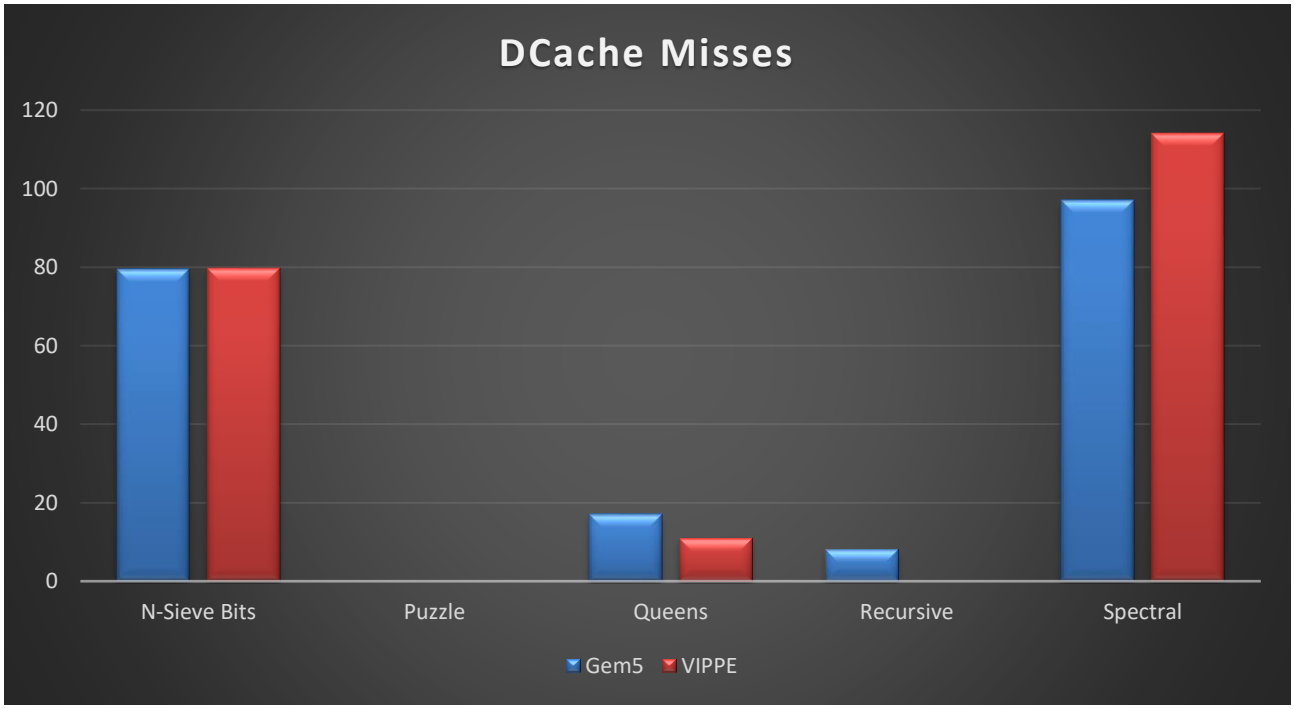
VIPPE vs GEM5





VIPPE vs GEM5





3 RISC-V Benchmarks [TO-DO]

3.1 RISC-V Simulated Model

Parameter	Simulated System
ISA	RISC-V
CPU	Generic
CPU frequency	1 GHz
Num. of cores	1
Main memory size	500 MB
L1 data cache size	32 kB
L1 instruction cache size	32 kB

Table 1. Simulated system parameters

3.2 About Gem5 Test for RISC-V

RISC-V was not supported by Clang until its version 9 released at 19 September 2019. So we can't use clang 3.5 to cross-compile source files as we did for ARM. In order to cross-compile for RISC-V platform, firstable we'll need to install the RISC-V open source GNU Toolchain.

The binary file we will simulate is previously cross-compiled with **riscv32-unknown-elf-gcc** and statically linked as Gem5 simulator pre-requisite.

For instance, in bubbleSort example, we get gem5 input binary typing in terminal:

- riscv32-unknown-elf-gcc bubbleSort.cpp -o bubbleSort.riscv -static -O2 -fPIC –Wall

A rapid sight of *BubbleSort.riscv*, typing in linux terminal:

- *file Bubble.riscv*

ELF 32-bit LSB executable, UCB RISC-V, version 1 (SYSV), statically linked, not stripped

From *<GEM5 INSTALL DIR>/examples*, we can run gem5 simulation with this:

- *../build/RISCV/gem5.opt ../configs/example/se.py --cpu-type=DerivO3_CPU --num-cpus=1 --cpu-clock=1GHz --mem-type=SimpleMemory --mem-size=512MB --caches --cacheline_size=32 --l1d_size=32kB --l1i_size=32kB --l1i_assoc=1 --l1d_assoc=1 -c bubbleSort.riscv*

VIPPE vs GEM5

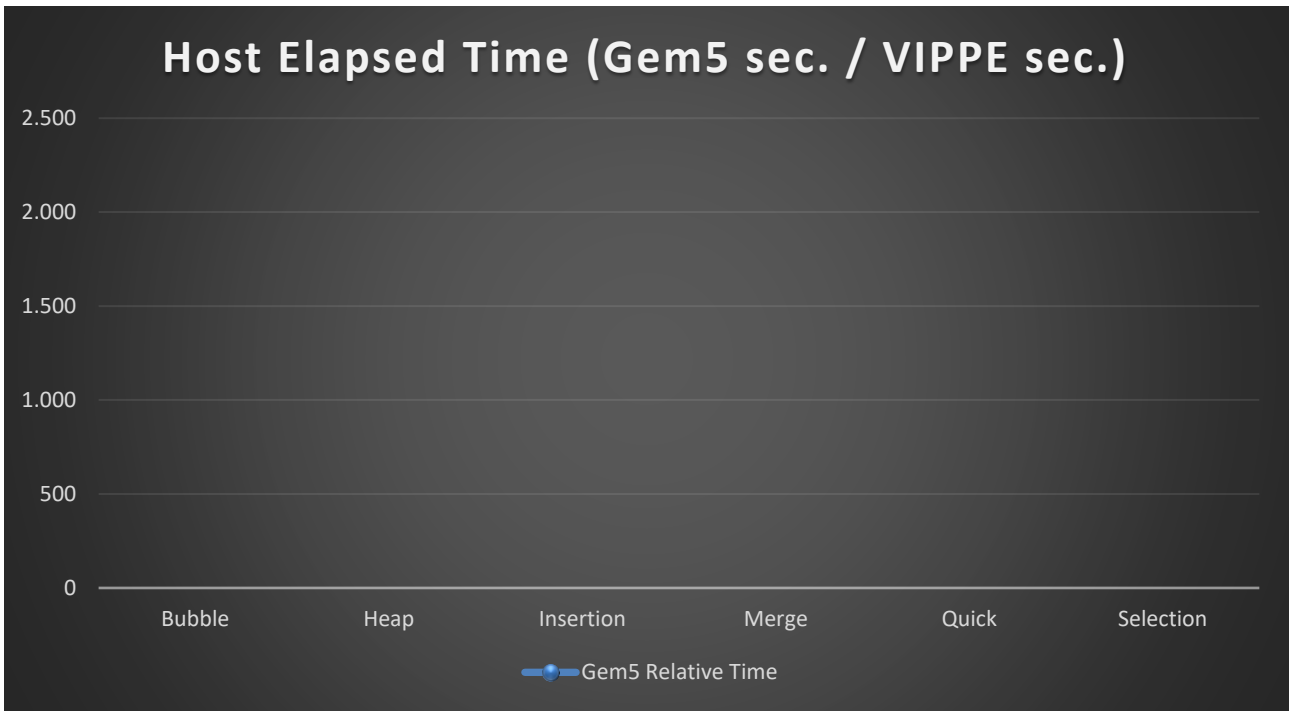
3.3 About VIPPE Test for RISC-V

To see a detailed explanation of VIPPE usage, please read the VIPPE manual. For this benchmarks, we'll just use a simple usage way:

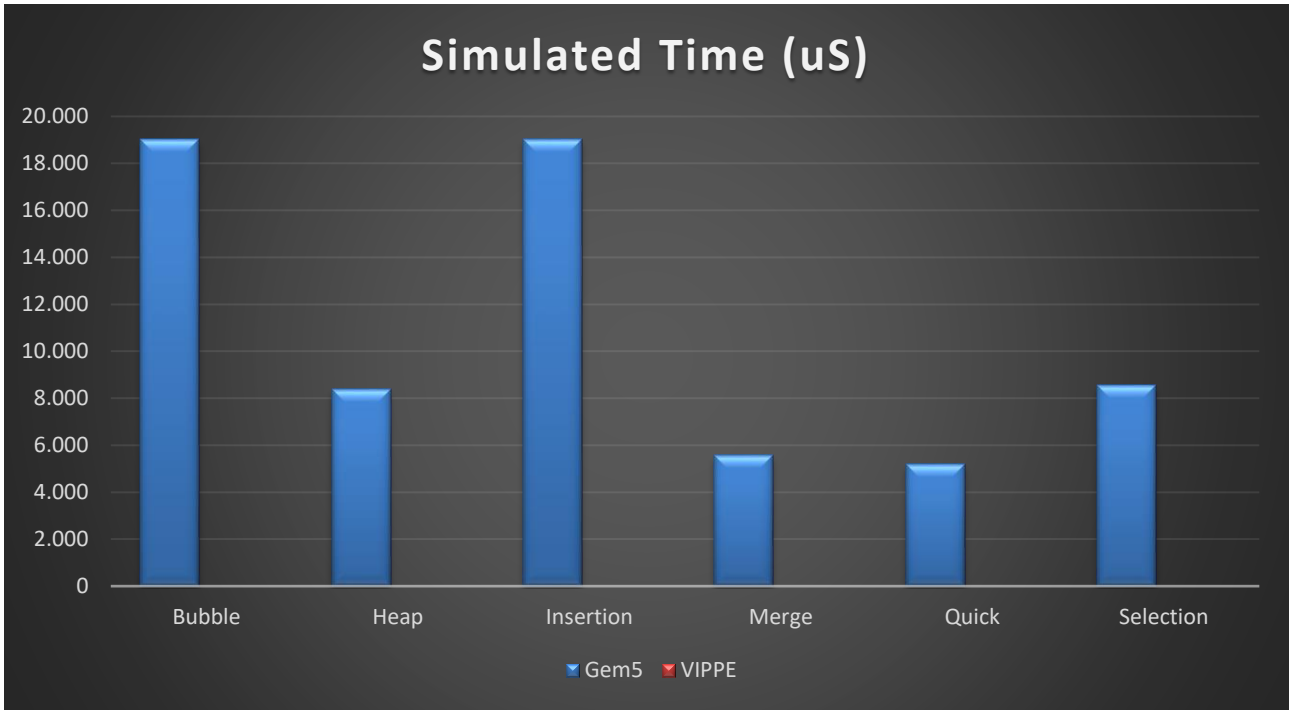
- vippe-llvm++ [-O2] --vippe-verbose --vippe-asm --vippe-cpu=riscv example.cpp -o example.o
- clang++ [LINKER_FLAGS] example.o -o example.x [INCLUDES] [LIBS]
- ./example.x

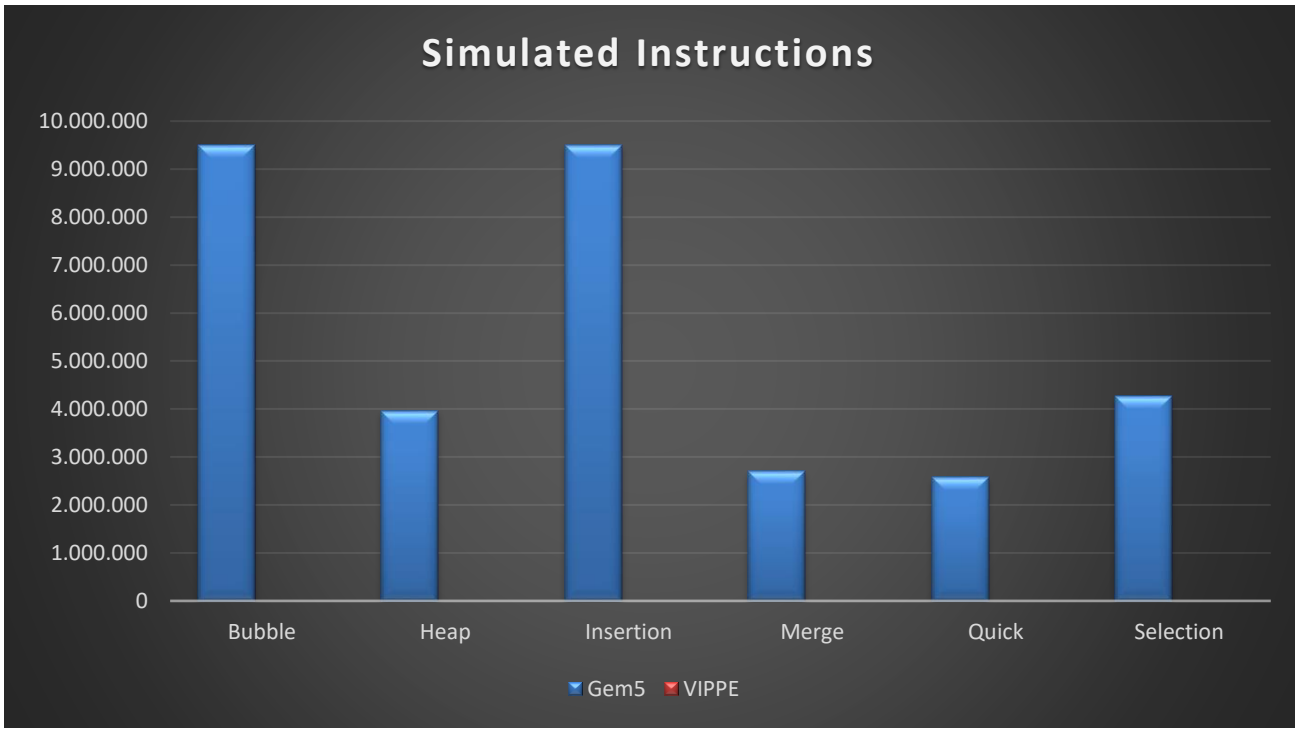
3.4 Sorting Benchmarks

Here we use two classes of most common sorting algorithms to compare benchmarks results. The O(n2) complexity Bubble, Insertion, Selection sorts, and the O(n log n) complexity Heap, Merge and Quick sorts.

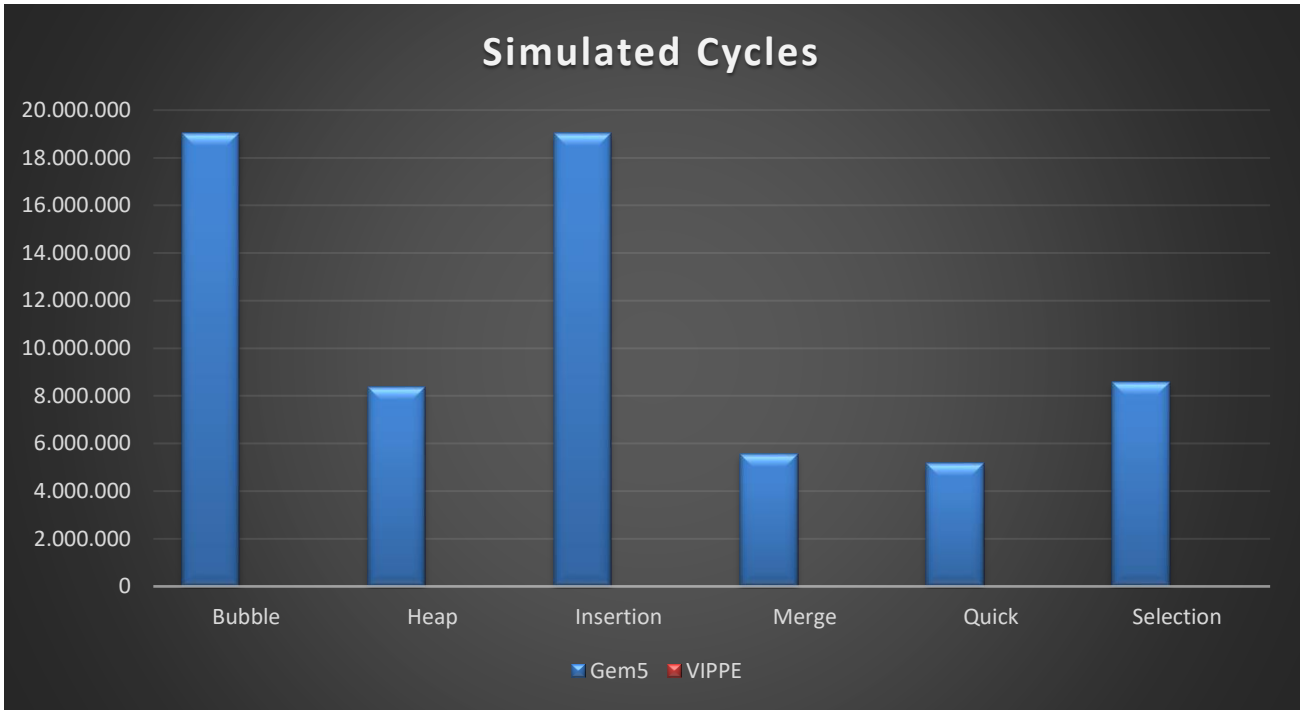


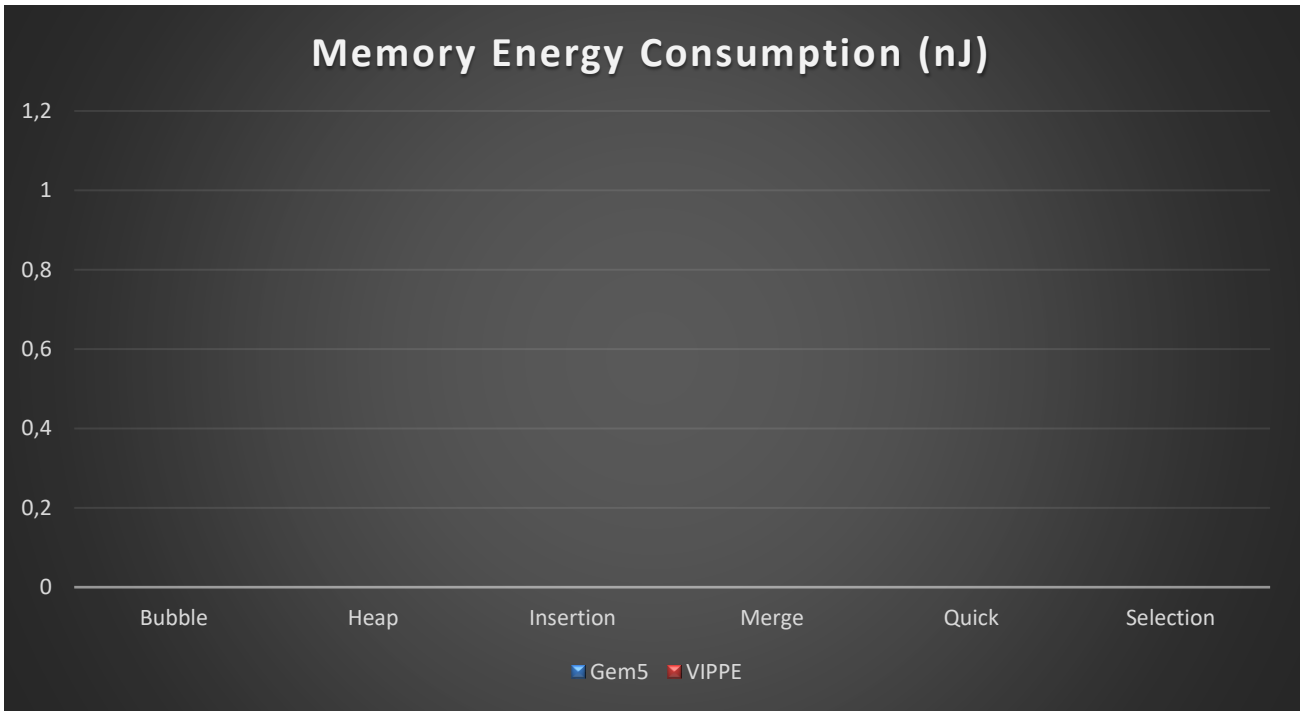
VIPPE vs GEM5



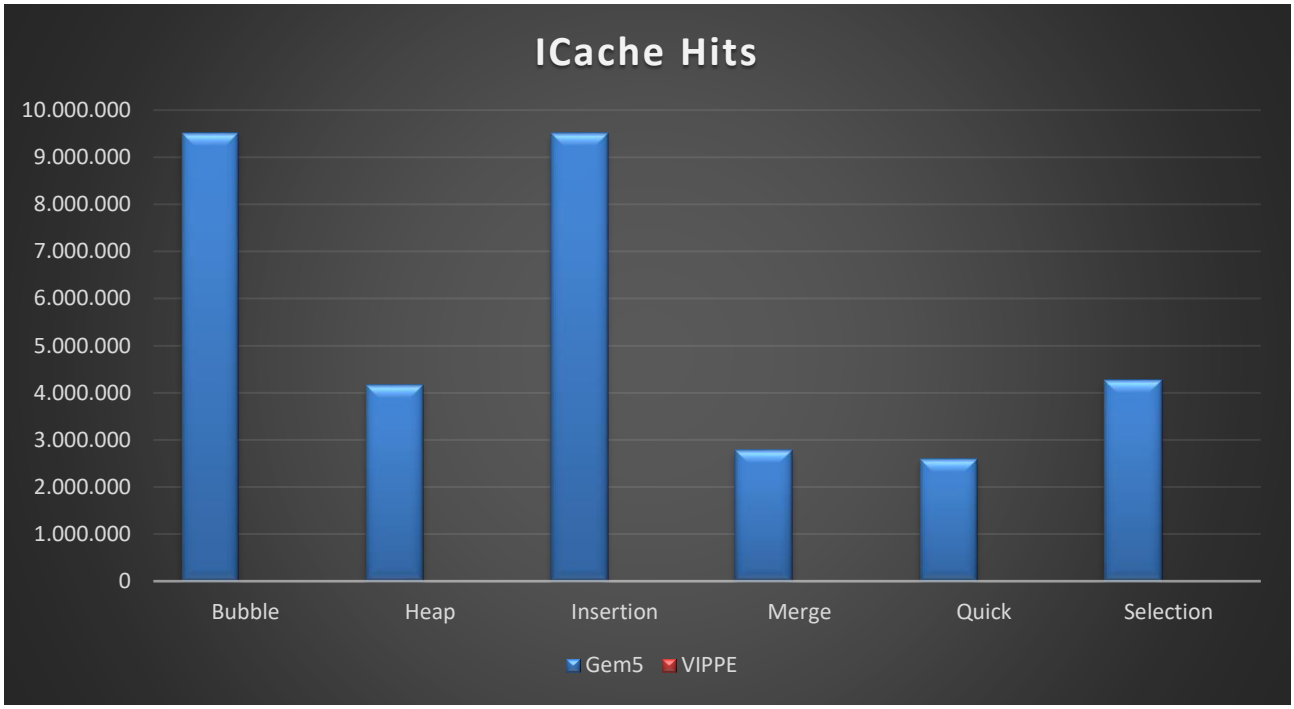


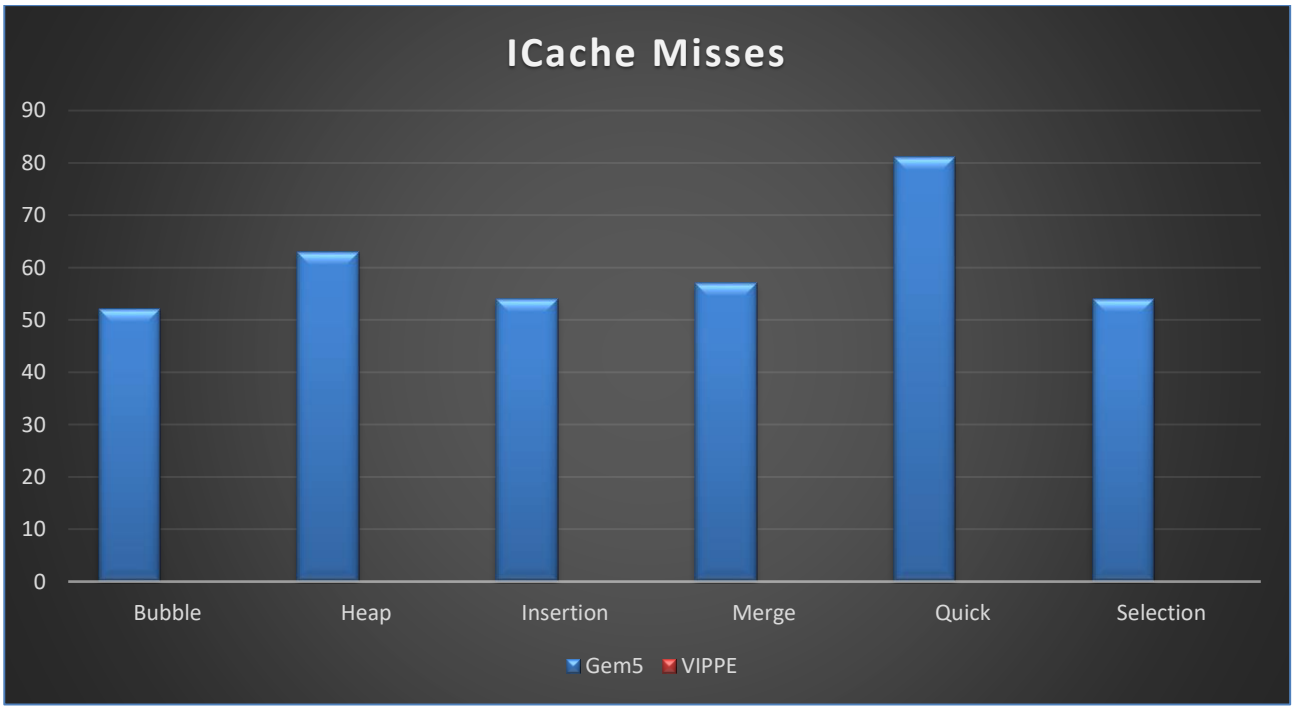
VIPPE vs GEM5



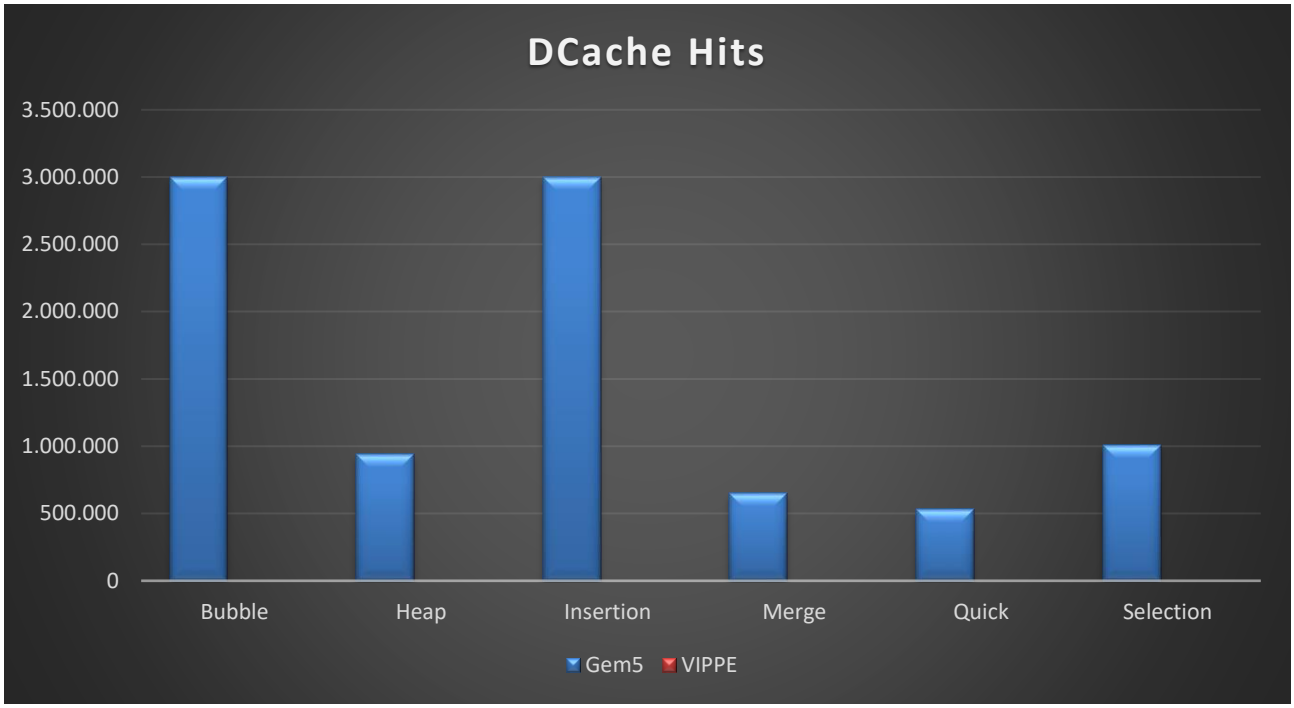


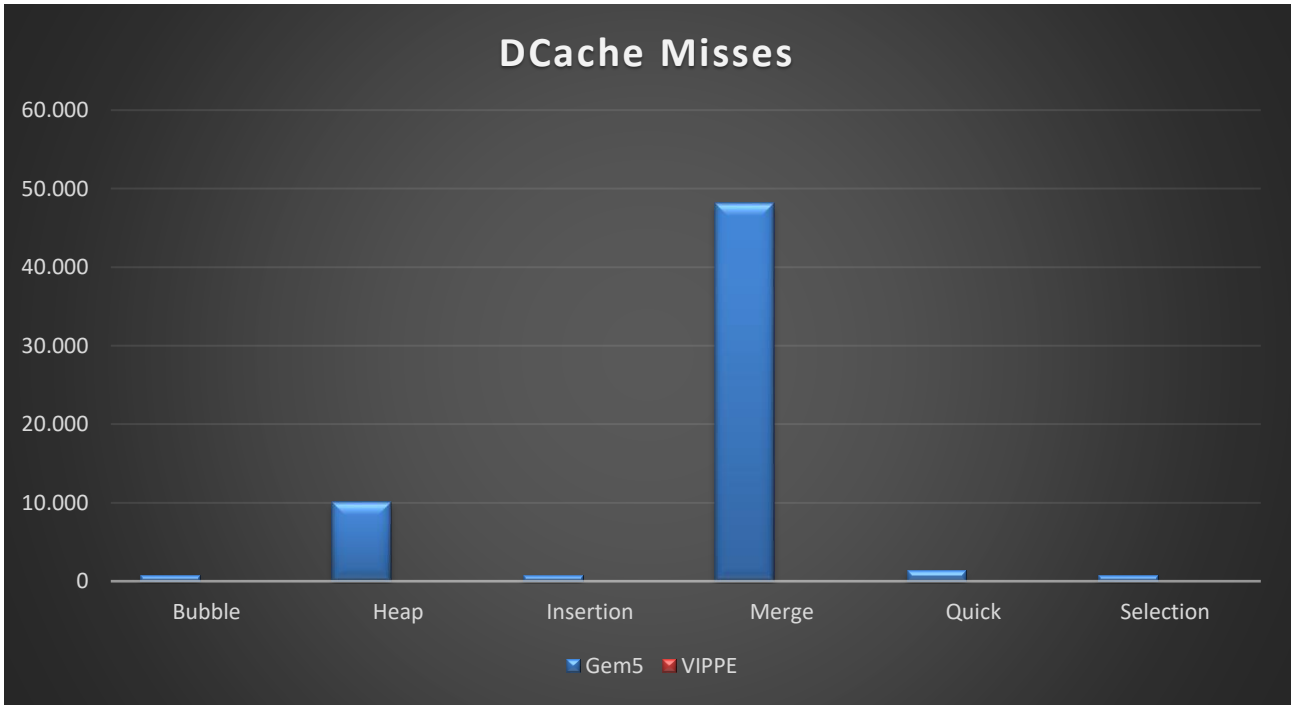
VIPPE vs GEM5





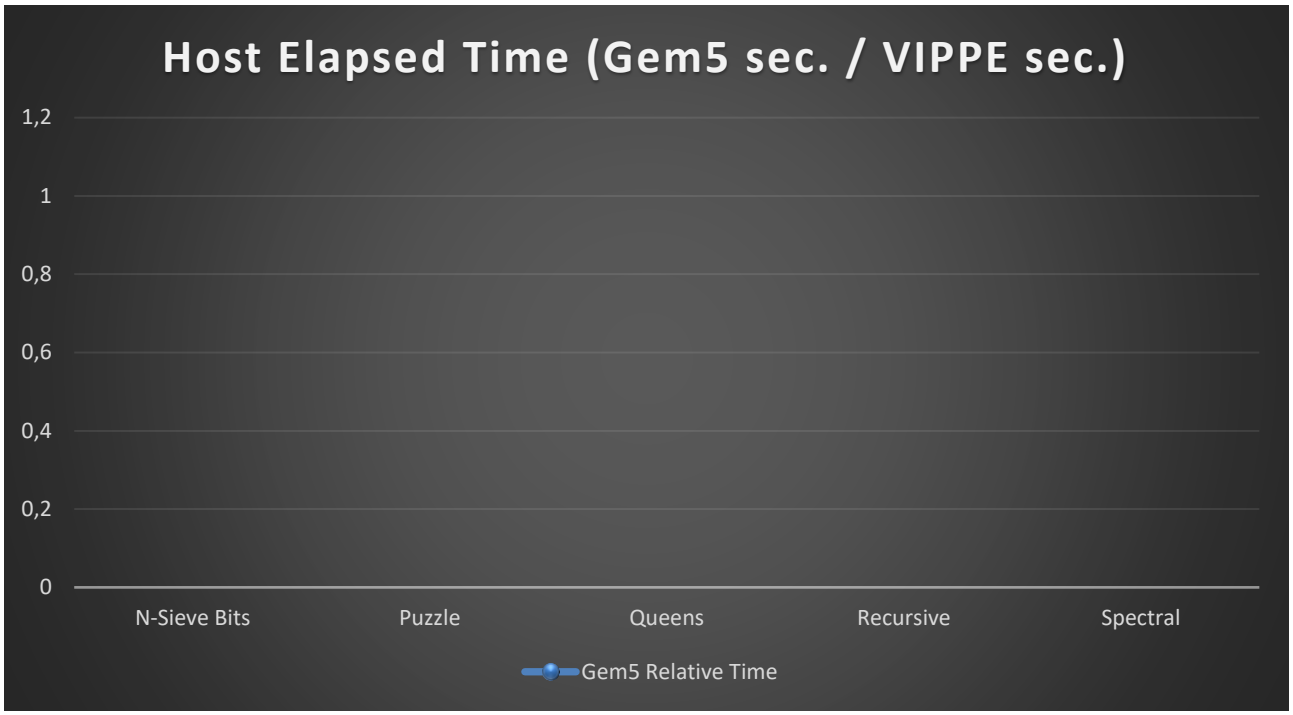
VIPPE vs GEM5



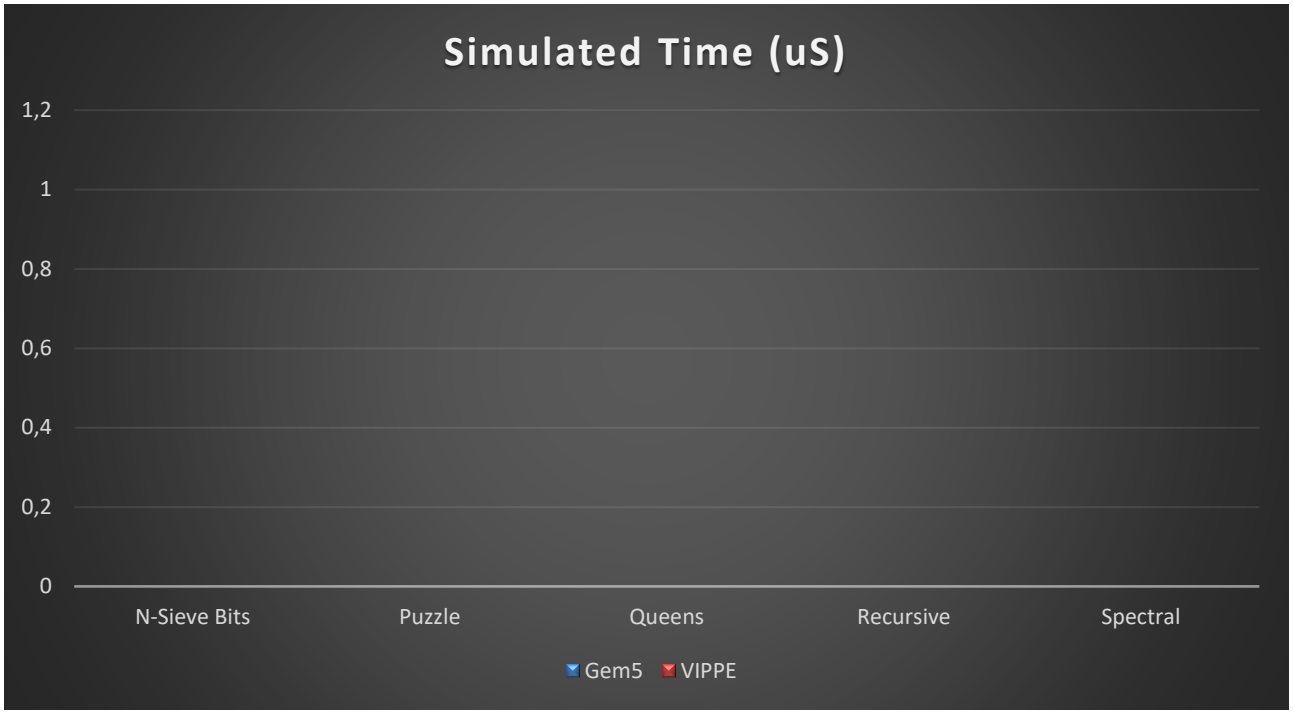


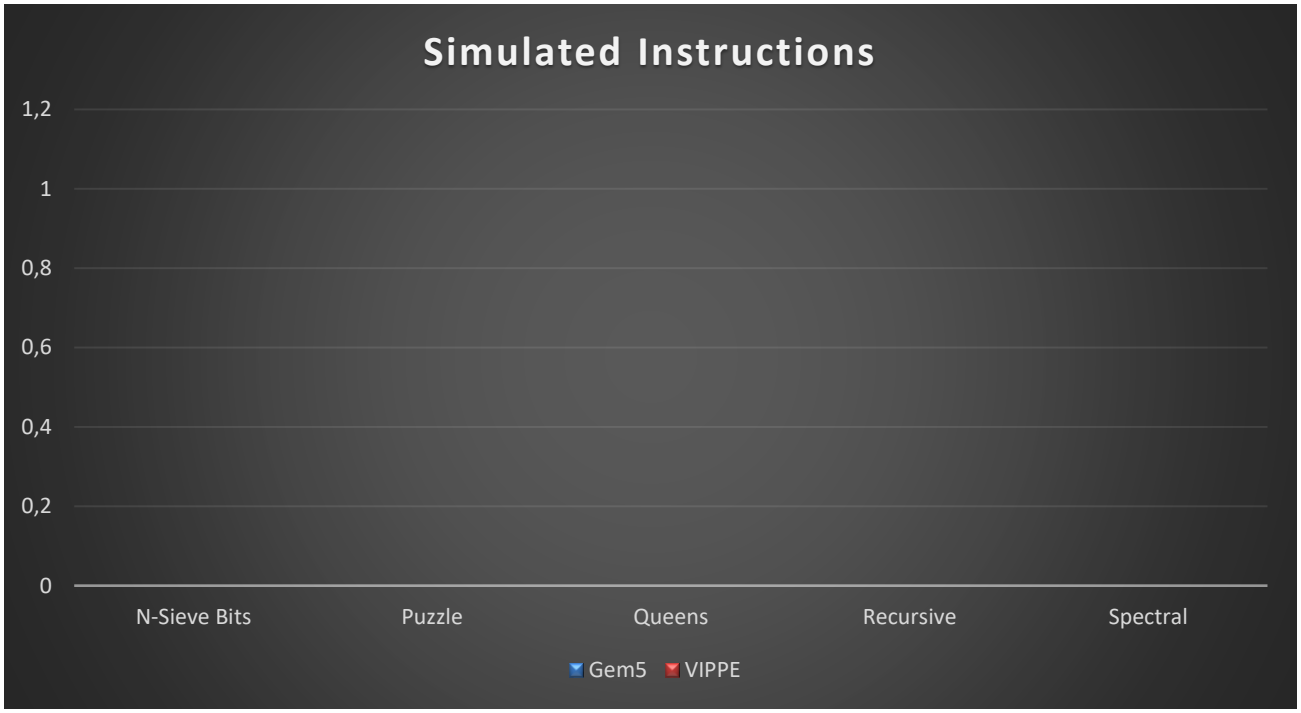
3.5 Complex Algorithms Benchmarks [TO-DO]

For benchmarking we use SingleSource workloads from the LLVM test-suite and Computer Language Benchmark Game Project.

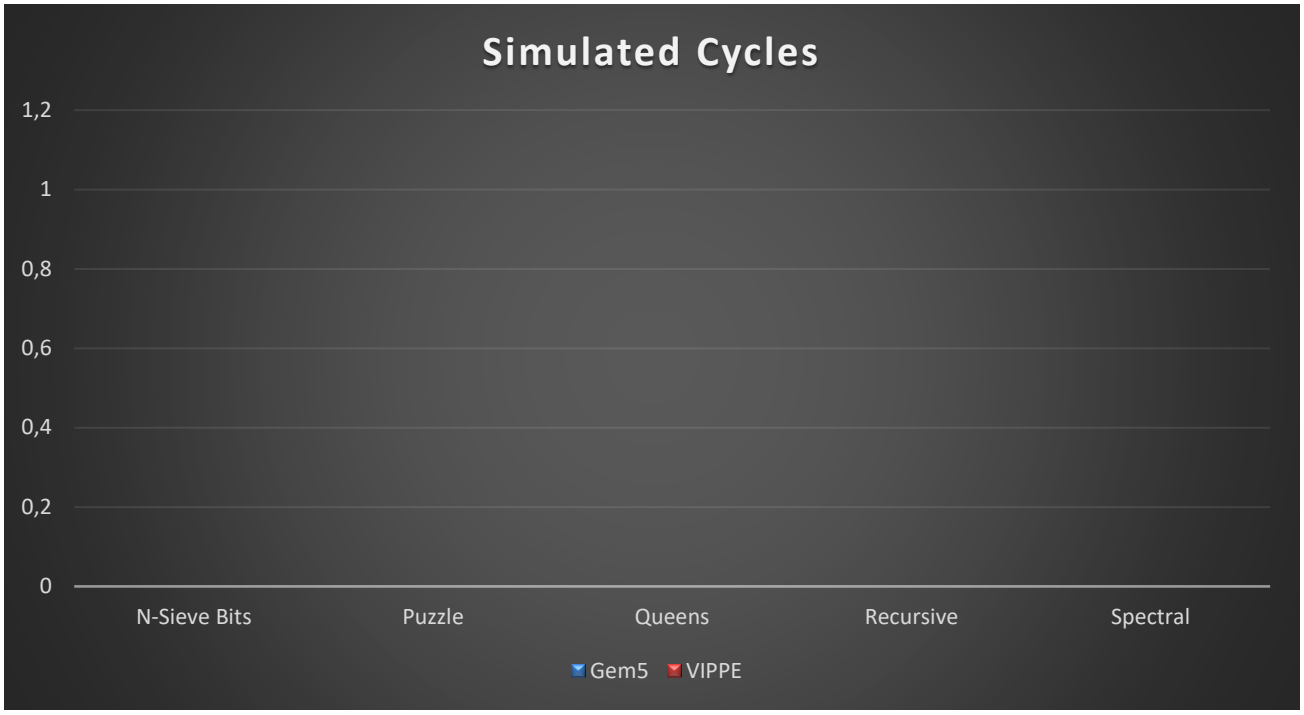


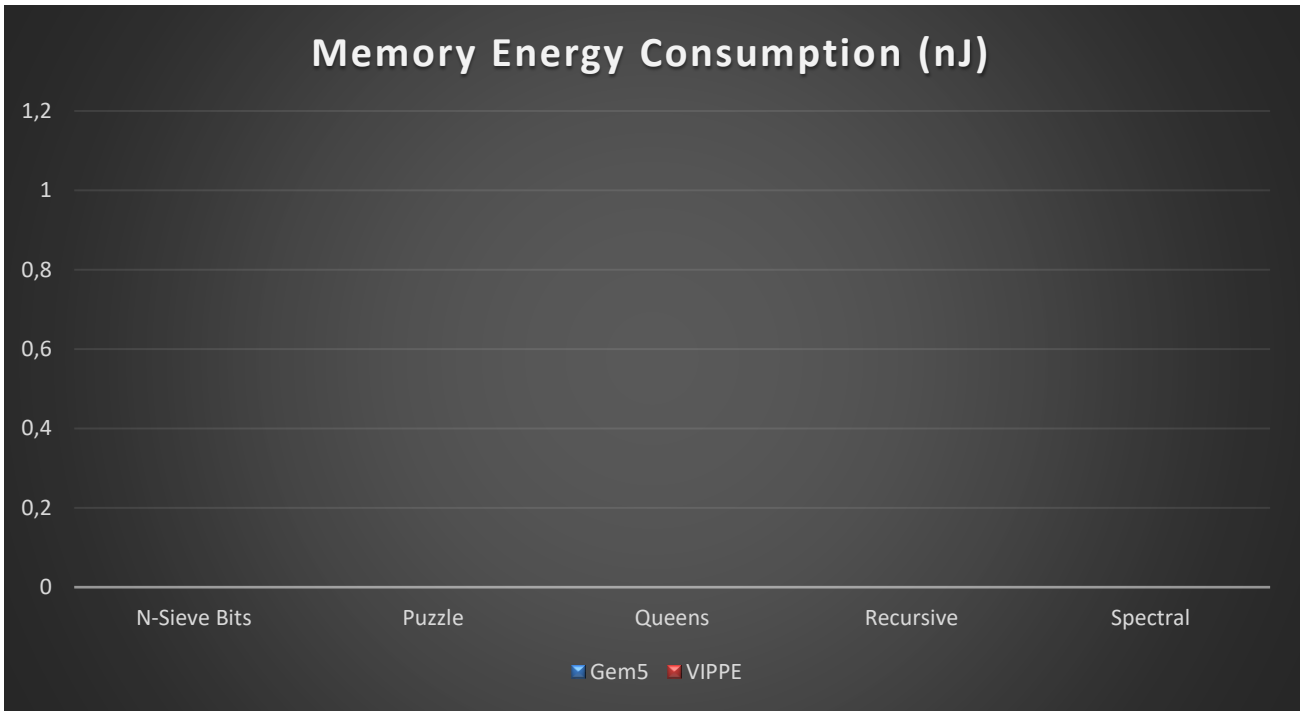
VIPPE vs GEM5



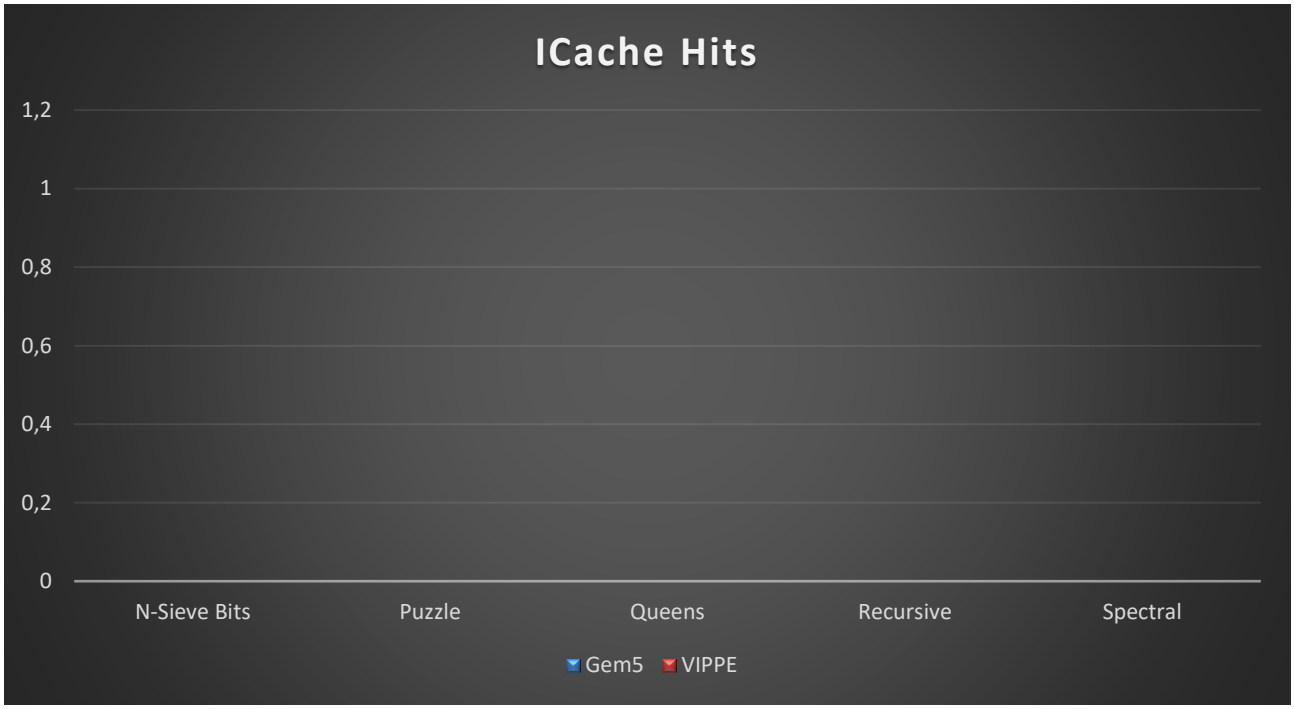


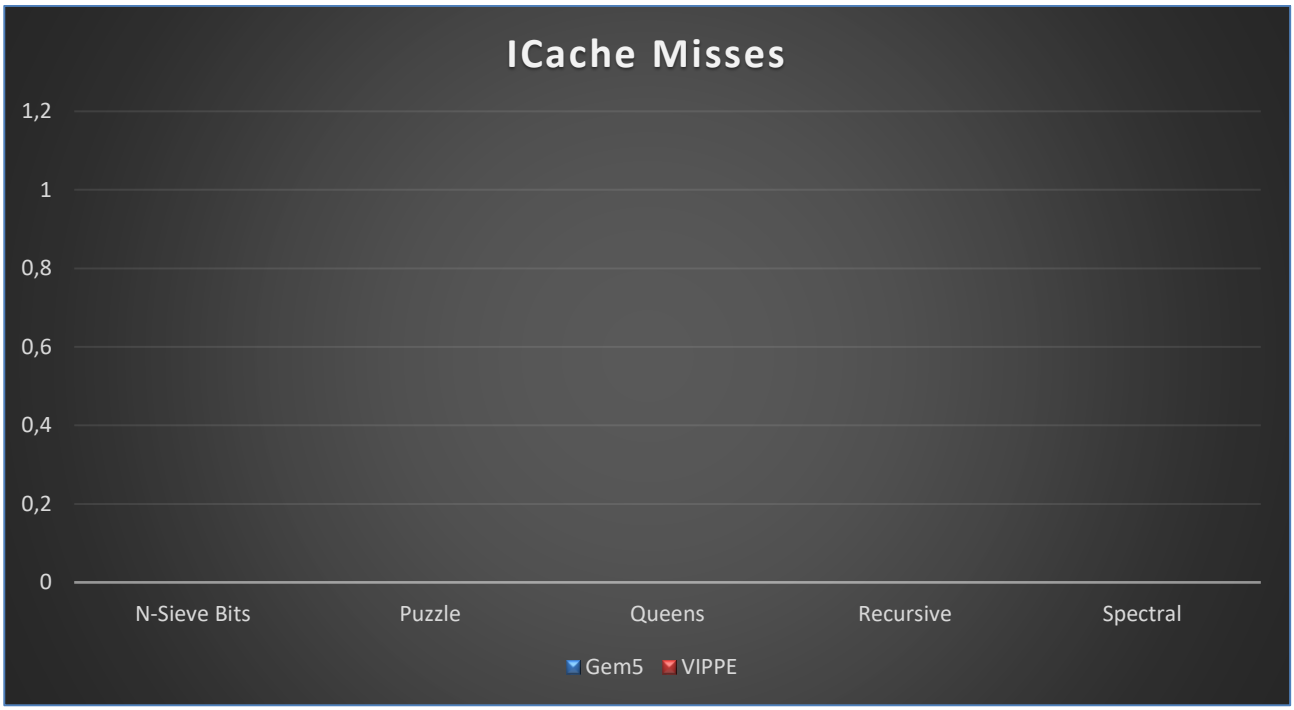
VIPPE vs GEM5



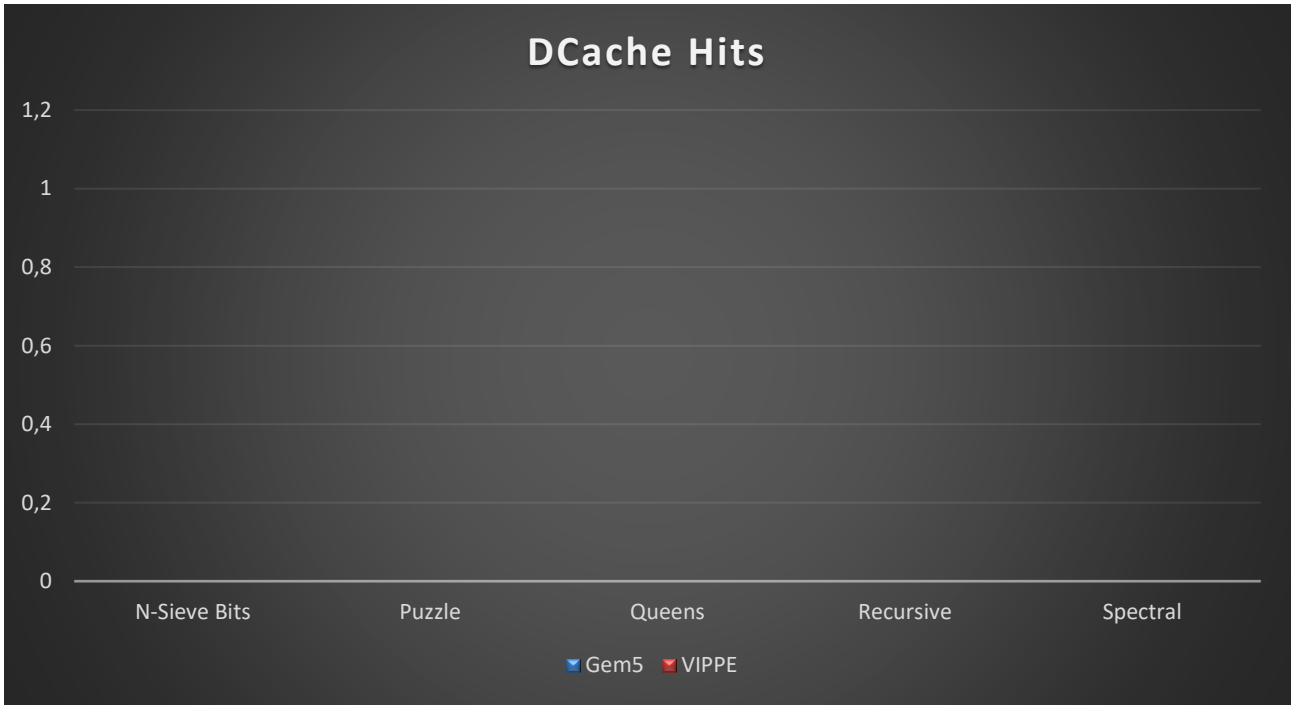


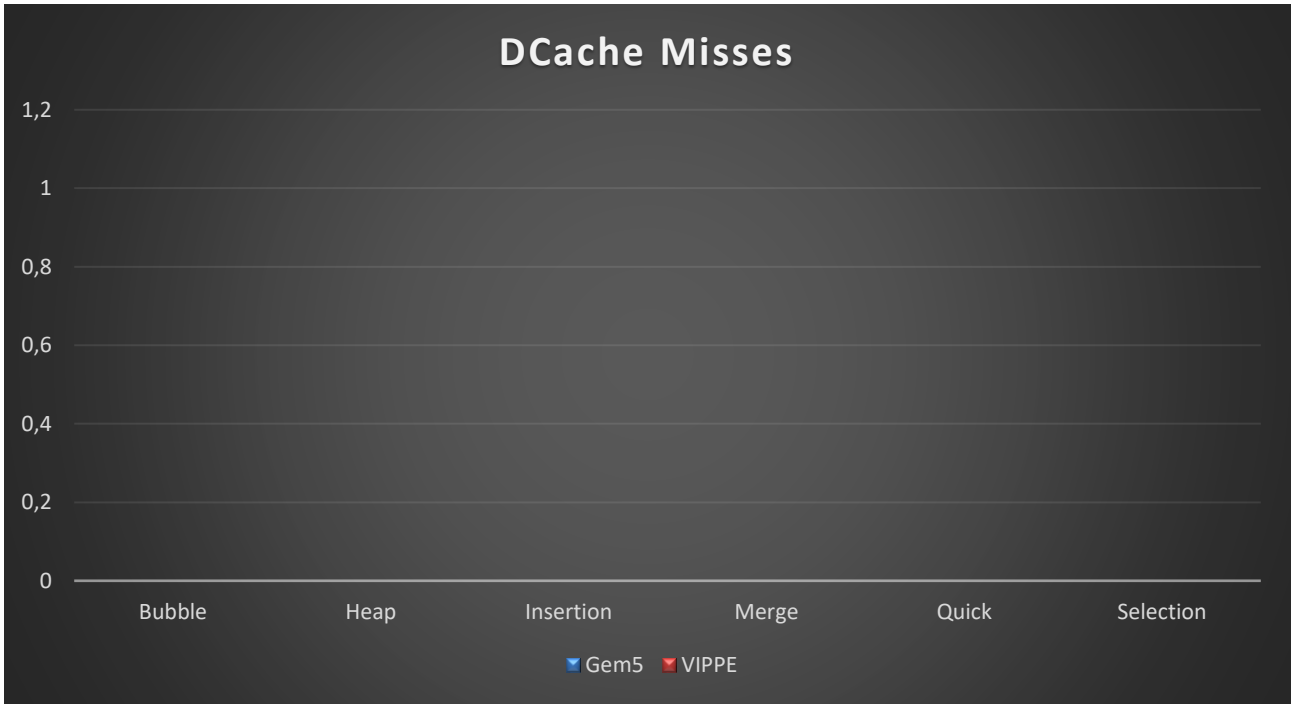
VIPPE vs GEM5





VIPPE vs GEM5





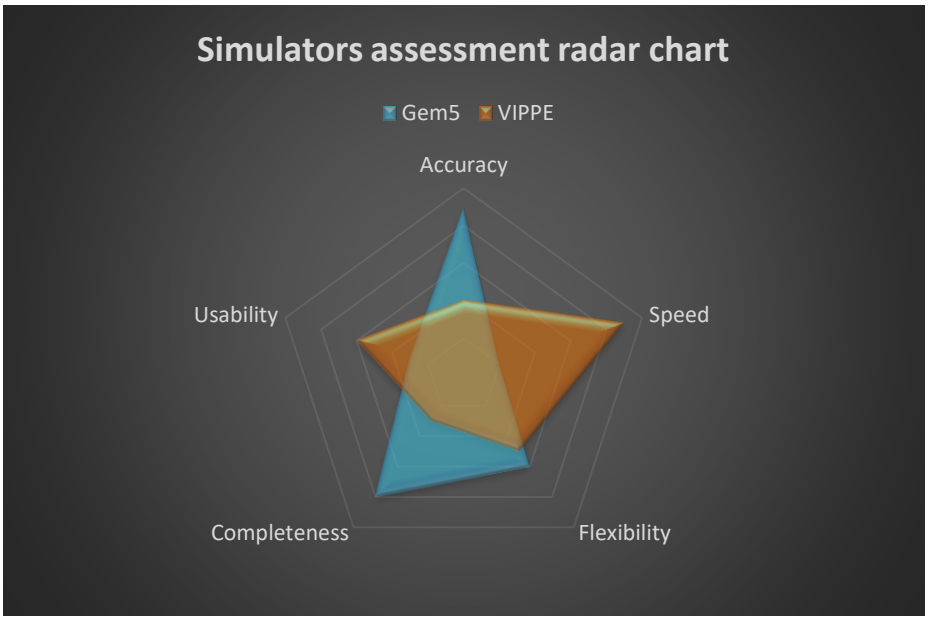
3 Conclusions

Gem5 is the most full-featured architecture simulator. It leverages execute-at-execute semantics for high-fidelity cycle-by-cycle simulation. But it's also true there are some points that makes it absolutely hostile to new users, as the gem5 developers seems to do not think of the user first.

- There's no well-defined static API and the interface to different modules is constantly changing.
- On-line documentation is incomplete, mostly outdated, and it contains a lot of broken resource links and doc pages.
- Besides it is an incredibly flexible configuration system, this process are confusing, and mostly inconsistent. There are many features that users depend on that are not covered by the documentation.
- Understanding of different parts of C++ code simulator infrastructure and python modelling are tedious and very frustrating.
- Running this benchmarking research I found a lot of incoherence results, as thousands of initializing and finishing instructions blocks added to user code that originally didn't existed.
- Debugging is unnecessary difficult, inaccessible for new users and very slow down process.
- Lack of new-user support in general.

VIPPE is a simulation and performance estimation tool for research field, far away from commercial purposes. It provides a clearly and simply understanding API to users and is extremely fast, and easy to use compared with gem5.
Despite of that it's not a totally stable tool or deeply tested software. Actually version 3.1 has some important developing issues yet.

- Cache model is unfinished and has some incoherencies at the time of this documentation making, as annotation tool has some bugs related to this.
- There are too few CPUs and ISAs modelled cases, so is user-side work to add and configure a specific study model if it's not already included.



4 References

[1] L. Díaz, E. González, E. Villar, P. Sánchez. "VIPPE: Parallel simulation and performance analysis of complex embedded systems". HiPPES4CogApp: High Performance, Predictable Embedded Systems for Cognitive Application. 2015-01

[2] L. Díaz, E. González, E. Villar, P. Sánchez. "VIPPE: Native simulation and performance analysis framework for multi-processing embedded systems". Proceedings of the JCE-Sarteco2014. 2014-09.

[3] QUEMU website. http://wiki.qemu.org/Main_Page. Last visit. Oct., 2014.

[4] OVP website. http://www.ovpworld.org/. Last visit. Oct., 2014.

[5] Coremu website. http://sourceforge.net/p/coremu/home/Home/. Last visited, August, 2015.

[6] GESE/UC UML/MARTE modelling methodology. Available in http://umlmarte.teisa.unican.es (see Documentation section). Last visited June, 22th, 2016.

[7] CONTREX Eclipse Plug-in site. http://contrep.teisa.unican.es . Last visited June, 22th, 2016.

[8] GEM5 Simulator Project. https://www.gem5.org/getting_started/.

[9] Complex Benchmark Algorithms stored at https://github.com/llvm-mirror/test-suite/tree/master/SingleSource/Benchmarks/BenchmarkGame