Parallelizing FPGA Technology Mapping using GPUs

Doris Chen
Deshanand Singh
Aug 31st, 2010
Motivation: Compile Time

In last 12 years:

- 110x increase in FPGA Logic, 23x increase in CPU speed, 4.8x gap

**Question:** Can we utilize the highly parallel GPU to speed up FPGA CAD algorithms?
What are GPUs?

- Graphics Processing Units (GPUs)
  - Optimized to maximize throughput

- Single-Instruction, Multiple Data (SIMD) Architecture
  - Optimized to run many threads in parallel
    - Each thread performs a fixed set of operations on a small subset of data
    - Threads are arranged in blocks, executed in warps (32 threads/warp)

- Consist of an array of Streaming Multiprocessors (SMs)
GPU Architecture

- Each SM operates on 1 warp at a time
  - Warps can be swapped in and out
- Each SM contains:
  - Instruction Cache
  - 8 Streaming Processors (SP), each capable of executing 1 thread at a time
  - 4 clock cycles are required to execute one instruction for all 32 threads
- Shared memory
  - 16k only
  - Accessible by all 8 SPs
Programming GPUs

- Used Nvidia’s CUDA API (Compute Unified Device Architecture)

Host Program

- Resides on CPU, responsible for setting up the memory
- Launches the GPU ‘kernel’

GPU ‘kernel’

- Code executed by every thread on a different data element

![Diagram showing the interaction between Host CPU and GPU Device]
GPUs vs. CPUs

CPU: **minimize latency** experienced by 1 thread
- big on-chip caches
- sophisticated control logic to extract parallelism

GPU: **maximize throughput** of all threads
- Achieves massive parallelism through explicitly parallel code with thousands of lightweight threads
- Large latency incurred for global memory access
- Multithreading is used hide latency
  - Can avoid the big caches of CPUs
Warp Scheduling

- Multiple warps can be assigned to the same SM
- Whenever a warp stalls, the GPU can swap in a new warp
  - Hides latency
- Some things are not swapped out
  - Shared memory
  - Register file

Thread Warp

| Thread 1 | Thread 2 | Thread 3 | Thread 32 |

Active Warps

- Thread Warp 1
- Thread Warp 2
- Thread Warp 8

GPU Pipeline
Limitations of the kernel

- Threads on the same SM must execute the same instruction
  - All other threads will do nothing
- Control structures and data-dependent operations are inefficient

```c
if (x == 0)
    {
        ......
    }
else
    {
        ......
    }
Max( array[...] );
```

Sequential execution of threads limits GPU parallelism
Graph Algorithms on GPU??

- GPUs are great when performing the same computation on lots of data
- FPGA CAD algorithms are graph-based
  - A lot of control structures inherent in the algorithm
    - Control structures → not very good for GPUs
    - Requires serialization
    - Loses the benefits of parallelism on GPU
  - Irregular data access
    - Edges between nodes in the graph are random

**Objective:** Prototype this idea through parallelizing one step of the FPGA CAD flow on the GPU
FPGA CAD Flow

- **Logic Synthesis**
  - Converts HDL into an optimized netlist of 2-input gates

- **Technology Mapping**
  - Maps the netlist into logic blocks that exist on the FPGA (LUTs)

- **Clustering**
  - Logic blocks are clustered to satisfy legality constraints and to minimize placement effort

- **Placement**
  - Clusters are assigned locations on the FPGA

- **Routing**
  - Implements the connections between logic elements using the programmable fabric of the FPGA
Technology Mapping Steps

- Convert the netlist into an AND-Inverter Graph (AIG)
  - Only have 2-input AND gates, and inversions on edges of the graph
- Each node has an associated depth
  - The distance between this node and its Primary Inputs

![Diagram of AND gate connections]

AND gates

Inverted Edge

c a b d x y
Generate Cuts

- Generate Coverings, or “Cuts”
  - Each cut will result in 1 Look-Up Table (LUT)
  - Represent a cut by its inputs
- Find the optimal set of Cuts that achieves the following:
  - Minimal Area
    - Fewer total cuts, the better
  - Minimal Depth
    - Good for timing

\[
\begin{align*}
\text{Cut1} &= \{x, y, z, w\} \\
\text{Cut2} &= \{r, w, u, v\}
\end{align*}
\]

Mapped depth = 2 LUTs
Cut Computation

- Start from Primary Inputs (PI) to Primary Outputs (PO)
  - PIs – input pins or register outputs
  - POs – output pins or register inputs
- Add the current node to the cutset
- Enumerate new cuts by taking the cross-product of cutsets of input nodes
- Compute the cost of the new cut
- Add this new cut into the sorted cutset
- After making all computations, select the ‘best cut’ for each node
  - Starting from POs and go backwards
- This set of best cuts is now the optimal techmapped netlist

```
r, pbc, pq, abq, abc
p, ab
q, bc
a, b, c
```
Iterative Tech Mapping

- Cut computation is run multiple times, at different stages of the synthesis flow
- Each iteration may optimize for different metrics
  - Iter 1: Optimize for Delay
  - Iter 2: Optimize for Area
  - Iter 3: Optimize for Delay
  - ...
  - Iter 5: Optimize for True Area
  - Iter 6: Optimize for True Area
Try to estimate the **real area cost** required by a cut
- We need to recursively un-do the cut, only count the additional LUTs that we would need to make

---

**True Area Flow**

- Original Network
- Unmapped Network
- True-Area Remapped Network
Choice Nodes

- Gives techmapping more flexibility when it selects cuts
- Either $a$ or $e$ could be timing-critical
  - If critical, we should shift the signal closer to its output
- Postpone the actual cut selection until we know more accurate timing info
- Don’t know what’s critical until we have mapped its input nodes

$X_1$: Cuts=$edcr,...$

$X_2$: Cuts=$abcs,...$
How to Parallelize Techmapping?

- Levelize the circuit
- Perform cutset enumeration on all nodes in the same level in parallel
  - No data dependencies

![Diagram]

- Level 1: a, b, c
- Level 2: p, ab
- Level 3: r, pbc, pq, abq, abc

PIs to POs
Algorithm

- Created 3 kernels
  - *compute_and* implements cut computation
  - *compute_choice* performs choice node operations
  - *finalize* updates data that the next level may need

```
Levelize the netlist
For i=0 to max_levels do
    launch *execute_and* kernel
    if choice nodes exist in level i then
        launch *execute_choice* kernel
    end if
execute *finalize_kernel*
End For
```
Memory Management

- Global memory writes and reads are expensive
- Copy the netlist to the GPU once (remains unchanged)
  - Only copy the set of selected cuts back and forth
- Remove all un-utilized fields in the netlist, and only retain a bare-bones netlist on the GPU
- Data caching is important
  - Consider a simple for loop:
    
    ```
    for (i = 0; i < netlist->size(); i++)
    ```
  - Every time the exit condition is checked, there is a global access penalty
Dark Side of Warp Scheduling

- Multiple warps are assigned to the same SM
  - Can be swapped in and out to hide latency
- Some things are not swapped out
  - Shared Memory – must have enough to accommodate all active blocks assigned to this SM
  - Register file – must have enough to accommodate all active threads assigned to this SM
- Either metric can limit how many warps assigned per SM
- Registers became the limitation for us

Thread

<table>
<thead>
<tr>
<th>Registers allocated Per-thread</th>
</tr>
</thead>
</table>

Block

| Shared Memory allocated Per-block |
Register Tradeoffs

- The **more active** threads running on an SM, the **fewer** registers each thread can use
  - Fewer registers implies larger number of accesses to global memory
  - Fewer active threads inhibits the SM’s ability to hide latency

![Graph showing the relationship between registers/threads and percent occupancy vs. relative time.](image)
Implementation of True Area

- The GPU cannot handle recursion natively
- Instead, we implement an iterative approach
  - Allocate a queue, where we put the nodes to visit
  - If the queue size is exceeded, then give up
    - This is ok because True Area is just a fixup stage
    - In other words, just stick with the existing solution (the chosen cut)
- Is this safe?
Is This Safe?

- We are unmapping/remapping nodes that are in different levels
- Wouldn’t we run the risk of working on the same node?
- True Area operates only on the MFFC cone
- So it is actually SAFE to unmap nodes in the MFFC cone
  - Guaranteed no other threads will be accessing the same node
## Runtime Results

- Implemented an alternate Tech Mapper in ABC
  - ABC is a state-of-the-art logic synthesis tool
- Used a set of large industrial benchmarks
  - Need large circuits to see speedups due to GPU overhead
- Achieved 4X speedup on parts that are parallelized

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Max # Levels</th>
<th>Avg Gates/Level</th>
<th>if Map (ms)</th>
<th>gpuMap (ms)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>beamforming</td>
<td>184</td>
<td>1531</td>
<td>5920</td>
<td>2560</td>
<td>2.2</td>
</tr>
<tr>
<td>netcard</td>
<td>86</td>
<td>1502</td>
<td>2450</td>
<td>1100</td>
<td>2.2</td>
</tr>
<tr>
<td>cc_des25</td>
<td>30</td>
<td>4033</td>
<td>1740</td>
<td>670</td>
<td>2.6</td>
</tr>
<tr>
<td>oc_jpeg</td>
<td>49</td>
<td>2061</td>
<td>2440</td>
<td>910</td>
<td>2.7</td>
</tr>
<tr>
<td>radar20</td>
<td>92</td>
<td>2137</td>
<td>4040</td>
<td>1410</td>
<td>2.9</td>
</tr>
<tr>
<td>sat_solver</td>
<td>77</td>
<td>6071</td>
<td>7610</td>
<td>2480</td>
<td>3.1</td>
</tr>
<tr>
<td>uoft_raytracer</td>
<td>111</td>
<td>4954</td>
<td>8950</td>
<td>2740</td>
<td>3.4</td>
</tr>
<tr>
<td>uoft_stereo_vision</td>
<td>24</td>
<td>7599</td>
<td>4860</td>
<td>1290</td>
<td>3.8</td>
</tr>
<tr>
<td>blowfish_area</td>
<td>104</td>
<td>27121</td>
<td>54310</td>
<td>13200</td>
<td>4.1</td>
</tr>
<tr>
<td>blowfish_pipe</td>
<td>22</td>
<td>134964</td>
<td>57900</td>
<td>13710</td>
<td>4.2</td>
</tr>
<tr>
<td>vga_lcd</td>
<td>40</td>
<td>27435</td>
<td>17490</td>
<td>4070</td>
<td>4.3</td>
</tr>
<tr>
<td><strong>Geometric Average</strong></td>
<td><strong>139</strong></td>
<td><strong>141</strong></td>
<td></td>
<td></td>
<td><strong>3.1</strong></td>
</tr>
</tbody>
</table>
QoR Measurements

- Measured the quality of results by running both mapping algorithms through the Quartus II CAD tool.

Diagram:
- HDL
  - Replace Black Boxes
    - Quartus II MAP
      - Write Blif
  - ABC ifMap
  - ABC gpuMap
    - Write Verilog
      - Annotate Preserve Constraints
        - Quartus II Fitter
          - Quartus II STA
Quality of Result (QoR)

- Placed and Routed the tech mapped result to verify the correctness of \texttt{gpuMap}
- Used Altera’s Quartus II tool to P&R the \texttt{ifMap} netlist, and the \texttt{gpuMap} netlist
- Shows equivalent QoR to the original algorithm

<table>
<thead>
<tr>
<th>Circuit</th>
<th>if Fmax</th>
<th>gpu Fmax</th>
<th>if Area</th>
<th>gpu Area</th>
<th>if LU%</th>
<th>gpu LU%</th>
</tr>
</thead>
<tbody>
<tr>
<td>beamforming</td>
<td>149</td>
<td>174</td>
<td>18414</td>
<td>19425</td>
<td>45</td>
<td>46</td>
</tr>
<tr>
<td>netcard</td>
<td></td>
<td></td>
<td>164730</td>
<td>168912</td>
<td></td>
<td></td>
</tr>
<tr>
<td>cc_des25</td>
<td>299</td>
<td>299</td>
<td>137750</td>
<td>137750</td>
<td>56</td>
<td>56</td>
</tr>
<tr>
<td>oc_jpeg</td>
<td>191</td>
<td>188</td>
<td>13368</td>
<td>12601</td>
<td>23</td>
<td>22</td>
</tr>
<tr>
<td>radar20</td>
<td>144</td>
<td>165</td>
<td>12218</td>
<td>12471</td>
<td>27</td>
<td>28</td>
</tr>
<tr>
<td>sat_solver</td>
<td>125</td>
<td>124</td>
<td>20855</td>
<td>20249</td>
<td>42</td>
<td>44</td>
</tr>
<tr>
<td>uoft_raytracer</td>
<td>75</td>
<td>75</td>
<td>44566</td>
<td>45432</td>
<td>72</td>
<td>67</td>
</tr>
<tr>
<td>uoft_stereo_vision</td>
<td>129</td>
<td>120</td>
<td>77307</td>
<td>80461</td>
<td>60</td>
<td>61</td>
</tr>
<tr>
<td>blowfish_area</td>
<td>184</td>
<td>193</td>
<td>50191</td>
<td>48854</td>
<td>72</td>
<td>73</td>
</tr>
<tr>
<td>blowfish_pipe</td>
<td>73</td>
<td>62</td>
<td>214067</td>
<td>212716</td>
<td>66</td>
<td>66</td>
</tr>
<tr>
<td>vga_lcd</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Geometric Average</td>
<td>139</td>
<td>141</td>
<td></td>
<td>48</td>
<td></td>
<td>48</td>
</tr>
</tbody>
</table>
Conclusion

- We can use GPUs to parallelize graph-based algorithms
- Memory management is essential to getting good results
- Can achieve an average of 3.1X speedup by parallelizing Technology Mapping on a GPU
- Additional speedups may be possible by parallelizing ALL parts of Technology Mapping
  - Timing analysis
  - Truth table computation of the final selected cuts
Thank You!

- Questions?
Backup Slides
Overview

- Motivation
- Introduction to GPUs
- What is Technology Mapping?
- Approach
- Optimizations
- Results
- Conclusion
Motivation

- Field-Programmable Gate Arrays (FPGA) are programmable logic devices
  - Has a regular, grid-like structure consisting of configurable Look-Up Tables (LUTs) surrounded by programmable routing network
- Shorter time-to-market
- Excellent for prototyping
- Cheaper than ASICs
- Requires a CAD tool to help the designers take their HDL design, and place and route it on the FPGA
What are GPUs?

- Extremely efficient at performing computations on large quantities of data
- Runs thousands of lightweight threads in parallel
  - Threads are arranged in blocks, executed in warps (32 threads/warp)
GPU Advantages

- Extremely efficient at performing computations on large quantities of data
- Achieves massive parallelism
  - \(32 \text{ SMs} \times 1 \text{ warp/SM} \times 32 \text{ threads/warp} = 1024\) parallel threads
  - Although optimized for throughput, it’s not optimized for latency
- To hide latency, warps can be swapped in and out if a stall occurs
FPGA CAD Flow

- **Technology Mapping**
  - **Input:** A netlist consisting of 2-input primitive gates
  - **Output:** A netlist consisting of technology-dependent logic blocks
The Technology Mapping Problem

- Take a circuit consisting of 2-input combinational gates, and map it to nodes that exist on the target device.

- The resulting netlist can then be clustered, and placed on the FPGA.
Memory Hierarchy

- Registers and Shared memory accesses happen in a single cycle
- Global (off chip) memory can take hundreds of cycles
Copying data back and forth between the GPU and CPU is extremely expensive.

Consider the differences in bandwidth:
- PCIe gen2 peak bandwidth = 6 GB/s (communication link from GPU to GPU)
- GPU load/store DRAM peak bandwidth = 150 GB/s

CUDA memcpy()
Caching of Data

- Global memory access are SLOW
- Consider a simple for loop:
  \[ for \ (i = 0; \ i < \text{netlist->size()}; \ i++) \]

- The netlist is in global memory
- Every time the exit condition is checked results in a global access penalty
  - Loops are everywhere!!

- Solution:
  - just cache these frequently accessed data (that don’t change) in local variables
  - Reduces global memory accesses

- Speedup:
  - 45%
Still More Memory Overhead

- Because a large number of Cuts are generated, space is allocated for them on the GPU
  - The CPU does NOT care!
  - Don’t copy back useless data
- Solution:
  - Create a ‘scratch space’ on the GPU global memory to store cuts
- Speedup:
  - 4X improvement
Register File

- Threads require exclusive access to registers in large register file
- There must be enough total registers to hold information for all threads in the group of active warps
- Allows the streaming multiprocessor to quickly swap between warps since the register values don’t leave the SM
Future Work

- Our current approach partitions the netlist into disjoint levels and performs mapping on each node in parallel.
  - Results from one level may be necessary in the computation of subsequent levels.
    - Stored in off-chip, global memory = SLOW
- **Better method** would involve writing results to shared memory.
  - Complex to implement because nodes may require results from multiple different shared memories.
Caching of Data

- Global memory access are SLOW
- Consider a simple for loop:
  ```cpp
  for (i = 0; i < netlist->size(); i++)
  ```
- The `netlist` is in global memory
- Every time the exit condition is checked results in a global access penalty
  - Loops are everywhere!!
- Solution:
  - just cache these frequently accessed data (that don't change) in local variables
  - Reduces global memory accesses
How to Measure

- Implemented an alternate Tech Mapper in ABC
- ABC is a state-of-the-art logic synthesis tool

ABC Modifications

- Read Blif
- Structural Hash
- Insert Choices
- ifMap
- Write Verilog
- gpuMap

gpuMap Implementation

- ABC
- GPU Netlist
- Extract Covers
- Kernel1
- Kernel2
- Kernel3