Final Project
Announcements

▶ Midterm grades released
  - High: 40 (out of 40); Mean: 33; Median: 34

▶ Lab 4 due tomorrow (zero late penalty until Monday noon)
  - Useful notes on array partitioning and unrolling posted on Ed
  - Focus on reducing estimated latency at HLS level first before generating bitstream
  - Designs with high resource utilization may cause routing failures

▶ Instructor OH today cancelled
More Logistics

- Fill out project teaming sheet today!

- Project abstract due Monday 10/31 (no extension)

- Project meetings (with individual team) start next week at Rhodes 471 (CSL)
  - Meeting schedule for next week will be posted on Ed tomorrow
Final Project (35%)

- 3-4 students per group (11-13 groups expected)
  - **Finalize teaming today**
  - Project meetings start next Tuesday

- Two themes
  - **App**: Accelerator design for compute/data-intensive applications
  - **Tool**: Compilation/synthesis for accelerator design/programming

- Project abstract due Monday 10/31
  - Up to one page that describes the project at a high-level (in a few paragraphs)
  - Please list title + theme and team members
Theme 1 (App): Application-Specific Accelerator Design

- Leverage HLS and FPGA to quickly create hardware accelerators for compute-demanding applications
  - Hardware/software co-design with CPU + FPGA
    - Speed up interesting applications from emerging domains such as computer vision, cryptography, machine learning, etc.

- Design languages: C++ or DSL
- HLS tools: Xilinx Vivado/Vtis HLS
- FPGA platforms
  - ZedBoard (embedded): small device, short compile time
  - Alveo (data center): large device equipped with high-bandwidth memory (HBM), long compile time (~hours)
Theme 1: Sample Projects from Previous Years

Canny Edge Detection

Real-Time Vocal Processor of Pop Music

Digit Recognition
Theme 1: Topics to Consider

Ideal application characteristics for HW acceleration
- Abundant parallelism
- Customizable (low-bitwidth) numeric types
- Distributed memory accesses

- Deep learning
  - Systolic arrays for binarized matrix multiplication (or convolution)
  - Domain-specific programming for DNN kernels (using HeteroCL)

- Workloads with high sparsity
  - Sparse linear algebra kernel (e.g., SpMV)
  - Graph processing (e.g., BFS)
Theme 1: More Topics to Consider

- Networking
  - Network security using data sketching (e.g., bloom filter)
  - Data serialization/deserialization for distributed cloud services

- Other applications
  - Stencil-based image/video processing
  - Genomics (low-bitwidth data types)
Theme 1: Do’s and Don’ts

▸ Accelerate an algorithm/application you are familiar with (or easy to understand)

▸ Avoid spending significant effort (>1 week) setting up the baseline implementation (with proper tests)

▸ Do first-order analysis on the parallelism and OI before writing HLS code

▸ Optimize performance at the HLS level and minimize runs of bitstream generation
Theme 2 (Tool): Accelerator-Centric Compilation/Synthesis

- Develop new compilation and/or HLS techniques
  - Compiler analysis & transformations for accelerators
  - Improving core HLS algorithms: scheduling, pipelining, binding
  - Optimizing new design metrics (e.g., security)

- Software frameworks
  - Open-source compiler infrastructure: LLVM, MLIR, CIRCT
  - Commercial HLS as a back end: Vivado/Vitis HLS
Theme 2 Sample Project: Trace-Based Array Partitioning

<table>
<thead>
<tr>
<th>Cycles</th>
<th>RD0</th>
<th>RD1</th>
<th>RD2</th>
<th>RD3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0000</td>
<td>0000</td>
<td>0010</td>
<td>0010</td>
</tr>
<tr>
<td>2</td>
<td>0000</td>
<td>0001</td>
<td>0011</td>
<td>0010</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>10</td>
<td>0000</td>
<td>1001</td>
<td>1011</td>
<td>0010</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

**Important bits (mask)**

```
<table>
<thead>
<tr>
<th>i/j</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
```

**Hash function**

<table>
<thead>
<tr>
<th>Mask</th>
<th>Bank</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>0</td>
</tr>
<tr>
<td>01</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>3</td>
</tr>
</tbody>
</table>

**Partitioning array A**

```c
int A[Rows][Cols];
int sum;

for ( int i = 1; i < Rows - 1; i ++ )
    for ( int j = 1; j < Cols - 1; j ++ )
```
Theme 2: Do’s and Don’ts

- Leverage open-source compiler infrastructures (e.g., LLVM, MLIR)
  - Avoid building a new IR from scratch

- Formulate the problem in an exact way before implementing any heuristic algorithms
Theme 2: Topics to Consider

- Memory system customization
  - Optimization/analysis for memory banking (conflicting accesses allowed)
  - Customized cache generation for irregular data access patterns

- Automated OI analysis
  - First-order data reuse analysis for DNNs
  - Scheduling for fast throughput estimation

- Faster design closure
  - Parallelized HLS and physical implementation
  - Dynamic partial reconfiguration