A Study of Data Partitioning on OpenCL-based FPGAs

Zeke Wang (NTU Singapore), Bingsheng He (NTU Singapore), Wei Zhang (HKUST)
Outline

• Background and Motivations
  – Data Partitioning on FPGA
  – OpenCL on FPGA
• Design
• Experiment
• Conclusion
What is Data Partitioning?

• Data partitioning divides the input table (of tuples) into a number of partitions according to input partitioning function.
  – It splits the big input table into many small sub-tables (divide-and-conquer manner).
  – It is a building block in many database applications (e.g., hash join and aggregation).
What is Data Partitioning?

It is a memory intensive operation.
Benchmarking Memory Subsystem

1. Sequential bandwidth > Random bandwidth
Benchmarking Memory Subsystem

1, Sequential bandwidth > Random bandwidth

2, Random memory access is more sensitive to data access type.

Use Long8, not byte
Outline

• Background and Motivations
  – Data Partitioning on FPGA
  – OpenCL on FPGA

• Design

• Experiment

• Conclusion
What is OpenCL?

- OpenCL has been developed for heterogeneous computing environments, e.g. CPU+GPU/FPGA, with a host-accelerator model of program execution.
Global memory: external DDR.
Local memory: on-chip memory blocks.
Pipeline: DSP blocks, memory blocks and logic blocks.
OmniDB on FPGA

• OmniDB [1]: State-of-the-art OpenCL-based query processor on CPU/GPU.
  – Mature.
  – Good performance.

• How OmniDB performs on FPGA?
  Lock overhead

Why Lock is Required?

Input tuples

<table>
<thead>
<tr>
<th>2</th>
<th>1</th>
<th>P</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>P</td>
</tr>
</tbody>
</table>

Partitions

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>P</td>
</tr>
</tbody>
</table>

Partitioning function

Conflict

Consistency: one lock for each partition.

4 work items
Existing Approaches

- **Multiple kernels**: High latency, Global lock
- **One kernel**: Low latency, Local lock

<table>
<thead>
<tr>
<th>Global lock</th>
<th>Local lock</th>
</tr>
</thead>
<tbody>
<tr>
<td>High latency</td>
<td>Low latency</td>
</tr>
<tr>
<td>Multiple kernels</td>
<td>One kernel</td>
</tr>
</tbody>
</table>
• “global”: global lock. “local”: local lock.
• “xCU”: x compute units (kernels).
• “dummy”: just get lock and release lock.
• “global”: global lock. “local”: local lock.
• “xCU”: x compute units (kernels).
• “dummy”: just get lock and release lock.

Both approaches are not good enough.
## Optimal Approach

<table>
<thead>
<tr>
<th></th>
<th>Global lock</th>
<th>Local lock</th>
<th>Optimal</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>High latency</strong></td>
<td>Low latency</td>
<td>Low latency</td>
<td>Low latency</td>
</tr>
<tr>
<td><strong>Multiple kernels</strong></td>
<td><strong>One kernel</strong></td>
<td></td>
<td>Multiple kernels</td>
</tr>
</tbody>
</table>

We need help from new OpenCL feature (channel).
Impact of Channel

Kernel 1
DDR
Kernel 2

Kernel 1
DDR
Kernel 2

Kernel : Verilog Module
Channel : FIFO
Outline

• Background and Motivations
• Design
• Experiment
• Conclusion
Our Proposal

• **Multi-kernel partitioning with channel** is presented to attack the lock overhead.

• **On-chip buffers** are used to efficiently utilize memory subsystem on FPGAs.
Multi-kernel Partitioning

Multiple kernels execute concurrently in producer-consumer manner.
Data_in Kernel

1. Load $W$ tuples from DDR to $tuples[W]$.

2. For ($i \leftarrow 0$ to $W$) do
   
   Compute index $j$ of consumer kernel for $tuple[i]$.
   
   Write $tuple[i]$ to consumer kernel $j$ via channel.

consumer kernel: Data_out or Skewed_handling kernel.

Dispatch rate: one cycle for one tuple.

$\frac{1}{W}$ memory read transactions.
**Data_out Kernel**

1. Read *tuple* from *Data_in* kernel via channel.
2. Compute the partition index of *tuple*.
3. Update the counter *(local)* of partition.
4. Store the *tuple* to on-chip *buffer*.
5. If (*buffer* has $S$ tuples) then
   Store the whole *buffer* to global memory.

Lock handling rate: seven cycles for one tuple.

$\frac{1}{S}$ memory write transactions.
**Skewed_handling Kernel**

1. Read *tuple* from *Data_in* kernel via channel.
2. Update counter (*private*) of skewed partition.
3. Store *tuple* to on-chip *buffer*.
4. If (*buffer* has *S* tuples) then
   
   Store the whole *buffer* to global memory.

**Lock handling rate:** one cycle for one tuple.

\[
\frac{1}{S} \text{ memory write transactions.}
\]
Cost Model

• Given the limitation of FPGA resource, choosing the optimal configuration for two parameters is challenging:
  
  – \textit{DO}: number of \textit{Data\_out} kernels at the consumer stage, [1, 2, 4, 8,16].
  
  – S: number of slots in the on-chip buffer for each partition, [1, 2, 4, 8, 16, 32].

The ranges of S and DO are small, so we consider all the possible combinations.

The cost model is required to predict the performance for each combination.
Outline

• Background and Motivations
• Design
• Experiment
• Conclusion
Experimental Setup

• Platform:
  – Terasic’s DE5-Net board: Altera Stratix V A7 and 4GB 2-bank DDR3.
  – Altera OpenCL SDK version 14.0.

• Data Sets:
  – Tuple format: <key, payload>. Both keys and payloads are 4-bytes.
  – The probability of individual keys follows a Zipf distribution, with the Zipf factor \([0, 1.75]\).
Our cost model can roughly predict the performance for each combination. Optimal combination: (DO = 8, S = 16).
Impact of Skewed_handling Kernel

Significant speedup for the skewed data set.

Optimal combination: \((DO = 8, S = 16)\)
Impact of Number of Partitions

More Stable

Optimal combination: (DO = 8, S = 16)
Impact of Number of Tuples

<table>
<thead>
<tr>
<th>Tuples</th>
<th>Elapsed time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8M</td>
<td>local_1CU</td>
</tr>
<tr>
<td>16M</td>
<td>multi-kernel</td>
</tr>
<tr>
<td>32M</td>
<td></td>
</tr>
<tr>
<td>64M</td>
<td></td>
</tr>
<tr>
<td>128M</td>
<td></td>
</tr>
<tr>
<td>196M</td>
<td></td>
</tr>
</tbody>
</table>

Good scalability
Optimal combination: (DO = 8, S = 16)

10.7X
Outline

- Background and Motivations
- Design
- Experiment
- Conclusion
Conclusion

• We demonstrate the significant overheads of data partitioning on FPGAs.
• We develop a new multi-kernel partitioning approach with on-chip buffers.
• Our proposed approach can achieve 10.7X speedup over the existing implementation.

• Further work: We want to accelerate all the database operators on OpenCL-based FPGAs.
Q & A

• Our Terasic’s DE5-Net FPGA board is denoted by Altera University Program.

• Our research group: Xtra Computing Group  