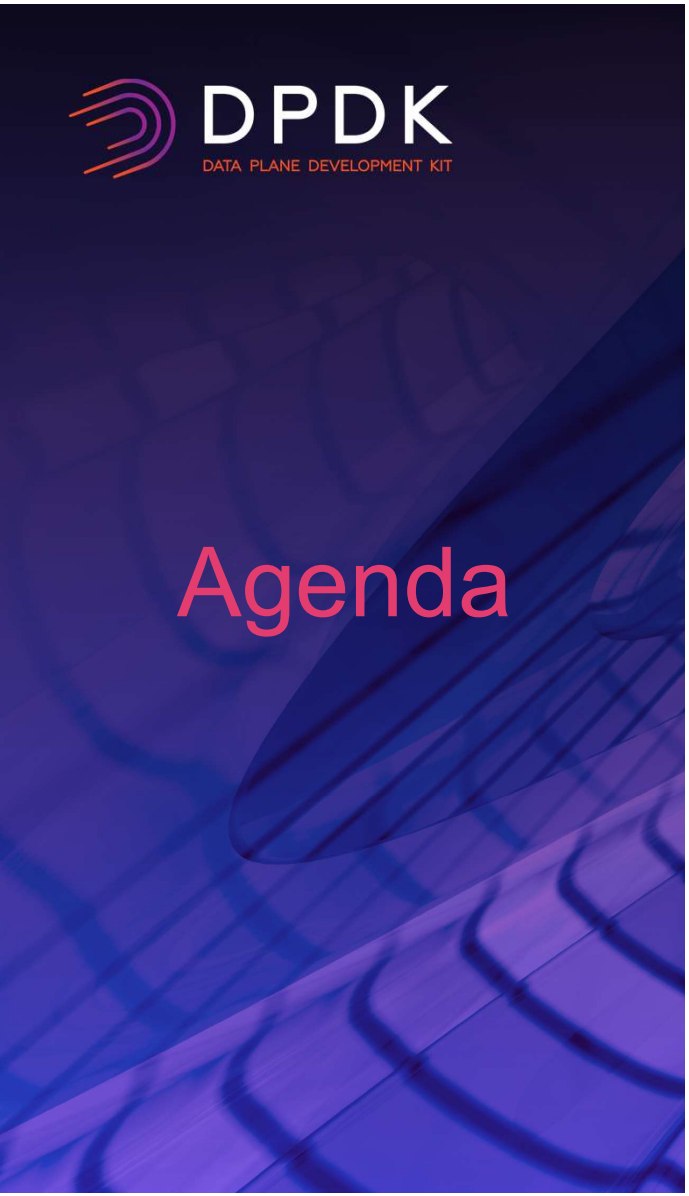
Arm's Effort For DPDK and Future Plan

GAVIN HU
HONNAPPA NAGARAHALLI

ARM



- Relaxed memory ordering
- __sync vs. __atomic
- WFE and SEV for v8.0
- Atomic instructions for V8.1
- Neon Optimization
- Build system and Documentation
- DTS internal CI

C11 memory ordering

- Aarch64 is relaxed memory ordering
- Program order might be broken for performance optimization
  - By HW
  - By compiler
- This memory reordering is transparent to programmers most of time
- Still there are use cases the program order should be kept, especially for multi-core/thread execution environments.
  - That is why memory fences are introduced
- Memory fences degrade performance
  - Use less restrictive memory ordering model as possible to mitigate the degradation

Relaxed memory ordering

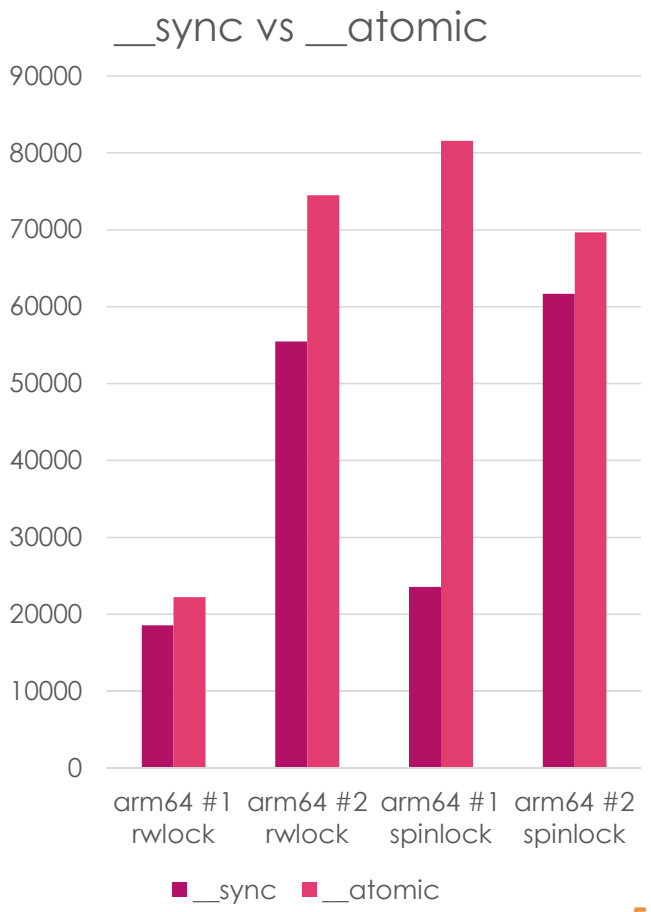
- Work done So far
  - KNI
  - Ring
  - Hash
- Future Work
  - Vhost
  - Virtio
  - ring

- Remove memory fences in the wrong places
- Lessen the barriers to make them as weak as possible
- Ensure correctness(multi-core safe) by synchronizing to the correct points.

__sync vs __atomic

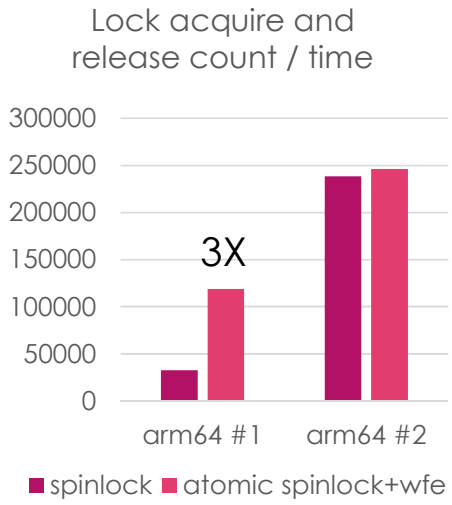
- Benefits of __atomic
  - Full memory barrier →one-way barrier.
  - Benchmarking shows constant improvements.

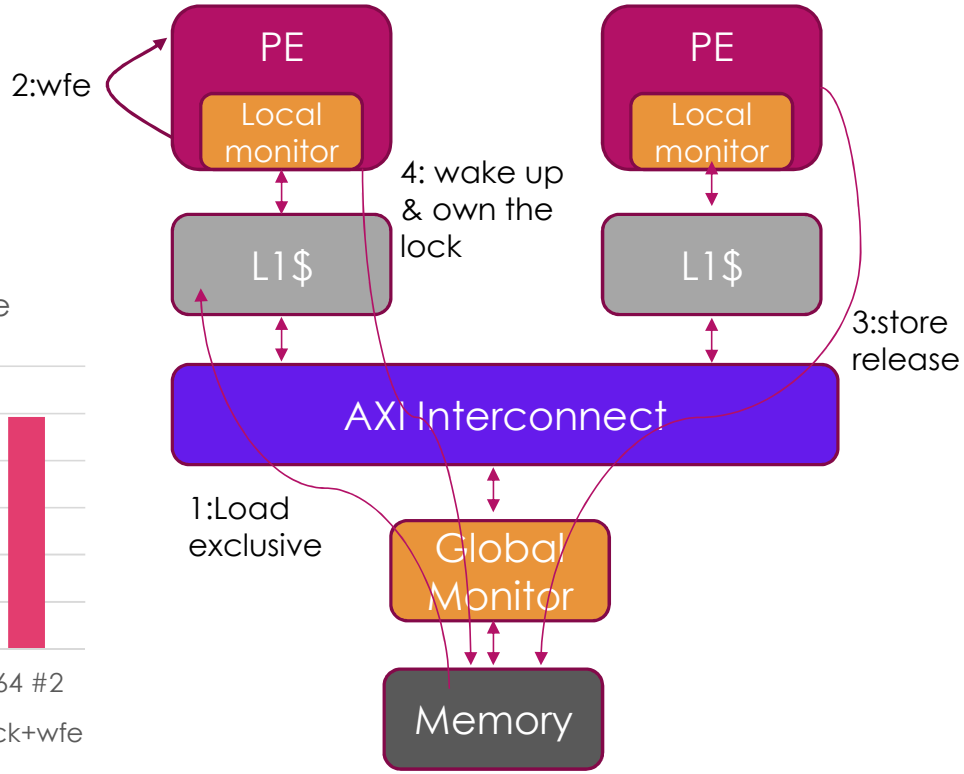
```
(gdb) disassemble /s rte_spinlock_lock
Dump of assembler code for function rte_spinlock_lock:
   0x0000000000468434 <+0>:     mov     w1, #0x1                        // #1
   0x0000000000468438 <+4>:     ldxr    w2, [x0]
   0x000000000046843c <+8>:     stxr    w3, w1, [x0]
   0x0000000000468440 <+12>:    cbnz    w3, 0x468438 <rte_spinlock_lock+4>
   0x0000000000468444 <+16>:    dmb     ish
   0x0000000000468448 <+20>:    cbz     w2, 0x46845c <rte_spinlock_lock+40>
   0x000000000046844c <+24>:    ldr     w2, [x0]
   0x0000000000468450 <+28>:    cbz     w2, 0x468438 <rte_spinlock_lock+4>
   0x0000000000468454 <+32>:    yield
   0x0000000000468458 <+36>:    b       0x46844c <rte_spinlock_lock+24>
   0x000000000046845c <+40>:    ret
End of assembler dump.
```



WFE and SEV for V8.0

- WFE
  - Suspend execution when the lock is held
  - Get wake up events if the exclusive monitor is cleared by other PEs
- SEV
  - Send event explicitly to break WFE
- Use cases
  - Spinlock
  - Rw lock
  - Ring
  - Other acquire/release semantics environments





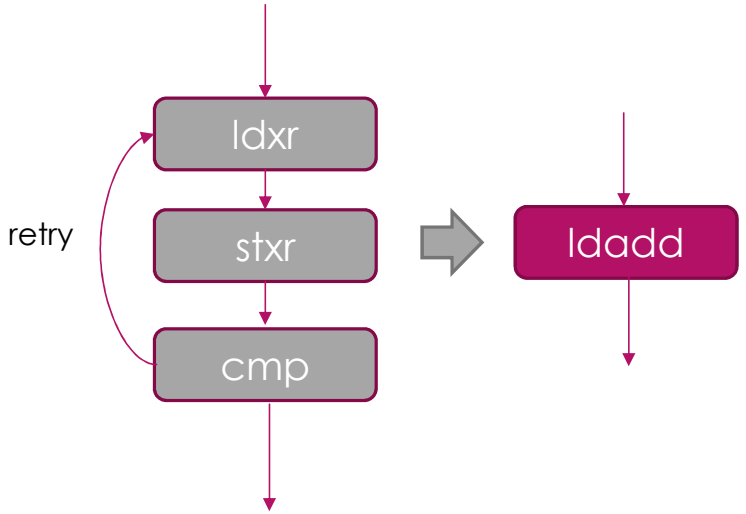
Rte_atomicN_xxx for V8.1

DPDK atomic implementation

- rte_atomicN_xxx APIs are implemented using __sync built-ins.
- These translate to retry loops with load/store exclusive instructions.

Proposed patches

- V8.1 ISA adds atomic instructions in atomic memory operations class.
- rte_atomicN_xxx APIs will be changed to use these new instructions.



Neon optimizations

Library(Planned)

- Vhost lib
- rte_ethdev lib
- rte_hash

Examples

- L3fwd ready
- More to come

PMD

- Intel NIC PMD ready by Arm
- Mellonax PMD ready by MLX
- More to come

Build system and Documentation

Make

- Gcc / clang
- Native / Cross
- Configuration settings

Meson/Ninja

- Native
- Clang
- Configurations

Documentation

- User guides
- Programming guide

DTS and Internal CI

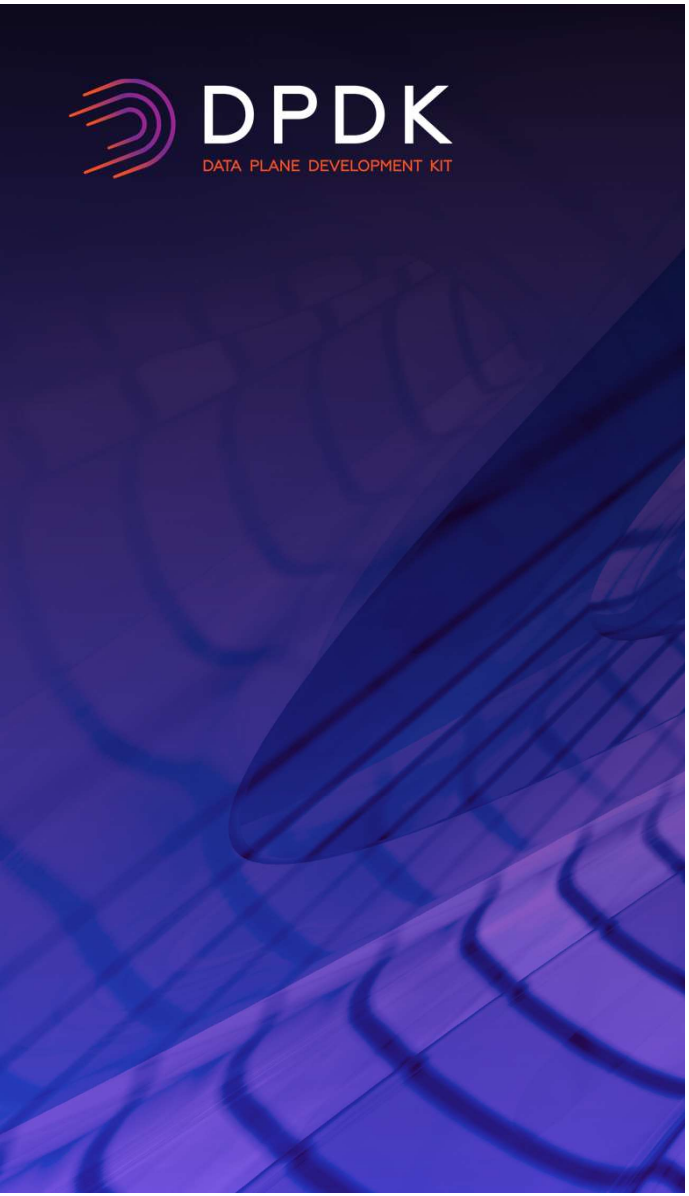
- Work done so far
  - Identify DTS test cases suitable for Arm platform
  - Enable DTS test cases on arm64
  - Integrated into internal CI
- Future work
  - Add More cases
  - Integrated into community lab

Key Takeaways

- Relaxed memory model and examples of optimization
- More performant, scalable rwlock, spinlock, rte ring, atomic APIs
- Neon optimization examples


DPDK
DATA PLANE DEVELOPMENT KIT
Arm’s efforts to DPDK and Future Plan
Gavin Hu
gavin.hu@arm.com
Honnappa Nagarahalli
Honnappa.Nagarahalli@arm.com

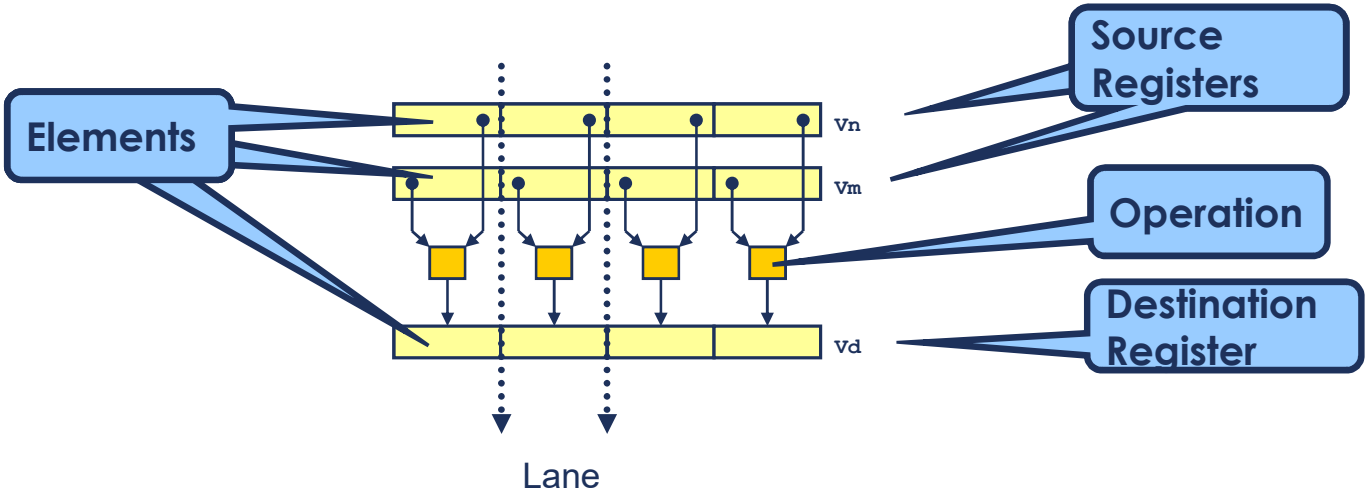
Thanks!



- BackUp

NEON for AArch64

- An extension to the AArch64 instruction set derived from the AArch32 Advanced SIMD syntax
- Same data processing principles
  - Registers are considered as **vectors** of **elements** of the same data type
  - Data types available: signed and unsigned 8-bit, 16-bit, 32-bit, 64-bit, single and double precision floating point
  - Instructions usually perform the same operation in all **lanes**



KNI optimization

Kernel Module

- Fix the synchronization issues using kernel SMP barriers

User Space

- Fix the synchronization issues using C11 memory model __atomic builtins

Spin lock

- __atomic instead of __sync builtin
  - Full memory barrier →one-way barrier
- SEV and WFE
  - Tight loop → suspend execution
  - Less stress to memory subsystem
  - Less power

Rw lock

- __atomic instead of __sync builtin
  - Full memory barrier →one-way barrier
- SEV and WFE
  - Tight loop → suspend execution
  - Less stress to memory subsystem
  - Less power

Vhost/virtio optimization Plan

- Relaxed memory ordering
- Spinlock and rwlock
- Prefetch
- Neon optimization

ring

Fixes

- Synchronization to tail update
- Keep deterministic order allowing the CAS retry to work

More to come…

- Set Weak = true to allow spurious failure
- Replace rte_pause with “WFE” instruction, when updating tails.

Optimization

- Relaxed ordering for load and store of head
- Remove duplicate atomic loads

```
65      while (!__atomic_compare_exchange_n(&sl->locked, &exp, 1, 0,
__ATOMIC_ACQUIRE, __ATOMIC_RELAXED));
   0x000000000091694c <+36>:    1f 20 03 d5 nop
   0x0000000000916950 <+40>:    e1 0f 40 f9 ldr x1, [sp, #24]
   0x0000000000916954 <+44>:    e0 93 00 91 add x0, sp, #0x24
   0x0000000000916958 <+48>:    03 00 40 b9 ldr w3, [x0]
   0x000000000091695c <+52>:    24 00 80 52 mov w4, #0x1
// #1
   0x0000000000916960 <+56>:    22 fc 5f 88 ldaxr   w2, [x1]
   0x0000000000916964 <+60>:    5f 00 03 6b cmp w2, w3
   0x0000000000916968 <+64>:    61 00 00 54 b.ne    0x916974
<rte_spinlock_lock+76>  // b.any
   0x000000000091696c <+68>:    24 7c 05 88 stxr    w5, w4, [x1]
   0x0000000000916970 <+72>:    85 ff ff 35 cbnz    w5, 0x916960
<rte_spinlock_lock+56>
   0x0000000000916974 <+76>:    e1 17 9f 1a cset    w1, eq  // eq = none
   0x0000000000916978 <+80>:    3f 00 00 71 cmp w1, #0x0
   0x000000000091697c <+84>:    41 00 00 54 b.ne    0x916984
<rte_spinlock_lock+92>  // b.any
   0x0000000000916980 <+88>:    02 00 00 b9 str w2, [x0]
   0x0000000000916984 <+92>:    e0 03 01 2a mov w0, w1
   0x0000000000916988 <+96>:    00 00 00 52 eor w0, w0, #0x1
   0x000000000091698c <+100>:   00 1c 00 12 and w0, w0, #0xff
   0x0000000000916990 <+104>:   1f 00 00 71 cmp w0, #0x0
   0x0000000000916994 <+108>:   e1 fd ff 54 b.ne    0x916950
<rte_spinlock_lock+40>  // b.any
```