High-Level Synthesis of Multiple Dependent CUDA Kernels for FPGA

Swathi Gurumani\textsuperscript{2}, Hisham Cholakkai\textsuperscript{2}, Yun Liang\textsuperscript{3}, \textbf{Kyle Rupnow}\textsuperscript{12}, Deming Chen\textsuperscript{4}

\textsuperscript{1}Nanyang Technological University
\textsuperscript{2}Advanced Digital Sciences Center, Illinois at Singapore
\textsuperscript{3}Peking University
\textsuperscript{4}Univ. of Illinois Urbana-Champaign
High-Level Synthesis

- Automatic generation of hardware from algorithm descriptions
  - RTL design time high for complex designs

- Different input languages
  - Extensions to C/C++
    (SystemC, ImpulseC)
  - Functional (Haskell), GPGPU (CUDA), Graphical (LabView)
High-Level Synthesis Tools

- Facilitate design space exploration
  - Compiler directives or language features
  - Automate (partially) selection of design parameters

- Challenge – extracting parallelism
  - Require restructuring or reimplementation of code in HLS specific manner

- Data-parallel input languages provide inherent advantage
Parallel Computing & GPU Languages

- Shift towards parallel computing & heterogeneous

- CUDA programming model (NVIDIA)
  - Minimal extensions to C/C++
  - CUDA (GPU), MCUDA (Multi-core), FCUDA (FPGA)

- CUDA advantages for HLS
  - Easier analysis of application parallelism
  - Exploration of parallelism granularity options
Synthesis of CUDA Kernel

- FCUDA - CUDA to FPGA [SASP’09], [FCCM ‘11]
  - Automates design space exploration of single CUDA kernel
    - Match GPU performance with significantly less power
  - Currently supports only single kernel synthesis
Synthesis of Multiple CUDA Kernels

- Possible to create single enclosing wrapper kernel

- Single Enclosing Wrapper Kernel is not Ideal
  - Must fully-buffer all sub-kernel communications on-chip
  - Must use the same thread organization for sub-kernels
  - Forces all sub-kernels to be CUDA device-only functions
Objective

- Map multiple communicating CUDA kernels onto FPGA
  - Allow fine-grained communication
  - Enable data streaming
  - Handle different thread organizations

- Key Contributions to synthesize communicating CUDA kernels to RTL
  - Identify key challenges in automation
  - Case study of stereo-matching algorithm
Multi-Kernel Synthesis - Steps

- Individual Kernel Synthesis
- Communication Buffer Generation
- Analytical Design Space Exploration
- Implementation and Verification
Individual Kernel Synthesis

- Kernel extraction and FCUDA flow

- Initial solution of cores to be minimal in area
  - Perform joint design space exploration kernels later!

- Measure resource usage and latency
Communication Buffers

- Generate control flow graph (CFG) for kernels
  - ASAP scheduling to determine execution critical path

- Buffers between each pair of communicating kernels
Communication Buffers

- Size of buffers?
  - Full-size buffers infeasible

- Data access pattern analysis
  - Initial buffer size = \textit{minimal data processing quanta}
  - Bigger sizes explored in analytical model

- Growth rate of communication buffer

- Include overlap data size for correctness
  - Boundary data for algorithms with windowed computations
Buffering Schemes

a. Single-Buffer Flow

b. Dual-Buffer Flow
Analytical Design Exploration Model
Analytical Design Exploration

nQuanta = 8  ➔  Max cores = (4,2,4)
Analytical Design Exploration

nQuanta = 8 \Rightarrow \text{Max cores} = (4,2,4)
Analytical Design Exploration

\[ n\text{Quanta} = 8 \implies \text{Max cores} = (4, 2, 4) \]
Analytical Design Exploration

\( n\text{Quanta} = 8 \Rightarrow \text{Max cores} = (4,2,4) \)

Only 2 cores of K2 is possible!
Analytical Design Exploration

$nQuanta = 16 \Rightarrow \quad \text{Max cores} = (8,4,8)$

Increase $nQuanta$
Analytical Design Exploration

nQuanta = 16 ➔ Max cores = (8, 4, 8)
Implementation and Verification

- Core allocations from analytical model
  - AutoPilot-C pragmas for suggested parallelism

- Communication buffers and kernel-level parallelism

- SystemC simulation

- Vivado Synthesis
Stereo Matching

- Two spatially separated color cameras
- Distance in pixels between the same object in the images infers depth
- Complex algorithms to match pixels
Case Study – Stereo Matcher

- Left Image
  - Census Transform Kernel
    - Left GridBuilding Kernel
      - RGB to Lab Conversion Kernel
      - Matching Kernel
    - Cross Correction Kernel
      - Pre Filtering Kernel
        - Median Filtering Kernel
          - Left Depth Map
      - Pre Filtering Kernel
        - Median Filtering Kernel
          - Right Depth Map
  - Census Transform Kernel
  - Right GridBuilding Kernel
    - RGB to Lab Conversion Kernel

- Right Image
  - Census Transform Kernel
  - Right GridBuilding Kernel
    - RGB to Lab Conversion Kernel
Design Space for Dual-Buffer Flow

Log Latency (Mcycles)

Sum of Normalized Resource Use
Design Space for Dual-Buffer Flow

![Graph showing log latency vs. sum of normalized resource use]

- 6x96(DB)
- 12x96(DB)
- 18x96(DB)
- 12X192(DB)
- 18X192(DB)
- 12x384(DB)
- 18x384(DB)
- 24x384(DB)
- 36x384(DB)
- 48x384(DB)
- 72x384(DB)
- 96x384(DB)
- 144x384(DB)
Design Space for Dual-Buffer Flow

Log Latency (Mcycles)

Sum of Normalized Resource Use

- 6x96(DB)
- 12x96(DB)
- 18x96(DB)
- 12x192(DB)
- 18x192(DB)
- 12x384(DB)
- 18x384(DB)
- 24x384(DB)
- 36x384(DB)
- 48x384(DB)
- 72x384(DB)
- 96x384(DB)
- 144x384(DB)
Design Space for Dual-Buffer Flow

Sum of Normalized Resource Use

Log Latency (Mcycles)
Design Space for Dual-Buffer Flow

Selected solution
Design Space for Dual and Single Buffer Flow
Design Space for Dual and Single Buffer Flow
Performance–Power Comparison

- HLS of sequential code achieved speedup of 6.9x over software [FPT’11]

- HLS of CUDA parallel code achieved speedup of >50x over sequential software **

- Greater exposed parallelism provides synthesis tool greater opportunity for optimization
Challenges in Automation – Single Kernel

- Single kernel synthesis
  - Critical: Replicating the initial solution for concurrency

- Multi-kernel synthesis
Challenges in Automation – Single Kernel

- Optimize thread index computations
  - **Solution:** Improved analytical techniques in FCUDA to optimize index computations

- Floating-point to fixed-point computations
  - **Solution:** Automatic transformation with functional verification of transform

- Inefficient implementations of difficult operations
  - **Solution:** Automatic instantiation of library elements for common but challenging operations
Challenges in Automation – Multiple Kernel

- Selection of single-core implementation
  - Solution: Complex value function, knowledge of resource criticality, and iteration of entire design flow

- Automatic buffer-generation and insertion
  - Solution: Complex memory access pattern analysis and transformations (See upcoming FPGA 13 paper)

- Performance estimation within synthesis process
  - Solution: Improved analytical model for loop bounds, trip counts, resource estimates

- Sub-kernel optimizations to match pipeline stage latencies
  - Solution: Improved ability to combine or split pipeline stages
Conclusion

- Multi-kernel CUDA synthesis is important

- Manual process for mapping multiple dependent CUDA kernels to FPGA

- Performance parity with GPU consuming 16x less energy than GPU
  - Benefit of data-parallel input language for HLS

- Fully automating multi-kernel synthesis is challenging
Acknowledgement

- A*STAR HSS Funding
- Peking University
- University of Illinois at Urbana-Champaign
- ADSCs Lab Colleagues
  - Hongbin Zheng
  - Muhammad Teguh Satria