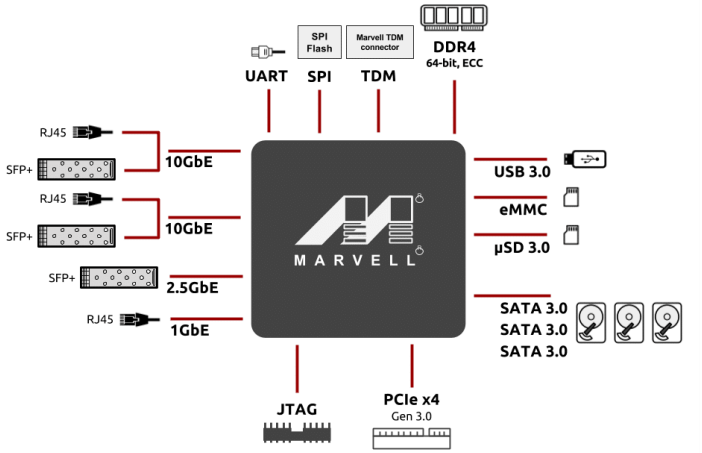
arm

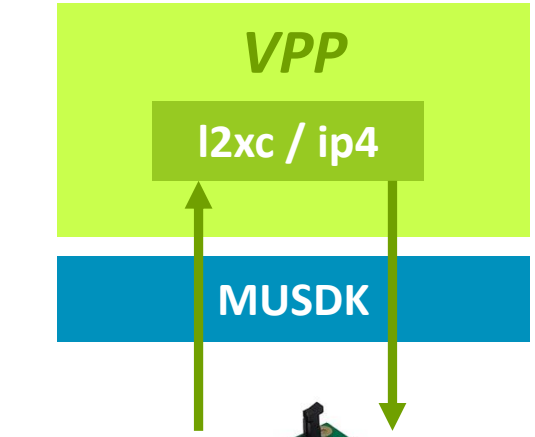
DP Benchmarking on Arm

NFV Data Plane Benchmarking @ ONS NA ‘18

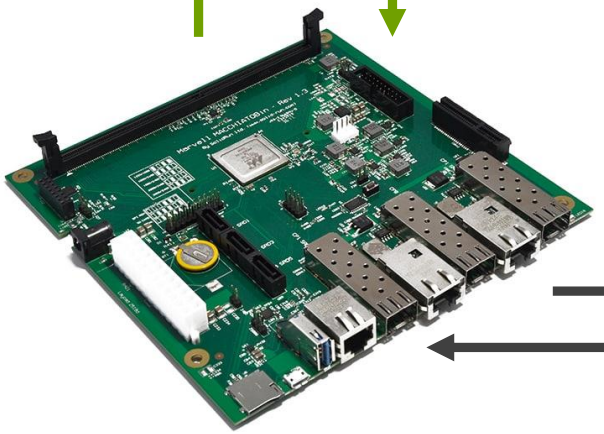
Brian Brooks <brian.brooks@arm.com>
Tina Tsou <tina.tsou@arm.com>
Sirshak Das <sirshak.das@arm.com>

Use Case





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



MACCHIATObin

TRex
Realistic Traffic Generator

arm

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_frg 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 (0x335368c0) 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_pri 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 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

ARMv8 Generic Timer
Not CPU clock cycles!

```
vpp# show run
Time 125.1, average vectors/node 255.99, last 128 main loops 12.00 per node 256.00
  vector rates in 2.5052e6, out 2.5052e6, drop 0.0000e0, punt 0.0000e0
             Name                 State         Calls          Vectors        Suspends         Clocks       Vectors/Call
acl-plugin-fa-cleaner-process  event wait            0                0               1          1.10e2            0.00
..
dns-resolver-process            any wait            0                0               1          4.30e1            0.00
ethernet-input                   active         1224065        313360152               0          1.54e0          255.99
fib-walk                        any wait            0                0              63          7.37e1            0.00
..
ip6-reassembly-expire-walk      any wait            0                0              13          6.49e1            0.00
l2-input                         active         1224065        313360152               0         4.25e-1          255.99
l2-output                        active         1224065        313360152               0         2.28e-1          255.99
..
mrvl-pp2-input                   polling       47489276        313360408               0          3.87e0            6.59
mv-ppio0/0-output                active               1                1               0          2.20e1            1.00
mv-ppio0/0-tx                    active               1                1               0          4.80e1            1.00
mv-ppio1/0-output                active         1224064        313360151               0         1.75e-1          255.99
mv-ppio1/0-tx                    active         1224064        313360151               0          1.27e0          255.99
nat-det-expire-walk               done               1                0               0          3.40e1            0.00
..
unix-cli-stdin                   active               0                0             141          2.59e7            0.00
unix-epoll-input                 polling          46332                0               0          2.60e1            0.00
..
```

62ns

17ns

9ns

155ns

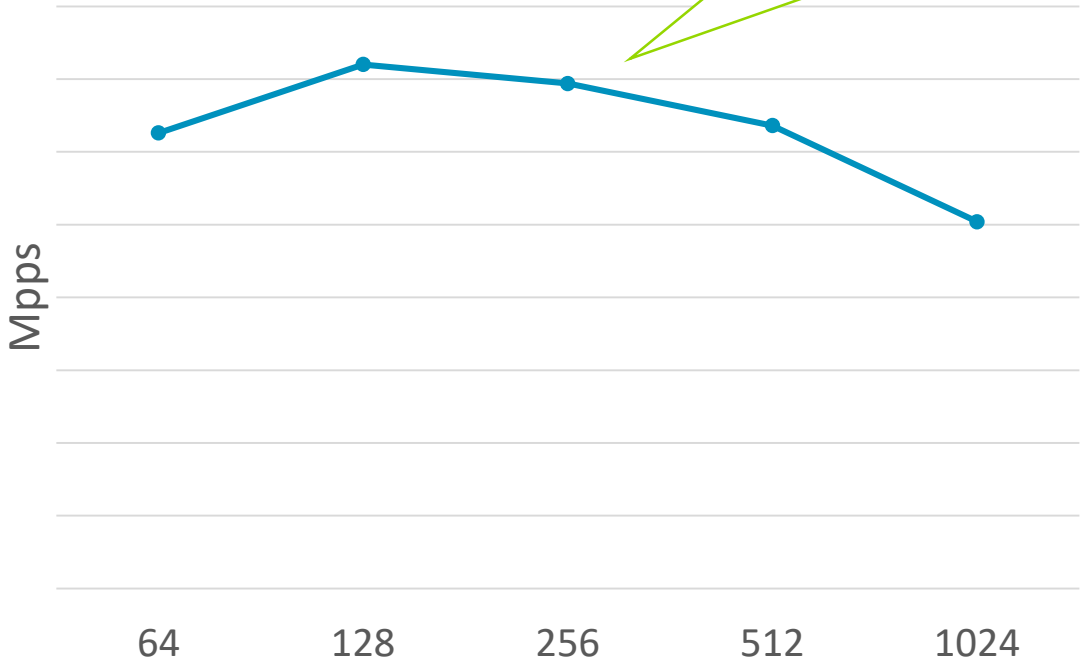
7ns

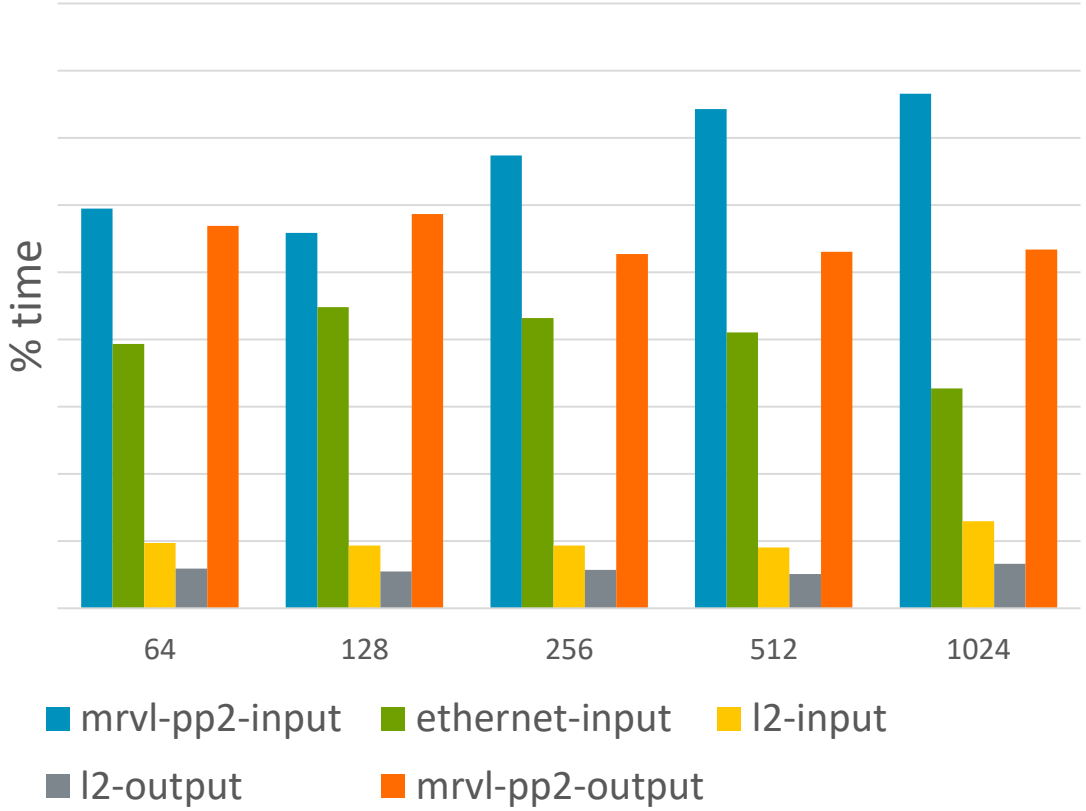
51ns

```
$ 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 4398046511100ns
```

Batch Size

Does batch size affect performance?





VLIB_FRAME_SIZE	64	128	256	512	1024
Vectors/Call	64	128	256	512	1024

I/O device *can* fill big frames

Identifying Hotspots

First access to packet data

```
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_INPU
                  && !ethernet_frame_is_an
```

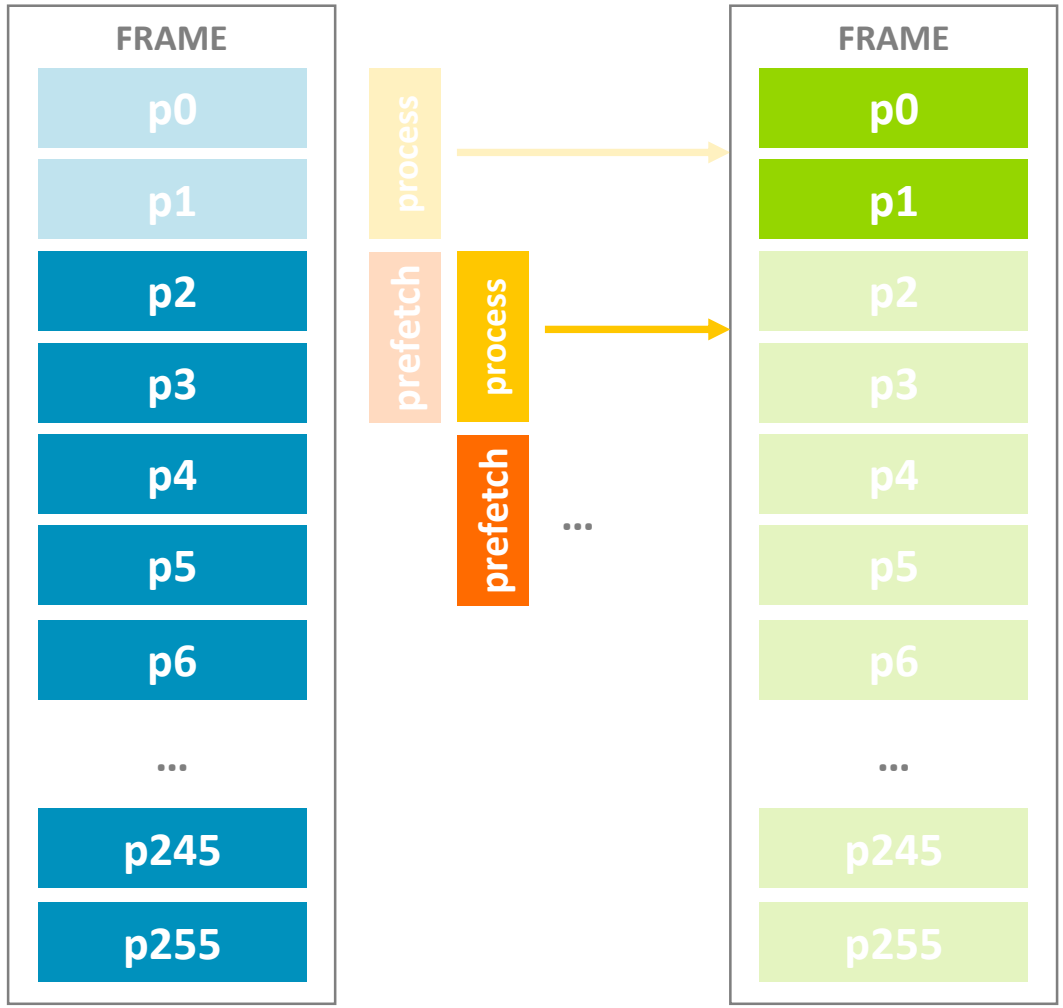
```
 0.44   prfm   pldl1keep, [x0]
        add    x0, x25,
59.48   ldrh   w15, [x12,#12]
 0.03   prfm   pldl1keep, [x1]
 0.45   str    x0, [x29,#592]
        sub    w0, w28, #0x2
        mov    w1, w15
 0.05   str    w0, [x29,#576]
        rev16  w1, w1
10.05   ldrh   w0, [x11,#12]
        mov    v0.h[0], w1
        rev16  w0, w0
        mov    v1.h[0], w0
```

Why is memory access the hotspot?

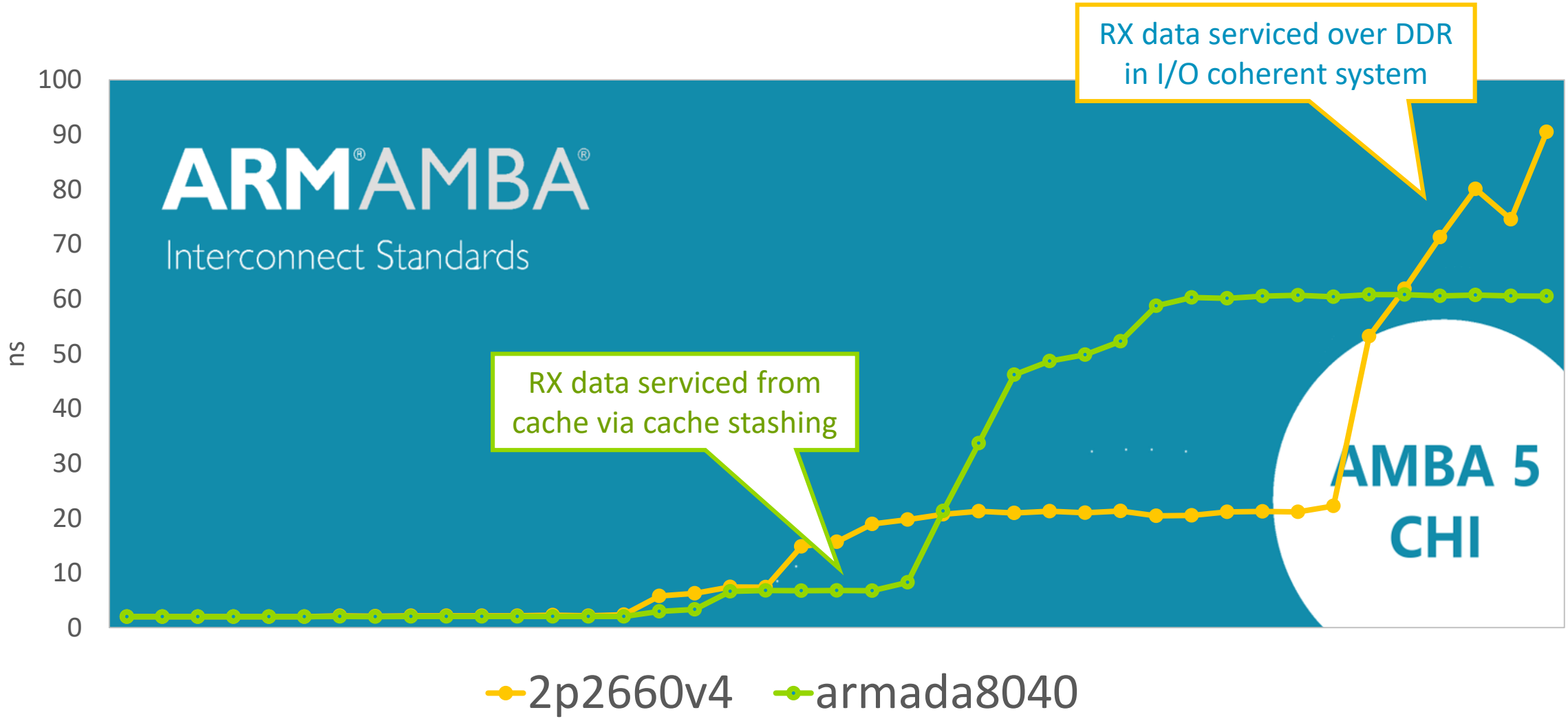
Dual-loop Explained

```
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



Tuning Prefetches

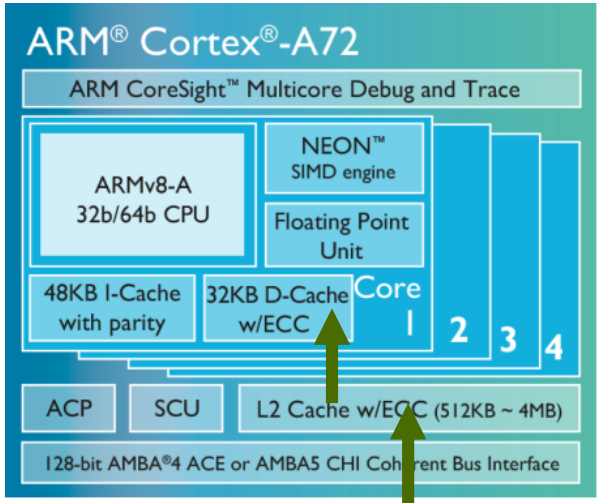
Dual Loop		
Stride	Mpps	Clocks
+1	**3.47**	**1.53e0**
+2	3.74	1.13e0
+3	**3.82**	**1.03e0**
+4	3.78	1.04e0
+5	3.76	1.04e0

61ns

Single Loop		
Stride	Mpps	Clocks
+1	3.33	1.85e0
+2	3.48	1.49e0
+3	3.62	1.29e0
+4	3.62	1.18e0
+5	3.75	1.10e0
+6	3.77	1.05e0
+7	3.77	1.03e0
+8	**3.78**	**1.03e0**
+9	3.77	1.04e0
+10	3.75	1.06e0

Single Loop – Split PF			
S1	S2	Mpps	Clocks
+1	+8	3.68	1.03e0
+2	+8	3.77	1.01e0
+3	+8	3.71	1.05e0
+2	+9	3.81	9.58e-1
+2	+10	3.81	9.58e-1
+2	+11	3.82	9.49e-1
+2	+12	3.84	9.24e-1
+2	**+13**	**3.84**	**9.15e-1**
+2	+14	3.82	9.37e-1
+3	+13	3.80	9.49e-1

37ns



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."*

ARM® Cortex®-A72 MPCore Processor Technical Reference Manual

```
                add     x5, x1,
1.57            prfm    pldl1keep, [x1]
                mov     x1, x9
0.25            prfm    pldl1keep, [x2]
                add     x6, x2,
                add     x8, x4,
                ubfiz   x2, x1, #6, #32
0.67            add     x7, x3,
0.41            prfm    pldl1keep, [x4]
1.02            prfm    pstl1keep, [x6]
1.20            prfm    pldl1keep, [x3]
2.37            prfm    pstl1keep, [x8]
21.62           prfm    pstl1keep, [x7]
2.95            prfm    pstl1keep, [x5]
                add     x5, x20,
0.52            ldrsh   x15, [x2,x0]
0.70            ldr     x3, [x27,#384]
```

Load Store Unit busy?

Prefetching combined with loop unrolling is demanding!

Types of Data Accesses

Device descriptor

```
 0.33   b->total_length_not_including_f
        b->flags = VLIB_BUFFER_TOTAL_LE
        mov    w8,
        pp2_ppio_inq_desc_get_pkt_len():
15.41   ldrh   w0, [x21,#6]
        mrvl_pp2_set_buf_data_len_flags()
        mrvl_pp2_set_buf_data_len_flags (
```

Frame Size	Vector (4B)	Descriptor (32B)	Buffer (128B)
64	.25KB	2KB	8KB
128	.5KB	4KB	16KB
256	1KB	8KB	32KB
512	2KB	16KB	64KB

vlib_buffer_t

```
        d += 2;
        buffers += 2;
        n_desc -= 2;
14.60   strh   w1, [x29,#360]
            if (n_trace > 0)
              mrvl_pp2_input_trac
```

vlib_physmem_region_t

```
 3.88   add    x1, x0,
11.98   ldrb   w0, [x0,#22]
        add    x0, x7, x0, lsl
14.23   ldrb   w0, [x0,#24]
        umaddl x0, w0, w9, x10
27.81   ldr    x14, [x0,#8]
```

Initial Remarks

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

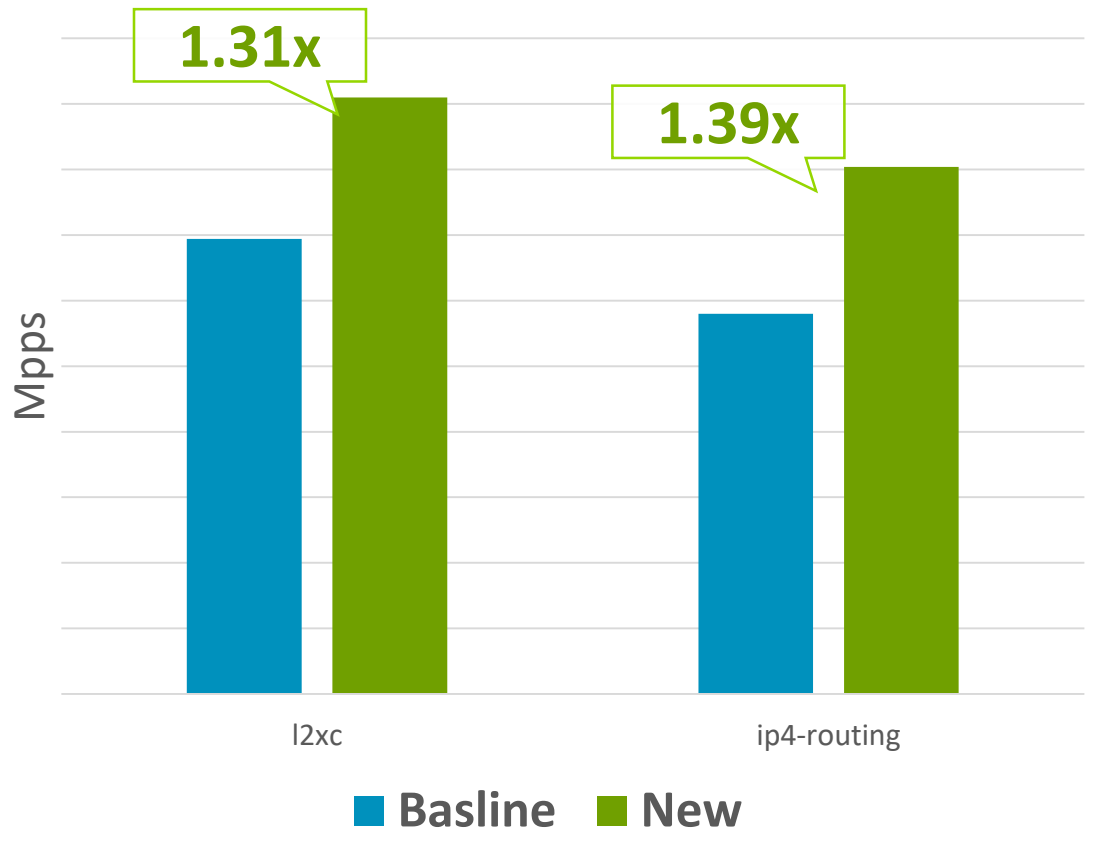
Further Directions

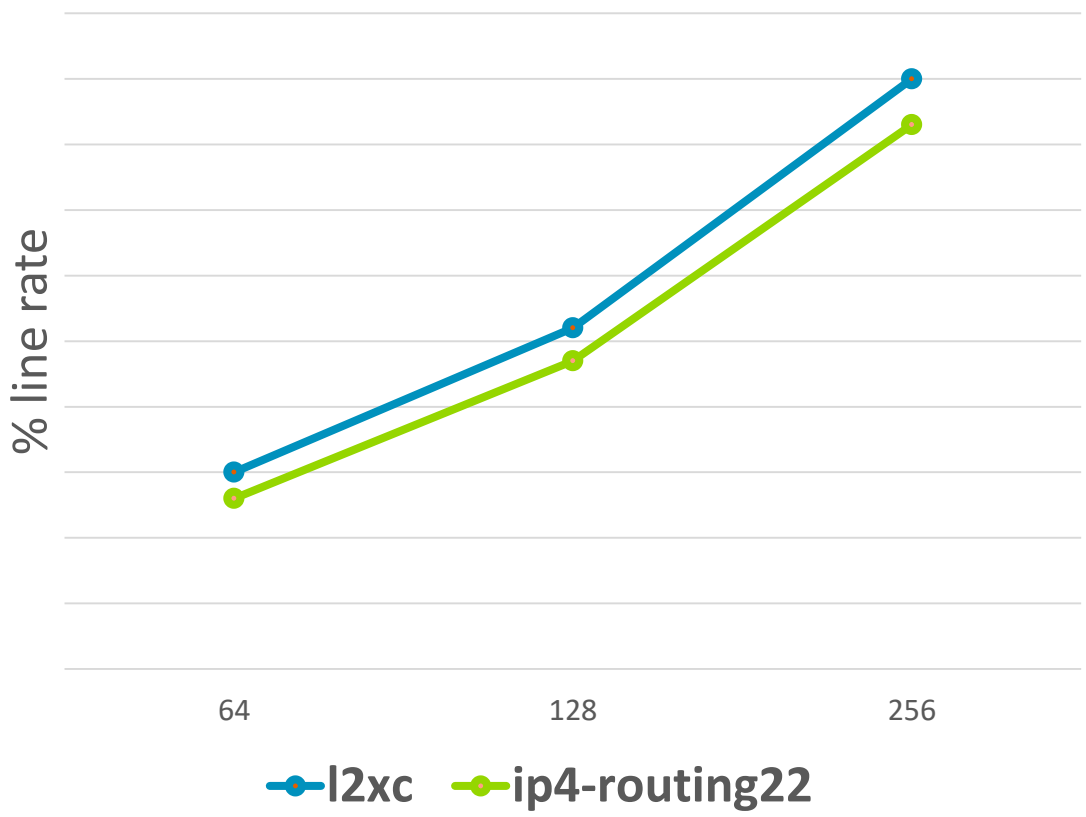
- Leverage PMU data
- Compiler and C library versions
- Multicore scaling
- Platforms
  - Cavium, Huawei, Qualcomm, …

arm

Preliminary Results

64B packet – single flow – single core





The path to FD.io on Arm

Workload Scale	Performance Analysis	Software
CSIT	Hotspot & Bottleneck Identification	Upstream
		Libraries
		OS
		Toolchain
FD.io Lab	Tuning & Optimization	**Hardware**
		Processors
		I/O
		Accelerators

CAVIUM HUAWEI MARVELL

arm

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

arm

```
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

arm