Emulation library for a modular cyber-physical systems simulation platform

Renato Oliveira, Cláudio Maia, and Luís Miguel Pinho

CISTER Research Centre, Porto, Portugal

CISTER – Research Centre in Real-Time & Embedded Computing Systems

Table of contents

Problem

- CPS tests may be expensive, i.e. aerospace systems
- CPS tests can be destructive, i.e. automotive systems
- Shorter development cycles require faster development

Our approach

- Emulation
  - Replicate the behavior of the real hardware, at the **functional** level
  - Otherwise these tests can’t be taken into account while testing these systems

Emulator requirements

- Full-system emulation
- Emulation of multiple architectures
- Adequate licensing
- Multi-processor system emulation capabilities

Candidates

- Bochs
- Gem5
- Emul8
- QEMU

Inadequate

Emulator of choice

QEMU's characteristics

- **NOT** user friendly
- Difficult to integrate with other modules
- Executes at the maximum speed possible

First problem

Second problem

Solving the first problem

- Library to make QEMU more user-friendly and integrable with other modules
- Written in C++

QEMU-Lib

- QEMU instance independence
- Communicating with QEMU

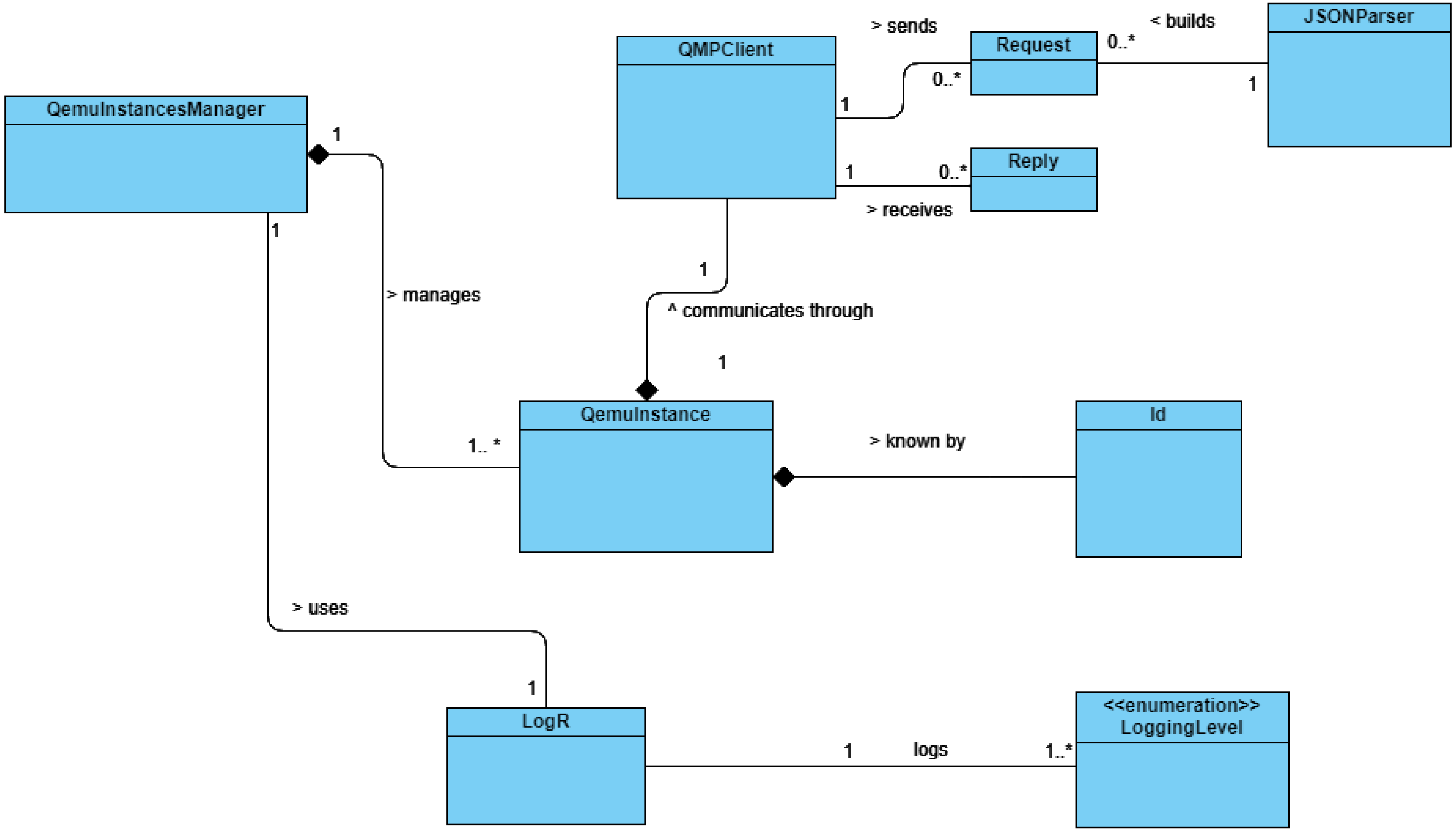
QEMU instance independence

- One virtual machine is one QEMU instance
- Library should be able to communicate with one independent instance

Communicating with QEMU

- QEMU Monitor
- QEMU Machine Protocol (QMP) via TCP
- Human-Monitor Commands (HMP) via TCP
- Library should parse simpler commands into QMP and HMP valid commands

Library architecture



Solution of the first problem

With the solution implemented it is now possible to:

- Provide control of individual instances
- Provide ease of access on configuring virtual machines
- Provide ease of integration with other modules (by exposing these features in a module to be consumed by other modules)

QEMU's characteristics

- ~~NOT user friendly~~
- ~~Difficult to integrate with other modules~~
- Executes at the maximum speed possible

~~First problem~~

Second problem

Solving the second problem

- By nature, QEMU runs at the maximum speed allowed by the *host CPU*
- Provide a mechanism to **throttle** the **virtual CPU** and the **virtual clocks**, in order to achieve a faithfull replication of the functional behavior of the real system

The throttle mechanism

- Reduce execution speed
- Reflect execution speed changes in QEMU's **clocks**

Reducing execution speed

- QEMU introduced a migration feature in version **2.4**
- This feature throttles the CPU in order to be able to complete **live migrations** (else these migrations could not complete)
- The **throttling** is in actuality induced via making the CPU sleep in timeslices measured in **microseconds**, varying these timeslices according to a percentage
- We take advantage of these facts and expose pieces of this **throttling** feature as a command

Reducing execution speed

However:

- The rest of the system does not reflect this change in the execution speed, I.e., the **clocks**
- This may cause desynchronization issues between components and communication with other systems

Reflect execution speed changes in QEMU's clocks

- We took advantages of the existing 'pause' mechanism, which pauses the virtual machine
- When the machine is paused, the virtual clock (QEMU_CLOCK_VIRTUAL) is paused
- We used pieces of the pause code to effectively stop the clock while the machine is **throttling**, making it only increase when the machine is actively executing

Solution of the second problem

The throttle mechanism provides:

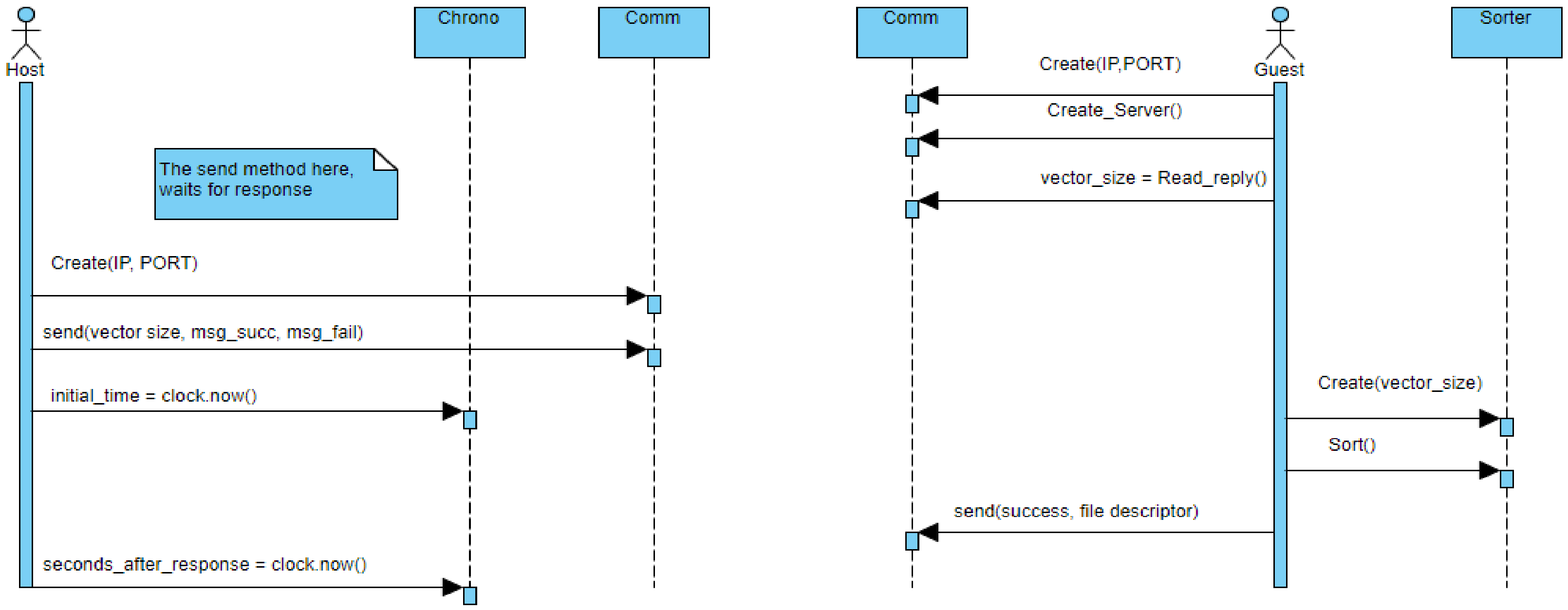
- A command which **throttles** the virtual machine according to a percentage
  - This change in execution speed is now reflected in the virtual machine's **clocks**

Results of the study

- Task execution times were measured via a client-server application, since the throttled machine's measures would be incorrect (since the clocks reflect the speed change)
- The task is the sorting of a vector of size 250000
- Sorting algorithm has $n * \log(n)$ complexity
- The measurements were performed in a x64 virtual machine running Ubuntu

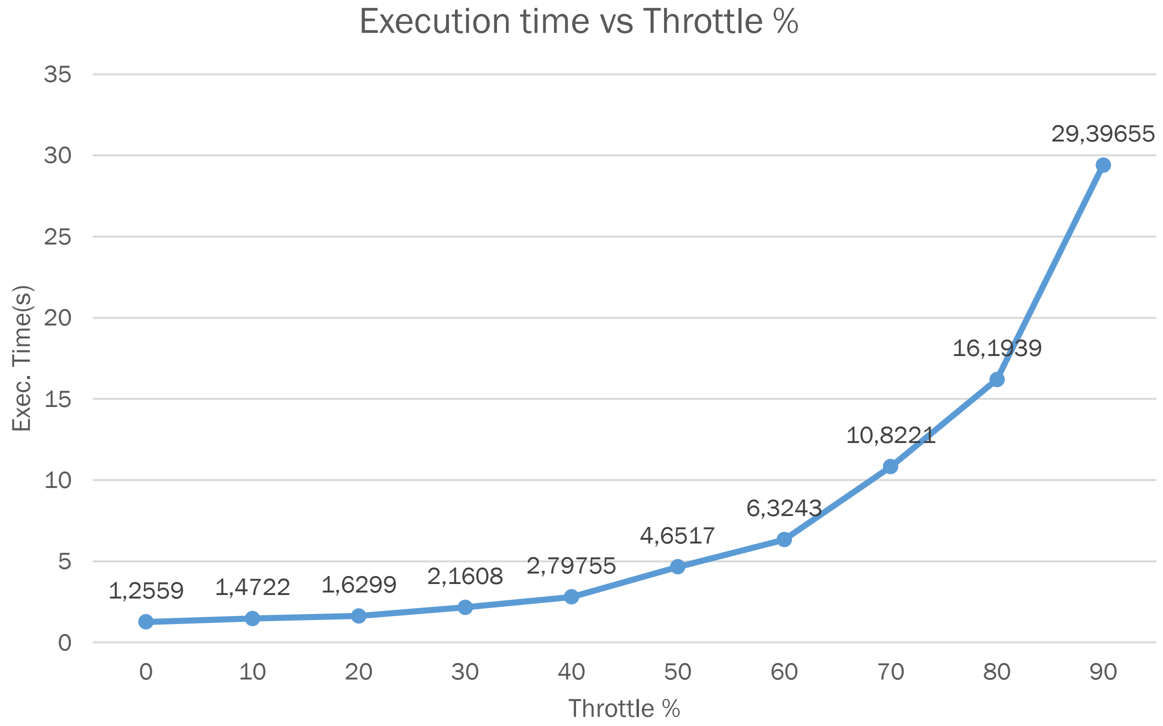
Results of the study

Test application architecture



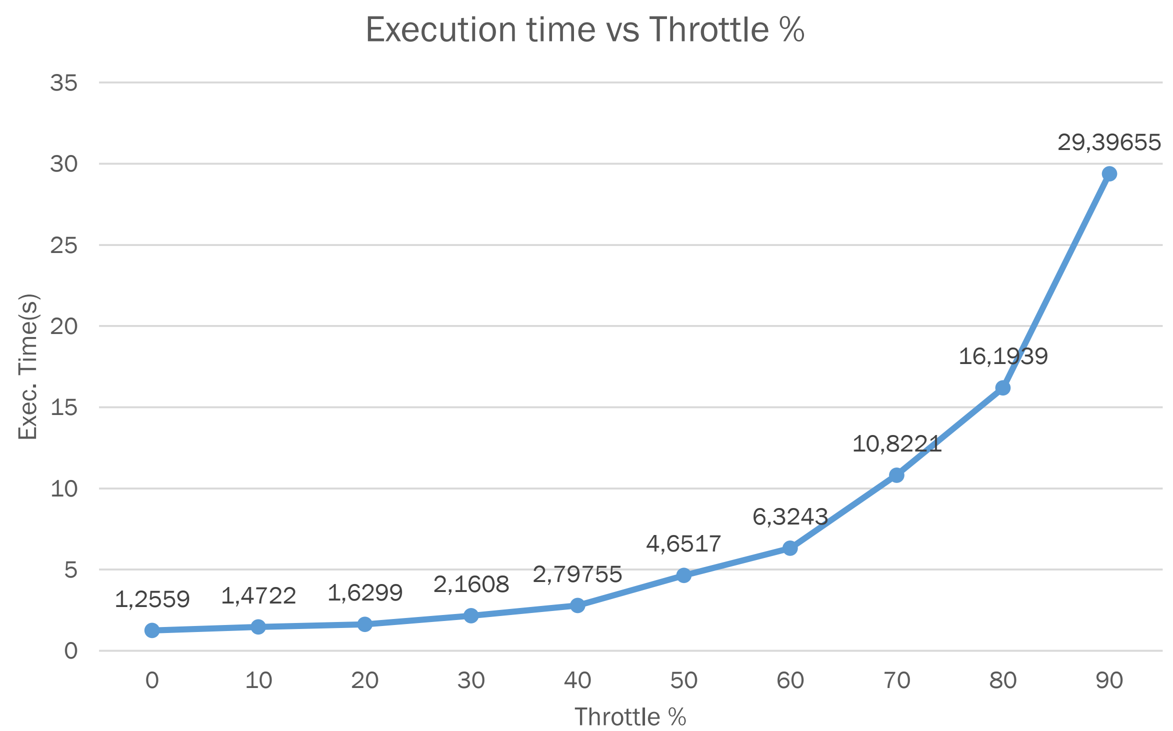
Results of the study

- Executing a task which takes 1.2559 seconds to complete with 0% throttle



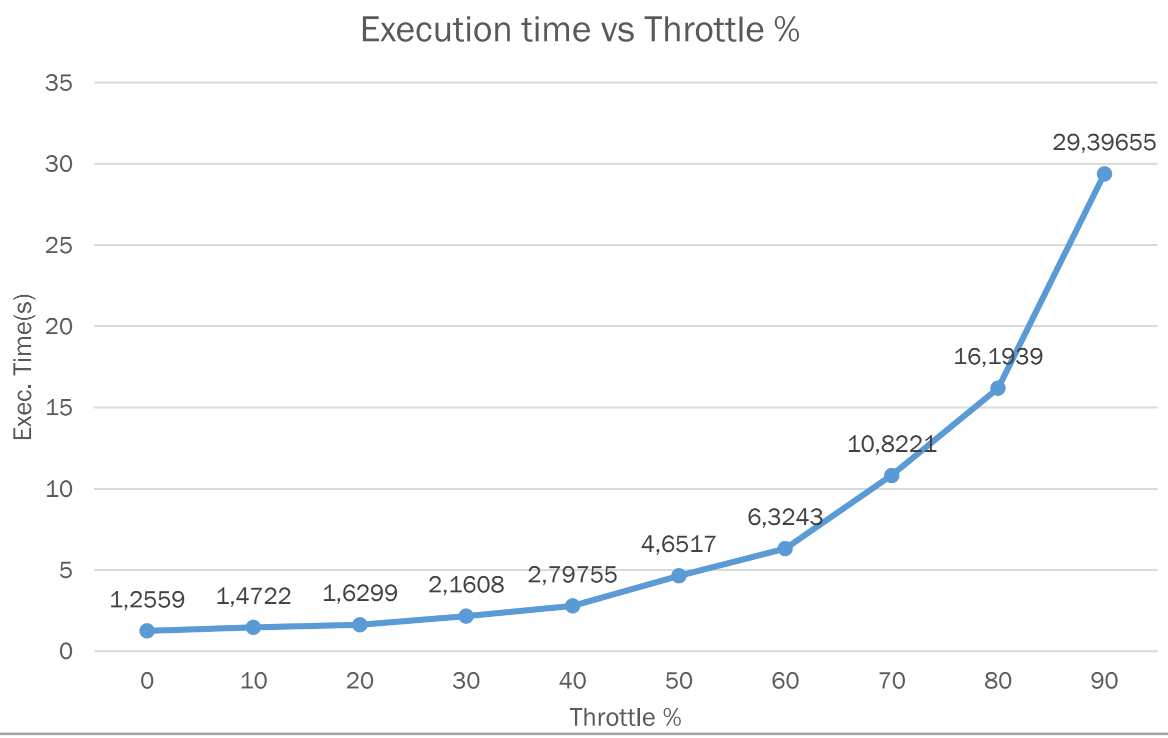
Results of the study

- As we can see, this same task, when executed in a machine submitted to 90% **throttling**, executes in approximately 29.39 seconds



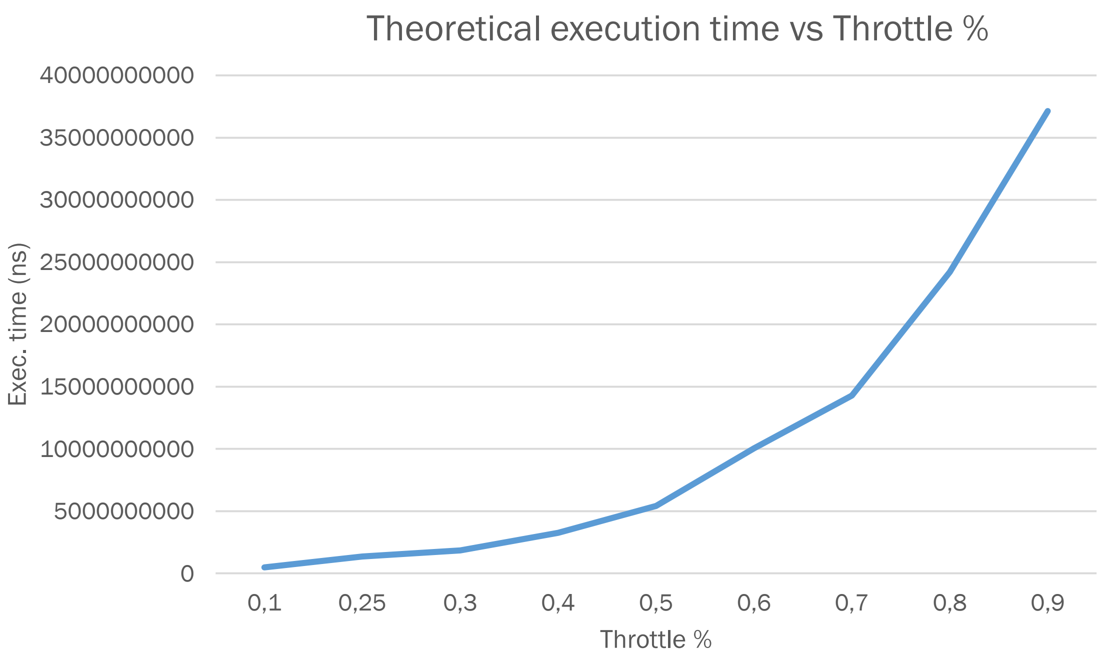
Results of the study

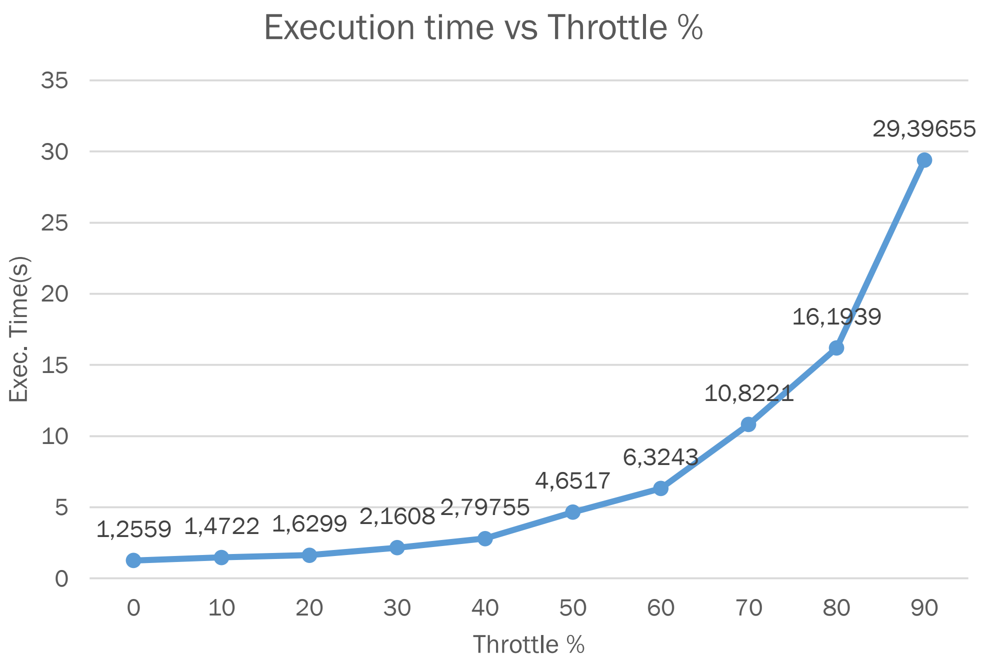
- After 40% throttle the curve begins to accentuate exponentially



Results of the study

- We can see that the measured curve is similar to the one obtained in a theoretical study of the throttle mechanism





Concluding remarks

- ~~NOT user friendly~~
- ~~Difficult to integrate with other modules~~
  - QEMU is now easier to use and easier to integrate with other modules
- ~~Executes at the maximum speed possible~~
  - QEMU's execution speed can now be controlled

~~First problem~~

~~Second problem~~

Thank you for your time