The evaluation of RISC-V HiFive1 board by using TOPPERS/ASP

December 13, 2019
Kioxia Corporation
Institute of Memory Technology Research & Development
System Technology Research & Development Center
Masahiro Yamada

KIOXIA

Agenda

1. Background
2. Goal
3. Approach
4. Issues in work process
5. Deal with these issues
6. Measurement result
7. Conclusion

RISC-V becomes popular

- RISC-V Foundation
- ISA (Instruction set architecture) is open
  - Don’t say the implementation is open, but there are some open RISC-V implementations
- Adopt often used instruction set (scrap and build)
  - Simple = the size of CPU would be small, but code size would increase
  - Enable to select standard extension which are really used
- Approach to echo system like toolchain, simulator/VM, OS porting
- Domain specific extension
  - Strive for performance of application
  - pros and cons : porting issues, maintenance cost

Points of common and difference: Is RISC-V similar with Linux®?

Selection of configuration

- RISC-V : Select standard extension
- Linux : Select Linux kernel modules

Foundation

- RISC-V Foundation
- The Linux Foundation

Open Source ?

- Universities and companies open the RISC-V implementation, they are not using same language
- Linux kernel has only 1 repository, and C language.

⇒ Because the resources of OSS development are distributed, the merit of OSS is not so high.

Evaluation of Embedded CPU

Increasing choices of CPU for embedded system is good.

⇒ Evaluate for that embedded system can replaced current CPU with RISC-V

⇒ The benchmark of CPU performance like CoreMark® was done

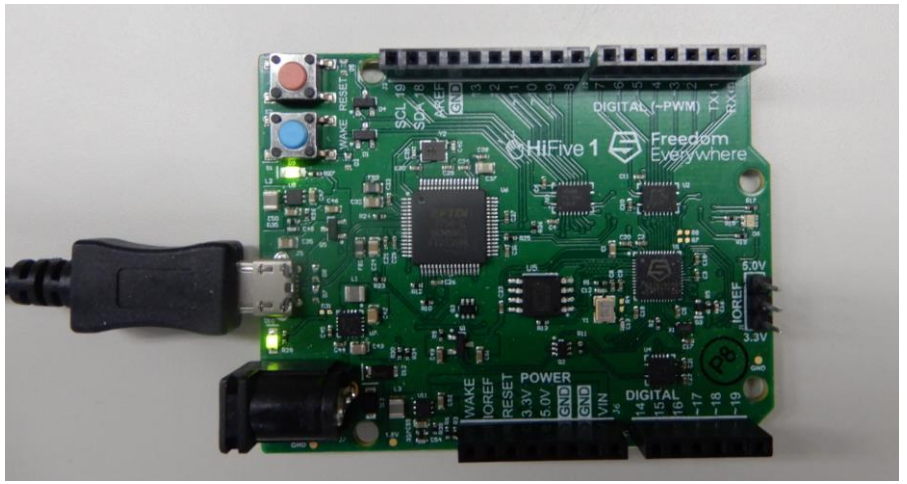
Evaluate RISC-V from RTOS point of view assuming Embedded system

- **Code size(text size)**
- **Measurement of jitter by perf**

Don’t evaluate from HW point of view in this presentation

- The number of gate
- Estimate limit performance of CPU frequency

Measurement of code size and perf by TOPPERS/ASP on HiFive1



System configuration for evaluation

- HW : HiFive1 board (not rev. B) of SiFive
  - Easy to get
  - Open the implementation (FPGA version)
- RTOS : TOPPERS/ASP
  - By architecture independent implementation, compare the code size
  - ASP has perf, which can measure some points like act_tsk in RTOS.

Just for reference, also measure NUCLEO-F401RE (Arm® Cortex®-M4). However, CoreMark performance : 3.42 CoreMarks/MHz (※1) is not suitable for comparing

※1: https://developer.arm.com/ip-products/processors/cortex-m/cortex-m4

About HiFive1 board and FE310-G000

- RISC-V board compatible with Arduino
- FE310-G000 RISC-V SoC
  - RV32IMAC：RV32 standard instruction(I), (M)ultiply, (A)tomic, (C)ompressed
  - Instruction cache : 16KB
  - DTIM : 16KB
  - Performance
    - By Data sheet : 2.73 CoreMarks/MHz (※１)
    - By actual survey : 1.496 CoreMarks/MHz (※２)
- SPI Flash memory

※1: https://sifive.cdn.prismic.io/sifive%2Ffeb6f967-ff96-418f-9af4-a7f3b7fd1dfc_fe310-g000-ds.pdf
※２：http://msyksphinz.hatenablog.com/entry/2017/03/22/014143

Introduction to TOPPERS/ASP

- About TOPPERS/ASP (※1)
  - Based on ITRON specification, developed highly complete Real-time kernel
  - Major target is embedded system needs high reliability, safety, real-time performance.
  - In terms of software volume, the program size (binary code) of major target is from several tens MB to 1MB
  - ASP (Advanced Standard Profile) = TOPPERS JSP (standard profile conform to μITRON4.0 spec) was extended and improved
- As other kernels, there are HRP kernel(memory protection) and SMP kernel(Multicore)
- ASP3 : Add tickles feature into ASP kernel. Configurator was developed by Ruby.
- From May 2019, RISC-V(HiFive1 board) is supported by ASP and the source is opened.

※ 1 : https://www.toppers.jp/asp-kernel.html

Build TOPPERS/ASP in build environment ⇒ measure the code size (1)

Construct build environment for TOPPERS/ASP of RISC-V

Procedure	Description
Install toolchain	Download from SiFive's HP : riscv64-unknown-elf-gcc-8.3.0-2019.08.0-x86_64-linux-ubuntu14.tar.gz
Get asp source code and tool	Download from TOPPERS's HP Common code: asp-1.9.3.tar.gz Target specific code : asp_arch_riscv32_gcc-1.9.3.tar.gz Configurator : cfg-linux-static-1_9_3.gz

Setting multi arch for Host (Configurator run in 32bit environment)	$ sudo dpkg --add-architecture i386
	$ sudo aptitude update
	$ sudo aptitude install -y libc6-dev-i386

Fix asp source code	Compiler name : riscv32-unknown-elf→riscv64-unknown-elf Change STACK_SIZE : default 4K > less than 1536 B

Build TOPPERS/ASP in build environment ⇒ measure the code size (2)

Build sample for measuring code size

```
$ mkdir asp/OBJ-SAMPLE
$ cd asp/OBJ-SAMPLE
$../configure -T ../target/hifive1_gcc/
$ make clean && make depend && make
```

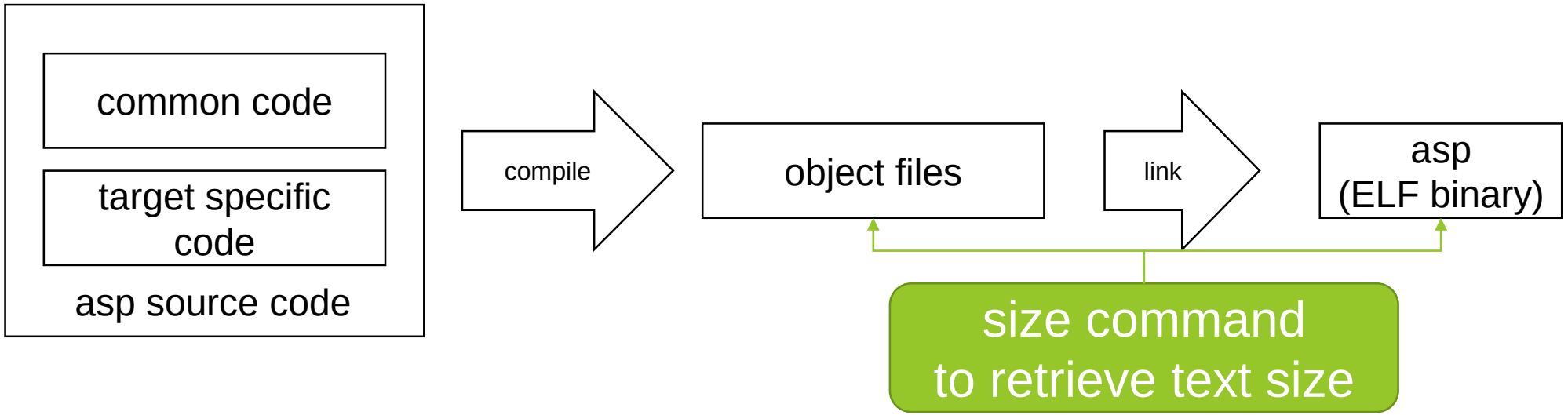
Bulid each perf for measuring latency histogram (e.g. perf0)

```
$ mkdir asp/OBJ-PERF0
$ cd asp/OBJ-PERF0
$ ../configure -T ../target/hifive1_gcc/ -A perf0 -a ../test -U "test_lib.o histogram.o"
$ make clean && make depend && make
```

Build TOPPERS/ASP in build environment ⇒ measure the code size (3)

Measure code size of *.o and asp binary by size command

- each object files of common code
  - Don’t measure target specific code and generated code by configurator
- Measure asp (ELF binary), include target specific code



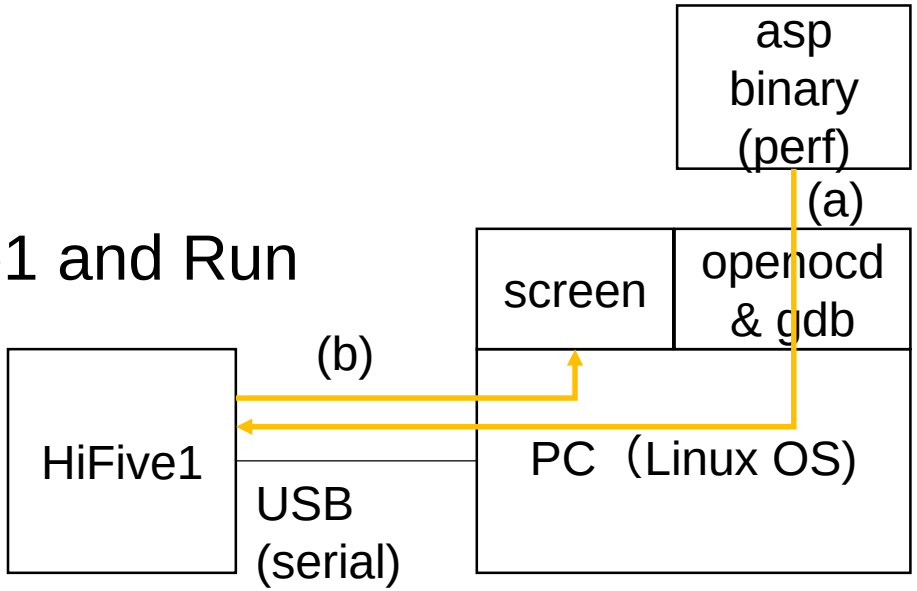
Measure latency of important point by running perf

Setup runtime environment

- Install screen
- Install openocd, prep for openocd.cfg
- Install gdb, prep for .gdbinit

Runtime

- (a) By openocd & gdb, write perf binary in HiFive1 and Run
  - text: 0x20000000(SPI Flash)
  - data: 0x80000000(DTIM)
- (b) Get perf's result by screen



Evaluation program perf of TOPPERS/ASP

perf0

- Do nothing with loop, measuring the overhead of timestamp function

perf1

- Time between when low priority task do wup_tsk to high priority task and when the high priority task start to run
- Time between when high priority task do slp_tsk and when low priority task start to run

perf4

- The processing time of act_tsk without task switch
- The processing time of act_tsk with task switch
- The processing time of iact_tsk from cyclic handler (not task context)

After each perf run 10000 times, perf output the result with the histogram format

Issues in work process

Problem1 : It requires a lot of work to construct build/runtime environment

- Install toolchain (Build toolchain)
- Retrieve source code and tool from several web sites
- To run cfg(configurator of asp) needs 32 bit environment
  - Some Linux distribution don’t support 32bit
  - Need to multi arch setting for 32bit binary in Debian
- Need to fix source code for measuring
- Need to prepare for openocd cfg file
- Need to prepare for gdb init file

⇒ Solution1 : Create docker container for build/runtime environment of TOPPERS/ASP

Problem2 : The bug of default get_utm (timestamp) function

- Run perf0 with default. The result is 0 or 1000 or INT_MAX

```
Performance evaluation program (0)
Measurement overhead
0 : 9988
> 1000 : 10
> INT_MAX : 2
```

- By test_utm1, found default get_utm doesn't have monotonous increase

```
system performance time goes back: 18616001(CYC) 18616000(TSK)
system performance time goes back: 18647001(TSK) 18647000(CYC)
system performance time goes back: 18649001(TSK) 18649000(CYC)
system performance time goes back: 18663001(TSK) 18663000(CYC)
system performance time goes back: 18667001(TSK) 18667000(CYC)
```

⇒ Solution2: Implement get_utm function to get timestamp with micro seconds order

Problem3 : The low CPU frequency

- TOPPERS/ASP on NUCLEO-F401RE run with CPU 84Mhz
- On the other hand, TOPPERS/ASP on HiFive1 run with CPU 16MHz
- The difference of CPU frequency is too large

⇒ **Solution3 : Setting of PLL, change the CPU frequency.**

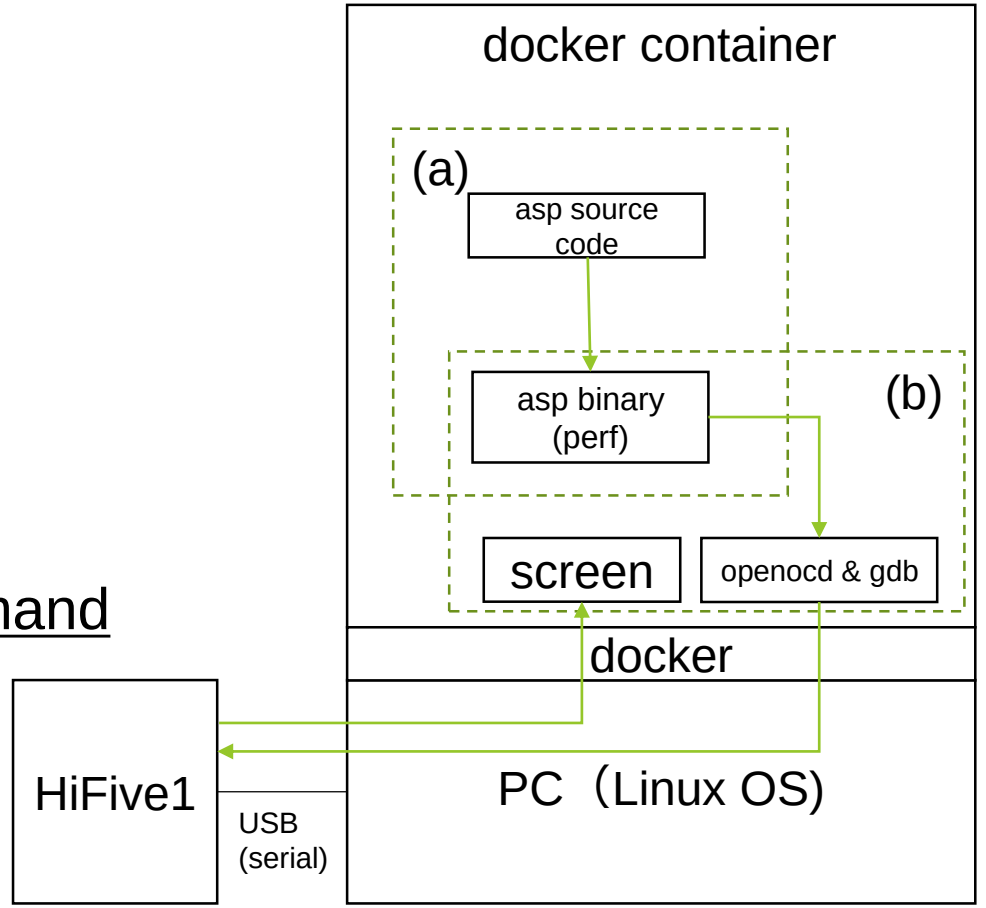
Solution1 : Easy to construct build/runtime environment by docker

(a) Build asp by docker build command

- Install some packages to build and run asp
- Retrieve asp source code and configurator
- Get toolchain and install it
- Setting of multi arch for 32bit binary
- Apply some patch files for asp source code
- Build asp to create sample and perf binaries.

(b) Run asp (perf) on HiFive1 board by docker run command

- Run screen command in docker
- Run openocd command in docker
- Run gdb command with perf binary in docker



Solution2 : Implement get_utm function to get timestamp with micro seconds

Refer to registers which count up by CPU frequency

- mcycle
- mcycleh

Measure CPU frequency at boot time (ASP do by default)

- SystemFrequency

⇒ Calculate micro seconds timestamp from register values

```
+#ifdef OMIT_GET_UTM
+#include "target_timer.h"
+ER
+get_utm(SYSUTM *p_sysutm)
+{
+    SYSUTM  utime = 0;
+    uint32_t mcycle_low, mcycle_high;
+    uint32_t mcycle_high_tmp;
+
+    do {
+        mcycle_high_tmp = read_csr(mcycleh);
+        mcycle_low = read_csr(mcycle);
+        mcycle_high = read_csr(mcycleh);
+    } while (mcycle_high_tmp != mcycle_high);
+
+    /* change clock count into micro sec */
+    utime += (SYSUTM)(mcycle_low / (SystemFrequency / 1000000));
+    utime += (SYSUTM)(UINT_MAX / (SystemFrequency / 1000000) * mcycle_high);
+
+    *p_sysutm = utime;
+
+    return(E_OK);
+}
+#endif
```

Solution3 : Change CPU frequency by setting of PLL

- TOPPERS ASP run with CPU frequency 16Mhz by default
- By macro DEFAULT_CLOCK
  - According to PLL specification, it run with 256Mhz
  - Measuring actual CPU frequency on HiFive1, it run with more than 280Mhz
- Modify PLL setting to change CPU frequency
  - According to PLL specification, R=1 F=41 Q=3 will be 84Mhz, but the actual CPU frequency on HiFive1 was 94Mhz
  - By R=1 F=37 Q=3, the actual CPU frequency on HiFive1 is almost 86Mhz.

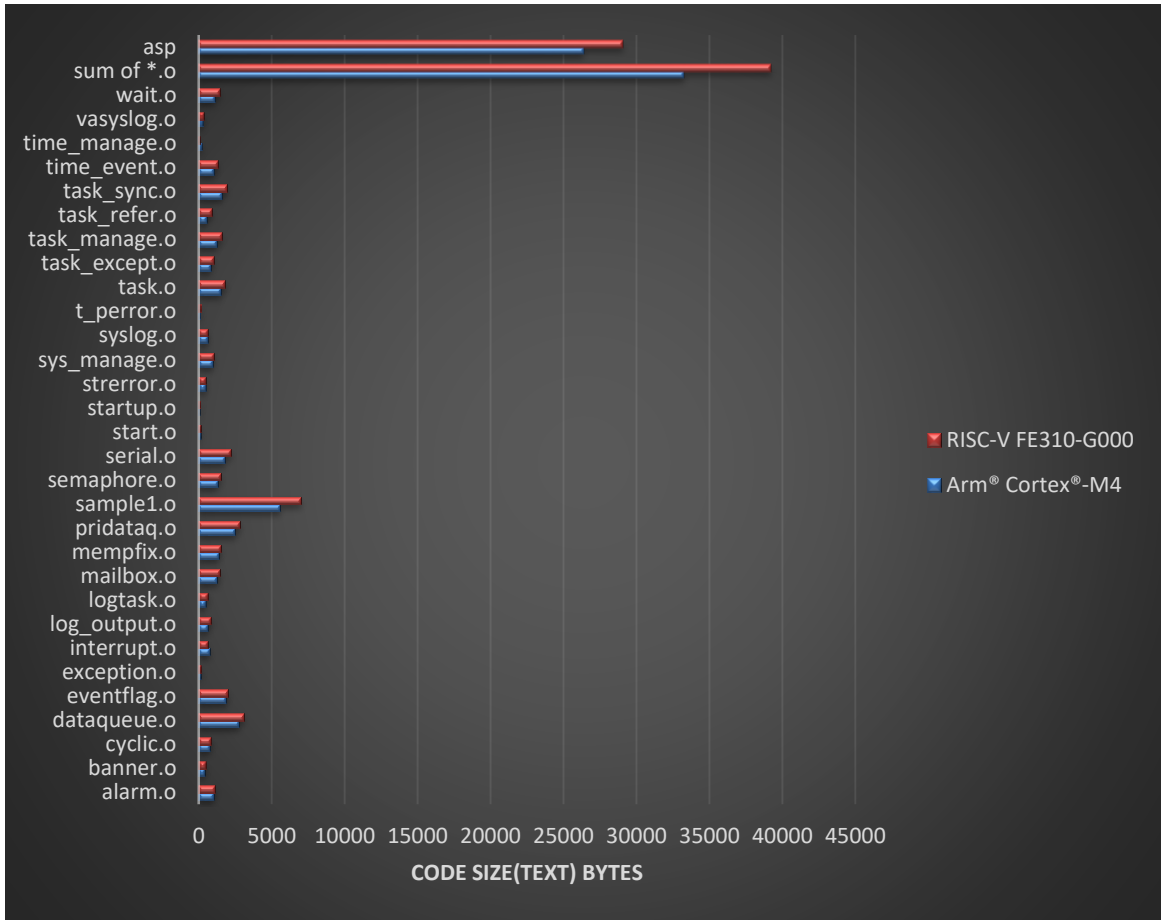
Evaluation of code size

Compiler (both use compressed instructions)

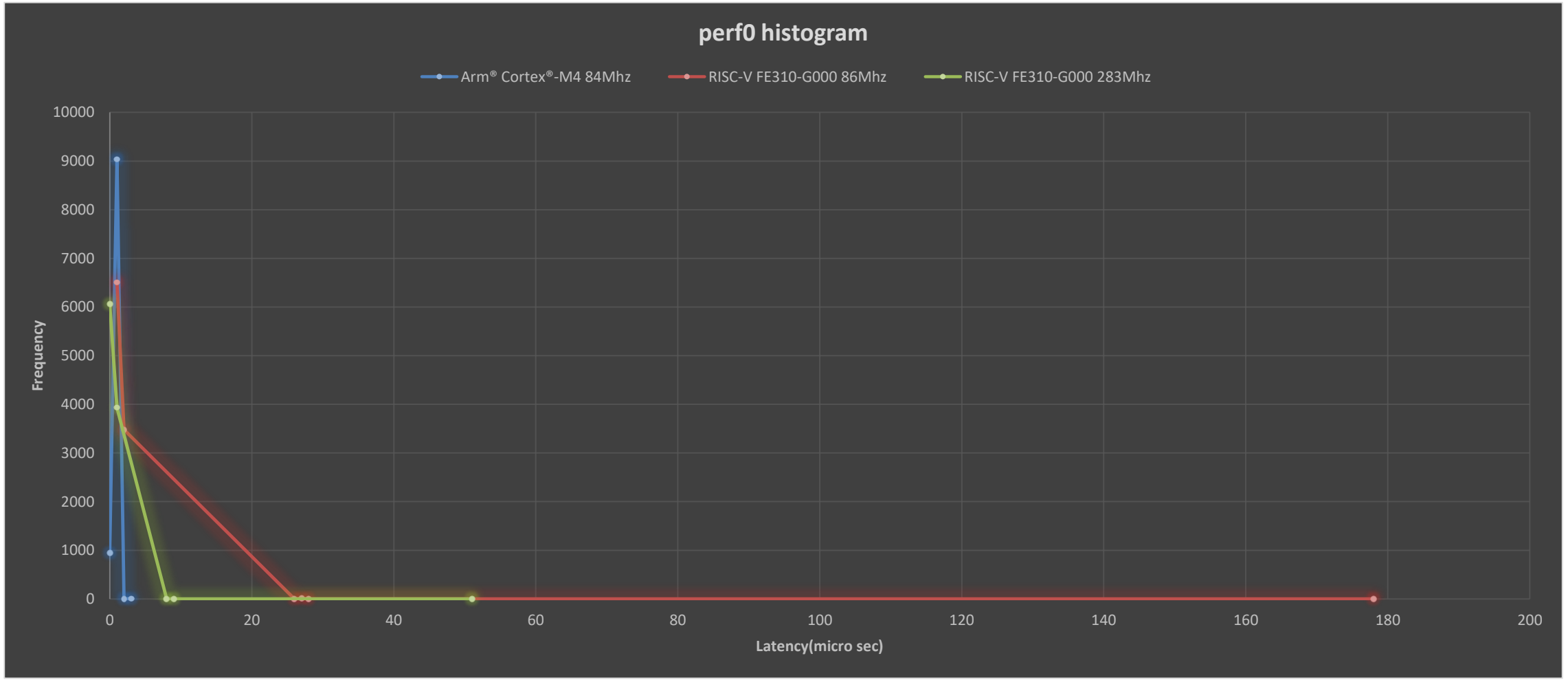
- Arm® core：gcc version 8.3.1 20190703 (release) [gcc-8-branch revision 273027] (GNU Tools for Arm Embedded Processors 8-2019-q3-update)
- RISC-V： gcc version 8.3.0 (SiFive GCC 8.3.0-2019.08.0)

Comparison of code size (text size)

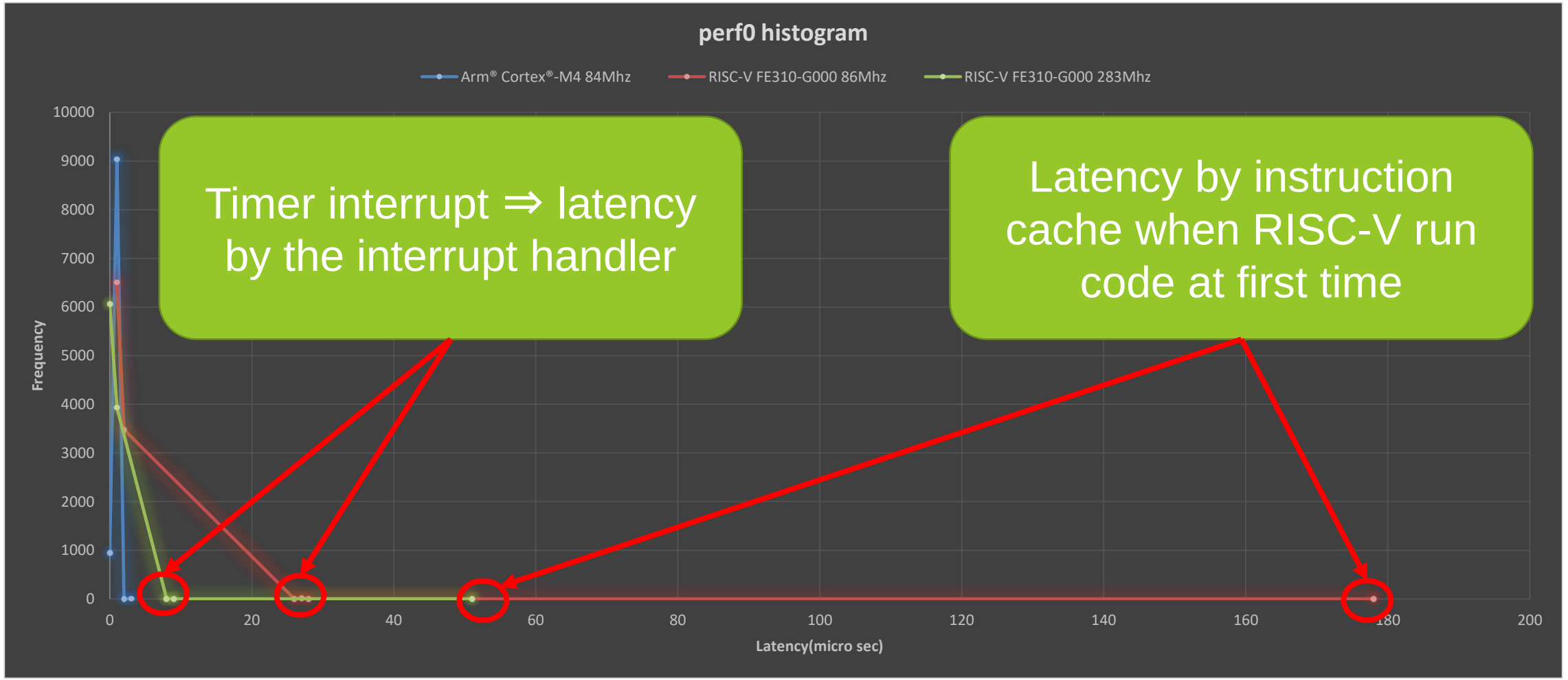
- RISC-V increases 17.8 % than Arm® core, regarding to sum of common object files
- RISC-V increases 10.3 % than Arm® core, regarding to asp bianry



perf0 : Evaluation of overhead when measuring

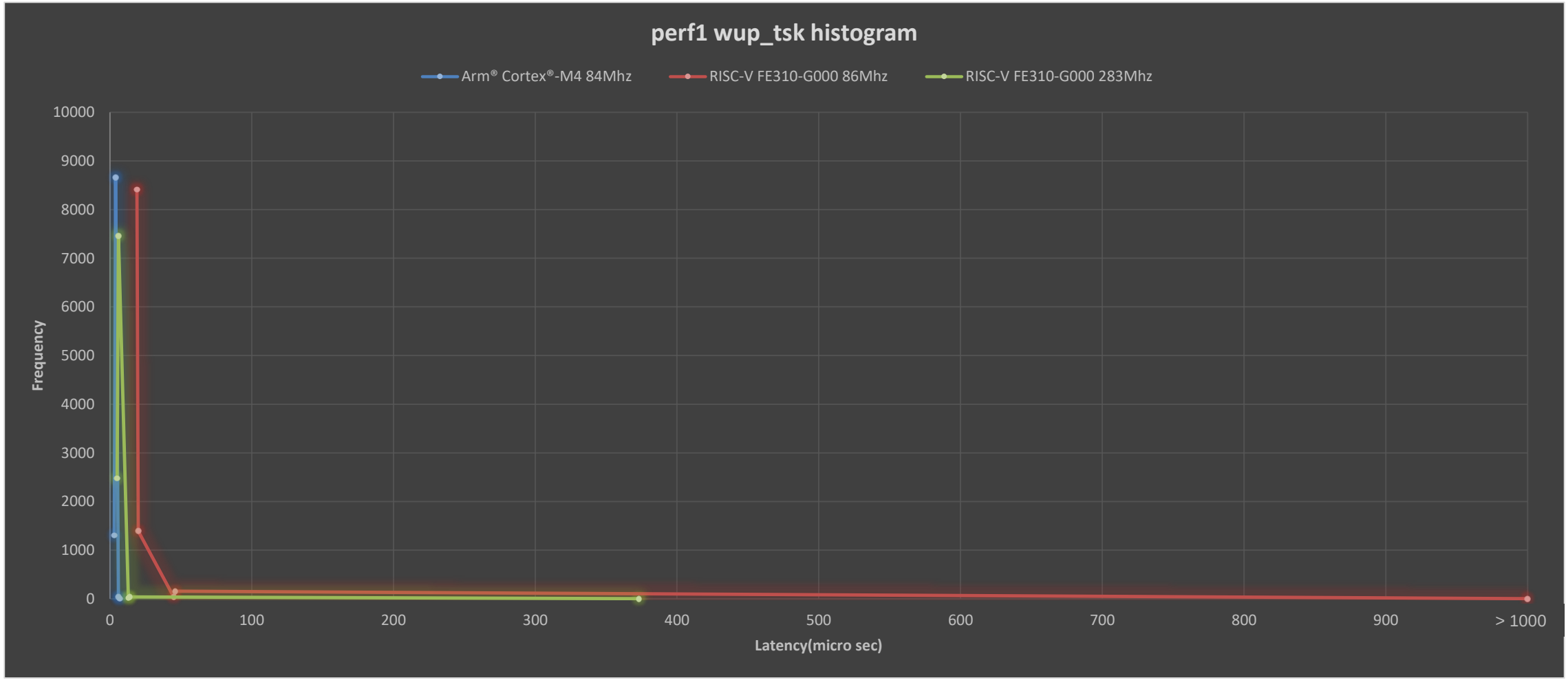


perf0 : Evaluation of overhead when measuring : consideration to latency

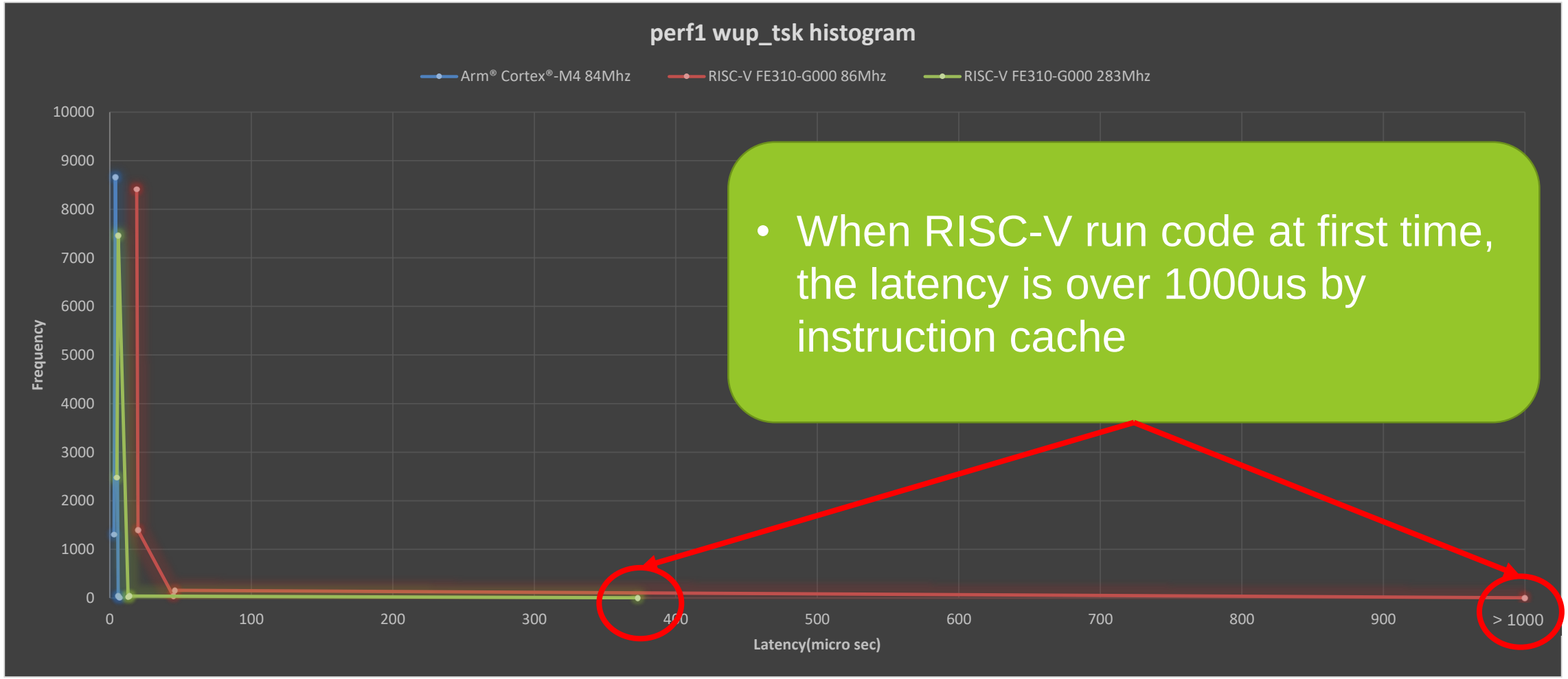


Measurement result

perf1 : Evaluation of task switch by wup_tsk

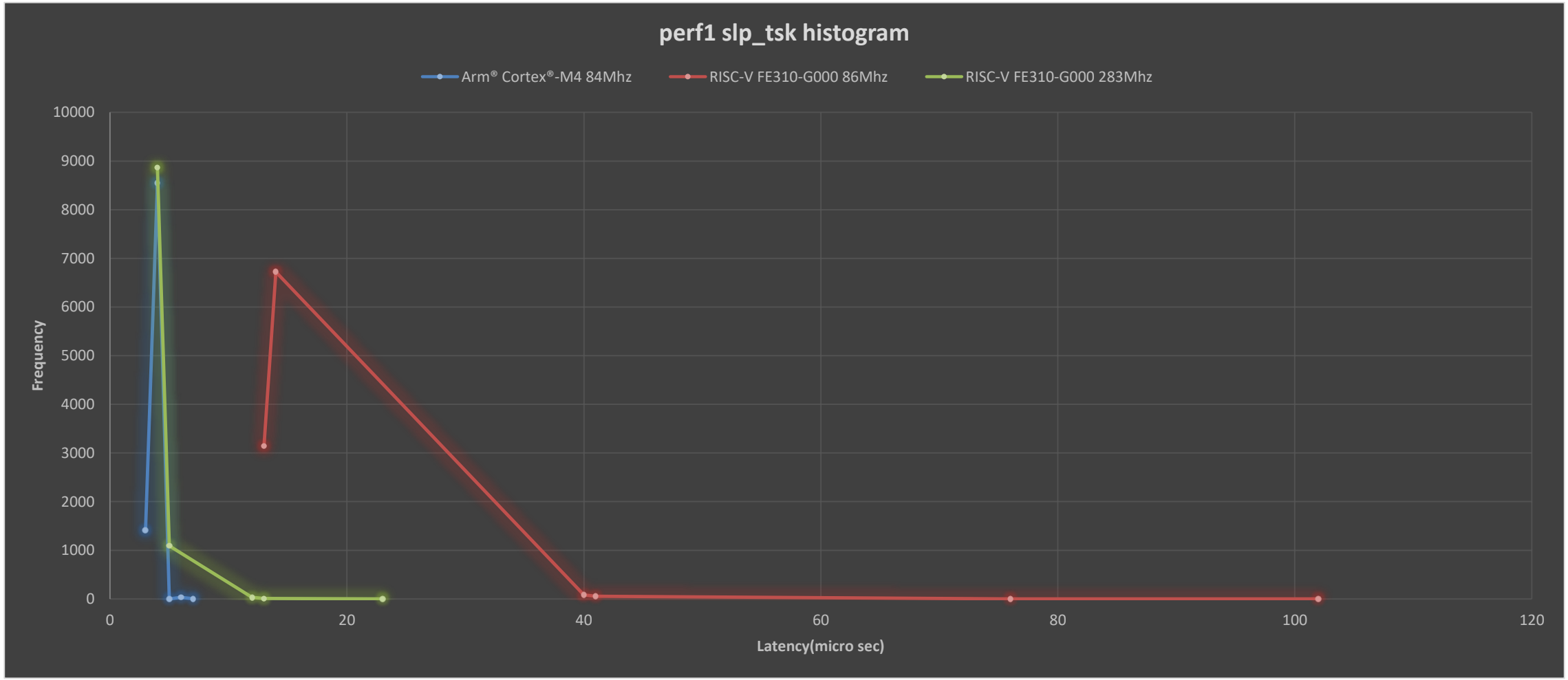


perf1 : Evaluation of task switch by wup_tsk : consideration to latency

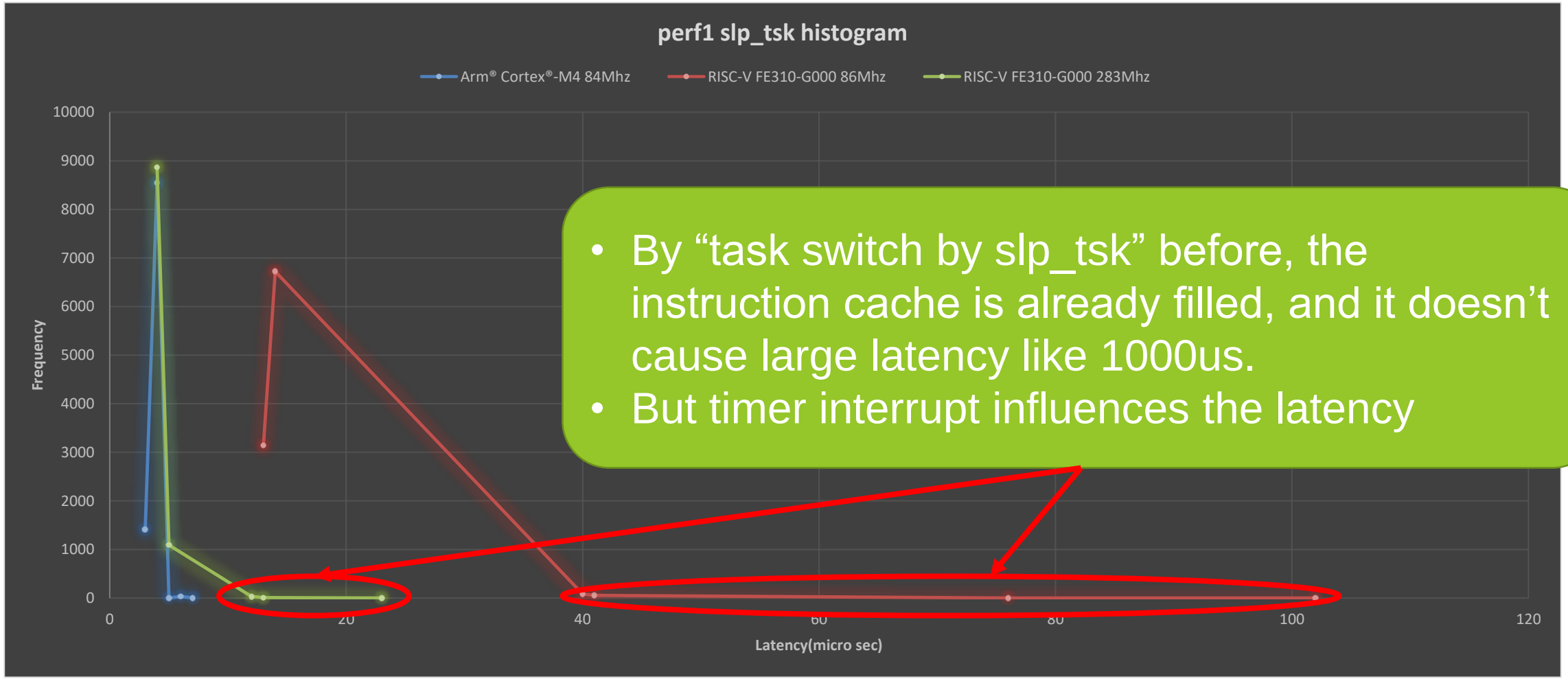


Measurement result

perf1 : Evaluation of task switch by slp_tsk

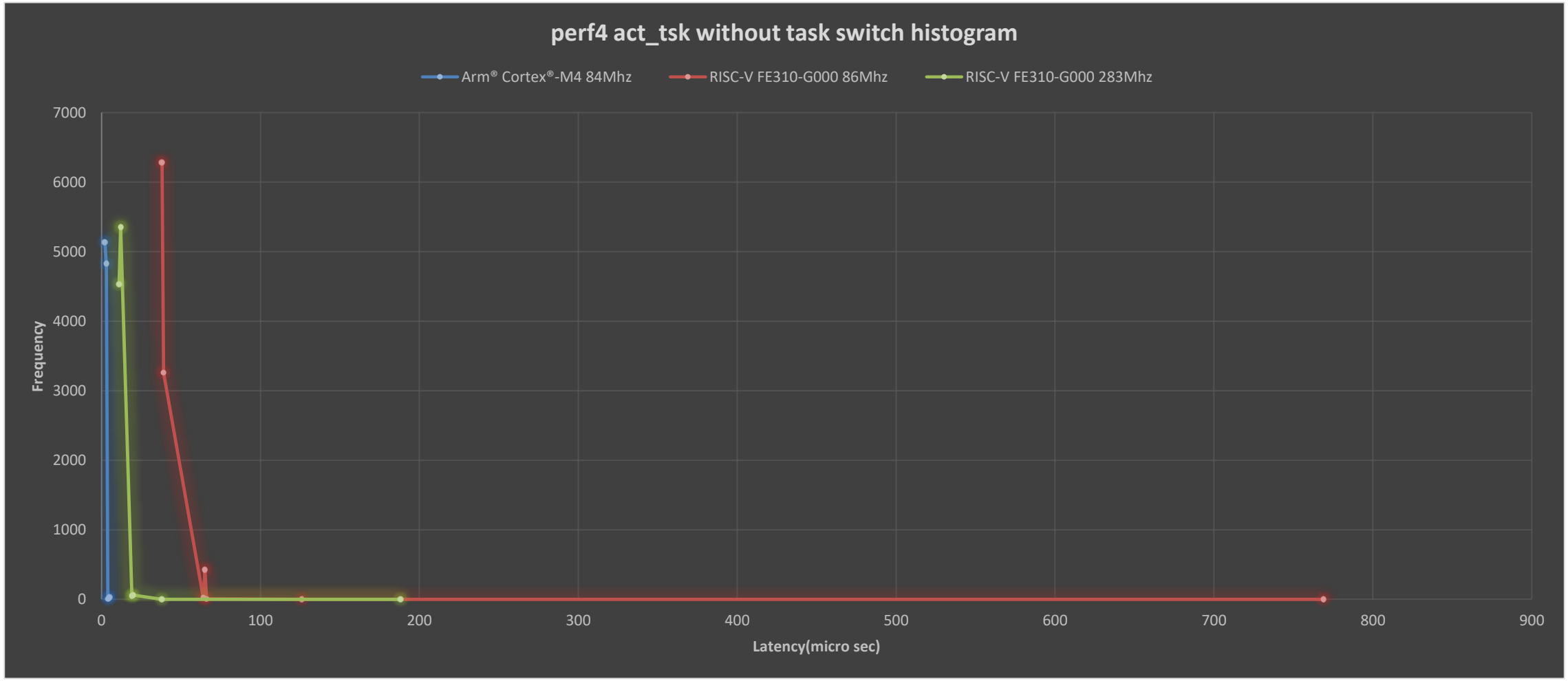


perf1 : Evaluation of task switch by slp_tsk : consideration to latency

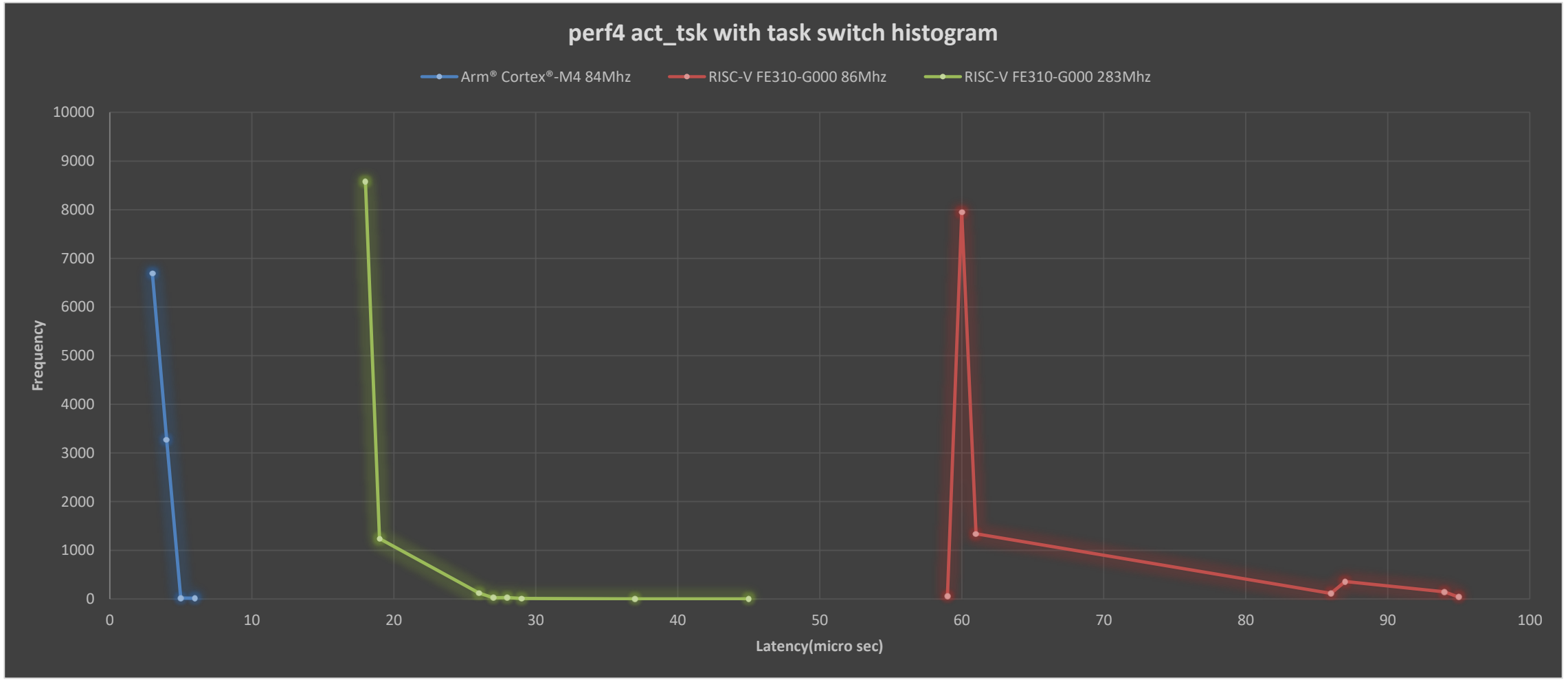


- By “task switch by slp_tsk” before, the instruction cache is already filled, and it doesn’t cause large latency like 1000us.
- But timer interrupt influences the latency

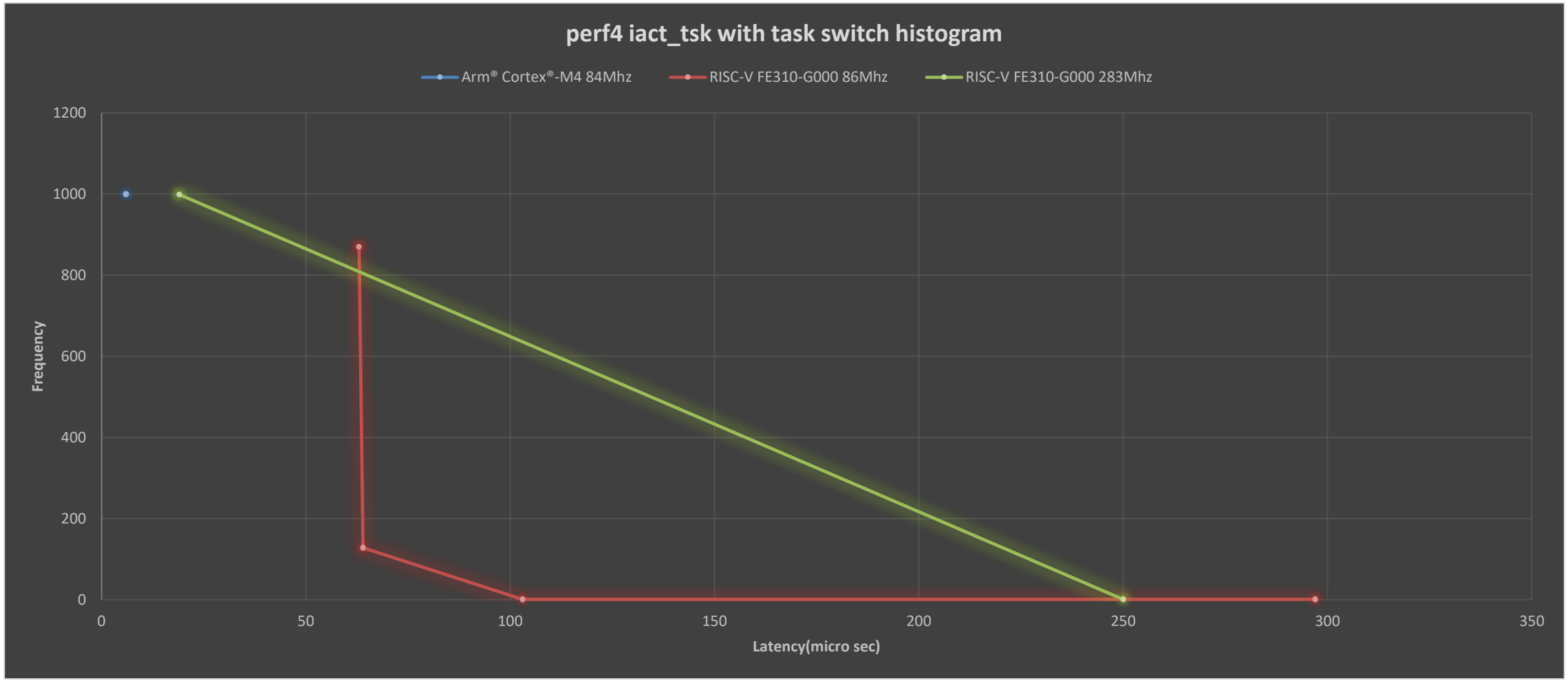
perf4 : Evaluation of act_tsk without task switch



perf4 : Evaluation of act_tsk with task switch



perf4 : Evaluation of iact_tsk from cyclic handler



Evaluate code size and perf by TOPPERS/ASP on HiFive1

- Problems and Solutions
  - Requires a lot of work to construct build/runtime environment ⇒ easy to do by docker
  - The bug of timestamp function ⇒ Add get_utm for this evaluation
  - CPU frequency ⇒ Change CPU Frequency by PLL
- Evaluation result: code size
  - Sum of object files of ASP for RISC-V increases 17.8 % than ASP for Arm® Cortex®-M4
- Evaluation result: latency by perf
  - Instruction cache and timer interrupt influence latency
  - If there is a kind of timer like systic of Arm® core, it could reduce the influence.
  - For Instruction cache, measurement is needed, like FW load text in boot time or run the code at once before application start

KIOXIA