GNU Tools Cauldron 2019

Adding Linux Restartable Sequences (RSEQ) Support in glibc

EfficiOS

mathieu.desnoyers@efficios.com

Content

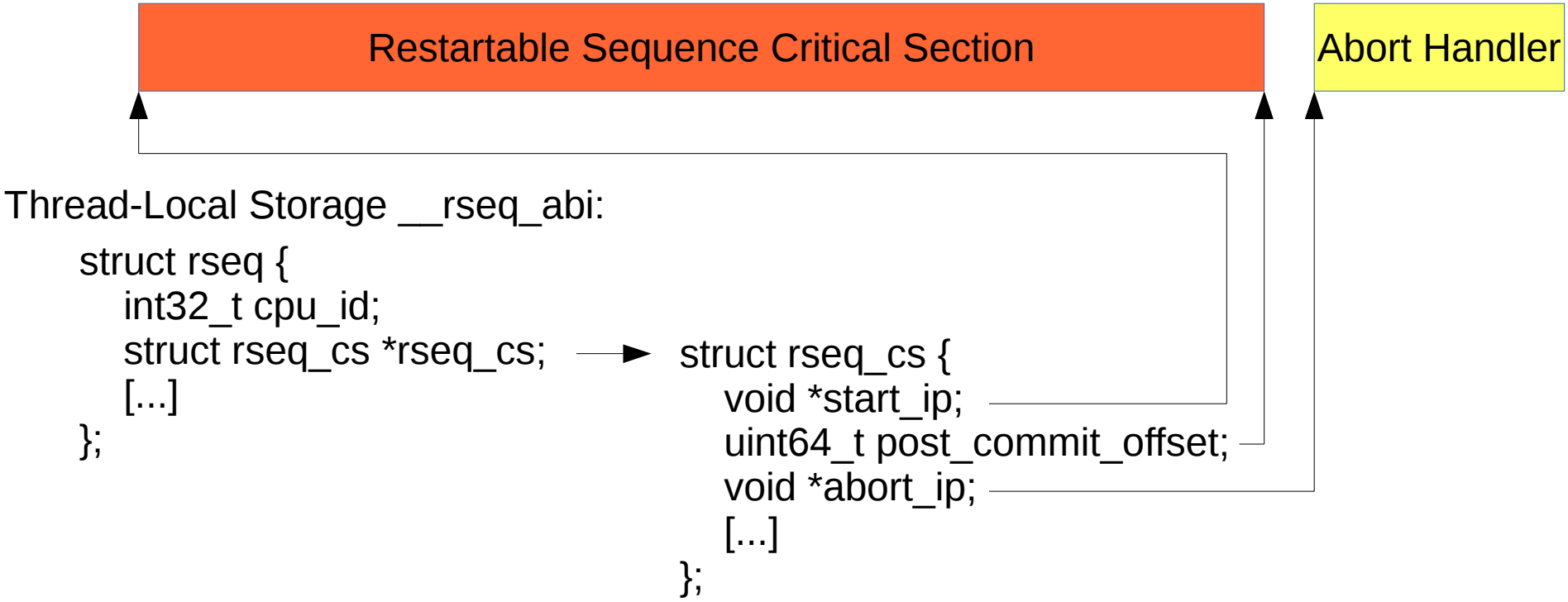
- Restartable Sequences (RSEQ) Introduction
- Use-Cases
- Benchmarks
- Linux Integration
- glibc Integration
- Requirements
- Missing Pieces
- Open Issues
- Ongoing Work

EfficiOS

What are Restartable Sequences (RSEQ) ?

- Linux kernel system call registering a Thread-Local Storage area allowing user-space to perform updates on per-cpu data efficiently,
- Achieve critical section atomicity with respect to scheduler by aborting critical sections on preemption and signal delivery rather than disabling preemption.

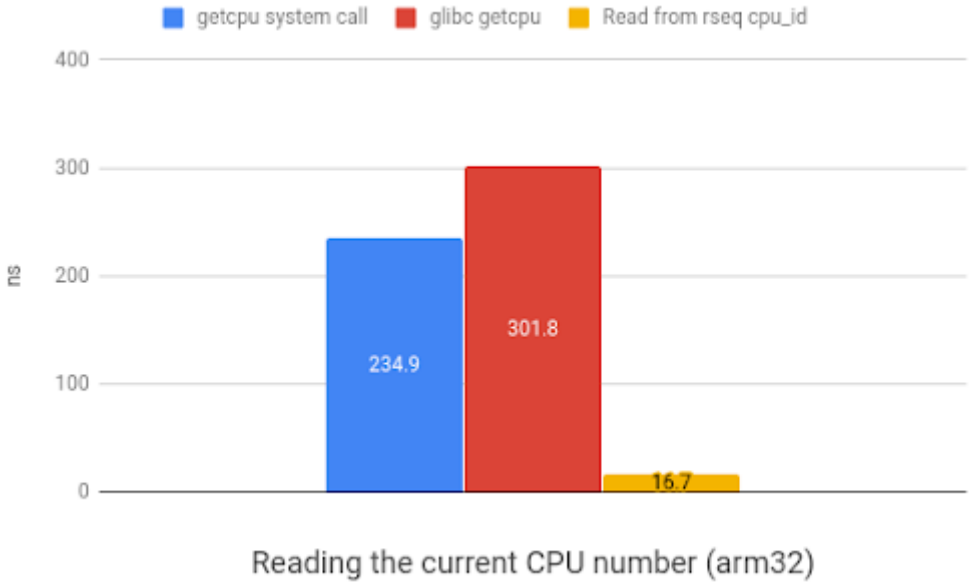
RSEQ Structure Members



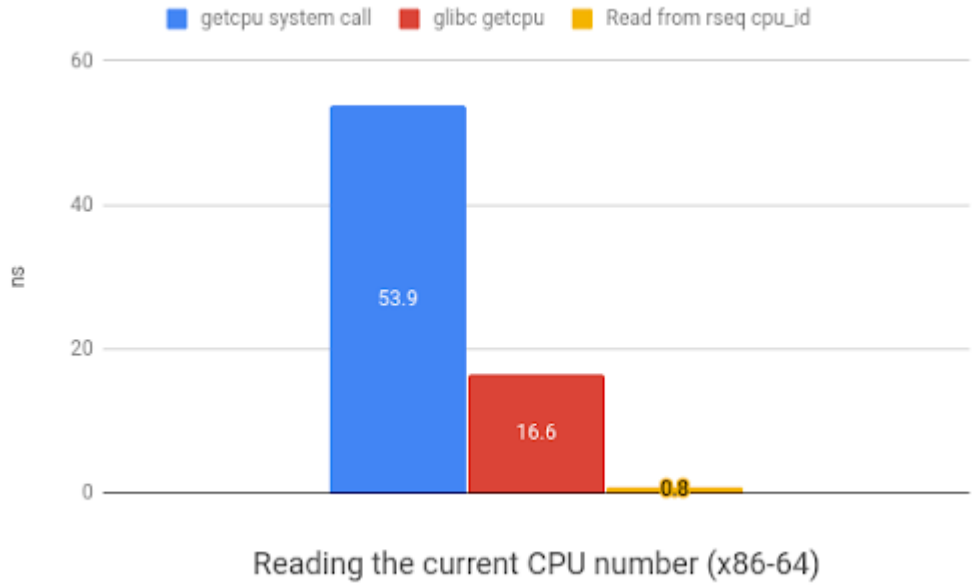
RSEQ Use-Cases

- Per-CPU pool memory allocation,
- Per-CPU ring buffer,
- Per-CPU statistics accounting,
- Per-CPU RCU grace period tracking,
- User-space PMU counters read from user-space on big/LITTLE ARM64,
- Spinlock improvements:
  - Preemption tracking, NUMA awareness.

RSEQ Benchmarks: Get Current CPU Number



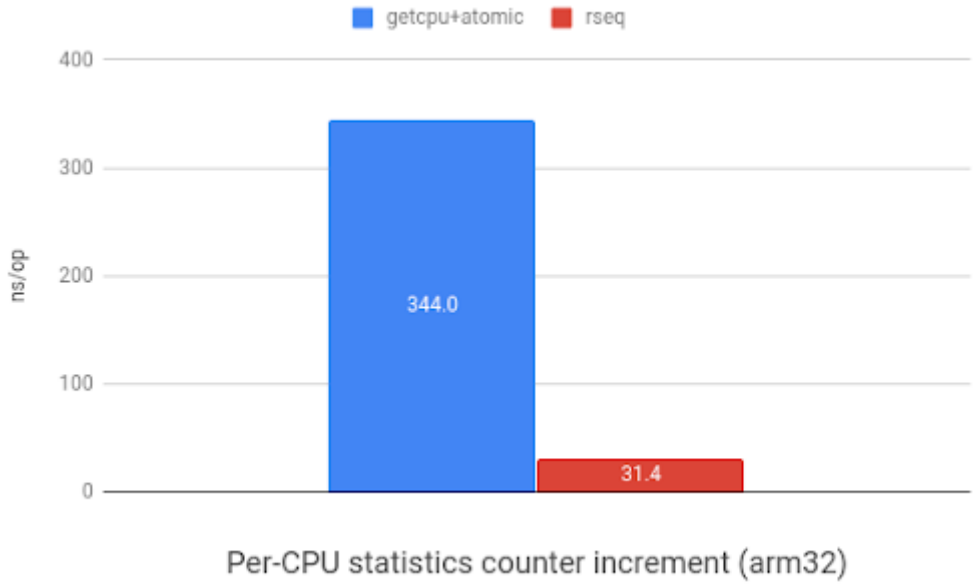
Reading the current CPU number (arm32)



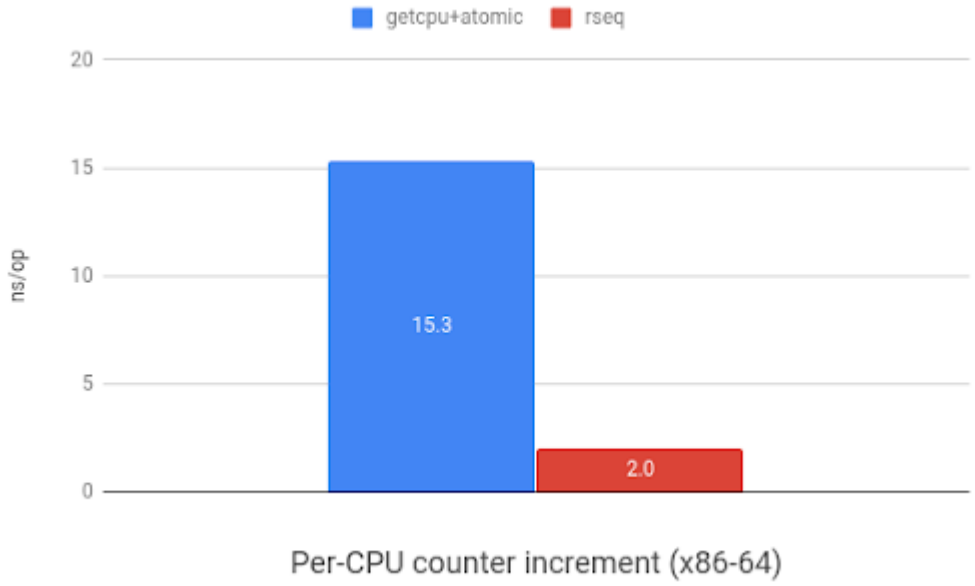
Reading the current CPU number (x86-64)

EfficiOS

RSEQ Benchmarks: Statistics Counter



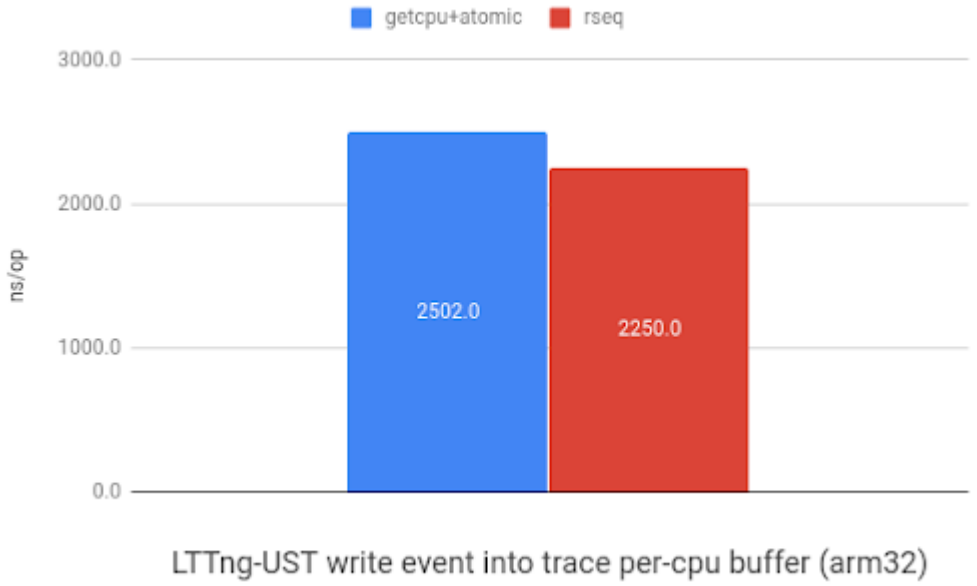
Per-CPU statistics counter increment (arm32)



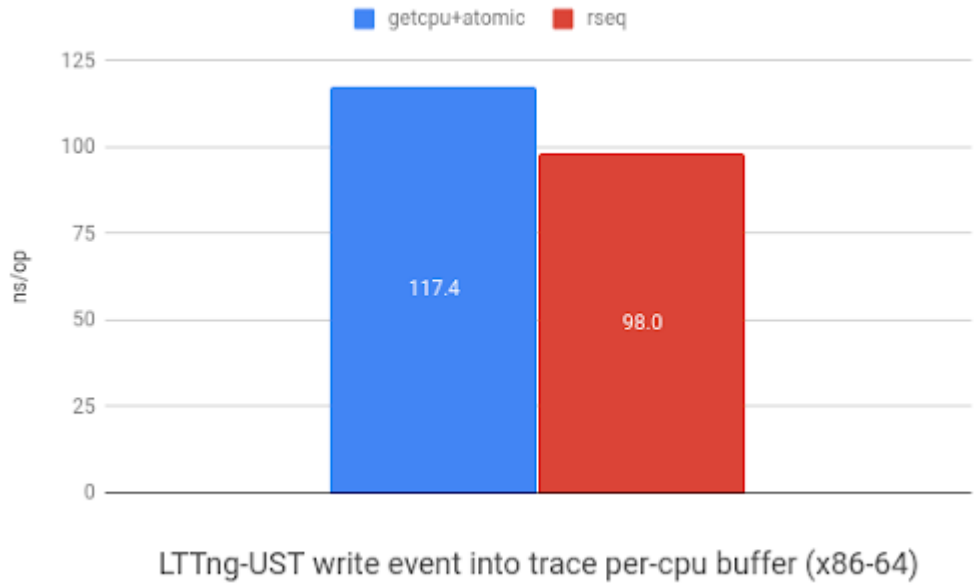
Per-CPU counter increment (x86-64)

EfficiOS

RSEQ Benchmarks: LTTng-UST Ring Buffer



LTTng-UST write event into trace per-cpu buffer (arm32)



LTTng-UST write event into trace per-cpu buffer (x86-64)

EfficiOS

Restartable Sequences Linux Integration

- Linux 4.18:
  - RSEQ system call merged,
  - RSEQ wired up for x86 32/64, powerpc 32/64, arm 32, mips 32/64,
- Linux 4.19:
  - RSEQ wired up for arm 64, s390 32/64,

RSEQ Integration within glibc

- Registration/Unregistration of __rseq_abi TLS within glibc on C startup, and thread start/exit.
- Public header exposing RSEQ signature to users:
  - Uncommon 4-byte signature prior to abort handlers,
  - Security: prevents use of RSEQ as mechanism to redirect execution to arbitrary code,
  - Typically never executed,
  - Ideally traps if reached, valid instruction within objdump.

RSEQ Requirements

- Use in application and libraries,
- Use in signal handler,
  - Nested on top of early/late thread lifetime, when RSEQ is not registered,
- Use in library constructors/destructors,
  - Dynamic linker needs to access TLS early for RSEQ registration before invoking library constructors,

RSEQ Requirements

- Allow internal use within glibc:
  - sched_getcpu(3),
  - Memory allocator,
  - Locking,
- Smooth integration of RSEQ support within the user-space ecosystem:
  - Allow applications/libraries to use RSEQ with older glibc,
  - Without breaking upgrade to glibc supporting RSEQ.

Missing Pieces: GDB Support

- If debugger/emulator single-steps within RSEQ critical section, it is always aborted,
- If abort triggers a retry: no progress.
- Proposed approach:
  - Skip RSEQ critical sections,
  - Similar to handling of LL/SC on various architectures,
- RSEQ headers emit information about all critical sections within __rseq_cs_ptr_array and __rseq_exit_point_array sections.

Missing Pieces: RSEQ glibc integration

- No concensus on __rseq_handled symbol,
  - Aims to allow applications/libraries to use RSEQ with old glibc, with smooth upgrade path.
- Could be removed from patch set if a few problems are solved in glibc.

Open Issues in glibc

- Signals are enabled on thread startup:
  - RSEQ is not registered yet,
  - Disabling signals on thread startup/teardown would be an option.
- TLS cannot be touched by dynamic linker code:
  - Change glibc to allow TLS to be touched by dynamic linker before running library constructors.

Ongoing Work (Linux kernel)

- Allow concurrent update of remote per-CPU data:
  - CPU-hotplug aware.
- Use-cases:
  - LTTng consumer daemon requiring to write into each per-CPU ring buffers periodically (flush timer),
  - Cleanup of free memory reserved for a CPU after it is unplugged.
    - The CPU may be brought online again (concurrently).

Questions ?