Automatically design efficient FPGA accelerators

Yuhao Ge
Background

- The demand for scalable, high-performance computing continues to increase
- Moore’s Law is gradually limited by energy consumption
- High performance platform FPGA/GPU becomes popular

Challenges

- Reimplementing the code from CPU to FPGA/GPU has a steep learning curve for those who don’t know architecture well
- Design efficient accelerator is time-consuming even for the experienced programmer
Program an FPGA

---

### Code 1. CNN HLS C Code Snippet

```c
// Skip constant variable initialization and macro definitions for brevity

void CnnKernel(const float* input, const float* weight, const float* bias, float* output) {
    float C[ParallelOut][InSize][InSize];
    for (int i = 0; i < NumOut / ParallelOut; ++i) {  // Initialization
        for (int w = 0; w < InSize; ++w) {
            for (int po = 0; po < ParallelOut; po++)
                C[po][i][w] = bias[i << shift] + po;
        }
    }

    // Convolution
    for (int j = 0; j < NumIn; ++j) {
        for (int h = 0; h < InSize; ++h)  // (5)
            for (int w = 0; w < InSize; ++w)  // (5)
                for (int po = 0; po < ParallelOut; po++)  // (5)
                    for (int p = 0; p < kKernel; ++p)  // (5)
                        for (int q = 0; q < kKernel; ++q)  // (5)
                            C[po][h][w] = weight(i, p, j, h, p, w, q) * input(j, h + p, w + q);  // (3)

    // ReLU + Max pooling
    for (int h = 0; h < OutSize; ++h)  // (5)
        for (int w = 0; w < OutSize; ++w)  // (5)
            for (int po = 0; po < ParallelOut; po++)  // (5)
                output(i, h, w) = max(0.f, C[po][h][w]);  // (3)
}
```

### Code 2. CNN Code Snippet in Merlin C

```c
void CnnKernel()
    const float input[NumIn][InSize][InSize],
    const float weight[NumOut][NumIn][kKernel][kKernel],
    const float bias[NumOut],
    float output[NumOut][OutSize][OutSize]) {

    float C[ParallelOut][InSize][InSize];
    for (int i = 0; i < NumOut / ParallelOut; ++i) {  // Initialization
        for (int h = 0; h < InSize; ++h)  // (5)
            for (int w = 0; w < InSize; ++w)  // (5)
                for (int po = 0; po < ParallelOut; po++)  // (5)
                    C[po][h][w] = bias[i << shift] + po;

    // Convolution
    for (int j = 0; j < NumIn; ++j) {
        for (int h = 0; h < InSize; ++h)  // (5)
            for (int w = 0; w < InSize; ++w)  // (5)
                for (int po = 0; po < ParallelOut; po++)  // (5)
                    for (int p = 0; p < kKernel; ++p)  // (5)
                        for (int q = 0; q < kKernel; ++q)  // (5)
                            C[po][h][w] = weight(i, p, j, h, p, w, q) * input(j, h + p, w + q);  // (3)

    // ReLU + Max pooling
    for (int h = 0; h < OutSize; ++h)  // (5)
        for (int w = 0; w < OutSize; ++w)  // (5)
            for (int po = 0; po < ParallelOut; po++)  // (5)
                output(i, h, w) = max(0.f, C[po][h][w]);  // (3)
```
Figure 3: High-level overview of the GNN-DSE framework
Genetic Algorithms

<table>
<thead>
<tr>
<th>kernel</th>
<th>solution space</th>
<th>RS</th>
<th>GA</th>
<th>HybridGA</th>
<th>HybridGAC</th>
<th>RS</th>
<th>GA</th>
<th>HybridGA</th>
<th>HybridGAC</th>
</tr>
</thead>
</table>

Adaptive search:

Crossover \( p = k_1 (f_{\text{max}} - f')/(f_{\text{max}} - \bar{f}), k1=1 \)

Single point crossover (randomly select the place)

Mutation \( p = k_2 (f_{\text{max}} - f')/(f_{\text{max}} - \bar{f}), k2=0.5 \)

Randomly find a gene (one pragma) and move one step

Clustering when initialize

Use KMeans algorithm to sample the starting population, and select the best individual in each cluster as the initial population
Simulated annealing

<table>
<thead>
<tr>
<th>kernel</th>
<th>solution space</th>
<th>Best GNN result</th>
<th>Average GNN result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>EH</td>
<td>RS</td>
</tr>
</tbody>
</table>

**SA**
- Cooling from $T_0 = 100, T_i = T_{i-1} \times 0.99$
- Acceptance $P = 1$ if $\Delta Perf > 0$ or else $P = e^{\Delta Perf/\text{temp}}$

**HybridSA**
- Random search for 10 mins, get the best 10 designs as starting points
- Run SA 10 times starting with those points, each for 5 mins
- Set initial temperature as $T_0 = \text{Max}(perf)$ which is the max performance during the random sample
Ant colony

• Build a graph for the design space.
  • x-axis for different pragma, y-axis for different configuration for each pragma
  • Edge is constructed between every two configurations from neighbor pragmas

• Put an ant randomly at the first column, let it to choose the next pragma configurations according to the pheromone on the edge

• Test the design after the ant decided all the pragmas

• Update the pheromone (P)
  • For the edges the ant travels,
  • \[ P = 0.9 \times P + \frac{\text{perf}}{100} \]

<table>
<thead>
<tr>
<th>kernel</th>
<th>solution space</th>
<th>Best GNN result</th>
<th>Average GNN result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EH</td>
<td>RS</td>
<td>Ant</td>
</tr>
</tbody>
</table>
OpenTuner: An Extensible Framework for Program Autotuning

• The search engine automatically choose the next design to test. **Multi-armed bandit** with sliding window, area under the curve credit assignment (AUC Bandit) meta technique is used here to decide which search technique to use every moment.
• The measurement engine test the design and report the performance
• They are connected through a database

**AUC bandit:**
Which technique $t$ to use is decided by: $\arg\max_t (\text{AUC}_t + C \sqrt{\frac{2 \log |H|}{H_t}})$

- $|H|$ is the length of the sliding history window (AUC bandit only consider a subset of the history)
- $H_t$ is the number of times the technique has been used in that history window
- $C$ is a constant to control the exploration/exploitation trade-off
- $\text{AUC}_t$ denotes the performance of each searching techniques (The area under the curve credit assignment mechanism)
  - $\text{AUC}_t = \frac{2}{|V_t|(|V_t|+1)} \sum_{i=1}^{|V_t|} iV_{t,i}$
  - $V_{t,i}$ is 1 if using technique $t$ the $i^{th}$ time in the history resulted in a speedup, otherwise 0
## Results

<table>
<thead>
<tr>
<th>Kernel</th>
<th>Solution Space</th>
<th>Best GNN Result</th>
<th>Average GNN Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>EH</td>
<td>RS</td>
</tr>
</tbody>
</table>

### Diagrams
- **2mm**: AUCBanditMetaTechniqueA
- **Gemm-p**: AUCBanditMetaTechniqueA
- **mvt**: AUCBanditMetaTechniqueA