DATE 20
DESIGN, AUTOMATION & TEST IN EUROPE
09 – 13 March 2020 · ALPEXPO · Grenoble · France
The European Event for Electronic
System Design & Test

Prospector: Synthesizing Efficient Accelerators via Statistical Learning

Atefeh Mehrabi, Aninda Manocha, Benjamin C. Lee, and Daniel J. Sorin

Department of Electrical and Computer Engineering, Duke University

Towards Specialized Hardware

Towards Specialized Hardware

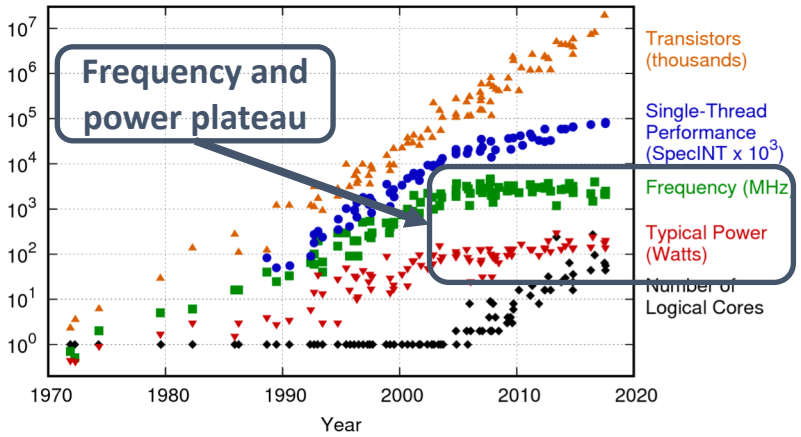
Growing Number of Transistors (Moore's Law)

Power Limits on Chips (End of Dennard Scaling)

Towards Specialized Hardware

Growing Number of Transistors (Moore's Law)

Power Limits on Chips (End of Dennard Scaling)



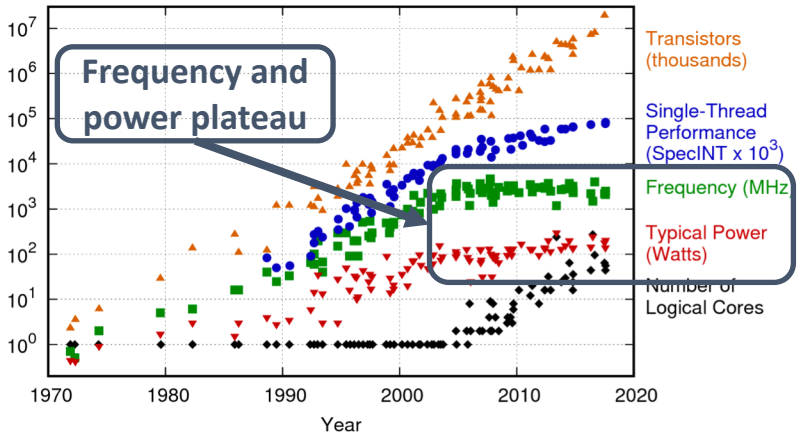
Rupp, "40 Years of Microprocessor Trend Data"

Towards Specialized Hardware

Growing Number of Transistors (Moore's Law)

Power Limits on Chips (End of Dennard Scaling)

Domain-specific Power-efficient Hardware



Rupp, "40 Years of Microprocessor Trend Data"

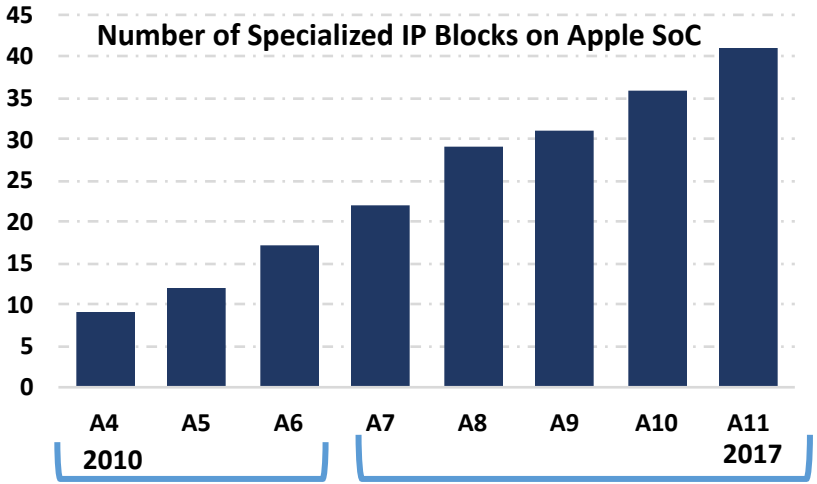
Towards Specialized Hardware

Growing Number of Transistors (Moore's Law)

Power Limits on Chips (End of Dennard Scaling)

Domain-specific Power-efficient Hardware

Accelerator Growth (increased use of specialized blocks)



Challenges of Accelerator Design

Challenges of Accelerator Design

Time
(writing RTL is tedious and time-consuming)

Challenges of Accelerator Design

Time
(writing RTL is tedious and time-consuming)

Cost
(custom circuits are expensive to fabricate)

Challenges of Accelerator Design

Time
(writing RTL is tedious and time-consuming)

Cost
(custom circuits are expensive to fabricate)

Design Effort
(efficient RTL requires digital design expertise)

Challenges of Accelerator Design

Time
(writing RTL is tedious and time-consuming)

Cost
(custom circuits are expensive to fabricate)

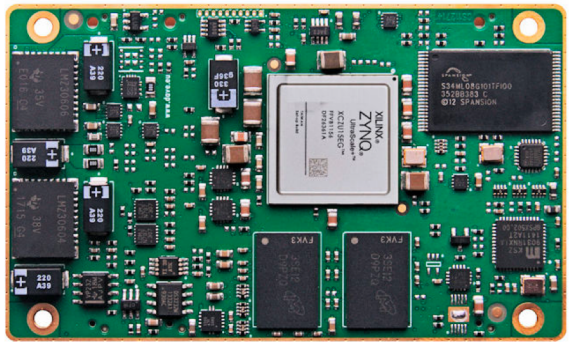
Design Effort
(efficient RTL requires digital design expertise)

Challenges of Accelerator Design

Time
(writing RTL is tedious and time-consuming)

Cost
(custom circuits are expensive to fabricate)

Design Effort
(efficient RTL requires digital design expertise)



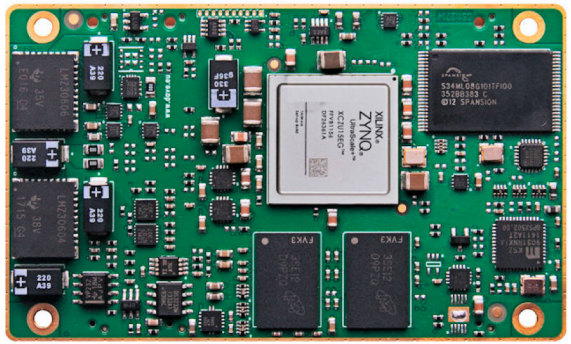
Xilinx Zynq UltraScale+ FPGA

Challenges of Accelerator Design

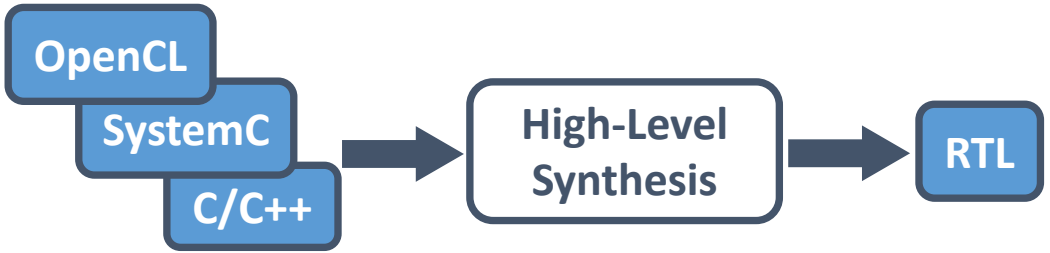
Time
(writing RTL is tedious and time-consuming)

Cost
(custom circuits are expensive to fabricate)

Design Effort
(efficient RTL requires digital design expertise)

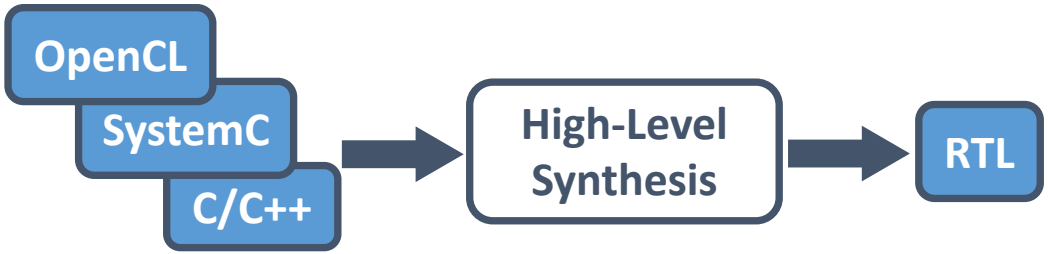


Xilinx Zynq UltraScale+ FPGA

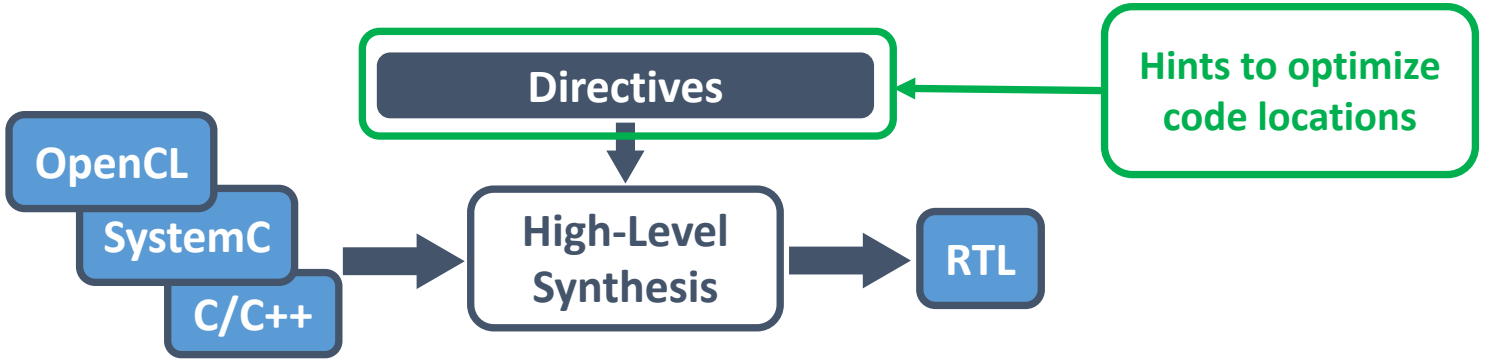


Automatic generation of RTL from high-level language

A Large Accelerator Design Space

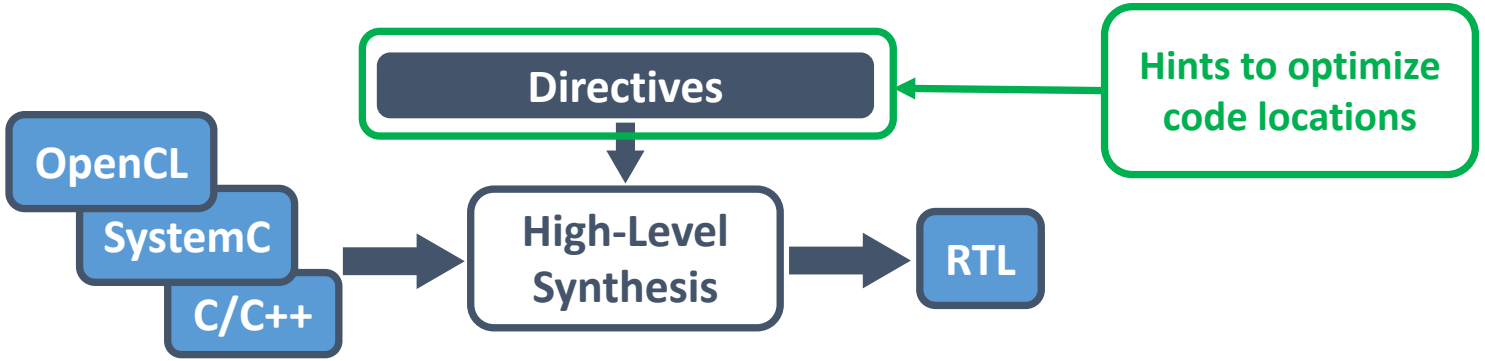


A Large Accelerator Design Space



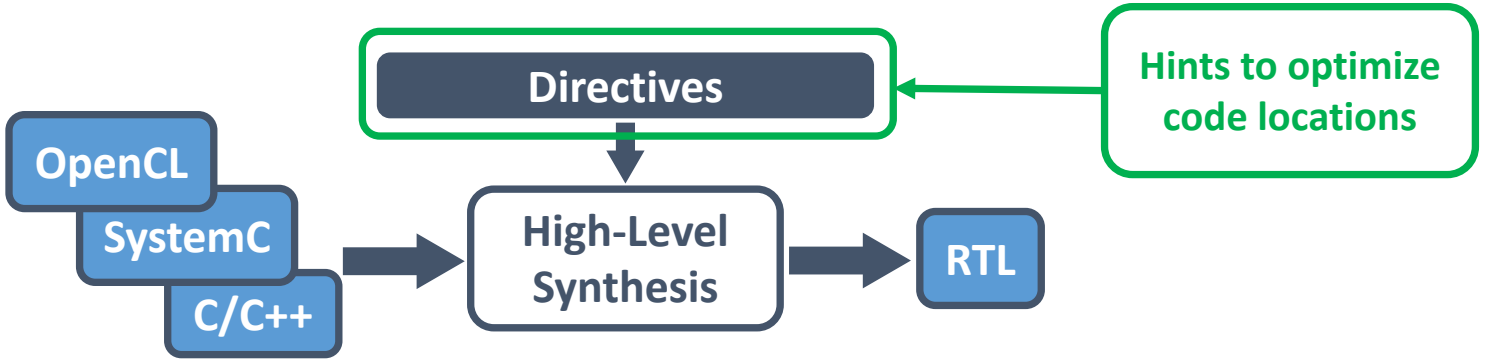
- **Hardware generated by High-level Synthesis can be inefficient.**

A Large Accelerator Design Space



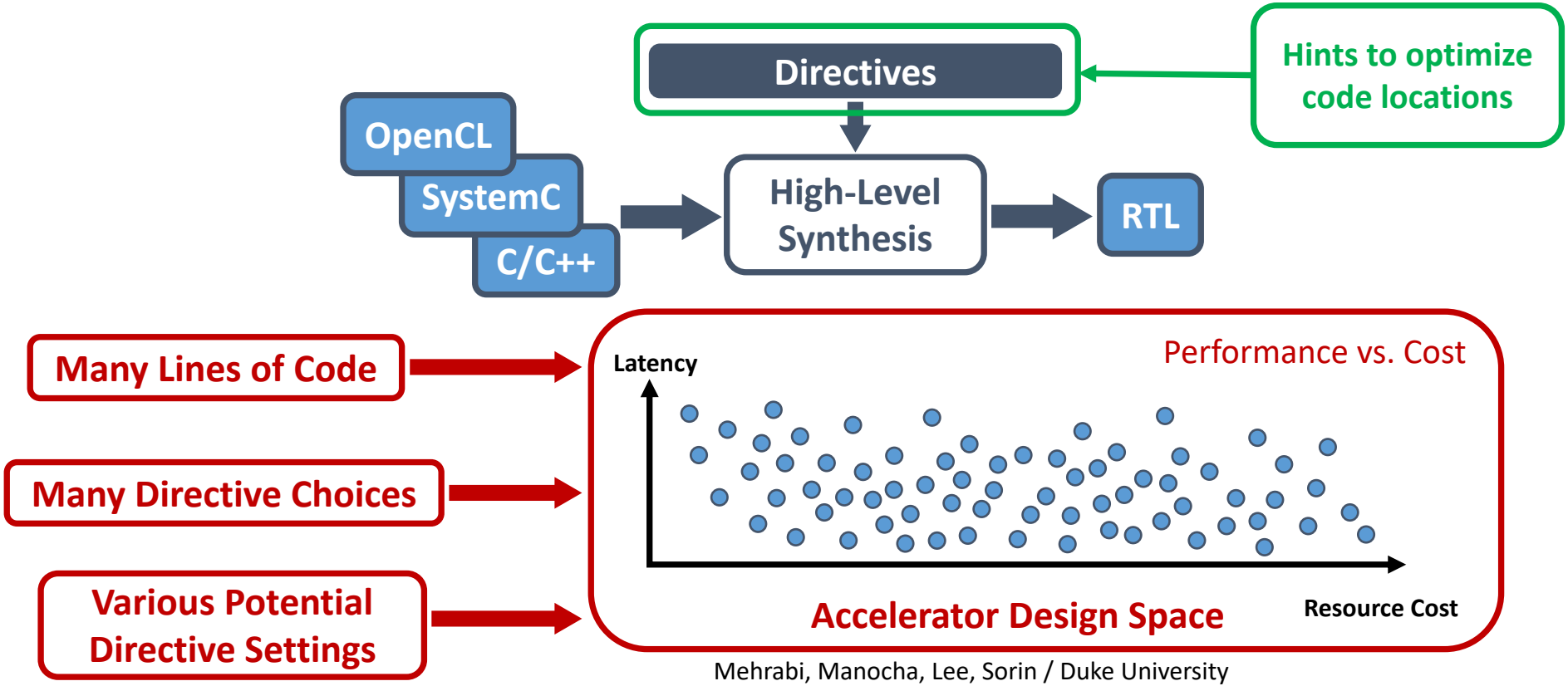
- Hardware generated by High-level Synthesis can be inefficient.
- Directives guide synthesis flow e.g. loop unrolling, array partitioning, etc.

A Large Accelerator Design Space



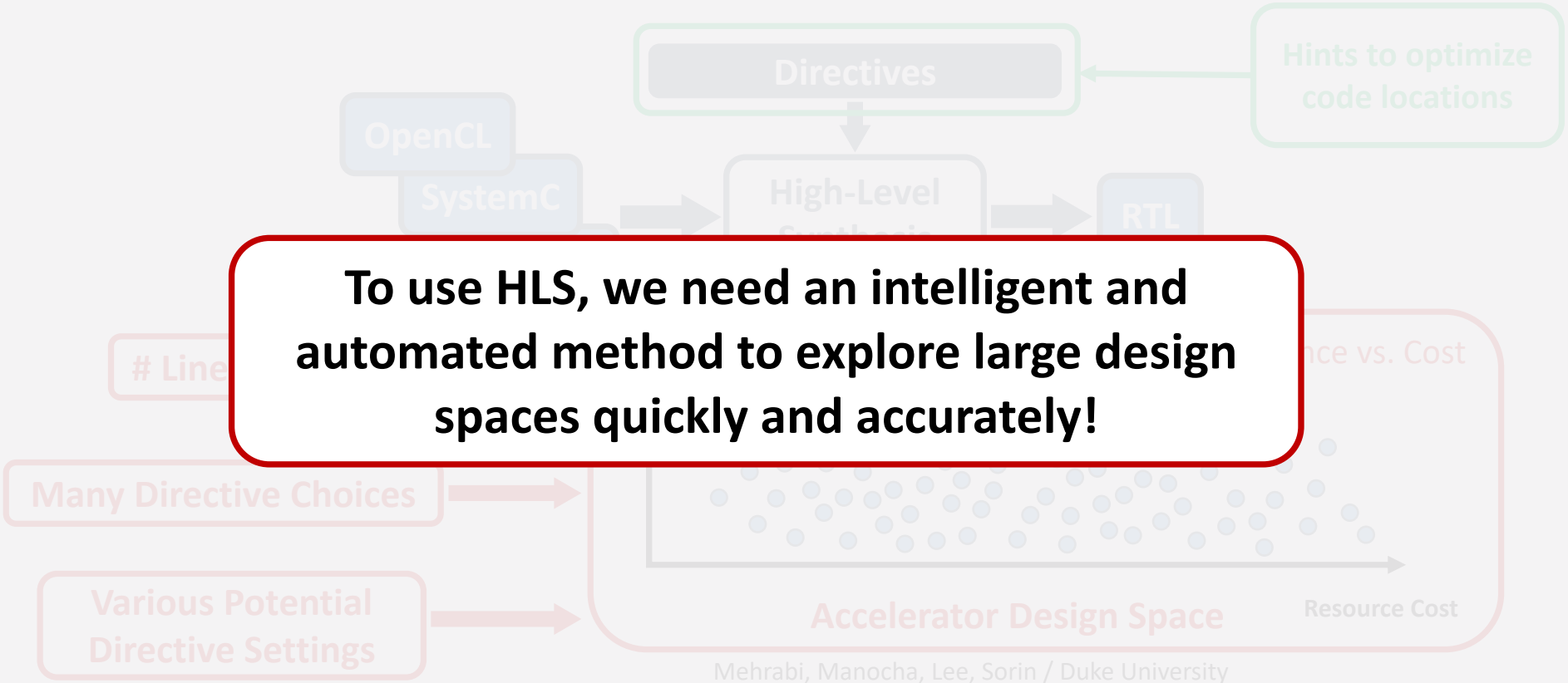
- Hardware generated by High-level Synthesis can be inefficient.
- Directives guide synthesis flow e.g. loop unrolling, array partitioning, etc.
- Tuning directives provides performance vs. cost trade-offs.

A Large Accelerator Design Space



Mehrabi, Manocha, Lee, Sorin / Duke University

A Large Accelerator Design Space



To use HLS, we need an intelligent and automated method to explore large design spaces quickly and accurately!

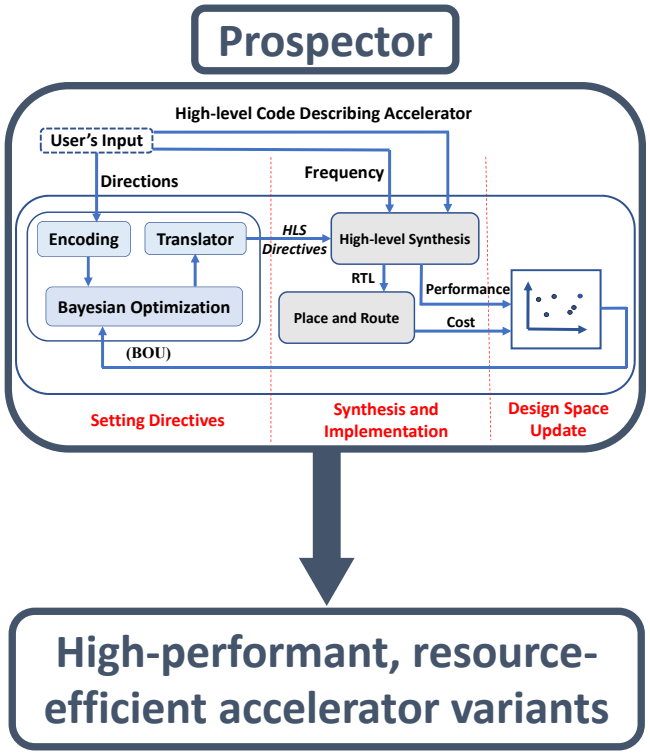
Our Approach

The Prospector Framework

- Multi-dimensional design space exploration
- Encoding of accelerator design space
- Directive placement and configuration

Prospector synthesizes efficient accelerators with Bayesian optimization

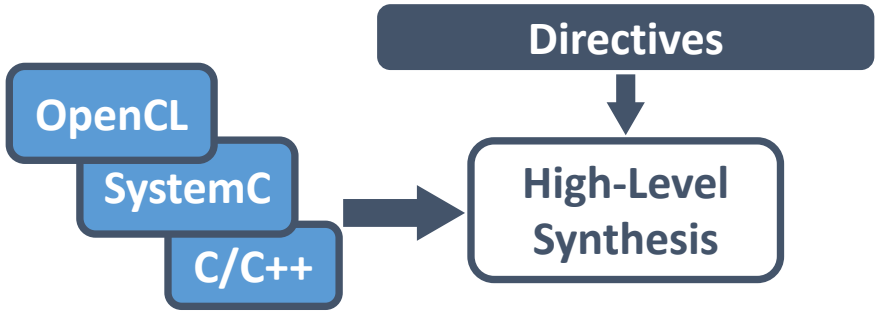
- Discovers pareto frontiers that balance performance and cost
- Outperforms prior search heuristics



Outline

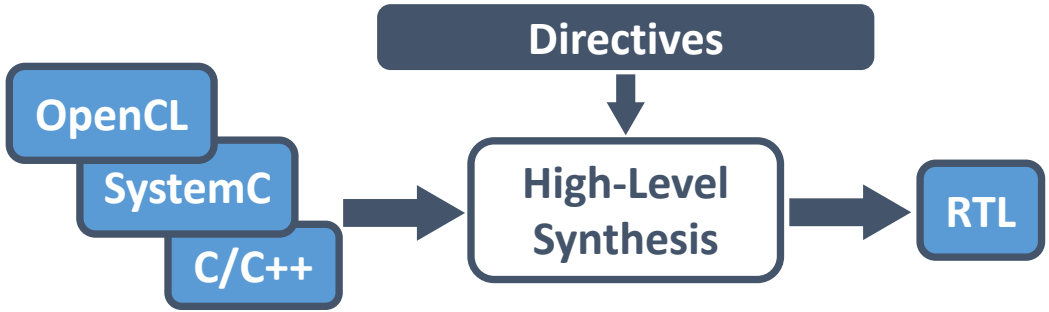
- Introduction
- **High-Level Synthesis**
- The Prospector Framework
  - Encoding the Design Space
  - Automatic Directive Placement and Configuration
- Evaluation
  - Methodology
  - Pareto Efficiency
- Conclusions

High-Level Synthesis (HLS) Overview



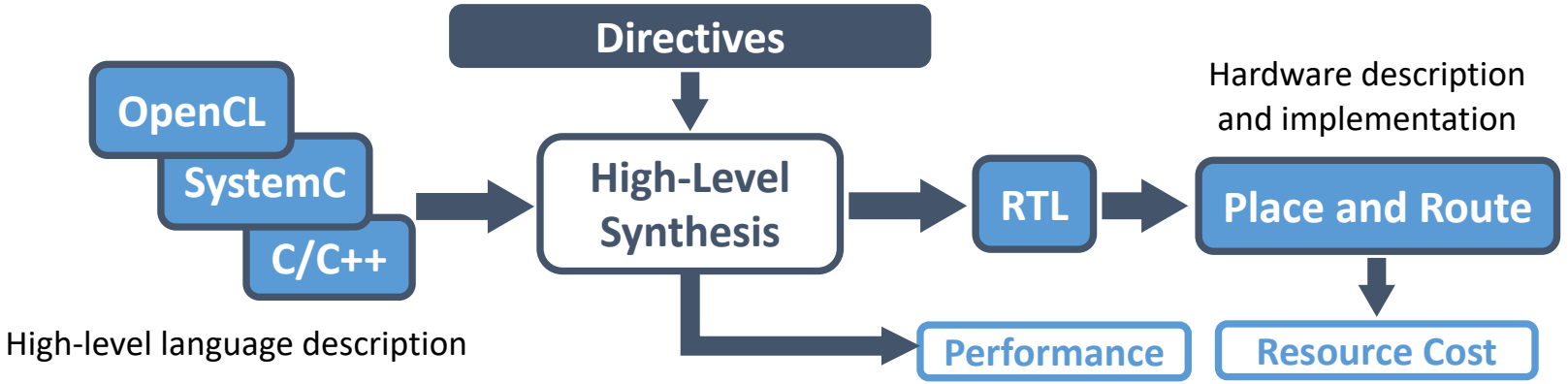
High-level language description

High-Level Synthesis (HLS) Overview

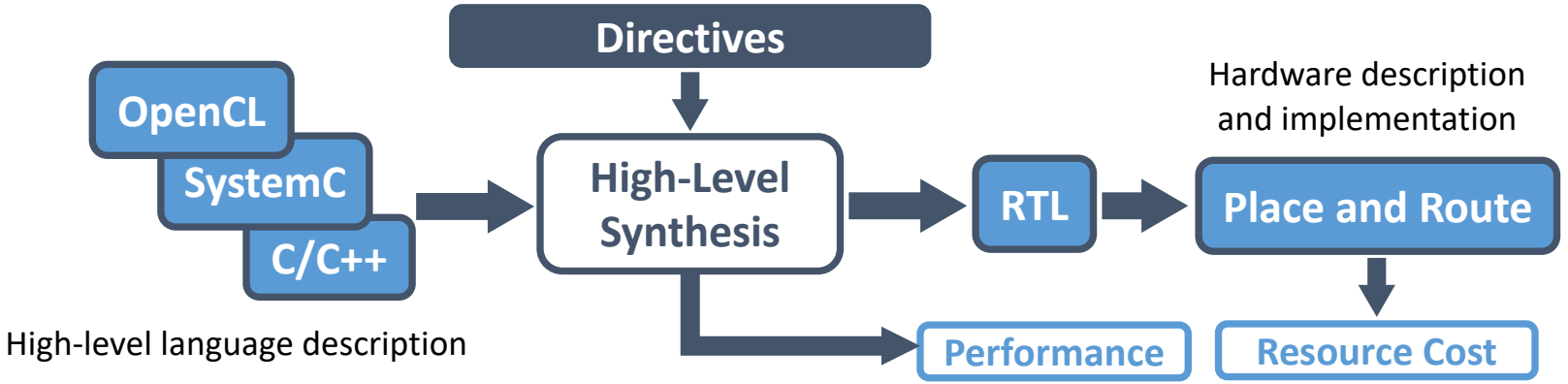


High-level language description

High-Level Synthesis (HLS) Overview

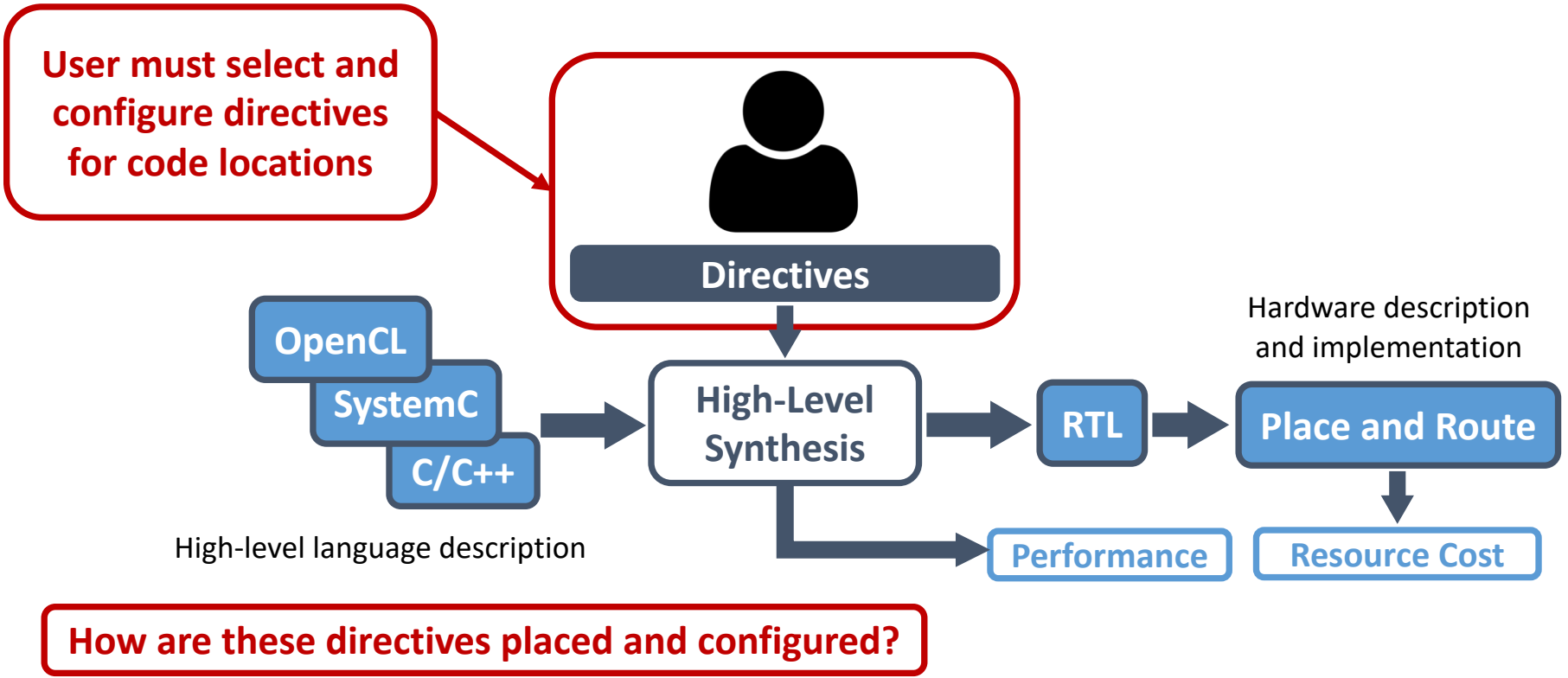


High-Level Synthesis (HLS) Overview



How are these directives placed and configured?

High-Level Synthesis (HLS) Overview



Directives Setting for HLS

Set directives for blocked matrix multiplication

```
int M1[N], int M2[N], int prod[N];

x:for(x=0; x < ROW_SIZE; x += BLK_SIZE)
  y:for(y=0; y < ROW_SIZE; y += BLK_SIZE)
    i:for(i=0; i < ROW_SIZE; ++i)
      k:for(k=0; k < BLK_SIZE; ++k)
        i_row = i * ROW_SIZE;
        k_row = (k + y) * ROW_SIZE;
        tmp_x = M1[i_row + k + y];
        j:for(j=0; j < BLK_SIZE; ++j)
          mul = tmp_x * M2[k_row + j + x];
          prod[i_row + j + x] += mul;
```

Directives Setting for HLS

Which Directives?

- Array Partitioning
- Loop Unrolling
- Loop Pipelining

```
int M1[N], int M2[N], int prod[N];

x:for(x=0; x < ROW_SIZE; x += BLK_SIZE)
  y:for(y=0; y < ROW_SIZE; y += BLK_SIZE)
    i:for(i=0; i < ROW_SIZE; ++i)
      k:for(k=0; k < BLK_SIZE; ++k)
        i_row = i * ROW_SIZE;
        k_row = (k + y) * ROW_SIZE;
        tmp_x = M1[i_row + k + y];
        j:for(j=0; j < BLK_SIZE; ++j)
          mul = tmp_x * M2[k_row + j + x];
          prod[i_row + j + x] += mul;
```

Directives Setting for HLS

Which Locations?

Array Partitioning
(M1, M2, Prod)

Loop Unrolling

Loop Pipelining

```
int M1[N], int M2[N], int prod[N];

x:for(x=0; x < ROW_SIZE; x += BLK_SIZE)
  y:for(y=0; y < ROW_SIZE; y += BLK_SIZE)
    i:for(i=0; i < ROW_SIZE; ++i)
      k:for(k=0; k < BLK_SIZE; ++k)
        i_row = i * ROW_SIZE;
        k_row = (k + y) * ROW_SIZE;
        tmp_x = M1[i_row + k + y];
        j:for(j=0; j < BLK_SIZE; ++j)
          mul = tmp_x * M2[k_row + j + x];
          prod[i_row + j + x] += mul;
```

Directives Setting for HLS

Which Locations?

Array Partitioning
(M1, M2, Prod)

Loop Unrolling
(x, y, i, k, j)

Loop Pipelining
(x, y, i, k, j)

```
int M1[N], int M2[N], int prod[N];

x:for(x=0; x < ROW_SIZE; x += BLK_SIZE)
  y:for(y=0; y < ROW_SIZE; y += BLK_SIZE)
    i:for(i=0; i < ROW_SIZE; ++i)
      k:for(k=0; k < BLK_SIZE; ++k)
          i_row = i * ROW_SIZE;
          k_row = (k + y) * ROW_SIZE;
          tmp_x = M1[i_row + k + y];
        j:for(j=0; j < BLK_SIZE; ++j)
            mul = tmp_x * M2[k_row + j + x];
            prod[i_row + j + x] += mul;
```

Directives Setting for HLS

Which Configuration Values?

Array Partitioning
(M1, M2, Prod)
[1, 2, 4, 8…]

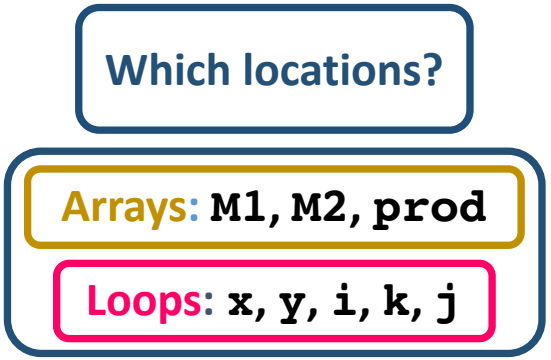
Loop Unrolling
(x, y, i, k, j)
[1, 2, 4, 6, …]

Loop Pipelining
(x, y, i, k, j)
[0, 1, 2, .., 7]

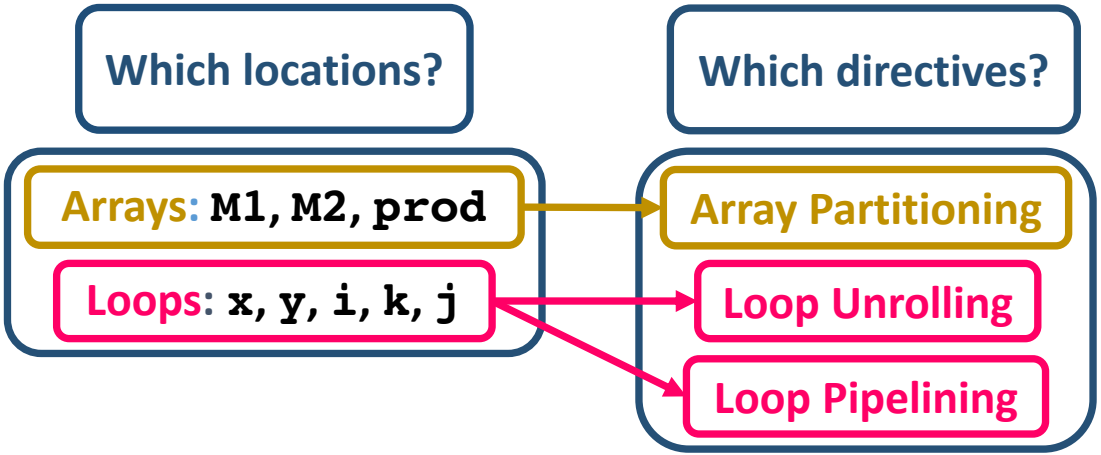
```
int M1[N], int M2[N], int prod[N];

x:for(x=0; x < ROW_SIZE; x += BLK_SIZE)
  y:for(y=0; y < ROW_SIZE; y += BLK_SIZE)
    i:for(i=0; i < ROW_SIZE; ++i)
      k:for(k=0; k < BLK_SIZE; ++k)
          i_row = i * ROW_SIZE;
          k_row = (k + y) * ROW_SIZE;
          tmp_x = M1[i_row + k + y];
        j:for(j=0; j < BLK_SIZE; ++j)
            mul = tmp_x * M2[k_row + j + x];
            prod[i_row + j + x] += mul;
```

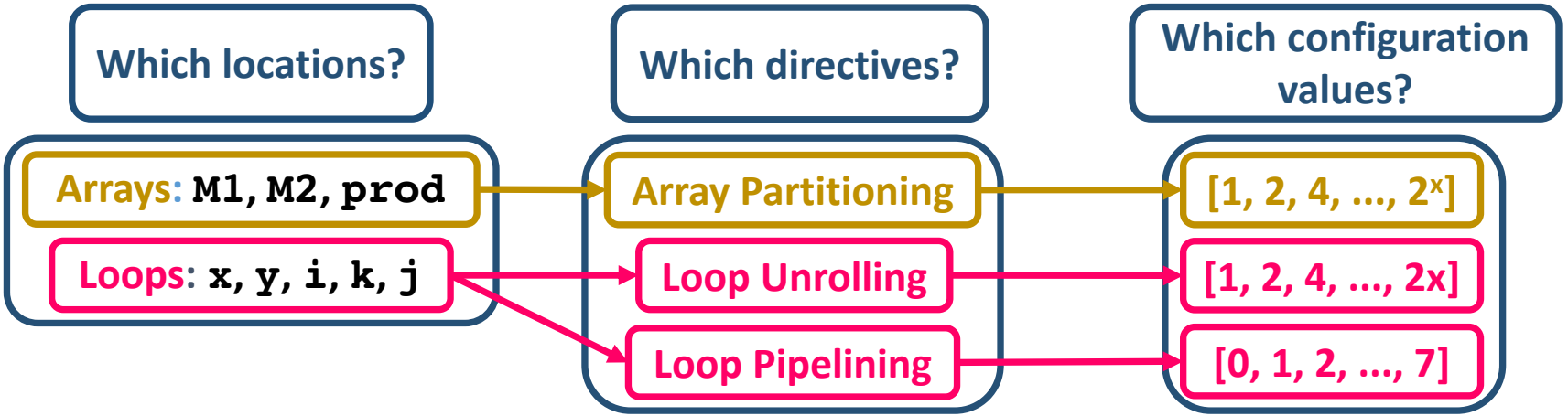
Design Space Exploration



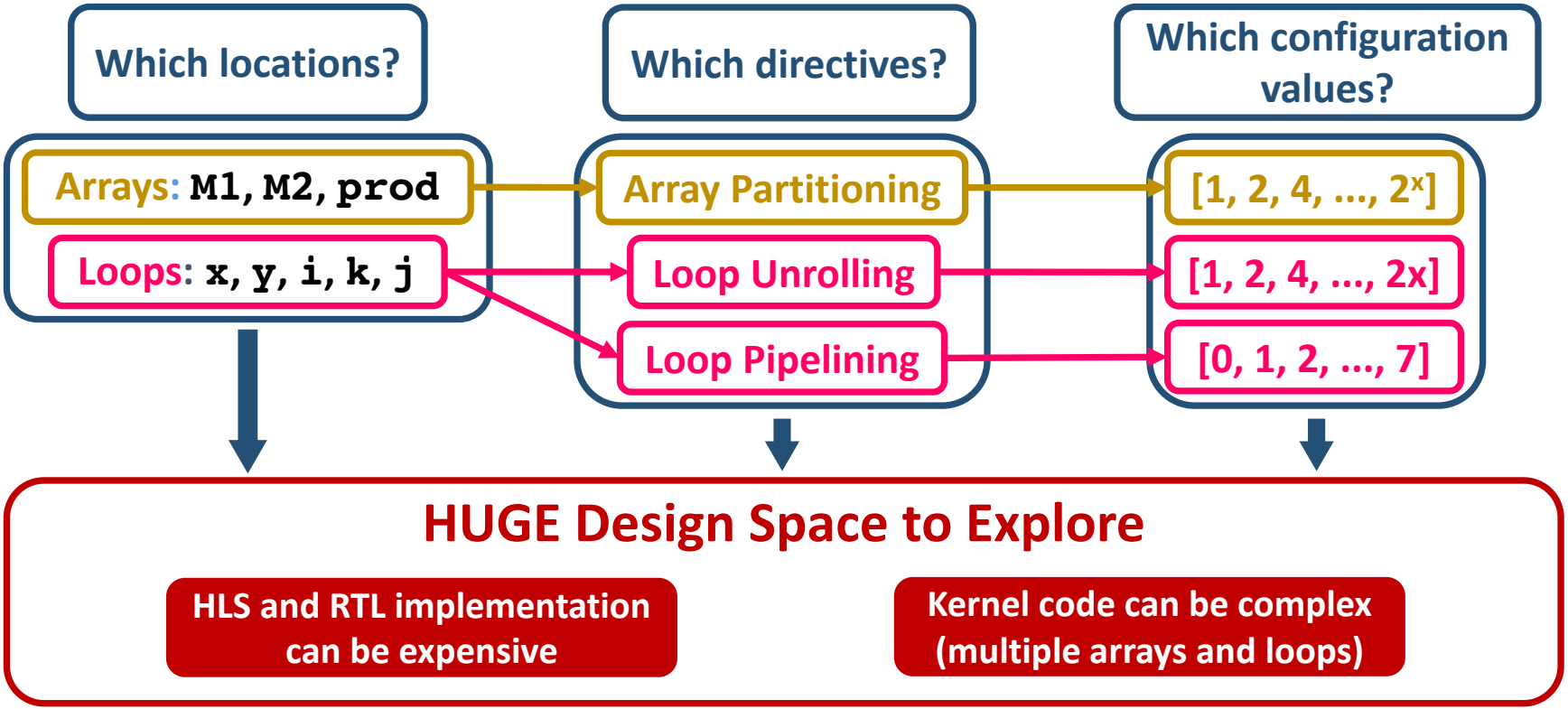
Design Space Exploration



Design Space Exploration



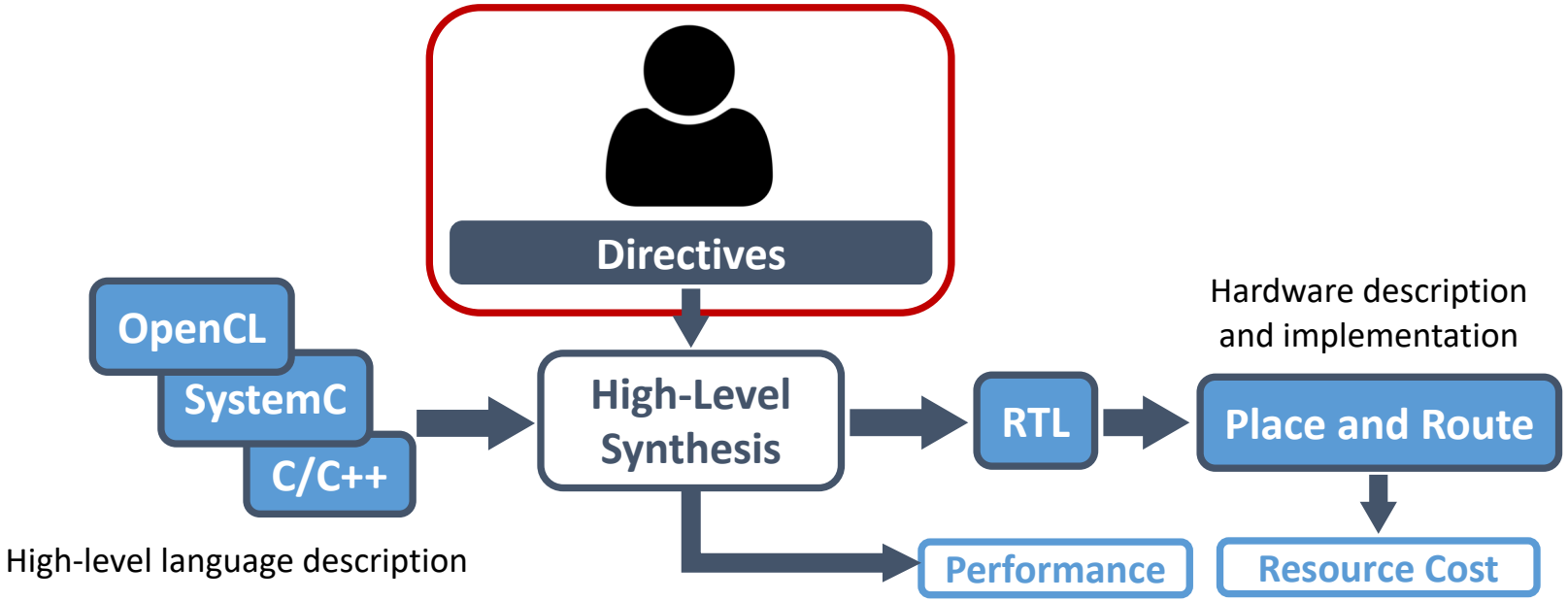
Design Space Exploration



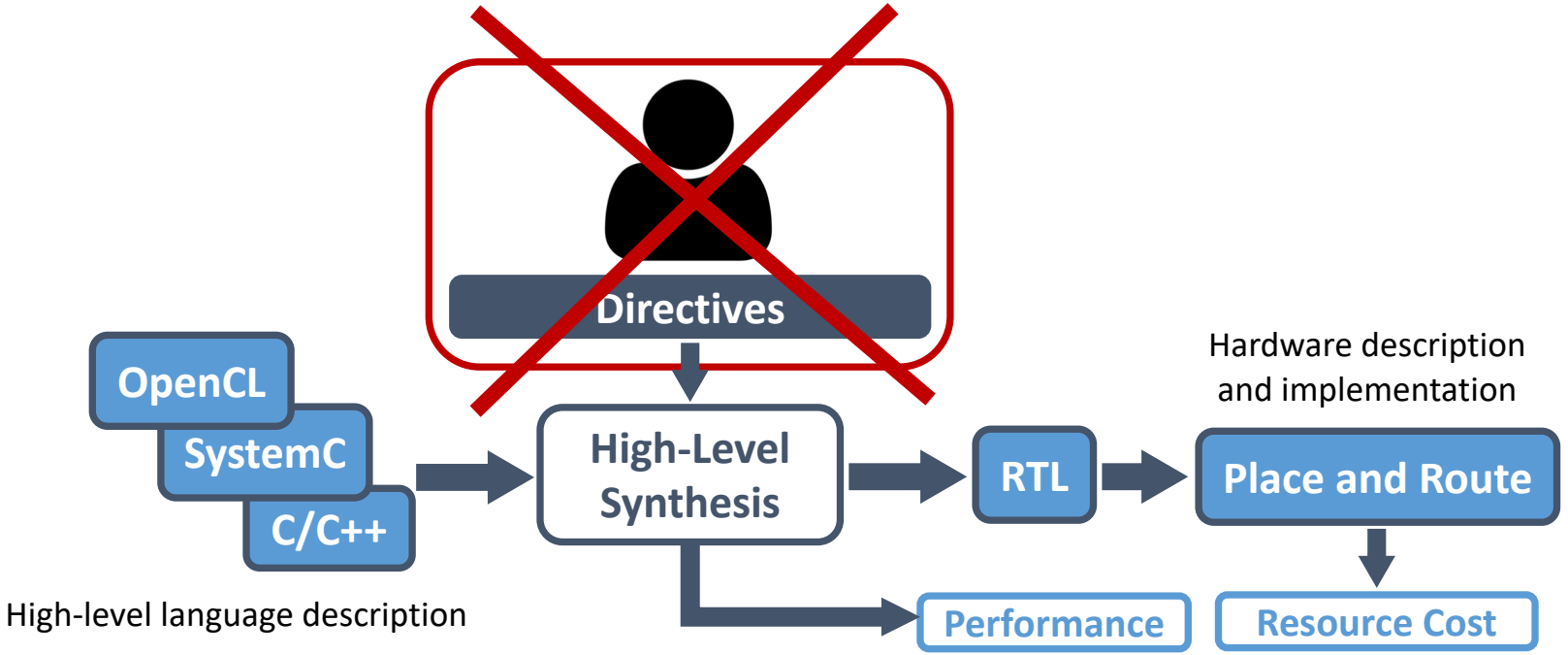
Outline

- Introduction
- High-Level Synthesis
- **The Prospector Framework**
  - **Encoding the Design Space**
  - **Automatic Directive Placement and Configuration**
- Evaluation
  - Methodology
  - Pareto Efficiency
- Conclusions

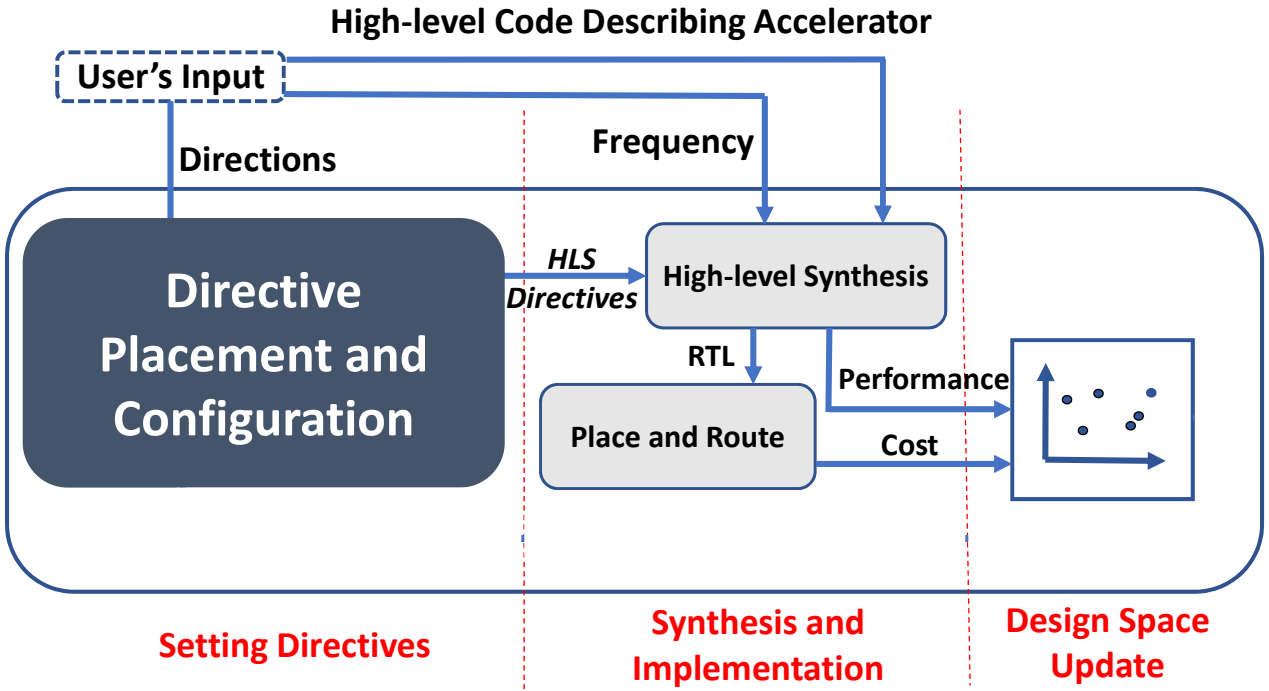
The Prospector Framework



The Prospector Framework

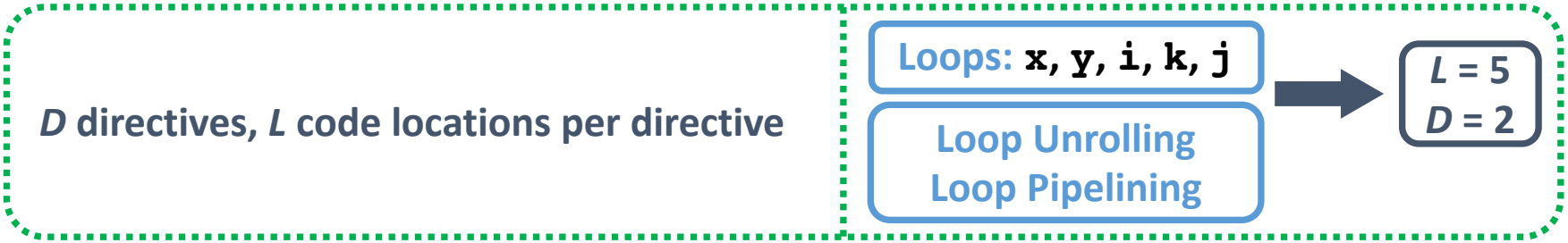


The Prospector Framework



Encoding the Design Space

Encoding the Design Space (Placement)



Encoding the Design Space (Placement)

D directives, *L* code locations per directive

Loops: `x, y, i, k, j`

Loop Unrolling
Loop Pipelining

→ *L* = 5, *D* = 2

Each directive *d* represented with bitmap of *L* bits (*D* x *L* bits) for each location *l*

l_1 = `x` l_2 = `y` l_3 = `i` l_4 = `k` l_5 = `j`

d_1 = loop unrolling ($b_1 b_2 b_3 b_4 b_5$)
d_2 = loop pipelining ($b_1 b_2 b_3 b_4 b_5$)

Encoding the Design Space (Placement)

D directives, *L* code locations per directive

Loops: `x, y, i, k, j`

Loop Unrolling
Loop Pipelining

→ *L* = 5, *D* = 2

Each directive *d* represented with bitmap of *L* bits (*D* x *L* bits) for each location *l*

l_1 = `x` l_2 = `y` l_3 = `i` l_4 = `k` l_5 = `j`

d_1 = loop unrolling ($b_1 b_2 b_3 b_4 b_5$)
d_2 = loop pipelining ($b_1 b_2 b_3 b_4 b_5$)

Each bit represents whether or not the location has the directive applied

Encoding the Design Space (Placement)

D directives, *L* code locations per directive

Loops: `x, y, i, k, j`

Loop Unrolling
Loop Pipelining

→ $L = 5$, $D = 2$

Each directive *d* represented with bitmap of *L* bits (*D* x *L* bits) for each location *l*

l_1 = `x` l_2 = `y` l_3 = `i` l_4 = `k` l_5 = `j`

d_1 = loop unrolling ($b_1 b_2 b_3 b_4 b_5$)
d_2 = loop pipelining ($b_1 b_2 b_3 b_4 b_5$)

Each bit represents whether or not the location has the directive applied

Unroll_ x
Unroll_ k
Unroll_ j
Pipeline_ j

Encoding the Design Space (Placement)

D directives, *L* code locations per directive

Loops: `x, y, i, k, j`

Loop Unrolling
Loop Pipelining

→ *L* = 5, *D* = 2

Each directive *d* represented with bitmap of *L* bits (*D* x *L* bits) for each location *l*

l_1 = `x` l_2 = `y` l_3 = `i` l_4 = `k` l_5 = `j`

d_1 = loop unrolling ($b_1 b_2 b_3 b_4 b_5$)
d_2 = loop pipelining ($b_1 b_2 b_3 b_4 b_5$)

Each bit represents whether or not the location has the directive applied

Unroll_ x
Unroll_ k
Unroll_ j
Pipeline_ j

→ d_1 = 10011 = 19
d_2 = 00001 = 1

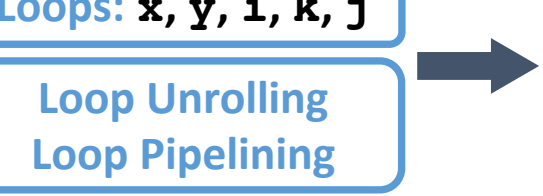
Encoding the Design Space (Configuration)

Encoding the Design Space (Configuration)

D directives, *L* code locations per directive

Loops: `x, y, i, k, j`

Loop Unrolling
Loop Pipelining



Each directive has a range of permissible values for configuration

Encoding the Design Space (Configuration)

D **directives,** ***L*** **code locations per directive**

Loops: `x, y, i, k, j`

Loop Unrolling
Loop Pipelining

L **= 5**
D **= 2**

Each directive has a range of permissible values for configuration

Loop unrolling factor:
e.g. [1, 2, 4, 6, .., 18]
Loop pipelining Initiation Interval:
e.g. [0, 1, 2, 3, .., 7]

Encoding the Design Space (Configuration)

D directives, *L* code locations per directive

Loops: `x, y, i, k, j`

Loop Unrolling
Loop Pipelining

→ *L* = 5, *D* = 2

Each directive has a range of permissible values for configuration

Loop unrolling factor:
e.g. [1, 2, 4, 6, .., 18]
Loop pipelining Initiation Interval:
e.g. [0, 1, 2, 3, .., 7]

One parameter for each directive's configuration value on each potential location

Encoding the Design Space (Configuration)

D directives, *L* code locations per directive

Loops: `x, y, i, k, j`

Loop Unrolling
Loop Pipelining

→ *L* = 5, *D* = 2

Each directive has a range of permissible values for configuration

Loop unrolling factor:
e.g. [1, 2, 4, 6, .., 18]
Loop pipelining Initiation Interval:
e.g. [0, 1, 2, 3, .., 7]

One parameter for each directive's configuration value on each potential location

Unroll-x-value
Unroll-y-value
Unroll-i-value
Unroll-k-value
Unroll-j-value

Encoding the Design Space (Configuration)

D directives, *L* code locations per directive

Loops: `x, y, i, k, j`

Loop Unrolling
Loop Pipelining

L = 5
D = 2

Each directive has a range of permissible values for configuration

Loop unrolling factor:
e.g. [1, 2, 4, 6, .., 18]
Loop pipelining Initiation Interval:
e.g. [0, 1, 2, 3, .., 7]

One parameter for each directive's configuration value on each potential location

Unroll-x-value
Unroll-y-value
Unroll-i-value
Unroll-k-value
Unroll-j-value

Pipeline-x-value
Pipeline-y-value
Pipeline-i-value
Pipeline-k-value
Pipeline-j-value

Encoding the Design Space (Configuration)

D directives, *L* code locations per directive

Loops: `x, y, i, k, j`

Loop Unrolling

L = 5
D = 2

Each c

al:

One parameter for each directive's configuration on each potential location

Unroll_x_value
Unroll_k_value
Unroll_j_value
Unroll_j_value
Unroll_k_value

Pipeline_x_value
Pipeline_k_value
Pipeline_j_value
Pipeline_j_value
Pipeline_k_value

Placement → D parameter values
Configuration → D × L parameter values

Bayesian Optimization

Statistical technique to optimize black-box functions

$$\hat{x} = argmax_{x \epsilon X} f(x)$$

Bayesian Optimization

Statistical technique to optimize black-box functions

$$\hat{x} = argmax_{x \epsilon X} f(x)$$

Statistical Model

Data Acquisition

Bayesian Optimization

Statistical technique to optimize black-box functions

$$\hat{x} = argmax_{x \epsilon X} f(x)$$

Statistical Model

Estimate *f(x)* output and uncertainty

Data Acquisition

Bayesian Optimization

Statistical technique to optimize black-box functions

$$\hat{x} = argmax_{x \epsilon X} f(x)$$

Statistical Model

- Estimate *f(x)* output and uncertainty
- Evaluate *f(x)* and update Gaussian Process model

Data Acquisition

Bayesian Optimization

Statistical technique to optimize black-box functions

$$\hat{x} = argmax_{x \epsilon X} f(x)$$

Statistical Model

- Estimate *f(x)* output and uncertainty
- Evaluate *f(x)* and update Gaussian Process model
- Reduce uncertainty and refine estimates

Data Acquisition

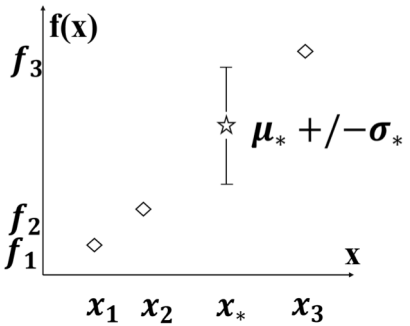
Bayesian Optimization

Statistical technique to optimize black-box functions

$$\hat{x} = argmax_{x \epsilon X} f(x)$$

Statistical Model

- Estimate *f(x)* output and uncertainty
- Evaluate *f(x)* and update Gaussian Process model
- Reduce uncertainty and refine estimates



Gaussian Process

Data Acquisition

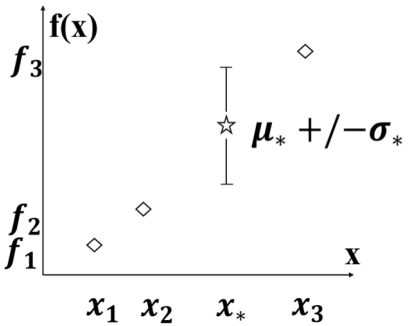
Bayesian Optimization

Statistical technique to optimize black-box functions

$$\hat{x} = argmax_{x \epsilon X} f(x)$$

Statistical Model

- Estimate *f(x)* output and uncertainty
- Evaluate *f(x)* and update Gaussian Process model
- Reduce uncertainty and refine estimates



Gaussian Process

Data Acquisition

- Select next input sample

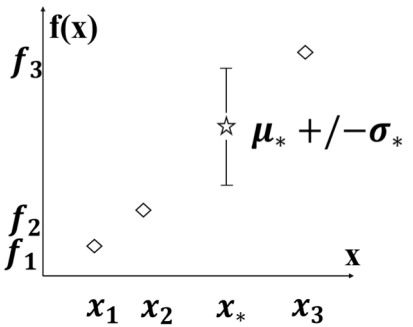
Bayesian Optimization

Statistical technique to optimize black-box functions

$$\hat{x} = argmax_{x \epsilon X} f(x)$$

Statistical Model

- Estimate *f(x)* output and uncertainty
- Evaluate *f(x)* and update Gaussian Process model
- Reduce uncertainty and refine estimates



Gaussian Process

Data Acquisition

- Select next input sample
- Balance **Exploration** vs. **Exploitation**

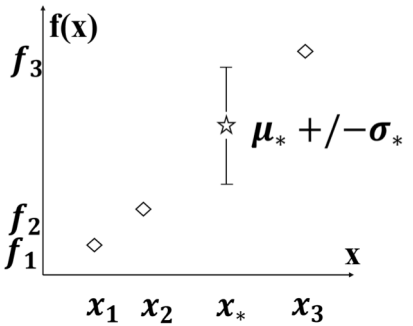
Bayesian Optimization

Statistical technique to optimize black-box functions

$$\hat{x} = argmax_{x \epsilon X} f(x)$$

Statistical Model

- Estimate *f(x)* output and uncertainty
- Evaluate *f(x)* and update Gaussian Process model
- Reduce uncertainty and refine estimates



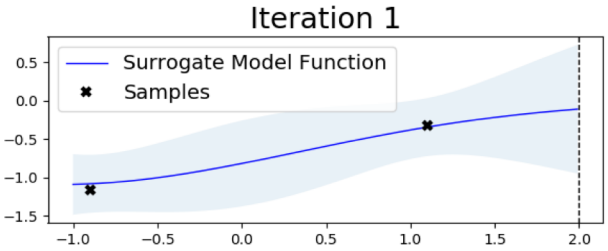
Gaussian Process

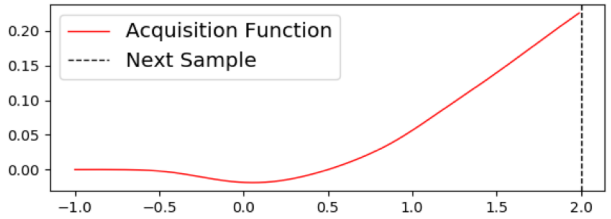
Data Acquisition

- Select next input sample
- Balance **Exploration** vs. **Exploitation**
- Model and optimize objectives

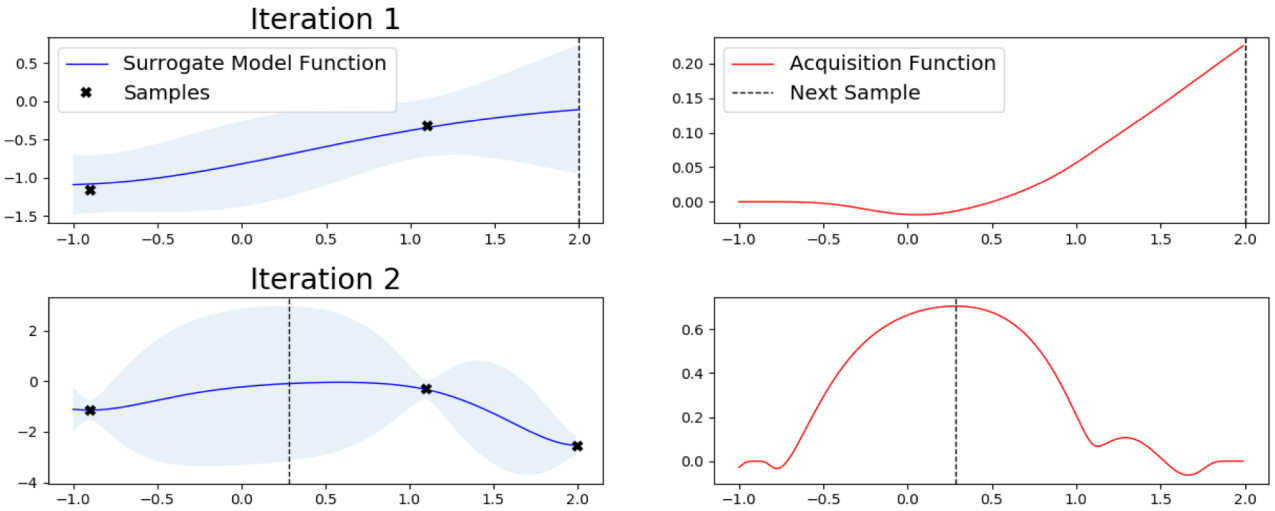
Bayesian Optimization

Bayesian Optimization



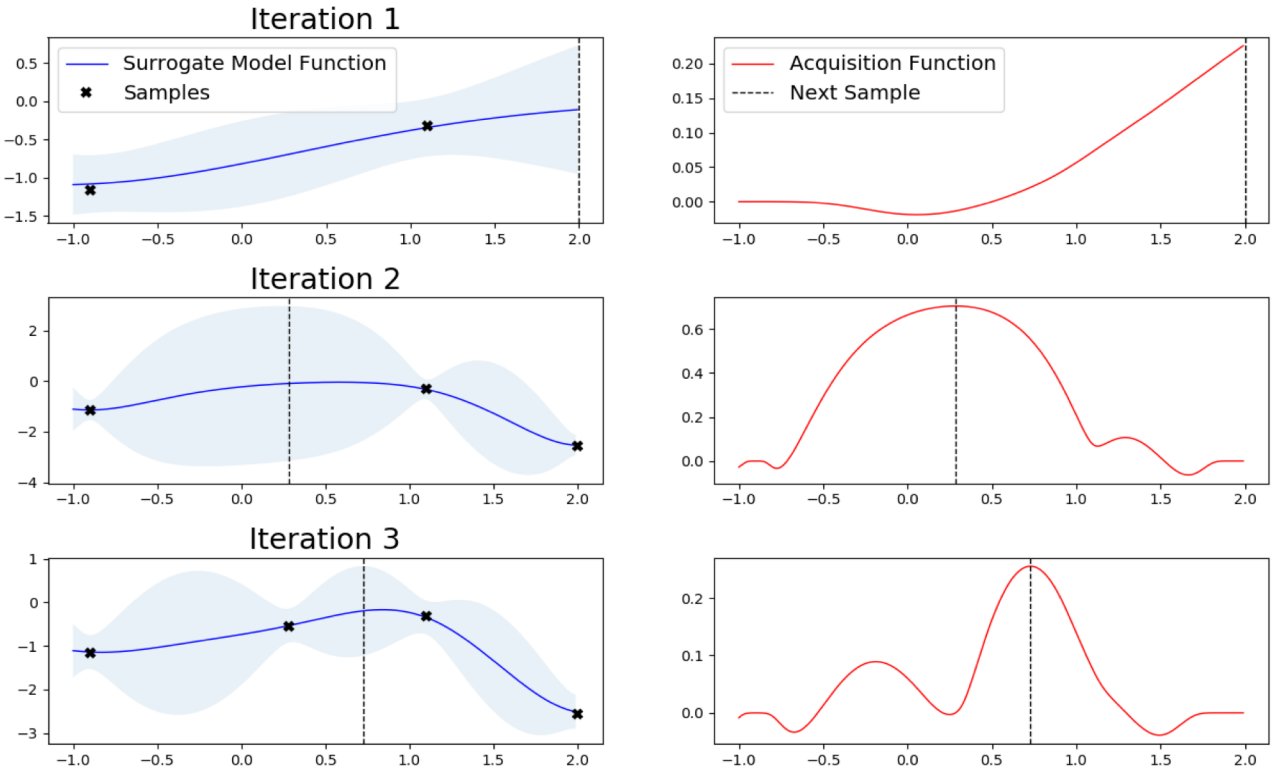


Bayesian Optimization

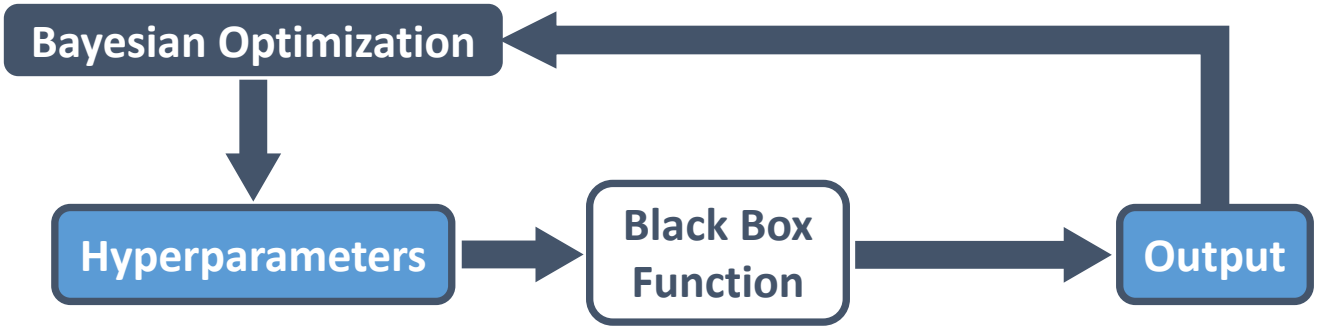


Bayesian Optimization

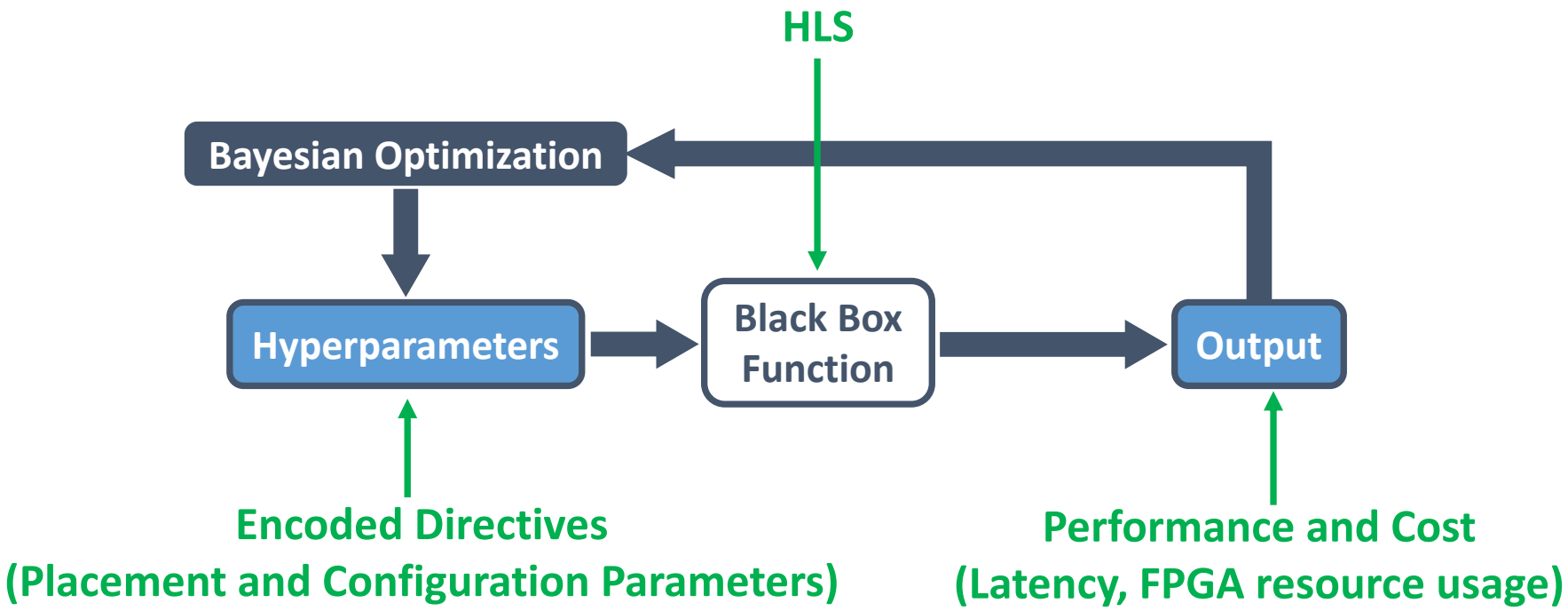
Prediction accuracy improves



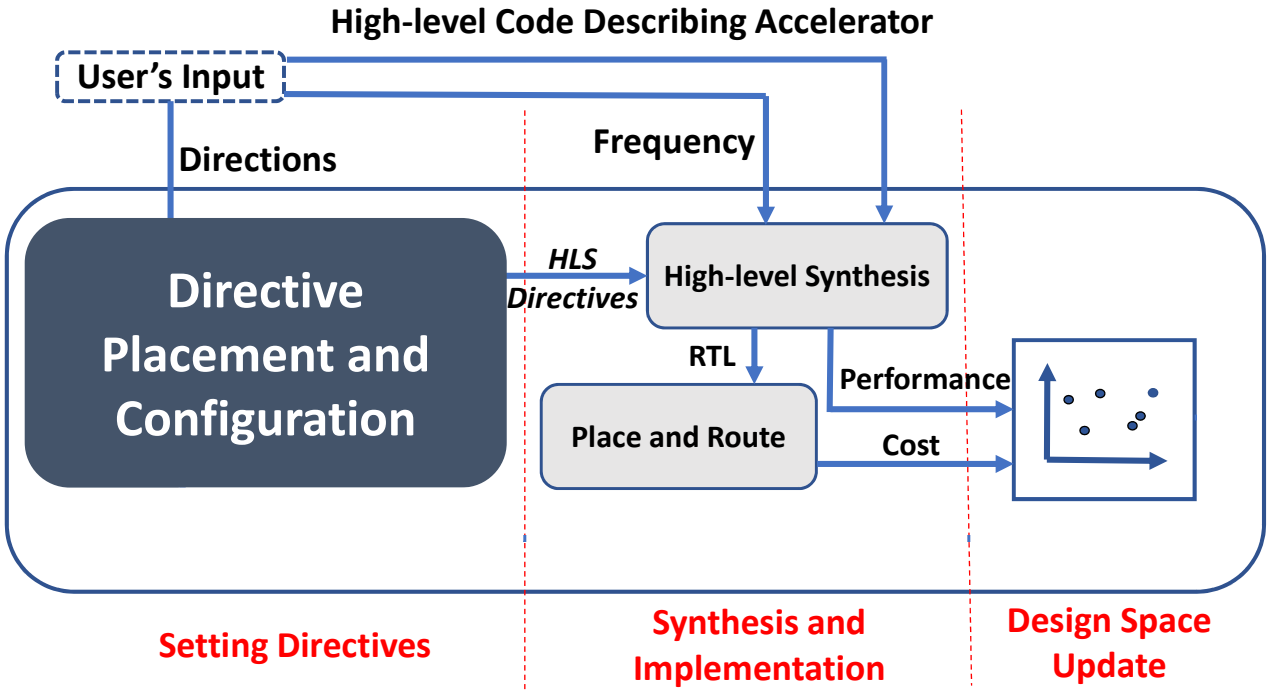
Directive Placement and Configuration



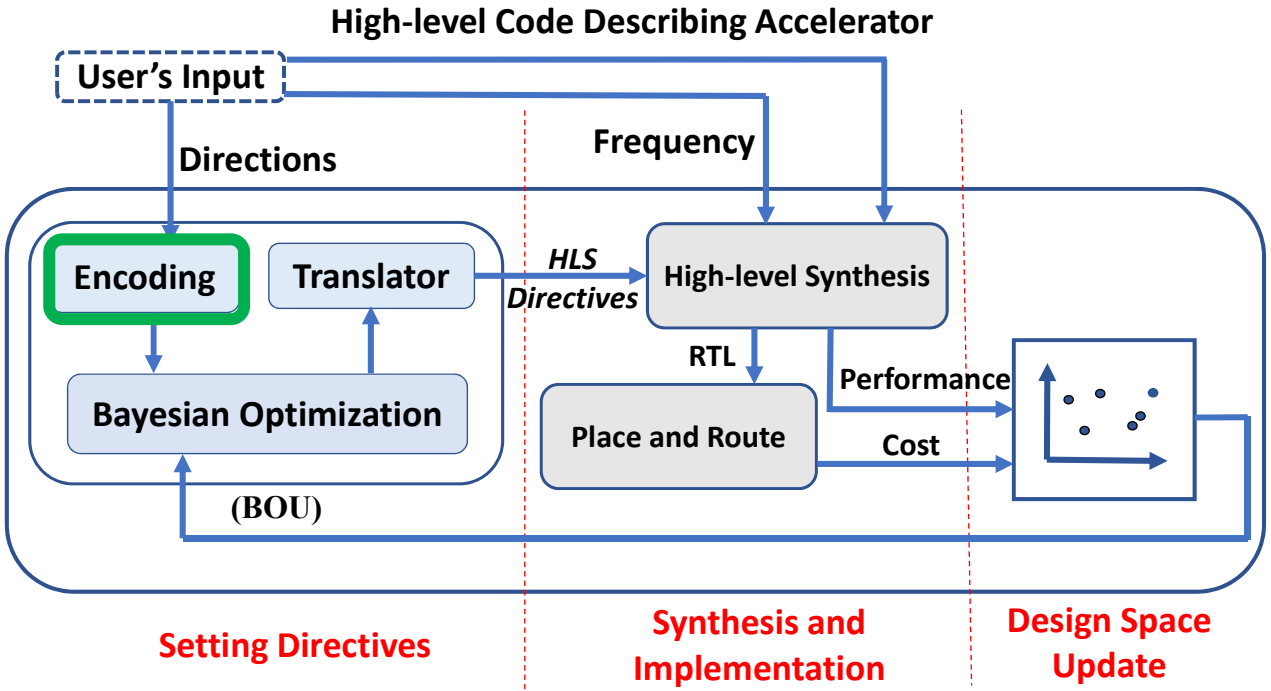
Directive Placement and Configuration



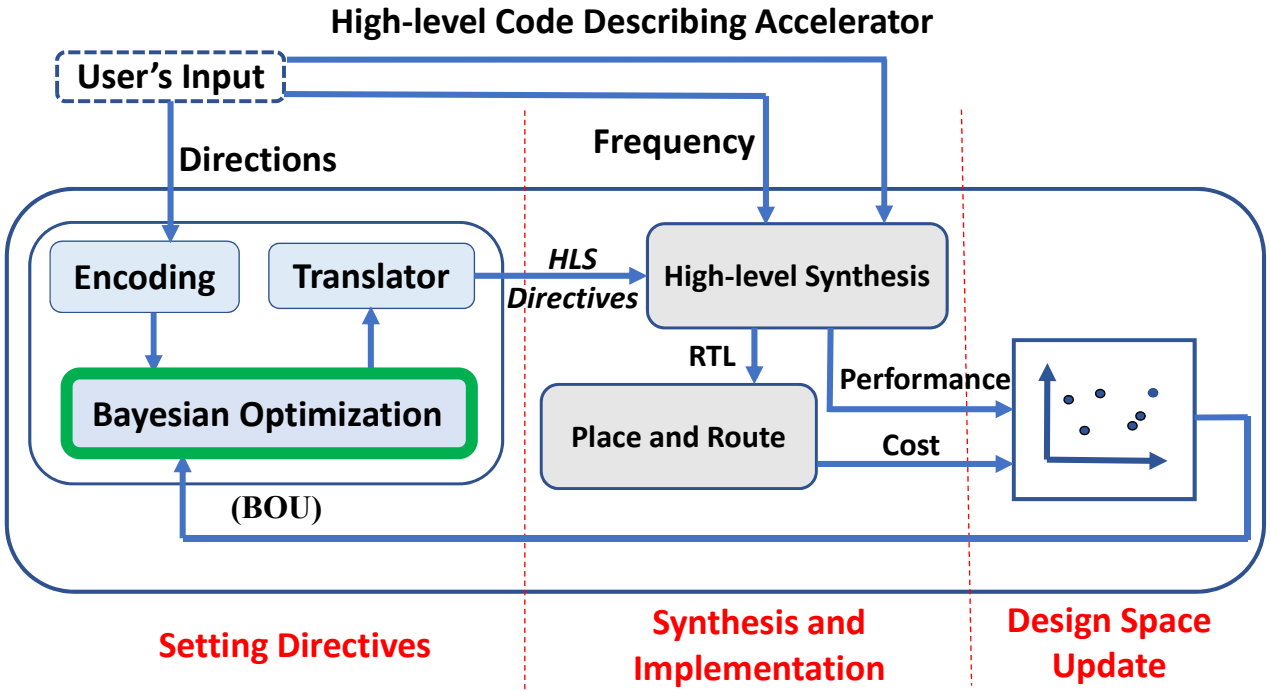
Prospector: Putting It All Together



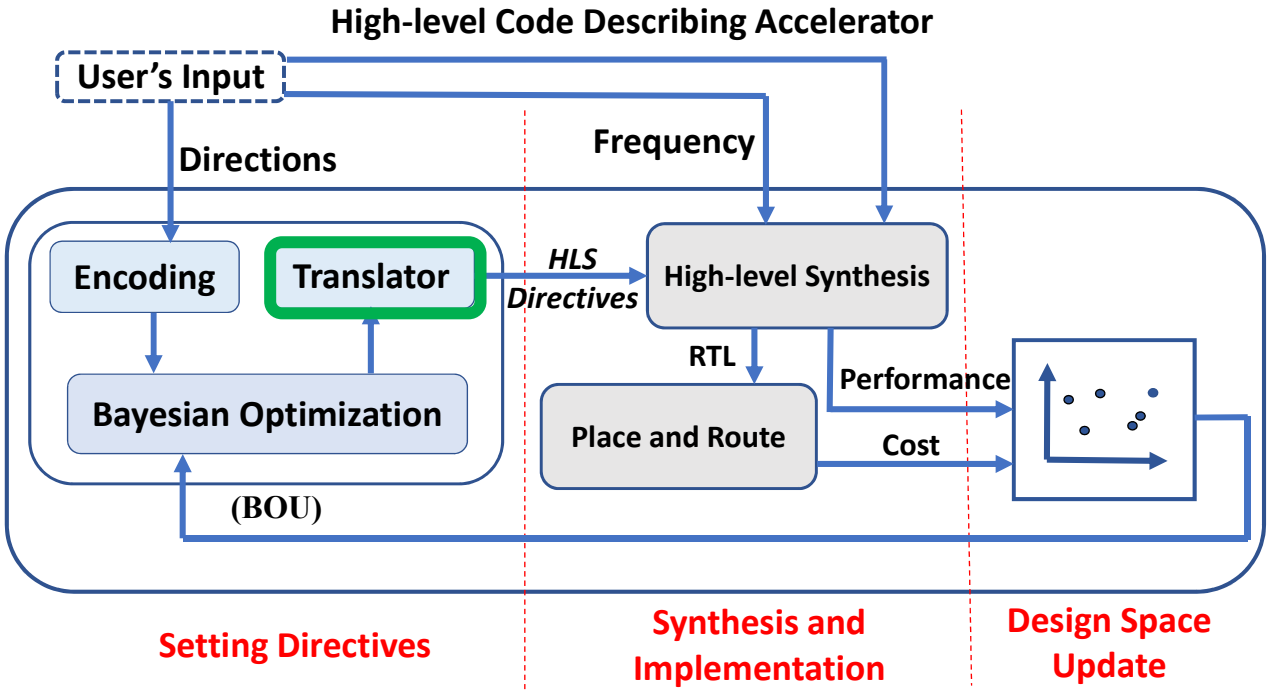
Prospector: Putting It All Together



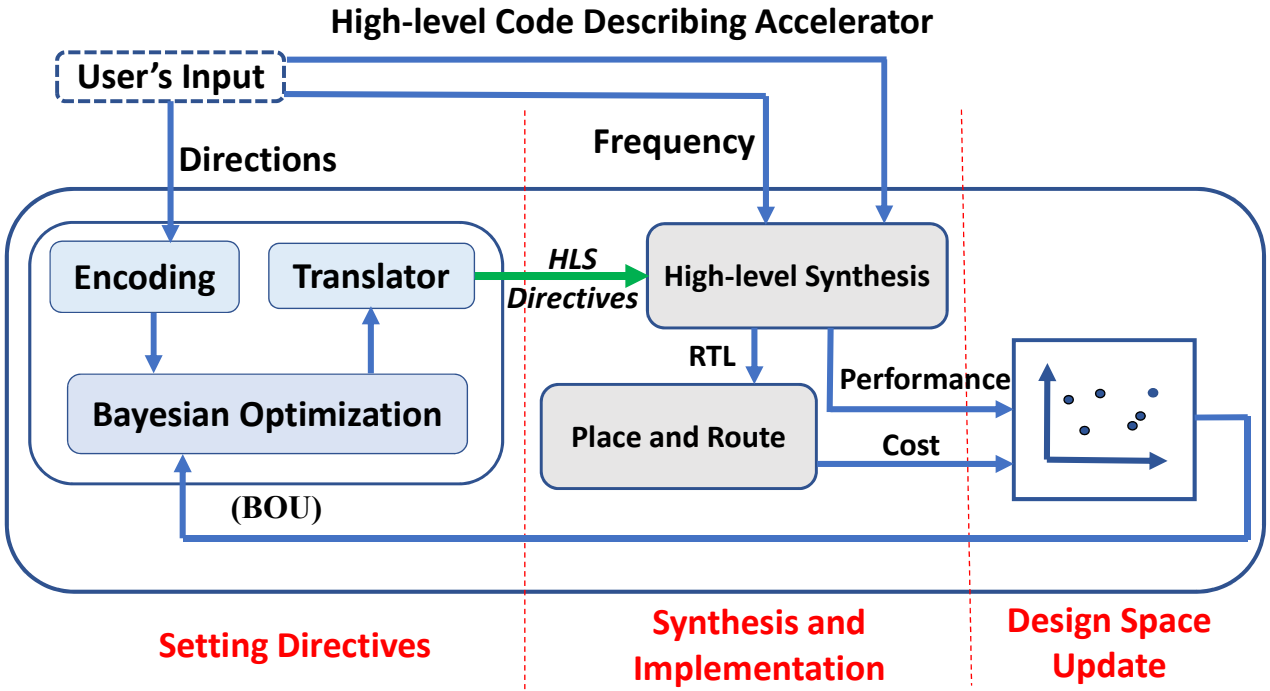
Prospector: Putting It All Together



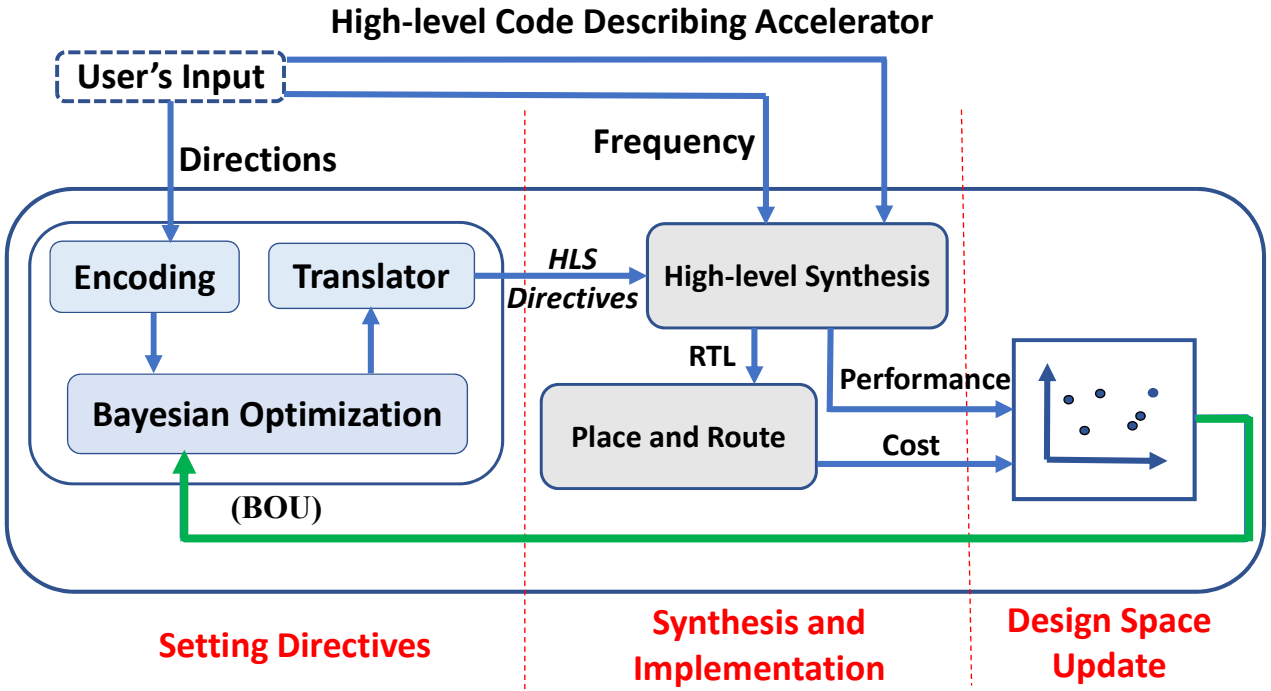
Prospector: Putting It All Together



Prospector: Putting It All Together



Prospector: Putting It All Together



Outline

- Introduction
- High-Level Synthesis for Accelerator Design
- The Prospector Framework
  - Encoding the Design Space
  - Automatic Directive Placement and Configuration
- **Evaluation**
  - **Methodology**
  - **Pareto Efficiency**
- Conclusions

Evaluation Methodology

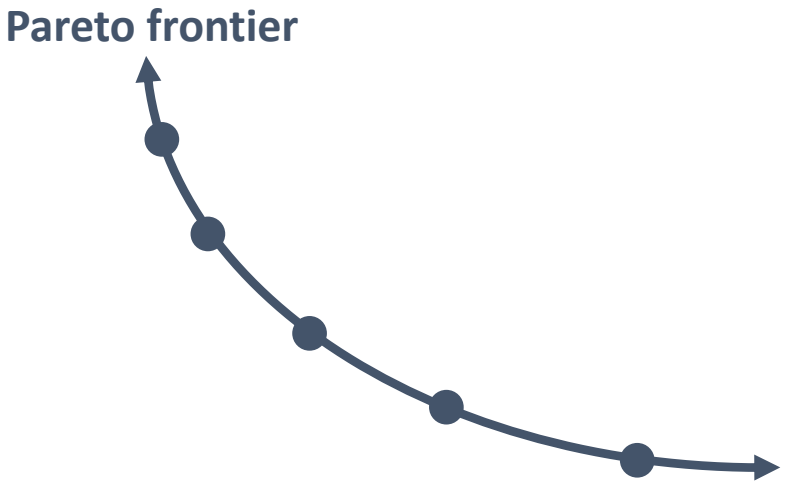
- Application kernels from **PolyBench** and **MachSuite** benchmark suites
- **Spearmint** Bayesian Optimization with PESMO acquisition function
- **Xilinx Vivado HLS:** synthesis of RTL from high-level code
- **Xilinx Vivado Design Suite:** place-and-route and FPGA bitstream generation
- **Xilinx Zynq UltraScale+** FPGA

Benchmark	Optimization Targets	Design Space Size (# Points)
fdtd-2d	4 loops	6561
2mm	4 loops, 1 array	15625
fft	9 loops, 3 arrays, 3 functions, 2 allocs	36000
bbgemm	2 loops, 1 array	960
stencil-3d	2 loops, 1 array	960
heat-3d	4 loops	2304

Performance vs. Cost: **Latency vs. {# LUTs # FFs # DSPs # BRAMs}**

Pareto Frontier

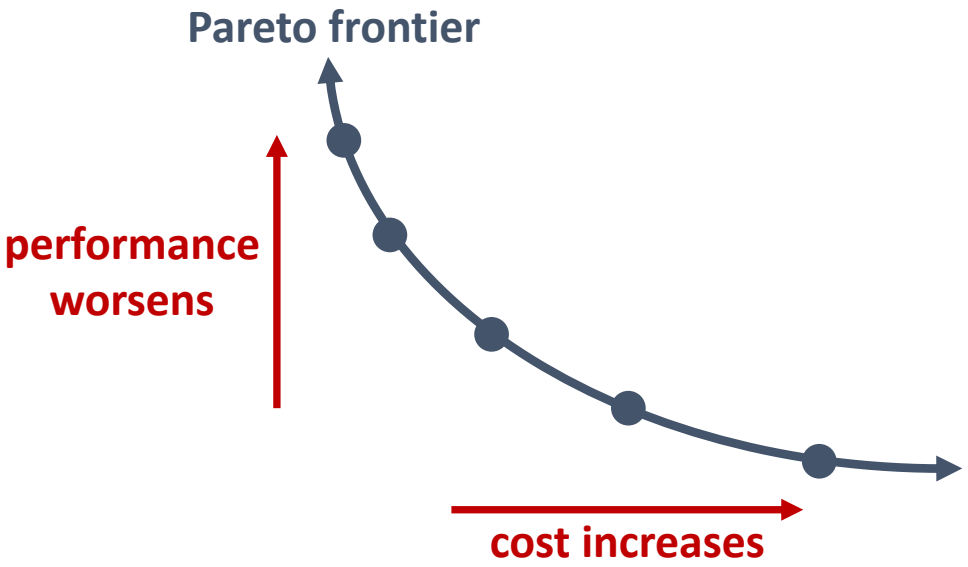
Pareto frontier: design points such that no other design improves one metric without harming another



Pareto Frontier

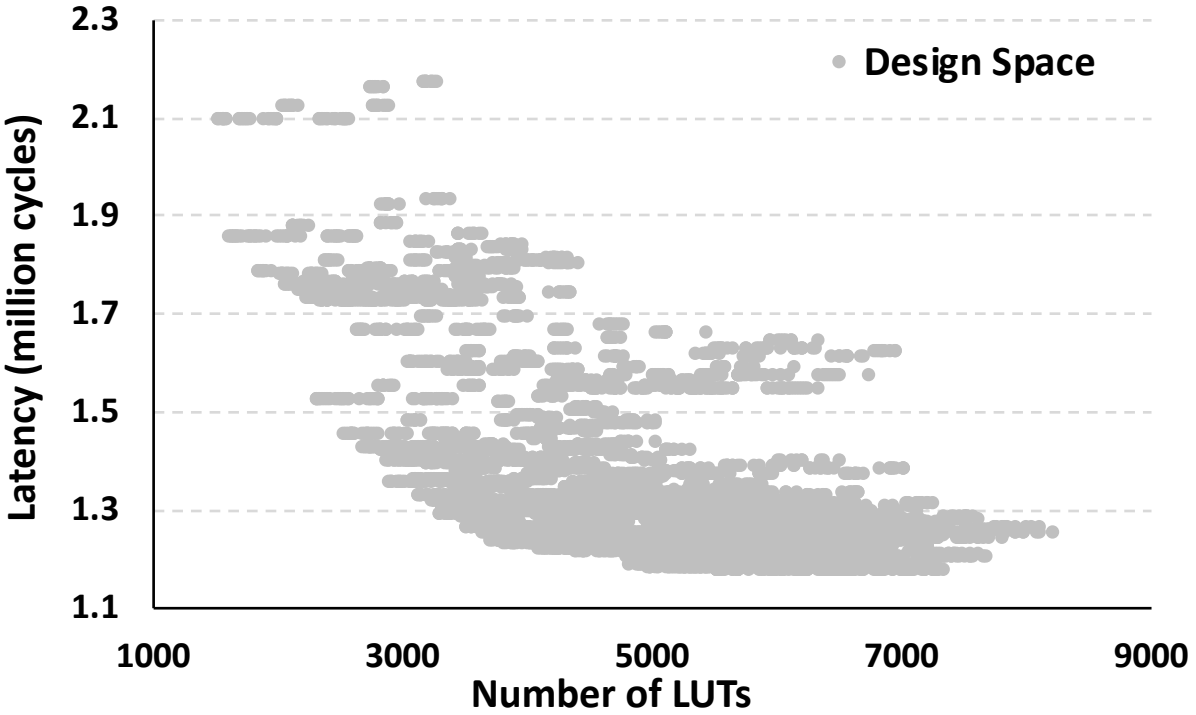
Pareto frontier: design points such that no other design improves one metric without harming another

→ Reason about design trade-offs



Prospector Efficiently Reveals Pareto Frontier

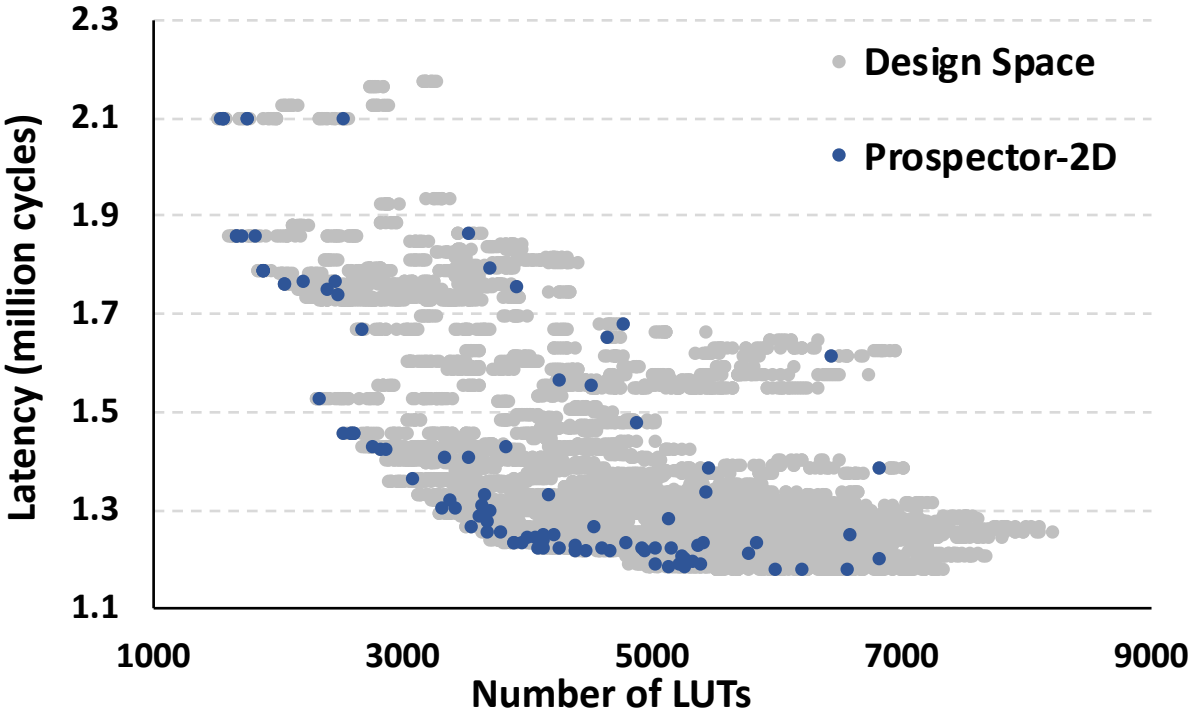
Performance/ cost trade-off:
Latency vs. LUT Usage
fdtd-2d benchmark



Prospector Efficiently Reveals Pareto Frontier

Performance/ cost trade-off:
Latency vs. LUT Usage
fdtd-2d benchmark

Prospector-2D after 100 iterations

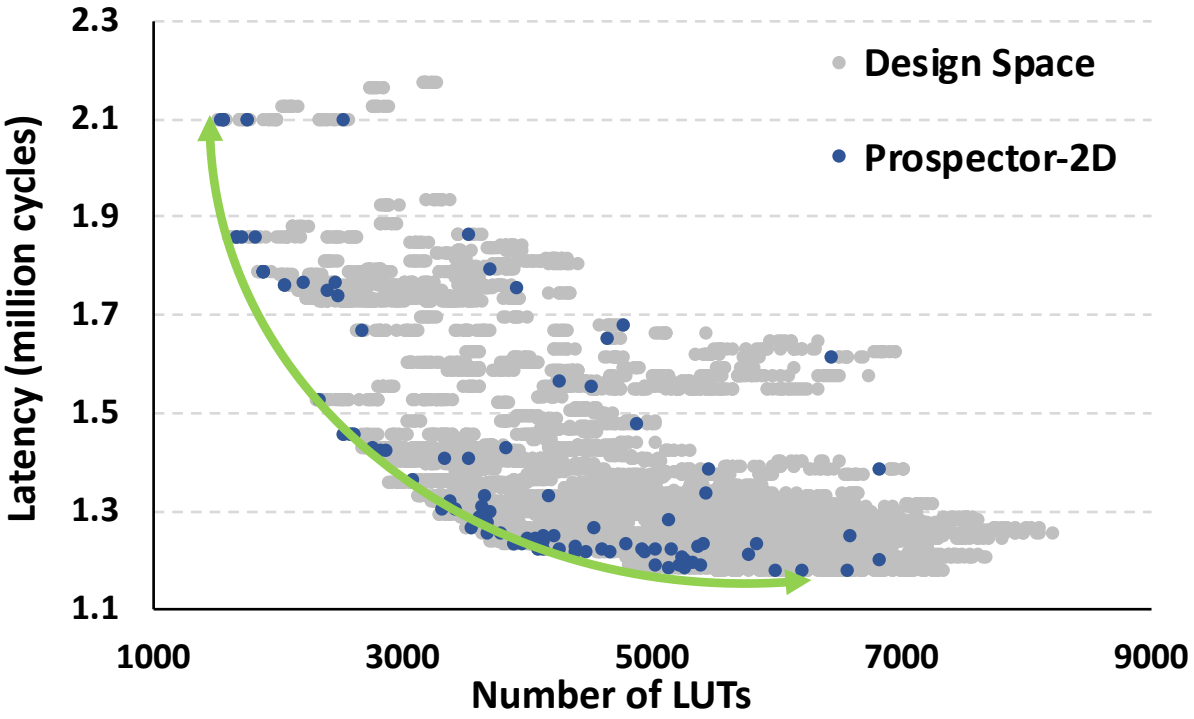


Prospector Efficiently Reveals Pareto Frontier

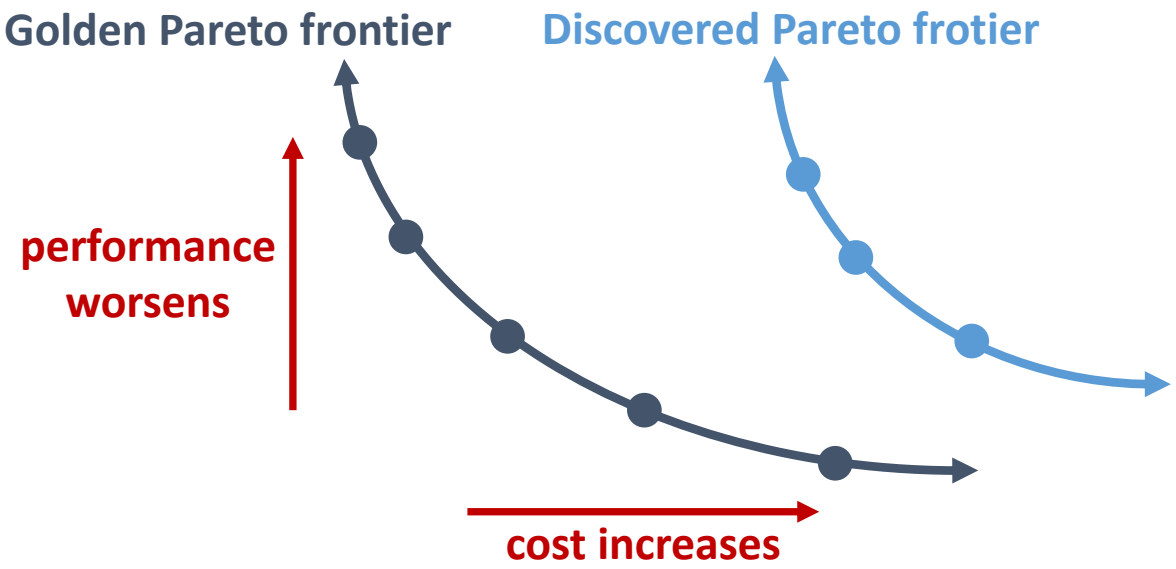
Performance/ cost trade-off:
Latency vs. LUT Usage
fdtd-2d benchmark

Prospector-2D after 100 iterations

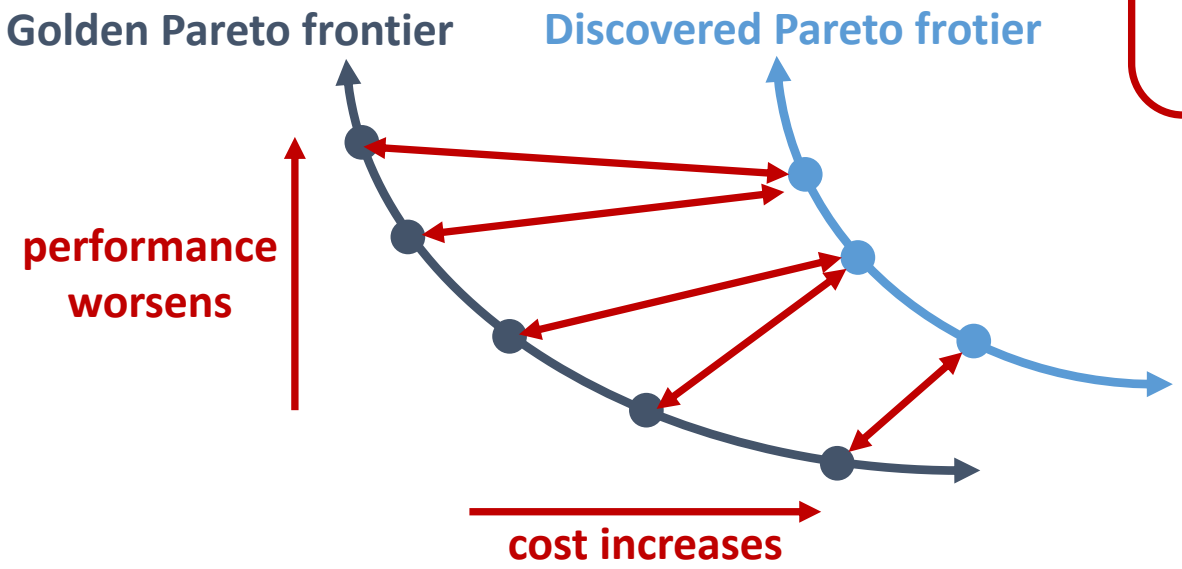
Points along Pareto frontier discovered in few iterations



Pareto Frontier Efficiency

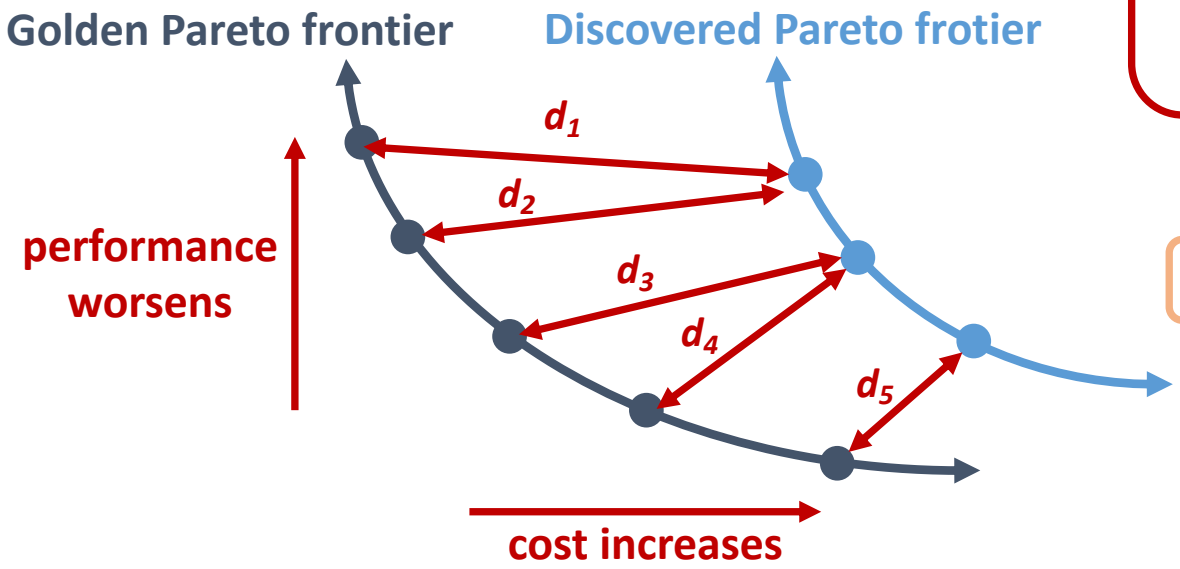


Pareto Frontier Efficiency



***Goodness*: average distance between all points from golden pareto frontier to closest point on discovered Pareto frontier**

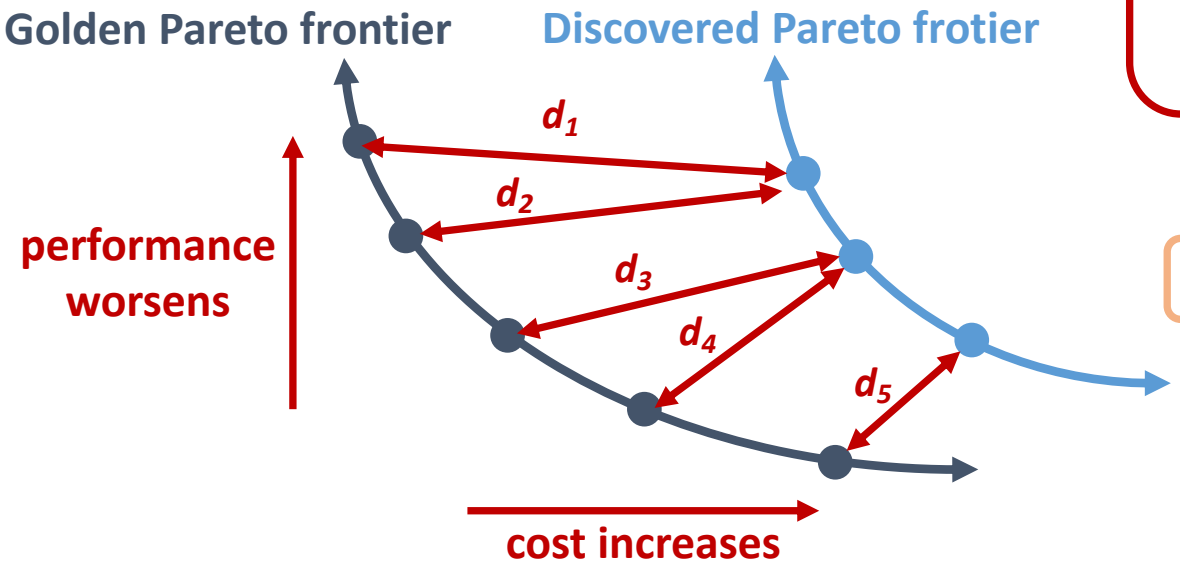
Pareto Frontier Efficiency



***Goodness*: average distance between all points from golden pareto frontier to closest point on discovered Pareto frontier**

$$X = \text{avg}\ (d_1 + d_2 + d_3 + d_4 + d_5)$$

Pareto Frontier Efficiency



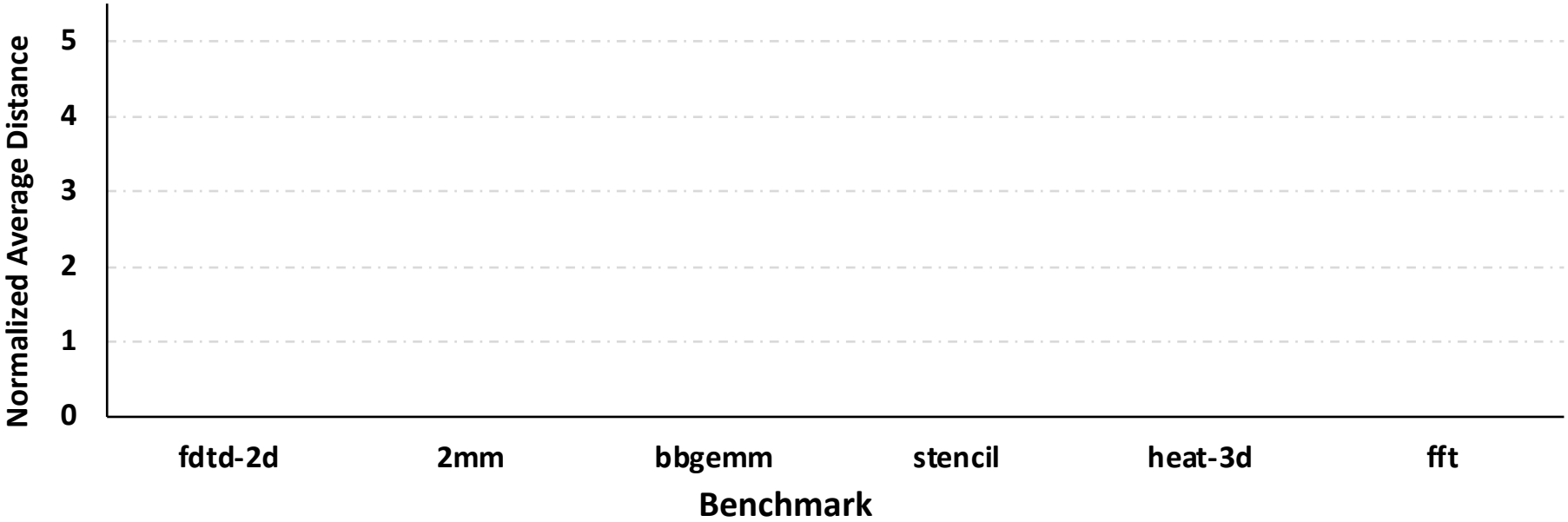
Goodness: average distance between all points from golden pareto frontier to closest point on discovered Pareto frontier

$X = avg\ (d_1 + d_2 + d_3 + d_4 + d_5)$

A good search algorithm results in a small *x* value

Prospector Outperforms Search Algorithms

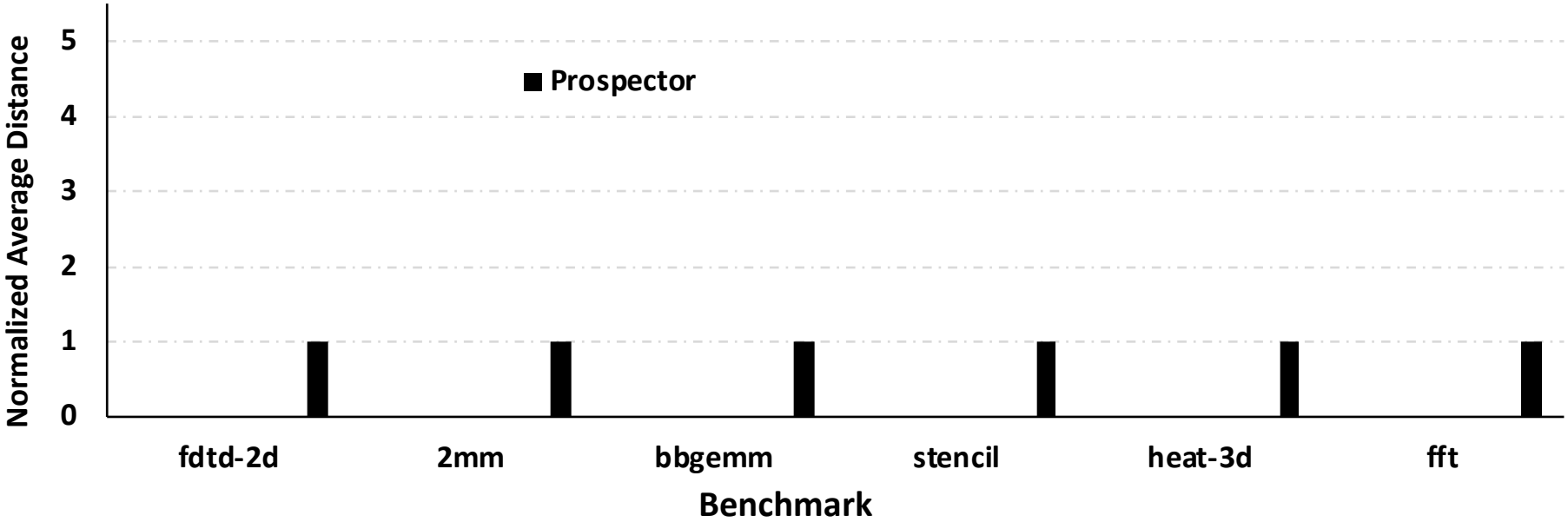
Avg. distances normalized to Prospector-5D



Prospector Outperforms Search Algorithms

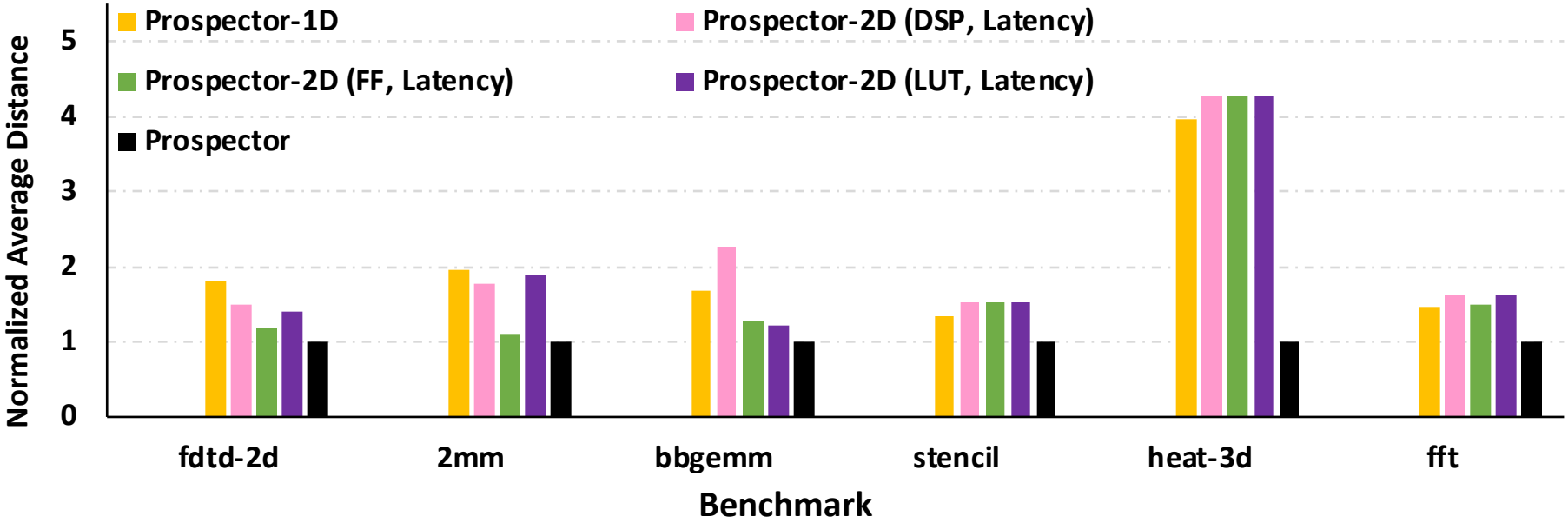
Avg. distances normalized to Prospector-5D

Lower is better



Prospector Outperforms Search Algorithms

Lower dimensional search finds less accurate pareto frontiers



Prospector Outperforms Search Algorithms

Design	2D Pareto	FFs	LUTs	DSPs	Cycles (M)
1	Latency-DSPs	4,926	7,289	32	1.175
2	Latency-LUTs	2,927	5,048	32	1.185
3	Latency-FFs	3,199	5,151	32	1.204

Prospector Outperforms Search Algorithms

Design	2D Pareto	FFs	LUTs	DSPs	Cycles (M)
1	Latency-DSPs	4,926	7,289	32	1.175
2	Latency-LUTs	2,927	5,048	32	1.185
3	Latency-FFs	3,199	5,151	32	1.204

Prospector Outperforms Search Algorithms

Design	2D Pareto	FFs	LUTs	DSPs	Cycles (M)
1	Latency-DSPs	4,926	7,289	32	1.175
2	Latency-LUTs	2,927	5,048	32	1.185
3	Latency-FFs	3,199	5,151	32	1.204

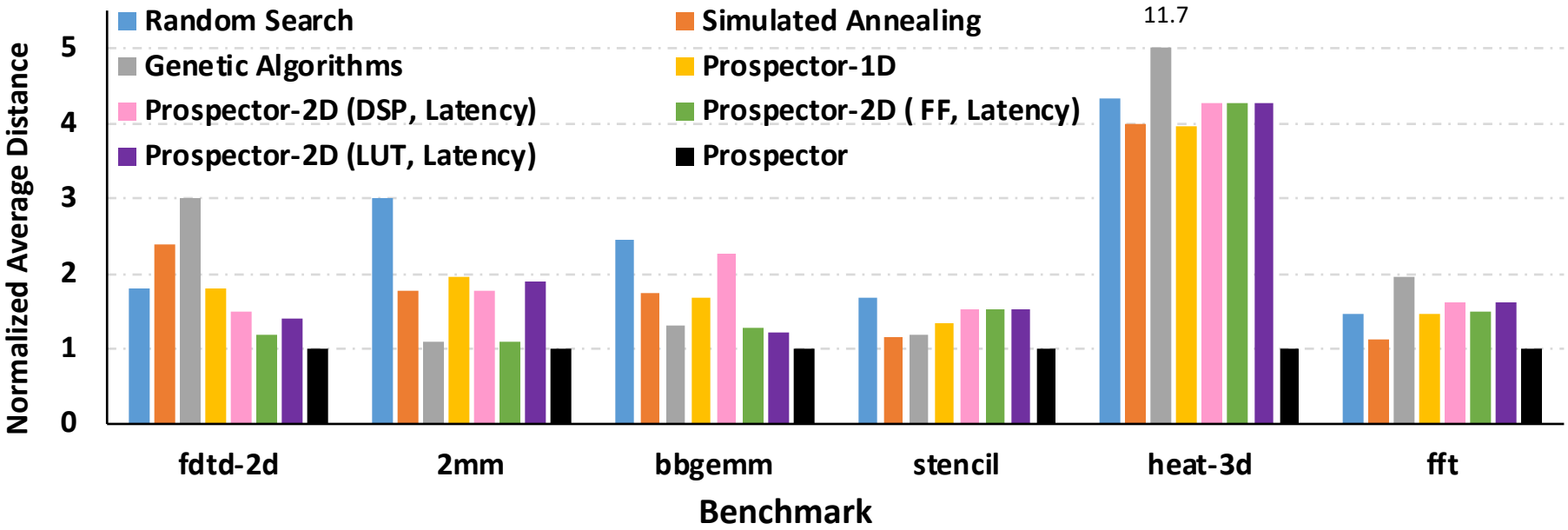
Prospector Outperforms Search Algorithms

Lower dimensional search will lead to sub-optimal design points

Design	2D Pareto	FFs	LUTs	DSPs	Cycles (M)
1	Latency-DSPs	4,926	7,289	32	1.175
2	Latency-LUTs	2,927	5,048	32	1.185
3	Latency-FFs	3,199	5,151	32	1.204

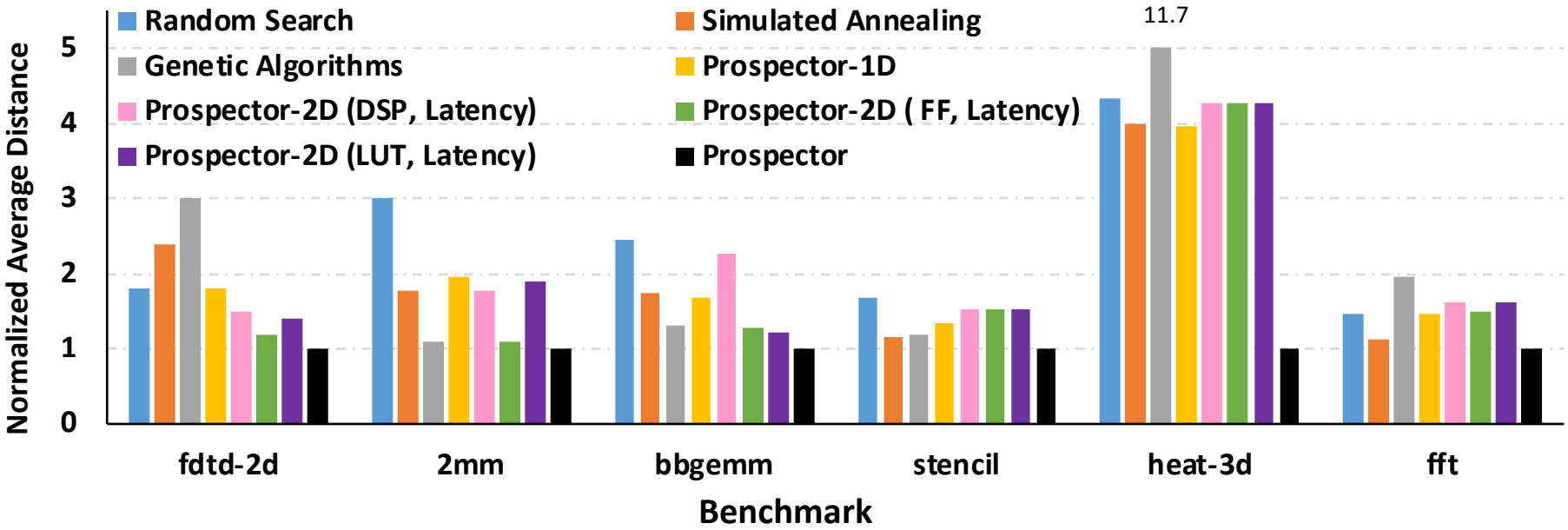
~40% fewer LUTs and FFs

Prospector Outperforms Search Algorithms



Prospector Outperforms Search Algorithms

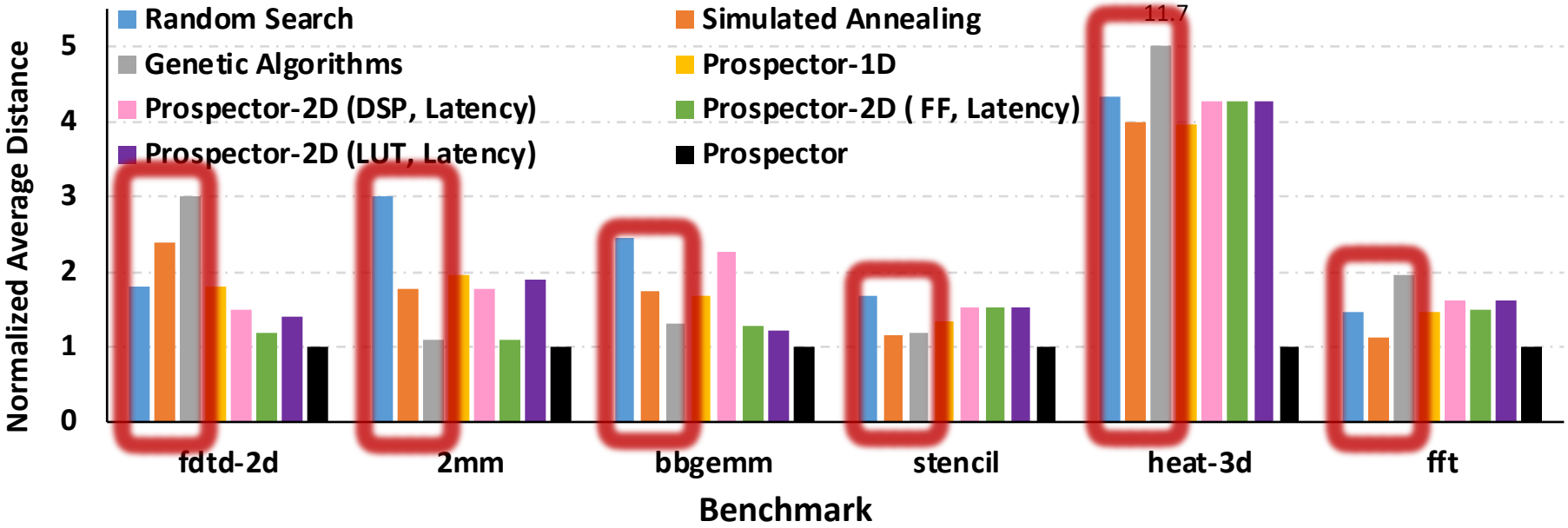
Prospector reveals the most accurate Pareto frontier



Prospector Outperforms Search Algorithms

Alternative search heuristics are less accurate

Prospector reveals the most accurate Pareto frontier



Outline

- Introduction
- High-Level Synthesis for Accelerator Design
- The Prospector Framework
  - Encoding the Design Space
  - Automatic Directive Placement and Configuration
- Evaluation
  - Methodology
  - Pareto Efficiency
- **Conclusions**

Conclusions

- **Efficient accelerator design is time-consuming**
- **HLS can automatically produce RTL, but RTL quality is sensitive to directive placement and configuration**
- **Prospector offers a framework for accelerator design with HLS**
  - Multi-dimensional accelerator design space exploration
  - Automatic placement and configuration of directives
- **Prospector synthesizes efficient accelerators**
  - Efficiently discovers Pareto optimal design space points
  - Outperforms efficiency of prior search heuristics

Thank you!

atefeh.mehrabi@duke.edu