DPDK

DATA PLANE DEVELOPMENT KIT

DPDK Virtio Performance Analysis and Tuning on Armv8

JOYCE KONG/GAVIN HU

ARM

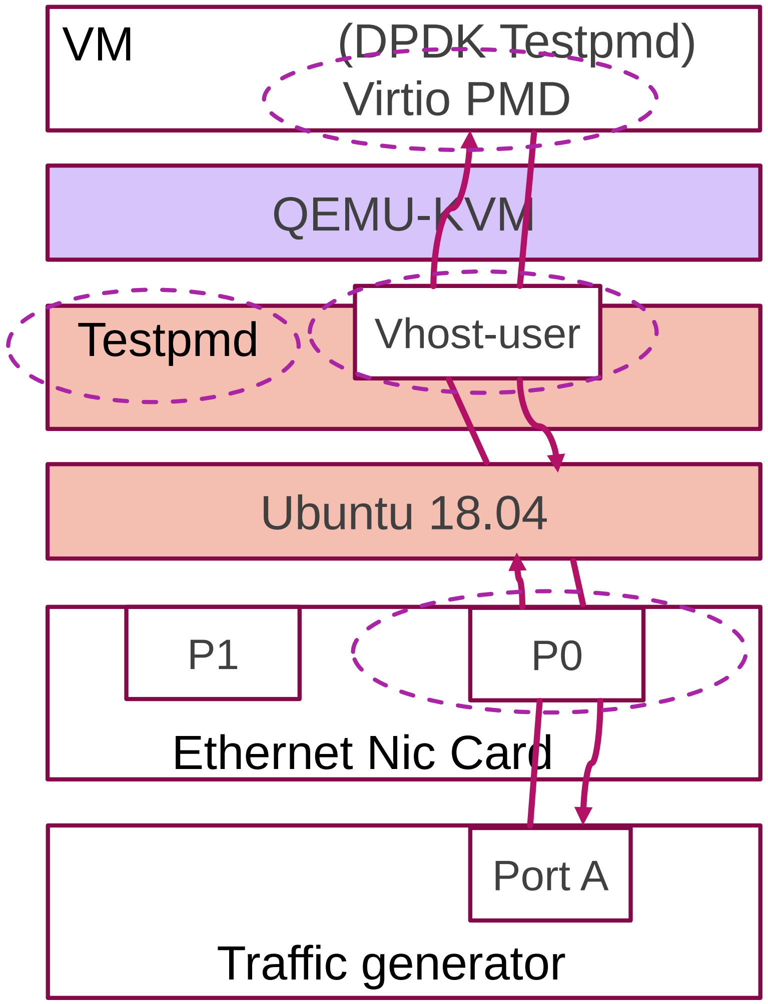
DPDK
DATA PLANE DEVELOPMENT KIT

Agenda

- Test Scenario
- Testbed and Baseline Performance
  - NUMA balancing
  - VHE
- Analysis and Tuning
  - Weak-Memory Model
  - Loop unrolling
  - Prefetch
  - Inline function
- Performance Data Result

Test Scenario

- Traffic generator: IXIA
- Physic Port: 40Gbps NIC
- Packet Flow: IXIA Port A -> NIC Port 0 -> Vhost-user -> Virtio -> Vhost-user -> NIC port 0 -> IXIA Port A
- Test case: RFC2544 zero packet loss test



DPDK PVP test setup

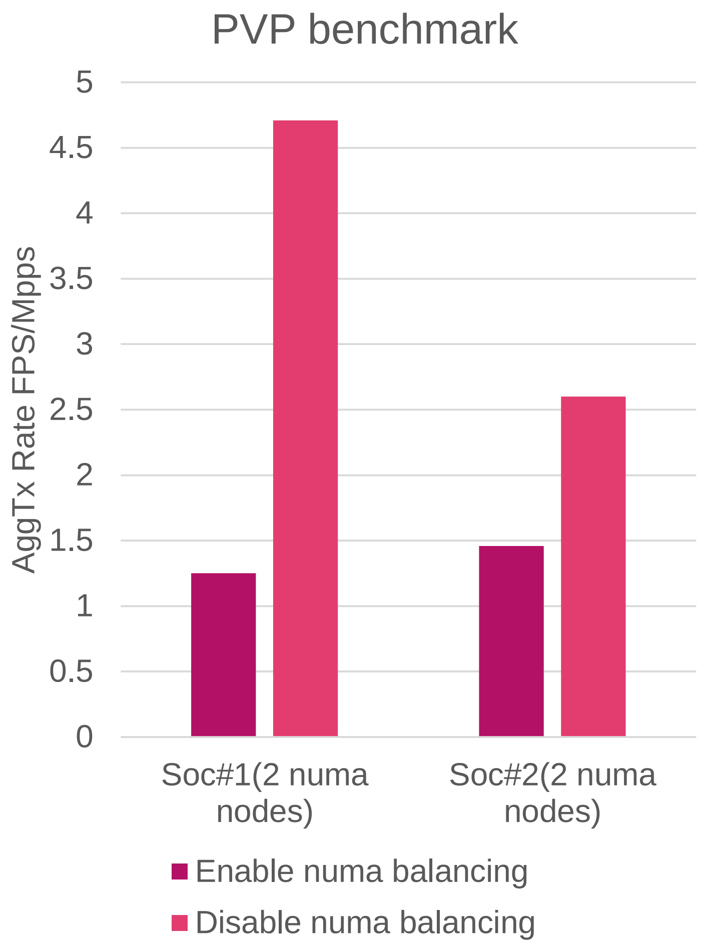
DPDK
DATA PLANE DEVELOPMENT KIT

Agenda

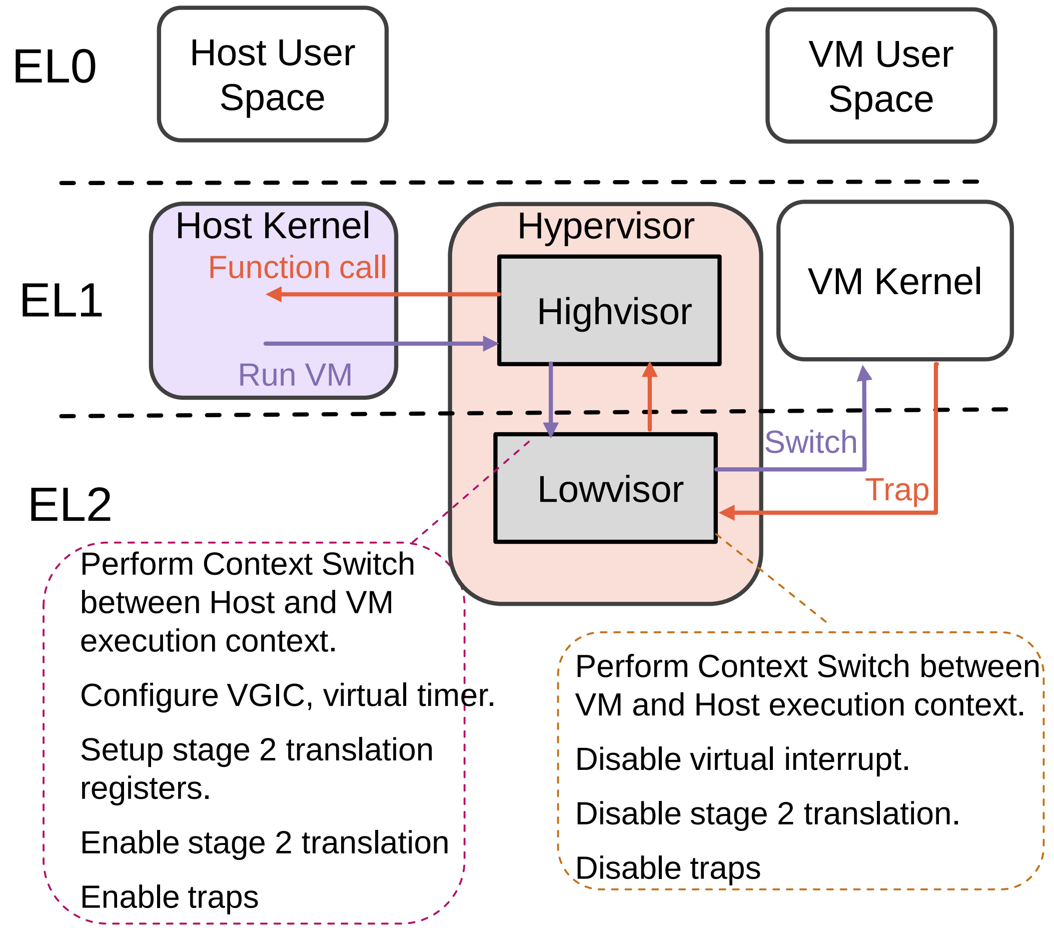
- Test Scenario
- Testbed and Baseline Performance
  - NUMA balancing
  - VHE
- Analysis and Tuning
  - Weak-Memory Model
  - Loop unrolling
  - Prefetch
  - Inline function
- Performance Data Result

NUMA Balancing

- NUMA balancing
  - Move tasks(threads/ processes) closer to the memory they are accessing
  - Move application data to memory closer to the tasks that reference it
- Automatic NUMA balancing internals
  - Periodic NUMA unmapping of process memory
  - NUMA hinting page fault
  - Migrate-on-Fault(MoF): move memory to where the program using it runs
  - Task_numa_placement: move running programs closer to their memory
- Unmapping of memory, NUMA faults, migration and NUMA placement incur overhead
- Configuration
  - # numactl –hardware shows multiple nodes
  - # echo 0 > /proc/sys/kernel/numa_balancing



VHE (Virtualization Host Extensions)



Perform Context Switch between Host and VM execution context.

Configure VGIC, virtual timer.

Setup stage 2 translation registers.

Enable stage 2 translation

Enable traps

Perform Context Switch between VM and Host execution context.

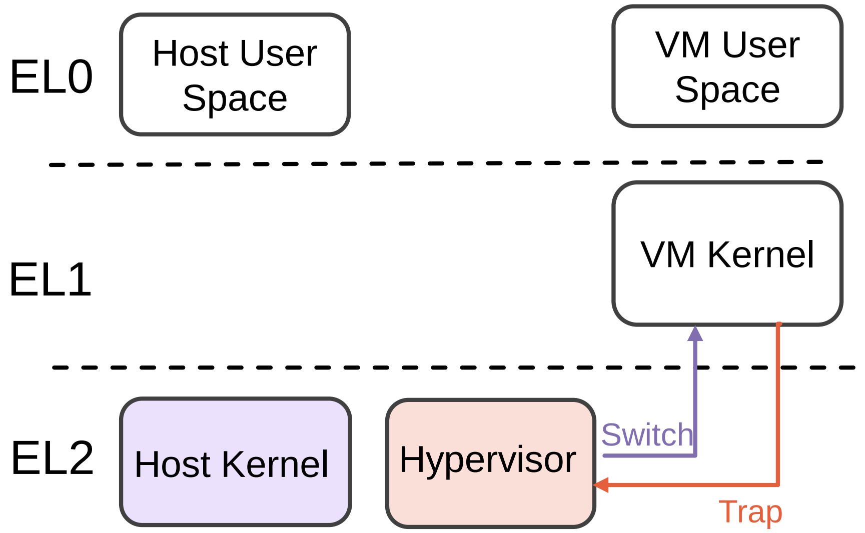
Disable virtual interrupt.

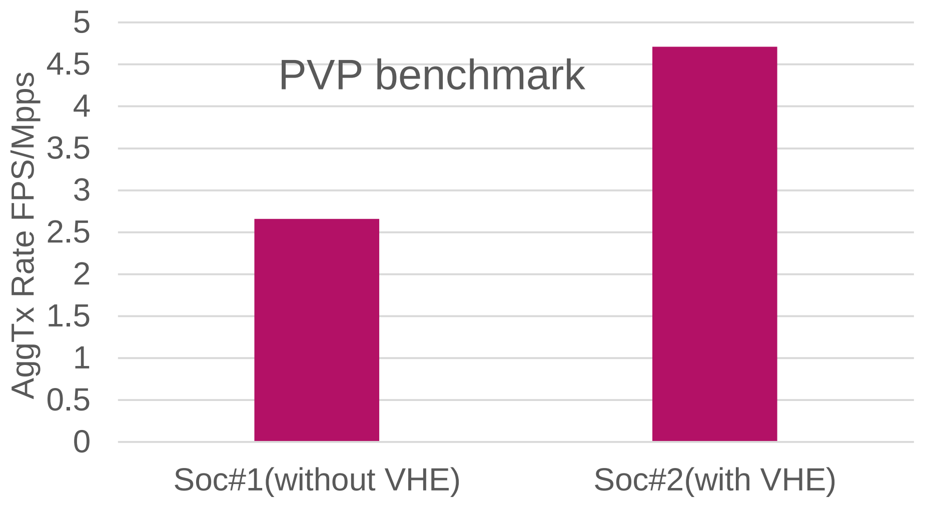
Disable stage 2 translation.

Disable traps

KVM/ARM without VHE

KVM/ARM with VHE





Agenda

- Test Scenario
- Testbed and Baseline Performance
    - NUMA balancing
    - VHE
- Analysis and Tuning
    - Weak-Memory Model
    - Loop unrolling
    - Prefetch
    - Inline function
- Performance Data Result

Weak-Memory Model

Strong-Memory Order	Weak-Memory Order
All reads and writes are in-order	Reads and Writes are arbitrarily re-ordered, subject only to data dependencies and explicit memory barrier instructions

- Hardware re-ordering improves performance
  - Multiple issue of instructions
  - Out-of-order execution
  - Speculation
  - Speculative loads
  - Load and store combine
  - External memory systems
  - Cache coherent multi-core processing
  - Optimizing compilers

- Certain situations require stronger ordering rules – barriers by software
  - Prevent unsafe optimizations from occurring
  - Enforce a specific memory ordering

- Memory barriers degrade performance
  - Whether a barrier is necessary in a specific situation
  - Which is the correct barrier to use

Full Barriers

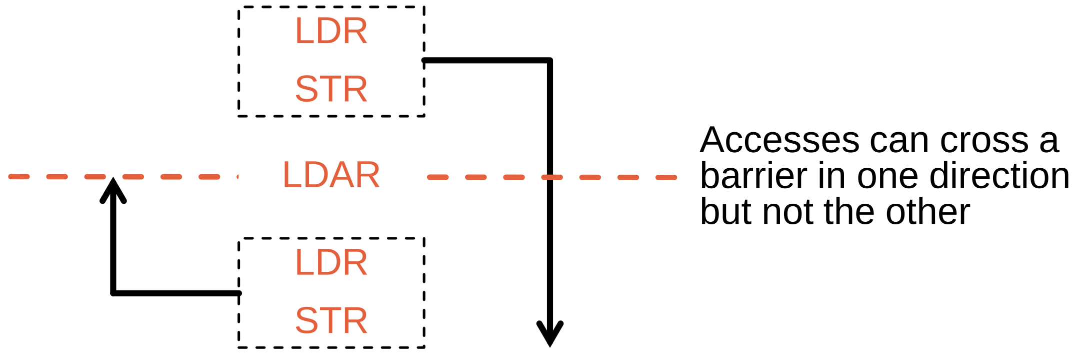
- The ARM architecture includes barrier instructions forcing access order and access completion at a specific point
- `ISB` – Instruction Synchronization Barrier
  - Flush the pipeline, and refetch the instructions from the cache (or memory)
  - Effects of any completed context-changing operation before the ISB are visible to instructions after the ISB
  - Context-changing operations after the ISB only take effect after the ISB has been executed
- `DMB` – Data Memory Barrier
  - Prevent re-ordering of data access instructions across the barrier instruction
  - Data accesses (loads/stores) before the DMB are visible before any data access after the DMB
  - Data/unified cache maintenance operations before the DMB are visible to explicit data access after the DMB
- `DSB` – Data Synchronization Barrier
  - More restrictive than a DMB, any further instructions (not just loads/stores) can be observed until the DSB is completed

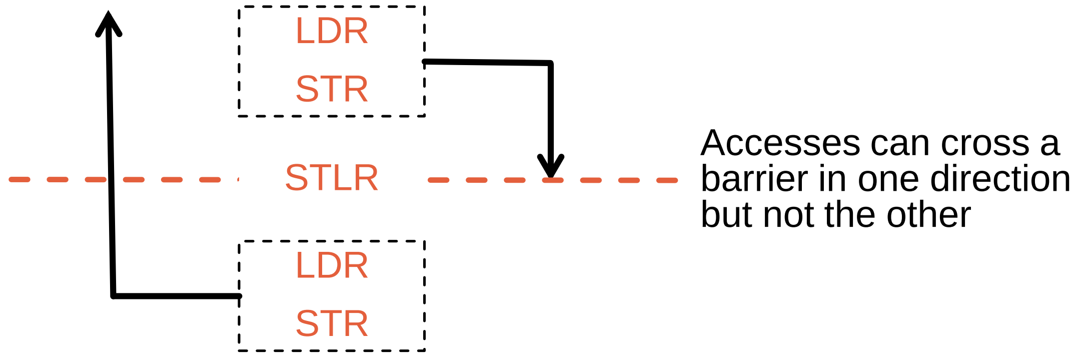
“One-way” barrier optimization

- Aarch64 adds new load/store instructions with implicit barrier semantics

- Load-Acquire (LDAR)
  - All accesses after the LDAR are observed after the LDAR
  - Accesses before the LDAR are not affected

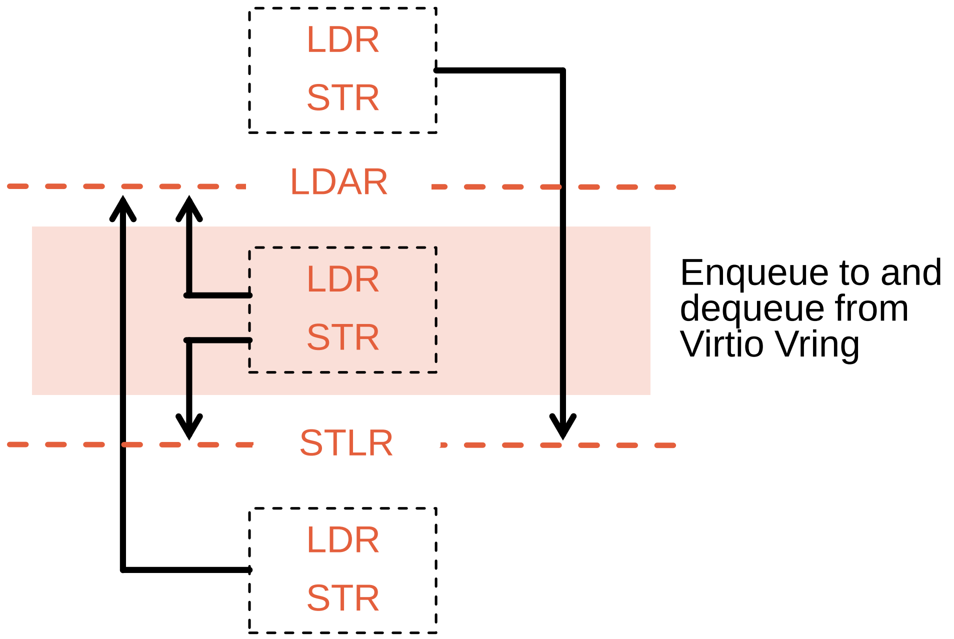
- Store-Release (STLR)
  - All accesses before the STLR are observed before the STLR
  - Accesses after the STLR are not affected


LDR
STR
LDAR
LDR
STR
Accesses can cross a barrier in one direction but not the other

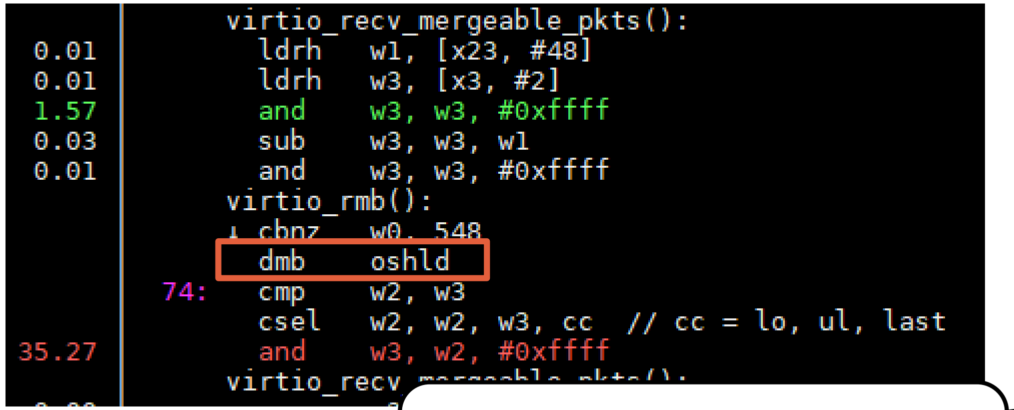

LDR
STR
STLR
LDR
STR
Accesses can cross a barrier in one direction but not the other

“One-way” barrier optimization

- LDAR and STLR may be used as a pair
  - To protect a critical section of code
  - May have lower performance impact than a full DMB
  - No ordering is enforced within the critical section
- Scope
  - DMB/DSB take a qualifier to control which shareability domains see the effect
  - LDAR/STLR use the attribute of the address accessed

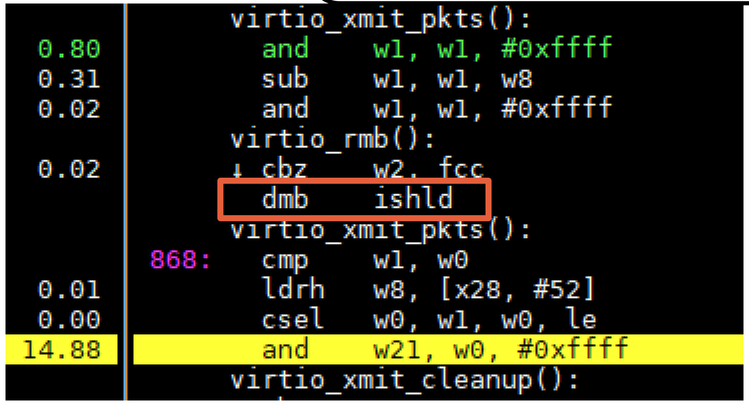


“One-way” barrier optimization



```
       virtio_recv_mergeable_pkts():
0.01     ldrh   w1, [x23, #48]
0.01     ldrh   w3, [x3, #2]
1.57     and    w3, w3, #0xffff
0.03     sub    w3, w3, w1
0.01     and    w3, w3, #0xffff
       virtio_rmb():
       ↓ cbnz   w0, 548
         dmb    oshld
   74:   cmp    w2, w3
         csel   w2, w2, w3, cc   // cc = lo, ul, last
35.27    and    w3, w2, #0xffff
       virtio_recv_mergeable_pkts():
```

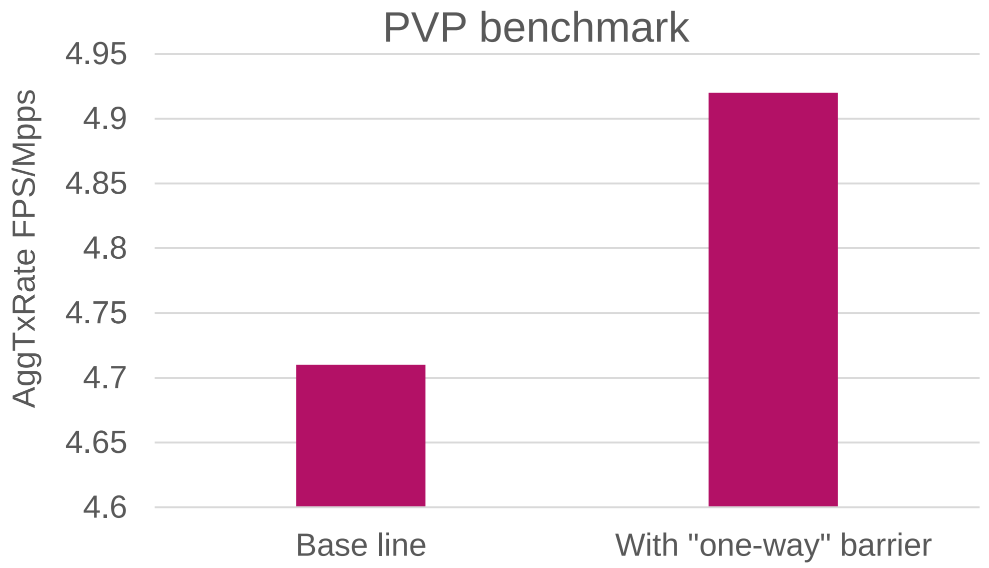
Relaxed memory ordering to save DMB operation



```
       virtio_xmit_pkts():
0.80     and    w1, w1, #0xffff
0.31     sub    w1, w1, w8
0.02     and    w1, w1, #0xffff
       virtio_rmb():
0.02   ↓ cbz    w2, fcc
         dmb    ishld
       virtio_xmit_pkts():
  868:   cmp    w1, w0
0.01     ldrh   w8, [x28, #52]
0.00     csel   w0, w1, w0, le
14.88    and    w21, w0, #0xffff
       virtio_xmit_cleanup():
```

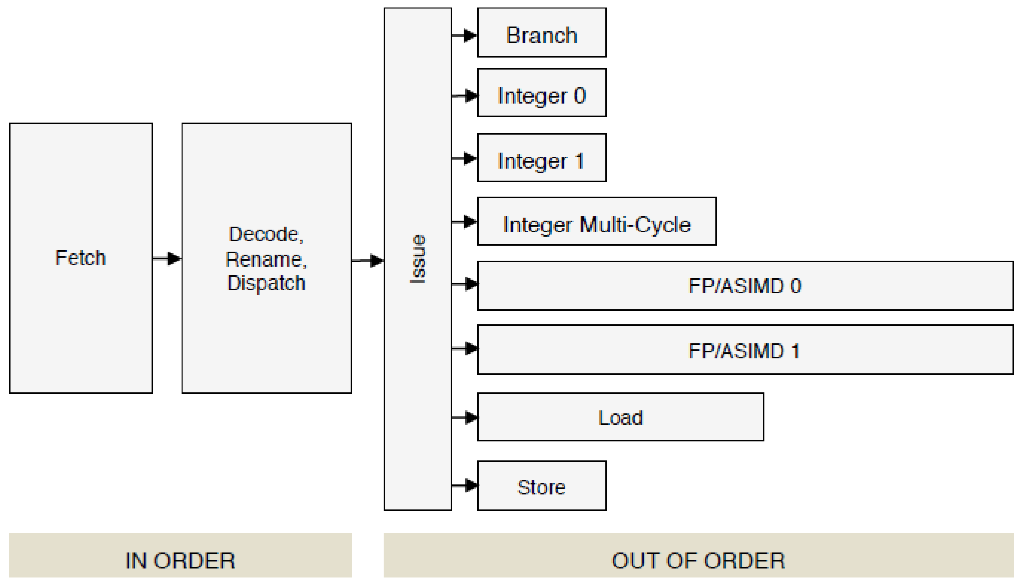
/drivers/net/virtio/virtqueue.h

```
static inline void vq_update_avail_idx(struct virtqueue *vq)
{
-       virtio_wmb(vq->hw->weak_barriers);
-       vq->vq_split.ring.avail->idx = vq->vq_avail_idx;
+       __atomic_store_n(&vq->vq_split.ring.avail->idx, vq->vq_avail_idx, __ATOMIC_RELEASE);
}
```



Loop unrolling

Loop unrolling attempts to optimize a program's execution speed by eliminating instructions that control the loop, which is an approach known as space-time tradeoff. Loops can be re-written as a repeated sequence of similar independent statements.



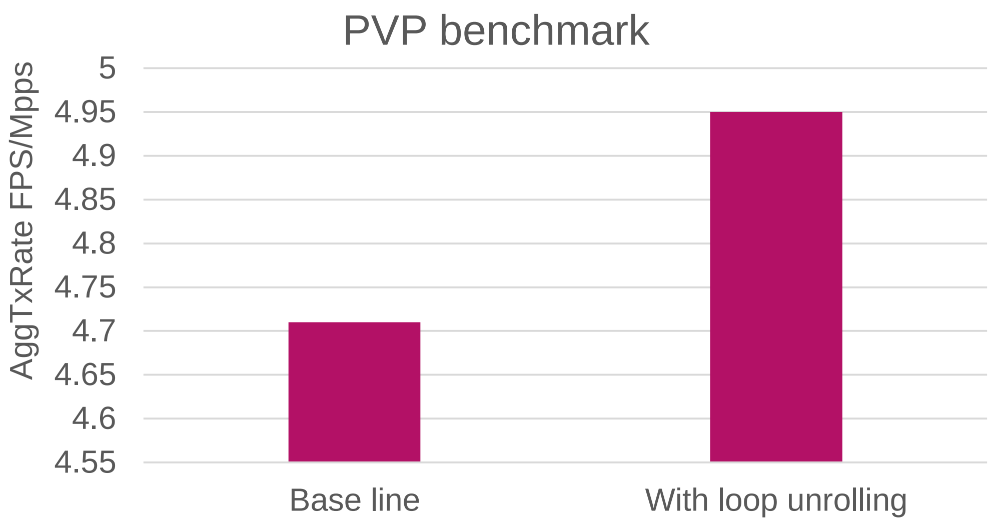
CortexA72 Pipeline

Advantages

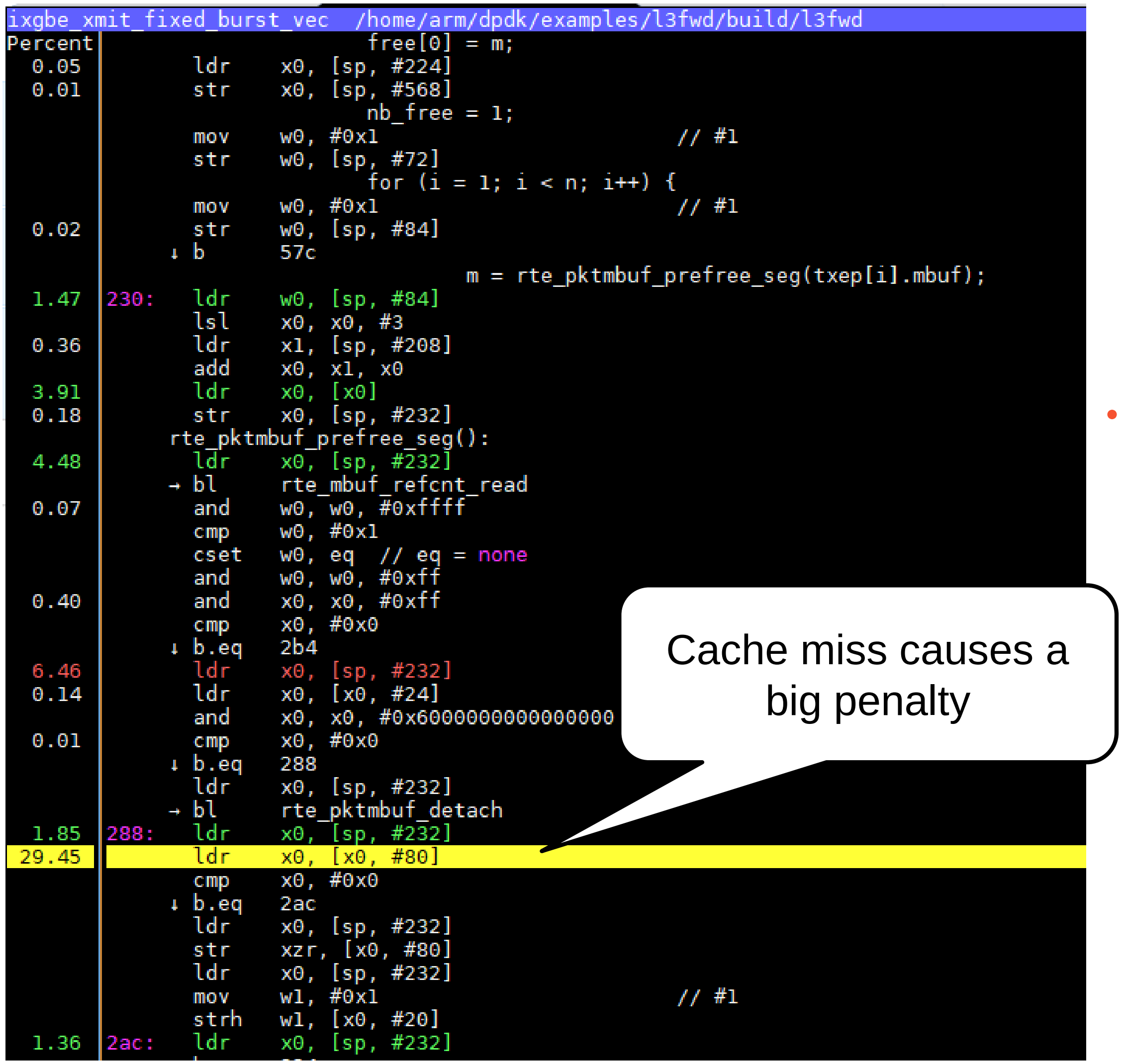
- Branch penalty is minimized
- Can potentially be executed in parallel if the statements in the loop are independent of each other
- Can be implemented dynamically if the number of array elements is unknown at compile time

Loop unrolling example and benefit

```
    i40e_tx_free_bufs(struct i40e_tx_queue *txq)
            if (likely(m != NULL)) {
                    free[0] = m;
                    nb_free = 1;
-                   for (i = 1; i < n; i++) {
+                   for (i = 1; i < n-1; i++) {
+                           rte_prefetch0(&txep[i].mbuf->next);
+                           m = rte_pktmbuf_prefree_seg(txep[i].mbuf);
+                           if (likely(m != NULL)) {
+                                   if (likely(m->pool == free[0]->pool)) {
+                                           free[nb_free++] = m;
+                                   } else {
+                                           rte_mempool_put_bulk(free[0]->pool,
+                                                   (void *)free, nb_free);
+                                           free[0] = m;
+                                           nb_free = 1;
+                                   }
+                           }
+                           m = rte_pktmbuf_prefree_seg(txep[++i].mbuf);
+                           if (likely(m != NULL)) {
+                                   if (likely(m->pool == free[0]->pool)) {
+                                           free[nb_free++] = m;
+                                   } else {
+                                           rte_mempool_put_bulk(free[0]->pool,
+                                                           (void *)free,
+                                                           nb_free);
+                                           free[0] = m;
+                                           nb_free = 1;
+                                   }
+                           }
+                   }
+                   if (i == (n-1)) {
                            rte_prefetch0(&txep[i].mbuf->next);
                            m = rte_pktmbuf_prefree_seg(txep[i].mbuf);
                            if (likely(m != NULL)) {
```



Prefetch



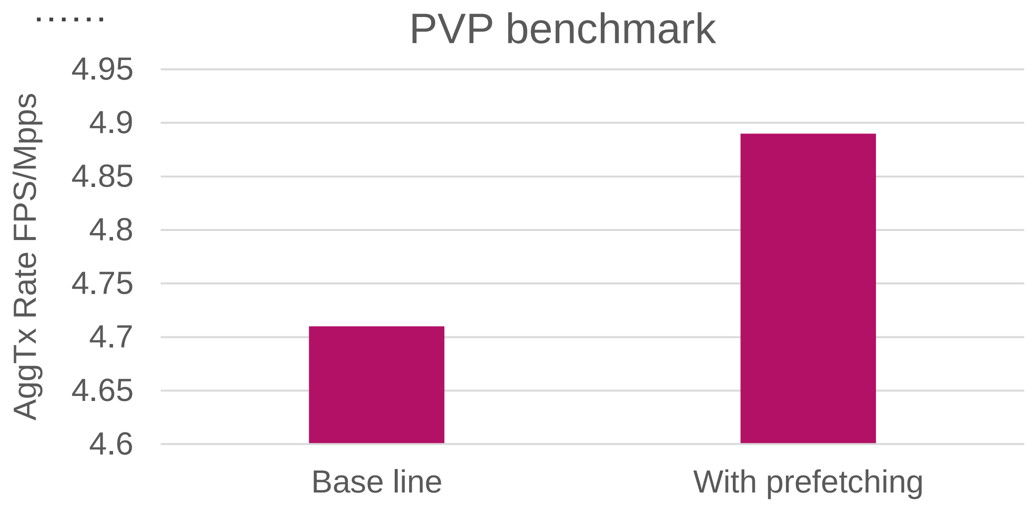
```
ixgbe_xmit_fixed_burst_vec  /home/arm/dpdk/examples/l3fwd/build/l3fwd
Percent│                   free[0] = m;
  0.05 │       ldr    x0, [sp, #224]
  0.01 │       str    x0, [sp, #568]
       │                   nb_free = 1;
       │       mov    w0, #0x1                        // #1
       │       str    w0, [sp, #72]
       │                   for (i = 1; i < n; i++) {
       │       mov    w0, #0x1                        // #1
  0.02 │       str    w0, [sp, #84]
       │     ↓ b      57c
       │                           m = rte_pktmbuf_prefree_seg(txep[i].mbuf);
  1.47 │230:   ldr    w0, [sp, #84]
       │       lsl    x0, x0, #3
  0.36 │       ldr    x1, [sp, #208]
       │       add    x0, x1, x0
  3.91 │       ldr    x0, [x0]
  0.18 │       str    x0, [sp, #232]
       │     rte_pktmbuf_prefree_seg():
  4.48 │       ldr    x0, [sp, #232]
       │     → bl     rte_mbuf_refcnt_read
  0.07 │       and    w0, w0, #0xffff
       │       cmp    w0, #0x1
       │       cset   w0, eq  // eq = none
       │       and    w0, w0, #0xff
  0.40 │       and    x0, x0, #0xff
       │       cmp    x0, #0x0
       │     ↓ b.eq   2b4
  6.46 │       ldr    x0, [sp, #232]
  0.14 │       ldr    x0, [x0, #24]
       │       and    x0, x0, #0x6000000000000000
  0.01 │       cmp    x0, #0x0
       │     ↓ b.eq   288
       │       ldr    x0, [sp, #232]
       │     → bl     rte_pktmbuf_detach
  1.85 │288:   ldr    x0, [sp, #232]
 29.45 │       ldr    x0, [x0, #80]
       │       cmp    x0, #0x0
       │     ↓ b.eq   2ac
       │       ldr    x0, [sp, #232]
       │       str    xzr, [x0, #80]
       │       ldr    x0, [sp, #232]
       │       mov    w1, #0x1                        // #1
       │       strh   w1, [x0, #20]
  1.36 │2ac:   ldr    x0, [sp, #232]
```

- Prefetch is the loading of a resource before it is required to decrease the time waiting for that resource

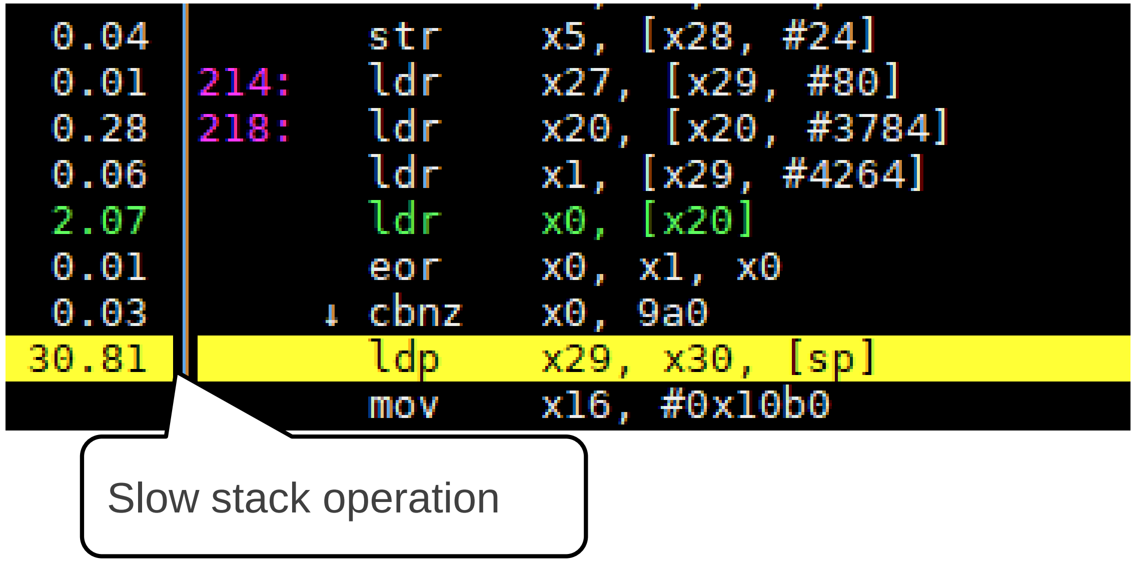
Prefetch example and benefit

```
i40e_tx_free_bufs(struct i40e_tx_queue *txq)
……
txep = &txq->sw_ring[txq->tx_next_dd - (n - 1)];
+  rte_prefetch0(&txep[0].mbuf->next);
   m = rte_pktmbuf_prefree_seg(txep[0].mbuf);
   if (likely(m != NULL)) {
      free[0] = m;
      nb_free = 1;
      for (i = 1; i < n; i++) {
+           rte_prefetch0(&txep[i].mbuf->next);
         m = rte_pktmbuf_prefree_seg(txep[i].mbuf);
         if (likely(m != NULL)) {
            if (likely(m->pool == free[0]->pool)) {
……
```

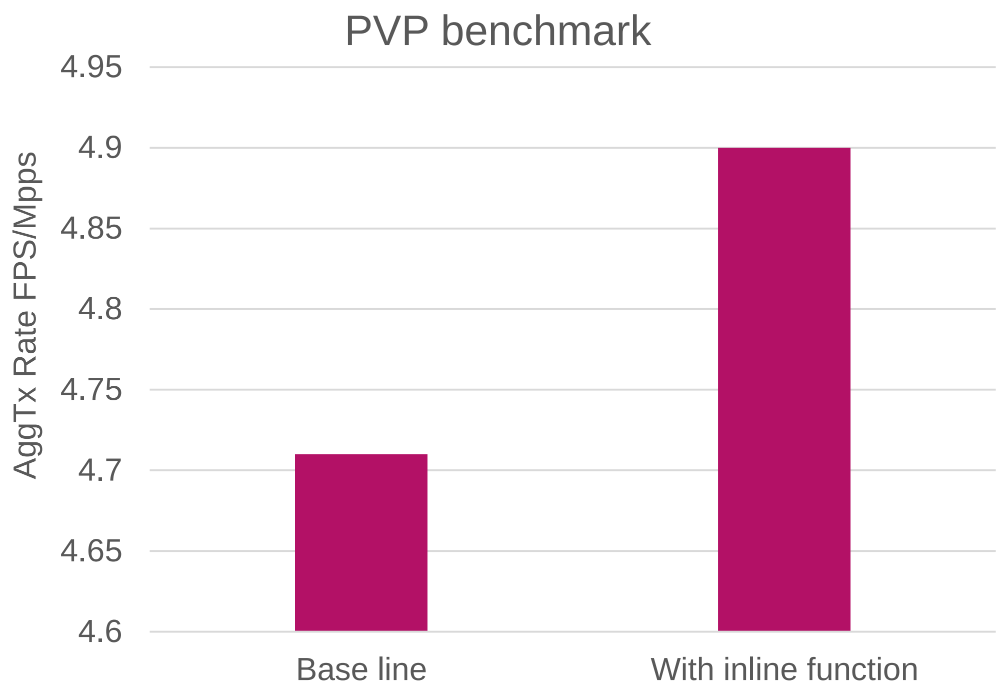
```
i40e_tx_free_bufs(struct i40e_tx_queue *txq)
……
      rte_mempool_put_bulk(free[0]->pool, (void **)free, nb_free);
} else {
      for (i = 1; i < n; i++) {
+        rte_prefetch0(&txep[i].mbuf->next);
         m = rte_pktmbuf_prefree_seg(txep[i].mbuf);
         if (m != NULL)
            rte_mempool_put(m->pool, m);
……
```



Inline function



```
  0.04             str     x5, [x28, #24]
  0.01  214:       ldr     x27, [x29, #80]
  0.28  218:       ldr     x20, [x20, #3784]
  0.06             ldr     x1, [x29, #4264]
  2.07             ldr     x0, [x20]
  0.01             eor     x0, x1, x0
  0.03           ↓ cbnz    x0, 9a0
 30.81             ldp     x29, x30, [sp]
                   mov     x16, #0x10b0
```



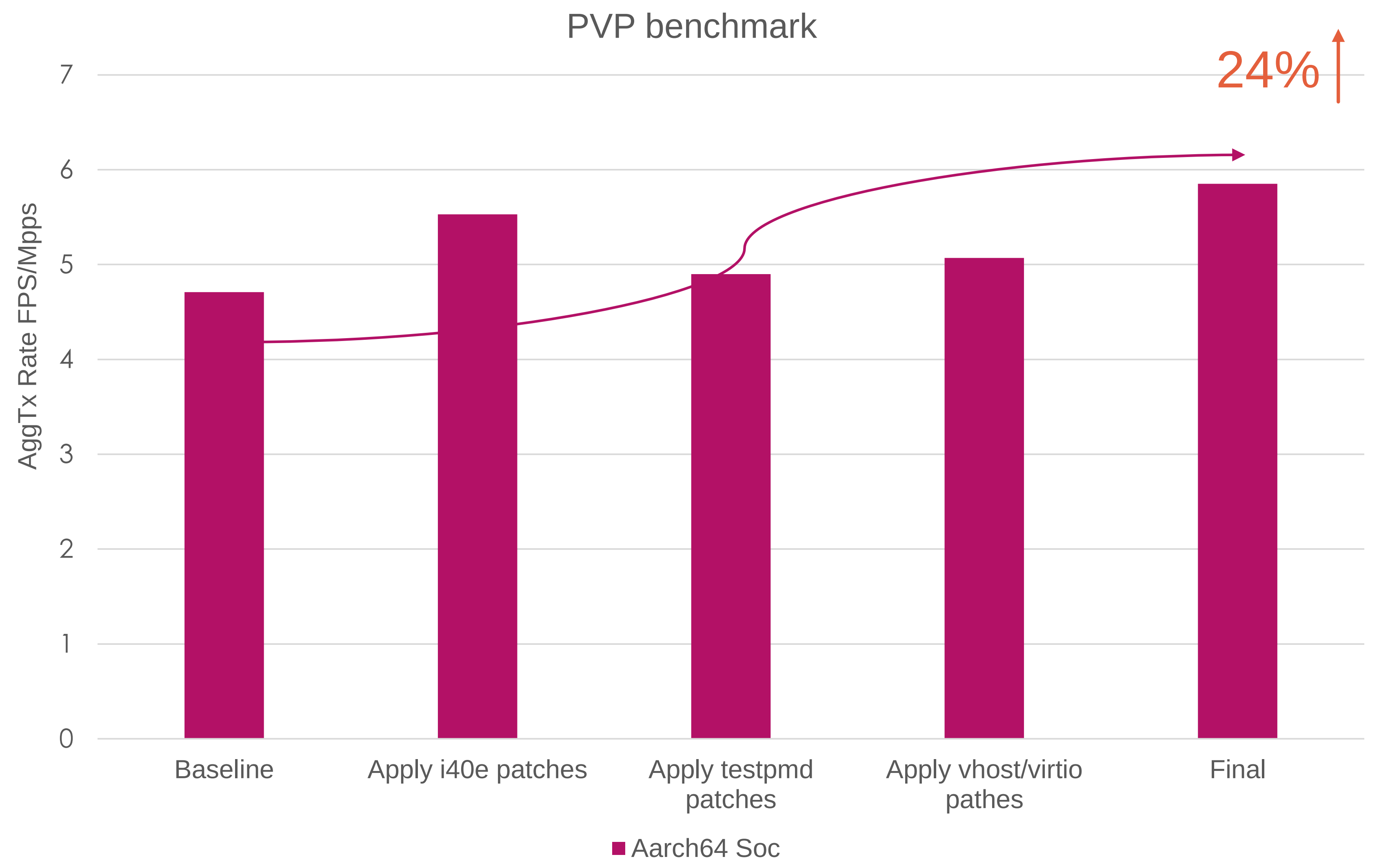
- Inline function can help to save the function call cost (stack operations)

DPDK
DATA PLANE DEVELOPMENT KIT

Agenda

- Test Scenario
- Testbed and Baseline Performance
  - NUMA balancing
  - VHE
- Analysis and Tuning
  - Weak-Memory Model
  - Loop unrolling
  - Prefetch
  - Inline function
- Performance Data Result

Performance Data Result

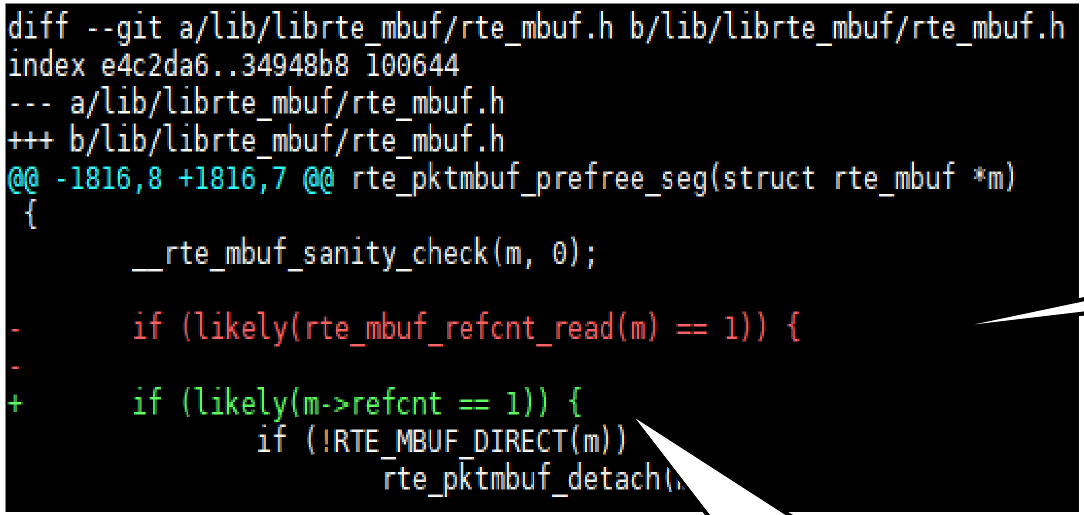


DPDK
DATA PLANE DEVELOPMENT KIT

Joyce Kong
joyce.kong@arm.com
Gavin Hu
gavin.hu@arm.com

Thanks

Misc



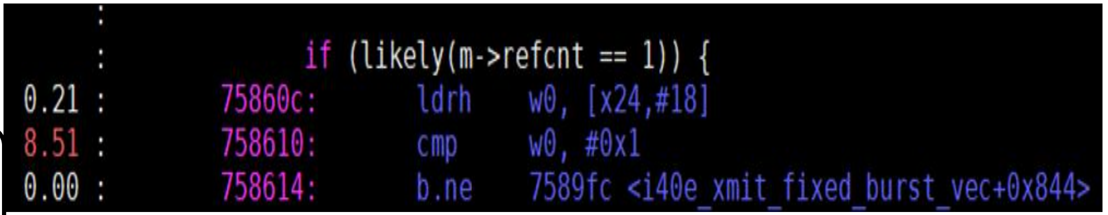
```
diff --git a/lib/librte_mbuf/rte_mbuf.h b/lib/librte_mbuf/rte_mbuf.h
index e4c2da6..34948b8 100644
--- a/lib/librte_mbuf/rte_mbuf.h
+++ b/lib/librte_mbuf/rte_mbuf.h
@@ -1816,8 +1816,7 @@ rte_pktmbuf_prefree_seg(struct rte_mbuf *m)
 {
        __rte_mbuf_sanity_check(m, 0);

-       if (likely(rte_mbuf_refcnt_read(m) == 1)) {
-
+       if (likely(m->refcnt == 1)) {
                if (!RTE_MBUF_DIRECT(m))
                        rte_pktmbuf_detach(
```

```
rte_mbuf_refcnt_read(const struct rte_mbuf *m)
{
    return(uint16_t)(rte_atomic16_read(&m- >refcnt_atomic));
}

rte_atomic16_read(const rte_atomic16_t *v)
{
                return v->cnt;
}
```

Compare directly



```
       :
       :             if (likely(m->refcnt == 1)) {
0.21 :        75860c:       ldrh    w0, [x24,#18]
8.51 :        758610:       cmp     w0, #0x1
0.00 :        758614:       b.ne    7589fc <i40e_xmit_fixed_burst_vec+0x844>
```