NANYANG TECHNOLOGICAL UNIVERSITY

ADSC
Illinois at Singapore Pte Ltd

High-Level Synthesis of Multiple Dependent CUDA Kernels for FPGA

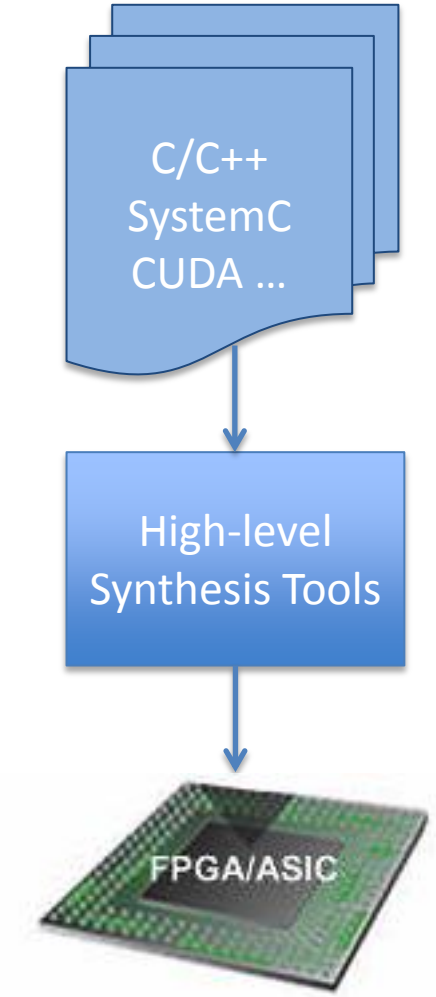
Swathi Gurumani[2], Hisham Cholakkai[2], Yun Liang[3],
Kyle Rupnow[12], Deming Chen[4]

[1]Nanyang Technological University
[2]Advanced Digital Sciences Center, Illinois at Singapore
[3]Peking University
[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 (Haskel), GPGPU (CUDA), Graphical (LabView)

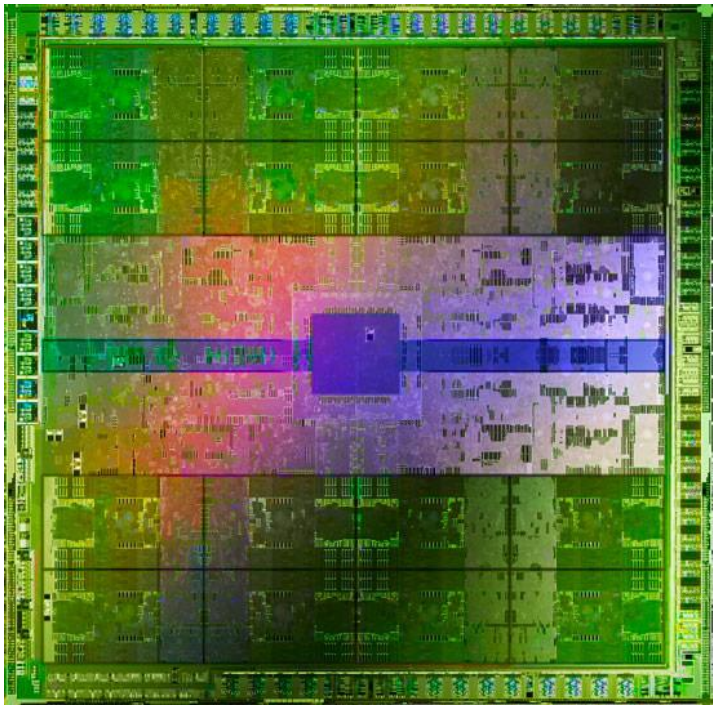


High-Level Synthesis Tools

- 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

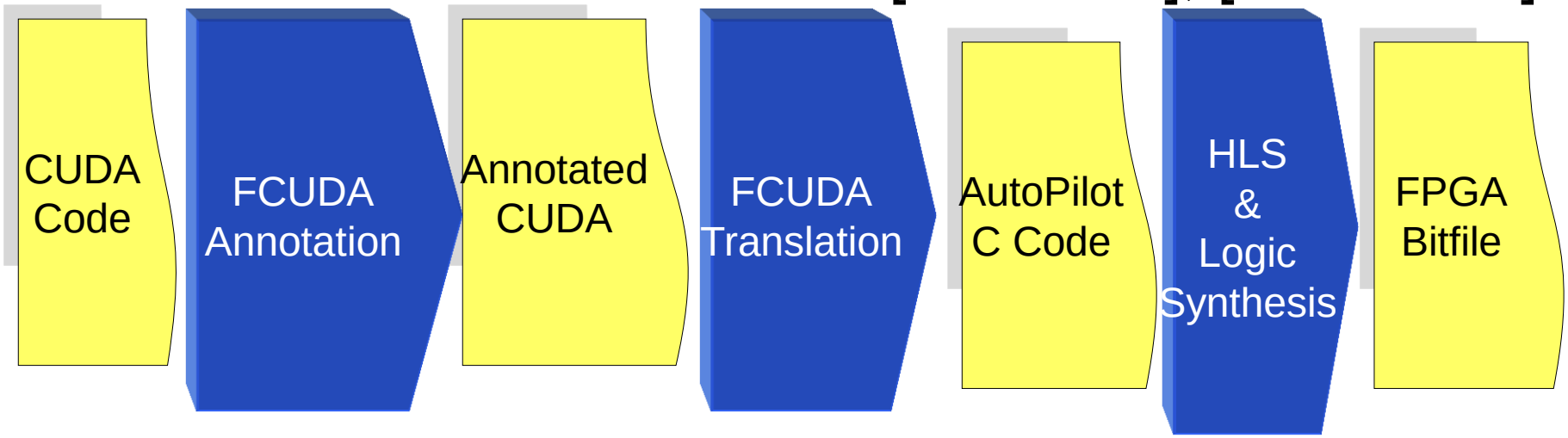
Parallel Computing & GPU Languages

- 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

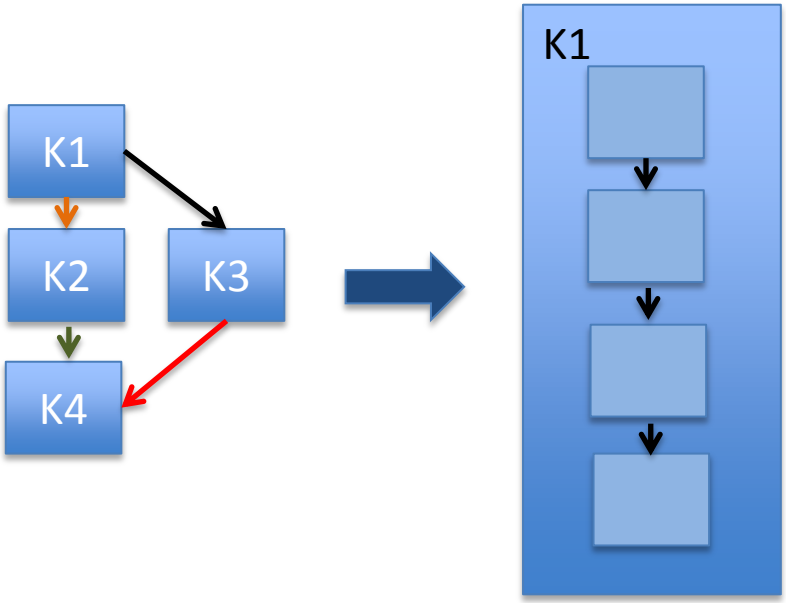
- FCUDA - CUDA to FPGA [SASP'09], [FCCM '11]



- 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

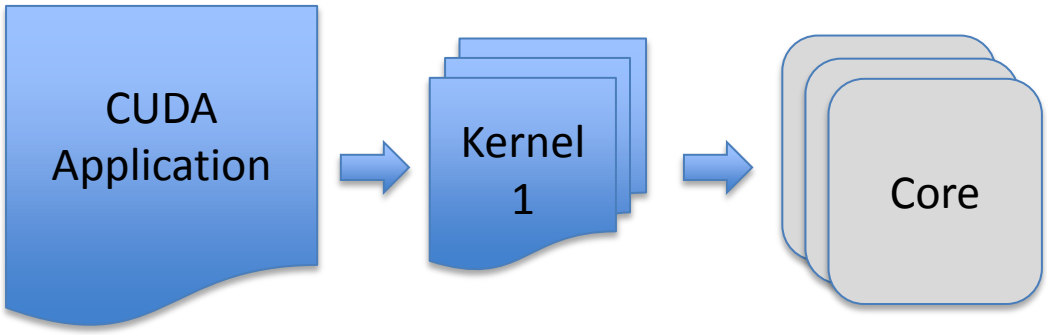
- Key Contributions to synthesize communicating CUDA kernels to RTL
  - Manual step-by-step procedure
  - Identify key challenges in automation
  - Case study of stereo-matching algorithm

Multi-Kernel Synthesis - Steps

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

Individual Kernel Synthesis

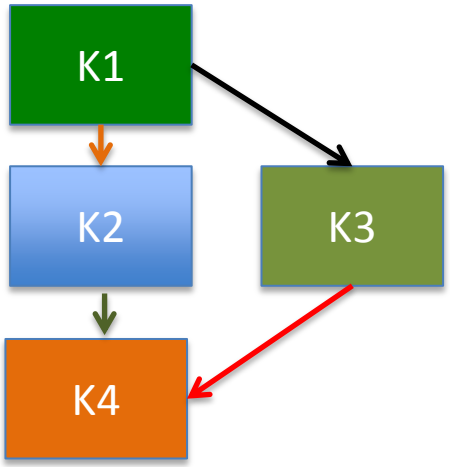
- Kernel extraction and FCUDA flow
- Initial solution of cores to be minimal in area
  - Perform joint design space exploration kernels later!
- Measure resource usage and latency

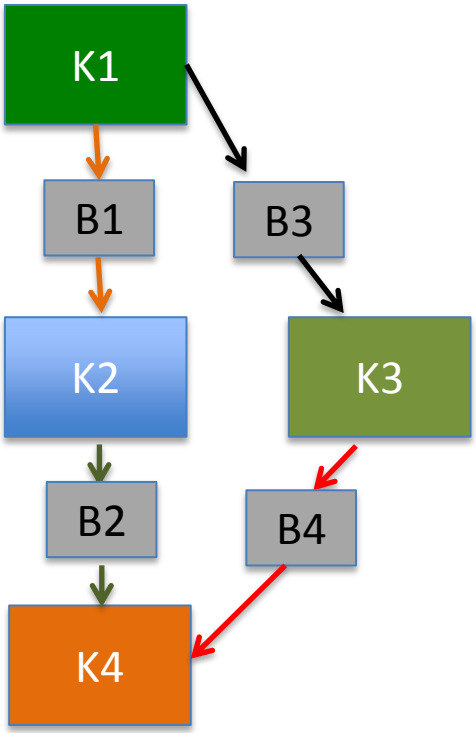


Communication Buffers

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

- Buffers between each pair of communicating kernels

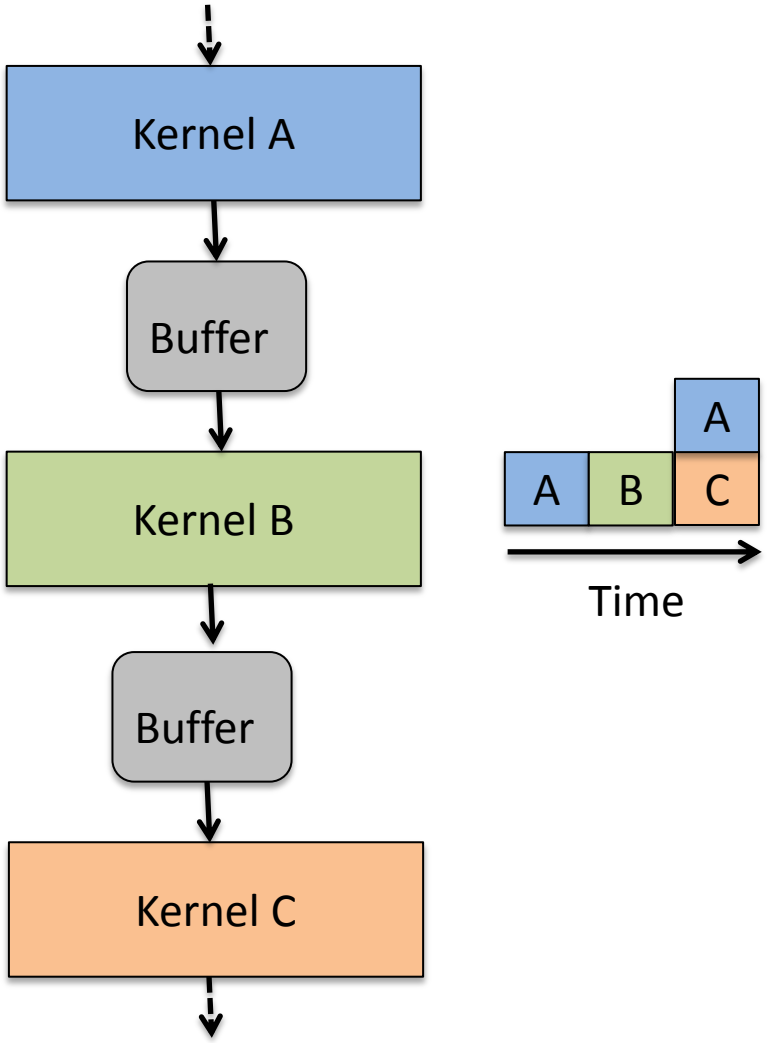




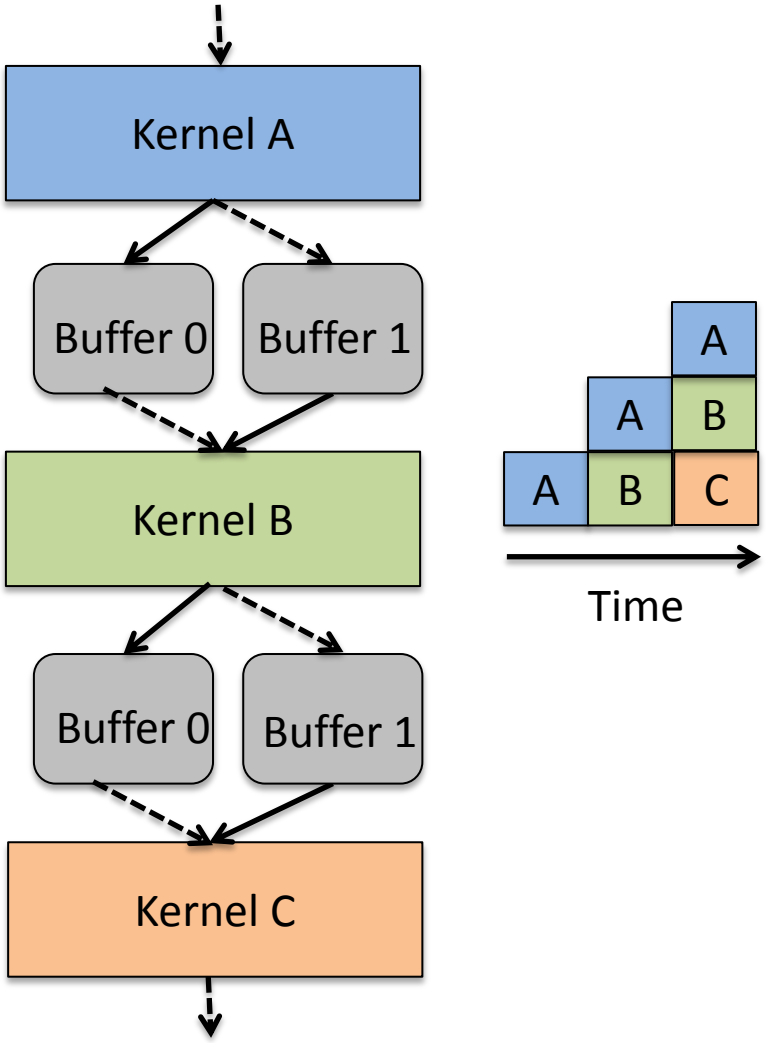
Communication Buffers

- Size of buffers?
  - Full-size buffers infeasible
- Data access pattern analysis
  - Initial buffer size = **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

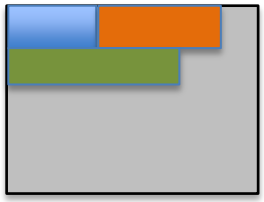


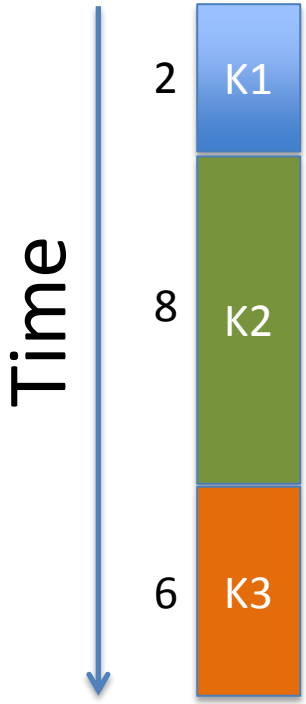
b. Dual-Buffer Flow

Analytical Design Exploration Model

Analytical Design Exploration

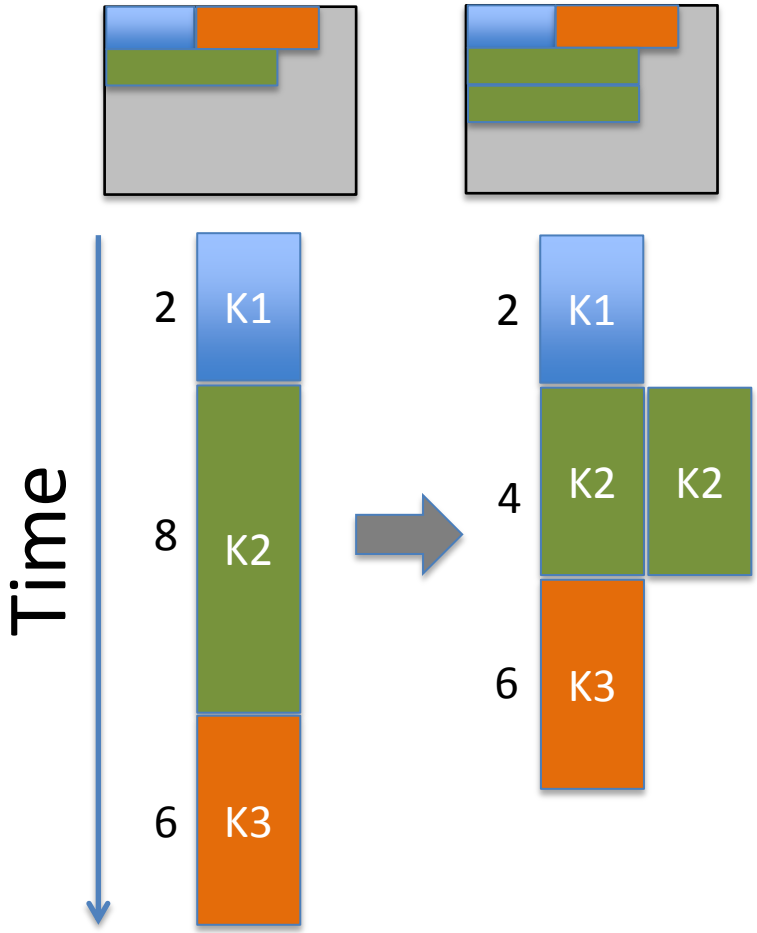
nQuanta = 8 ➔ Max cores = (4,2,4)





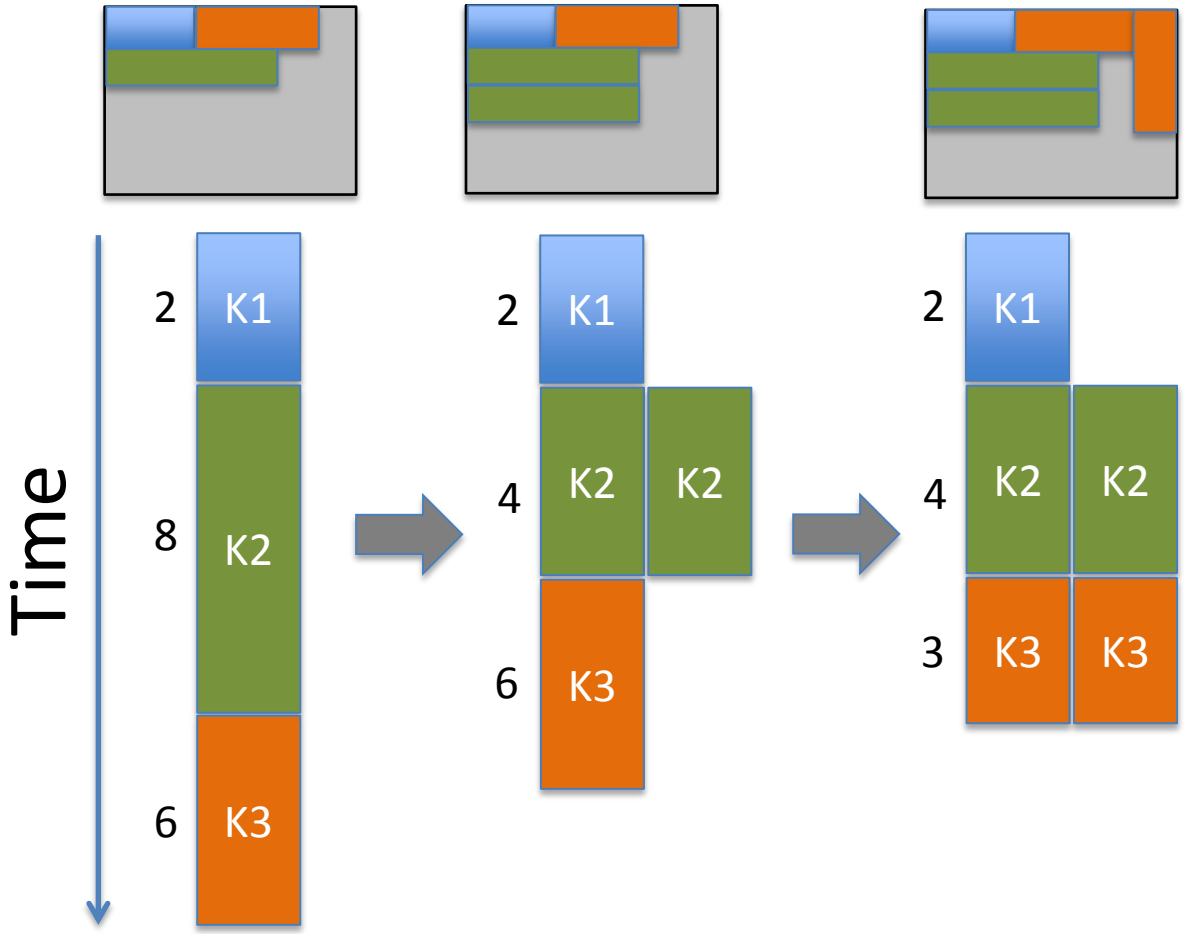
Analytical Design Exploration

nQuanta = 8 ➔ Max cores = (4,2,4)



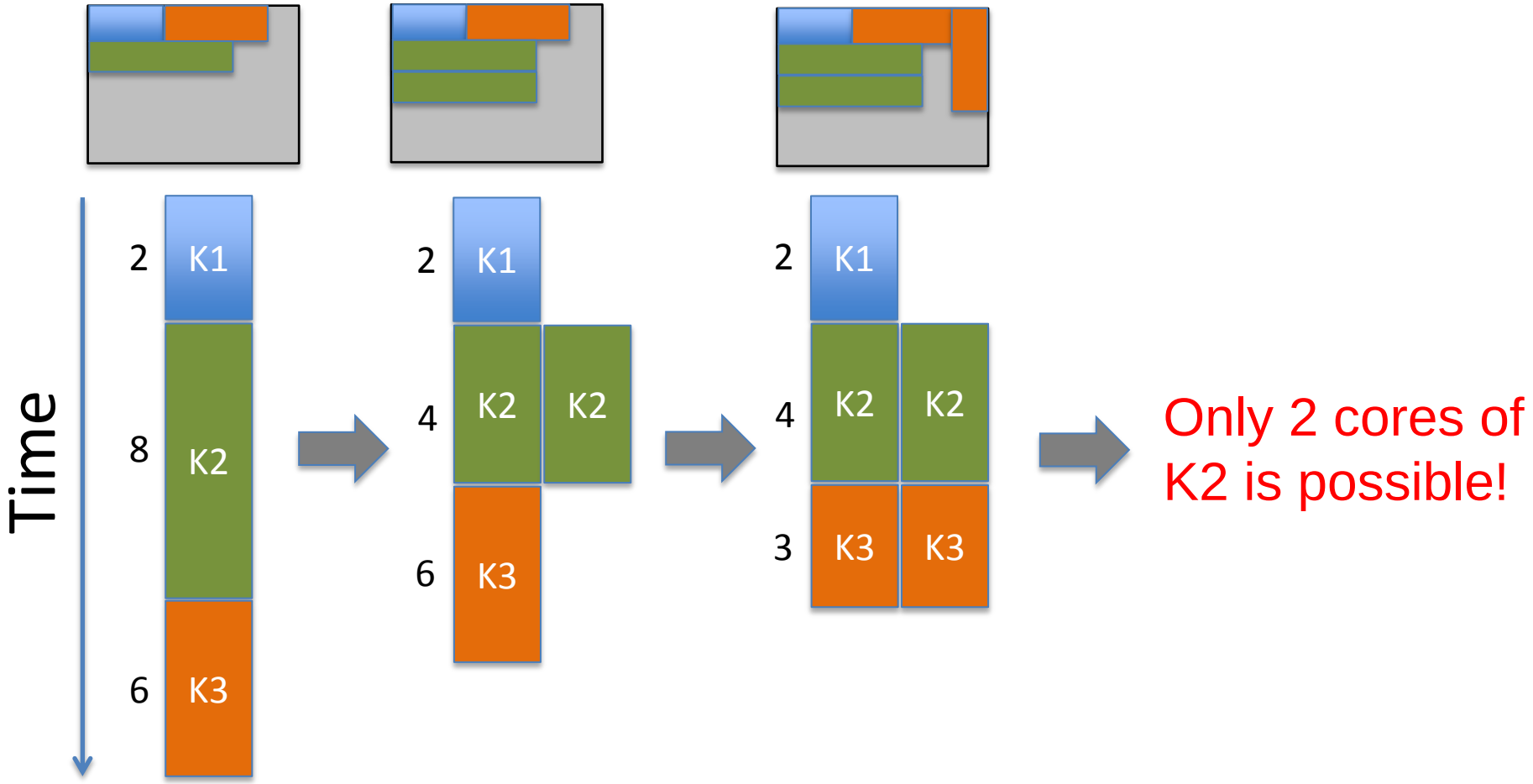
Analytical Design Exploration

nQuanta = 8 ➔ Max cores = (4,2,4)



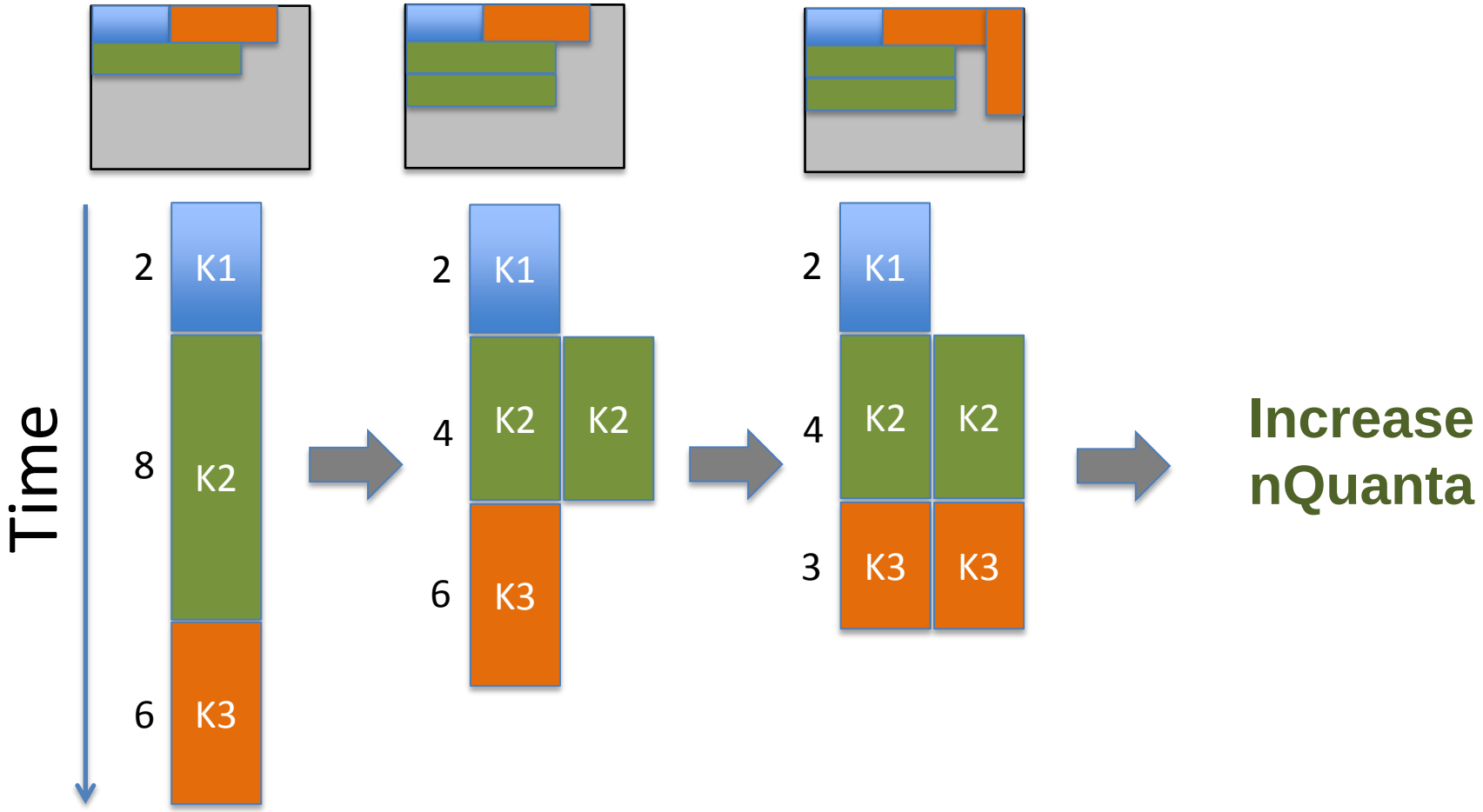
Analytical Design Exploration

nQuanta = 8 ➔ Max cores = (4,**2**,4)



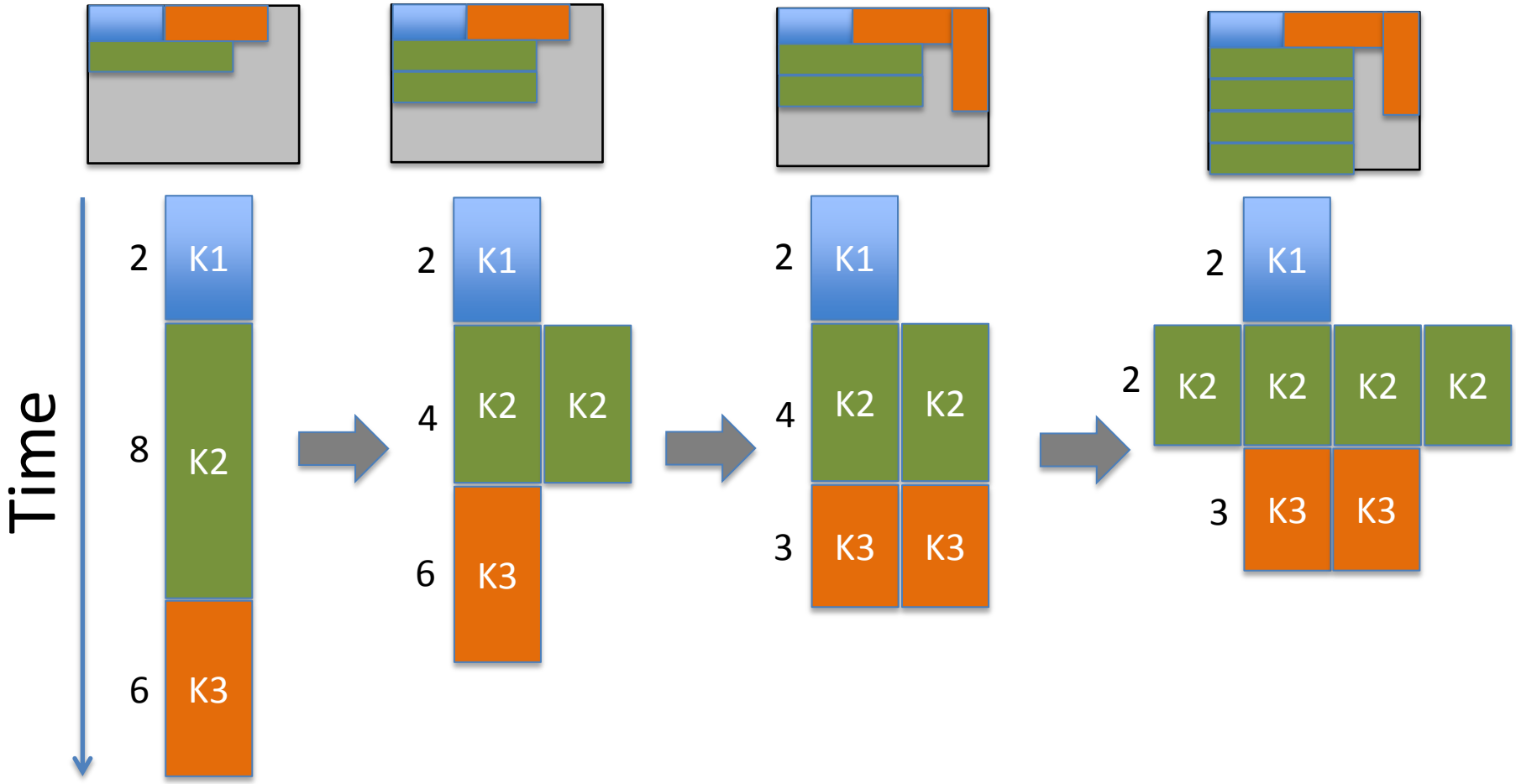
Analytical Design Exploration

nQuanta = 16 ➔ Max cores = (8,4,8)



Analytical Design Exploration

nQuanta = 16 ➔ Max cores = (8,4,8)

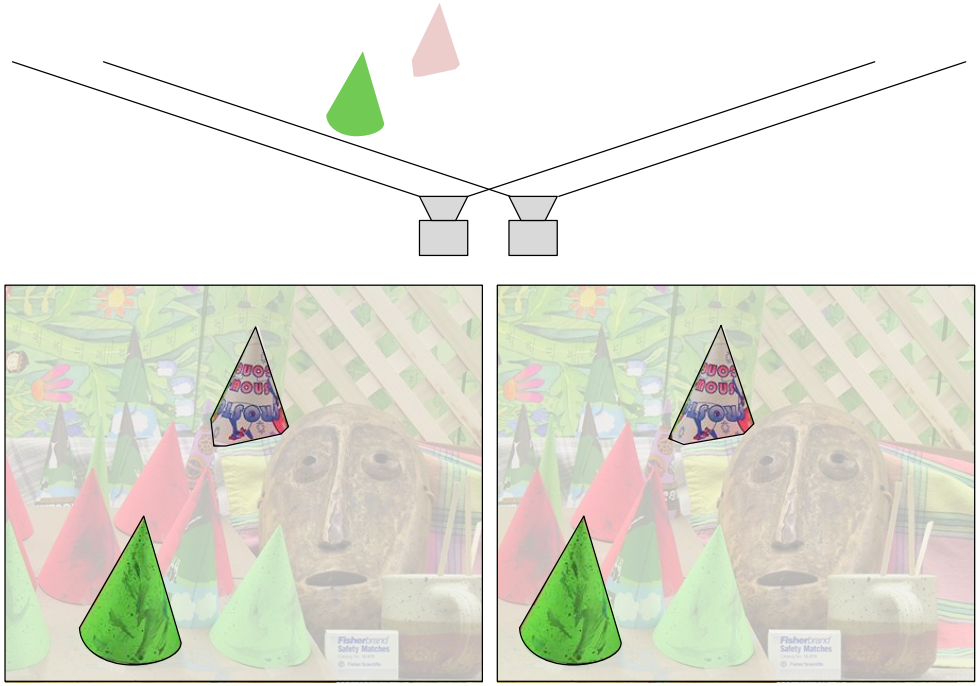


Implementation and Verification

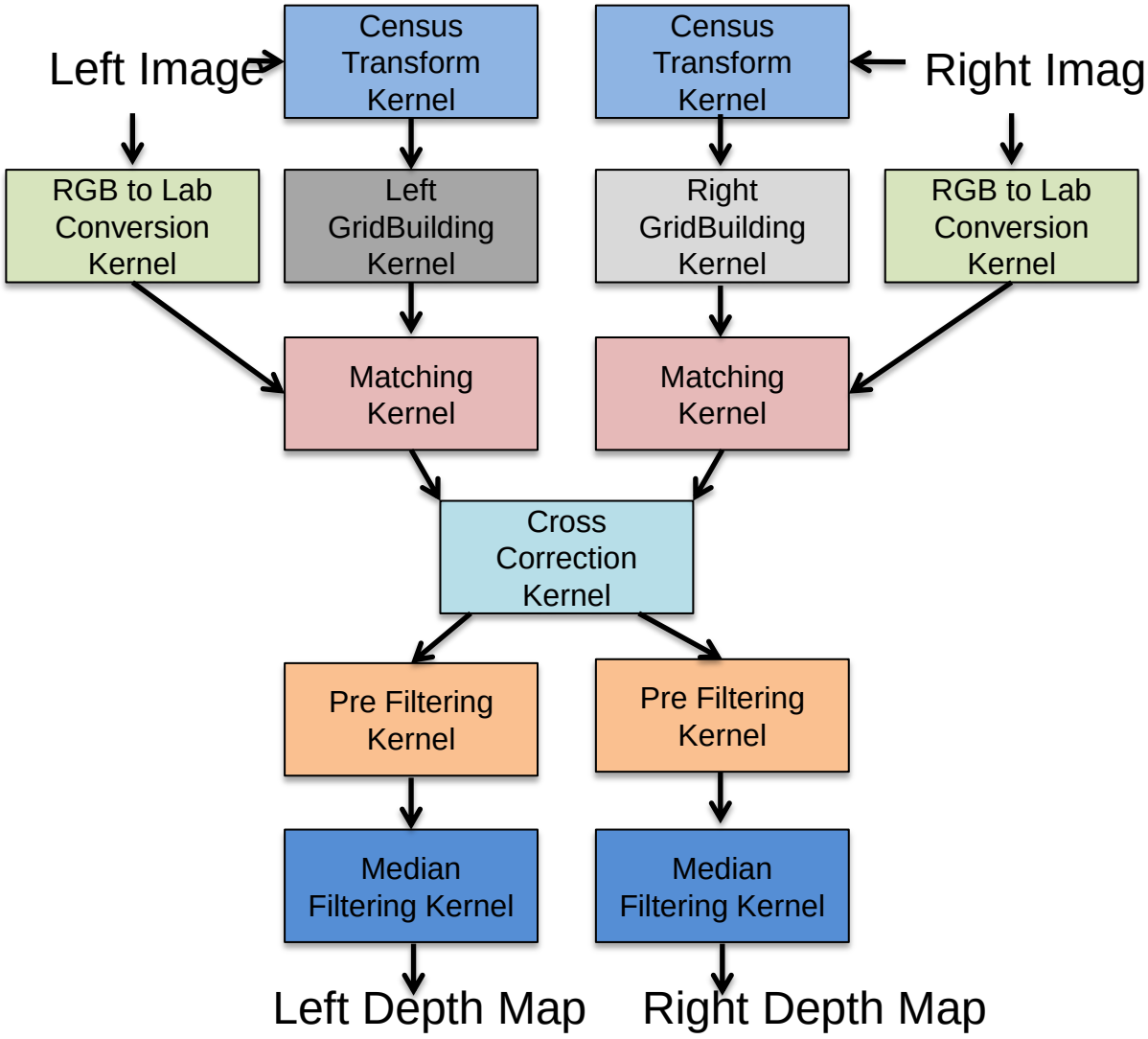
- Core allocations from analytical model
  - AutoPilot-C pragmas for suggested parallelism
- Communication buffers and kernel-level parallelism
- SystemC simulation
- Vivado Synthesis

Stereo Matching

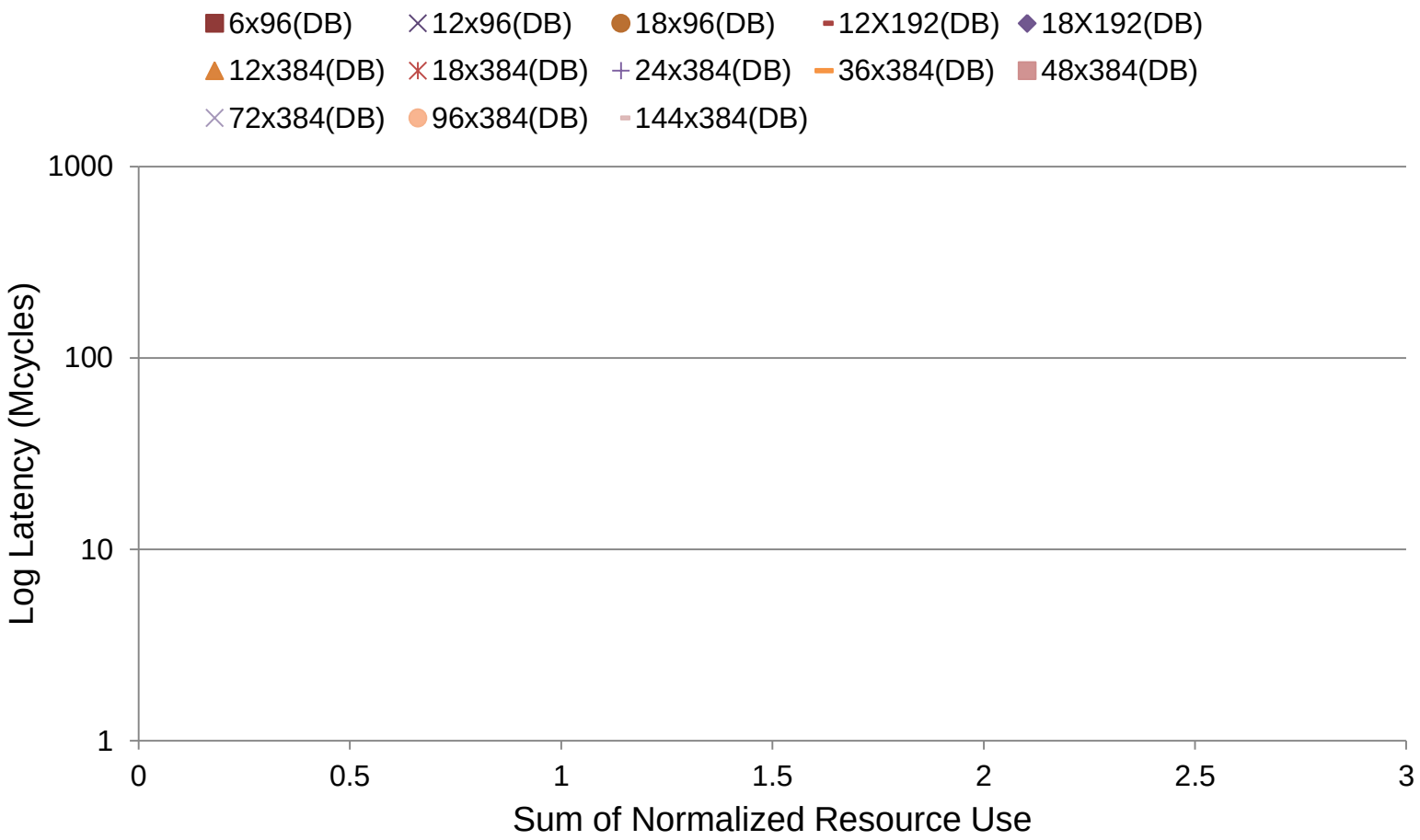
- Two spatially separated color cameras
- Distance in pixels between the same object in the images infers depth
- Complex algorithms to match pixels



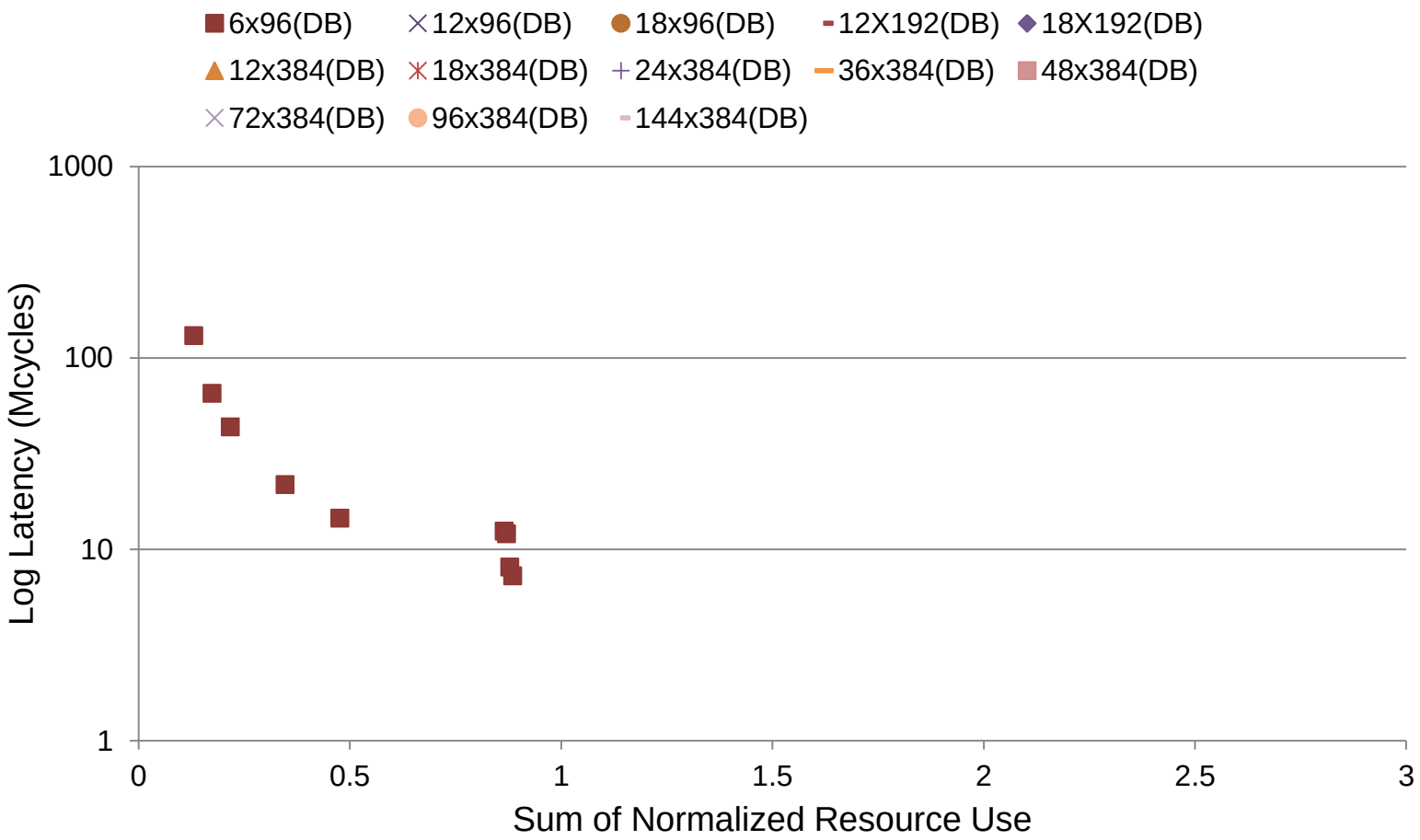
Case Study – Stereo Matcher



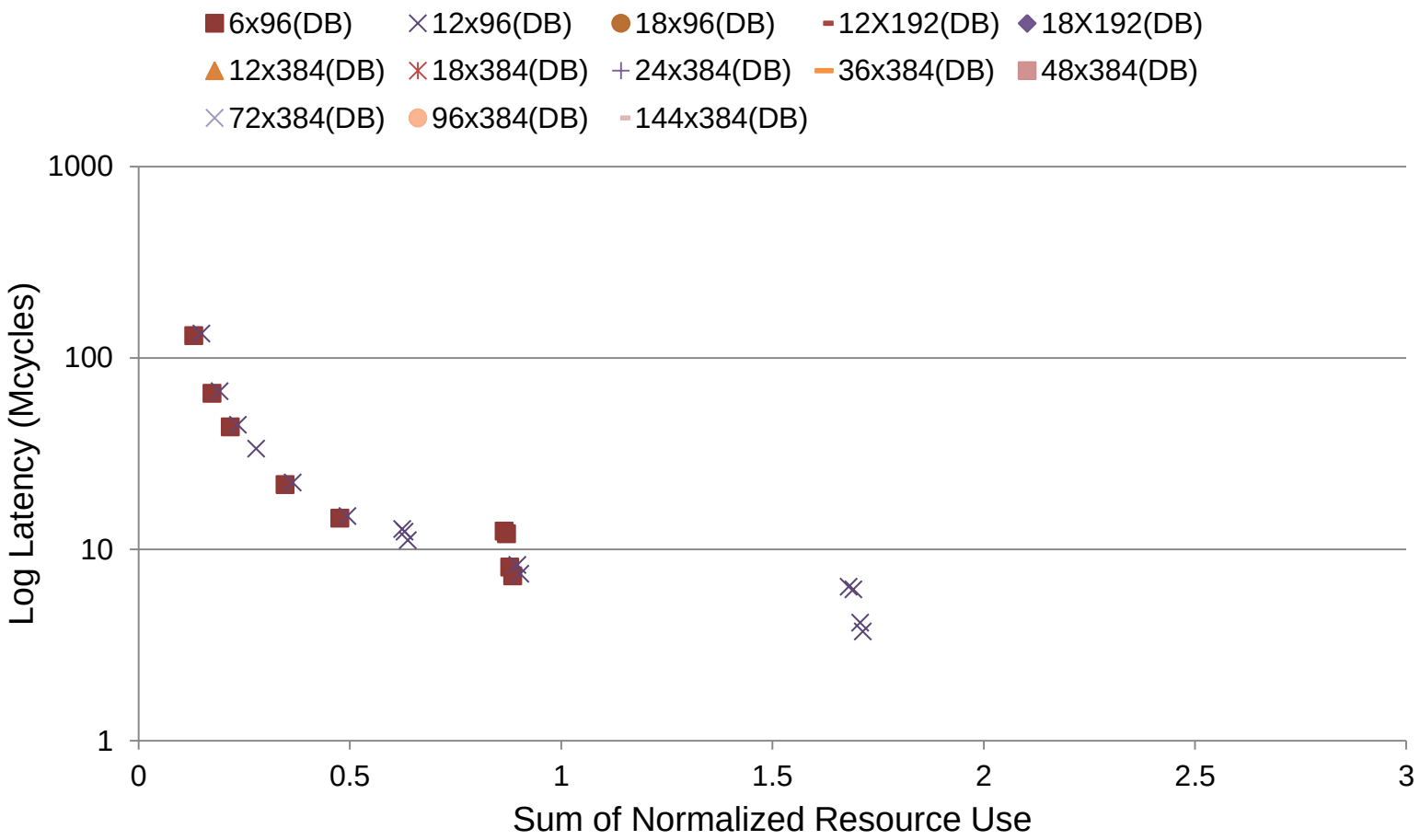
Design Space for Dual-Buffer Flow



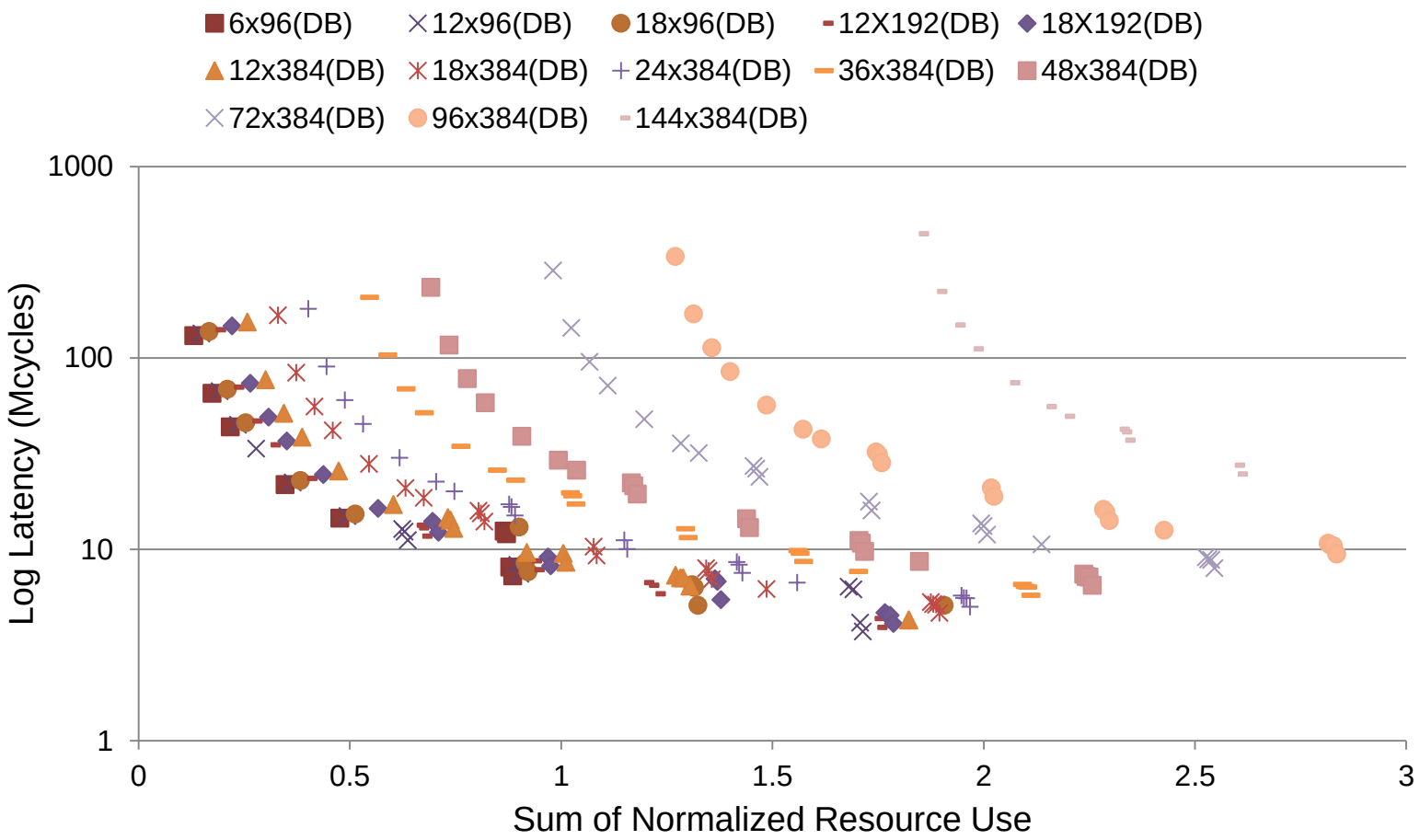
Design Space for Dual-Buffer Flow



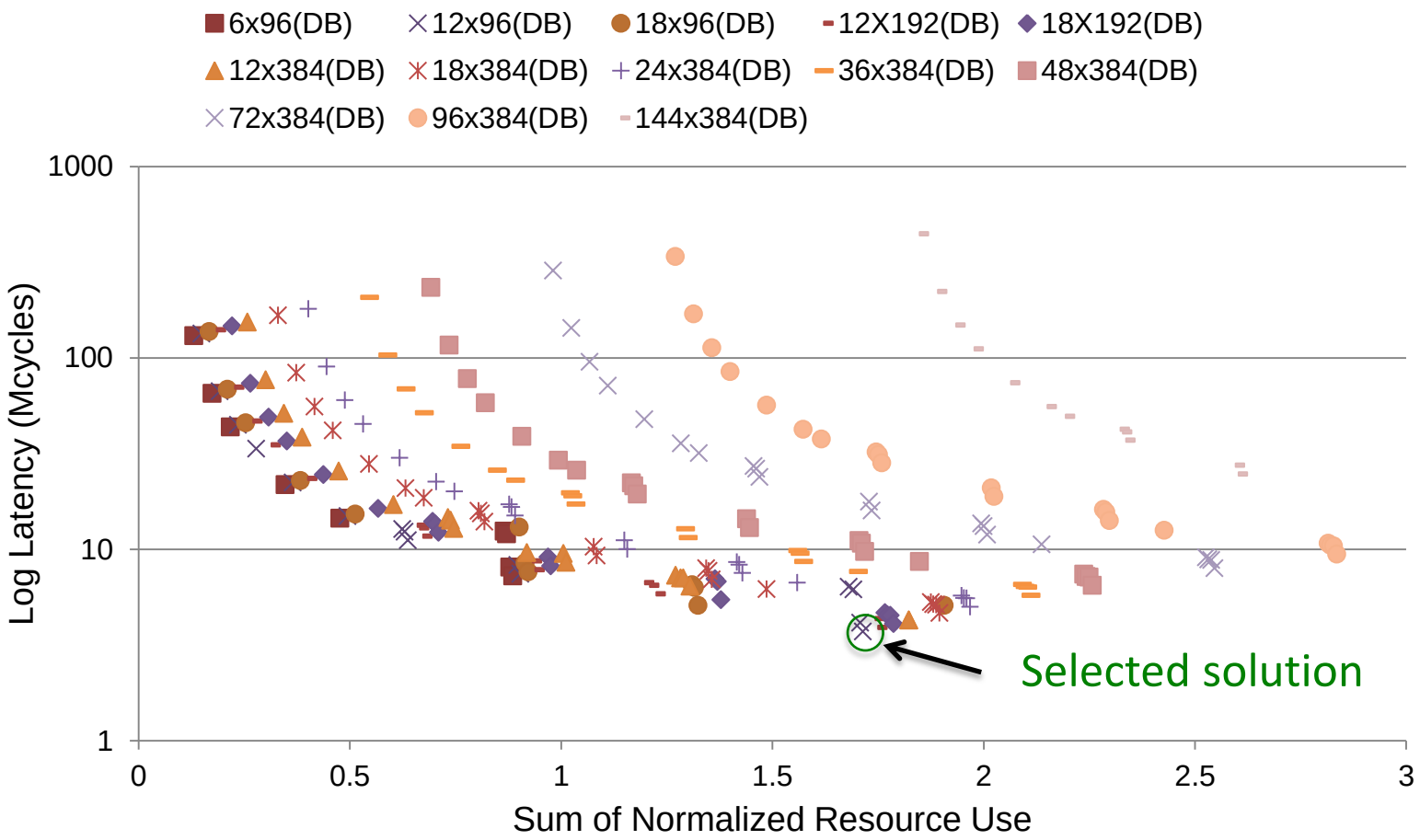
Design Space for Dual-Buffer Flow



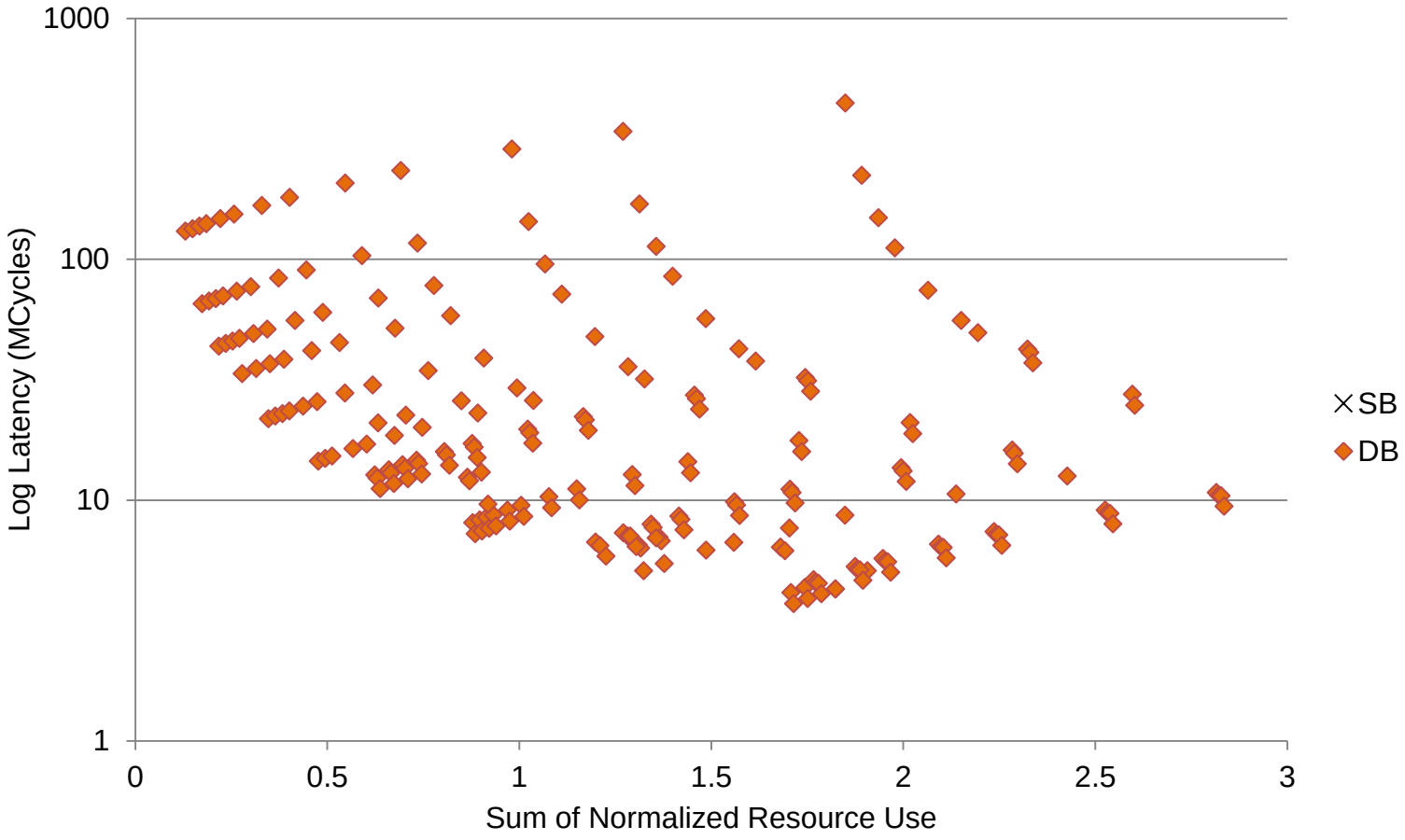
Design Space for Dual-Buffer Flow



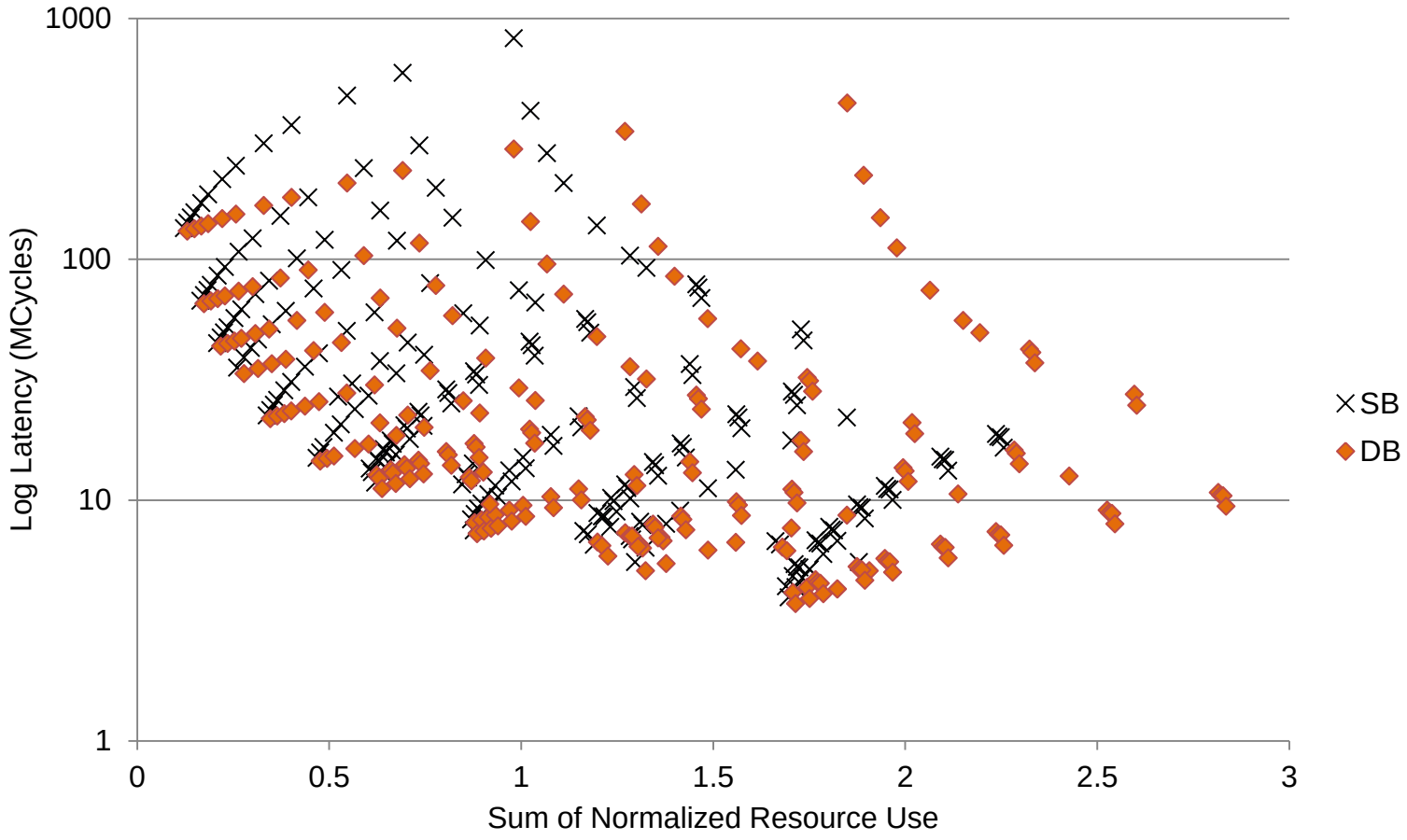
Design Space for Dual-Buffer Flow



Design Space for Dual and Single Buffer Flow

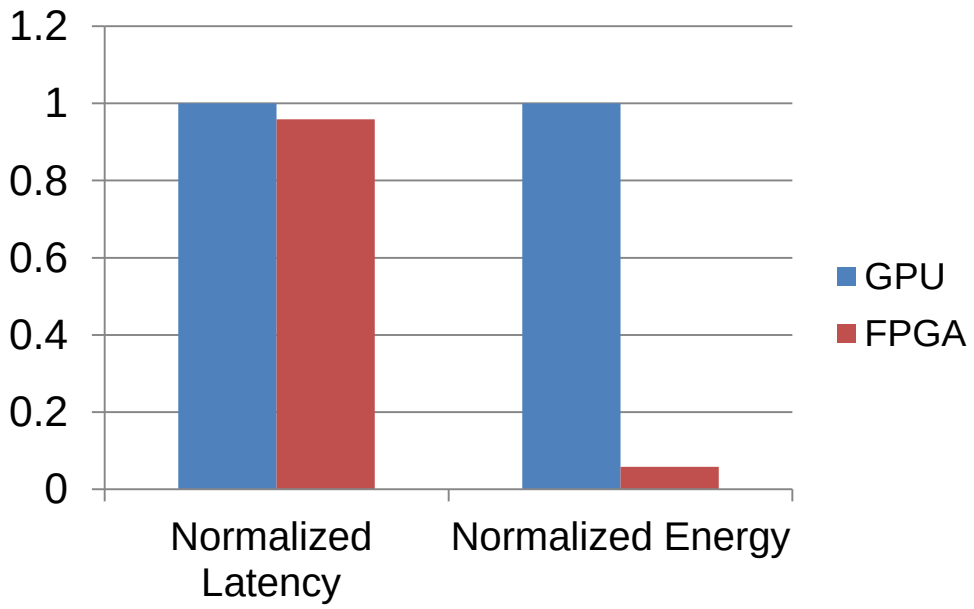


Design Space for Dual and Single Buffer Flow



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

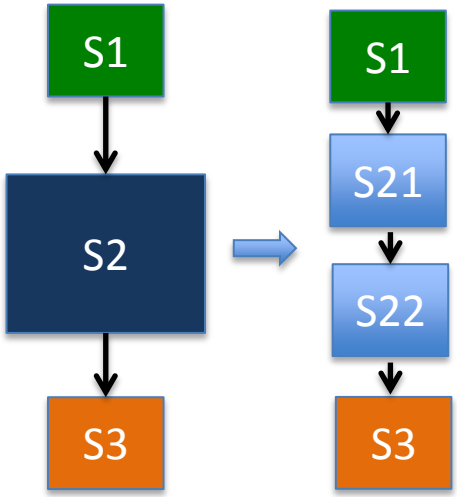
- Single kernel synthesis
  - Critical: Replicating the initial solution for concurrency
- Multi-kernel synthesis

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