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
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

PVP benchmark

AggTx Rate FPS/Mpps

Soc#1 (2 numa nodes) | Soc#2 (2 numa nodes)
--- | ---
Enable numa balancing | Disable numa balancing

0 | 0.5 | 1 | 1.5 | 2 | 2.5 | 3 | 3.5 | 4 | 4.5 | 5
--- | --- | --- | --- | --- | --- | --- | --- | --- | --- | ---

Enable numa balancing
Disable numa balancing
VHE (Virtualization Host Extensions)

KVM/ARM without VHE

- Host User Space
- VM User Space
- Hypervisor
- Host Kernel
- Lowvisor
- Function call
- Run VM
- Switch
- Trap
- Perform Context Switch between Host and VM execution context.
- Configure VGIC, virtual timer.
- Setup stage 2 translation registers.
- Enable stage 2 translation
- Enable traps

KVM/ARM with VHE

- Host User Space
- VM User Space
- Hypervisor
- Host Kernel
- VM Kernel
- Switch
- Trap
- Host Kernel
- Function call
- Run VM
- Switch
- Trap
- Perform Context Switch between VM and Host execution context.
- Disable virtual interrupt.
- Disable stage 2 translation.
- Disable traps

PVP benchmark

- AggTx Rate FPS/Mpps
  - Soc#1 (without VHE)
  - Soc#2 (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

- 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

<table>
<thead>
<tr>
<th>Strong-Memory Order</th>
<th>Weak-Memory Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>All reads and writes are in-order</td>
<td>Reads and Writes are arbitrarily re-ordered, subject only to data dependencies and explicit memory barrier instructions</td>
</tr>
</tbody>
</table>
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
“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

Enqueue to and dequeue from Virtio Vring
“One-way” barrier optimization

```
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);
}
```

Relaxed memory ordering to save DMB operation

PVP benchmark
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.

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

```c
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++) {
            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;
                }
            } 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;
            }
        }
    }
```

---

**PVP benchmark**

- **Base line**
  - Agg Tx Rate FPS/Mpps: 4.55
- **With loop unrolling**
  - Agg Tx Rate FPS/Mpps: 4.95
Prefetch

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

Cache miss causes a big penalty.
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)) {
                    ......
    }
}
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);
    }
}
......

i40e_tx_free_bufs(struct i40e_tx_queue *txq)
......
    rte_mempool_put_bulk(free[0]->pool, (void **)free, nb_free);
}

PVP benchmark

<table>
<thead>
<tr>
<th>AggTx Rate FPS/Mpps</th>
<th>Base line</th>
<th>With prefetching</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4.6</td>
<td>4.95</td>
</tr>
<tr>
<td></td>
<td>4.65</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4.7</td>
<td>4.95</td>
</tr>
<tr>
<td></td>
<td>4.75</td>
<td>4.95</td>
</tr>
<tr>
<td></td>
<td>4.8</td>
<td>4.95</td>
</tr>
<tr>
<td></td>
<td>4.85</td>
<td>4.95</td>
</tr>
<tr>
<td></td>
<td>4.9</td>
<td>4.95</td>
</tr>
<tr>
<td></td>
<td>4.95</td>
<td>4.95</td>
</tr>
</tbody>
</table>

Base line

With prefetching
Inline function

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

Slow stack operation

<table>
<thead>
<tr>
<th>Code</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.04</td>
<td>str x5, [x28, #24]</td>
</tr>
<tr>
<td>0.01</td>
<td>ldr x27, [x29, #80]</td>
</tr>
<tr>
<td>0.28</td>
<td>ldr x20, [x20, #3784]</td>
</tr>
<tr>
<td>0.06</td>
<td>ldr x1, [x29, #4264]</td>
</tr>
<tr>
<td>2.07</td>
<td>ldr x0, [x20]</td>
</tr>
<tr>
<td>0.01</td>
<td>eor x0, x1, x0</td>
</tr>
<tr>
<td>0.03</td>
<td>cbnz x0, 9a0</td>
</tr>
<tr>
<td>30.81</td>
<td>ldp x29, x30, [sp]</td>
</tr>
<tr>
<td></td>
<td>mov x16, #0x10b0</td>
</tr>
</tbody>
</table>

PVP benchmark

- Base line
- With inline function

<table>
<thead>
<tr>
<th>AggTx Rate FPS/Mpps</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.6</td>
</tr>
<tr>
<td>4.75</td>
</tr>
<tr>
<td>4.85</td>
</tr>
<tr>
<td>4.95</td>
</tr>
</tbody>
</table>
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

PVP benchmark

Baseline
Apply i40e patches
Apply testpmd patches
Apply vhost/virtio patches
Final

AggTx Rate FPS/Mpps

Aarch64 Soc

24%
Thanks

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

if (likely(m->refcnt == 1)) {
    if (!rTE_MBUF_DIRECT(m))
        rte_pktmbuf_detach(m);
}

Compare directly