HL-Pow: A Learning-Based Power Modeling Framework for High-Level Synthesis

Zhe Lin[1], Jieru Zhao[1], Sharad Sinha[2], Wei Zhang[1]
[1]Hong Kong University of Science and Technology (HKUST)
[2]Indian Institute of Technology (IIT) Goa

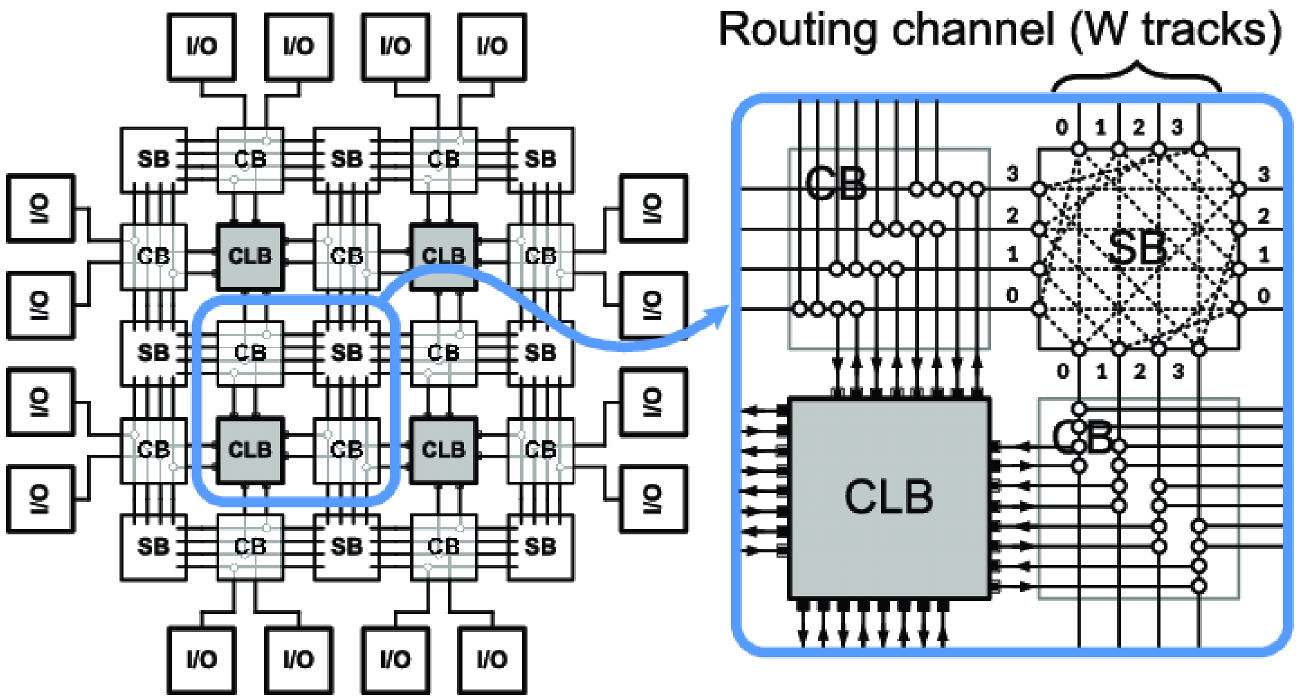
Outline

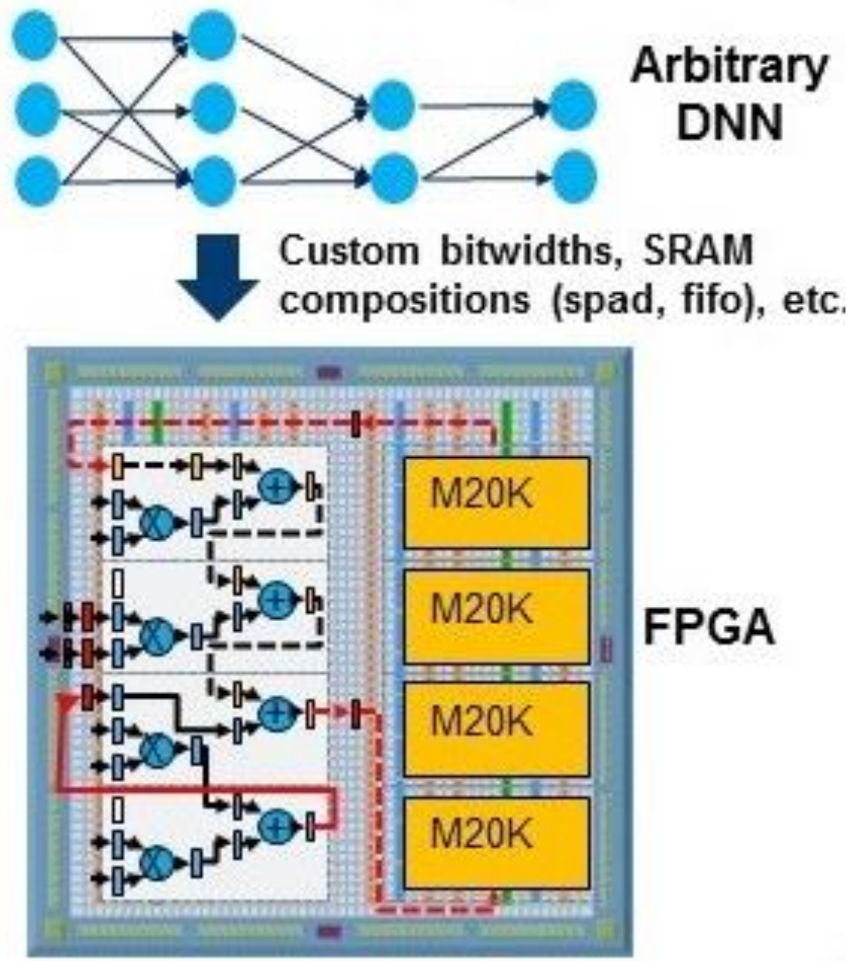
Outline

- **Introduction**
- Related Work
- Power Modeling Framework
- Design Space Exploration
- Conclusion

FPGA Overview

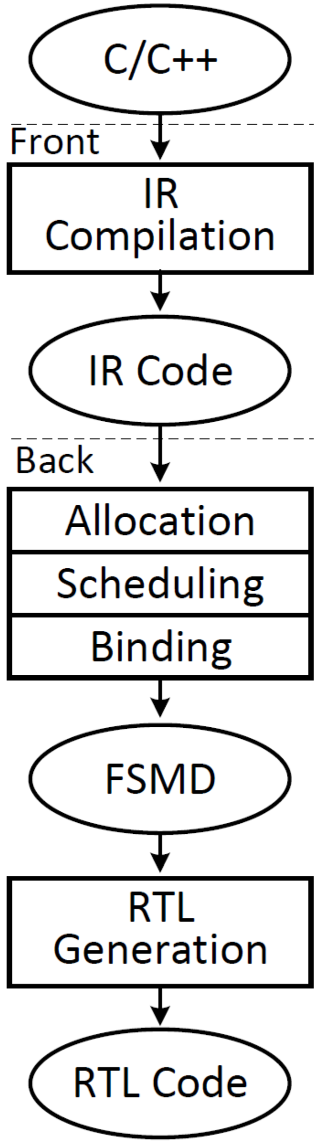
- **F**ield-**P**rogrammable **G**ate **A**rray (FPGA)
  - Customizable logic blocks and wirings
  - On-chip memory and arithmetic units





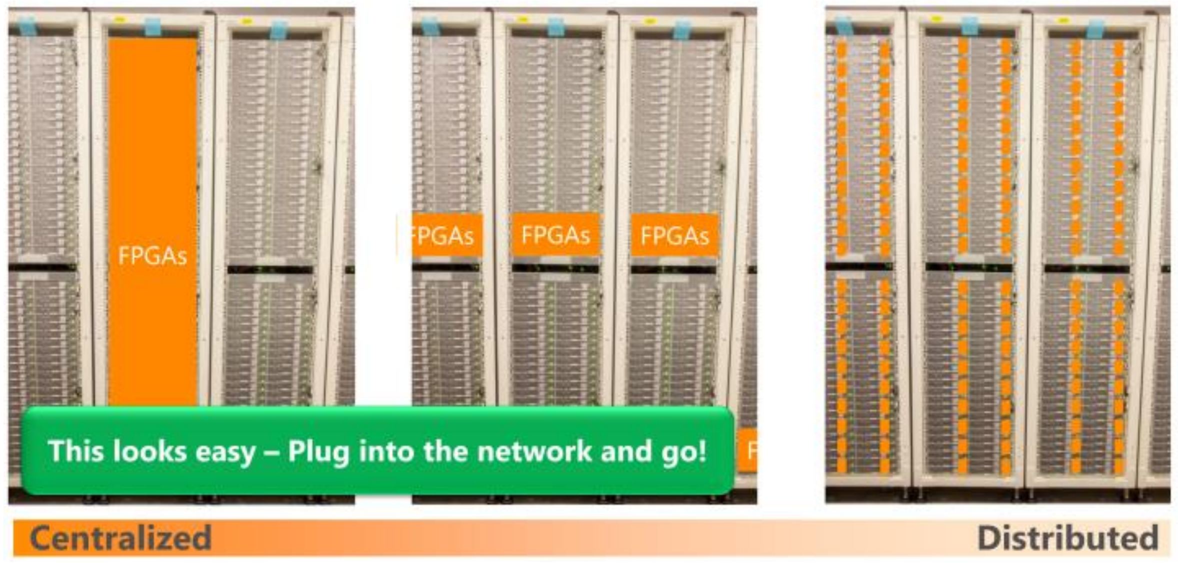
High-Level Synthesis (HLS)

- C/C++ → hardware description languages
- Front-end
  - Intermediate representation (IR) generation
  - Bitwidth reduction, loop unrolling, etc
- Back-end
  - Resource allocation, scheduling, binding
  - Register-transfer level (RTL) code generation
- Directives to tune latency/resource
  - Loop unrolling: duplicate loop copies
  - Loop pipelining: pipeline stages
  - Array partitioning: split memories



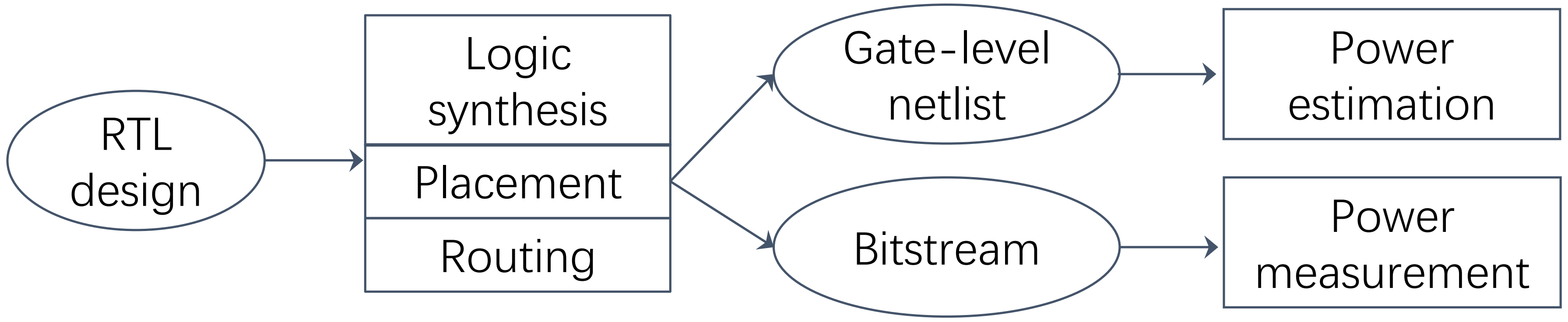
FPGA Power Issue

- A key design constraint
- Single FPGA: increase with size and density; heterogeneous systems
- Multiple FPGAs: large-scale applications / datacenters



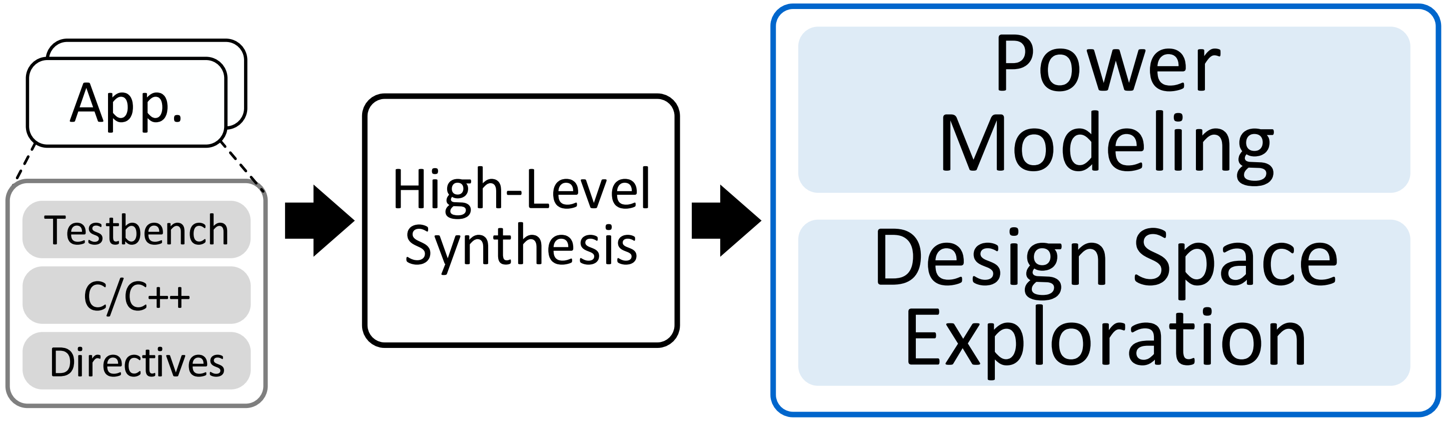
FPGA Power Evaluation

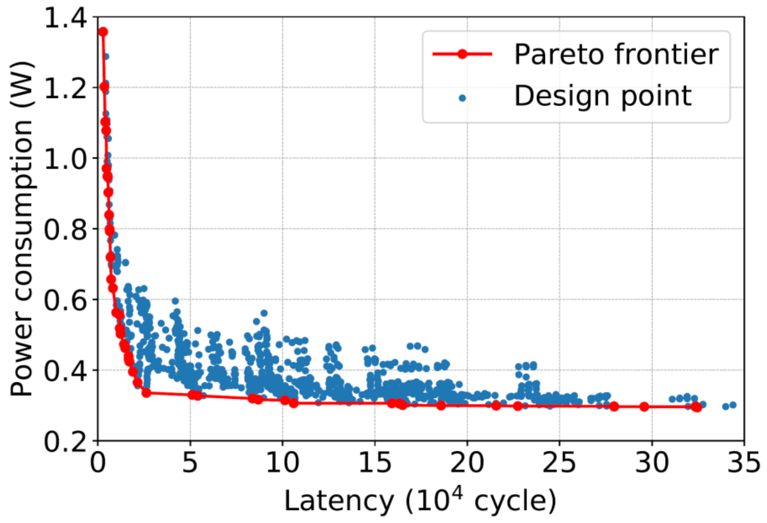
- Go through synthesis, placement and routing → a long time
- Estimation: gate-level power analysis
- Measurement: onboard measurement
- Design-time power analysis:
  - Trade off between performance / resource / power
  - → Large run-time overheads



Contributions

- HL-Pow framework: Fast and early-stage power estimation
- DSE framework: Power-oriented design space exploration (DSE)

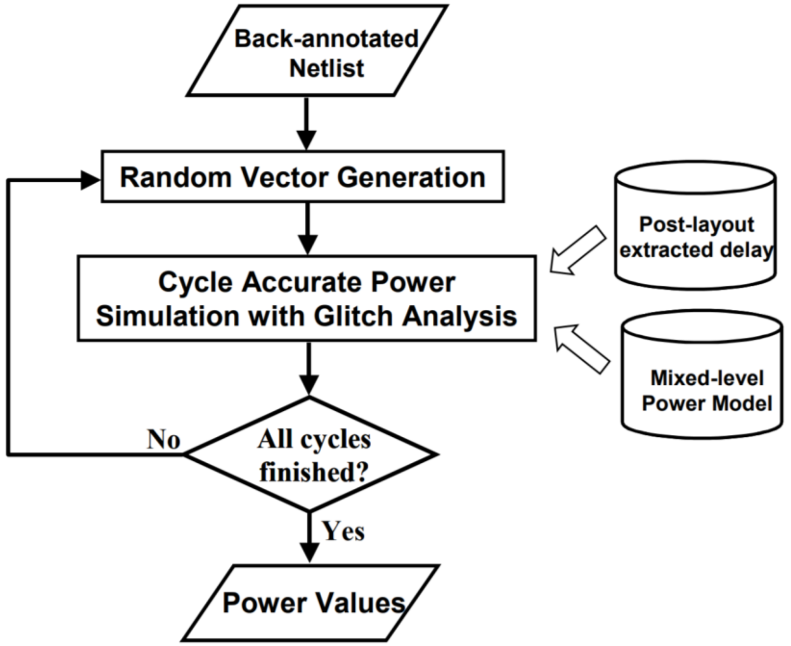


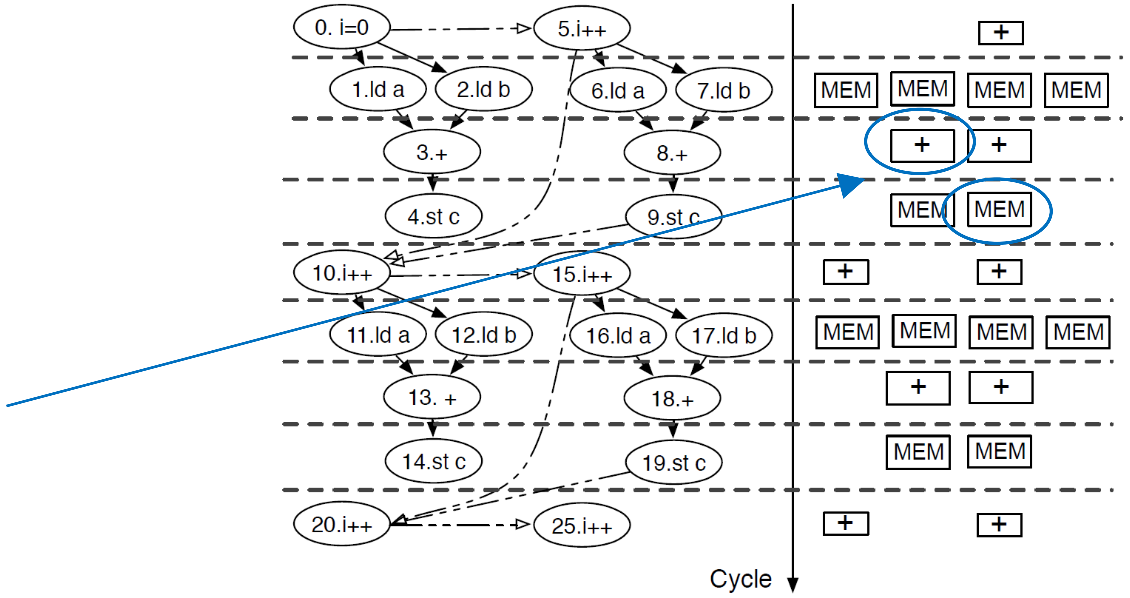


Outline

Low-Level Power Modeling

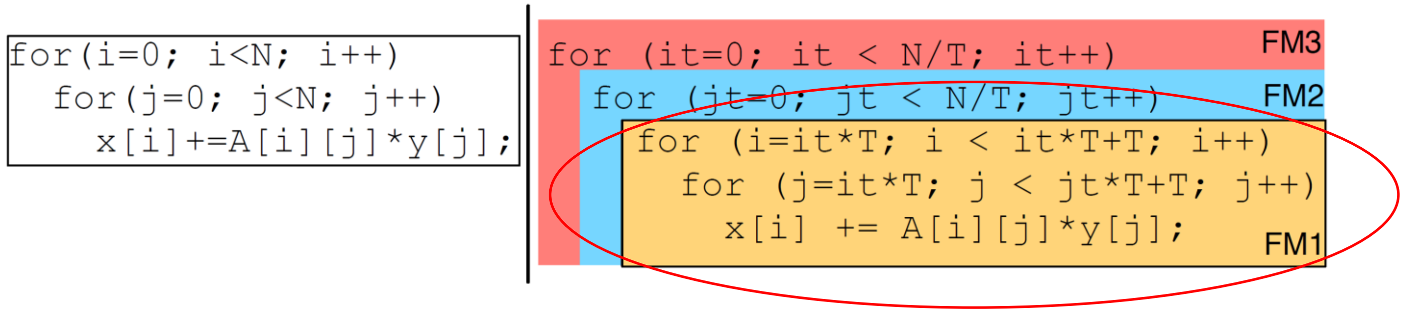
- Component-based power characterization
  - Power library: input-power mapping per component [shao et al, 2014]
  - Consider variations in bitwidths/cell selection, etc
  - Aggregation of component power for design power





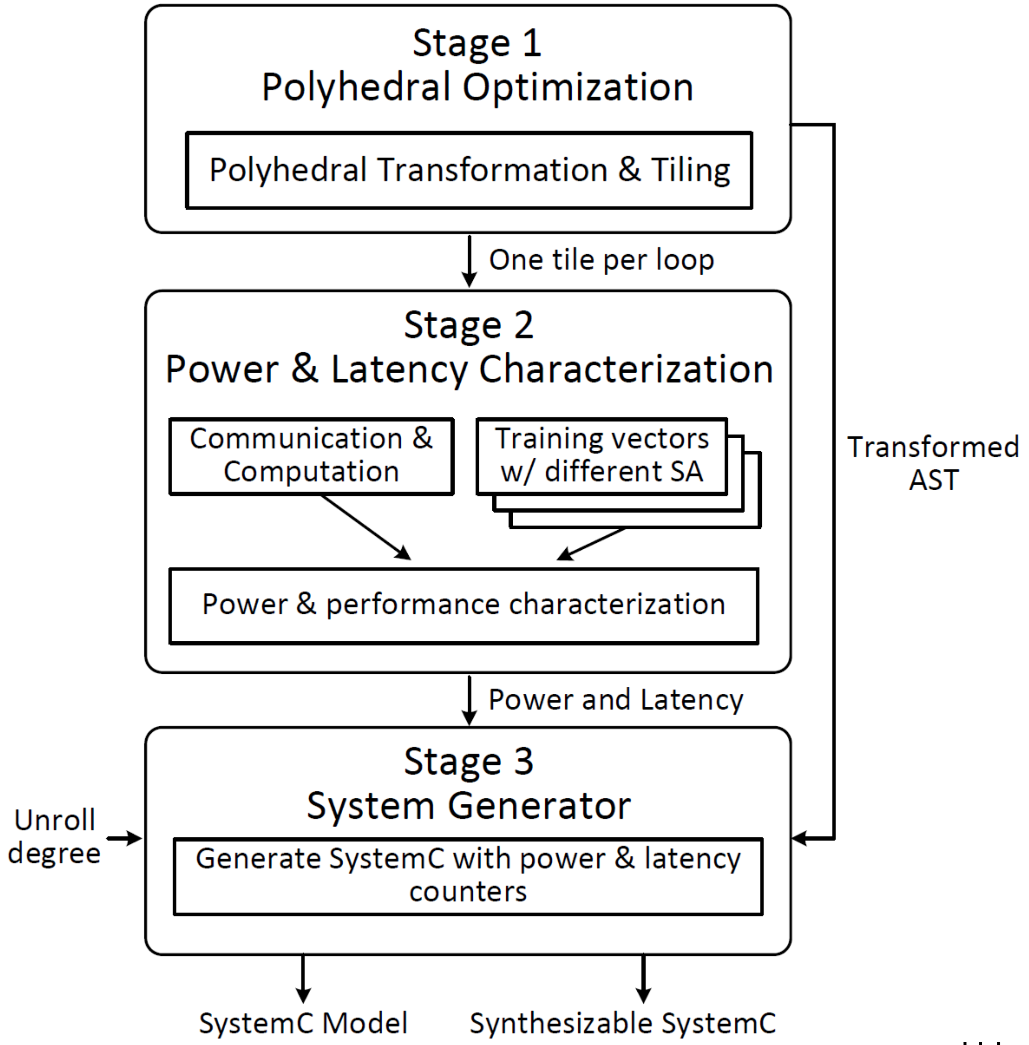
High-Level Power Modeling

- Tile-based power modeling
  - Target: affine functions [Zuo et al, 2015]
  - Decompose the functions into tiles
  - Power characterization for tiles



```
for(i=0; i<N; i++)
  for(j=0; j<N; j++)
    x[i]+=A[i][j]*y[j];
```

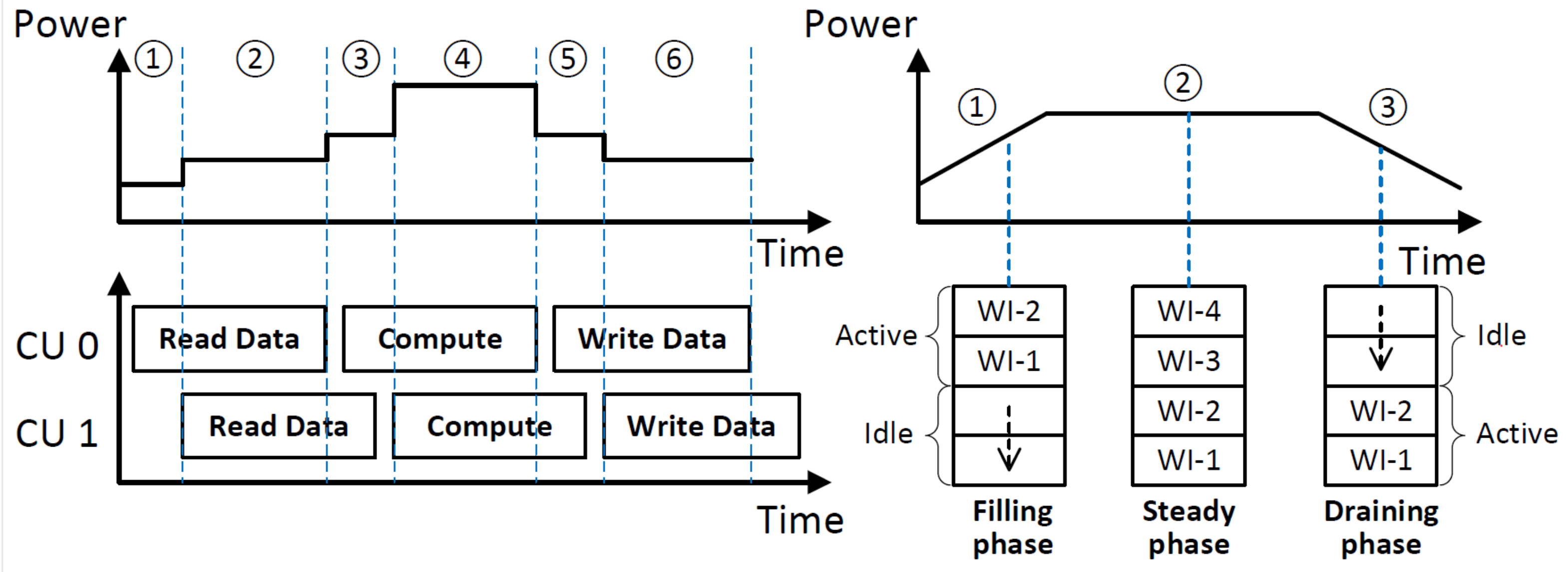
```
for (it=0; it < N/T; it++)
  for (jt=0; jt < N/T; jt++)
    for (i=it*T; i < it*T+T; i++)
      for (j=jt*T; j < jt*T+T; j++)
        x[i] += A[i][j]*y[j];
```



Stage 1
Polyhedral Optimization
Polyhedral Transformation & Tiling
One tile per loop
Stage 2
Power & Latency Characterization
Communication & Computation
Training vectors w/ different SA
Power & performance characterization
Power and Latency
Transformed AST
Unroll degree
Stage 3
System Generator
Generate SystemC with power & latency counters
SystemC Model
Synthesizable SystemC

High-Level Power Modeling

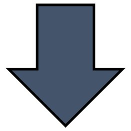
- Phase-based power modeling
  - Target: OpenCL
  - Work groups & work items in OpenCL [Liang et al, 2018]
  - Power phases in groups/items

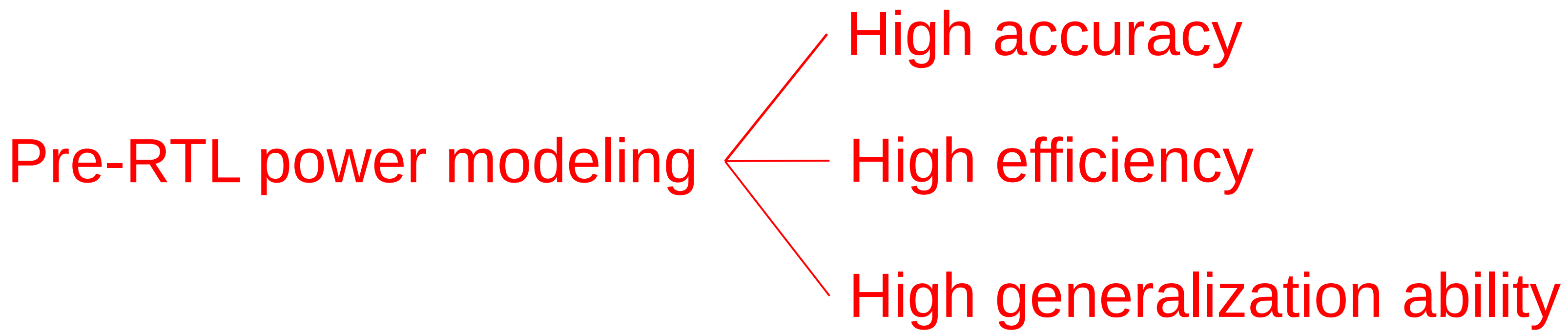


(a) Power phases decomposition at work-group level (b) Power phases decomposition at work-item level

Motivation

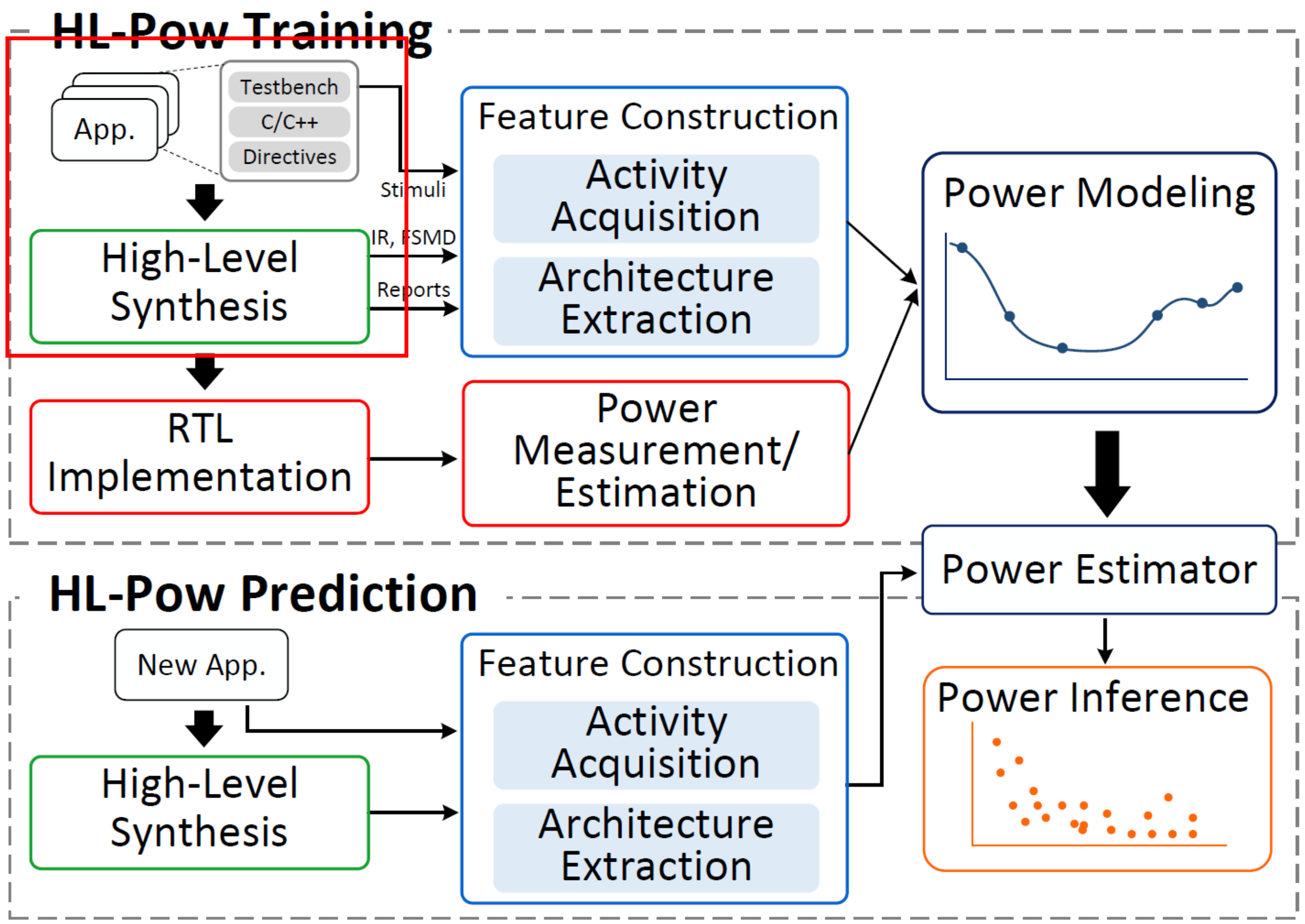
- Problems of power modeling works
  - Long time for power characterization
  - Require hardware expertise
  - Dedicate to specific code structures/applications





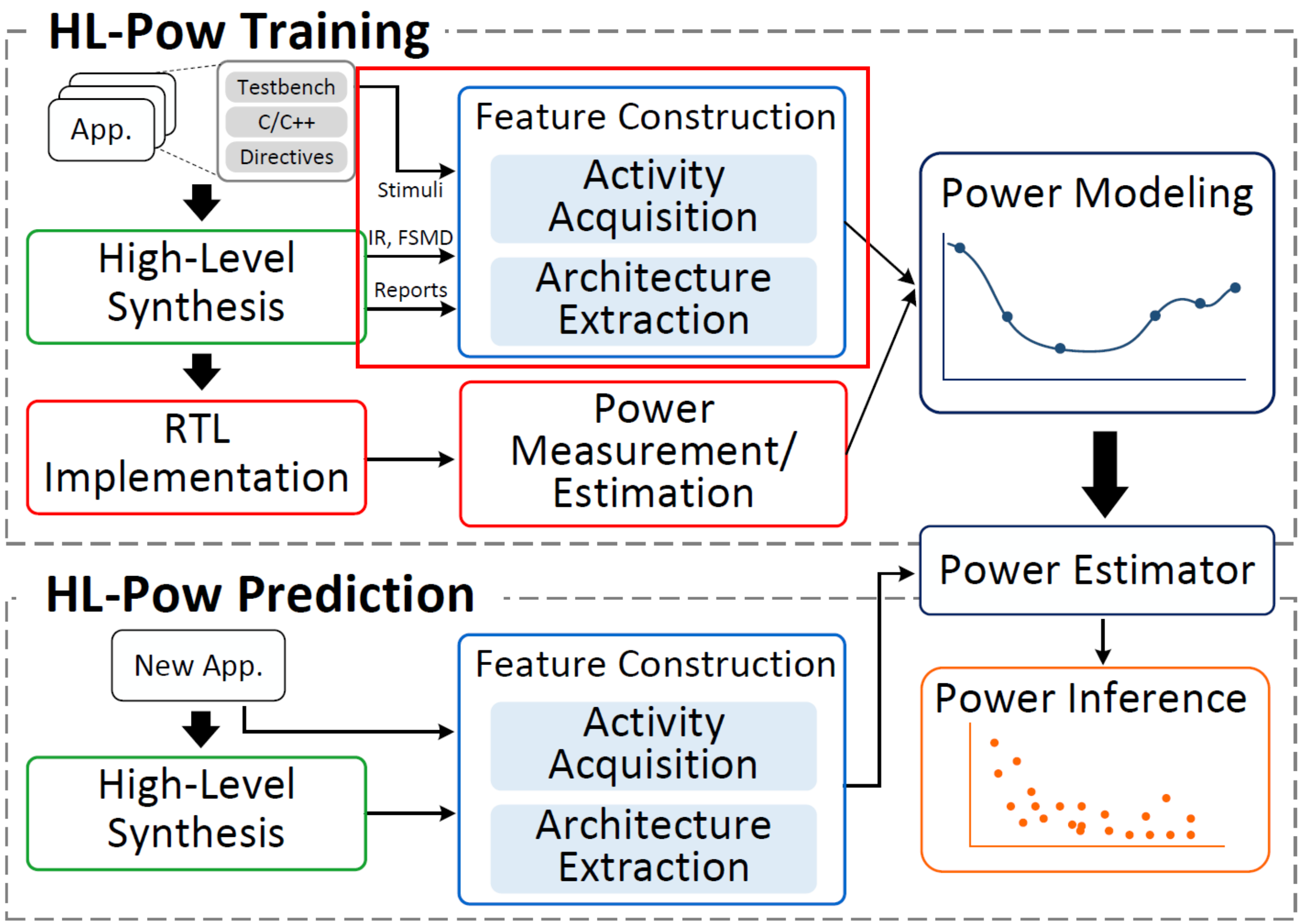
Outline

Overall Design Flow



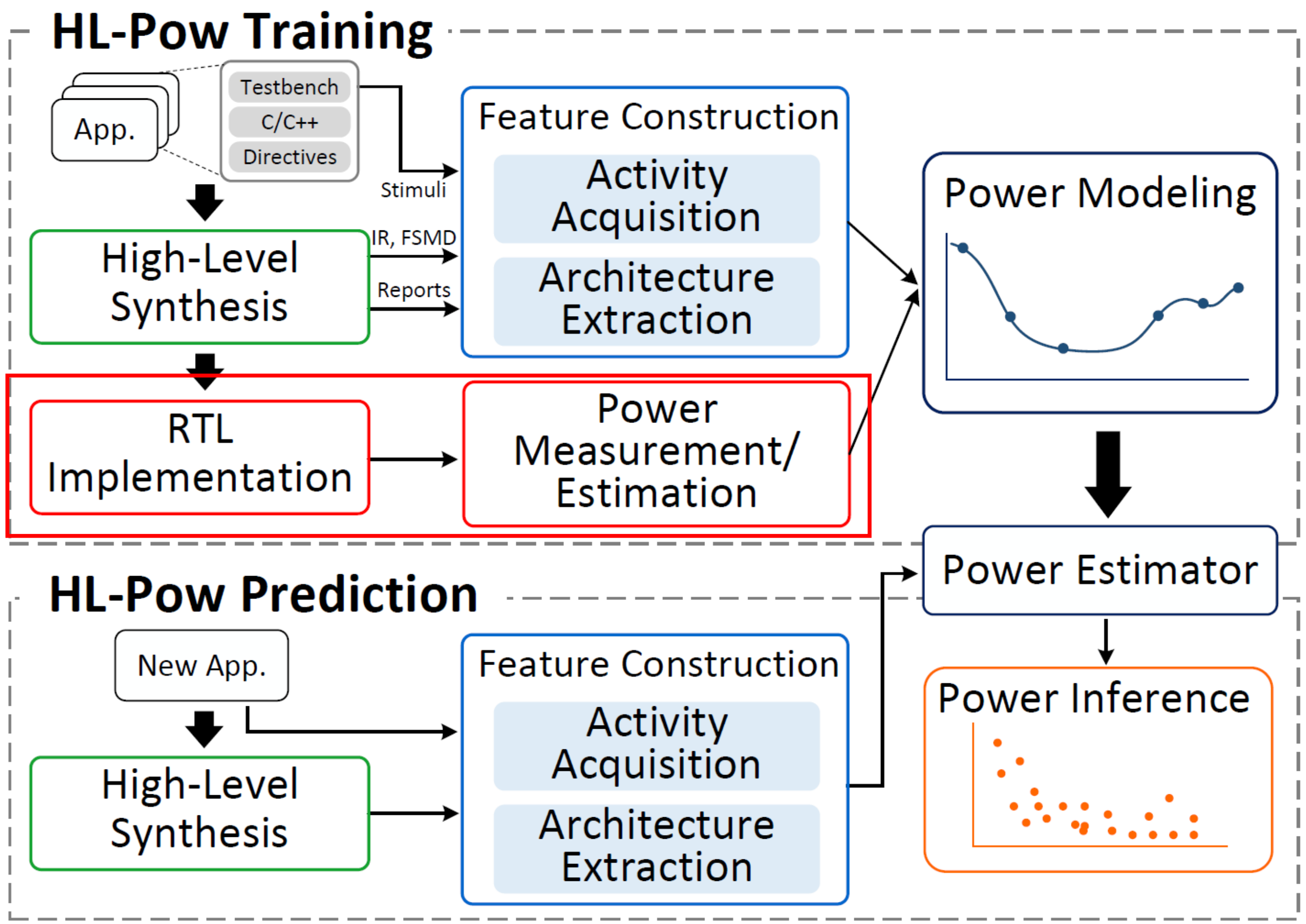
- Training phase
  - Collect training applications
  - Pass through HLS

Overall Design Flow



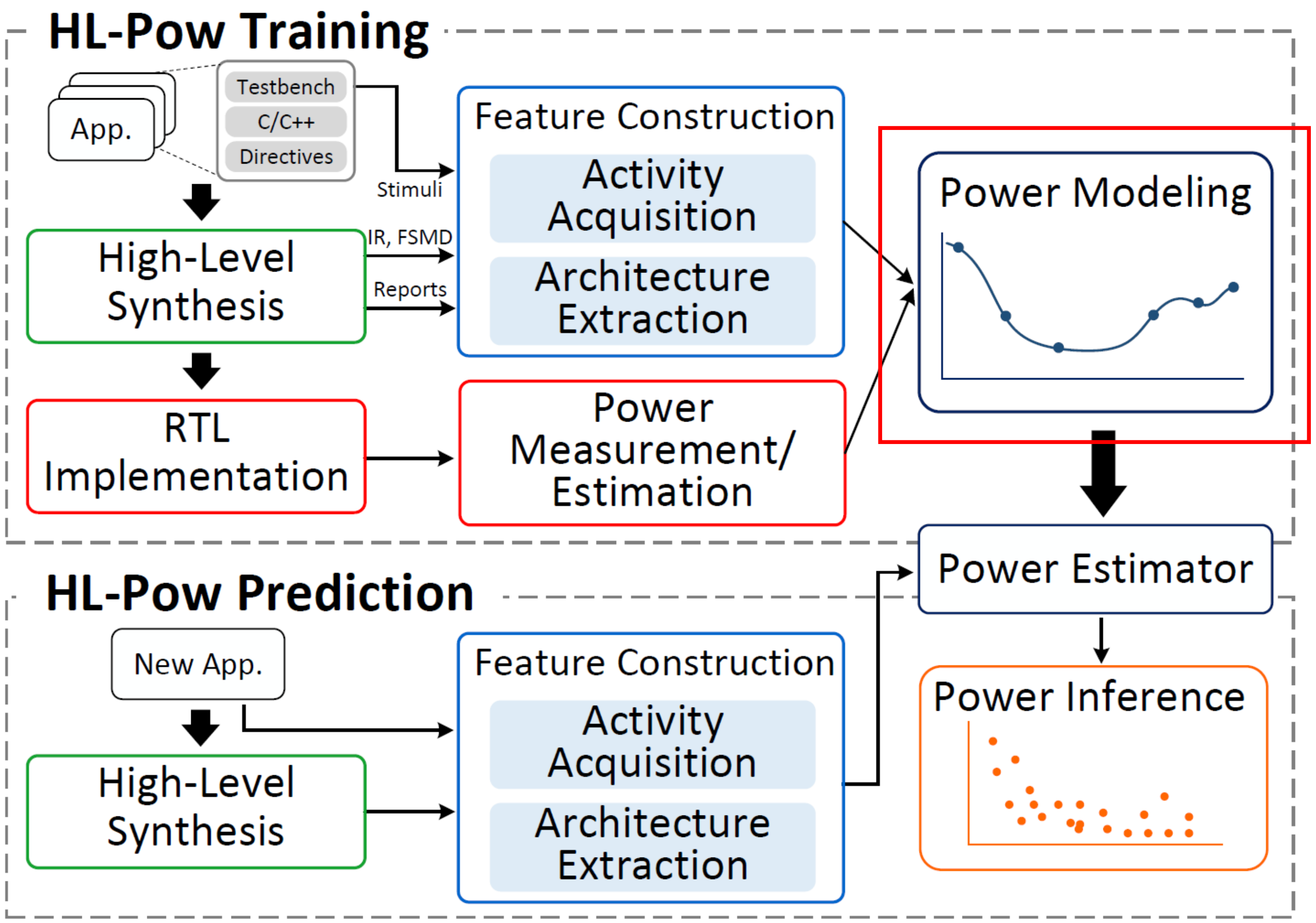
- Training phase
  - Collect training applications
  - Pass through HLS
  - Feature construction

Overall Design Flow



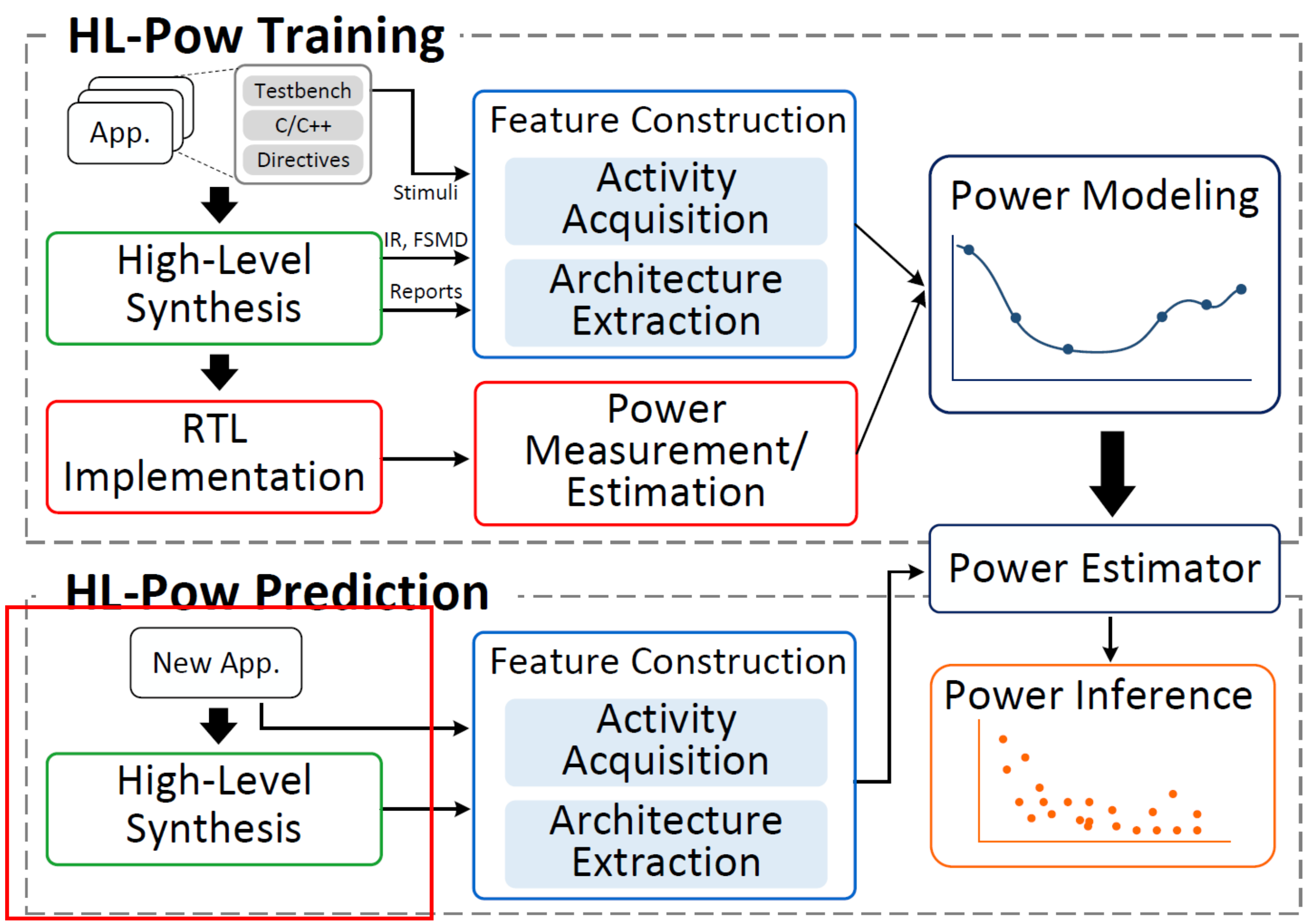
- Training phase
  - Collect training applications
  - Pass through HLS
  - Feature construction
  - RTL implementation
  - Collect ground truth power values

Overall Design Flow



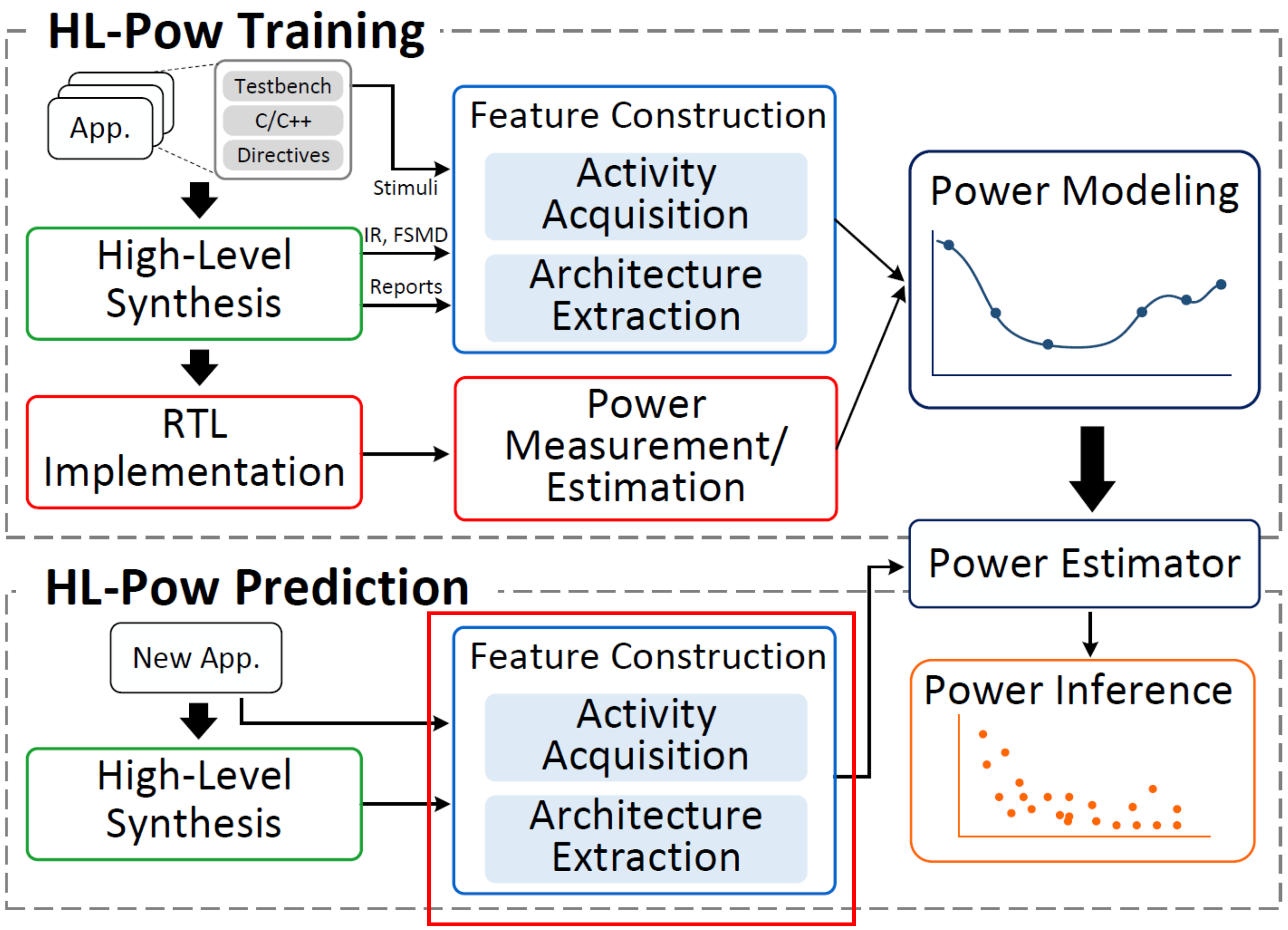
- Training phase
  - Collect training applications
  - Pass through HLS
  - Feature construction
  - RTL implementation
  - Collect ground truth power values
  - Power model training

Overall Design Flow



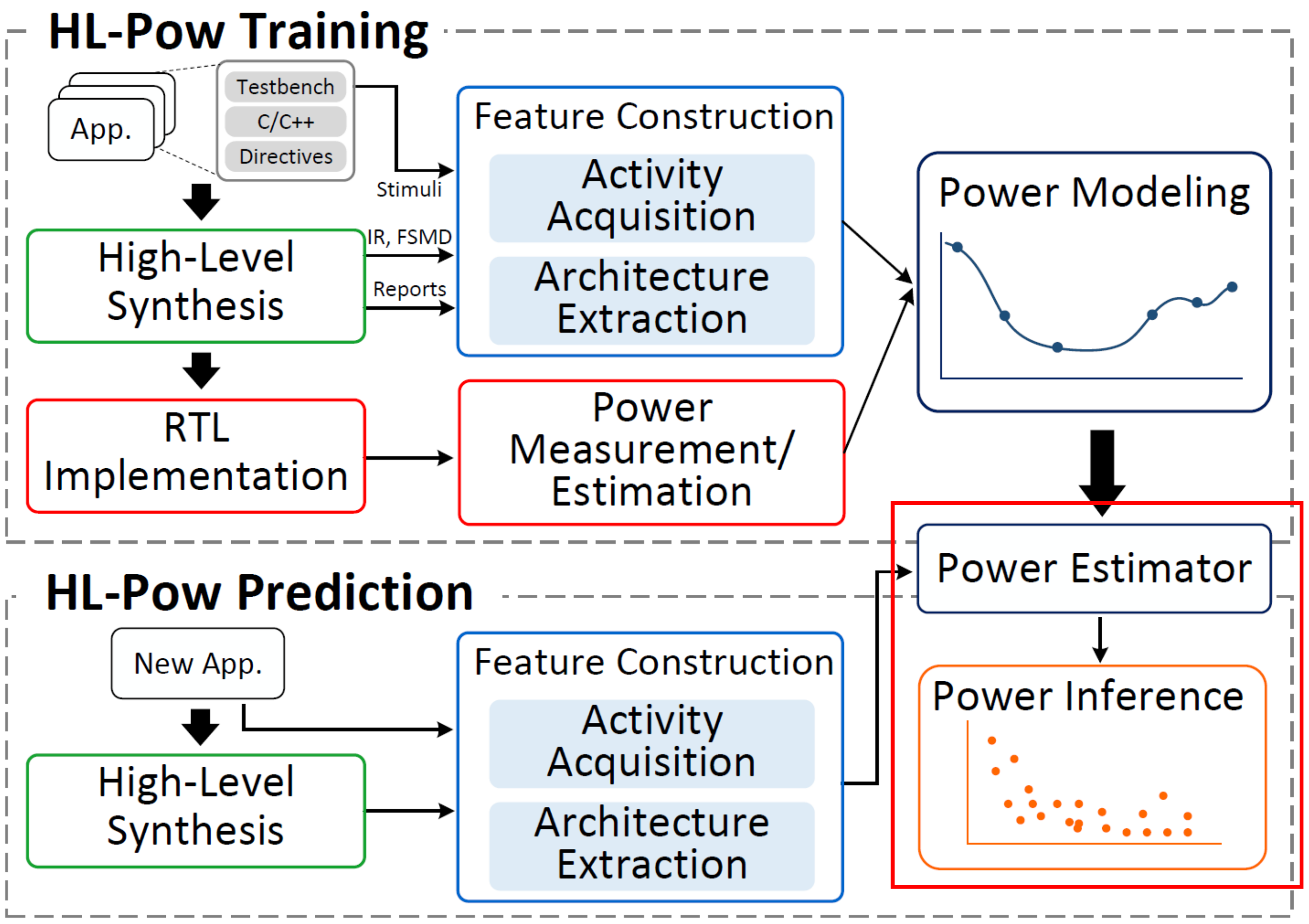
- Training phase
  - Collect training applications
  - Pass through HLS
  - Feature construction
  - RTL implementation
  - Collect ground truth power values
  - Power model training
- Prediction phase
  - Pass through HLS

Overall Design Flow



- Training phase
  - Collect training applications
  - Pass through HLS
  - Feature construction
  - RTL implementation
  - Collect ground truth power values
  - Power model training
- Prediction phase
  - Pass through HLS
  - Feature construction

Overall Design Flow



- Training phase
  - Collect training applications
  - Pass through HLS
  - Feature construction
  - RTL implementation
  - Collect ground truth power values
  - Power model training
- Prediction phase
  - Pass through HLS
  - Feature construction
  - Power inference

Data Collection

- FPGA power decomposition
  - Static power: leakage power when the design is powered up, related to the scale of the design
  - Dynamic power: proportional to signal activity, capacitance, voltage and frequency

$$P_{dyn} = \sum_{i \in N} \underline{\alpha_i} C_i V_{dd}^2 f$$

- Data files collected from HLS for each design point
  - Overall design: HLS report (xxx.verbose.rpt.xml), IR code (a.o.3.bc)
  - Individual operator: IR operator information (xxx.adb), RTL operator information (xxx.adb.xml)

Feature Construction: Architecture Features

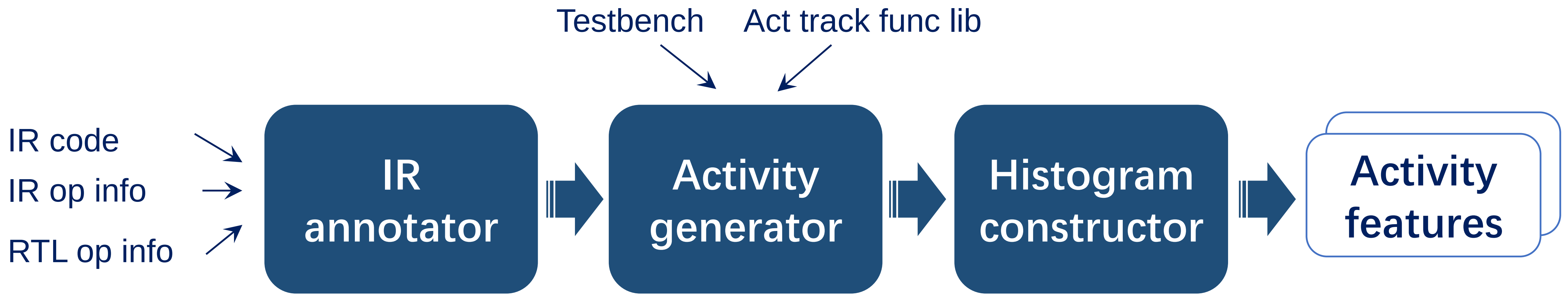
- Resource Utilization
  - Look-up table (LUT)
  - Digital signal processing (DSP)
  - Flip-flop (FF)
  - Block random access memory (BRAM)
- Performance
  - Latency in cycles
  - Frequency in nano-seconds
- Scaling factors of metrics
  - Baseline design: the design point without any optimization
  - Normalize across different applications

$$SF_M = \frac{M_{current}}{M_{base}}$$

Feature Construction: Activity Features

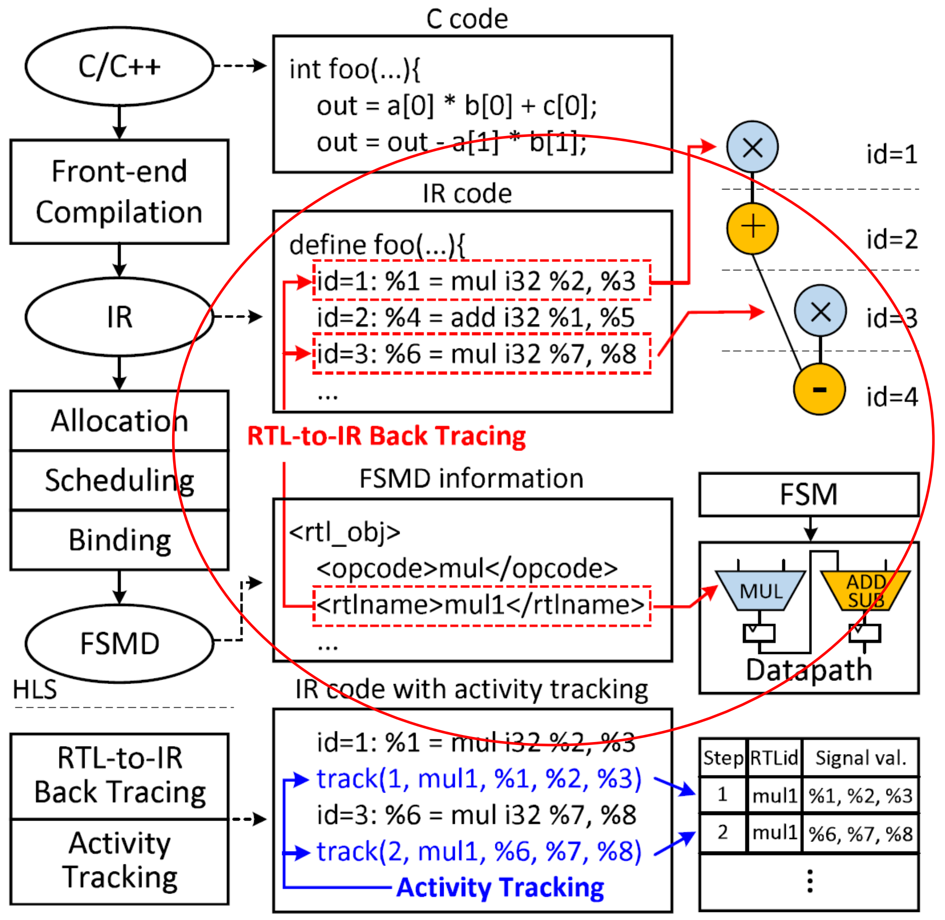
- Conduct in high-level instead of gate-level
- IR annotator: add activity tracking functions in IR
- Activity generator: collect cycle-level activities
- Histogram constructor: construct activity features

$$P_{dyn} = \sum_{i \in N} \underline{\alpha_i} C_i V_{dd}^2 f$$



Activity Features: IR Annotator

- Instrument the IR code with activity tracking functions for each RTL operator
- IR operator: in IR code
- RTL operator: in hardware
- RTL-to-IR back tracing
  - Multiple IR ops ⟷ one RTL op
  - Match IR ops to RTL ops by netlist name



```
int foo(...){
   out = a[0] * b[0] + c[0];
   out = out - a[1] * b[1];
```

```
define foo(...){
   id=1: %1 = mul i32 %2, %3
   id=2: %4 = add i32 %1, %5
   id=3: %6 = mul i32 %7, %8
   ...
```

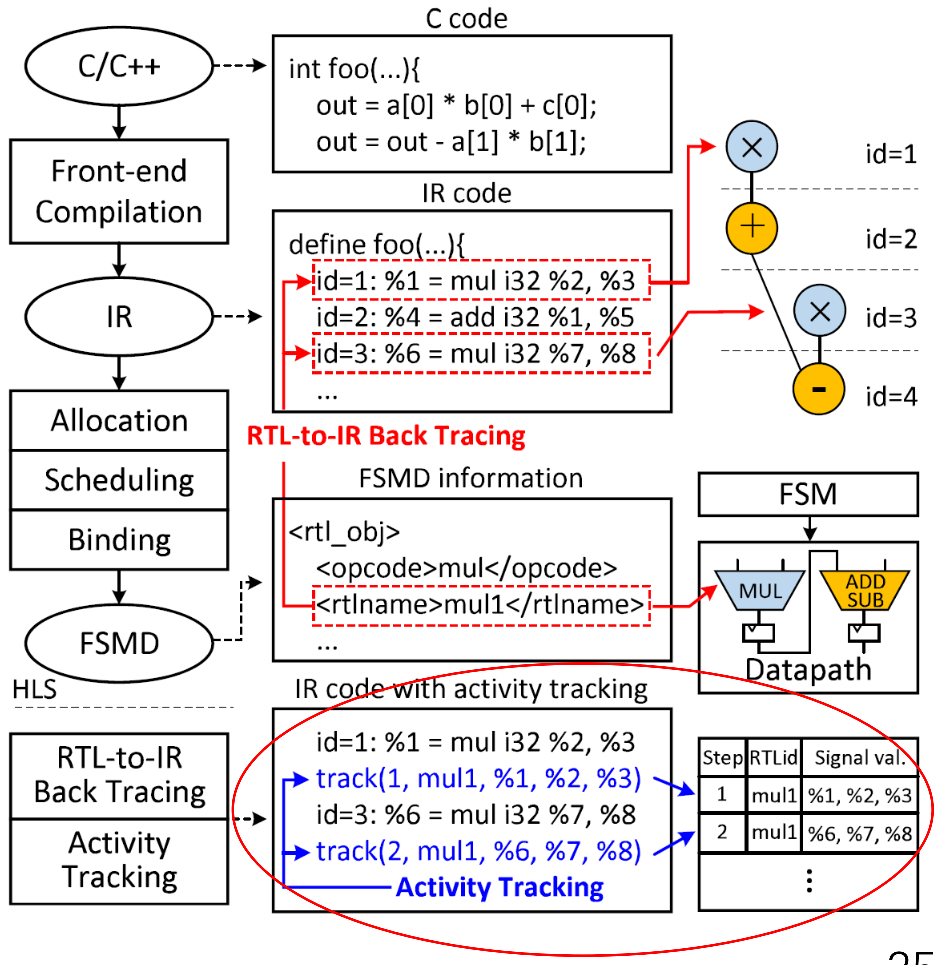
```
<rtl_obj>
   <opcode>mul</opcode>
   <rtlname>mul1</rtlname>
   ...
```

```
   id=1: %1 = mul i32 %2, %3
   track(1, mul1, %1, %2, %3)
   id=3: %6 = mul i32 %7, %8
   track(2, mul1, %6, %7, %8)
```

Step	RTLid	Signal val.
1	mul1	%1, %2, %3
2	mul1	%6, %7, %8
⋮		

Activity Features: IR Annotator

- Instrument the IR code with activity tracking functions for each RTL operator
- IR operator: in IR code
- RTL operator: in hardware
- RTL-to-IR back tracing
  - Multiple IR ops $\longleftrightarrow$ one RTL op
  - Match IR ops to RTL ops by netlist name
- Activity tracking
  - Add activity tacking functions
  - Record cycle-level input/output + RTL op ID
- Generate an annotated IR code



C code:

```
int foo(...){
   out = a[0] * b[0] + c[0];
   out = out - a[1] * b[1];
```

IR code:

```
define foo(...){
   id=1: %1 = mul i32 %2, %3
   id=2: %4 = add i32 %1, %5
   id=3: %6 = mul i32 %7, %8
   ...
```

FSMD information:

```
<rtl_obj>
   <opcode>mul</opcode>
   <rtlname>mul1</rtlname>
   ...
```

IR code with activity tracking:

```
   id=1: %1 = mul i32 %2, %3
   track(1, mul1, %1, %2, %3)
   id=3: %6 = mul i32 %7, %8
   track(2, mul1, %6, %7, %8)
```

Step	RTLid	Signal val.
1	mul1	%1, %2, %3
2	mul1	%6, %7, %8
⋮		

Activity Features: Activity Generator

- Compute average activities of each RTL operator
- Compile into link files (.o)

⬇

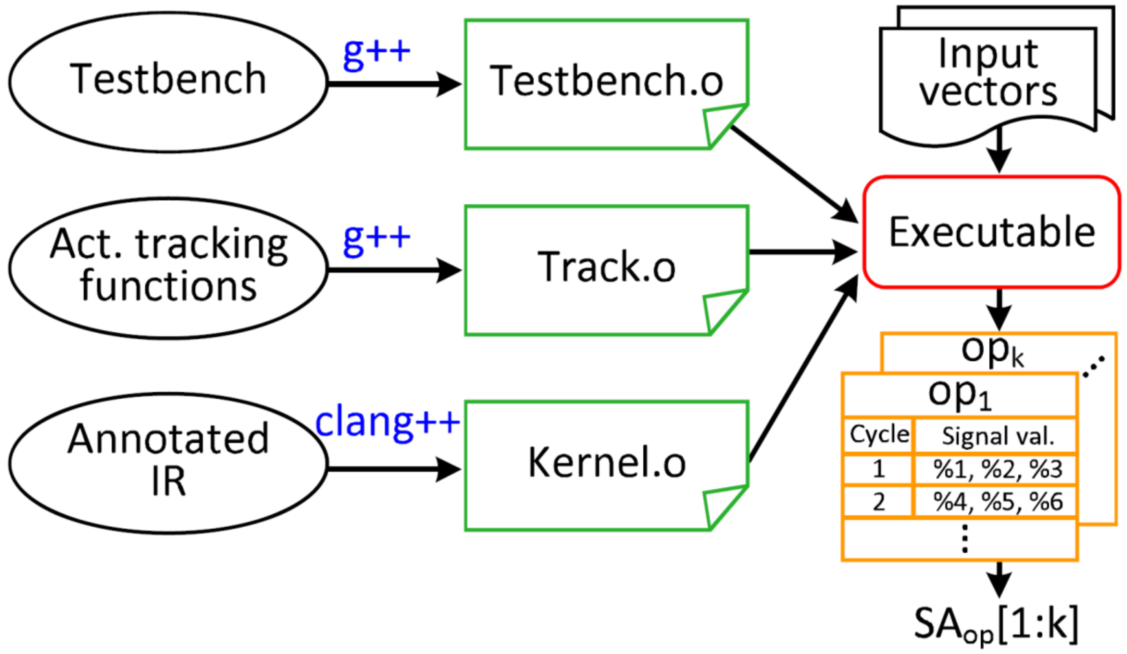
- Combine, compile into executable

⬇

- Run with input vectors

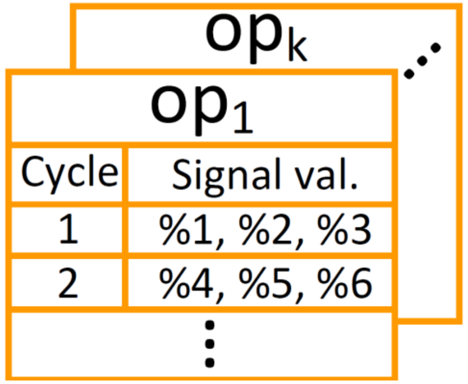
⬇

- Create an activity table per RTL op



Activity Features: Activity Generator

- Hamming distance: count differences bit by bit
- Average the activity per RTL op
- Amortize the activities over the total execution



Cycle	Signal val.
1	%1, %2, %3
2	%4, %5, %6
⋮	

Hamming distance

$$SA_{op} = \frac{\sum_{i=1}^{M_{op}} \sum_{j=1}^{N_{op}} HD(\boldsymbol{s}(i,j),\ \boldsymbol{s}(i,j-1))}{M_{op} \cdot N_{op}}$$

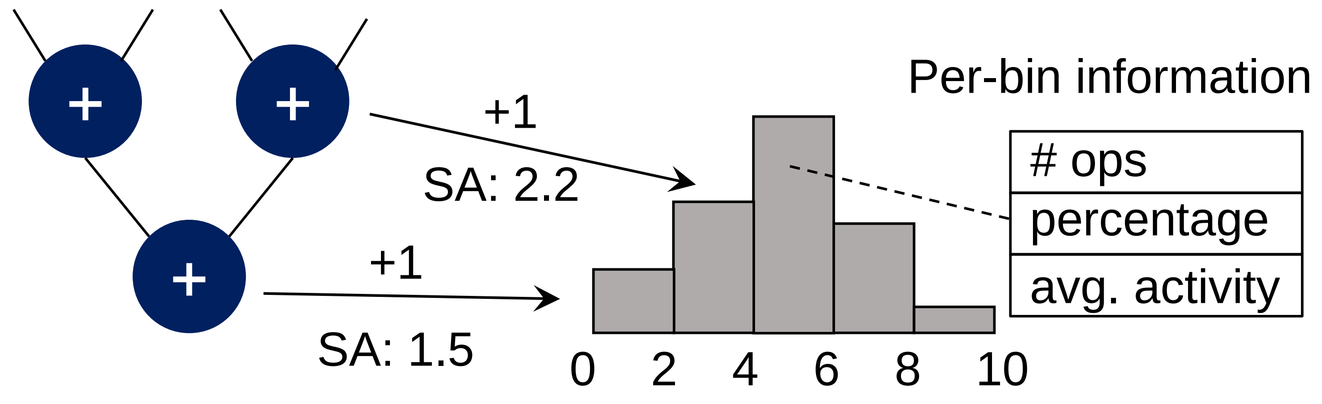
Avg. across cycles & operands

$$SA_{scaled} = \frac{N_{op}}{L} \cdot SA_{op}$$

Op exe cycles / latency

Activity Features: Histogram Constructor

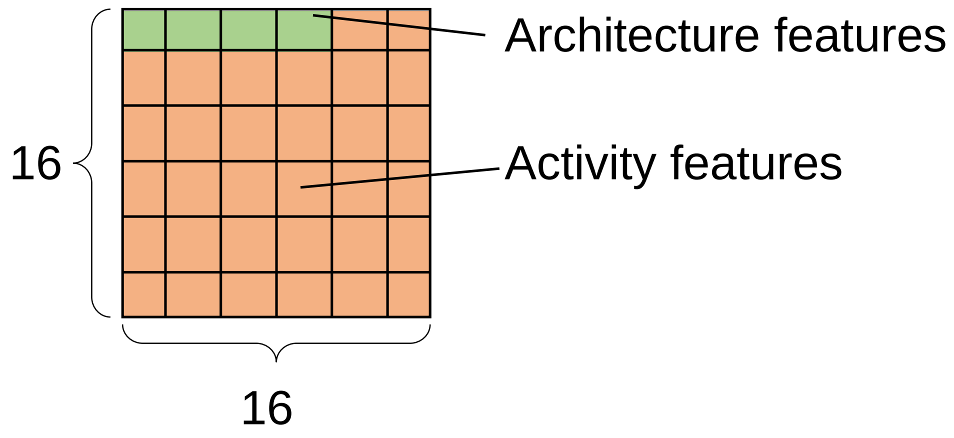
- Operator activities → features
- Fix the size of the feature set
- Use a histogram representation
- Allocate a histogram per opcode
- Features per bin: #ops, percentage, average activity



Power Modeling

- 256 features: 11 architecture features, 245 activity features
- Ground truth power values through onboard measurement
- Regression models
  - Linear regression: classic linear model, LASSO
  - Support vector machine (SVM): RBF kernel
  - Tree-based model: decision tree, ensemble model
  - Neural network: CNN, ResNet

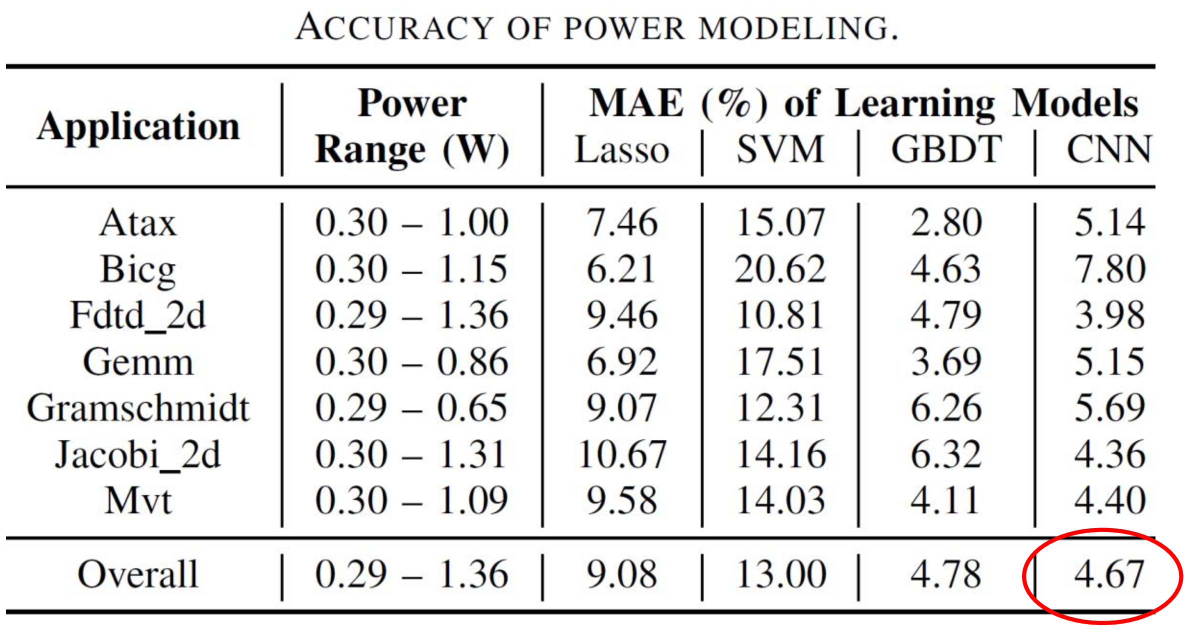
Data normalization
Feature selection
Cross validation



16-by-16 mesh
Data normalization
Based on widely used instances

Experimental Results

- Experimental setup
  - Training: 15 applications / 8784 design points
  - Testing: 6 applications / 2542 design points (> 20%)
- Accuracy of power modeling
  - Show the best instance per type
  - Linear/SVM: ~10% test error
  - CNN: 4.67% test error



ACCURACY OF POWER MODELING.

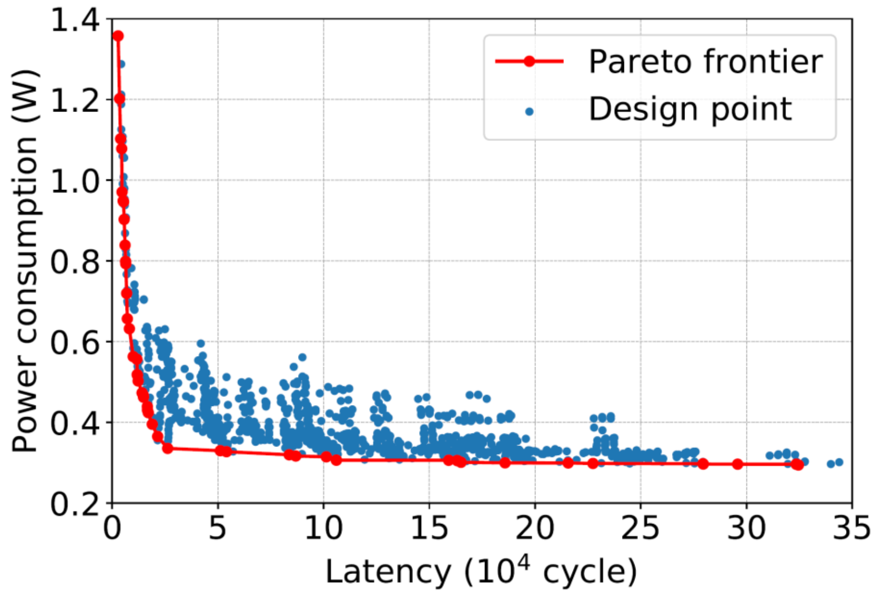
Application	Power Range (W)	MAE (%) of Learning Models			
		Lasso	SVM	GBDT	CNN
Atax	0.30 – 1.00	7.46	15.07	2.80	5.14
Bicg	0.30 – 1.15	6.21	20.62	4.63	7.80
Fdtd_2d	0.29 – 1.36	9.46	10.81	4.79	3.98
Gemm	0.30 – 0.86	6.92	17.51	3.69	5.15
Gramschmidt	0.29 – 0.65	9.07	12.31	6.26	5.69
Jacobi_2d	0.30 – 1.31	10.67	14.16	6.32	4.36
Mvt	0.30 – 1.09	9.58	14.03	4.11	4.40
Overall	0.29 – 1.36	9.08	13.00	4.78	4.67

Outline

- Introduction
- Related Work
- Power Modeling Framework
- **Design Space Exploration**
- Conclusion

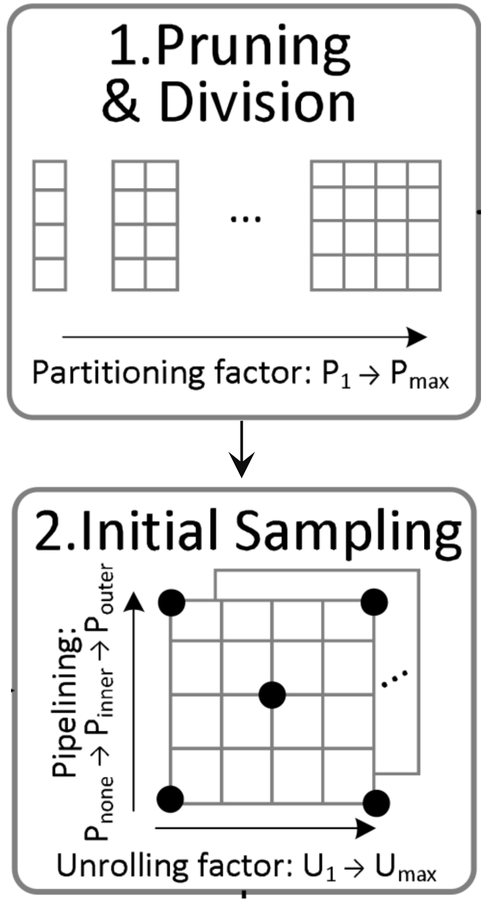
Design Space Exploration (DSE)

- Latency: by HLS
- Power: by our power modeling flow (HL-Pow)
- Trade off between latency and power
  - Brute-force DSE: HLS run-time overheads
  - To speed up DSE: design point sampling



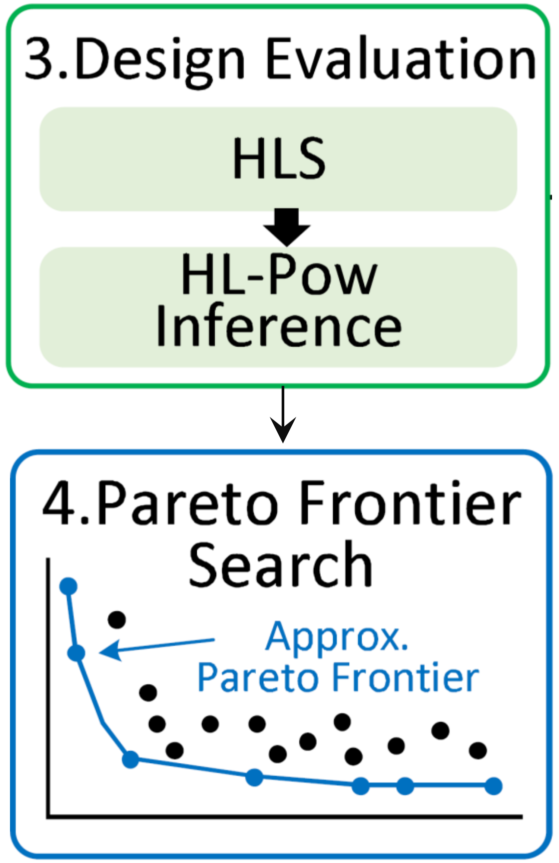
Design Space Exploration (DSE)

- 1. Design space pruning & division
  - Remove repetitive design points
  - Pipeline outer loop = unroll inner loops
  - Divide design space by partitioning factor
- 2. Initial sampling
  - Collect the first set of design points to assess
  - Power trend
    - Pipeline outer loop > pipeline inner loop > no pipeline
    - Large unrolling factor > small unrolling factor
  - Grid representation of unroll/pipeline
  - Select boundary and middle points


1.Pruning & Division
...
Partitioning factor: $P_1 \rightarrow P_{max}$
2.Initial Sampling
Pipelining: $P_{none} \rightarrow P_{inner} \rightarrow P_{outer}$
...
Unrolling factor: $U_1 \rightarrow U_{max}$

Design Space Exploration (DSE)

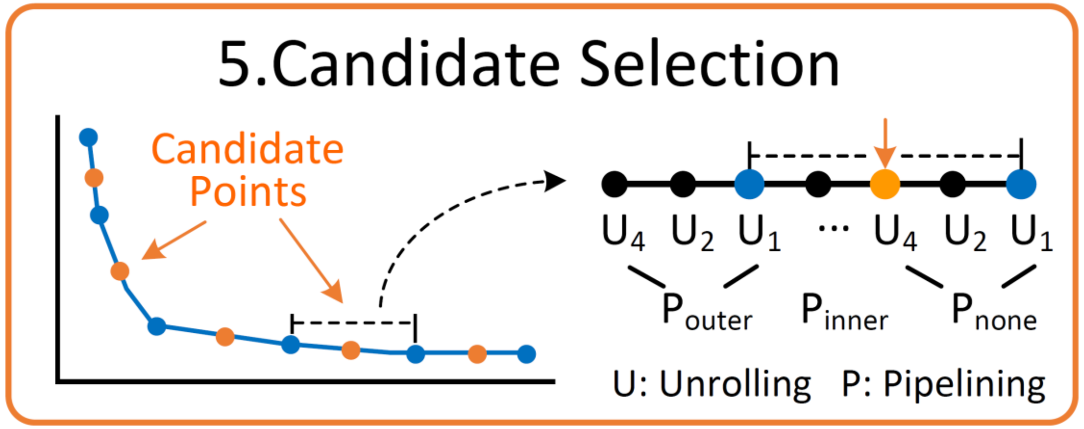
- 3. Design evaluation
  - Use HLS for latency evaluation
  - Use HL-Pow for fast power evaluation
- 4. Pareto frontier search
  - Approximate Pareto frontier from sampling points
  - Trade off between latency and power



Design Space Exploration (DSE)

- 5. Candidate selection
  - Standard deviation reduction (SDR): the higher SDR, the larger impact
  - SDR: pipeline > unroll → latency/power impact: pipeline > unroll
  - Order sequence: first pipeline, second unroll
  - Select middle points of two pareto-optimal points

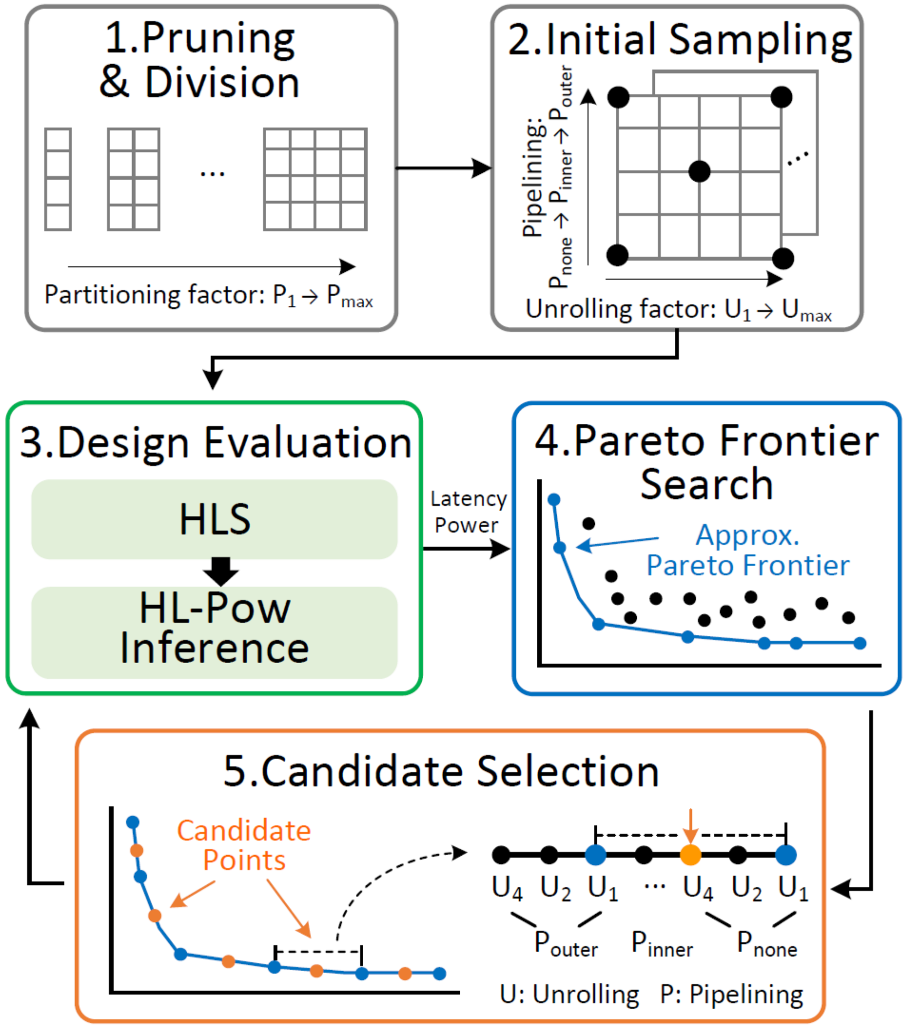
$$SDR = sd(T) - \sum_i \frac{|T_i|}{|T|} \times sd(T_i)$$



Design Space Exploration (DSE)

- Algorithm decomposition
  - 1. Design space pruning & division
  - 2. Initial sampling
  - 3. Design evaluation
  - 4. Pareto frontier search
  - 5. Candidate selection

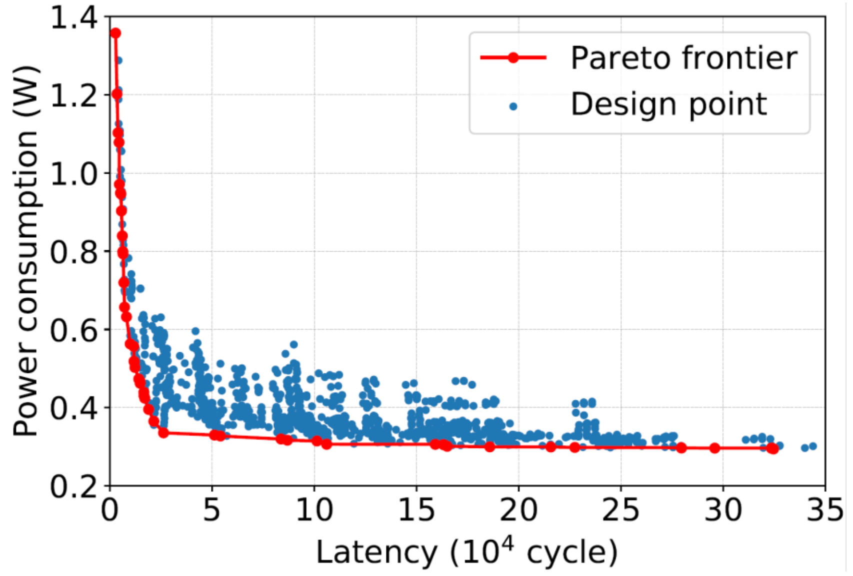
iterate

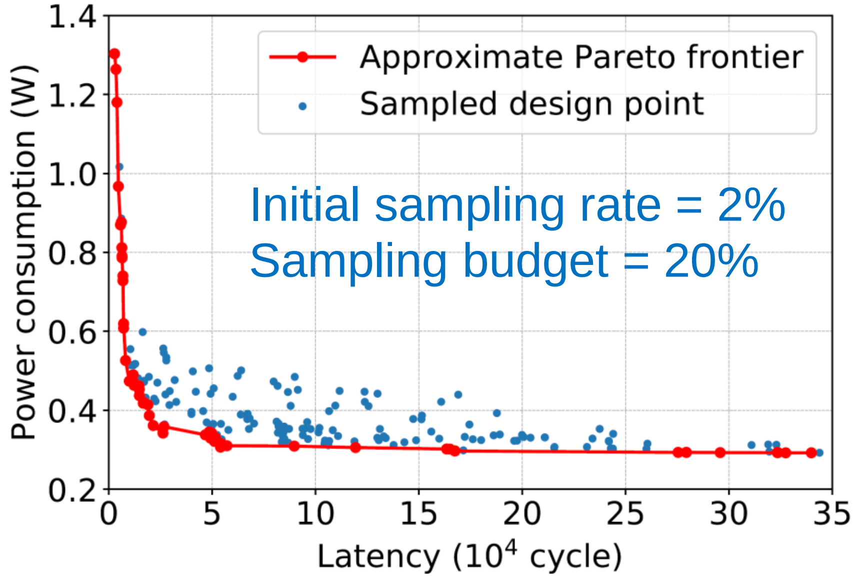


1.Pruning & Division
...
Partitioning factor: $P_1 \rightarrow P_{max}$
2.Initial Sampling
Pipelining: $P_{none} \rightarrow P_{inner} \rightarrow P_{outer}$
Unrolling factor: $U_1 \rightarrow U_{max}$
3.Design Evaluation
HLS
HL-Pow Inference
Latency
Power
4.Pareto Frontier Search
Approx. Pareto Frontier
5.Candidate Selection
Candidate Points
U_4 U_2 U_1 ··· U_4 U_2 U_1
P_{outer} P_{inner} P_{none}
U: Unrolling P: Pipelining

Experimental Results

- Design space exploration
  - Metric: *average distance to reference set (ADRS),* to see how close two Pareto set is
  - Initial sampling rates: different rates converge, but a small rate benefits small sampling budgets
  - Total sampling budgets:
    - 20%: ADRS = 2.35%
    - 40%: ADRS = 1.84%
  - ADRS decreases rapidly as the sampling budget increases from the start





Outline

- Introduction
- Related Work
- Power Modeling Framework
- Design Space Exploration
- **Conclusion**

Conclusion

HL-Pow: HLS power modeling

- Early-stage
- Accurate
- Fast
- High generalization ability

+

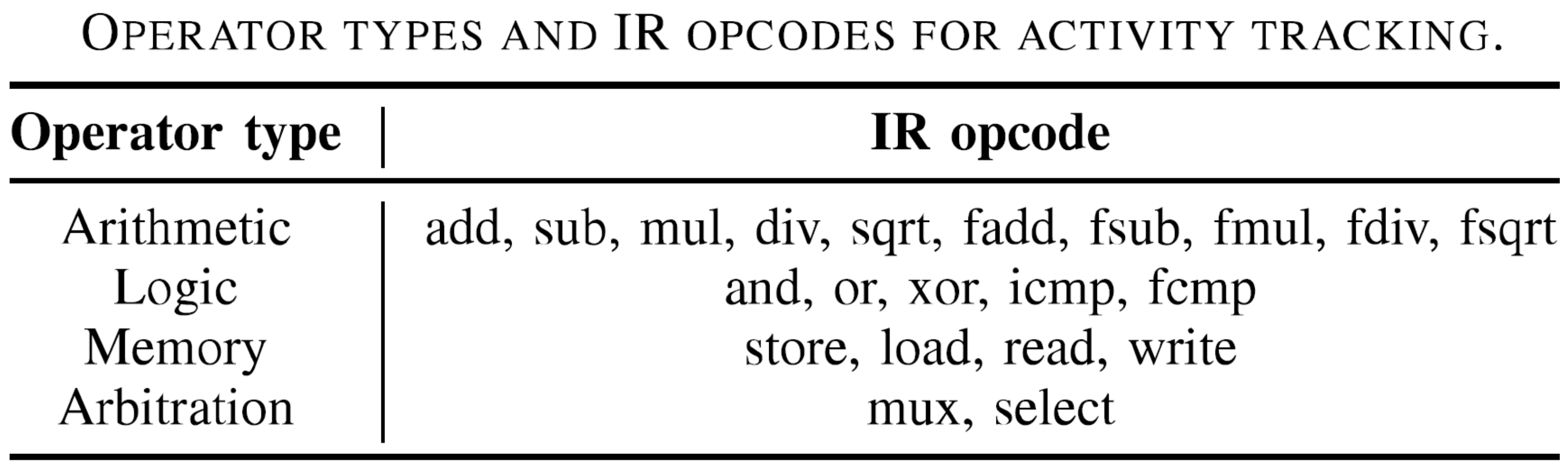
Power-oriented DSE

- Characteristics of HLS directives
- Trim down design space
- Iterative search flow

THANK YOU

Q & A

Operator type



OPERATOR TYPES AND IR OPCODES FOR ACTIVITY TRACKING.

Operator type	**IR opcode**
Arithmetic	add, sub, mul, div, sqrt, fadd, fsub, fmul, fdiv, fsqrt
Logic	and, or, xor, icmp, fcmp
Memory	store, load, read, write
Arbitration	mux, select

Directive

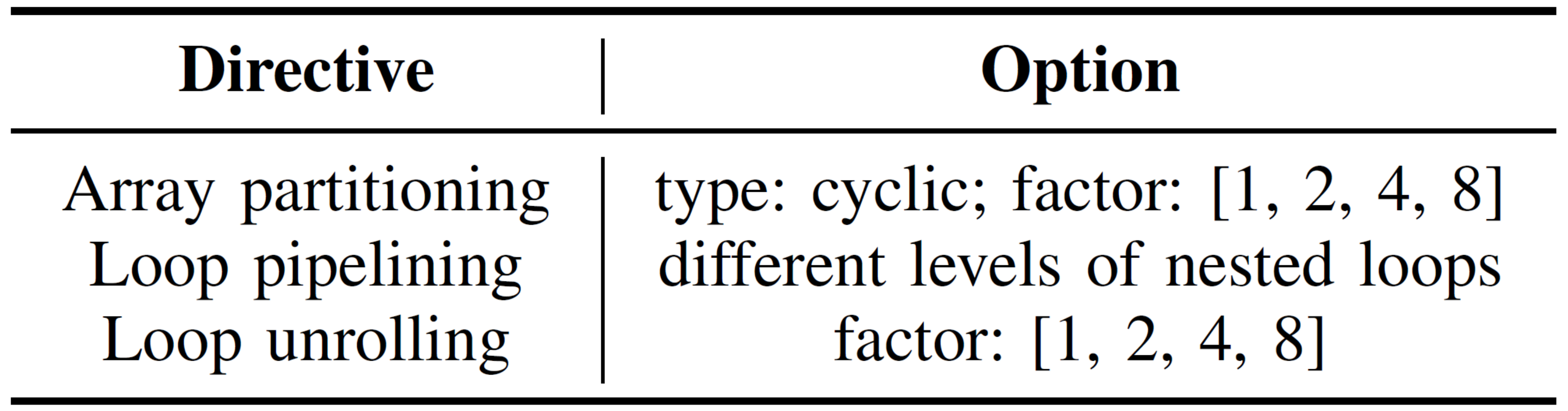


TABLE II
DIRECTIVE OPTIONS SUITABLE FOR THE TARGET PLATFORM.

Directive	**Option**
Array partitioning	type: cyclic; factor: [1, 2, 4, 8]
Loop pipelining	different levels of nested loops
Loop unrolling	factor: [1, 2, 4, 8]

ADRS

$$ADRS(\bar{P}, P) = \left[\frac{1}{|P|}\sum_{p \in P} \min_{\bar{p} \in \bar{P}} (\delta(\bar{p}, p))\right] \times 100\%,$$

$$\delta(\bar{p}, p) = \max\left\{0, \frac{Lat_{\bar{p}} - Lat_{p}}{Lat_{p}}, \frac{Pwr_{\bar{p}} - Pwr_{p}}{Pwr_{p}}\right\},$$