Accelerating Graph Neural Networks on CPU + FPGA co-processors for scalable track reconstruction tasks

Aneesh Heintz (Cornell University)
Isobel Ojalvo  (Princeton University)
Interaction Network

- TrackML Dataset
- Input data represented as a graph
  - Nodes -> hits
  - Edges -> linear approximations for particle trajectory
- Linear Layer
  - $\text{out} = \text{inp}^T W^T + b$

Accelerating GNNs on CPU + FPGA co-processors for scalable track reconstruction tasks

Diagrams courtesy of Gage Dezoort
Field-Programmable Gate Arrays

- Re-configurable Integrated circuits
  - Consists of several small computational units
- Integrates combinations of
  - Lookup tables (LUTs)
  - Registers
  - On-chip memories (global and local)
  - Arithmetic hardware
- Advantages of FPGAs
  - Support wide, heterogenous and unique parallel implementations
  - Lower latency & more energy efficient than GPUs

Accelerating GNNs on CPU + FPGA co-processors for scalable track reconstruction tasks

Aneesh Heintz
OpenCL

- Open source C-based interface for parallel computing (on CPUs, GPUs & FPGAs) using task and data-based parallelism.
- Implementation
  - Focus on CPU + FPGA co-processors
    - Intel Xeon (CPU, host)
    - Intel Arria 10 (FPGA)
- Adaptable to changes in network architecture

Accelerating GNNs on CPU + FPGA co-processors for scalable track reconstruction tasks
OpenCL Optimizations

- Open source C-based interface for parallel computing (on CPUs, GPUs & FPGAs) using task and data-based parallelism.
- Implementation
  - Focus on CPU + FPGA co-processors
    - Intel Xeon (CPU, host)
    - Intel Arria 10 (FPGA)
  - Adaptable to changes in network architecture

<table>
<thead>
<tr>
<th>Explored OpenCL Optimization</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kernel Types</td>
<td>NDRange (parallelize data) vs. Single Work-Item (parallelize tasks)</td>
</tr>
<tr>
<td>Matrix Multiplication</td>
<td>Endless optimization is possible. Focused on local memory tiling and register blocking</td>
</tr>
<tr>
<td>Bit Precision</td>
<td>Varying input graph data sizes by changing floating point bits</td>
</tr>
<tr>
<td>Double Buffering</td>
<td>Transfer data for the next kernel to the fpga while the previous one is being executed</td>
</tr>
</tbody>
</table>
NDRange Kernel

- Parallelize loop operations across work-items for simultaneous data processing
- Local and global work group dimensions explicitly specify data parallelism
  - Global Size
    - # of work-items to execute
    - Example in figure: 4 x 5
  - Local Size
    - # of work-items to group into a work group

```c
#define global_idx(x_idx, y_idx, m) (x_idx * m + y_idx)
__attribute__((uses_global_work_offset(0)))
__kernel void transpose(__global float *a_t,
                       __global float *a,
                       ushort n,
                       ushort m)
{
    int x_idx = get_global_id(0);
    int y_idx = get_global_id(1);
    a_t[global_idx(y_idx, x_idx, n)] = a[global_idx(x_idx, y_idx, m)];
}
```

Accelerating GNNs on CPU + FPGA co-processors for scalable track reconstruction tasks
Single Work-Item Kernel

- Inherently parallelizes tasks
- Ability to generate custom pipelines
  - Pipelining: Executes tasks in parallel on multiple work items ("cores"). Each task is executed sequentially at the next clock cycle with available resources.
- Optimization specified by "pragmas" that direct behaviors of sections of code
- Smaller enqueuing overhead than NDRange kernels
Local Memory Tiling

- Tiling -- caches sub-blocks of matrices in on-chip local memory
- Reduces repetitive reading from FPGA globally shared memory
- Matrix Multiplication using local memory tiling works as fast as what Intel has written for their in-house FPGA acceleration library (Intel FAL).
Register Blocking

- Increase the amount of work done per thread in each work-item
- Reduces repetitive reading from FPGA local memory
- Decreases global and local memory sizes
  - More device side resources used
  - Fewer host side resources used
- Adding register tiling on top of local memory tiling leads to even faster matrix multiplication execution

Matrix Multiplication Example

Accelerating GNNs on CPU + FPGA co-processors for scalable track reconstruction tasks
Kernel Type Comparisons

- NDRange is more suitable for this application due to the ability to parallelize data across multiple work items.

### System Area Analysis: Component Descriptions

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALUTs</td>
<td>Specifies the number of combinational adaptive look-up tables (ALUTs) that the module uses.</td>
</tr>
<tr>
<td>FFs</td>
<td>Specifies the number of dedicated logic registers that the module uses.</td>
</tr>
<tr>
<td>RAMs</td>
<td>Specifies the number of block RAMs that the module uses.</td>
</tr>
<tr>
<td>DSPs</td>
<td>Specifies the number of digital signal processing (DSP) blocks that the module uses.</td>
</tr>
<tr>
<td>MLABs</td>
<td>Specifies the number of memory logic arrays (MLABs) that the module uses. This value is equal to the number of adaptive logic modules (ALMs) that is used for memory divided by 10 because each MLAB consumes 10 ALMs.</td>
</tr>
</tbody>
</table>
Scalability Study

- Largest Bottlenecks
  - Matrix multiplication
  - Overhead due to enqueuing NDRange kernels
  - 0.75 Pt cut event graphs were too large for the resources available

Accelerating GNNs on CPU + FPGA co-processors for scalable track reconstruction tasks

Aneesh Heintz
Bit Precision Study

- C++ and OpenCL allows for implicit conversions between 8, 16, 32 bits
- Kernel: NDRange loc. + reg. tiling
- No significant difference in runtime executions
- Lower bit precision allows for more on-device data (lower Pt cuts)
- Runtime results presented for Pt cut > 5 GeV
Thank you!

Special thanks to Gage DeZoort, Savannah Thais, Bei Wang, and the GNN Accelerator Group for your continuous help and support during the project!

Proposal Goals Met

✓ Deploy IN in OpenCL on CPU + FPGA
✓ Optimize OpenCL code for efficient execution
✓ Investigate data scalability
✓ Study effects different network architectures & data scalability have on co-processor approach
Backup Slides
Bottlenecks

- Matrix multiplication
- NDRange kernels enqueuing overhead

Accelerating GNNs on CPU + FPGA co-processors for scalable track reconstruction tasks
Bottlenecks

- Matrix multiplication
- Overhead due to enqueuing NDRange kernels
Bottlenecks

- Matrix multiplication
- Overhead due to enqueuing NDRange kernels
  - Higher data value bit precision means larger event graph sizes, limiting the range of possible pt cuts that will run
  - 1 pt runs using 8- and 16-bit precision, but not with 32 bits
    - Necessary RAM allocation is too large
Areas for Future Improvements

- Matrix Multiplication
- Enqueuing NDRange kernel overhead
  - Single work-item kernels have a lower overhead (might scale better)
- Explore fully on-device implementation
  - Array Streaming (Using Pipes / Channels)
  - Pro: more energy efficient, faster
  - Con: won’t be able to handle as much data as the co-processor approach