



Automatic Generation of Efficient Accelerator Designs for Reconfigurable Hardware

David Koeplinger

Raghu Prabhakar

Yaqi Zhang

Christina Delimitrou

Christos Kozyrakis

Kunle Olukotun

Stanford University

ISCA 2016

FPGAs in Data Centers

- Increasing interest in use of FPGAs as application accelerators in data centers

Microsoft[®]

 **Bing**

Baidu  **百度**

intel[®]

ALTERA[®]
now part of Intel

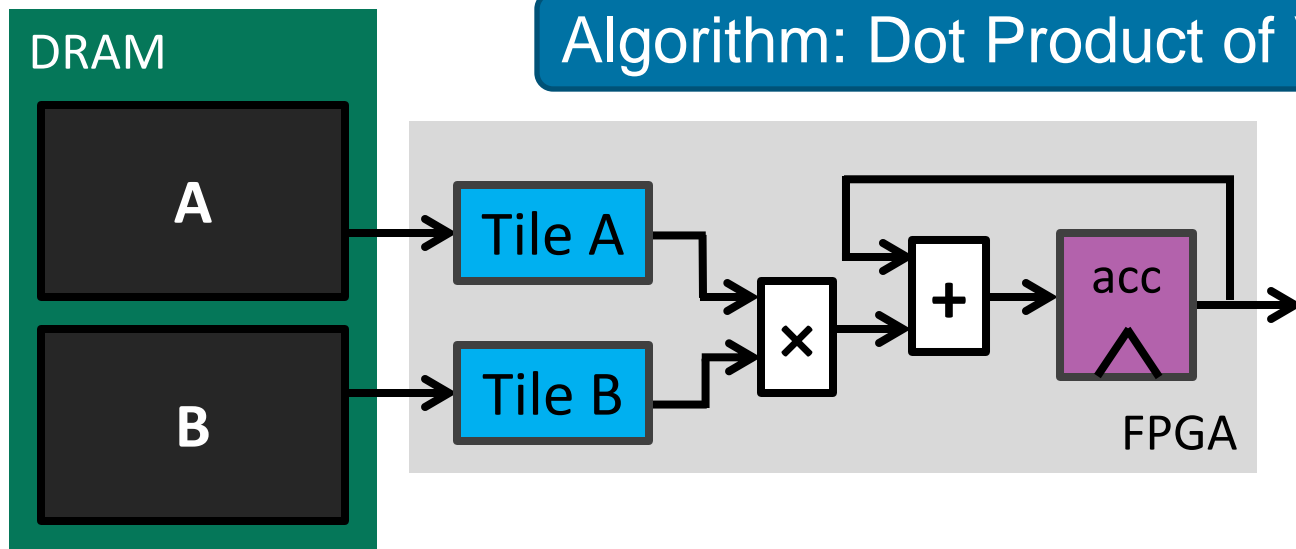
Key advantage: **Performance/Watt**

Problem: Large Design Spaces

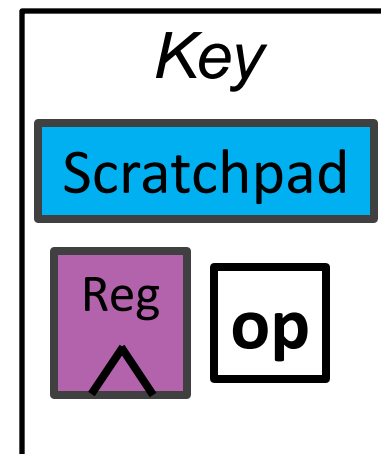
- Design spaces grow exponentially with the number of parameters
- Even relatively small designs can have very large spaces
- Parameters can change runtime by orders of magnitude
- Parameters typically aren't independent
- Manual exploration is tedious, may result in suboptimal designs

Design Space Example: Dot Product

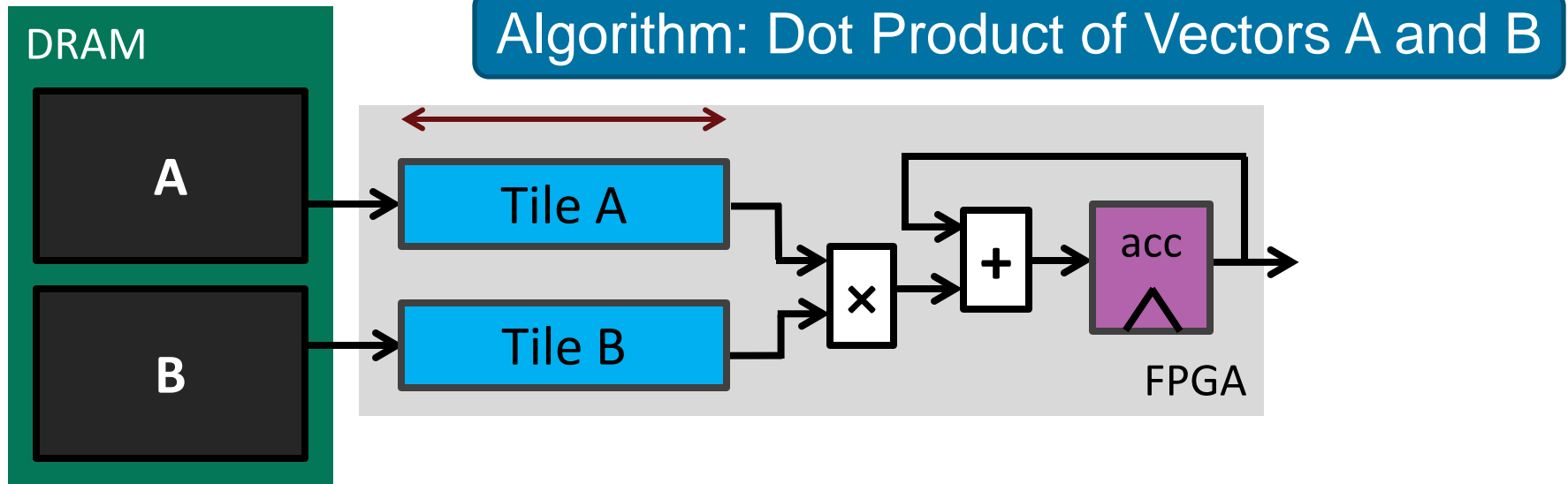
Algorithm: Dot Product of Vectors A and B






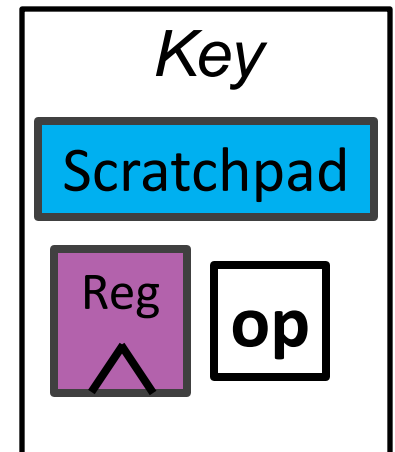
Small and simple, but slow!



Important Parameters: Tile Sizes

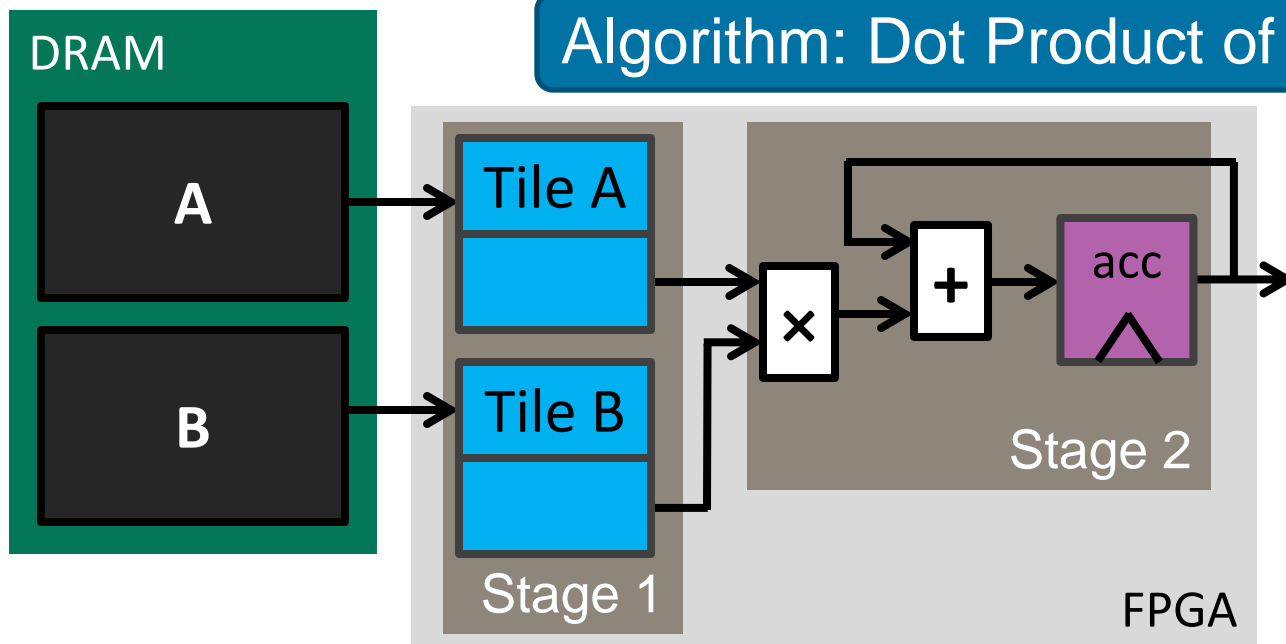





- Increases length of DRAM accesses  Runtime
- Increases exploited spatial locality  Runtime
- Increases local memory sizes  Area

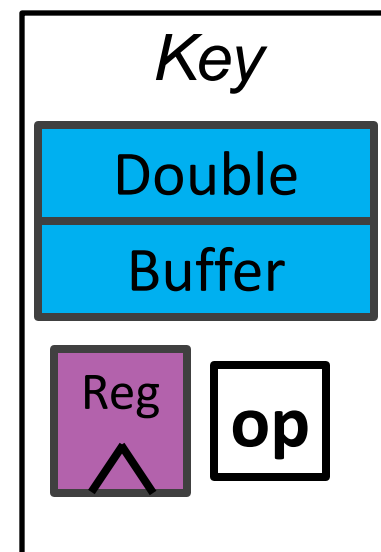


Important Parameters: Pipelining

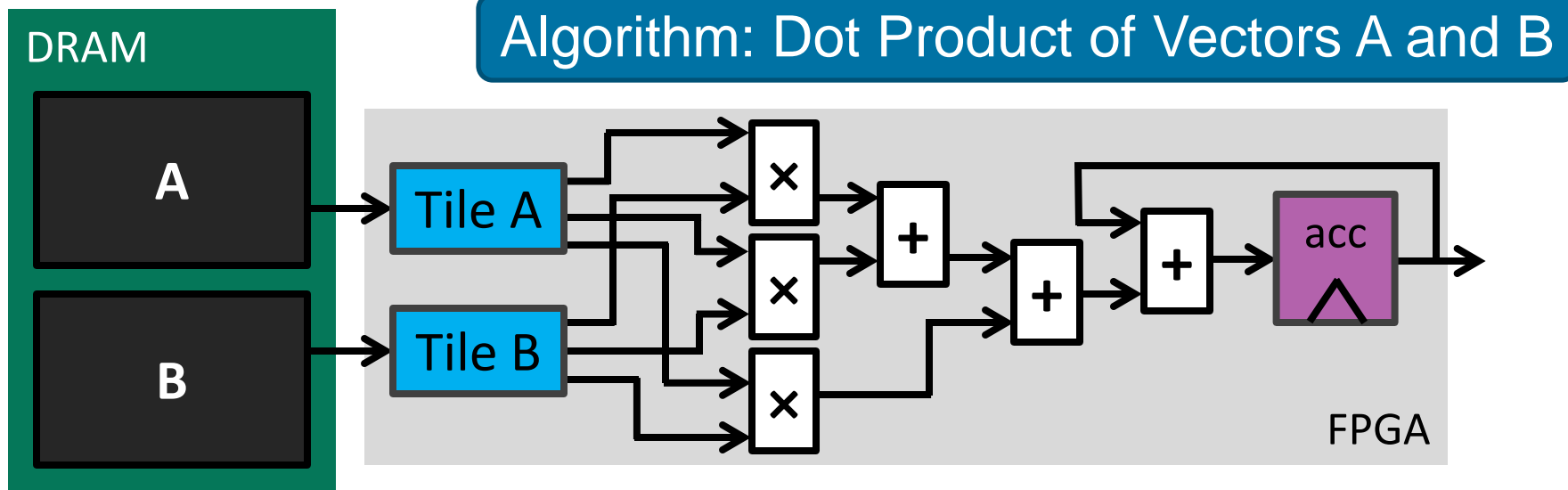
Algorithm: Dot Product of Vectors A and B





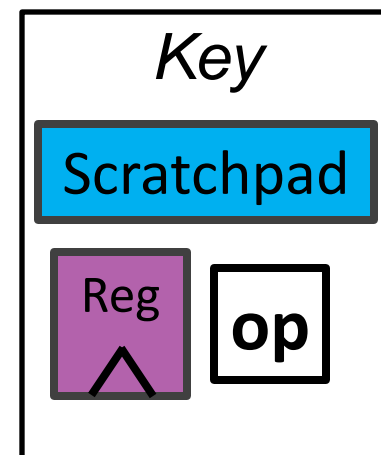
- Overlaps memory and compute  Runtime
- Increases local memory sizes  Area
- Adds synchronization logic  Area



Important Parameters: Parallelization



- Improves element throughput  Runtime
- Duplicates compute resources  Area



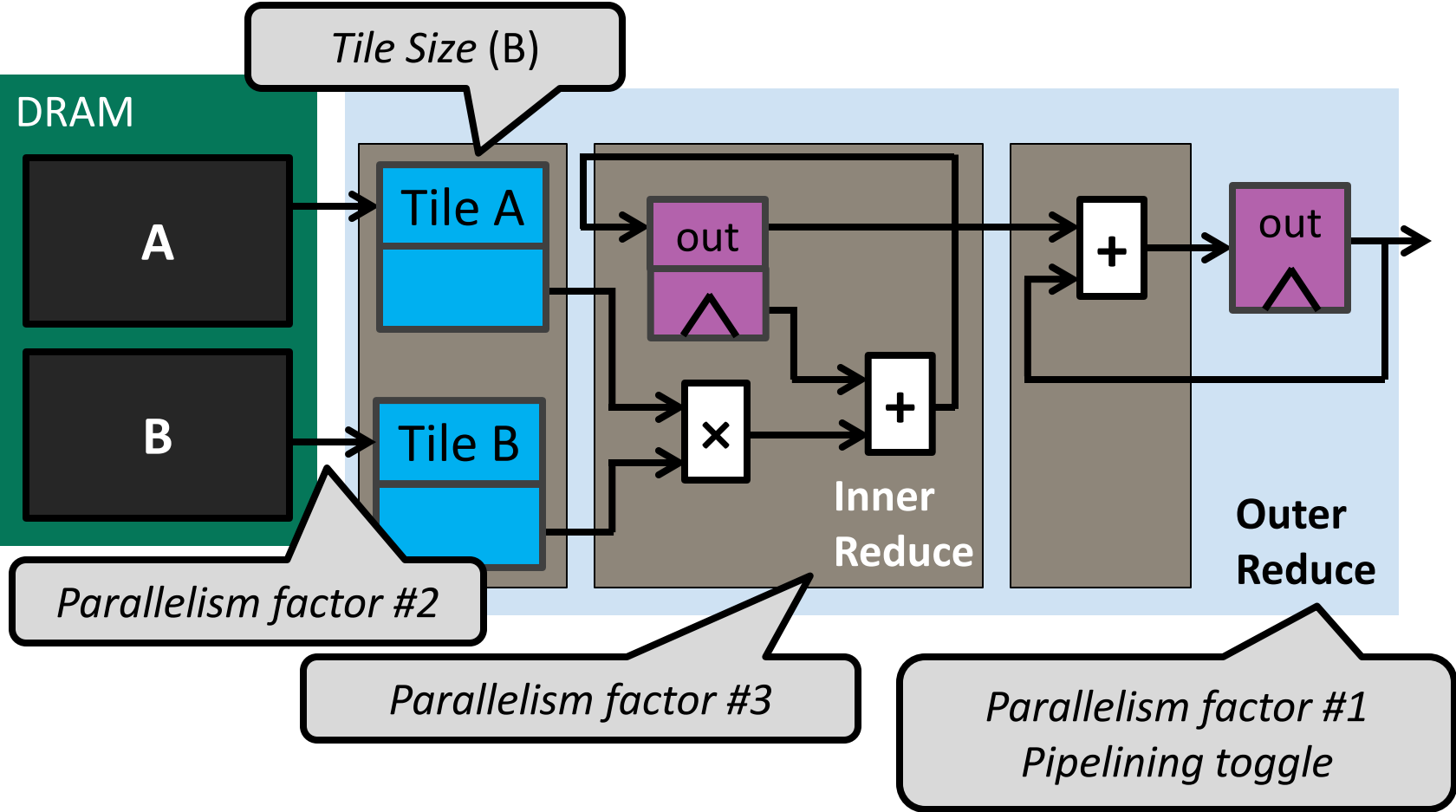
Language/Tool Requirements

	VHDL Verilog	LegUp	Vivado HLS OpenCL SDK	Aladdin	DHDL
Targets FPGAs	✓	✓	✓	✗	✓
Enables pipelining at arbitrary loop levels	✓	✗	✗	✗	✓
Exposes design parameters to the compiler	✗	✗	✗	✓	✓
Evaluates designs prior to synthesis	✗	✓	✓	✓	✓
Explores design space automatically	✗	✗	✗	✓	✓
Generates synthesizable code	✓	✓	✓	✗	✓

Delite Hardware Definition Language

- Includes a variety parameterized templates
 - **Parallel patterns** with implicit parallelization factors
 - **Pipeline constructs** for pipelining at arbitrary levels
 - **Explicit size parameters** for loop step size and buffer sizes
- All parameters are exposed to compiler
- Compiler includes latency and area models for quick design evaluation
- Compiler automatically explores design space
- Generates synthesizable MaxJ HGL after exploration

Dot Product DHDL Diagram



Dot Product in DHDL

```
val output    = Reg[Float]
val vectorA   = OffChipMem[Float](N)
val vectorB   = OffChipMem[Float](N)
```

```
Reduce(N by B)(output) { i =>
  val tileA = Scratchpad[Float](B)
  val tileB = Scratchpad[Float](B)
  val acc    = Reg[Float]
```

```
1  tileA load vectorA(i :: i+B)
   tileB load vectorB(i :: i+B)
```

```
2  Reduce(B by 1)(acc) { j =>
    tileA(j) * tileB(j)
  }{a, b => a + b}
  }{a, b => a + b}
```

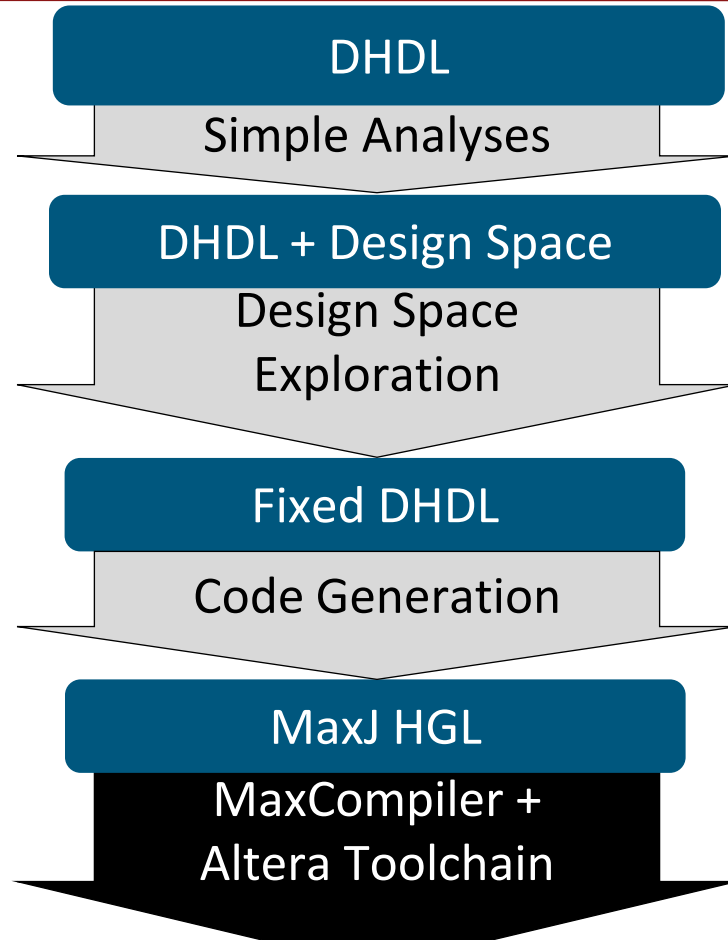
Parallelism factor #1
Pipelining toggle

Tile Size (B)

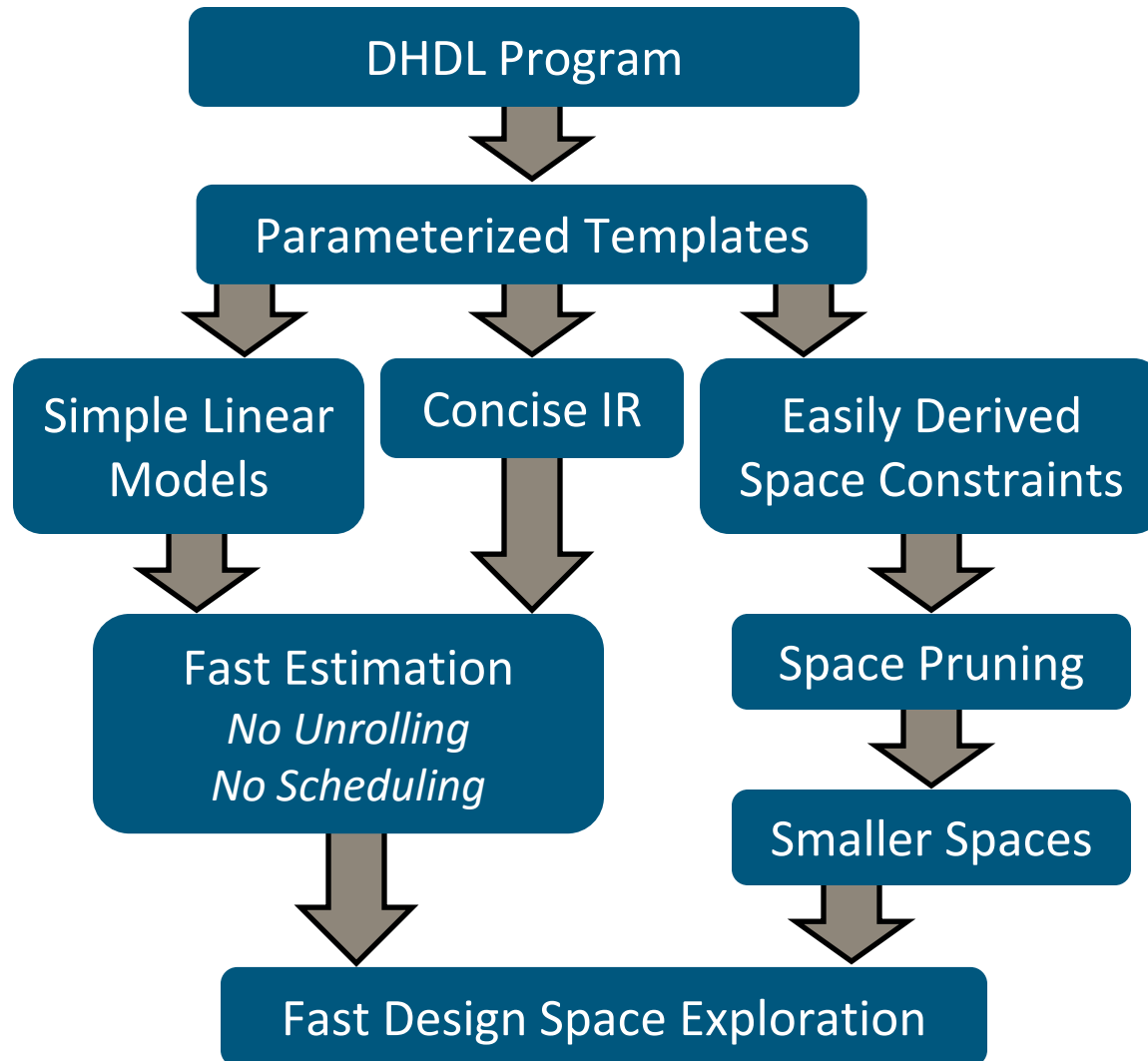
Parallelism factor #2

Parallelism factor #3

DHDL to Hardware



DHDL Enables Fast DSE



Latency Modeling

- Analytical model
 - Uses depth-first search to get critical path of pipelines
 - Accurate estimation requires data size annotations
- Main-memory model
 - Mathematical model fit to observed runtimes
 - Parameterized by:
 - Number of contending readers/writers
 - Number of commands issued in sequence
 - Command length

Area Modeling

- Analytical model
 - Simple summation of area of each template
 - Includes estimates for delay lines, banked memories
- Neural network models
 - Models routing costs and memory duplication
 - Simple, 3 layer networks suffice here (we use 11-6-1)
 - Trained on about set of 200 characterization designs
- Total area = analytical area + neural net area

Evaluation

- **Accuracy:**

How accurate are the models, compared to observations?

- **Speed:**

How fast are the predictions, compared to commercial tools?

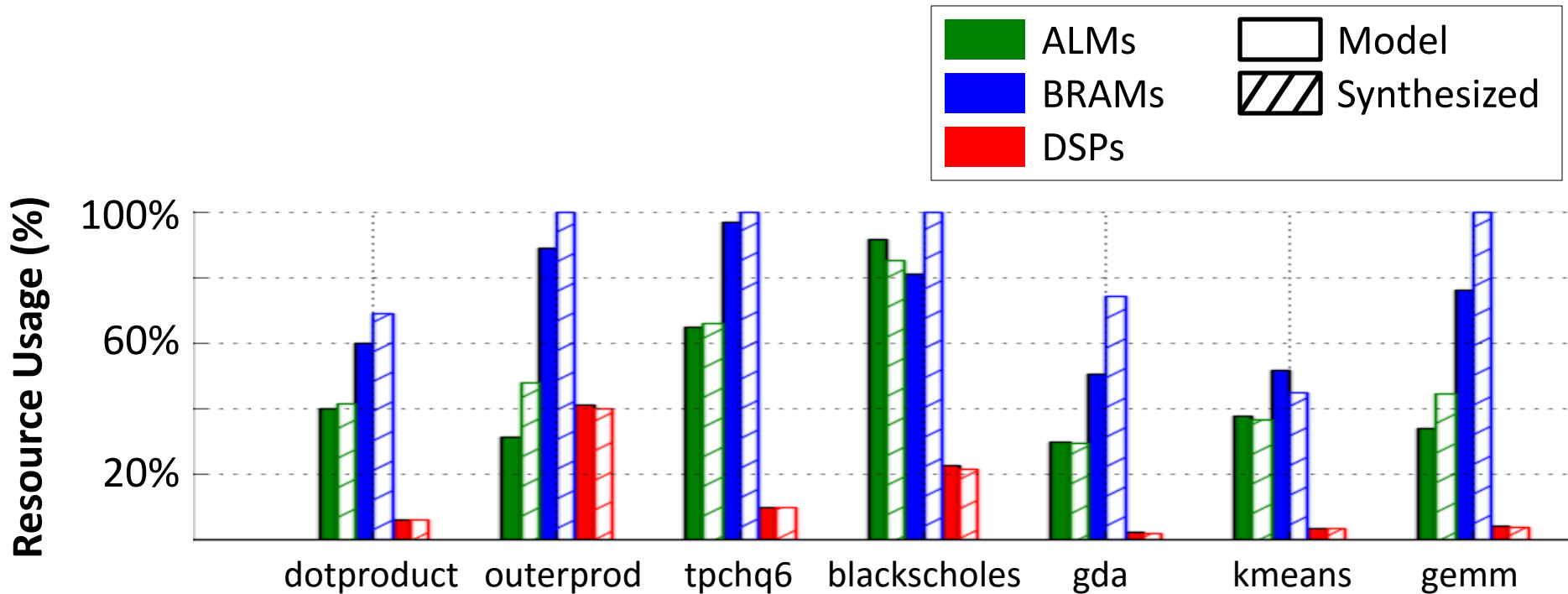
- **Space:**

Do the design parameters help capture an interesting space?

- **Performance:**

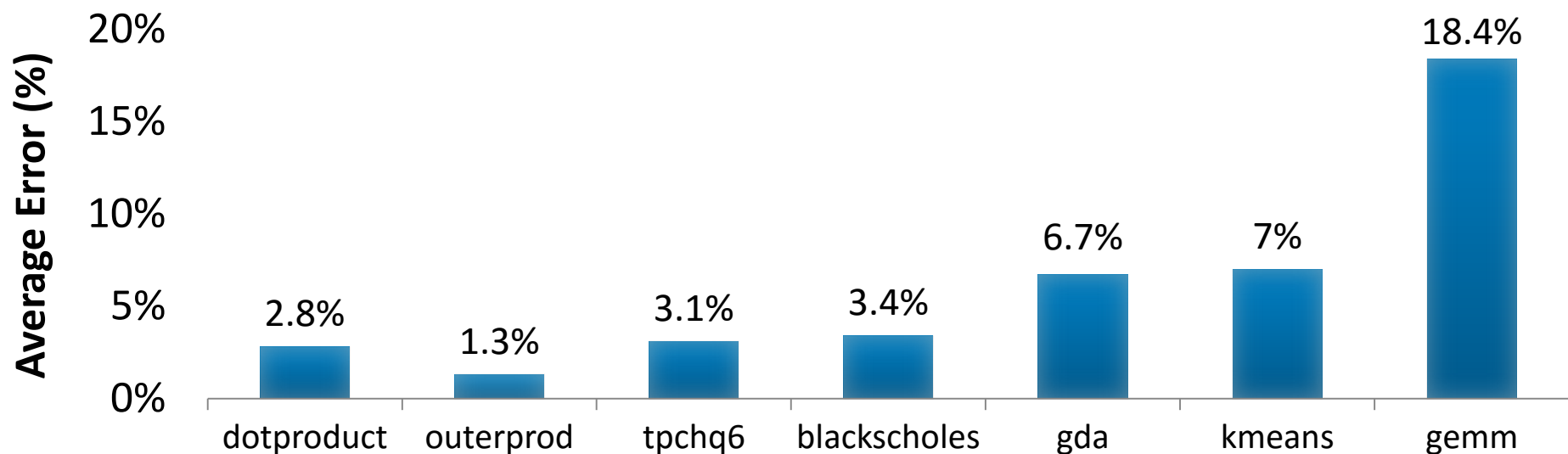
How good is the best generated design?

Results: Model Accuracy (Area)



**Area models follow important trends
and are accurate enough to drive
automatic design space exploration**

Results: Model Accuracy (Latency)

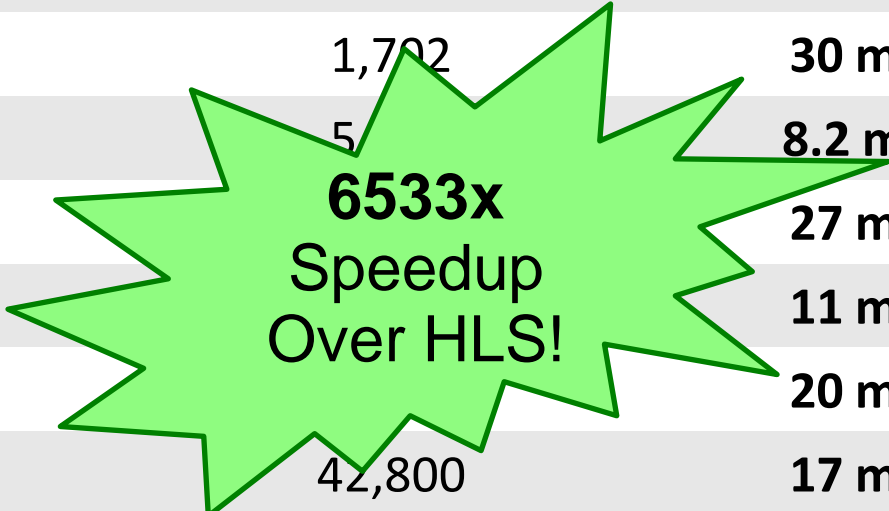


**Latency models follow important trends
and are accurate enough to drive
automatic design space exploration**

Results: Prediction Speed

DHDL:

Benchmark	Designs	Search Time
Dot Product	5,426	5.3 ms / design
Outer Product	1,702	30 ms / design
TPCHQ6	5	8.2 ms / design
Blackscholes		27 ms / design
Matrix Multiply		11 ms / design
K-Means		20 ms / design
GDA	42,800	17 ms / design

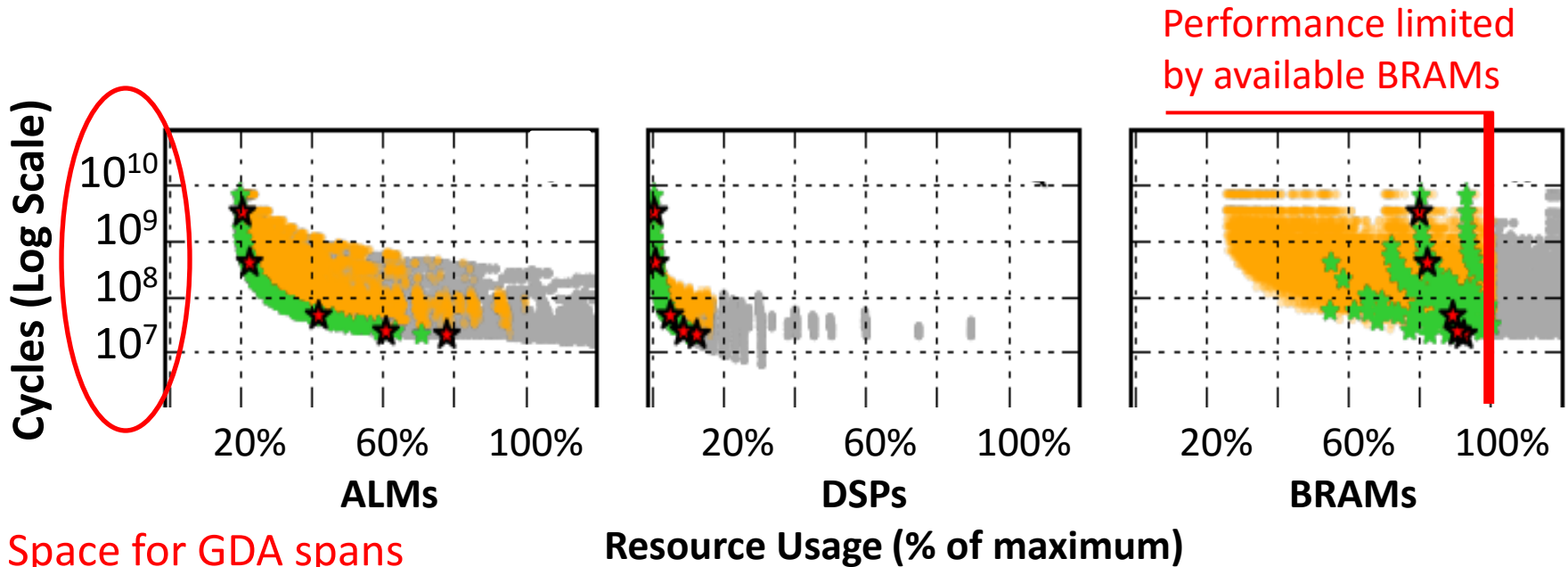


**6533x
Speedup
Over HLS!**

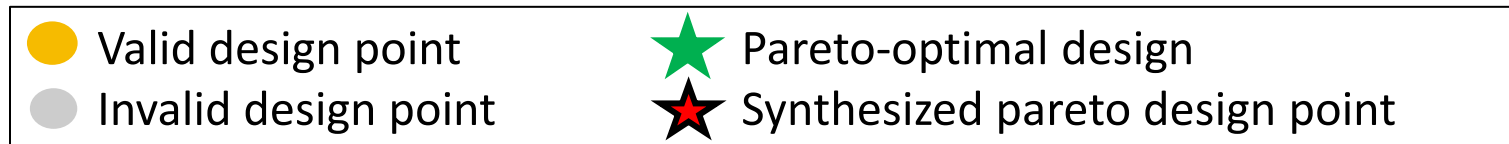
Vivado HLS:

	Designs	Search Time
GDA	250	1.85 min / design

Results: GDA Design Space



Space for GDA spans
four orders of magnitude



Evaluation: Multi-Core Comparison

■ FPGA

- Altera Stratix V (28 nm)
- 150 MHz clock
- Peak main memory bandwidth of 37.5 GB/sec

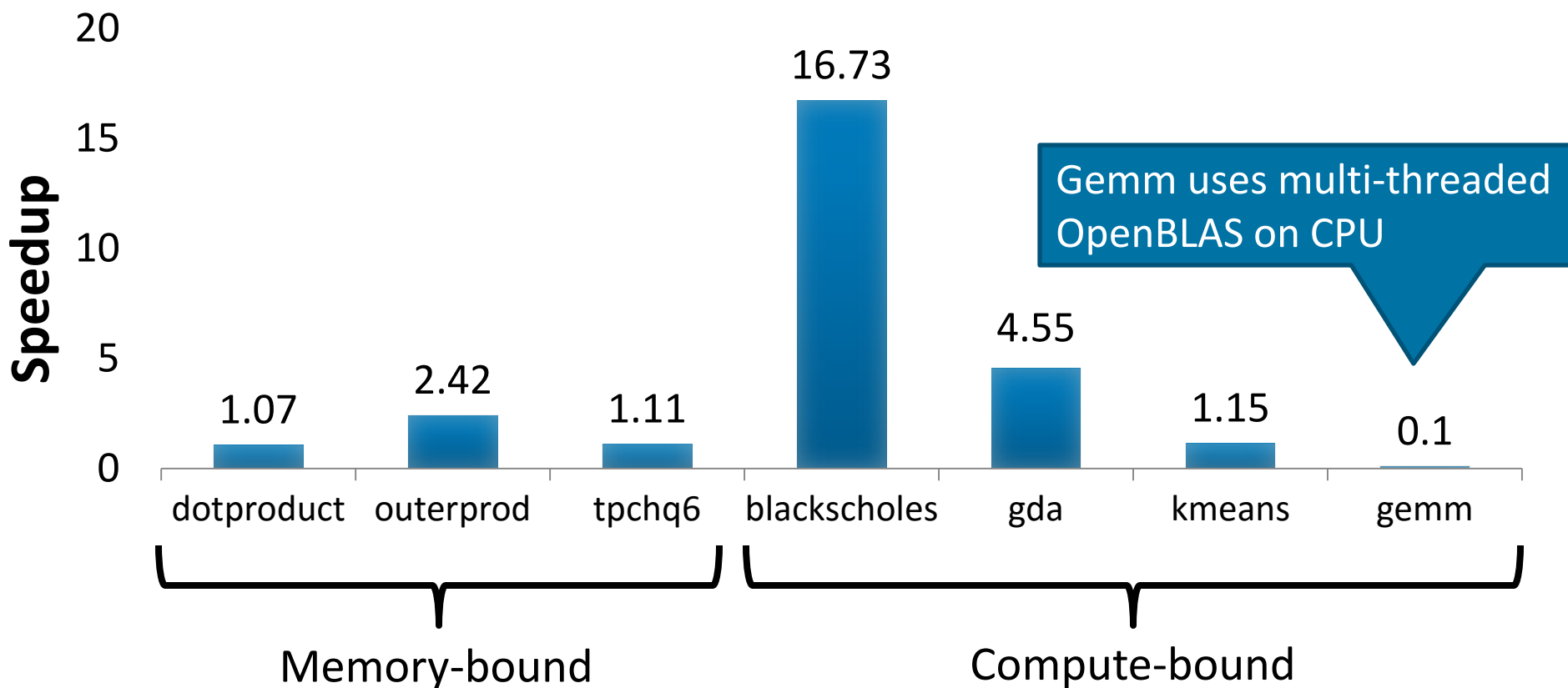
■ Multi-core CPU

- Intel Xeon E5-2630 (32nm)
- 2.3 GHz
- Peak main memory bandwidth of 42.6 GB/sec
- 6 cores, 6 threads
- Multi-threaded C++ code generated from Delite

■ Execution time = ***FPGA execution time***

- Does not include CPU $\leftarrow \rightarrow$ FPGA communication or configuration time

Results: Comparison with Multi-Core

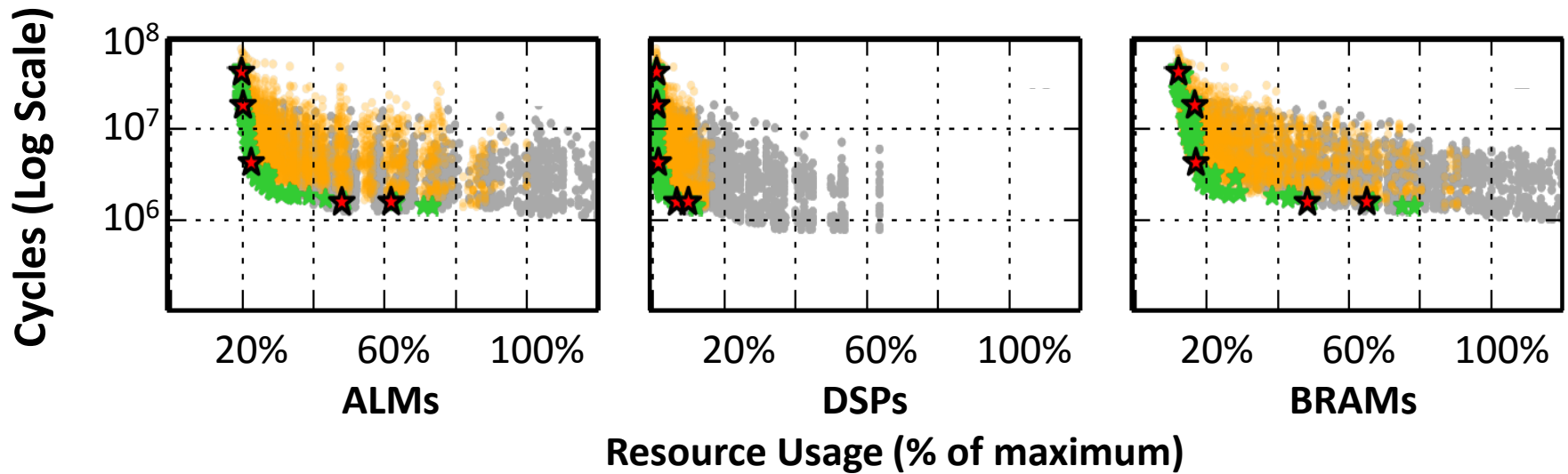


Summary

- **DHDL** exposes large design spaces to the compiler
- Parameterized templates enable **fast, accurate estimators**
- Fast estimators enable **rapid automated DSE**
- Up to **6533x faster** estimation compared to Vivado HLS
- Up to **16.7x speedup** over 6-core CPU



Results: TPCHQ6 Design Space



● Valid design point
● Invalid design point

★ Pareto-optimal design
★ Synthesized pareto design point

Results: Blacksholes Design Space

