LegUp: Accelerating Memcached on Cloud FPGAs

Xilinx Developer Forum
December 10, 2018

Andrew Canis & Ruolong Lian
LegUp Computing Inc.
COMPUTE IS BECOMING SPECIALIZED

**GPU**
Nvidia graphics cards are being used for floating point computations

**TPU**
Google tensor processing unit used for machine learning

**FPGA**
Reconfigurable hardware. FPGAs excel at real-time data processing.
LEGUP HLS PLATFORM

Software

A Unified Hardware Acceleration Platform

Software

Test/Debug

Hardware

System

Hardware

Vendor

Agnostic

CPU

Intel Xeon processor

XILINX VIRTEX UltraScale+

SDAccel Environment

XILINX

ALL PROGRAMMABLE

LEGUP

Agnostic

LEGUp HLS PLATFORM
The Era of FPGA Cloud Computing is Here

Microsoft accelerates Bing Search with FPGAs

Microsoft rolls out FPGAs in every new datacenter

Nov 2016

July 2017

Baidu, Huawei deploy FPGAs in their cloud

June 2014

Microsoft and Nimbix deploy FPGAs in their cloud

Alibaba and Tencent deploy FPGAs in their cloud

Microsoft and Amazon deploy FPGAs in their cloud

Microsoft and Tencent deploy FPGAs in their cloud

SKT deploys FPGAs for AI acceleration

Aug 2018
CLOUD PLATFORM

- Network processing engines on cloud FPGAs and on-premises FPGA acceleration cards

Input your C/C++ Code

void accelerator (FIFO *input, FIFO *output) {
    int in = fifo_read(input);
    loop: for (int i = 0; i < NUM; i++) {

Hardware Compilation

Cloud or On-prem FPGA Server

Real-Time Data Stream from Network

Real-Time Analysis Output to Network
What is Memcached?

- Memcached is a distributed in-memory key-value store
  - Used as a cache by Facebook, Twitter, Reddit, Youtube, etc
  - Facebook Memcached cluster handles billions of requests per second
- Memcached Commands:
  - Set key value
  - Get key
- Typical deployments:
  - Amazon ElastiCache
  - Google Cloud App Engine
  - Self-hosted
- Easy horizontal scaling:
  - Cluster of Memcached servers handles the load
Introducing: World’s Fastest Cloud-Hosted Memcached

Powered by AWS FPGAs with LegUp’s Platform

- 9X HIGHER REQUESTS/SEC
- 9X LOWER LATENCY
- 10X LOWER TCO
Memcached vs. AWS ElastiCache

- Benchmarked Memcached against AWS ElastiCache
  - AWS provides a fully-managed CPU Memcached service
  - Different instance types based on RAM size, network bandwidth, and hourly cost
  - Chose an ElastiCache instance with the closest specs to F1

<table>
<thead>
<tr>
<th>AWS Instance</th>
<th>vCPUs</th>
<th>RAM</th>
<th>Network Speed</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>cache.r4.4xlarge (CPU)</td>
<td>16</td>
<td>101 GB</td>
<td>Up to 10 Gbps</td>
<td>$1.82/hour</td>
</tr>
<tr>
<td>f1.2xlarge (FPGA)</td>
<td>8</td>
<td>122 GB</td>
<td>Up to 10 Gbps</td>
<td>$1.65/hour</td>
</tr>
</tbody>
</table>
• Memtier_benchmark: Open-source Memcached benchmarking tool
• 100-byte size data, pipelining (batching) of 16
• Varied number of connections to Memcached
Throughput Results

- Up to 9X better ops/sec vs. ElastiCache
Latency Results

• Up to 9X lower latency vs. ElastiCache
Total Cost of Ownership Results

- Up to **10X** better throughput/\$ vs AWS ElastiCache
Where is the speedup from coming from?

1. We accelerated both TCP/IP network and Memcached completely in FPGA
2. Fully pipelined FPGA hardware – new input every clock cycle
3. Multiple Memcached commands in-flight processed in streaming fashion
4. At high packets/sec, software network stack can become a bottleneck
Memcached Demonstration on AWS F1

- Spins up an AWS F1 instance and another client instance
[RUN #1] Preparing benchmark client...
[RUN #1] Launching threads now...
[RUN #1 100%,  8 secs] 0 threads: 10240000 ops, 1245820 (avg: 1237368) ops/sec, 104.44MB/sec (avg: 103.74MB/sec), 13.17 (avg: 13.23) msec latency

32 Threads
32 Connections per thread
100000 Requests per thread

ALL STATS

<table>
<thead>
<tr>
<th>Type</th>
<th>Ops/sec</th>
<th>Hits/sec</th>
<th>Misses/sec</th>
<th>Latency</th>
<th>KB/sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sets</td>
<td>636437.22</td>
<td>---</td>
<td>---</td>
<td>13.23000</td>
<td>88203.36</td>
</tr>
<tr>
<td>Gets</td>
<td>636437.22</td>
<td>636437.22</td>
<td>0.00</td>
<td>13.23000</td>
<td>21878.75</td>
</tr>
<tr>
<td>Waits</td>
<td>0.00</td>
<td>---</td>
<td>---</td>
<td>0.00000</td>
<td>---</td>
</tr>
<tr>
<td>Totals</td>
<td>1272874.45</td>
<td>636437.22</td>
<td>0.00</td>
<td>13.23000</td>
<td>109282.11</td>
</tr>
</tbody>
</table>

Mentier Client: $
Fl_Server -> start_demo

AFI: 0
AFIDevice: 0 0x1d0f 0x7100 0000:00:01:
EAL: Detected 0 lcore(s)
EAL: No free hugepages reported in hugepages-1048576kB
EAL: Probing VFIO support...
EAL: PCI device 0000:00:03.0 on NUMA socket -1
EAL: Invalid NUMA socket, default to 0
EAL: probe driver: 1d0f:ec20 net_ena
EAL: PCI device 0000:00:04.0 on NUMA socket -1
EAL: Invalid NUMA socket, default to 0
EAL: probe driver: 1d0f:ec20 net_ena
PMD: eth_ena_dev_init(): Initializing: 0:0:4:0
Port 0 MAC: f8 59 b3 0d 9e 15
device_file_name=/dev/edma0_queue_0

Core 0 processing packets. [Ctrl+C to quit]
CPU freq = 2300000000 Hz
Processing on core 0
Time interval = 1000000000ns, PPS RX = 0, PPS TX = 0

[RUN #1] Preparing benchmark client...
[RUN #1] Launching threads now...
[RUN #1] 100%, 0 secs 0 threads: 10240000 ops, 0 (avg: 11138885) ops/sec, 0.00KB/sec (avg: 933.91MB/sec), 0.00 (avg: 1.46) msec latency

32 Threads
32 Connections per thread
10000 Requests per thread

ALL STATS

<table>
<thead>
<tr>
<th>Type</th>
<th>Ops/sec</th>
<th>Hits/sec</th>
<th>Misses/sec</th>
<th>Latency</th>
<th>KB/sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sets</td>
<td>5523383.76</td>
<td>5460735.43</td>
<td>62648.33</td>
<td>1.46300</td>
<td>765481.66</td>
</tr>
<tr>
<td>Gets</td>
<td>5523383.76</td>
<td>5460735.43</td>
<td>62648.33</td>
<td>1.46300</td>
<td>182934.04</td>
</tr>
<tr>
<td>Waits</td>
<td>0.00</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
<td>0.00000</td>
</tr>
<tr>
<td>Totals</td>
<td>11046776.53</td>
<td>5460735.43</td>
<td>62648.33</td>
<td>1.46300</td>
<td>948415.70</td>
</tr>
</tbody>
</table>

MEMIERT_CLIENT: -->
AWS Cloud-Deployed FPGAs (F1)

- On F1, the FPGA is not directly connected to the network
- CPU is connected to the network and FPGA is connected over PCIe.
Memcached System Architecture

AWS F1 Instance

- CPU
  - Virtual Network to FPGA (S/W)
- FPGA
  - Virtual Network to FPGA (H/W)
  - TCP/IP Offload Engine
  - Memcached Accelerator
- 64GB DDR4
- 10Gbps Network

PCIe

LegUp
Virtual Network to FPGA (VN2F)

- **VN2F SW**
  - Bypass Linux kernel, send/receive raw network packets, DMA from/to FPGA
- **VN2F HW**
  - Split/combine DMA data to/from individual network packets
  - Each direction takes 20~50us, transfers are overlapped
Network Offload: TCP/IP & UDP

• Supports TCP/UDP/IP network protocols
• 10Gbps ethernet support
• 1000s of TCP connections
• Implemented in C++, synthesized by LegUp
• Can be used by other applications
  • Interface with application via AXI-S

* D. Sidler, et al., Scalable 10Gbps TCP/IP Stack Architecture for Reconfigurable Hardware, in FCCM’15
Memcachad Core

- The Memcached core is fully pipelined with Initiation Interval of 1
  - Request Decoder block decodes the requests and partitions them into key and value pairs.
  - Hash Lookup hashes keys to hash values and looks up the corresponding addresses
  - Values are stored/retrieved to/from the memory by the Value Store block.
  - Response Encoder creates Memcached responses to return to the clients

Network Bandwidth on AWS F1

- The f1.2xlarge instance has an “Up to 10 Gbps” network
- Bottleneck: bandwidth and PPS

Small packets can’t saturate 10 Gbps network

Max PPS is around 700K
Memcached Request Batching

- Batching in Memcached permits packing multiple requests into a single network packet
  - Reduces packet processing overhead
  - Important feature for performance, especially when PPS is the bottleneck

- Batching Adapter splits up aggregated requests into individual requests
  - Sends to Memcached core each request in a pipelined fashion

**Batching is not needed for on-premise FPGAs**

<table>
<thead>
<tr>
<th>Without Pipelining</th>
<th>PKT1</th>
<th>PKT2</th>
<th>PKT3</th>
<th>PKT4</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQ1</td>
<td>REQ2</td>
<td>REQ3</td>
<td>REQ4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>With Pipelining</th>
<th>PKT1</th>
<th>PKT2</th>
<th>PKT3</th>
<th>PKT4</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQ1</td>
<td>REQ2</td>
<td>REQ3</td>
<td>REQ4</td>
<td>REQ5</td>
</tr>
</tbody>
</table>
Streaming b/w Host & Kernel in SDAccel

Host

PCIe

PCIS AXI-4

WR channel

RD channel

BAR1 AXI-L (read & write)

Custom Logic on FPGA

Streaming Kernel

MM2S

S2MM

RX FIFO

TX FIFO

data count

AXI-S

AXI-S

HDK Shell Interface

LegUp
Streaming b/w Host & Kernel in SDAccel

- Host
  - PCIe
  - AXI-4
  - BAR1 AXI-L (read & write)

- SDAccel Environment
  - Kernel Interface
  - WR ch.: AXI-4
  - DDR Memory
  - RD char.: AXI-4

- Custom Logic on FPGA
  - RX FIFO
  - TX FIFO (data count)

- Streaming Kernel
  - MM2S
  - AXI-S
  - S2MM
  - AXI-S
Streaming b/w Host & Kernel in SDAccel
Streaming b/w Host & Kernel in SDAccel

Host
- PCIe
- PCIS AXI-4
- BAR1 AXI-L (read & write)

Kernel
- SDAccel Environment
- Kernel Interface
- DDR Memory
- S2MM

Custom Logic on FPGA
- RX FIFO
- TX FIFO

Streaming Kernel

LegUp
Streaming b/w Host & Kernel in SDAccel

Send transfer size to the AXI-S pipe when,
• Enough data has been accumulated
• No new incoming data for X cycles
Come talk to us about:

- Memcached Acceleration
- FPGA Network Stack
- SDAccel Streaming Handler
- LegUp high-level synthesis tool
- Any other FPGA acceleration needs

Andrew Canis & Ruolong Lian
www.LegUpComputing.com
info@legupcomputing.com | 647-834-6654 | Toronto, Canada