Boyi: A Systematic Framework for Automatically Deciding the Right Execution Model of OpenCL Applications on FPGAs

Jiantong Jiang (Northeastern University, China),
Zeke Wang (Zhejiang University, China),
Xue Liu (Northeastern University, China),
Juan Gómez-Luna (ETH Zürich, Switzerland),
Nan Guan (Hong Kong Polytechnic University),
Qingxu Deng (Northeastern University, China),
Wei Zhang (Hong Kong University of Science and Technology),
Onur Mutlu (ETH Zürich, Switzerland)
Outline

• Background and Motivations
• Our Solution
• Experiment
• Conclusion
What is OpenCL?

• OpenCL stands for *Open Computing Language*.

• OpenCL has been developed for heterogeneous computing environments with a host-accelerator execution model.
  - The CPU runs the control task.
  - The GPU/ FPGA runs the computing kernel.
OpenCL on FPGA

- Hardware-centric $\rightarrow$ fine-gained parallelism
- Users need to program with HDL.

- Software-centric $\rightarrow$ FPGA as a parallel architecture.
- Users can program with OpenCL.
Conventional OpenCL: NDRange Kernel

Explicit multi-threaded $\rightarrow$ executing the same operation on multiple data concurrently

• How conventional OpenCL performs on an FPGA?
Issue of Conventional OpenCL

- Conventional OpenCL cannot always represent FPGA architecture in an efficient manner.
The optimal performance is enabled by two OpenCL features!
OpenCL Feature 1: SWI Kernel

- The SWI model executes the kernel in only one CU that contains only one work-item.

Pipelined parallelism ➔ No conflict among work items.
OpenCL Feature 2: OpenCL Channel

- OpenCL channel can be used to pass data between two OpenCL kernels (typically SWI).
  - Synchronizing the kernels
  - Reducing the number of global memory accesses
Effect of Two OpenCL Features

An order of magnitude performance improvement over conventional OpenCL!
Challenges

- Two OpenCL features exponentially increase design space.
  - Enabled by two features, we have four OpenCL execution models:
    - NDRange
    - SWI
    - NDRange+Channel
    - SWI+Channel
  - For each execution model, we have at least six optimization methods:
    - SM
    - MC
    - PM
    - UL
    - SIMD
    - CU
  - For each optimization method, we have different pragmas.

The compilation time is extremely long!
There is no systematic study to guide OpenCL programmers on efficiently leveraging these two OpenCL features.
Observation

• Different execution models can significantly affect the performance.

Observation

- Different execution models can significantly affect the performance.

![Graph showing speedup over the GPU baseline for different execution models]

- Execution model should be decided first.
Our Goal

• Can we explicitly determine the most suitable execution model (i.e., whether or not to use two OpenCL features) to optimize OpenCL programs on FPGAs?

• We provide a systematic framework to make such automated decisions.
Outline

• Background and Motivations

• Our Solution
  - OpenCL Pattern Recognition
  - Execution Model Prediction

• Experiment

• Conclusion
Architecture of Boyi

- Boyi explicitly determines the most suitable execution model to optimize OpenCL programs on FPGAs.
  - OpenCL Pattern Recognition
  - Execution Model Prediction

AO: Atomic Operation, MPS: Multi-Pass Scheme, KKC: Kernel-to-Kernel Communication
Outline

• Background and Motivations

• Our Solution
  ➢ OpenCL Pattern Recognition
  ➢ Execution Model Prediction

• Experiment

• Conclusion
Pattern AO

- Issues on FPGAs:
  - Noticeable resource overhead
  - Long latency and low bandwidth
  - Low frequency

$\Rightarrow$ AO is not a good fit on FPGAs.
Pattern AO

- Potential on FPGAs:

<table>
<thead>
<tr>
<th>Input data</th>
<th>Hash index</th>
<th>Histogram</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Pattern MPS

Step 3: 4 work-groups

| in  | 1  | 6  | 4  | 7  | 2  | 8  | 9  | 0  | 1  | 4  | 3  | 5  | 6  | 0  | 2  | 3  |
|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| out | 0  | 1  | 7  | 11 | 0  | 2  | 10 | 19 | 0  | 1  | 5  | 8  | 0  | 6  | 6  | 8  |
| local_sum | 18 | 19 | 13 | 11 |
| pre_sum | 0  | 18 | 37 | 50 |
| out | 0  | 1  | 7  | 11 | 18 | 20 | 28 | 37 | 37 | 38 | 42 | 45 | 50 | 56 | 56 | 58 |
Pattern MPS

• Issues on FPGAs:
  More memory traffic
  → MPS is not a good fit on FPGAs.

• Potential on FPGAs:
• Issues on FPGAs:

  The communication via global memory is expensive.
Pattern KKC

• Potential on FPGAs
  Reducing the number of memory accesses
  Inter-kernel parallelism (i.e., concurrent kernel execution)
OpenCL Pattern Recognition

• We develop nine LLVM passes to recognize three OpenCL patterns.
  • AO recognition
  • KKC recognition
  • MPS recognition

The implementation details can be found in our paper!
Outline

• Background and Motivations
• Our Solution
  ➢ OpenCL Pattern Recognition
  ➢ Execution Model Prediction
• Experiment
• Conclusion
Execution Model Prediction

• Direct prediction

<table>
<thead>
<tr>
<th>AO</th>
<th>MPS</th>
<th>KKC</th>
<th>Direct prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>N</td>
<td>N</td>
<td>NDR</td>
</tr>
<tr>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>SWI</td>
</tr>
<tr>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>SWI</td>
</tr>
<tr>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>SWI</td>
</tr>
<tr>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>NDR+C</td>
</tr>
<tr>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>SWI+C</td>
</tr>
<tr>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>SWI+C</td>
</tr>
<tr>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>SWI+C</td>
</tr>
</tbody>
</table>

Kernels with AO and MPS benefit from the SWI kernel. Kernels with KKC benefit from the OpenCL channel.
Execution Model Prediction

Potential evolution of SWI

<table>
<thead>
<tr>
<th>AO</th>
<th>MPS</th>
<th>KKC</th>
<th>Direct prediction</th>
<th>Potential evolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>N</td>
<td>N</td>
<td>NDR</td>
<td>NDR</td>
</tr>
<tr>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>SWI</td>
<td>SWI+C</td>
</tr>
<tr>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>SWI</td>
<td>SWI+C</td>
</tr>
<tr>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>SWI</td>
<td>SWI+C</td>
</tr>
<tr>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>NDR+C</td>
<td>NDR+C</td>
</tr>
<tr>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>SWI+C</td>
<td>SWI+C</td>
</tr>
<tr>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>SWI+C</td>
<td>SWI+C</td>
</tr>
<tr>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>SWI+C</td>
<td>SWI+C</td>
</tr>
</tbody>
</table>

Conditions: Sufficient FPGA resource.
The SWI kernel is compute-bound.
Outline

• Background and Motivations
• Our Solution
• Experiment
  ➢ Experimental Setup
  ➢ Effect of Execution Model
  ➢ Prediction of Execution Model
• Conclusion
## Experimental Setup

- **Workloads**:

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Source</th>
<th>Description</th>
<th>AO</th>
<th>MPS</th>
<th>KKC</th>
<th>Datasets</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFS</td>
<td>Chai</td>
<td>Breadth-First Search</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>NY, NE, UT</td>
</tr>
<tr>
<td>RSCD</td>
<td></td>
<td>RANSAC</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>2000 iterations</td>
</tr>
<tr>
<td>TQH</td>
<td></td>
<td>Task Queue System</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Basket</td>
</tr>
<tr>
<td>HSTO</td>
<td></td>
<td>Histogram</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>256bins</td>
</tr>
<tr>
<td>SC</td>
<td></td>
<td>Stream Compaction</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>50%</td>
</tr>
<tr>
<td>PAD</td>
<td></td>
<td>Padding</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>1000*999</td>
</tr>
<tr>
<td>CEDD</td>
<td></td>
<td>Canny Edge Detection</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Peppa, Maradona, Paw</td>
</tr>
<tr>
<td>KM</td>
<td>Rodinia</td>
<td>K-Means</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>25600 points, 8 features</td>
</tr>
<tr>
<td>MM</td>
<td>Intel demo</td>
<td>Matrix Multiplication</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>A: 2k<em>1k, B: 1k</em>1k</td>
</tr>
<tr>
<td>MS</td>
<td></td>
<td>Mandelbrot Set</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>640*800, 2000 iterations</td>
</tr>
<tr>
<td>PS</td>
<td>CUDA demo</td>
<td>Prefix Sum</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>262144 points</td>
</tr>
</tbody>
</table>
Performance Comparison

- Exploring optimization combinations

---

Performance speedup of a subset of optimization combinations over baseline for KM. “CUx” indicates x CUs, “SIMDx” indicates the kernel vectorization factor x, “ULx-y-z” indicates the loop unrolling factors x,y,z to the inner, middle and outer loop respectively. “1-x” indicates a multi-kernel design with one producer kernel and x consumer kernel(s). “P” indicates the use of private memory for input feature array and “T” indicates the transposition of input feature array for a better access pattern.
### Performance Comparison

- Comparison of execution models

<table>
<thead>
<tr>
<th>App</th>
<th>Number of combinations</th>
<th>Maximum speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NDR</td>
<td>SWI</td>
</tr>
<tr>
<td>BFS</td>
<td>17</td>
<td>7</td>
</tr>
<tr>
<td>RSCD</td>
<td>25</td>
<td>10</td>
</tr>
<tr>
<td>TQH</td>
<td>9</td>
<td>15</td>
</tr>
<tr>
<td>KM</td>
<td>33</td>
<td>11</td>
</tr>
<tr>
<td>MM</td>
<td>25</td>
<td>9</td>
</tr>
<tr>
<td>MS</td>
<td>7</td>
<td>6</td>
</tr>
<tr>
<td>PS</td>
<td>26</td>
<td>20</td>
</tr>
</tbody>
</table>

It is critical to decide the most suitable execution model when optimizing OpenCL applications on FPGAs. Different execution models result in significant performance differences.
Performance Comparison

• Comparison of execution models

<table>
<thead>
<tr>
<th>Application</th>
<th>Ours (ms)</th>
<th>Existing work (ms)</th>
<th>Our/Existing Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSCD</td>
<td>0.8</td>
<td>28.9</td>
<td>38.3</td>
</tr>
<tr>
<td>TQH</td>
<td>66.9</td>
<td>150.6</td>
<td>2.3</td>
</tr>
<tr>
<td>HSTO</td>
<td>38.8</td>
<td>487.9</td>
<td>12.6</td>
</tr>
<tr>
<td>CEDD</td>
<td>161.9</td>
<td>237.8</td>
<td>1.5</td>
</tr>
<tr>
<td>MM</td>
<td>9.1</td>
<td>34.3</td>
<td>3.8</td>
</tr>
<tr>
<td>MS</td>
<td>27.2</td>
<td>944.1</td>
<td>34.7</td>
</tr>
</tbody>
</table>

## Prediction of Execution Model

<table>
<thead>
<tr>
<th>Application</th>
<th>AO</th>
<th>MPS</th>
<th>KKC</th>
<th>Actual</th>
<th>Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFS</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>SWI</td>
<td>SWI</td>
</tr>
<tr>
<td>RSCD</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>NDR+C</td>
<td>SWI+C</td>
</tr>
<tr>
<td>TQH</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>SWI+C</td>
<td>SWI+C ※</td>
</tr>
<tr>
<td>HSTI</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>SWI+C</td>
<td>SWI+C ※</td>
</tr>
<tr>
<td>SC</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>SWI+C</td>
<td>SWI+C ※</td>
</tr>
<tr>
<td>PAD</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>SWI+C</td>
<td>SWI+C ※</td>
</tr>
<tr>
<td>CEDD</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>KM</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MM</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>NDR</td>
<td>NDR</td>
</tr>
<tr>
<td>MS</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>NDR</td>
<td>NDR</td>
</tr>
<tr>
<td>PS</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>SWI</td>
<td>SWI</td>
</tr>
</tbody>
</table>

*SWI+C ※ indicates the potential evolution of SWI

The actual and predicted execution models roughly match.
Outline

• Background and Motivations
• Our Solution
• Experiment
• Conclusion
Conclusion

• Two OpenCL features are needed to exploit the performance potential of FPGA, since the architecture of FPGA is significantly different from that of GPU.

• We propose Boyi, a systematic framework for deciding the most suitable execution model of an OpenCL application.

• Our automatic prediction of execution model can roughly match the real experimental results.
OpenCL Pattern Recognition

• An LLVM-based OpenCL pattern recognition mechanism
  • AO recognition

  HasAO

  #AOTrue

  • KKC recognition
  • MPS recognition
KKC Recognition

- **R1**: the OpenCL application has two or more OpenCL kernels.
- **R2**: the consumer kernel reads from the same buffer object (in global memory) as a producer kernel writes to.
- **R3**: the producer and the consumer kernels access the buffer object with the same memory access pattern (MAP).
R1: NumKernels

#Kernels

#Kernels>1?

Y

N

#MPSTrue=0

R2: IsSameBuf

R2: IsRdWr

#R2Triplets

#R2Triplets>0?

Y

N

#MPSTrue=0

R4: IsSequential

R4BufferTriplets

R2BufferTriplets

R4Triplets

#R4Triplets

#R4Triplets>0?

Y

N

#MPSTrue=0

R5: VarBuffHost

Args, ArgVals

R5: VarInKernel

Vars, VarVals

Buffs

R5: BuffInKernel

#MPSTrue=0

• **R4**: two kernels access the same buffer object with different MAPs

• **R5**: the intermediate buffer objects are not needed when the OpenCL application is mapped to one work-group
Four Execution Models

- General comparison of four execution models:

<table>
<thead>
<tr>
<th></th>
<th>NDR</th>
<th>SWI</th>
<th>NDR+C</th>
<th>SWI+C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programmability</td>
<td>Moderate</td>
<td>High</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>Computing parallelism</td>
<td>High</td>
<td>Low</td>
<td>High</td>
<td>Moderate/High</td>
</tr>
<tr>
<td>Memory traffic</td>
<td>High</td>
<td>Low</td>
<td>Moderate</td>
<td>Low</td>
</tr>
</tbody>
</table>

- NDR can achieve higher computing parallelism.
- SWI is more easy to program and requires a lower memory traffic.
- OpenCL channel will increase the programming difficulty, reduce the memory traffic and lead to higher computing parallelism.