OpenCL-Based Erasure Coding on Heterogeneous Architectures

Guoyang Chen, Huiyang Zhou, Xipeng Shen, (North Carolina State University)
Josh Gahm, Narayan Venkat, Skip Booth, John Marshall (Cisco Systems, Inc)
Email: gchen11@ncsu.edu
Introduction

• A key challenge in storage system
  - Failure (disk sector, entire disk, storage site)

• A Solution:
  - Erasure Coding
    - Intel’s intelligent storage acceleration library (ISA-L)
Motivation

• Erasure Coding
  o Replication. (simple, high cost, low toleration)
  o Reed-Solomon coding. (less cost, high toleration, complex)
  o ......

• Motivation:
  o To explore using various heterogeneous architectures to accelerate Reed-Solomon coding.
Reed-Solomon Coding

- Block-based Parity Encoding
  - Inputs are partitioned into ‘srcs’ blocks, with a block size of ‘length’ bytes.
  - Encode matrix: \( \text{dests} > \text{srcs} \)

\[
\text{Dest} = V \times \text{Src}
\]
Reed-Solomon Coding

- Block-based Parity Encoding
  - Inputs are partitioned into ‘srcs’ blocks, with a block size of ‘length’ bytes.
  - Encode matrix: dests > srcs

\[
Dest[l][i] = \sum_{j=0}^{srcs-1} V[l][j] \times Src[j][i]
\]
Reed-Solomon Coding

- Block-based Parity Encoding
  - Inputs are partitioned into ‘srcs’ blocks, with a block size of ‘length’ bytes.
  - Encode matrix: dests > srcs

- sum: 8-bit XOR operation; mul: GF($2^8$) multiplication

\[
Dest[l][i] = \sum_{j=0}^{srcs-1} V[l][j] \times Src[j][i]
\]
GF(2⁸) multiplication

- 3 Ways for Galois Field Multiplication:
  - Russian Peasant Algorithm: pure logic operations.
  - 2 small tables: 256 bytes per table, 3 table lookups, 3 logic operations.
  - 1 large table: 256*256 bytes, no logic operations, one lookup

Refer to paper for details.
Reed-solomon Coding On CPUs

- Intel ISA-L.
  - Single thread.
  - Baseline.

- Adding Multithreading support.
  - Partition input matrix in a column-wise manner.

\[
\begin{align*}
\text{Dest} & = V \cdot X \\
\text{T1} & \quad \text{T2} \\
\text{Tn} & \quad \text{Src}
\end{align*}
\]
Reed-solomon Coding
On GPUs

• Computation for one element in output matrix is independent from others.

• Fine-grain parallelization
  o Each workitem for one byte in output matrix. (Baseline)

• Optimizations???
Reed-solomon Coding On GPUs-Opt(A)

- A. Optimize GPU Memory Bandwidth.
  - Memory coalescing (workitems in one group access data in the same row).
  - Vectorization (reads uint4 one time) ==> higher bandwidth.
    - Each workitem for 16 bytes data.
Reed-solomon Coding On GPUs-Opt(B)

- B. Overcoming Memory Bandwidth Limit Using Texture Caches, Tiling.
  - Workitems in the same row share same value in $V$.
  - $\Rightarrow$ Putting encode matrix and large look up table (64KB, for $GF(2^8)$ Multiplication) in texture cache.

\[
\text{Dest} = V \times \text{Src}
\]
Reed-solomon Coding On GPUs-Opt(B)

B. Overcoming Memory Bandwidth Limit Using Texture Caches, Tiling.

- Workitems in the same row share same value in V.
  ==> Putting encode matrix and large look up table (64KB, for GF(2^8) Multiplication) in texture cache.
- Src in texture cache by using tiling (like MM).
  * Not helpful. Bottleneck: computation bound

\[ \text{Dest} = V \times \text{Src} \]
Reed-solomon Coding
On GPUs-Opt(C)

- C. Hiding Data Transmission Latency Over PCIe
  - Partition input into multiple groups.
    - One stream for one group
  - Hide data copy time with computation time.

Stream 1: H2D | Compute | D2H

Stream 2: H2D | Compute | D2H

Stream 3: H2D | Compute | D2H

......

Stream N: H2D | Compute | D2H
Reed-solomon Coding On GPUs-Opt(D)

- D. Shared virtual memory to eliminate memory copying
  - Shared virtual memory (SVM) is supported in OpenCL 2.0
    - AMD APUs.
    - No need for data copy.
Reed-solomon Coding On FPGAs

• FPGAs
  o Abundant on-chip logics for computation.
  o Pipelined parallelism instead of data parallelism on GPU.
  o Relatively low memory access bandwidth

• Reed-solomon Coding
  o Computation bound
  o A good candidate for FPGAs
  o Same baseline code as used on GPUs. (1 workitem for 1 byte)
Reed-solomon Coding
On FPGAs-Opt(A)

• A. Vectorization to Optimize FPGA Memory Bandwidth
  o One workitem reads 64 bytes from input.
Reed-solomon Coding On FPGAs-Opt(B)

• B. Overcoming memory bandwidth limit using tiling.
  - Load a tile from input matrix to local memory shared by workgroup.
  - A large tile size results in high data reuse and reduces off-chip memory bandwidth
Reed-solomon Coding
On FPGAs-Opt(C)

• C. Unroll loop and Kernel replication to fully utilize FPGA logic resources.
  o __attribute__(num_compute_units(n)): n pipelines.
  o Loop unroll: deeper pipeline.
Experiments

- Input: 836.9MB file.
- On CPU: Intel(R) Xeon(R) CPU E5-2697 v3 (28 cores)
- On GPU: NVIDIA K40m, CUDA7.0; AMD Carrizo.
On CPU

- $\text{srcs} = 30$, $\text{dests} = 33$

![Graph showing Encode Bandwidth vs. number of threads]

- Encode Bandwidth at $2.84$ GB/s with $56$ number of threads.
On NVIDIA K40m

- One Stream:
  - Best: large table (2.15GB/s)
- 8 Streams: == 3.9GB/s

Encode Bandwidth

![Graph showing encode bandwidth for different stream types and datasets.](image)
On AMD Carrizo SVM

- Not as good as streaming.
  - Texture cache doesn't work well.
  - Overhead of blocking functions to map and unmap SVM buffers.
• DMA read/write about 3GB/s.
• Only focus on kernel throughput.
• Assume DMA engine can be easily increased.

On FPGA

Encode Bandwidth

<table>
<thead>
<tr>
<th></th>
<th>GB/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large Table</td>
<td>char</td>
</tr>
<tr>
<td></td>
<td>int</td>
</tr>
<tr>
<td></td>
<td>int16+tiling</td>
</tr>
<tr>
<td></td>
<td>int16+tiling+unroll</td>
</tr>
<tr>
<td>Small Table</td>
<td>char</td>
</tr>
<tr>
<td></td>
<td>int</td>
</tr>
<tr>
<td></td>
<td>int16+tiling</td>
</tr>
<tr>
<td></td>
<td>int16+tiling+unroll</td>
</tr>
<tr>
<td>Russian Peasant</td>
<td>char</td>
</tr>
<tr>
<td></td>
<td>int</td>
</tr>
<tr>
<td></td>
<td>int16</td>
</tr>
<tr>
<td></td>
<td>int16+tiling+unroll</td>
</tr>
</tbody>
</table>
Overall

• Considering the price, FPGA platform is most promising but needs to improve its current PCIe DMA interface.

dests = srcs + 3
file 1 has a size of 29MB; file 2 has a size of 438MB
BDW: Integrated FPGA (arria 10) on Xeon core.
SVM (Shared Virtual Memory): the Map/unMap overhead is included
Arria 10: discrete FPGA board through PCIe.
Stratix V: discrete FPGA board through PCIe.
Conclusions

• Explore different computing devices for erasure codes.
• Different optimizations for different devices.
• FPGA is the most promising device for erasure codes.