Convey’s Acceleration of the Memcached and Imagemagick Applications

CARL – June 14, 2015
Tony Brewer
tbrewer@micron.com
Target Platforms

- **Wolverine**
  - Full size PCIe form factor
  - 75W maximum power
  - Up to 4 channels of DDR3 memory

- **Merlin**
  - ½ height, ½ length PCIe form factor
  - 75W maximum power
  - 1 HMC 4GB memory, 2 16-bit channels
Wolverine Architecture

- GPU Form Factor and power envelope
- Single and Dual card versions

x86-64 Server
- CentOS
- Choice of processors, I/O devices, chassis form factor, memory size,

Convey coprocessor
- FPGA-based
- High-bandwidth memory
- Single or Dual card versions
• **Storage is the faster growing piece of IT spend**
  – More and more images are being uploaded/stored
  – Images accumulate over time and with growing users

• **Resizing images takes time**
  – jpegs must first be expanded
  – Expanded image scaled to desired size and recompressed

• **Common approach has been to store multiple resized version of an image**
  – Consumes up to 30% of social network storage
  – Pre-computed “thumbnails” not always optimal for display at target page of customer/consumer

• **What if I could resize images on the fly?**
  – For a small increment to the IT budget
  – Non-accelerated solution requires 48x more image servers $$$$$$
  – Dell-Convey solution delivers on all requirements
  – And save 30% on storage

---

**Image Resizing – 48x speedup**

**Social Network Example**
#### Jpeg Resize Implementation

**x86 Host**
- reads arguments and builds queue of images to be scaled
- worker threads read input files and write output files

**coprocessor**
- Hardware threads decode, resize, and encode images
- data transferred via memory

---

- **jobsFile.txt**
- **input.jpeg**
  - **job queue**
  - **main thread**
  - **worker thread**
    - **queue**
    - **decode**
    - **vert. scaling**
    - **encode**
- **output.jpeg**

- Host
- Co-processor

- **input image queue**
- **result image queue**
Jpeg Resize Application

• Host objective is to keep the FPGA busy
• Multiple simultaneous jobs are required
  – ~6 jobs at the FPGA are required, 4 active + 2 to cover queuing latencies and job overlap
• A host thread is used to handle each resize job
  – Read Jpeg image from disk or the network
  – Process Jpeg header and construct job control structure
  – Start job on FPGA
  – Write resized image to disk or network
Jpeg Resize Application

• Many host threads are used
  – 10-15 host threads are needed to keep 6 jobs active on the FPGA

• Application uses a client / server model
  – Client library is compiled into an application that needs resizing capability
  – Server processes jobs submitted to it
  – Client and Server can be on same platform or across a network with potentially multiple clients

• Initial client is the Imagemagick application
A Host Thread processes the JPEG image header and creates a job control structure (up to 200KB in size)

```c
struct JobInfo {
    JobApp m_app;  // APP marker info
    JobDec m_dec;  // jpeg decode info
    JobHorz m_horz; // horizontal scaling info
    JobVert m_vert; // vertical scaling info
    JobEnc m_enc;   // jpeg encode info
    JobCom m_com;   // COM marker info

    uint8_t * m_pInPic; // input picture
    uint32_t m_inPicSize;
};
```

Using Sinc Func as Resize Filter

```
struct _ALIGNED(64) JobHorz {
    uint64_t m_compCnt : 2;
    uint64_t m_inImageRows : 14;
    uint64_t m_inImageCols : 14;
    uint64_t m_outImageRows : 14;
    uint64_t m_outImageCols : 14;
    uint64_t m_maxBlkColsPerMcu : 2;
    uint64_t m_maxBlkRowsPerMcu : 2;
    uint64_t _ALIGNED(8) m_mcuRows : 11;
    uint64_t m_mcuCols : 11;
    uint64_t m_mcuBlkRowCnt : 3;
    uint64_t m_mcuRowRstInc : 4;
    JobHcp m_hcp[MAX_MCU_COMPONENTS];
    uint16_t m_filterWidth;
    uint16_t m_pntWghtListSize;
};
```

```
JobPntInfo _ALIGNED(64)
    m_pntInfo[COPROC_MAX_IMAGE_PNTS];
JobPntWght _ALIGNED(64)
    m_pntWghtList[COPROC_MAX_IMAGE_PNTS];
};
```
A job is submitted to the FPGA as
  - A pointer to the job control structure
  - The job control structure has a pointer to both the input and output image host memory

FPGA performs image resizing
  - Output is a resized JPEG image without the header
  - Output is written by the FPGA to host memory
  - FPGA completes the jobs by returning to host

Host completes the jobs by
  - Constructing the JPEG file including a header and the image data
  - Writing the JPEG file to disk or the network
void JpegResizeHif::SubmitJob(JobInfo * pJobInfo) {
    // multi-threaded job submission routine
    ObtainLock();

    // obtain job ID
    while (m_jobIdQue.empty()) {
        ReleaseLock();
        usleep(1);
        ObtainLock();
    }

    uint8_t jobId = m_jobIdQue.front();
    m_jobIdQue.pop();

    // clear job finished flag
    m_bJobDoneVec[jobId] = false;
    m_bJobBusyVec[jobId] = true;

    // send job to coproc
    while (!m_pUnit->SendCall_htmain(jobId, (uint64_t)pJobInfo)) {
        ReleaseLock();
        usleep(1);
        ObtainLock();
    }

    // wait for job to finish
    while (m_bJobDoneVec[jobId] == false) {
        uint8_t recvJobId;
        if (m_pUnit->RecvReturn_htmain(recvJobId)) {
            m_bJobDoneVec[recvJobId] = true;
        } else {
            ReleaseLock();
            usleep(1);
            ObtainLock();
        }
    }

    // free jobId
    m_bJobBusyVec[jobId] = false;
    m_jobIdQue.push(jobId);
    ReleaseLock();
}

• SubmitJob routine handles interface to coprocessor
  – Multi-threaded, one thread per resizing job
  – Up to eight resize jobs are active in coprocessor
  – SendCall_htmain() performs a remote procedure call (RPC) to start job
  – RecvReturn_htmain() is used to poll on completed jobs
Detailed Diagram of Personality

HT Diagram for Image
Scaling Personality
Jpeg Resize Performance

Dell-Convey Accelerated Image Resizing Solution

- Characterization of Resize App.
  - Sweep across
    - Resize %
    - Image Size
  - Sweet spot is in the 25-75% resize range with increased performance as the image gets larger
# Total Cost of Ownership

**Dell-Convey Accelerated Image Resizing Solution**

### Performance

<table>
<thead>
<tr>
<th></th>
<th>r720 + Wolverine 690 ≈ 48x vs. One Socket, 4-core 3.3 GHz</th>
</tr>
</thead>
</table>

### Power

<table>
<thead>
<tr>
<th>Requirements[1]</th>
<th>Dell-Convey Accelerated Server</th>
<th>50 racks (1000 nodes) Dell-Convey</th>
<th>850 racks (34000 nodes) x86</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2,190 MW-h/yr</td>
<td>59,568 MW-h/yr</td>
<td></td>
</tr>
</tbody>
</table>

1 Year Electricity costs ( @ 0.08 /kWh)[2]

<table>
<thead>
<tr>
<th>System</th>
<th>Dell-Convey Accelerated Server</th>
<th>x86</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Year costs</td>
<td>315 K$/yr</td>
<td>8,578 K$/yr</td>
</tr>
</tbody>
</table>

### Site

<table>
<thead>
<tr>
<th></th>
<th>Dell-Convey Accelerated Server</th>
<th>x86</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Year Infrastructure costs[3]</td>
<td>52 K$/yr</td>
<td>2,098 K$/yr</td>
</tr>
</tbody>
</table>

### TCO

<table>
<thead>
<tr>
<th></th>
<th>Dell-Convey Accelerated Server</th>
<th>x86</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-Year TCO[4]</td>
<td>11,073 K$</td>
<td>63,106 K$</td>
</tr>
</tbody>
</table>

### TCO Multiplier

- Dell-Convey Accelerated Server: 1.0
- x86: 5.7

---

### 3-Year Cost of Ownership

- 3 yr UPS+Floorspace Costs (K$)
- 3 yr Datacenter power costs (K$)
- System cost (K$)

---

June 14, 2015
Memcached Appliance

Key / Value Cache Solution

Who uses memcached?
- 85% of top 20 web sites
- 50% of top 5,000 sites
- Up to 30% of data center space
Why Is Performance Important?

How many accesses are required to create this page? ~100

Modern web services must operate at RAM speeds to retrieve the amount of data needed with acceptable responsiveness.
Achieving Performance

Cnymemcached based on memcached 1.4.15

Client connections are assigned to threads

memcached_set("foo:key",data)
memcached_get("foo:key")

Host executes commands and generates replies

Coprocessor offloads parsing of commands and key hashing

memcached_set("foo:key",data)
memcached_get("foo:key")

Parse request
Hash key(s)
Execute command(s)
Send reply
Parse request
Hash key(s)

Network stack
Object store
Hash table

June 14, 2015
Memcached Application

- Drop in replacement for original application
  - Does not support vendor specific commands
- All key / value pairs are stored in host memory
  - Memcached servers require large amounts of memory
  - Host memory is typically lower cost than PCIe card memory
- Supports TCP and UDP network protocols
- Support ASCII and binary commands
- Large amounts of host code was modified or replaced
  - Restructured for FPGA acceleration
  - Original code did not support binary mode aggregation of outbound responses to same destination
Memcached HW/SW Interface

- **Linux network stack is used without modification**
  - Standard network stack has limited packet receive / send rate that limited application performance
  - Added out bound packet aggregation prior to calling send system call
  - Experimented with Solarflare NIC cards. Cards reduced latency and improved throughput but not enough for extra cost for system

- **Packets are read into host memory**
  - A pointer to the packet, the packets length and port number is passed to FPGA in a host memory FIFO
• FPGA processing
  – Received network packets are reassembled into complete commands
  – Commands are parsed (binary or ASCII)
  – Key is hashed
  – Response to host is a command structure with command and parameters, including hashed key

• Host executes command using hashed key and formats output packets
  – Output packets are queued per destination for short period of time to determine if additional packets can be concatenated.
Convey throughput is highest with 20 or more gets per request
- request processing dominates
Maximum throughput is ~13M gets/second
- 48 gets per network packet
Throughput Scaling

Cnymemcached Throughput
as a function of number of server threads
r720 20x2.8GHz, 2x10GbE (X520), Wolverine 690

100% hits
100% misses
Load/Latency

8-byte keys, 32-byte objects

Convey Accelerated Memcached Load-Latency
Dell r720 20 cores@2.8Ghz, Wolverine 690

Throughput (gets per second) vs. latency (microseconds)

Wolverine cnymemcached  x86 memcached

June 14, 2015
Convey was acquired by Micron, customers have asked the future direction of Convey’s hardware and software products

- Micron is continuing to sell and support Convey’s products
- Micron values the partnerships that Convey established with Dell and IBM (CAPI)
- As expected, Micron is focused on large opportunities
• Convey’s customers consistently asked to have the HT tool set open sourced
  – For a customer to invest in FPGA systems they needed to have access to the development tools
  – Convey’s real IP is the ported FPGA applications
  – Convey decided that we could support this business model
  – Transition from license model to support model

• Convey has open sourced HT
  – OpenHT is available on github
  – Includes the tool set plus example applications: Jpeg resizing, Memcached, Bit Coin Mining, Graph 500 Benchmark, and Levinstein
Convey Development Tools

Convey Hybrid Threading

C/C++
OpenMP

Hybrid Threading Compiler

HT Linker & Verilog Generation

Xilinx OpenCL

User Hybrid Threading Code

User VHDL/Verilog

Personality Development Kit
Verilog/VHDL-to-RTL, FPGA place, route, timing

OpenHT
• HTL (runtime generator) and HTV (systemC to verilog)
  – Current stable release is tagged 1.4
  – Considerable work has gone into top-of-tree
    • Replace original memory interface and global variable runtime support
    • Continuing to add support for C++ constructs (overloading, user defined operators)
  – Once stable, will be tagged 2.0
    • Finding remaining issues with randomly generated tests
    • Porting applications and identifying performance / resource regressions

• HTC (OpenMP translator)
  – Considered a prototype and unlikely that Convey/Micron will productize HTC
  – Translation capability is fairly robust
    • A number of tests were released with OpenHT including Graph 500
    • Support a subset of OpenMP 4.0 language
  – Missing runtime libraries to make it fully usable
    • Standard C library (libc)
    • Math libraries
Thank You

Tony Brewer
tbrewer@micron.com