The path to Fast Data on Arm

FD.io Mini Summit @ ONS NA ‘18
Use case

VPP

- L2 cross connect
- IPv4 routing
- 64B @ 10Gbps
- Single flow & direction
- Single core
Tracing packets

How does the packet traverse the graph?

```
00:19:59:168489: mrvl-pp2-input
  pp2: mv-ppio0/0 (1) next-node ethernet-input
       l3_offset 16 (0x10) ip_hdlen 5 ec 2 es 0 pool_id 8 hwf_sync 0 l4.chk_ok 0
       ip_frag 0 ipv4_hdr_err 1 l4_info 0 l3_info 0 buf_header 0 lookup_id 28 (0x1c)
       cpu_code 0 pppoe 0 l3_cast_info 0 l2_cast_info 0 vlan_info 0 byte_count 62 (0x3e)
       gem_port_id 0 color 0 gop_sop_u 0 key_hash_enable 0 l4chk 56856 (0xde18)
       timestamp 0 buf_phys_ptr_lo 861104320 (0x353650e0) buf_phys_ptr_hi 1 key_hash 13215410 (0xc9a6b2)
       buf_virt_ptr_lo 249248 (0x3cda0) buf_virt_ptr_hi 0 buf_qset_no 0 buf_type 0
       mod_dscp 0 mod_flag 0 mdscp 0 mpri 0 mgpid 1 port_num 0

00:19:59:168555: ethernet-input
  RESERVED: 3c:fd:fe:12:26:e0 -> 00:51:82:11:22:00

00:19:59:168588: l2-input
  l2-input: sw_if_index 1 dst 00:51:82:11:22:00 src 3c:fd:fe:12:26:e0

00:19:59:168598: l2-output
  l2-output: sw_if_index 2 dst 00:51:82:11:22:00 src 3c:fd:fe:12:26:e0 data ff ff 0a aa aa aa aa aa

00:19:59:168605: mv-ppio1/0-output
  mv-ppio1/0
  RESERVED: 3c:fd:fe:12:26:e0 -> 00:51:82:11:22:00
```
Runtime clocks

Not CPU clock cycles!

```bash
dmesg | grep MHz
```

[0.000000] Architected cp15 timer(s) running at 25.00MHz (phys).
[0.000001] sched_clock: 56 bits at 25MHz, resolution 40ns, wraps every 439804651100ns
Batch size

Does batch size affect performance?

I/O device *can* fill big frames

<table>
<thead>
<tr>
<th>VLIB_FRAME_SIZE</th>
<th>64</th>
<th>128</th>
<th>256</th>
<th>512</th>
<th>1024</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vectors/Call</td>
<td>64</td>
<td>128</td>
<td>256</td>
<td>512</td>
<td>1024</td>
</tr>
</tbody>
</table>

Does batch size affect performance?

I/O device can fill big frames
Identifying hotspots

First access to packet data

```c
b0 = vlib_get_buffer (vm, bi0);
b1 = vlib_get_buffer (vm, bi1);
	error0 = error1 = ETHERNET_ERROR_NONE;
e0 = vlib_buffer_get_current (b0);
type0 = clib_net_to_host_u16 (e0->type);
e1 = vlib_buffer_get_current (b1);
type1 = clib_net_to_host_u16 (e1->type);

/* Speed-path for the untagged case */
if (PREDICT_TRUE (variant == ETHERNET_INPUT & & !ethernet_frame_is_an
```

Why is memory access the hotspot?

```c
prfm pldl1keep, [x0]
add x0, x25,
lshr w15, [x12,#12]
prfm pldl1keep, [x1]
str x0, [x29,#592]
sub w0, w28, #0x2
mov w1, w15
str w0, [x29,#576]
rev16 w1, w1
ldrh w0, [x11,#12]
mov v0.h[0], w1
rev16 w0, w0
mov v1.h[0], w0
```
Dual-loop explained

```c
void go_go_go(int *from, size_t num)
{
    while (num >= 4)
    {
        p0 = get_packet(from[0]);
        p1 = get_packet(from[1]);
        p2 = get_packet(from[2]);
        p3 = get_packet(from[3]);
        prefetch(p2->data);
        prefetch(p3->data);
        process(p0);
        process(p1);
        from += 2; num -= 2;
    }
    while (num > 0)
    {
        // ...
    }
}
```

Loop unrolled twice
Loop body interleaved
Prefetch 1 iteration ahead
I/O memory latency

New ARM AMBA 5 CHI Protocol Enhancements

RX data serviced over DDR in I/O coherent system

RX data serviced from cache via cache stashing

ns

2p2660v4 armada8040
## Tuning prefetches

### Dual Loop

<table>
<thead>
<tr>
<th>Stride</th>
<th>Mpps</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1</td>
<td>3.47</td>
<td>1.53e0</td>
</tr>
<tr>
<td>+2</td>
<td>3.74</td>
<td>1.13e0</td>
</tr>
<tr>
<td>+3</td>
<td>3.82</td>
<td>1.03e0</td>
</tr>
<tr>
<td>+4</td>
<td>3.78</td>
<td>1.04e0</td>
</tr>
<tr>
<td>+5</td>
<td>3.76</td>
<td>1.04e0</td>
</tr>
</tbody>
</table>

### Single Loop

<table>
<thead>
<tr>
<th>Stride</th>
<th>Mpps</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1</td>
<td>3.33</td>
<td>1.85e0</td>
</tr>
<tr>
<td>+2</td>
<td>3.48</td>
<td>1.49e0</td>
</tr>
<tr>
<td>+3</td>
<td>3.62</td>
<td>1.29e0</td>
</tr>
<tr>
<td>+4</td>
<td>3.62</td>
<td>1.18e0</td>
</tr>
<tr>
<td>+5</td>
<td>3.75</td>
<td>1.10e0</td>
</tr>
<tr>
<td>+6</td>
<td>3.77</td>
<td>1.05e0</td>
</tr>
<tr>
<td>+7</td>
<td>3.77</td>
<td>1.03e0</td>
</tr>
<tr>
<td>+8</td>
<td>3.78</td>
<td>1.03e0</td>
</tr>
<tr>
<td>+9</td>
<td>3.77</td>
<td>1.04e0</td>
</tr>
<tr>
<td>+10</td>
<td>3.75</td>
<td>1.06e0</td>
</tr>
</tbody>
</table>

### Single Loop – Split PF

<table>
<thead>
<tr>
<th>S1</th>
<th>S2</th>
<th>Mpps</th>
<th>Clocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>+1</td>
<td>+8</td>
<td>3.68</td>
<td>1.03e0</td>
</tr>
<tr>
<td>+2</td>
<td>+8</td>
<td>3.77</td>
<td>1.01e0</td>
</tr>
<tr>
<td>+3</td>
<td>+8</td>
<td>3.71</td>
<td>1.05e0</td>
</tr>
<tr>
<td>+2</td>
<td>+9</td>
<td>3.81</td>
<td>9.58e-1</td>
</tr>
<tr>
<td>+2</td>
<td>+10</td>
<td>3.81</td>
<td>9.58e-1</td>
</tr>
<tr>
<td>+2</td>
<td>+11</td>
<td>3.82</td>
<td>9.49e-1</td>
</tr>
<tr>
<td>+2</td>
<td>+12</td>
<td>3.84</td>
<td>9.24e-1</td>
</tr>
<tr>
<td>+2</td>
<td>+13</td>
<td>3.84</td>
<td>9.15e-1</td>
</tr>
<tr>
<td>+2</td>
<td>+14</td>
<td>3.82</td>
<td>9.37e-1</td>
</tr>
<tr>
<td>+3</td>
<td>+13</td>
<td>3.80</td>
<td>9.49e-1</td>
</tr>
</tbody>
</table>

**Is load-to-use time the best we can “predict”?**

**Tune separately for L1 and L2 latencies**
Avoiding bottlenecks

“The L1 memory system is non-blocking and supports hit-under-miss. For Normal memory, up to six 64-byte cache line requests can be outstanding at a time. While those requests are waiting for memory, loads to different cache lines can hit the cache and return their data.”


Prefetching combined with loop unrolling is demanding!
Types of data accesses

**Device descriptor**

```c
b->total_length_not_including_f
b->flags = VLIB_BUFFER_TOTAL_LE
mov w8,
    pp2_ppio_inq_desc_get_pkt_len();
ldrh w0, [x21,#6]
Mrvl_pp2_set_buf_data_len_flags()
Mrvl_pp2_set_buf_data_len_flags();
```

**vlib_buffer_t**

```c
    d += 2;
    buffers += 2;
    n_desc -= 2;
    strh w1, [x29,#360]
    if (n_trace > 0)
        Mrvl_pp2_input_trac
```

**Frame Size**

<table>
<thead>
<tr>
<th>Frame Size</th>
<th>Vector (4B)</th>
<th>Descriptor (32B)</th>
<th>Buffer (128B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>.25KB</td>
<td>2KB</td>
<td>8KB</td>
</tr>
<tr>
<td>128</td>
<td>.5KB</td>
<td>4KB</td>
<td>16KB</td>
</tr>
<tr>
<td>256</td>
<td>1KB</td>
<td>8KB</td>
<td>32KB</td>
</tr>
<tr>
<td>512</td>
<td>2KB</td>
<td>16KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

**vlib_physmem_region_t**

```c
    3.88 add    x1, x0,
    11.98 ldrb   w0, [x0,#22]
    14.23 add    x0, x7, x0, lsl
    27.81 ldr    x14, [x0,#8]
```
Initial remarks
64B packet – single flow – single core

Observations
• Most hotspots are memory accesses
• Software-defined data placement consumes processing cycles
• Unintentionally ordering memory accesses can slow the system down
• Compiler may fuse loops which alters memory access pattern from original program order

Further Directions
• Leverage PMU data
• Compiler and C library versions
• Multicore scaling
• Hardware offload
• Other platforms
Preliminary results

64B packet – single flow – single core

<table>
<thead>
<tr>
<th>Size</th>
<th>Baseline</th>
<th>New</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mpps</td>
<td></td>
</tr>
<tr>
<td>64B</td>
<td>1.31x</td>
<td></td>
</tr>
<tr>
<td>128B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>256B</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

% line rate vs. packet size

- l2xc
- ip4-routing

© 2017 Arm Limited
The path to on Arm

<table>
<thead>
<tr>
<th>Workload Scale</th>
<th>Performance Analysis</th>
<th>Software</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSIT</td>
<td>Hotspot &amp; Bottleneck Identification</td>
<td>Upstream</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Libraries</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Toolchain</td>
</tr>
<tr>
<td>FD.io Lab</td>
<td>Tuning &amp; Optimization</td>
<td>Hardware</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Processors</td>
</tr>
<tr>
<td></td>
<td></td>
<td>I/O</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Accelerators</td>
</tr>
</tbody>
</table>

- **CSIT**: CSIT (Cloud Service Integration Tools)
- **FD.io Lab**: FD.io Lab
- **Software**:
  - Upstream
  - Libraries
  - OS
  - Toolchain
- **Hardware**:
  - Processors
  - I/O
  - Accelerators
The path to FD.io on Arm

• Workload Scale
  • Continue integration of Arm-based platforms into FD.io lab
  • Adopt and run CSIT on a diverse range of machines and topologies

• Performance Analysis
  • Distill critical runtime components affecting performance
  • Identify solutions to hotspots and/or bottlenecks

• Upstream
  • Integrate solutions back into open source
Thank You!
Danke!
Merci!
谢谢!
ありがとう!
Gracias!
Kiitos!
감사합니다
धन्यवाद
gcc (Ubuntu/Linaro 5.4.0-6ubuntu1~16.04.9) 5.4.0 20160609

ldd (Ubuntu GLIBC 2.23-0ubuntu10) 2.23

$ cat /proc/cmdline
console=ttyS0,115200 root=/dev/sda1 rw

$ cat /sys/kernel/mm/hugepages/hugepages-2048kB/nr_hugepages
16

$ lscpu
Architecture:          aarch64
Byte Order:            Little Endian
CPU(s):                4
On-line CPU(s) list:   0-3
Thread(s) per core:    1
Core(s) per socket:    2
Socket(s):             2
CPU max MHz:           2000.0000
CPU min MHz:           100.0000
Hypervisor vendor:     (null)
Virtualization type:   full
“Debugging is the act of asking questions and answering them, not guessing what the answer is.

You want to form questions, not hypotheses. Answers to questions constrain hypotheses.

We repeat this process. Specific questions. Specific answers. More specific questions. More specific answers. And then that ‘hypothetical leap’ is often not a leap at all. It’s a step across a puddle.

That is how we debug. We debug by having the cycle of questions and answers.

We are not magicians. We are the wizard of Oz sweating behind a curtain frenetically turning a crank trying to figure out the problem.”

Bryan Cantrill

*Debugging Under Fire: Keep your Head when Systems have Lost their Mind*