RISE QEMU function benchmarks

These are hand-written versions of common memory and string library functions provided by SiFive Inc. For each benchmarked function, we are comparing a “baseline” QEMU against a “latest” QEMU. By default, we look at three versions of the code.

1. the standard library “scalar” implementation of the function
2. the hand-written vector code for VLEN=128 and LMUL=1 (“small vector”)
3. the hand-written vector code for VLEN=1024 and LMUL=8 (“large vector”)

We thus have a total of 6 datasets.

There are four graphs. The first looks at the average number of instructions executed per iteration, and is a sanity check. Although we plot all 6 datasets, only 3 should show on the graph - the “latest” versions of “scalar”, “small vector” and “large vector”, since the version of QEMU should have no impact on the number of instructions being executed. This graph is useful to developers of the vector implementations to see how their code behaves for different sizes of data, but this is outside the scope of this project.

The remaining three graphs capture QEMU performance for each of the three versions of the code. They use the metric of nanoseconds per instruction, to measure the efficiency of QEMU. Each graph shows this metric, plotted against problem size, one line for the “baseline” version of QEMU, the other the “latest” QEMU.

This report is generated entirely automatically using the command

```
./run-all-benchmarks.py
```

This takes less than 20 minutes to run on a 40 thread AMD Threadripper 1950X at 3.4GHz. The code is in the `strmem-benchmarks` directory of the rise-rvv-tcg-qemu-tooling repository. It is intended to be portable, but to date has only been tested on Ubuntu 22.04 LTS.

Document details

- Datestamp: 2024-12-05-16-41-57
- User: craig

Functions to be benchmarked

Any functions which failed to benchmark are noted.

- memchr
- memcmp

- memcpy
- memmove
- memset
- strcat
- strchr
- strcmp
- strcpy
- strlen
- strncat
- strncmp
- strncpy
- strnlen

QEMU versions

- 248f9209ed
- 9e5145d045

Tool chain configuration

GCC configuration

```
Using built-in specs. COLLECT_GCC=riscv64-unknown-linux-gnu-gcc COLLECT_LTO_WRAPPER=/home/craig/work/proj
ects/rise-qemu/install/libexec/gcc/riscv64-unknown-linux-gnu/14.1.0/lto-wrapper Target:
riscv64-unknown-linux-gnu Configured with: /home/craig/work/projects/rise-qemu/gcc/configure
--target=riscv64-unknown-linux-gnu --prefix=/home/craig/work/projects/rise-qemu/install
--with-sysroot=/home/craig/work/projects/rise-qemu/install/sysroot --with-pkgversion=gcd0059a1976-dirty
--with-system-zlib --enable-shared --enable-tls --enable-languages=c,c++,fortran --disable-libmudflap
--disable-libssp --disable-libquadmath --disable-libsanitizer --disable-nls --disable-bootstrap
--src=/home/craig/work/projects/rise-qemu/gcc --disable-default-pie --enable-multilib --with-abi=lp64d
```

```
--with-arch=rv64gc --with-tune= --with-isa-spec=20191213 'CFLAGS_FOR_TARGET=-O2    -mcmodel=medany'
'CXXFLAGS_FOR_TARGET=-O2    -mcmodel=medany' Thread model: posix Supported LTO compression algorithms:
zlib gcc version 14.1.0 (gcd0059a1976-dirty)
```

Assembler version

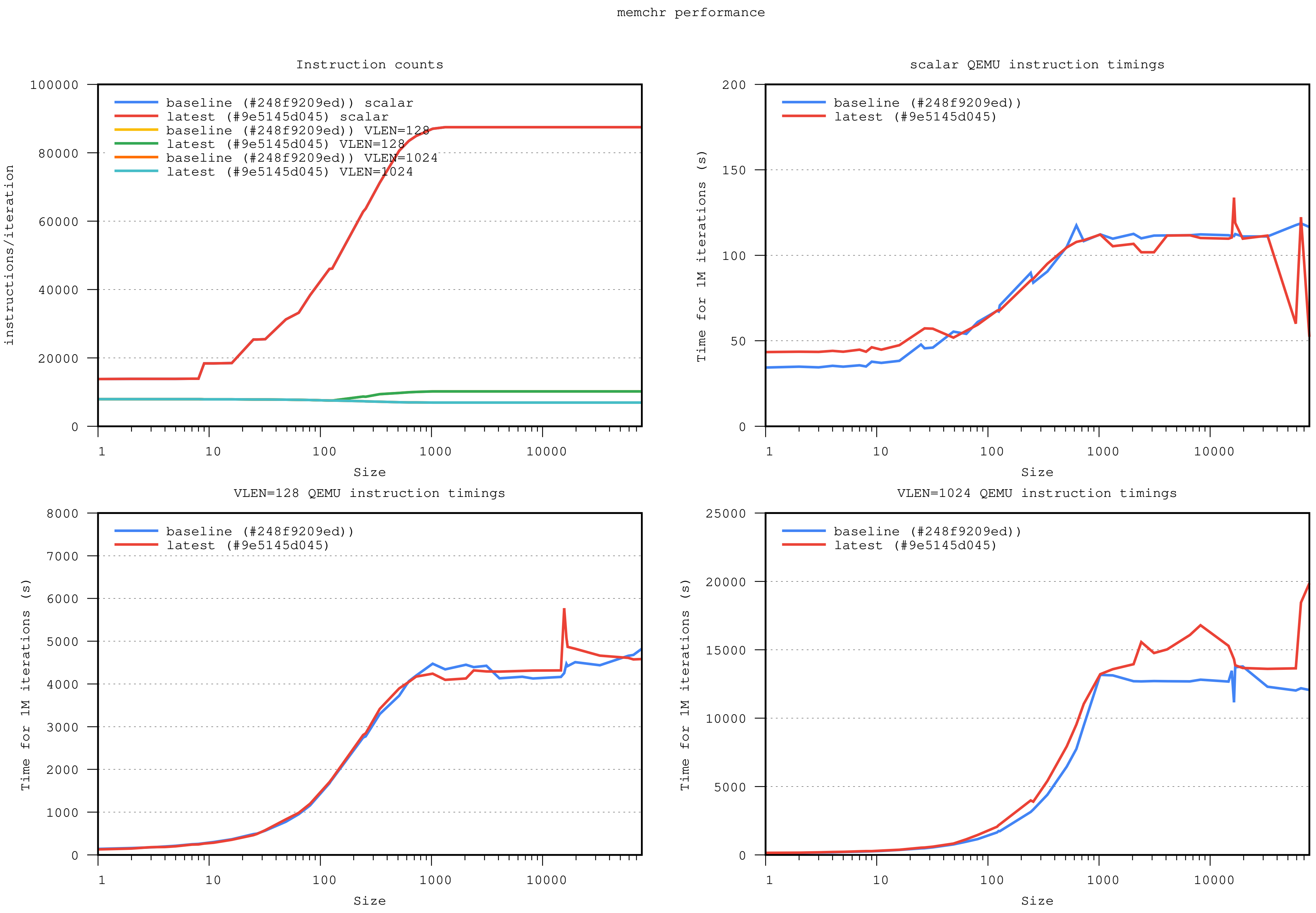
```
GNU assembler version 2.42 (riscv64-unknown-linux-gnu) using BFD version (GNU Binutils) 2.42
```

Linker version

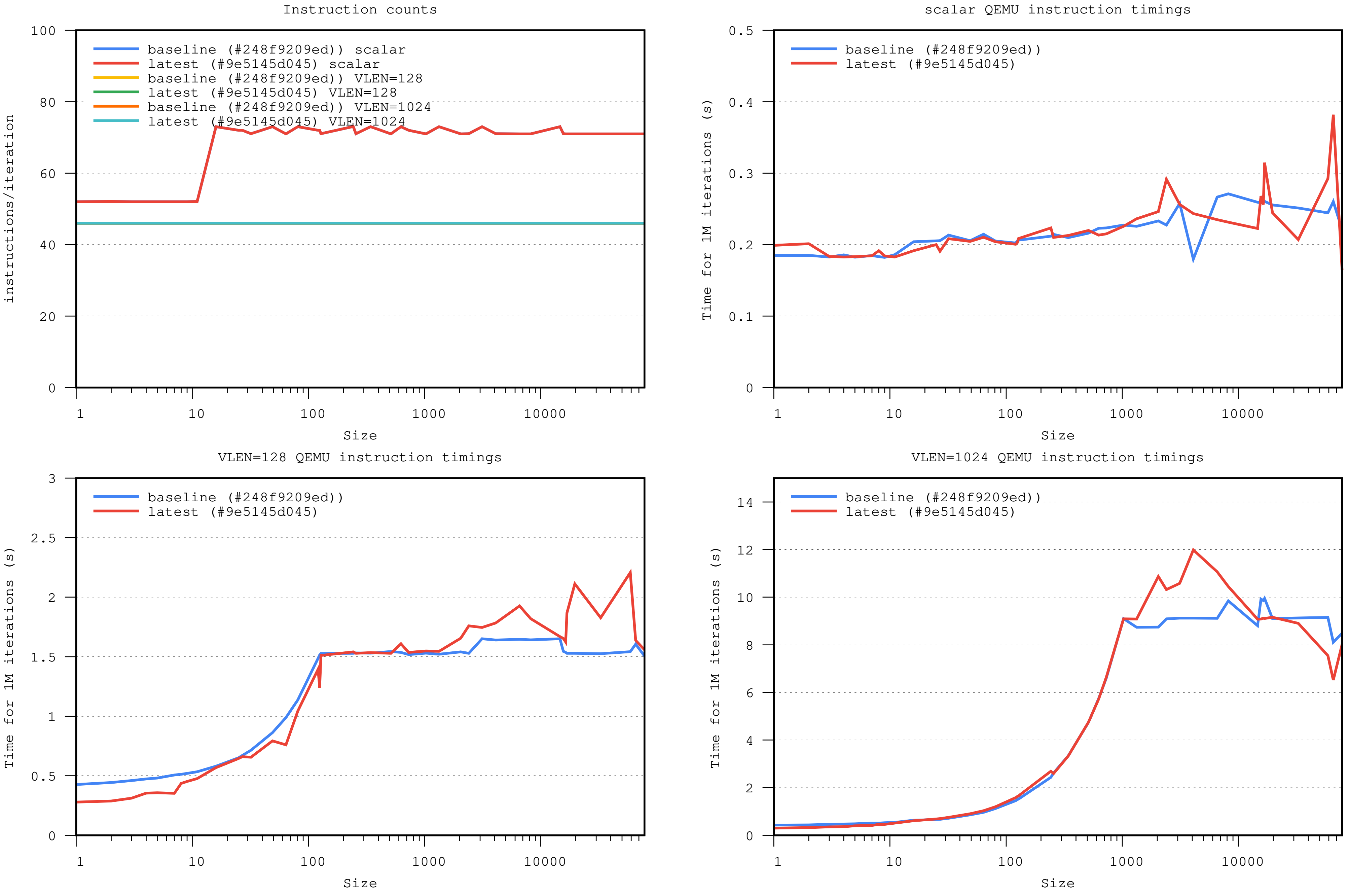
```
GNU ld (GNU Binutils) 2.42
```

Glibc version

```
ldd (GNU libc) 2.39 Copyright (C) 2024 Free Software Foundation, Inc. This is free software; see the
source for copying conditions.  There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE. Written by Roland McGrath and Ulrich Drepper.
```


memchr performance
Instruction counts
baseline (#248f9209ed)) scalar
latest (#9e5145d045) scalar
baseline (#248f9209ed)) VLEN=128
latest (#9e5145d045) VLEN=128
baseline (#248f9209ed)) VLEN=1024
latest (#9e5145d045) VLEN=1024
instructions/iteration
0
20000
40000
60000
80000
100000
1
10
100
1000
10000
Size
scalar QEMU instruction timings
baseline (#248f9209ed))
latest (#9e5145d045)
Time for 1M iterations (s)
0
50
100
150
200
1
10
100
1000
10000
Size
VLEN=128 QEMU instruction timings
baseline (#248f9209ed))
latest (#9e5145d045)
Time for 1M iterations (s)
0
1000
2000
3000
4000
5000
6000
7000
8000
1
10
100
1000
10000
Size
VLEN=1024 QEMU instruction timings
baseline (#248f9209ed))
latest (#9e5145d045)
Time for 1M iterations (s)
0
5000
10000
15000
20000
25000
1
10
100
1000
10000
Size

memcmp performance

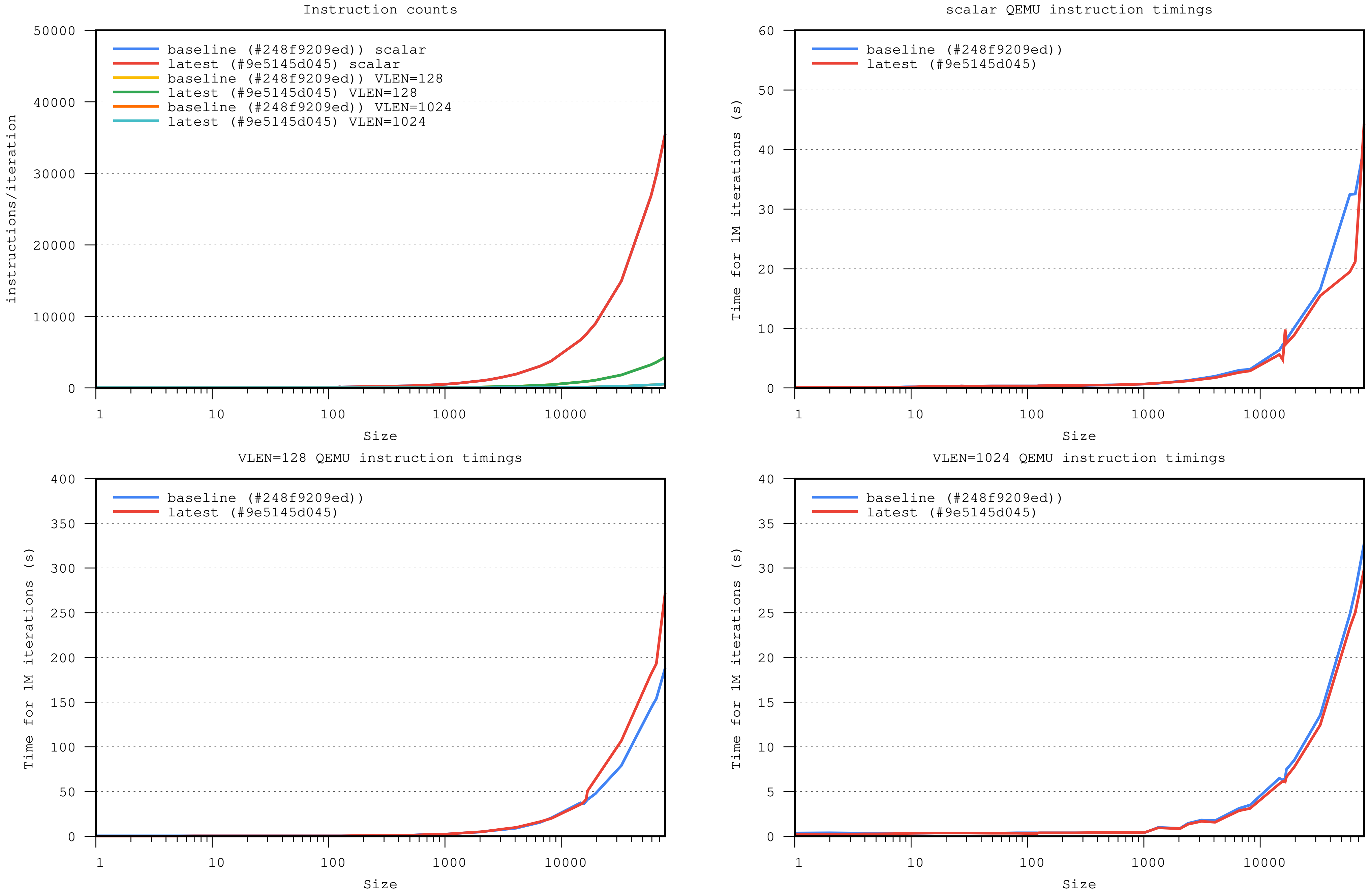

Instruction counts

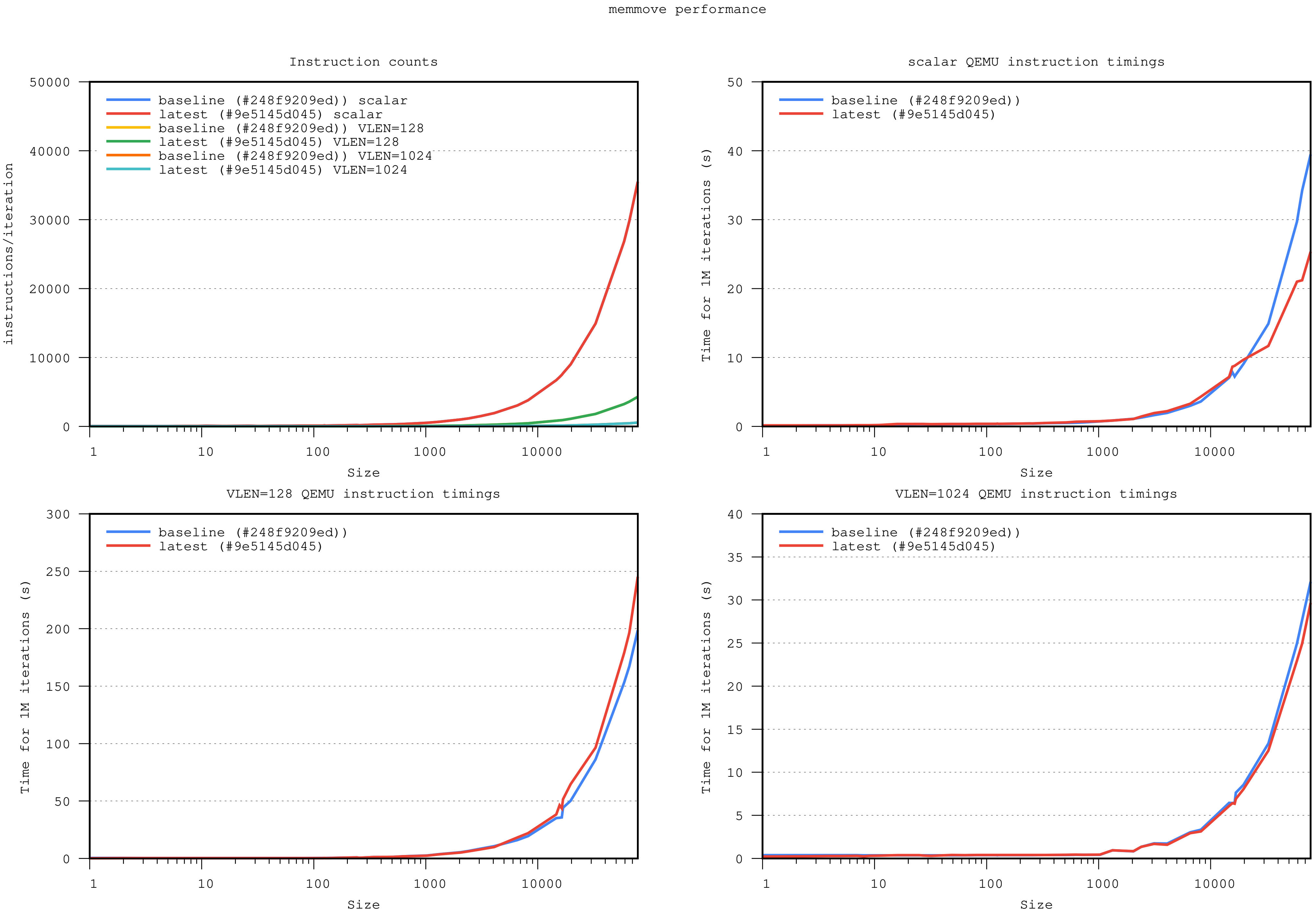
scalar QEMU instruction timings

VLEN=128 QEMU instruction timings

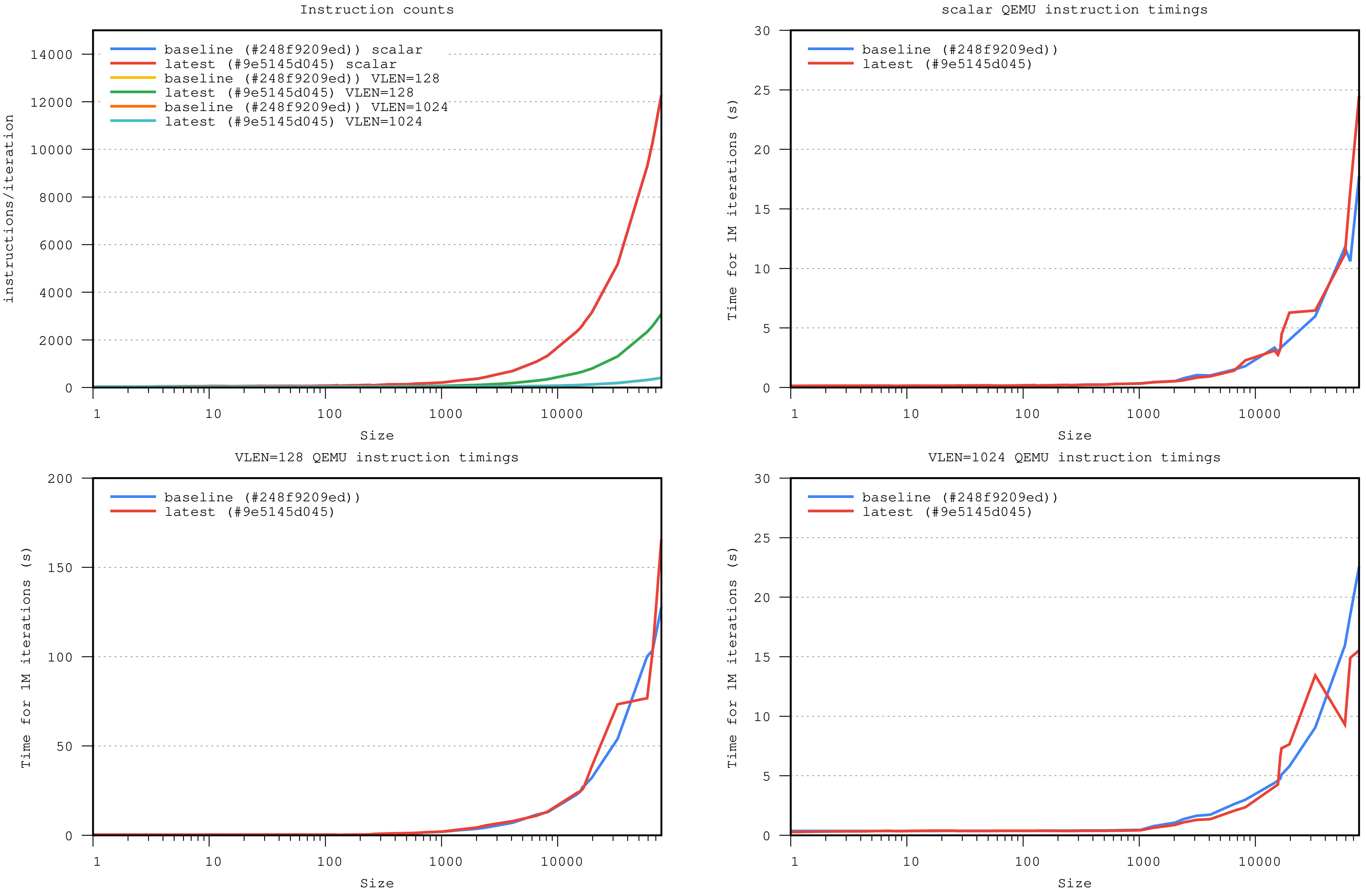
VLEN=1024 QEMU instruction timings

memcpy performance

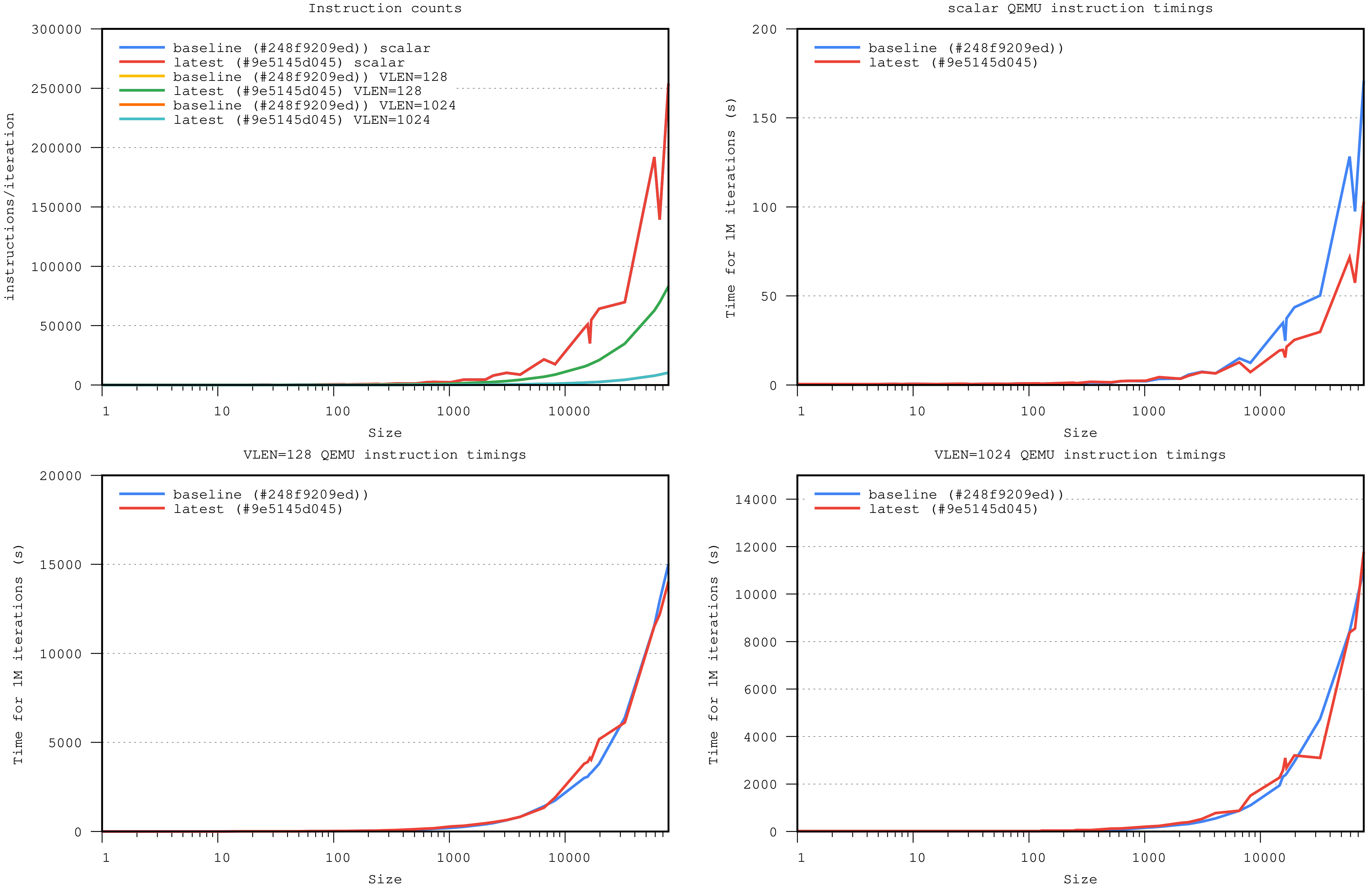



memmove performance
Instruction counts
instructions/iteration
50000
40000
30000
20000
10000
0
baseline (#248f9209ed)) scalar
latest (#9e5145d045) scalar
baseline (#248f9209ed)) VLEN=128
latest (#9e5145d045) VLEN=128
baseline (#248f9209ed)) VLEN=1024
latest (#9e5145d045) VLEN=1024
1
10
100
1000
10000
Size
scalar QEMU instruction timings
Time for 1M iterations (s)
50
40
30
20
10
0
baseline (#248f9209ed))
latest (#9e5145d045)
1
10
100
1000
10000
Size
VLEN=128 QEMU instruction timings
Time for 1M iterations (s)
300
250
200
150
100
50
0
baseline (#248f9209ed))
latest (#9e5145d045)
1
10
100
1000
10000
Size
VLEN=1024 QEMU instruction timings
Time for 1M iterations (s)
40
35
30
25
20
15
10
5
0
baseline (#248f9209ed))
latest (#9e5145d045)
1
10
100
1000
10000
Size

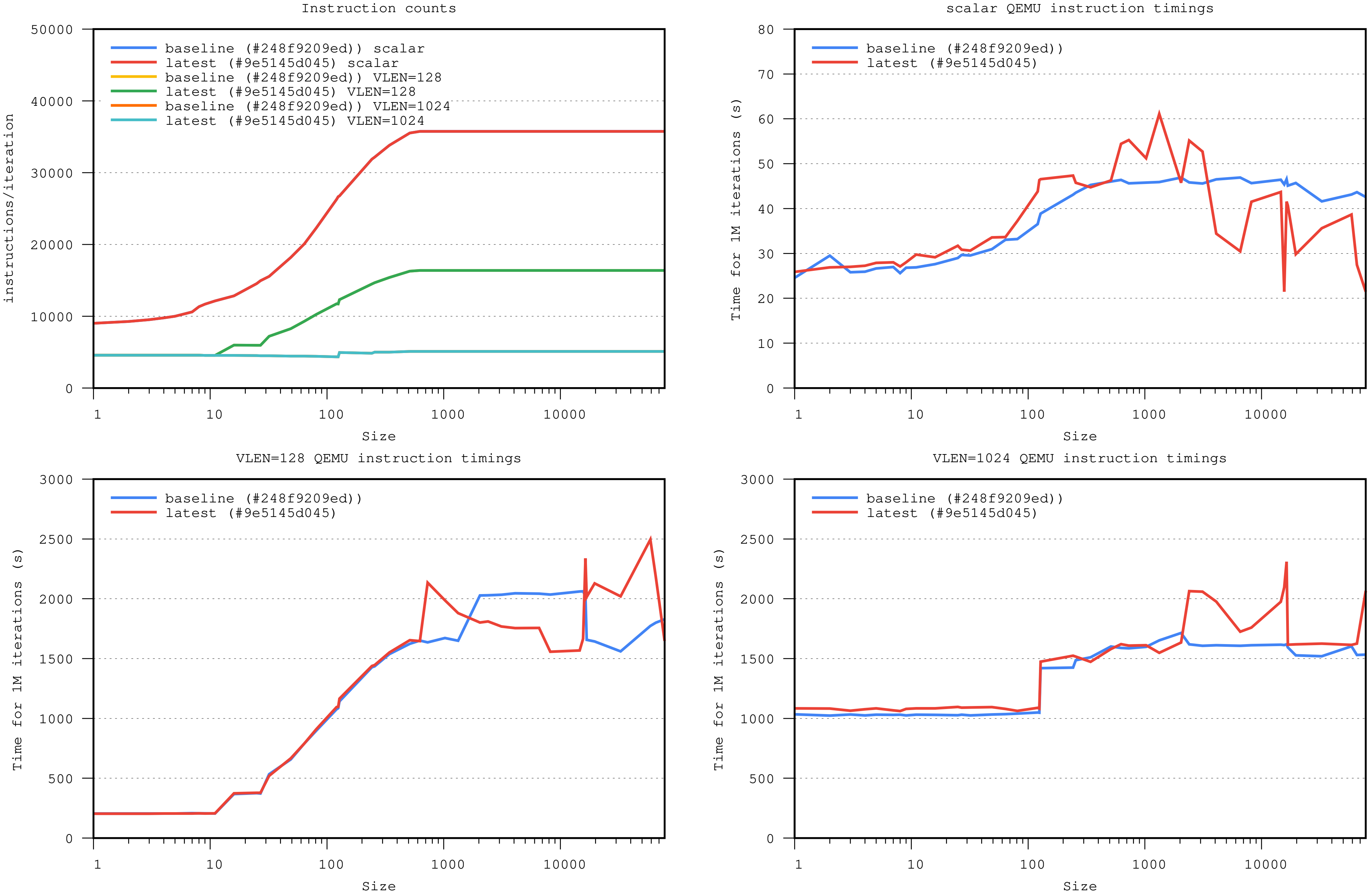
memset performance

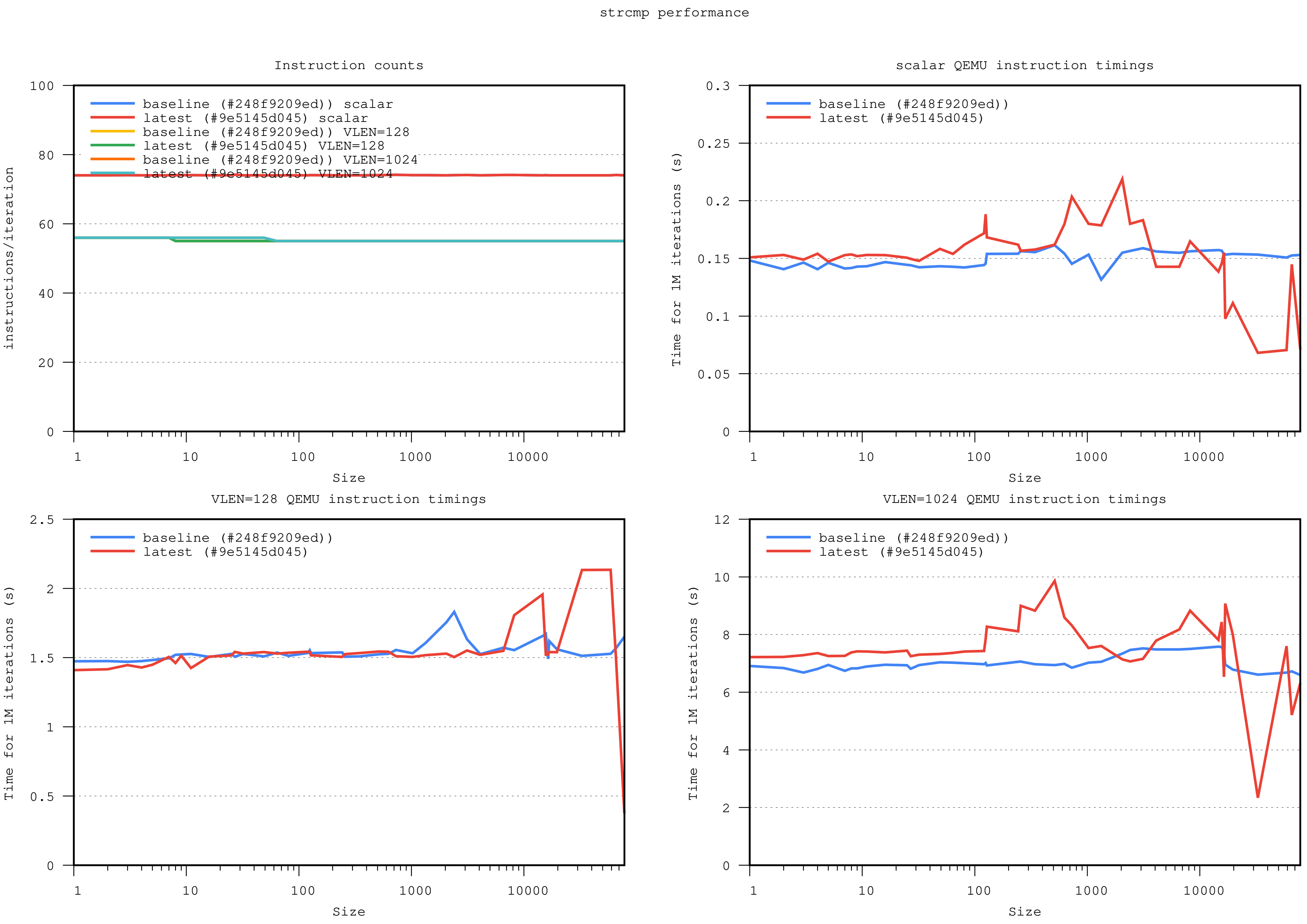


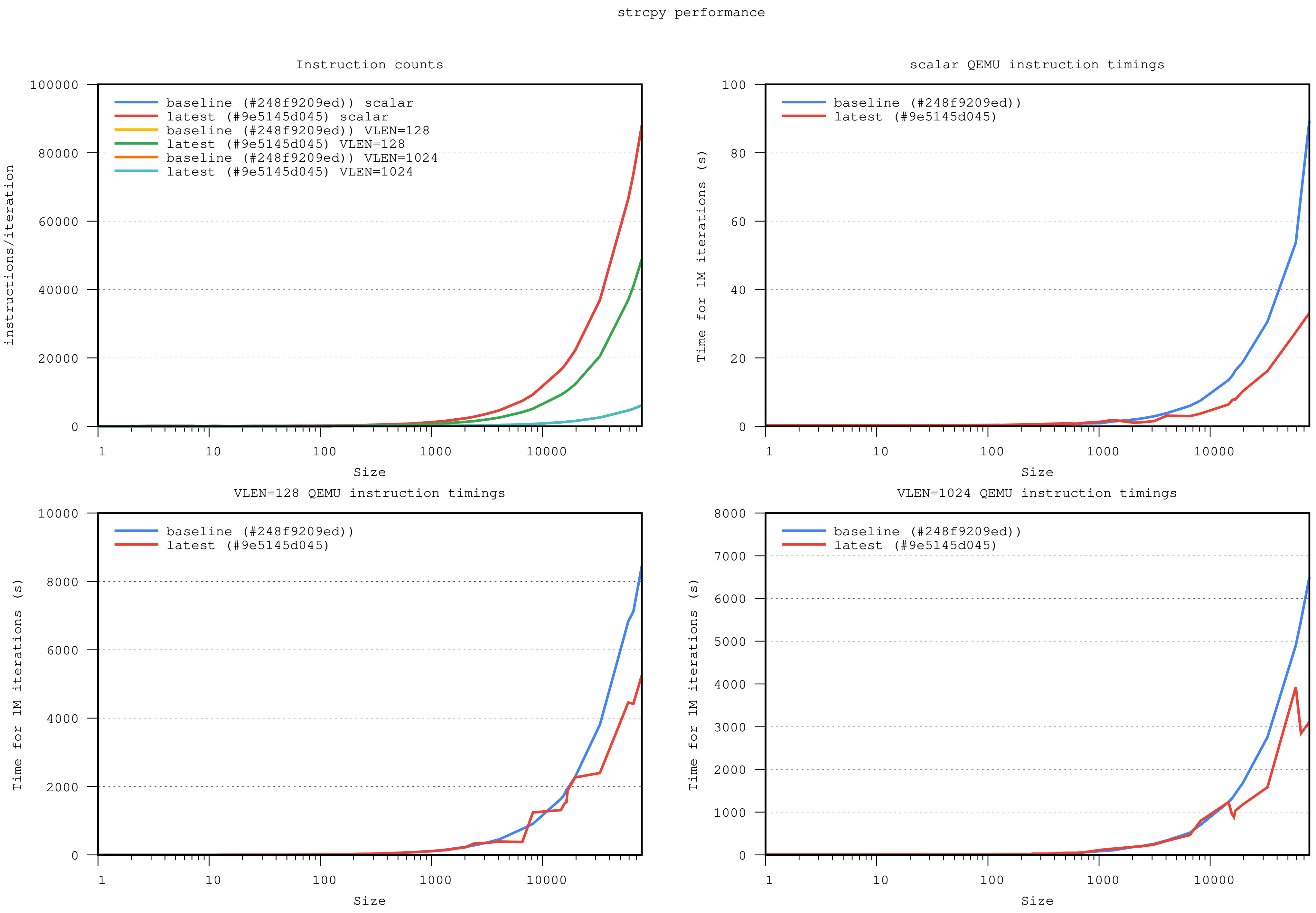
strcat performance



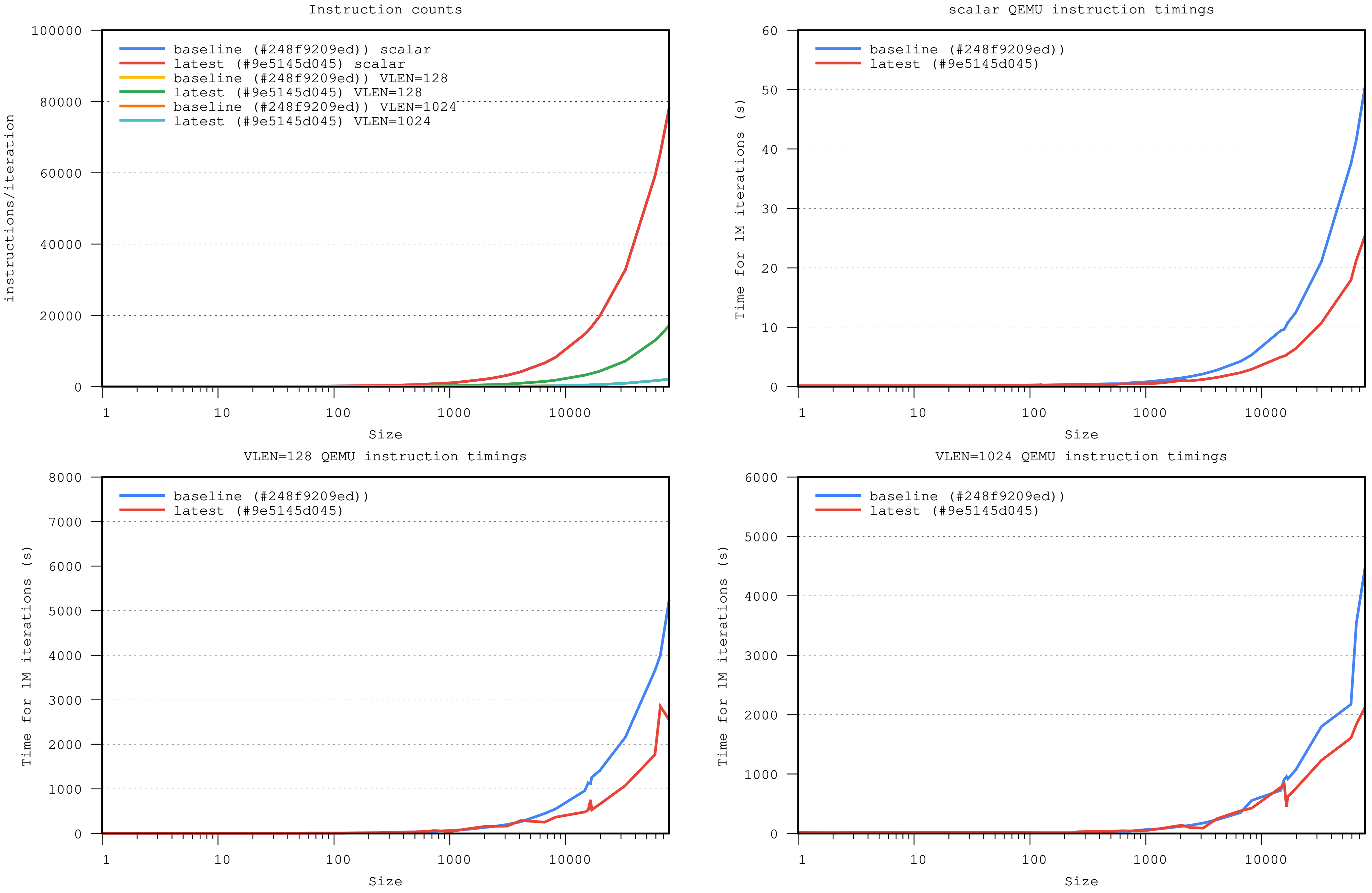
strchr performance




strcmp performance
Instruction counts
baseline (#248f9209ed)) scalar
latest (#9e5145d045) scalar
baseline (#248f9209ed)) VLEN=128
latest (#9e5145d045) VLEN=128
baseline (#248f9209ed)) VLEN=1024
latest (#9e5145d045) VLEN=1024
instructions/iteration
0
20
40
60
80
100
Size
1
10
100
1000
10000
scalar QEMU instruction timings
baseline (#248f9209ed))
latest (#9e5145d045)
Time for 1M iterations (s)
0
0.05
0.1
0.15
0.2
0.25
0.3
Size
1
10
100
1000
10000
VLEN=128 QEMU instruction timings
baseline (#248f9209ed))
latest (#9e5145d045)
Time for 1M iterations (s)
0
0.5
1
1.5
2
2.5
Size
1
10
100
1000
10000
VLEN=1024 QEMU instruction timings
baseline (#248f9209ed))
latest (#9e5145d045)
Time for 1M iterations (s)
0
2
4
6
8
10
12
Size
1
10
100
1000
10000


strcpy performance
Instruction counts
baseline (#248f9209ed)) scalar
latest (#9e5145d045) scalar
baseline (#248f9209ed)) VLEN=128
latest (#9e5145d045) VLEN=128
baseline (#248f9209ed)) VLEN=1024
latest (#9e5145d045) VLEN=1024
instructions/iteration
100000
80000
60000
40000
20000
0
1
10
100
1000
10000
Size
scalar QEMU instruction timings
baseline (#248f9209ed))
latest (#9e5145d045)
Time for 1M iterations (s)
100
80
60
40
20
0
1
10
100
1000
10000
Size
VLEN=128 QEMU instruction timings
baseline (#248f9209ed))
latest (#9e5145d045)
Time for 1M iterations (s)
10000
8000
6000
4000
2000
0
1
10
100
1000
10000
Size
VLEN=1024 QEMU instruction timings
baseline (#248f9209ed))
latest (#9e5145d045)
Time for 1M iterations (s)
8000
7000
6000
5000
4000
3000
2000
1000
0
1
10
100
1000
10000
Size

strlen performance



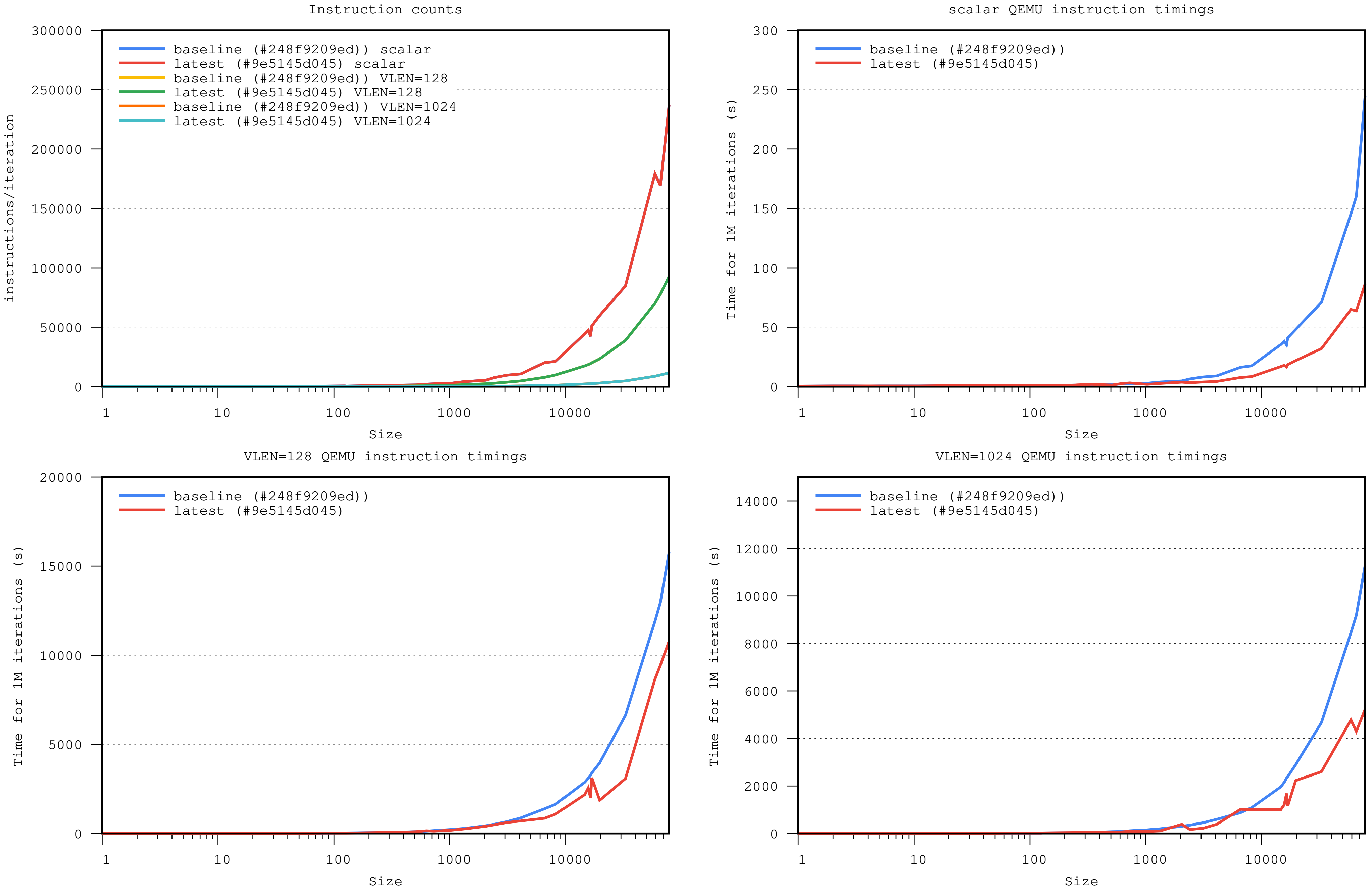
Instruction counts

scalar QEMU instruction timings

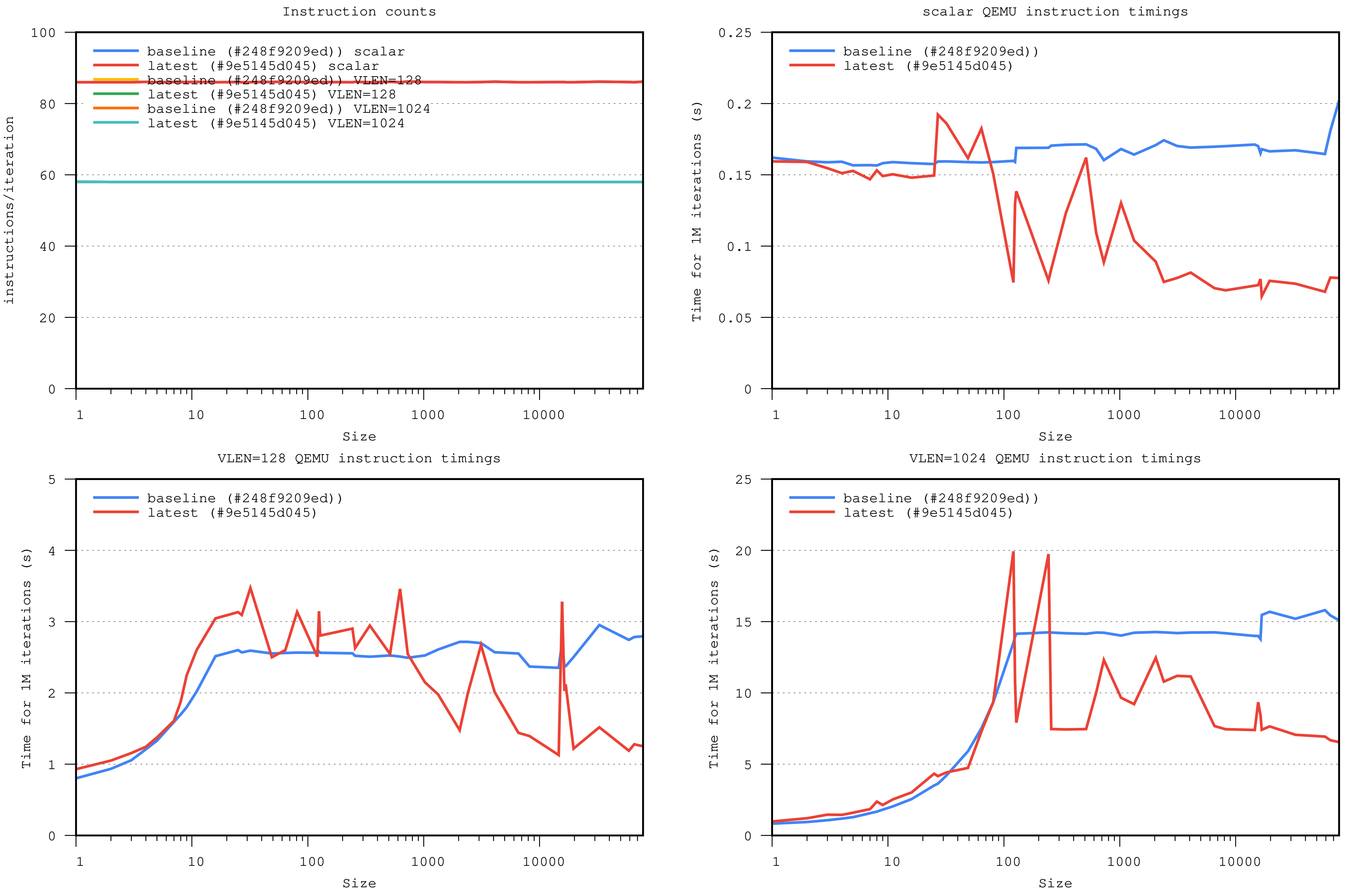
VLEN=128 QEMU instruction timings

VLEN=1024 QEMU instruction timings

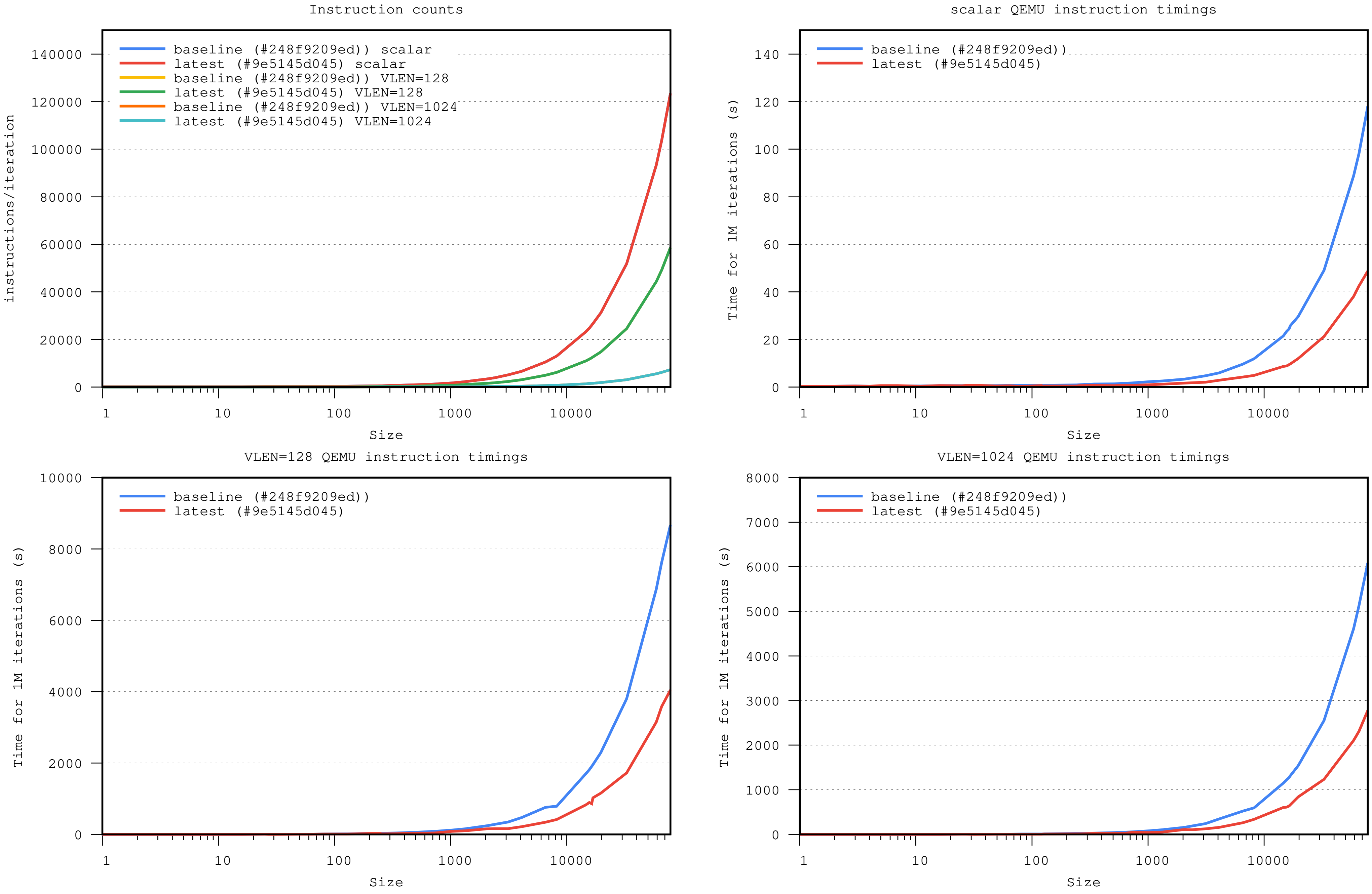
strncat performance



strncmp performance



strncpy performance



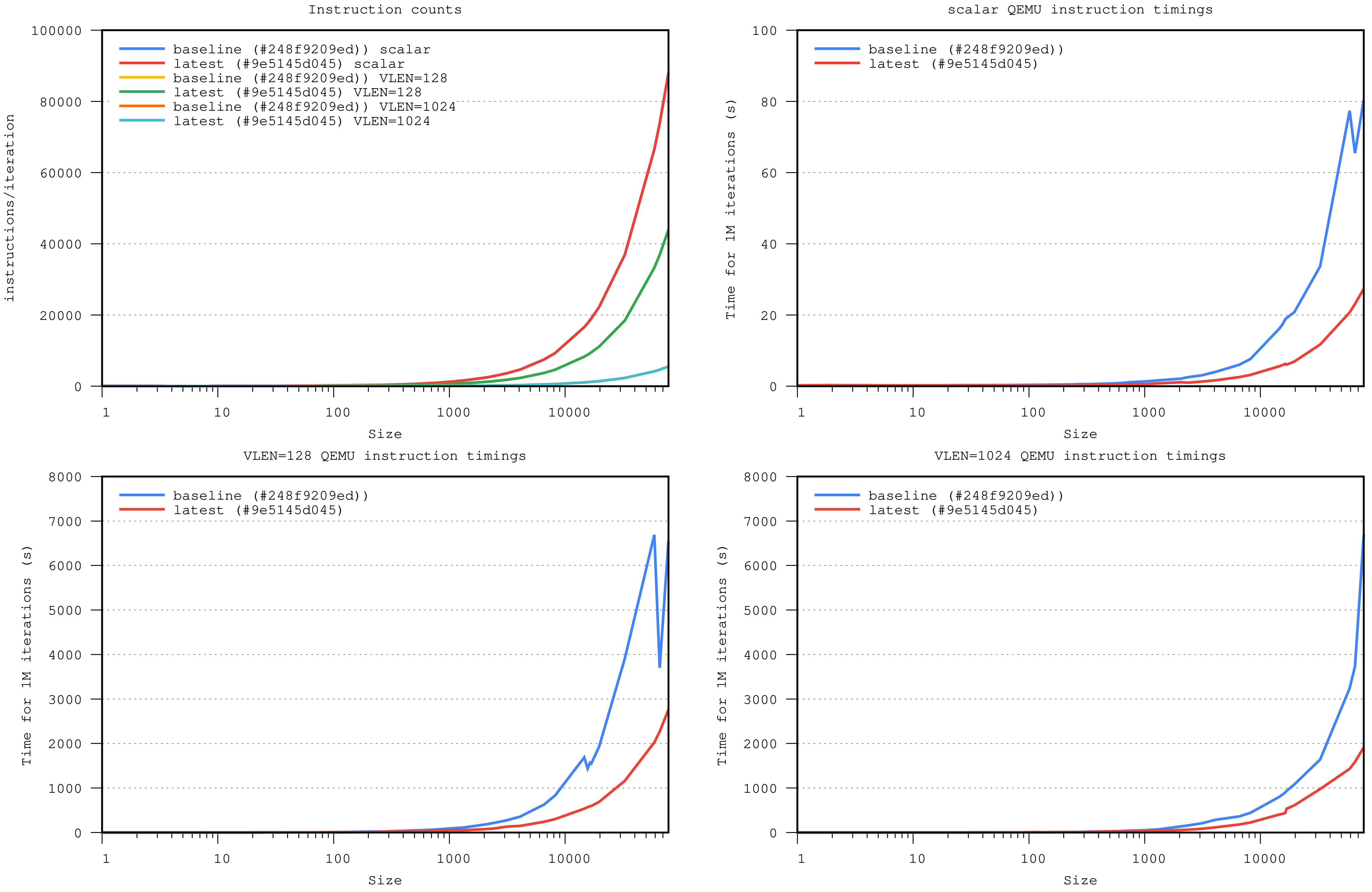
Instruction counts

scalar QEMU instruction timings

VLEN=128 QEMU instruction timings

VLEN=1024 QEMU instruction timings

strnlen performance



Instruction counts

scalar QEMU instruction timings

VLEN=128 QEMU instruction timings

VLEN=1024 QEMU instruction timings