DESIGNING HIGH-PERFORMANCE IN-MEMORY KEY-VALUE OPERATIONS WITH PERSISTENT GPU KERNELS AND OPENSHMEM

Ching-Hsiang Chu (The Ohio State University, NVIDIA), Sreeram Potluri (NVIDIA), Anshuman Goswami (NVIDIA), Manjunath Gorentla Venkata (Oak Ridge National Laboratory), Neena Imam (Oak Ridge National Laboratory), and Chris J. Newburn (NVIDIA)
OUTLINE

- Introduction
- Problem Statement & Challenges
- Background
- Proposed Solutions
- Evaluation
- Conclusion
HUGE DATA EXPLOSION

Predicted to continue

“Annual global IP traffic will reach 3.3 ZB (Zetta-bytes; 1000 Exa-bytes [EB]) by 2021”

source: Cisco Visual Networking Index: Forecast and Methodology, 2016-2021

All kinds of indexed traffic

Video, Search, Maps...

How to provide efficient access to such a large volume?

Cache!

IN-MEMORY KEY-VALUE STORES

What is it?

- In-Memory Key-Value (IMKV) Stores: Distributed in-memory data caching systems
  - Caching data and objects in system memory, i.e., RAM (Random Access Memory)
  - Querying the cache server(s) before retrieving data from database
  - Speeding up web applications by shortening query/response path

1. Cache access requests/responses
2. Cache miss: Fetching objects from DB
IN-MEMORY KEY-VALUE STORES (CONT’)

Why it becomes popular?

- Faster for I/O-intensive applications (especially, read-heavy workloads)
  - System memory (micro/nano-seconds) >>> database with persistent storage (milliseconds↑)
- Resources are available: Large system memory now common on cloud and HPC systems
  - Amazon EC2 X1e provides up to 4TB DDR4 Memory (https://aws.amazon.com/ec2/instance-types/)
  - Amazon ElastiCache offers Memcached/Redis-ready nodes equipped with up to 400 GB memory (https://aws.amazon.com/elasticache/
  - Summit (#1 supercomputer): 512GB DDR4 per node
IN-MEMORY KEY-VALUE STORES (CONT’)

How does it work?

- Fit hash table on system memory
  - Keys: 8Bytes - 250Bytes (for memcached)
  - Values: 8Bytes - 1Mbytes data objects (for memcached)
- Basic Key-Value (KV) operations: GET (the most commonly used) & SET

Diagram:
- Web Servers
- IMKV servers
- Back-end Database
- Network
- GET(Key) / SET(Key, Value)
- Cache miss: Fetching objects from DB
# LET’S MAKE IMKV SERVER FASTER

State-of-the-arts *(NOT a comparison)*

<table>
<thead>
<tr>
<th>KVS</th>
<th>Highlights</th>
<th>Reported Peak throughput (MOPS)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Key HW</td>
</tr>
<tr>
<td><strong>Memcached</strong></td>
<td>Traditional CPU based solutions, UDP</td>
<td>Intel 8-core Xeon CPU, One 40-GbE NIC</td>
</tr>
<tr>
<td><strong>MICA</strong></td>
<td>Kernel bypass, Intel DPDK</td>
<td>24-core CPU, 12-port NIC</td>
</tr>
<tr>
<td><strong>MemcachedGPU</strong></td>
<td>Offload packet parsing (UDP) and KV operations to GPU, GPU Direct</td>
<td>One Nvidia Tesla K20c, 10-GbE NIC</td>
</tr>
<tr>
<td><strong>Mega-KV</strong></td>
<td>Offload only indexing operations to GPU, Intel DPDK (UDP)</td>
<td>Singe Node: Dual socket Intel Xeon 8-core CPUs, one Nvidia Tesla K40c GPU, one 40-GbE NIC per socket 4 nodes: Same config as above</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SHMEMCache</strong></td>
<td>OpenSHMEM, RDMA, One-sided KV operations (CPU bypass)</td>
<td>10-core Intel Xeon CPU, IB ConnectX3 NIC 1024 nodes on Titan cluster</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>KV-Direct</strong></td>
<td>CPU Bypass, Offload KV operations to Programmable NIC (FPGA)</td>
<td>Both are on a single server: One 40-GbE NIC, two PCIe Gen3 x8 link Ten 40-GbE NICs, one PCIe Gen3 x8 link per NIC</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


GPU ACCELERATED IMKV SERVER

How Mega-KV works?

- KV operations are memory-intensive operations
  - GPU with HBM is a perfect fit!
  - But, GPU memory is not as BIG as system memory
- Mega-KV: Cache KVs in SysMem; Index KV on GPU
  - Introduce a new hash table in GPU memory
    - Compressed, fixed-width key
    - Value: location (in system memory)
  - Offload only indexing operations to GPUs
    - CPU threads complete KV operations

GPU ACCELERATED IMKV SERVER (CONT’)

Can we do better? What is still missing in Mega-KV?

• Batching is required to achieve high-throughput
  • Data copy and CUDA kernel launch dominate indexing processes

→ longer response time for the small requests

Small batch (1,000 keys): Data copy & kernel launch take longer than actual work

<table>
<thead>
<tr>
<th>Streams</th>
<th>cudaMemcpyHostToDevice + kernel + cudaMemcpyDeviceToHost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td></td>
</tr>
<tr>
<td>Stream 17</td>
<td></td>
</tr>
<tr>
<td>Stream 18</td>
<td></td>
</tr>
<tr>
<td>Stream 19</td>
<td></td>
</tr>
<tr>
<td>Stream 20</td>
<td></td>
</tr>
</tbody>
</table>

Large batch (50,000 keys): Multiple CUDA streams help overlap data copies and kernels, but underutilization still present...
OUTLINE

- Introduction
- Problem Statement & Challenges
- Background
- Proposed Solutions
- Background
- Evaluation
- Conclusion
PROBLEM STATEMENT

Challenges

• Can we further improve the throughput of GPU-based IMKV server?
  • How to optimize data movement?

• Can we enable high-throughput indexing operations on GPU even without batching large amount of input requests?
  • How to reduce overhead when offloading operations to GPU?
  • How to maintain high GPU utilization?
OUTLINE

- Introduction
- Problem Statement & Challenges
- Background
- Proposed Solutions
- Evaluation
- Conclusion
GPUDIRECT RDMA
Remote Direct Memory Access

• Introduced in CUDA 5, available on all Tesla GPUs starting from Kepler
  • Direct transfers between GPUs and NICs over PCIE
  • Eliminate CPU bandwidth and latency bottlenecks

https://developer.nvidia.com/gpudirect
OUTLINE

- Introduction
- Problem Statement & Challenges
- Background
- Proposed Solutions
- Evaluation
- Conclusion
OVERVIEW

Challenges and Proposed Solutions

- Avoid copy engine overheads
  - SM-based copy (fused with processing kernel)
- Avoid copies from System Memory to Device Memory
  - Direct access to pinned system memory
  - Direct write to device memory using GPUDirect RDMA (GDR) through NVSHMEM
- Avoid kernel launch overheads
  - Persistent CUDA kernel(s)
AVOIDING COPIES
Data and control paths

1. Set/Get requests from clients
2.a. Schedules memory copies
2.b. Schedules GPU Kernel(s)
3. Performs indexing operations
   Direct access pinned SysMem or SM-based copy
4. Copies result back to System Memory
5. Post-processing results and reply to client if required

Put data directly to GPU memory using GDR technology through NVSHMEM
PERSISTENT CUDA KERNELS
Synchronization and Resource Use

• CPU thread(s) launch a persistent CUDA kernel when initializing the server
• CPU thread(s) assign and signal threadblocks/CTAs for new requests
• CTAs perform indexing operations and signal back upon completion
• CPU thread(s) check the completion and perform post-processing

• Key design considerations
  • Low overhead signal mechanism
  • Efficient resource utilization
SIGNALING METHOD
How persistent kernel knows when and where to start processing?

• What to signal?
  • From CPU to a particular CTA, that new work is assigned
  • Size of the assigned work
  • Pointers of input and output buffers

• How to signal?
  • Two counters and a work queue per CTA
  • issued_seqnum (is): CPU updates, CTA polls
  • completed_seqnum (cs): CTA updates, CPU polls
SIGNALING METHOD (CONT’)

How persistent kernel knows when and where to start processing?

• Where to store the counters?
  • Pinned System Memory
    • Fast for CPU, slow for GPU to update and poll
  • Device Memory
    • Fast for GPU to poll and update but extra copies are required for CPU
    • Exploits low-latency GDRCopy\cite{10} library which enables fast CPU update
    • Issued_seqnum on system memory or GPU memory; completed_seqnum always on system memory

\cite{10} https://github.com/NVIDIA/gdrcopy
SIGNALING METHOD (CONT’)
Signal overhead on Tesla P100 with 56 SMs

- Padding (128 B)
- No padding
- GDRCopy-Sync
PERSISTENT KERNEL + GDR

Data and control paths

1.a Send SET/GET requests directly to GPU memory
1.b Client notifies server
2 CPU signals available CTAs
3 Performs indexing operations
4 Copies result back to System Memory
   (*SM-based or Copy Engine)
5 Post-processing results and reply to client if required
BENEFITS OF PERSISTENT KERNELS + GDR
Better GPU Utilization & performance

- Non-persistent kernel, without GDR (i.e., Mega-KV)

   Timeline

   CPU
   - Issues copy
   - Issues kernel
   - Issues copy
   - Issues kernel
   - Issues copy
   - Issues copy

   CTAs 0-7
   - cudaMemcpyH2D
   - Kernel
   - cudaMemcpyD2H

   CTAs 8-15
   - cudaMemcpyH2D
   - Kernel
   - cudaMemcpyD2H

- Persistent kernel, with GDR

   Timeline

   CPU
   - Signaling
   - Signaling

   CTAs 0-7
   - Kernel (indexing + D2H Copy)

   CTAs 8-15
   - Kernel (indexing + D2H Copy)

Note: Assume processing two small batches of 1,000 keys & 8 CTAs required each batch
OUTLINE

- Introduction
- Problem Statement & Challenges
- Background
- Proposed Solutions
- Evaluation
- Conclusion
EXPERIMENTAL ENVIRONMENT

Pascal P100 GPU

- Intel Xeon dual-socket 16-core Haswell (E5-2698) CPU 2.3 GHz, 256 GB Host Memory
- InfiniBand Connect-IB (MT27600) FDR (56Gbps)
- NVIDIA Tesla P100 GPUs connected with PCIE Gen 3
  - 16 GB DDR5 memory, 56 SMs
  - CUDA 9.0.176
- Benchmark
  - Adopted from Mega-KV (http://kay21s.github.io/megakv/)
  - Metric: Million search Operations Per Second (MOPS)
    - i.e., throughput, higher is better
BASELINE
Mega-KV on P100: Tuning # of CUDA Streams

1 CUDA stream performs best for small batches

2/4 CUDA streams performs best for large batches
AVOID COPIES

NVSHMEM with GDR improves throughput!
Choosing the appropriate copy back mechanism!
GDR, PERSISTENT KERNEL
vs. MegaKV

PEAK throughput: ~888 MOPS

Throughput (MOPS)

Batch Size (# of Keys)

- Mega-KV-TUNED
- G-IMKV-GDR-CE
- G-IMKV-PK-GDR-OPT

4.8x
OUTLINE

- Introduction
- Problem Statement & Challenges
- Background
- Proposed Solutions
- Evaluation
- Conclusion
CONCLUSION & FUTURE WORK

Key takeaways

• Use of GPUDirect RDMA eliminates data transfer and copy command overhead
  • 1.2x speedup for large batches and 4.8x speedup for small batches compared to Mega-KV
• Persistent CUDA kernel mitigate kernel launch overhead
  • 3.7x higher throughput for small batch sizes compared to Mega-KV
• Future Work & Open issue
  • Evaluate scalability: IMKV server(s) with multiple GPUs and NICs
  • Extend to other streaming applications
  • Investigate CUDA level guarantees required for concurrent progress of persistent kernels and memory copies