LOUISIANA STATE UNIVERSITY

Universal Rules Guided Design Parameter Selection for Soft Error Resilient Processors

Lide Duan, Ying Zhang, Bin Li, and Lu Peng
Louisiana State University

Soft Errors

- Interference from environment may cause bit flips in a computer.
- Only the data is destroyed, but the circuit is not damaged. Called Soft Errors (vs. Hard Errors)
- Not all soft errors will result in visible errors in the program output
  - Empty processor structure entries
  - Branch predictor

Architectural Vulnerability Factor

- AVF: the probability that a raw soft error finally produces a visible error in the output
- To calculate AVF (expensive):
  - Statistical Fault Injection
  - Architectural Correct Execution (ACE) Analysis
- Effective Soft Error Rate (SER)

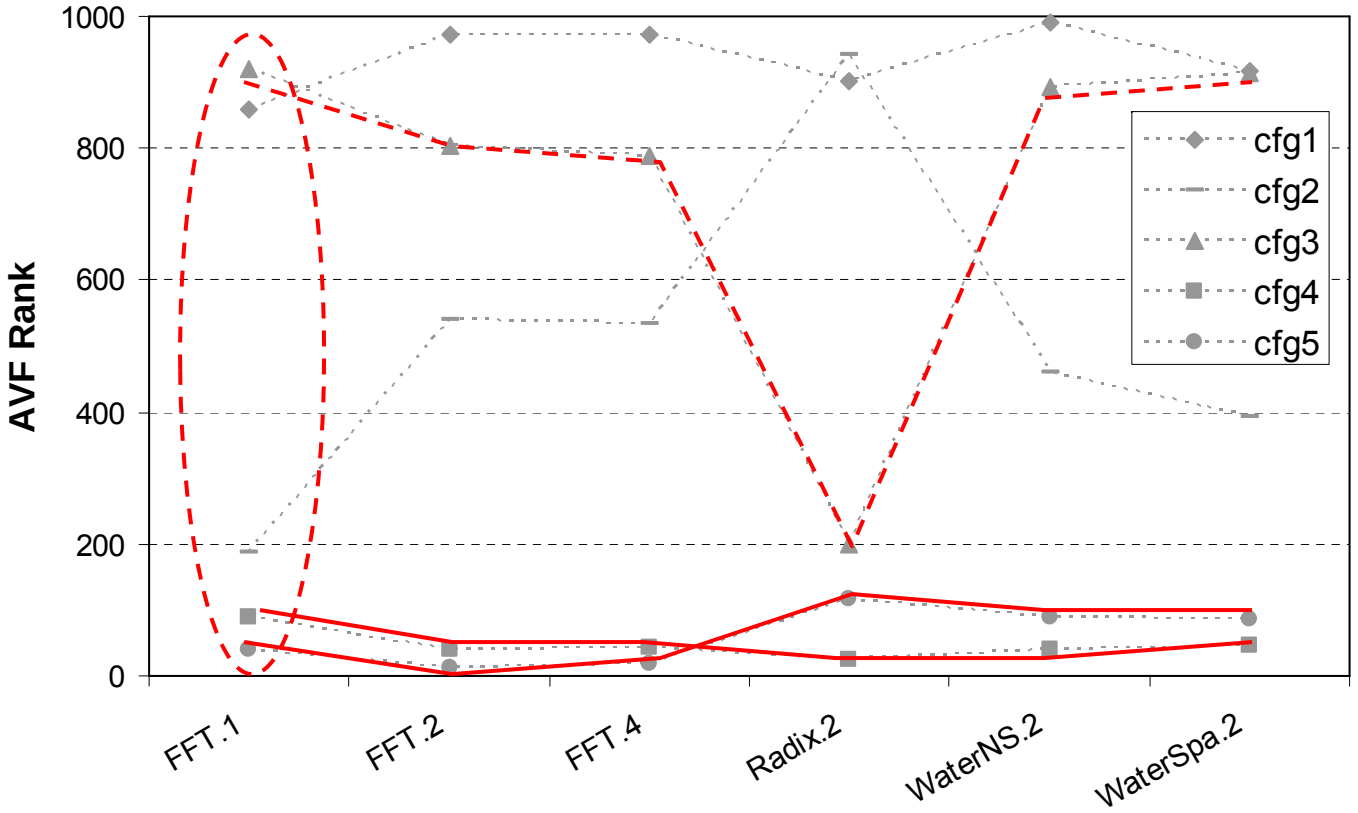
  = Raw SER * AVF

Depends on circuits, area, temperature, etc.

Quantified at architecture level, focused on in this work

Our Work

- Generate selective rules on design parameters to identify the design space subregion showing optimal soft error reliability (lowest AVF)
  - Are these rules effective across programs?
  - Can we do it efficiently?

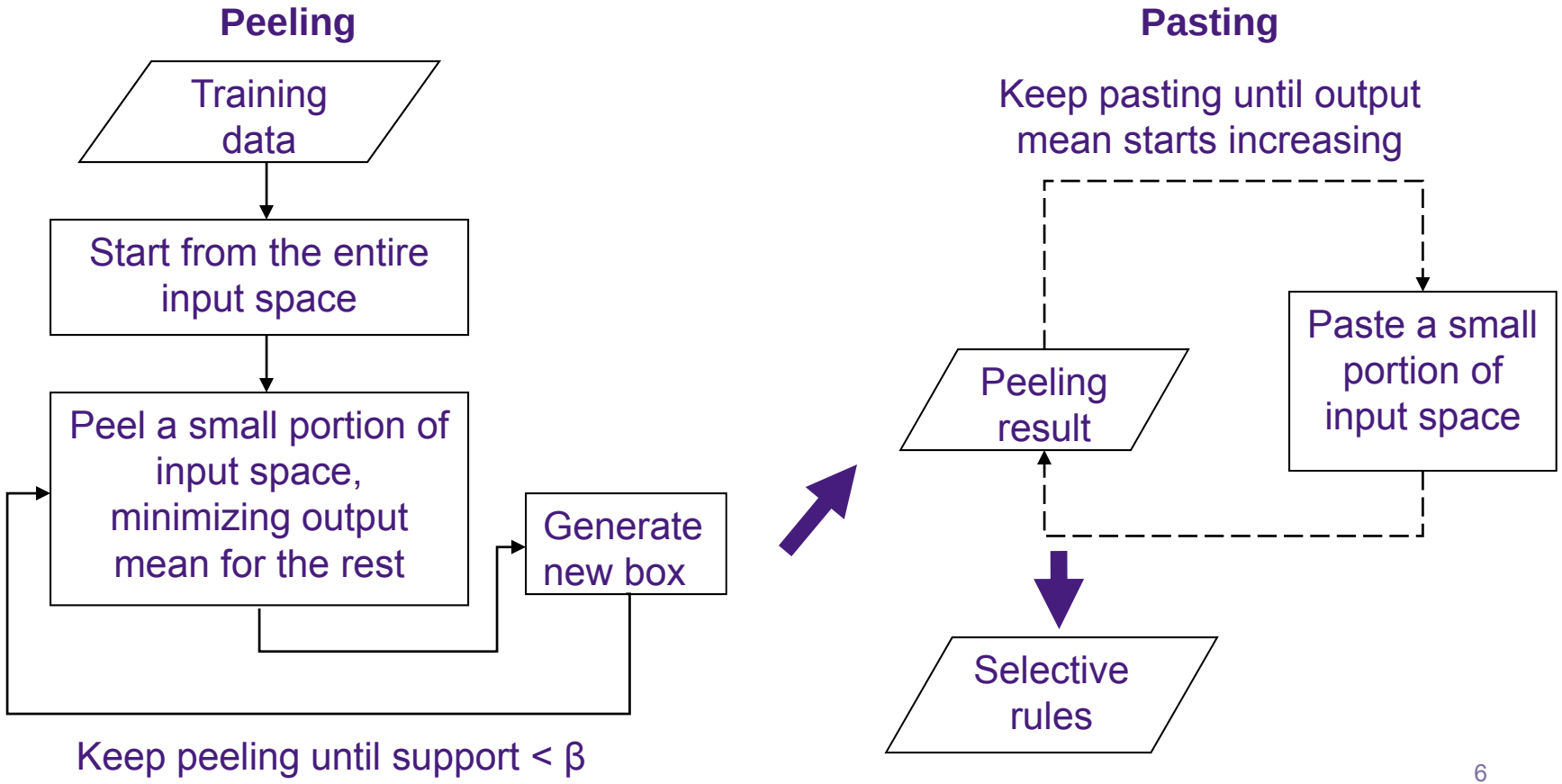


Contributions

- Design parameter selection for soft error resilient processors
  - Use a rule search strategy to identify the design space subregion showing lowest AVF
- Universal rules generation and validation
  - Propose a mechanism to generate universal rules with cross-program effectiveness
  - Validate the generated rules on unseen programs
- Balancing reliability, performance and power for multiprocessors
  - Perform a multi-objective optimization
  - Quantify proper tradeoffs for these metrics

Patient Rule Induction Method

- PRIM: generate selective rules on input variables, quantifying the subregion with lowest output values. I.e. “Valley Seeking”



Experiments on Uniprocessors

- Simulator: SimpleScalar3.0 + ACE analysis
  - Measure AVF for ROB, LSQ, Functional Units, and Register File
- Benchmarks: SPEC2000+SPEC2006
  - Use SimPoint to derive a representative 100-million instruction phase

Train (24)	Test (12)
applu, apsi, art, bzip2, crafty, equake, fma3d, gcc, mcf, mesa, perlbmk, twolf, vortex, astar, 06bzip2, gobmk, hmmer, libquantum, lbm, 06mcf, milc, namd, sjeng, sphinx3	*ammp, eon, facerec, gal-gel, gap, gzip, lucas, mgrid, parser, swim, vpr, wupwise*

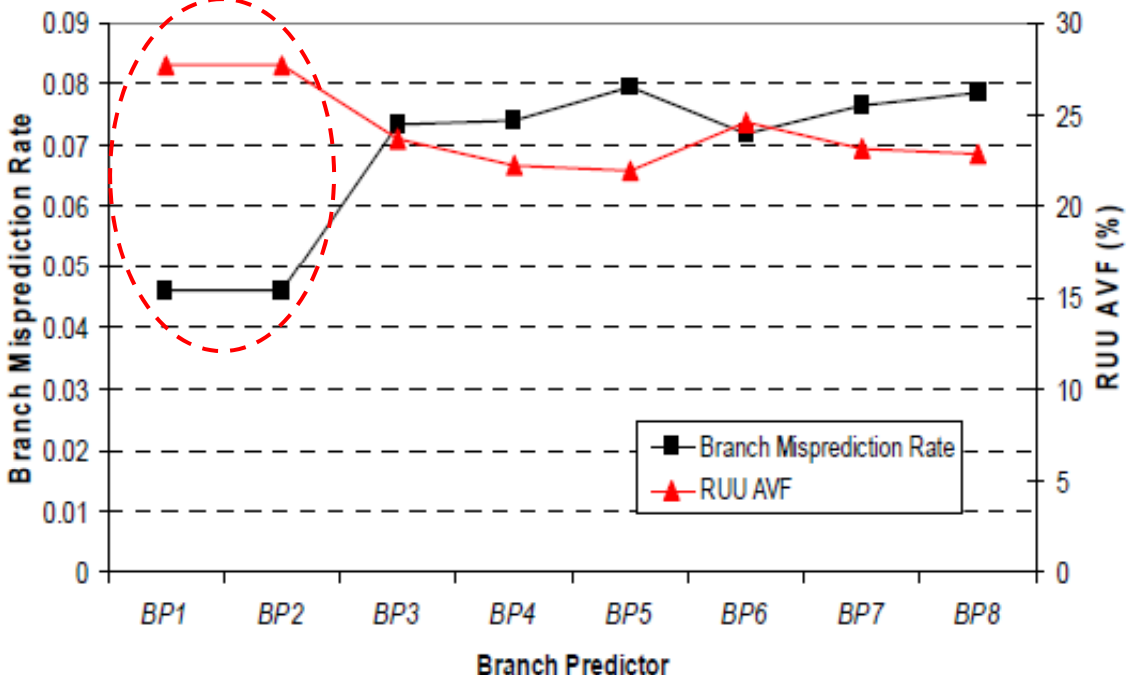
Uniprocessor Design Space

- Space size: 473,088
- Randomly sample 2,000 points for simulation

	Parameter	Selected Values	# Options
P_1	Processor width	2, 4, 8	6
	Fetch queue size	2, 4, 8 (vary with processor width)	
	# of Integer ALUs / # of FP ALUs	1/1, 2/1-associated with processor width 2 2/1, 2/2-associated with processor width 4 2/2, 4/4-associated with processor width 8	
P_2	ROB size	64, 70, 80, 90, 100, 110, 120, 130, 140, 150, 160	11
P_3	LSQ size	16, 24, 32, 40, 48, 56, 64	7
P_4	L1 I/D cache size (L1CS)	16, 32, 64, 128 KB (32B block, 2-way)	4
	L1 cache latency	1, 2, 3, 4 cycles (vary with L1 cache size)	
P_5	L2 cache size (L2CS)	512, 1024, 2048, 4096 KB (64B block)	4
	L2 cache latency	8, 12, 16, 20 cycles (vary with L2 cache size)	
P_6	L2 cache associativity (L2CA)	4, 8	2
P_7	Branch predictor (BP)	bimod/4096 (*BP1*), bimod/8192 (*BP2*), 2lev/1/4096 (*BP3*), 2lev/2/4096 (*BP4*), 2lev/4/4096 (*BP5*), 2lev/1/8192 (*BP6*), 2lev/2/8192 (*BP7*), 2lev/4/8192 (*BP8*)	8
P_8	BTB	1024/4, 2048/2, 1024/8, 2048/4	4

Program-Specific Design Parameter Selection

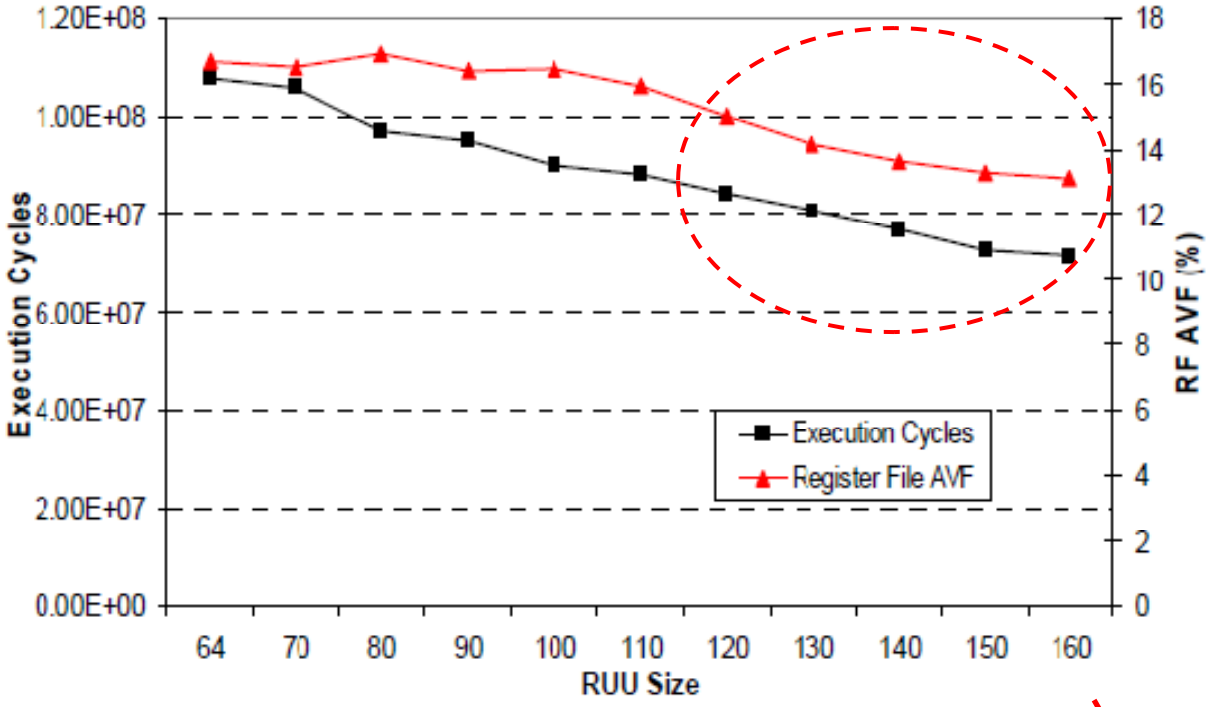
Minimizing AVF for different structures have different imp



Benchmark	ROB AVF			
mcf	(ROB > 130) & (LSQ < 64) & (BP!= *BP1*) & (BP!=*BP2*)	(LSQ > & (BP		
applu	(ROB > 90) & (LSQ < 32)	(ROB		
fma3d	(Width/ALUs=2/2/1) & (ROB > 90)	(Width 4		
gobmk	(ROB>110) & (L1CS<128) & (L2CS > 512) & (BP!=*BP2*) & (BP!=*BP1*)	(LSQ > (BP!=*BP1*) & (BP!=*BP2*)	& (L1CS < 64)	& (Width/ALUs!=2/2/1) & (L1CS > 32) & (L2CS > 512) & (BP!=*BP5*) & (BP!=*BP3*) & (BP!=*BP4*)
milc	(ROB>90) & (LSQ<32)	(ROB < 110) & (LSQ > 40)	Width/ALUs=8/4/4	(Width/ALUs!=2/1/1) & (ROB > 110) & (LSQ > 40)

Program-Specific Design Parameter Selection

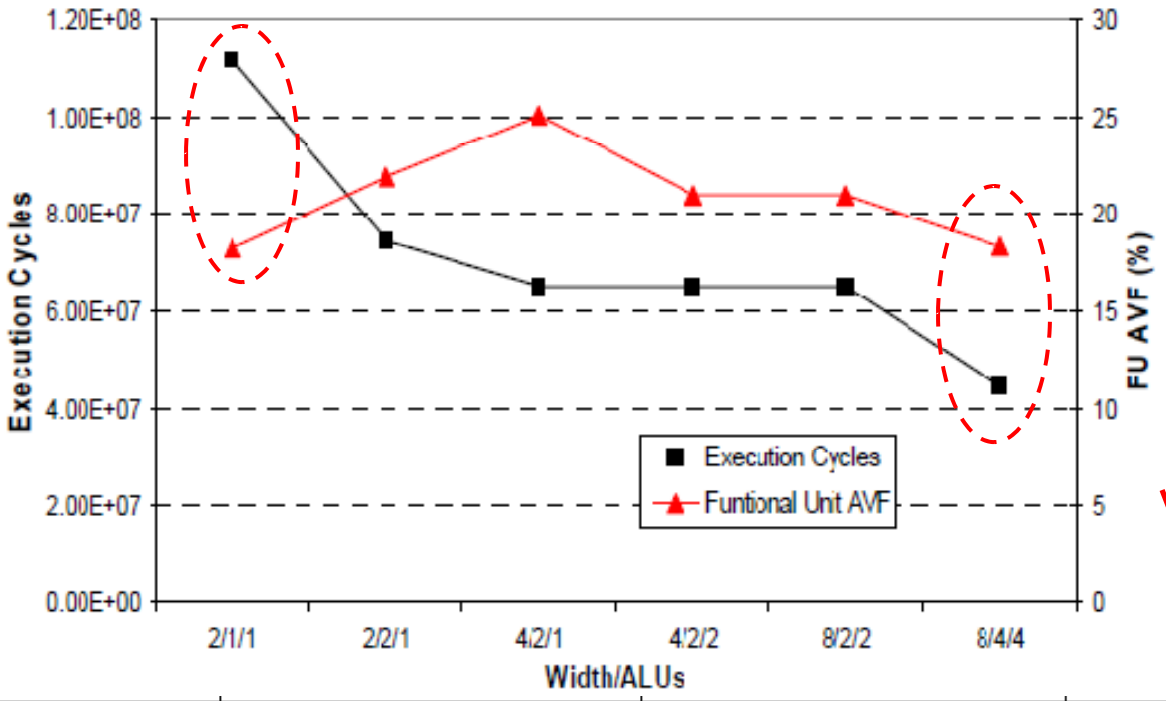
Ter



Benchmark				Register File AVF
mcf				(Width/ALUs!=2/1/1) & (LSQ > 24) & (BP=*BP1*‖*BP2*)
applu				(Width/ALUs=4/2/2‖8/2/2‖8/4/4) & (ROB > 80) & (LSQ > 24) & (L1CS > 16)
fma3d				Width/ALUs=8/4/4
gobmk	(ROB>110) & (L1CS<128) & (L2CS > 512) & (BP!=*BP2*) & (BP!=*BP1*)	(LSQ > 40) & (L1CS < 64) & (BP!=*BP1*) & (BP!=*BP2*)	(Width/ALUs=2/2/1‖8/4/4) & (L1CS < 64)	(Width/ALUs!=2/1/1) & (Width/ALUs!=2/2/1) & (L1CS > 32) & (L2CS > 512) & (BP!=*BP5*) & (BP!=*BP3*) & (BP!=*BP4*)
milc	(ROB>90) & (LSQ<32)	(ROB < 110) & (LSQ > 40)	Width/ALUs=8/4/4	(Width/ALUs!=2/1/1) & (ROB > 110) & (LSQ > 40)

:er Selection

o improve performance

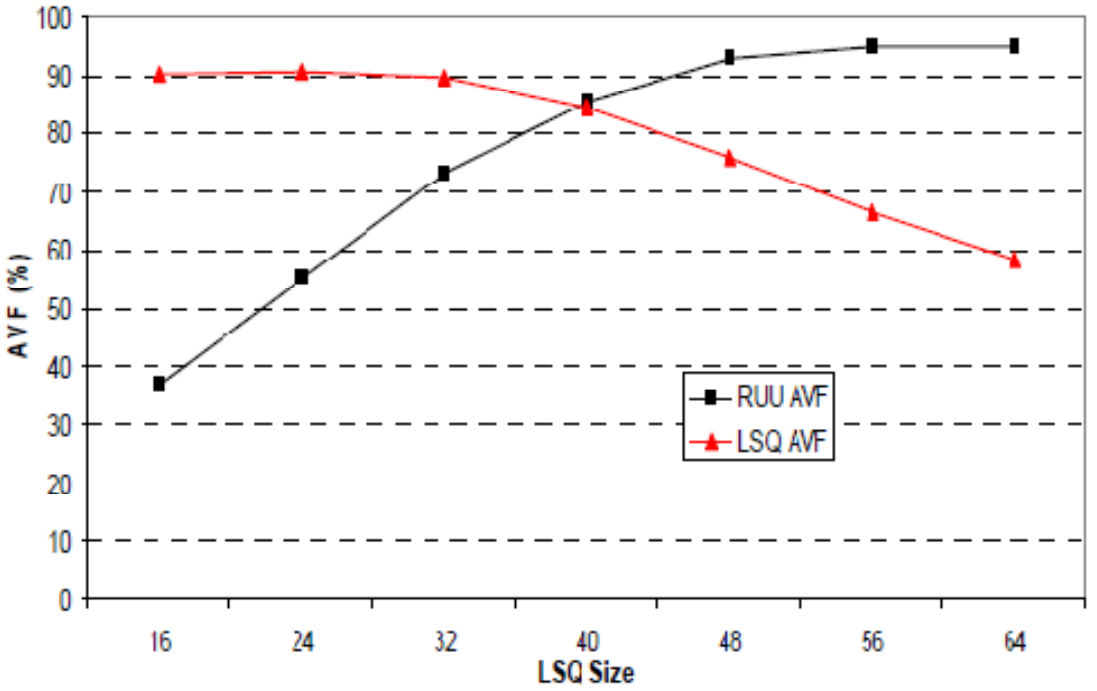


			tional Unit AVF	Register File AVF
			idth/ALUs=8/4/4	(Width/ALUs!=2/1/1) & (LSQ > 24) & (BP=*BP1*||*BP2*)
			/ALUs=4/2/2||8/2/2|| /4) & (LSQ < 32)	(Width/ALUs=4/2/2||8/2/2||8 /4/4) & (ROB > 80) & (LSQ > 24) & (L1CS > 16)
fma3d	(Width/ALUs=2/2/1) & (ROB > 90)	(Width/ALUs=2/2/1||4/2/1||8/ 4/4) & (LSQ > 40) & (L1CS < 128)	(Width/ALUs=2/1/1||8/4/4) & (L1CS < 64)	Width/ALUs=8/4/4
gobmk	(ROB>110) & (L1CS<128) & (L2CS > 512) & (BP!=*BP2*) & (BP!=*BP1*)	(LSQ > 40) & (L1CS < 64) & (BP!=*BP1*) & (BP!=*BP2*)	(Width/ALUs=2/2/1||8/4/4) & (L1CS < 64)	(Width/ALUs!=2/1/1) & (Width/ALUs!=2/2/1) & (L1CS > 32) & (L2CS > 512) & (BP!=*BP5*) & (BP!=*BP3*) & (BP!=*BP4*)
milc	(ROB>90) & (LSQ<32)	(ROB < 110) & (LSQ > 40)	Width/ALUs=8/4/4	(Width/ALUs!=2/1/1) & (ROB > 110) & (LSQ > 40)

Program-Specific Des

Reducing the AVF of
AVF of others. Nee

Can either impr



Benchmark	ROB AVF			
mcf	(ROB > 130) & (LSQ < 64) & (BP!= *BP1*) & (BP!=*BP2*)	(LSQ > & (BP!		
applu	(ROB > 90) & (LSQ < 32)	(ROB < 110) & (LSQ > 40)	(Width/ALUs=4/2/2‖8/2/2‖ 8/4/4) & (LSQ < 32)	(Width/ALUs=4/2/2‖8/2/2‖8 /4/4) & (ROB > 80) & (LSQ > 24) & (L1CS > 16)
fma3d	(Width/ALUs=2/2/1) & (ROB > 90)	(Width/ALUs=2/2/1‖4/2/1‖8/ 4/4) & (LSQ > 40) & (L1CS < 128)	(Width/ALUs=2/1/1‖8/4/4) & (L1CS < 64)	Width/ALUs=8/4/4
gobmk	(ROB>110) & (L1CS<128) & (L2CS > 512) & (BP!=*BP2*) & (BP!=*BP1*)	(LSQ > 40) & (L1CS < 64) & (BP!=*BP1*) & (BP!=*BP2*)	(Width/ALUs=2/2/1‖8/4/4) & (L1CS < 64)	(Width/ALUs!=2/1/1) & (Width/ALUs!=2/2/1) & (L1CS > 32) & (L2CS > 512) & (BP!=*BP5*) & (BP!=*BP3*) & (BP!=*BP4*)
milc	(ROB>90) & (LSQ<32)	(ROB < 110) & (LSQ > 40)	Width/ALUs=8/4/4	(Width/ALUs!=2/1/1) & (ROB > 110) & (LSQ > 40)

Universal Rules Generation

- Rank the 2,000 simulated configurations in each benchmark (in terms of AVF)
  - Rank 1 has the lowest AVF
- Train a single universal model, with the output variable being:
  - The average of ranks
  - The maximum of ranks
  - Mean(rank^3)

Rule Set I (Optimizing Uniprocessor AVF):

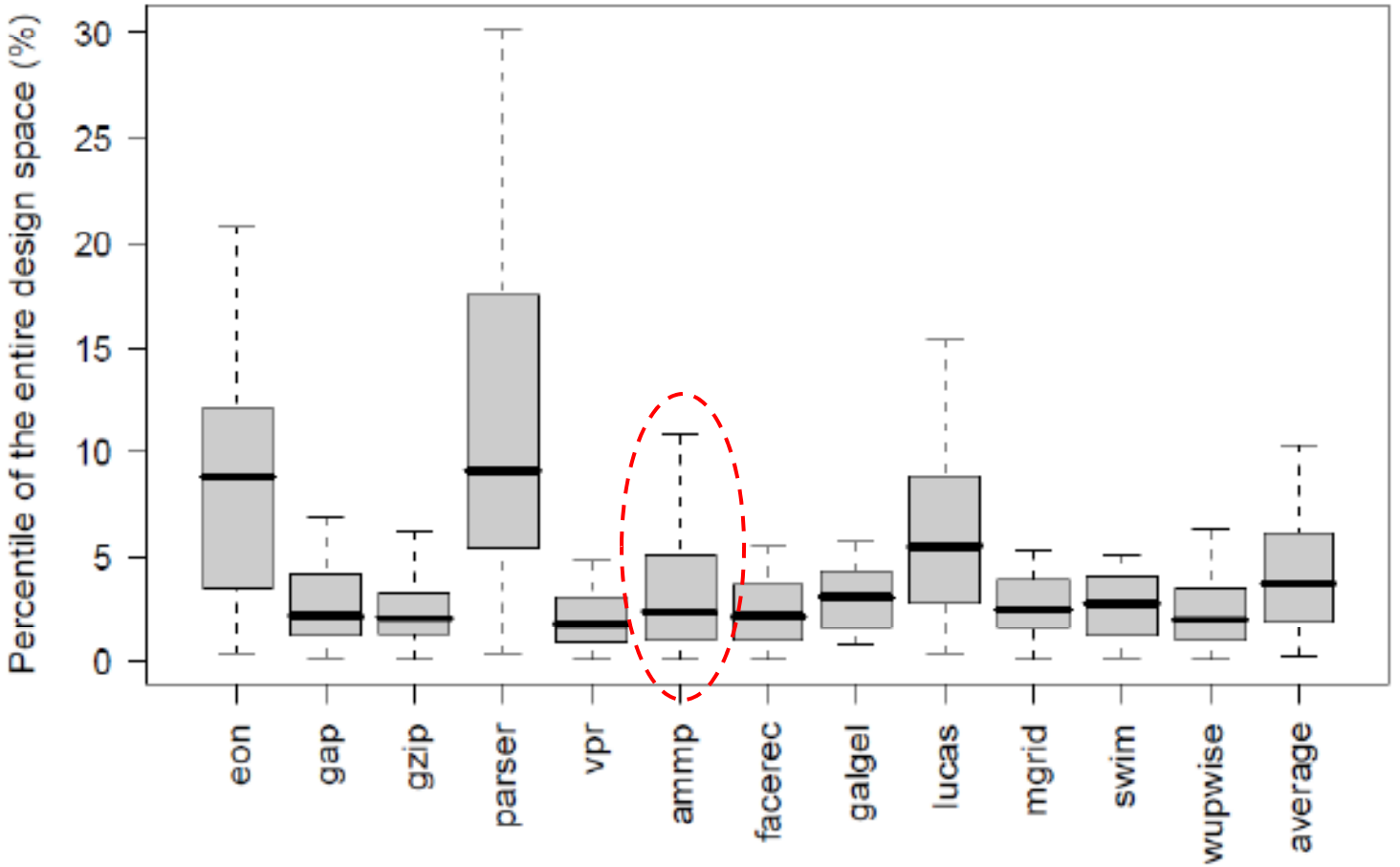
(Width/ALUs!=8/2/2) & (ROB>130) & (LSQ<24) & (L1CS<128kB) & (BP!=*BP1*) & (BP!=*BP2*)

Universal Rules Validation

- Simulate the 2,000 configurations for each of the test benchmarks
- Identify what points (among the simulated 2K ones) are selected by Rule Set I
  - βwas 2%, so ~40 points are selected.
- Identify where the selected points are located in the entire design space (not just the sampled 2K points!)

Use bootstrapping method

Validation of Rule Set I



Experiments on Multiprocessors

- Simulator: M5 + AVF measurement on multiprocessors
- Multithreaded workloads: 6 benchmarks from SPLASH2
- Each workload has three runs
  - 1-threaded, 2-threaded, and 4-threaded runs on single-core, dual-core, and quad-core processors, respectively
  - Each run contains 1K simulations sampled from multiprocessor design space

Multiprocessor Design Space

- More parameters
- Enlarged size: ~1.5 million points

	Parameter	Selected Values	# Options
M_1	Processor width	2, 4, 8	6
	# of Integer ALUs / # of FP ALUs	1/1, 2/1-associated with processor width 2 2/2, 4/3-associated with processor width 4 4/3, 6/4-associated with processor width 8	
M_2	ROB size	72, 84, 96, 108, 120, 132, 144, 156, 168	9
M_3	LQ/SQ sizes	16, 20, 24, 28, 32	5
M_4	IQ size	32, 40, 48, 56, 64, 72	6
M_5	Phys. Int/FP reg. file sizes	100, 120, 140, 160, 180	5
M_6	BTB	1024, 2048, 4096	3
M_7	RAS	8, 12, 16	3
M_8	L1 I/D cache sizes	16, 32, 64, 128 KB (64B block, 2-way assoc.)	4
	L1 cache latency	1, 2, 3, 4 cycles (vary with L1 cache size)	
M_9	(Shared) L2 cache size	512, 1024, 2048, 4096, 8192 KB (64B block, 8-way assoc.)	5
	(Shared) L2 cache latency	10, 12, 14, 16, 18 cycles (vary with L2 cache size)	

Optimizing Three Metrics

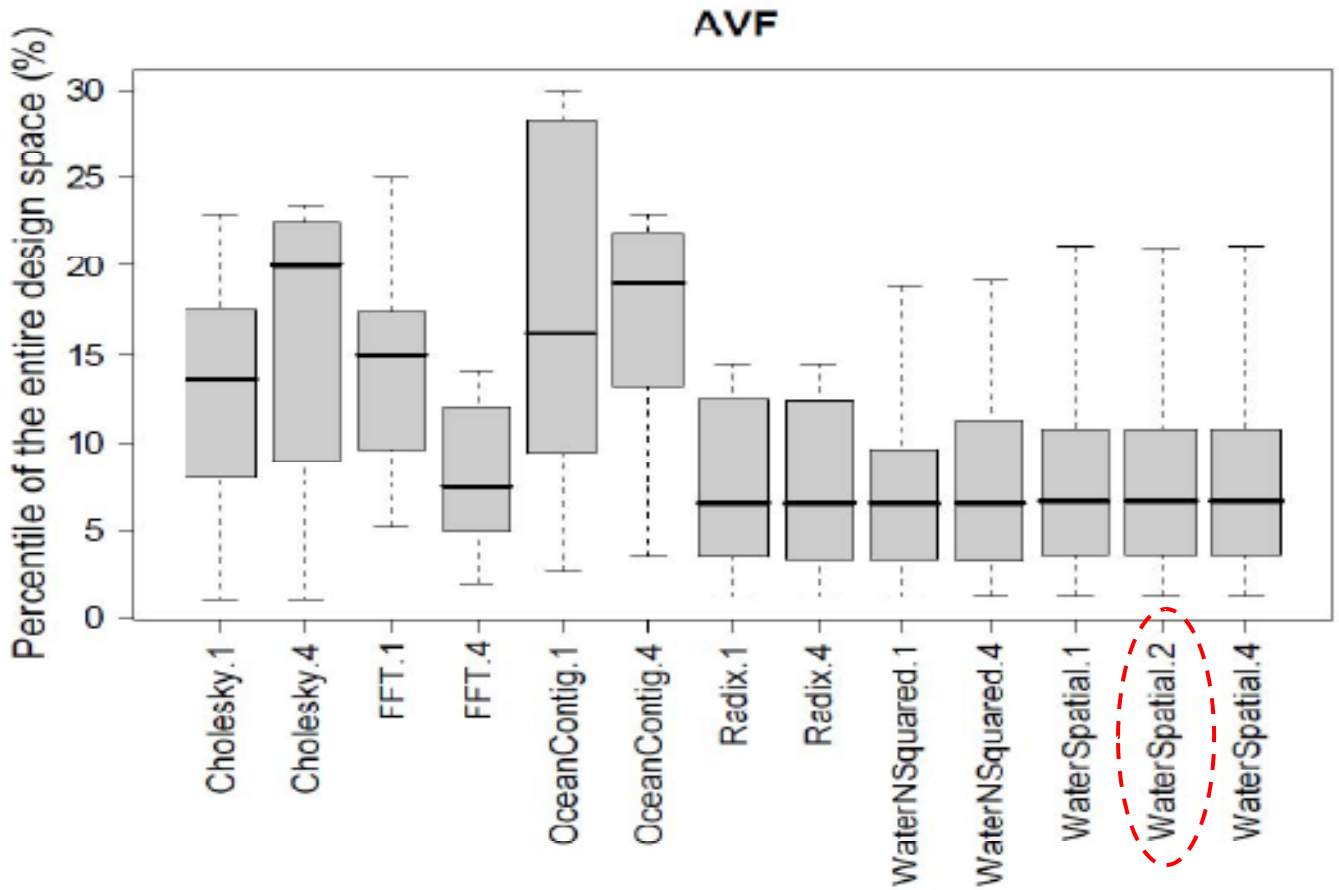
- Performance: 1/Throughput = $(\Sigma IPC_i)^{-1}$
- Reliability: AVF of the system
- Power: power consumption of the system

Individual optimization of the three metrics

Rule Set II (Optimizing AVF)	(Width/ALUs=4/2/2‖4/4/3‖8/4/3‖8/6/4) & (ROB>132) & (LSQ<32) & (IQ<48) & (L1CS>32kB) & (L2CS<2MB)
Rule Set III (Optimizing Throughput $^{-1}$)	(Width/ALUs=8/4/3‖8/6/4) & (IQ>64)
Rule Set IV (Optimizing Power)	(Width/ALUs=8/6/4) & (ROB<156) & (16<LSQ<32) & (IQ<72) & (Phy. Reg. File<140) & (16kB<L1CS<128kB) & (L2CS !=1MB)

Validation Example

2-threaded runs of 5 benchmarks used in training



Balancing the Three Metrics

- Simultaneously balancing reliability, performance, and power is a multi-objective optimization problem
- Requires a reasonable objective function to achieve good tradeoffs among conflicting metrics

$$f = AVF^{a} * (1/Throughput)^{b} * Power^{c}$$

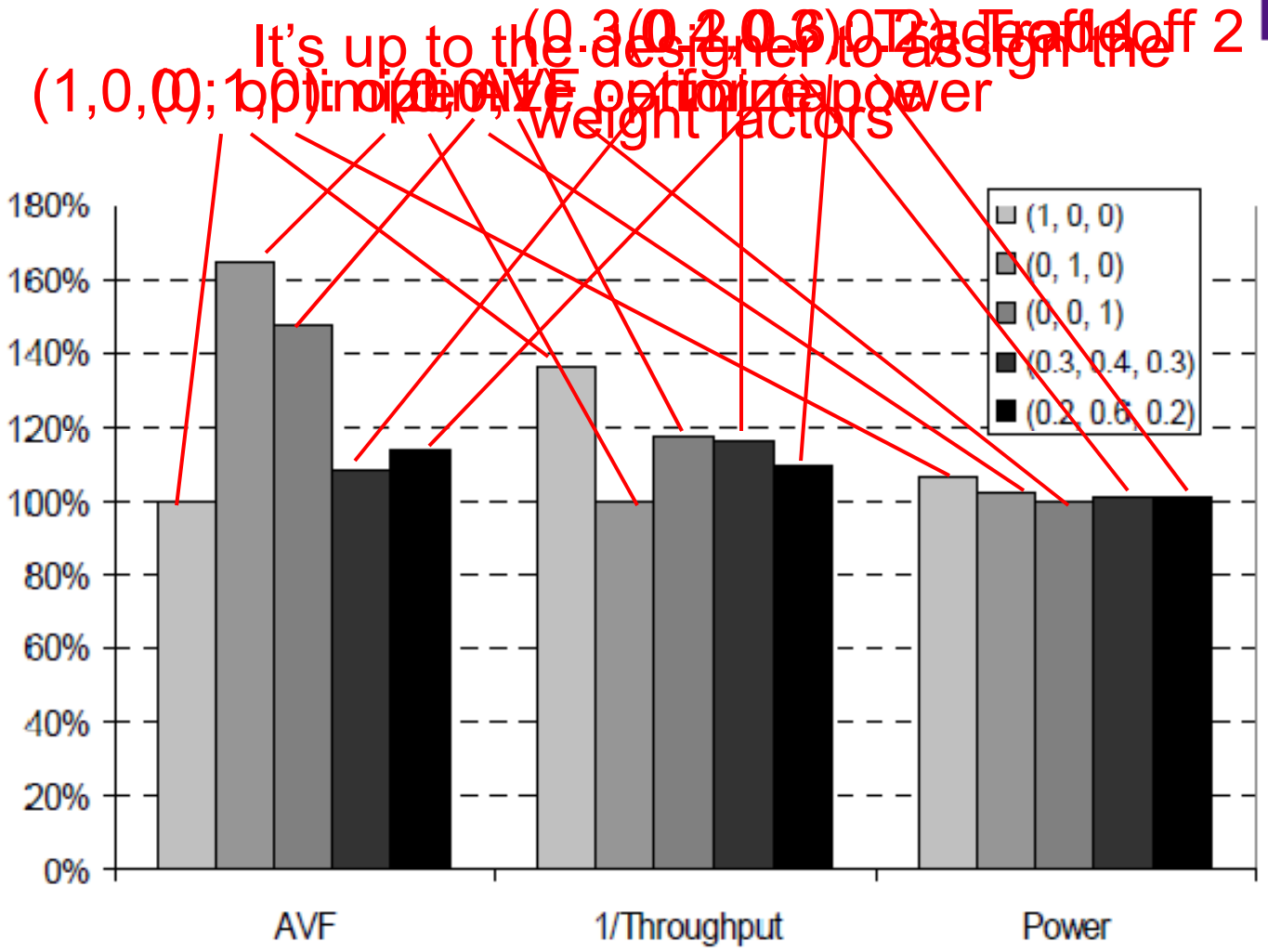
where a, b, c >= 0, and a+b+c = 1

➢ Adjust weight factors to prioritize different metrics

➢ Rule Set II, III, IV (individual optimizations) are special cases of *f*

➢ None of these rule sets achieves a good tradeoff

Tradeoffs


It's up to the designer to assign the weight factors
(1, 0, 0)
(0, 1, 0)
(0, 0, 1)
(0.3, 0.4, 0.3)
(0.2, 0.6, 0.2)
180%
160%
140%
120%
100%
80%
60%
40%
20%
0%
AVF
1/Throughput
Power

Summary

- Propose to use PRIM to generate simple rules on design parameters
  - Identify the optimal design choices
  - Provide useful guidelines at pre-silicon stage
- Generate universal rules effective across programs
  - Only a single model trained
  - Validated on unseen programs
- Balance conflicting metrics on multiprocessors
  - Propose an effective objective function
  - Achieve good tradeoffs via assigning weight factors

Thanks!

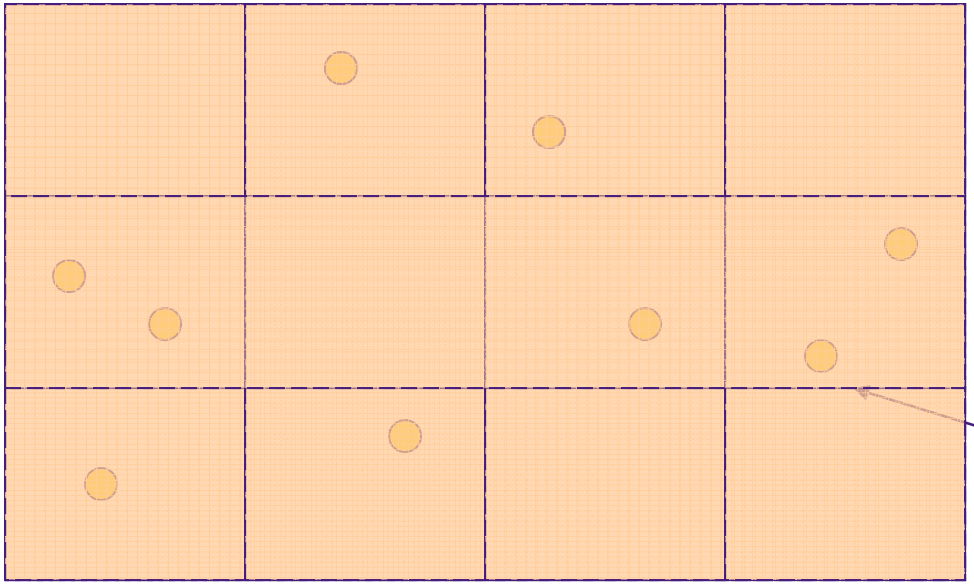
Questions?

Traditional Design Space Studies

- Program-specific: need to train a separate model for each program.

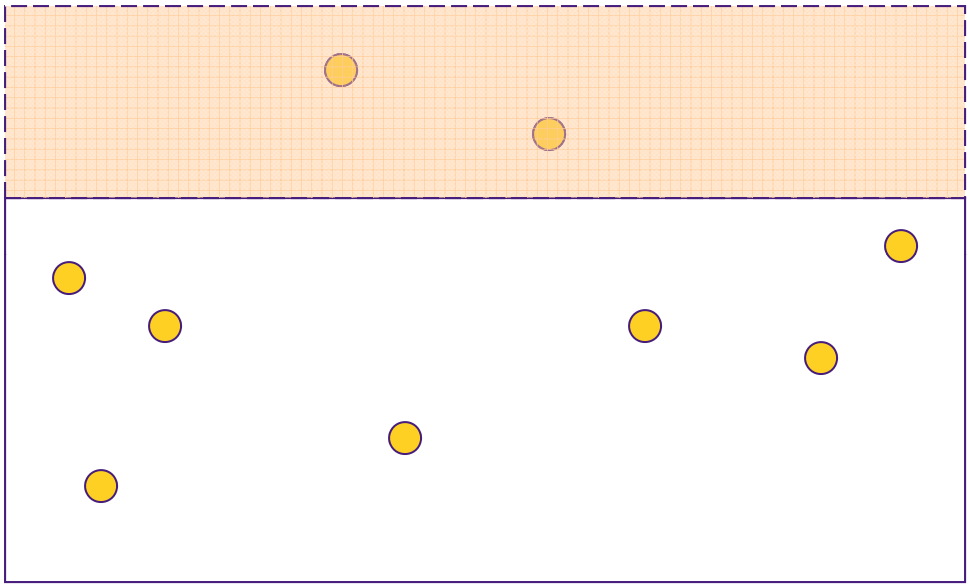
- Expensive!
- Exacerbated in multithreading
  - # of multiprogrammed workloads significantly increases
  - 100 (single programs) → 4,950 (2-program combinations) → ~4 million (4-program combinations) → …

PRIM Example



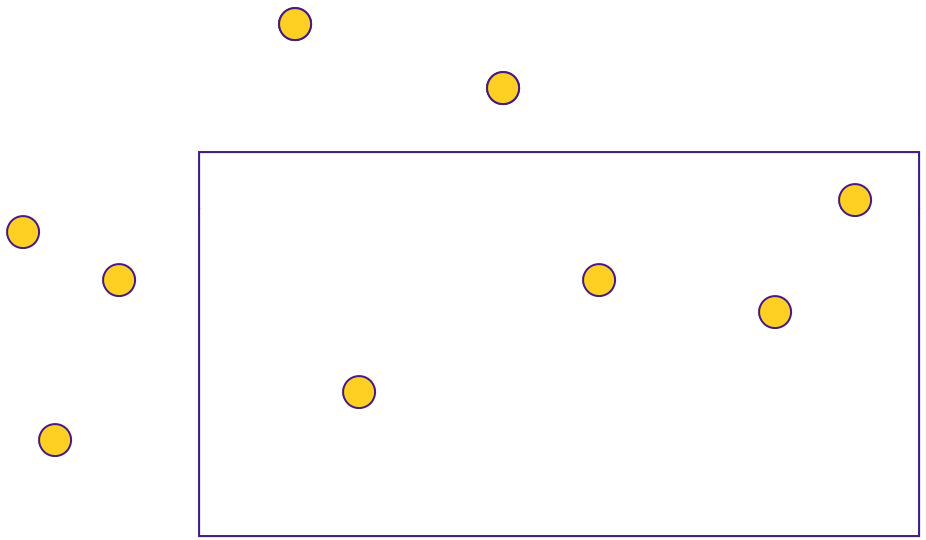
Try to peel a small portion, calculate the output mean of the rest.

PRIM Example



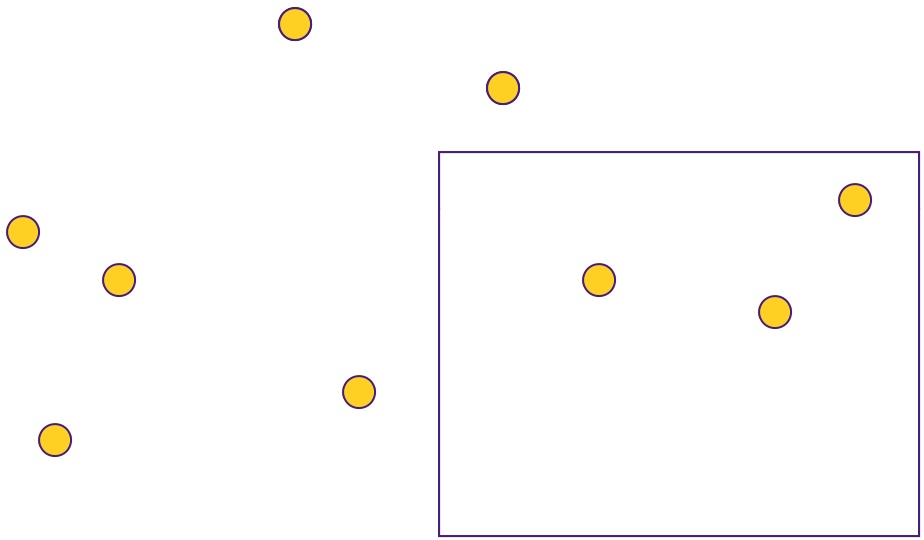
Peel off the portion that generates the lowest output mean for the rest

PRIM Example



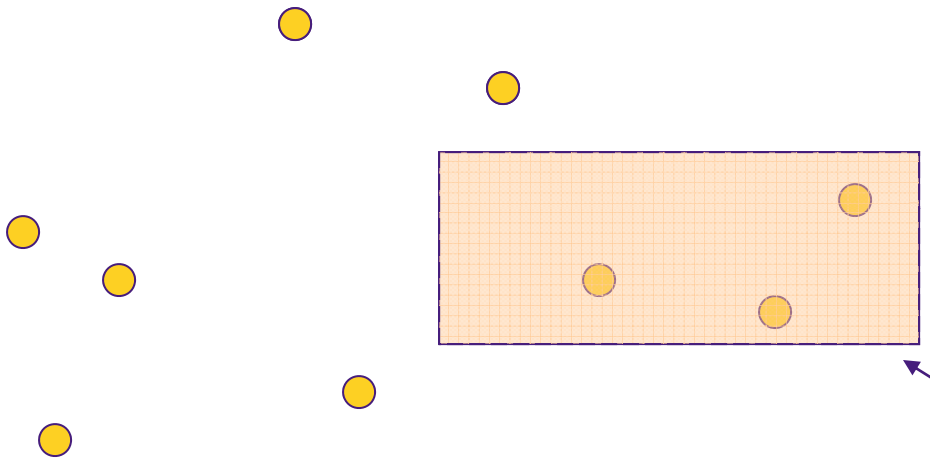
Repeat the above iteration,
and keep peeling

PRIM Example



Repeat the above iteration,
and keep peeling

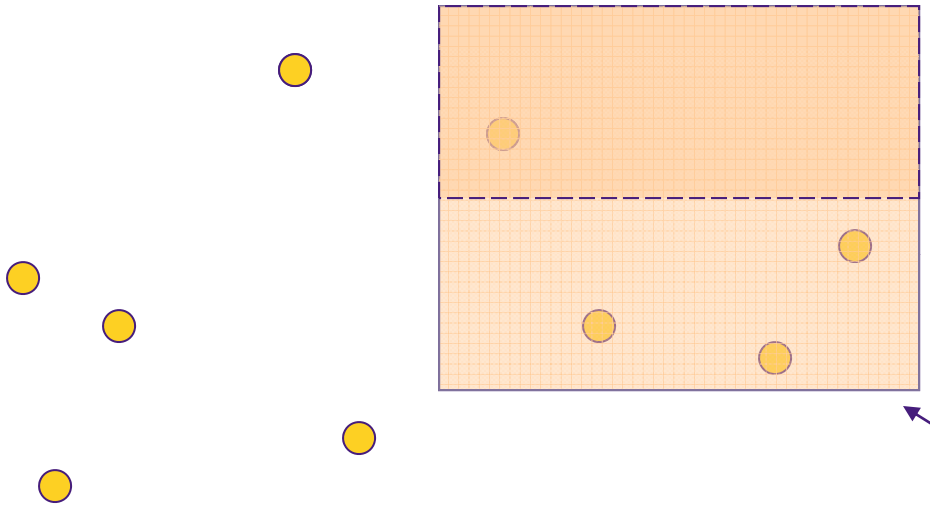
PRIM Example



The identified subregion, i.e. the “valley” of the input space

Peeling stops when the support is below a threshold β

PRIM Example



The identified subregion, i.e. the “valley” of the input space

Pasting is the reverse of peeling

Bootstrapping

- In order to estimate the *p*-percentile of the entire design space:
  - Sample (with replacement) 1,000 times for the 2K points
  - Calculate the *p*-percentile for the 1K bootstrap samples
  - Calculate the 5-percentile (say W) for the above 1K values. Then we have 95% confidence that the *p*-percentile of entire design space >= W.
- Adjust *p* to make W >= the selected point's value. Then, the selected point is within the top *p*% optima of the entire design space.