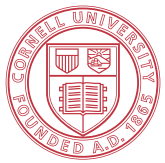
A Parallel Bandit-Based Approach for Autotuning FPGA Compilation

Chang Xu[1], Gai Liu[2], Ritchie Zhao[2], Stephen Yang[3], Guojie Luo[1], Zhiru Zhang[2]

1. CECA, Peking University
2. School of ECE, Cornell University
3. Xilinx Inc.



Cornell University

CSL

Explore FPGA CAD Tools Parameters

- FPGA CAD tools provide a large number of tunable options
  - These tunable options affect area/timing/power etc.
  - Challenges:
    - Large search space
    - Interactive parameters

Subset of Vivado configuration parameters

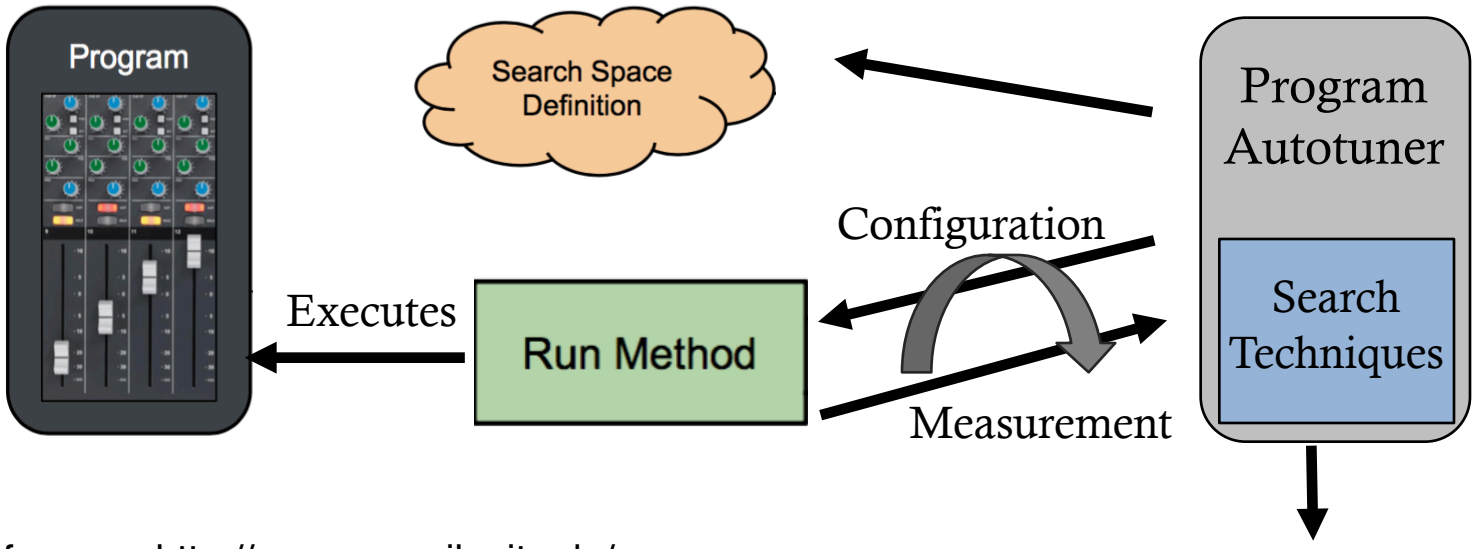
	Parameter	Value
Logic Synth	Flatten_hierarchy	{rebuilt, full, none}
	Keep_equivalent_registers	{on, off}
	Resource_sharing	{auto, on, off}
	Opt_design	{explore, exploreArea, addremap, default}
P&R	Place_design	{explore, extraNetDelay_high, medium, low, extraPostPlacementOpt, default}
	Phys_opt_design	{critical_cell_opt, critical_pin_opt, retime, rewire, fanout_opt, placement_opt }
	Route_design	{explore, default}

Software Program Autotuning

- Challenge to tune compiler parameters
  - E.g., gcc compiler contains over 10^{806} combinations of options

```
g++ example.cpp -o example -O3 -fno-align-loops –fno-branch-target-load-optimize –fno-combine-stack-adjustments –fdelete-null-pointer-checks –fearly-inlining –fno-inlint-functions –fno-loop-parallelization-all –fno-optimize-sibling-calls –fno-thread-jumps –param=l1-cache-line-size=128 –param=loop-block-tile-size=40 --param=max-pending-list-length=10 --param=max-hoist-depth=101…
```

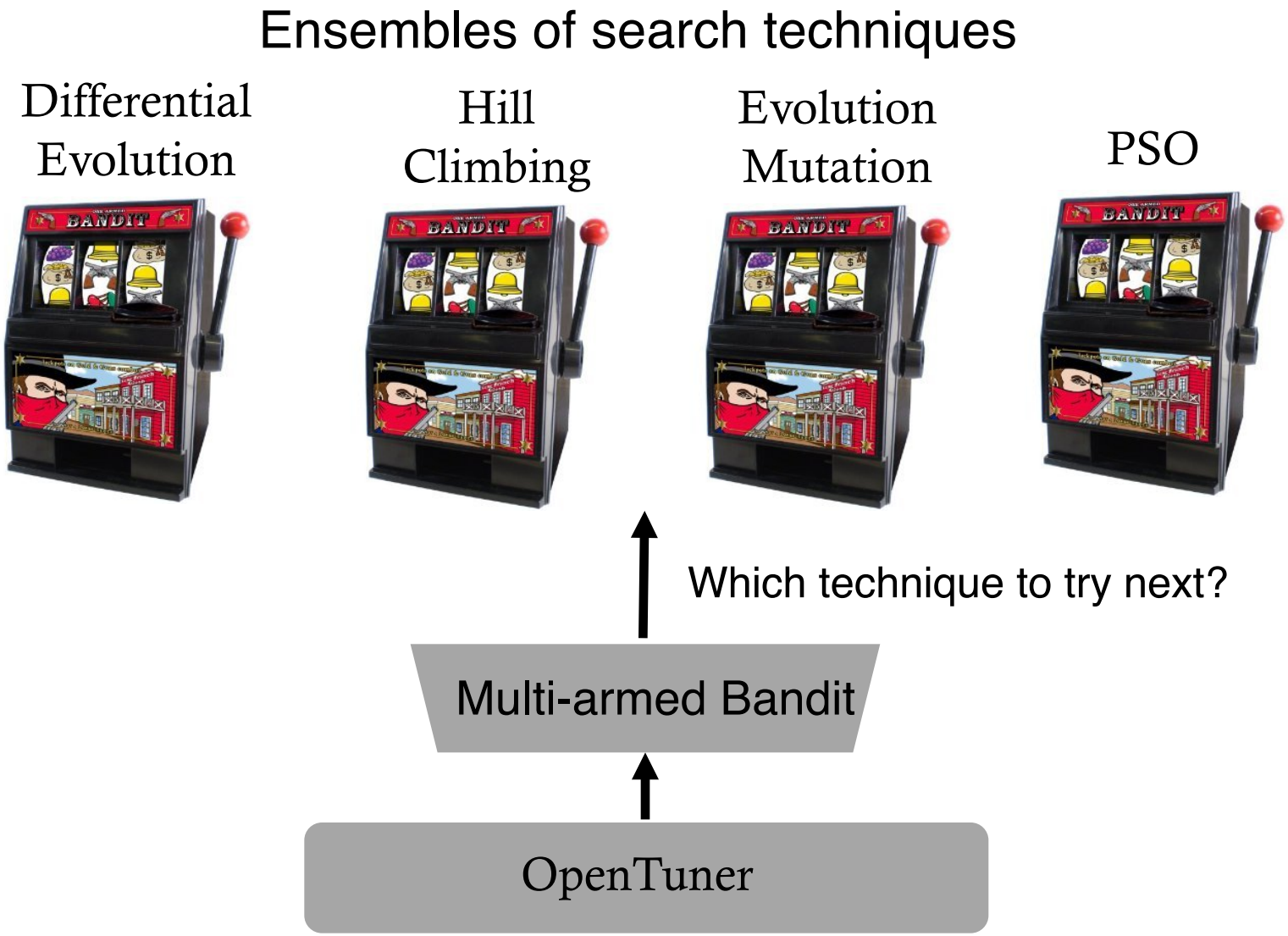
- Autotuning: automatic performance tuning



Reference: http://groups.csail.mit.edu/

Existing Work: OpenTuner [1]

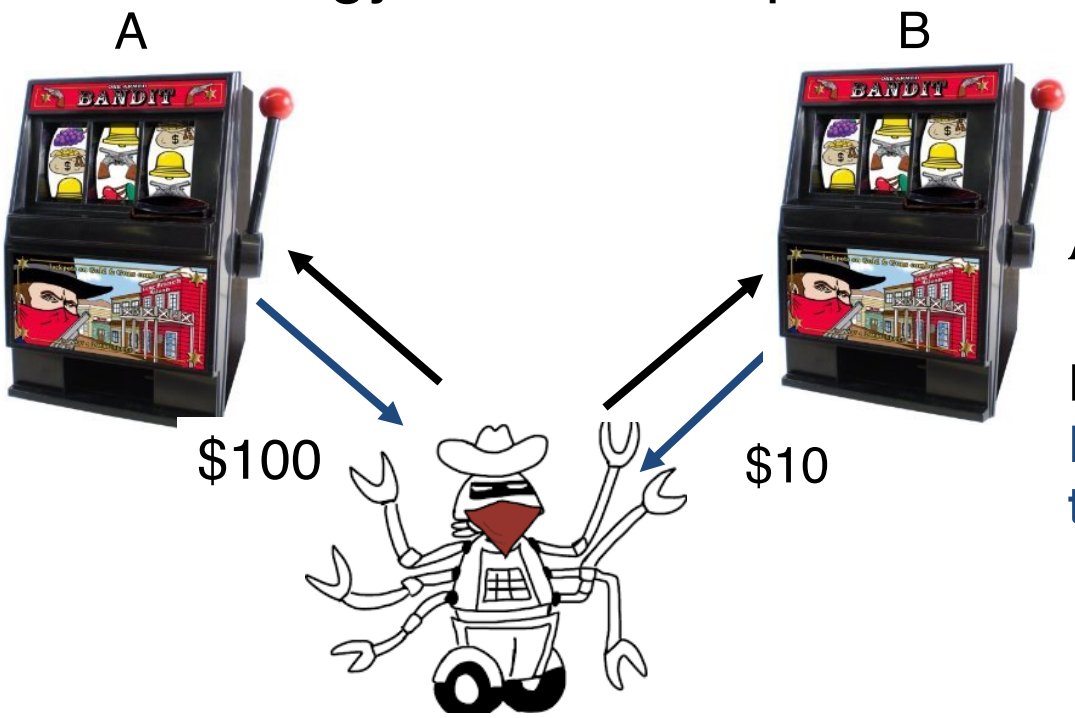
- Extensible multi-objective autotuning framework



[1] Ansel, Jason, et al. "Opentuner: An extensible framework for program autotuning." *PACT,* 2014.

Multi-Armed Bandit (MAB) Problem

- Problem:
  - **Pull an arm:** choose one slot machine to play
  - **Payoff:** sampled from unknown distribution
- Objective: maximize total payoff
- Strategy: balance exploitation and exploration



$$\text{Arg}\max_{t} (\text{Exploitation} + C * \text{Exploration})$$

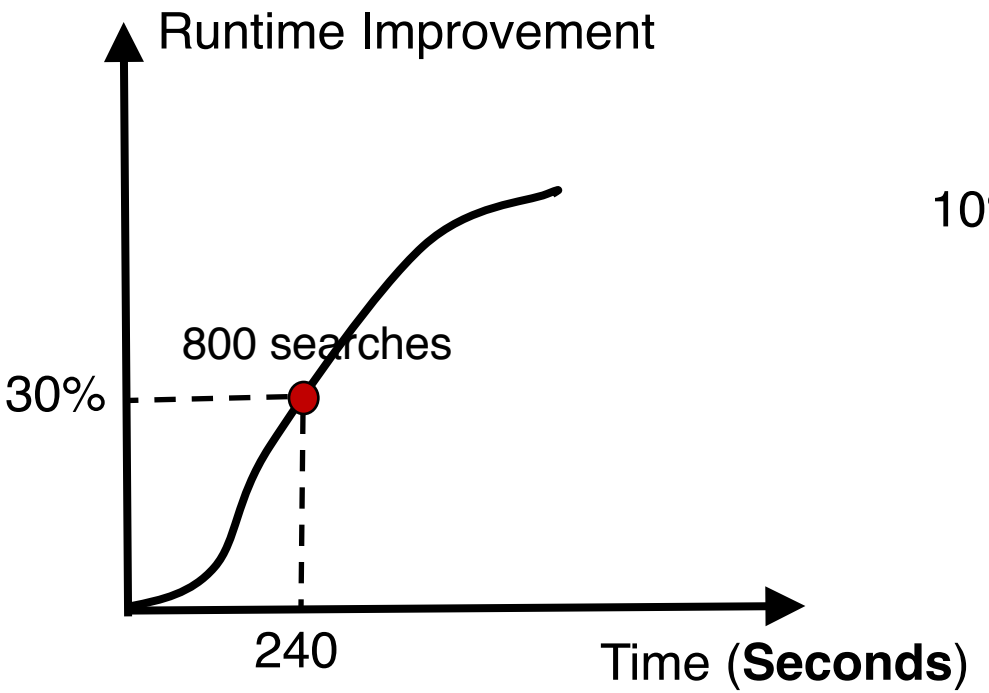
Exploitation: **Avg. payoff**
Exploration: inversely related to **the number of times** arm t used (H_t)

Payoff(A) : $100 $200 $300 $400
H_t(A) = 4

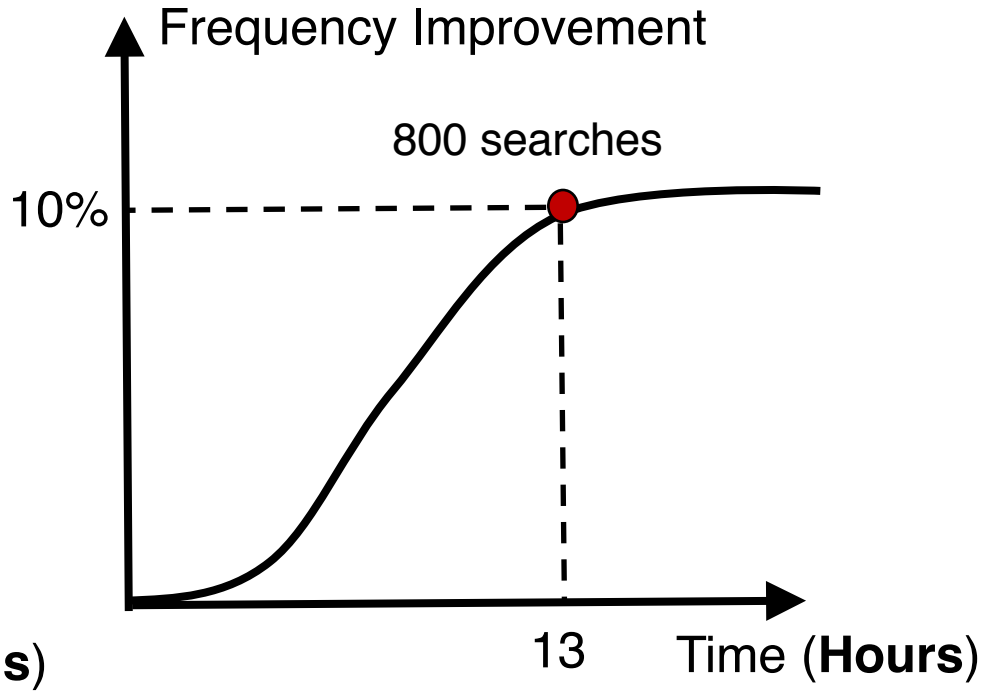
Payoff(B) : $10 $20
H_t(B) = 2

Challenges to Autotuning FPGA CAD Tools

- Large search space
- FPGA compilation is very time-consuming



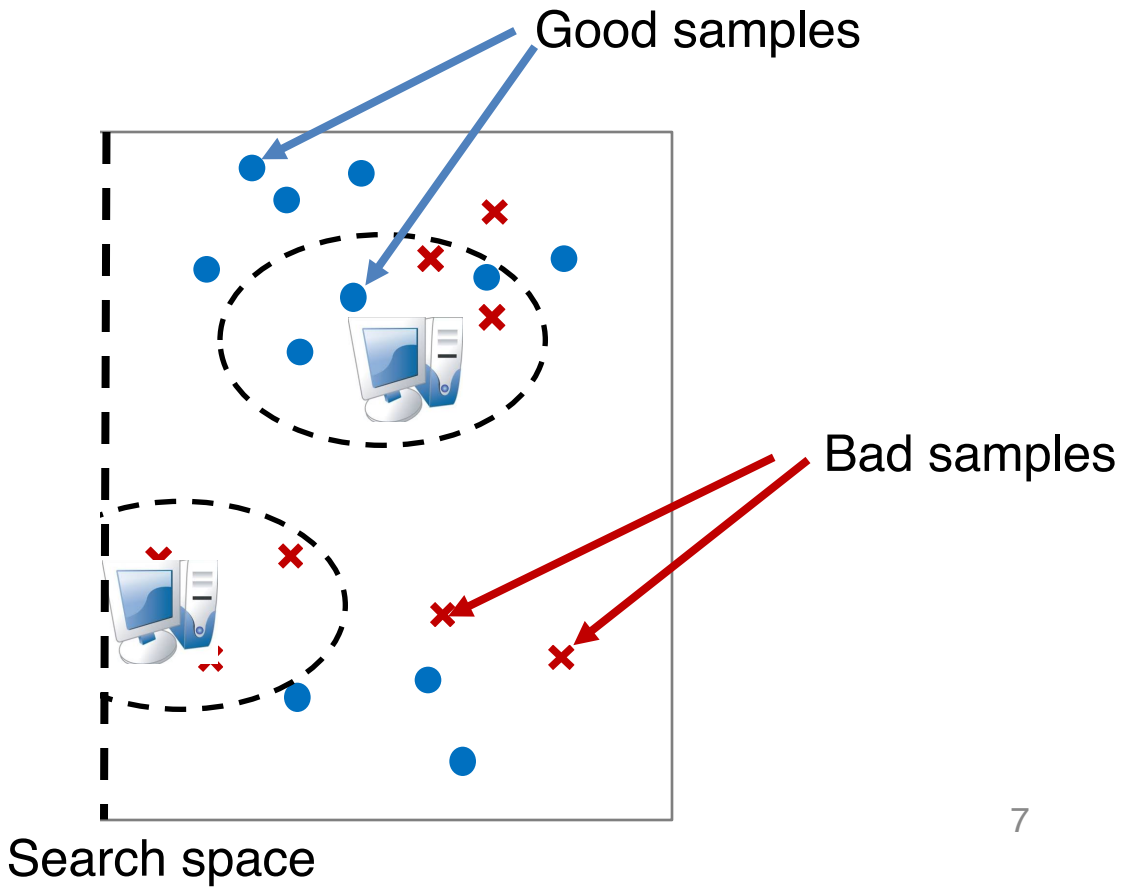
Tuning **GCC compiler** by OpenTuner for Matrixmultiply.c with Xeon@1.86 GHz



Tuning **VTR flow** by OpenTuner for Raygentop.v with Xeon@2.8GHz

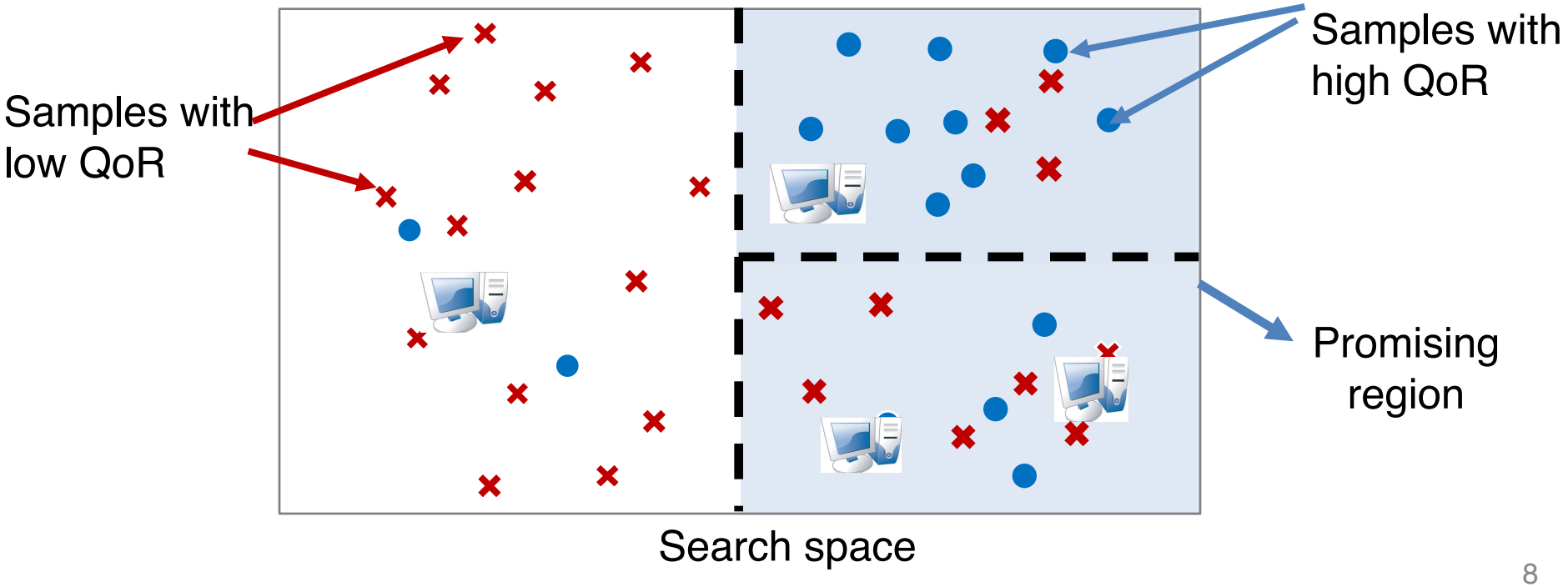
Trivial Parallelization Methods

- Run multiple autotuners in global space
  - With random starting points
  - No search space pruning
- Run multiple autotuners in subspace
  - Static space partitioning: relies heavily on experience



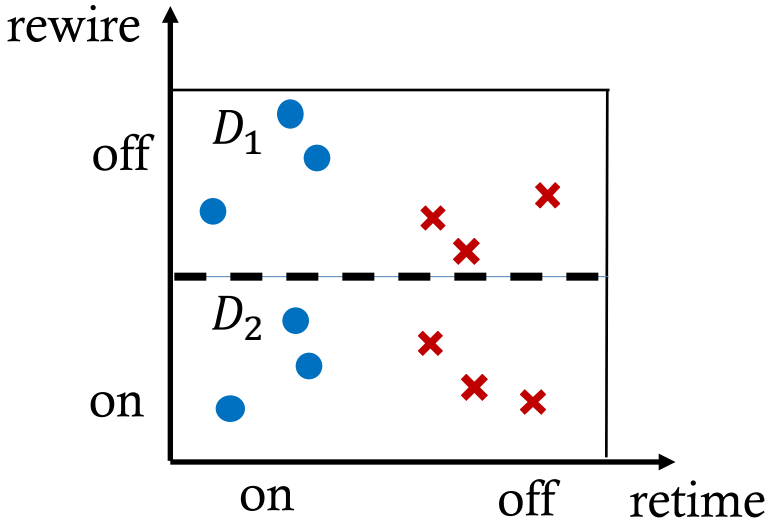
Our Proposal

- DATuner: 2-level bandit-based parallel autotuner
  - Improve quality of result (QoR): frequency, area, routability etc.
  - **Dynamic solution space partitioning:** narrow down promising regions
  - **Intelligent resource allocation** ⇒ 1st level bandit
    - Balance exploitation and exploration
  - Bandit-based subspace exploration ⇒ 2nd level bandit

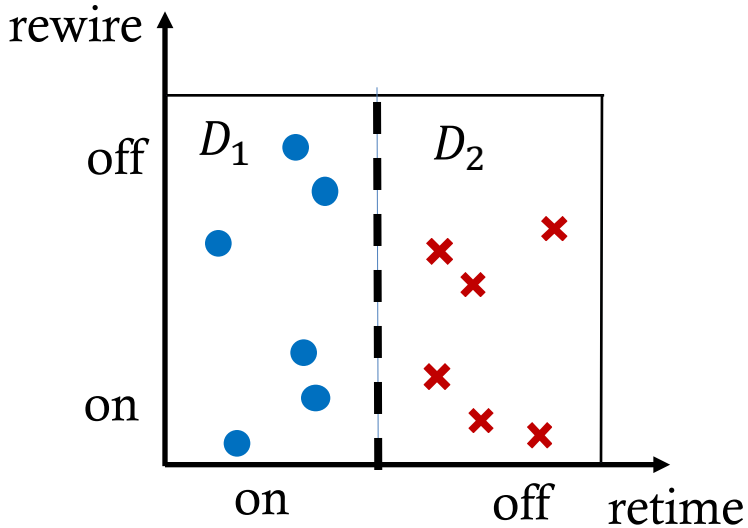


Dynamic Solution Space Partitioning

- Employ an information theoretic approach
  - Good partition = separate promising/unpromising regions
  - Information gain: decrease in *information entropy*
  - *Entropy of a subspace:* $H(D_p) = -(\frac{|C_g|}{|D_p|} log \frac{|C_g|}{|D_p|} + \frac{|C_b|}{|D_p|} log \frac{|C_b|}{|D_p|})$



Partition A: less informative



Partition B: more informative

$$H(D_0) = -\frac{6}{12} \times \log\left(\frac{6}{12}\right) - \frac{6}{12} \times \log\left(\frac{6}{12}\right) = 0.3 \quad (D_0 = D_1 \cup D_2)$$

$H(D_1) = H(D_2) = 0.3$

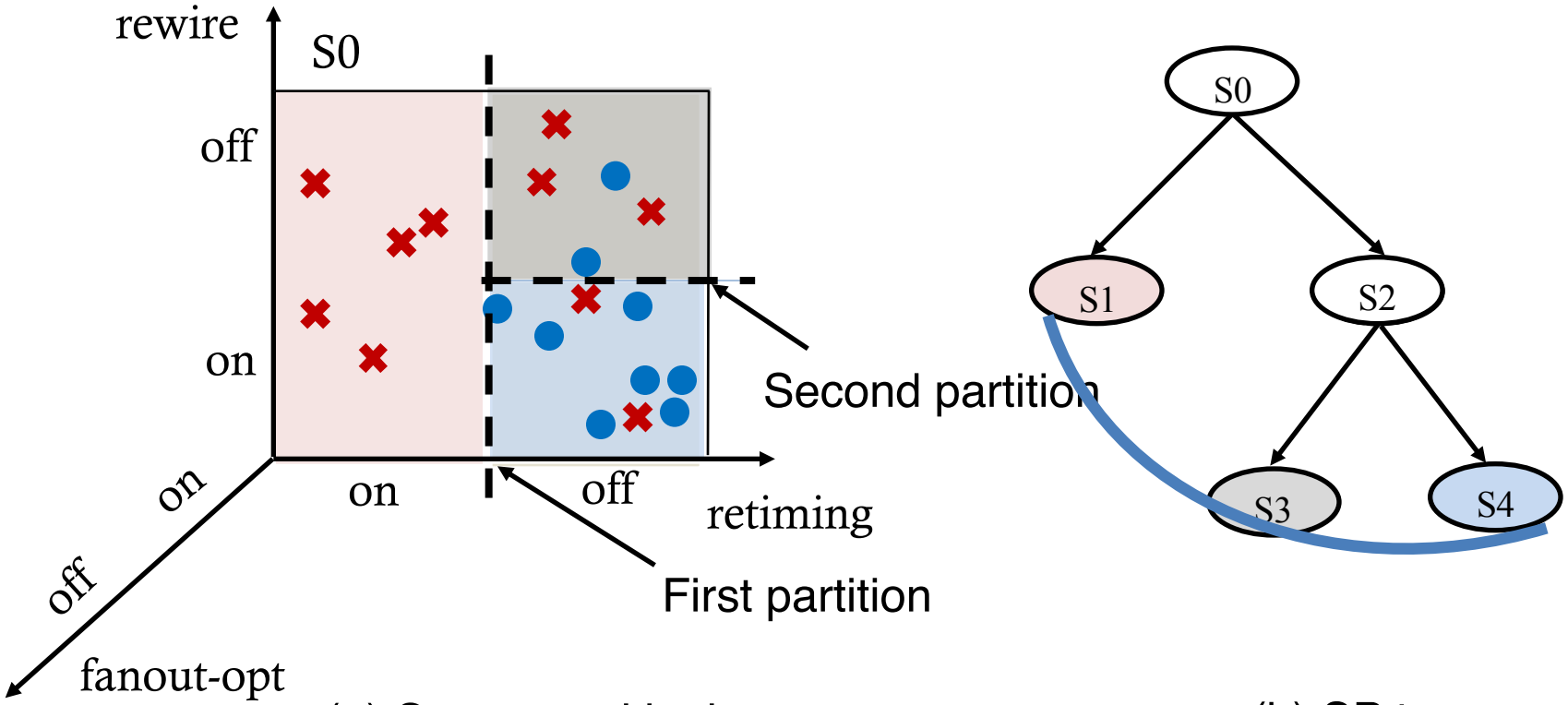
Information gain $= 0.3 - \frac{1}{2}(0.3 + 0.3) = 0$

$H(D_1) = H(D_2) = 0$

Information gain $= 0.3 - \frac{1}{2}(0 + 0) = 0.3$

MAB-directed Resource Allocation

- Question
  - Given a partitioned search space, how to allocate computing resources to maximize QoR?
    - Discard unpromising region? ⇒ Miss opportunity

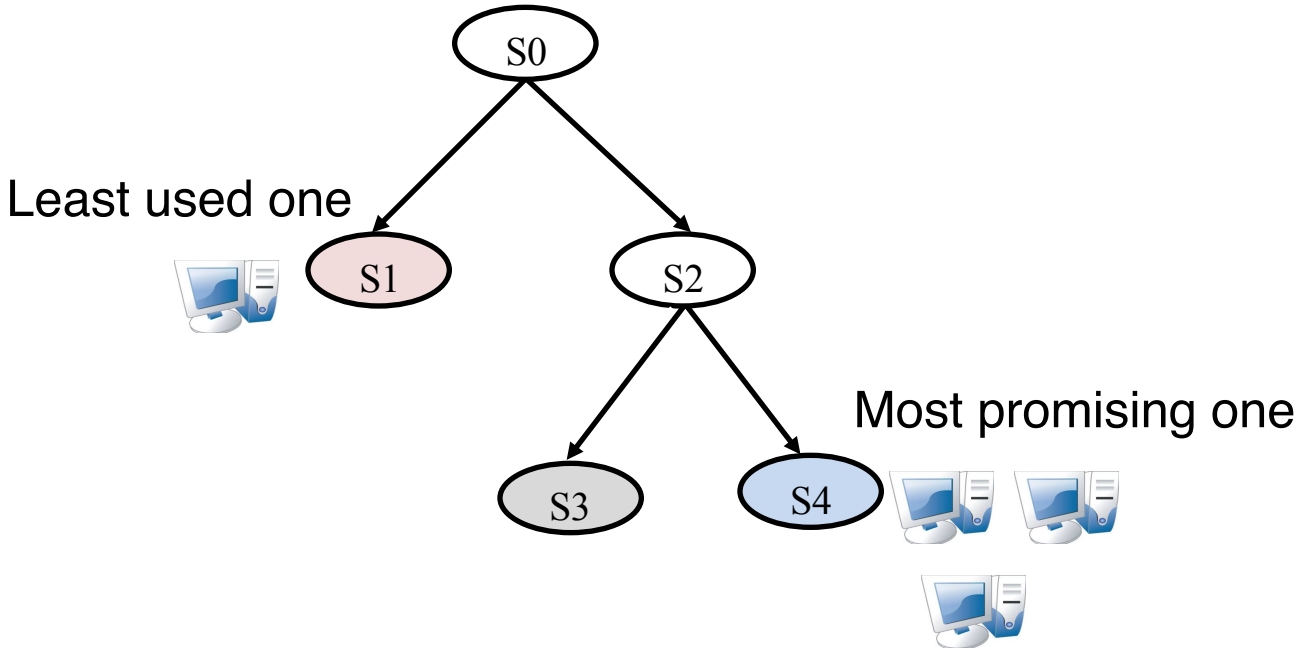


(a) Space partitioning (b) SP tree

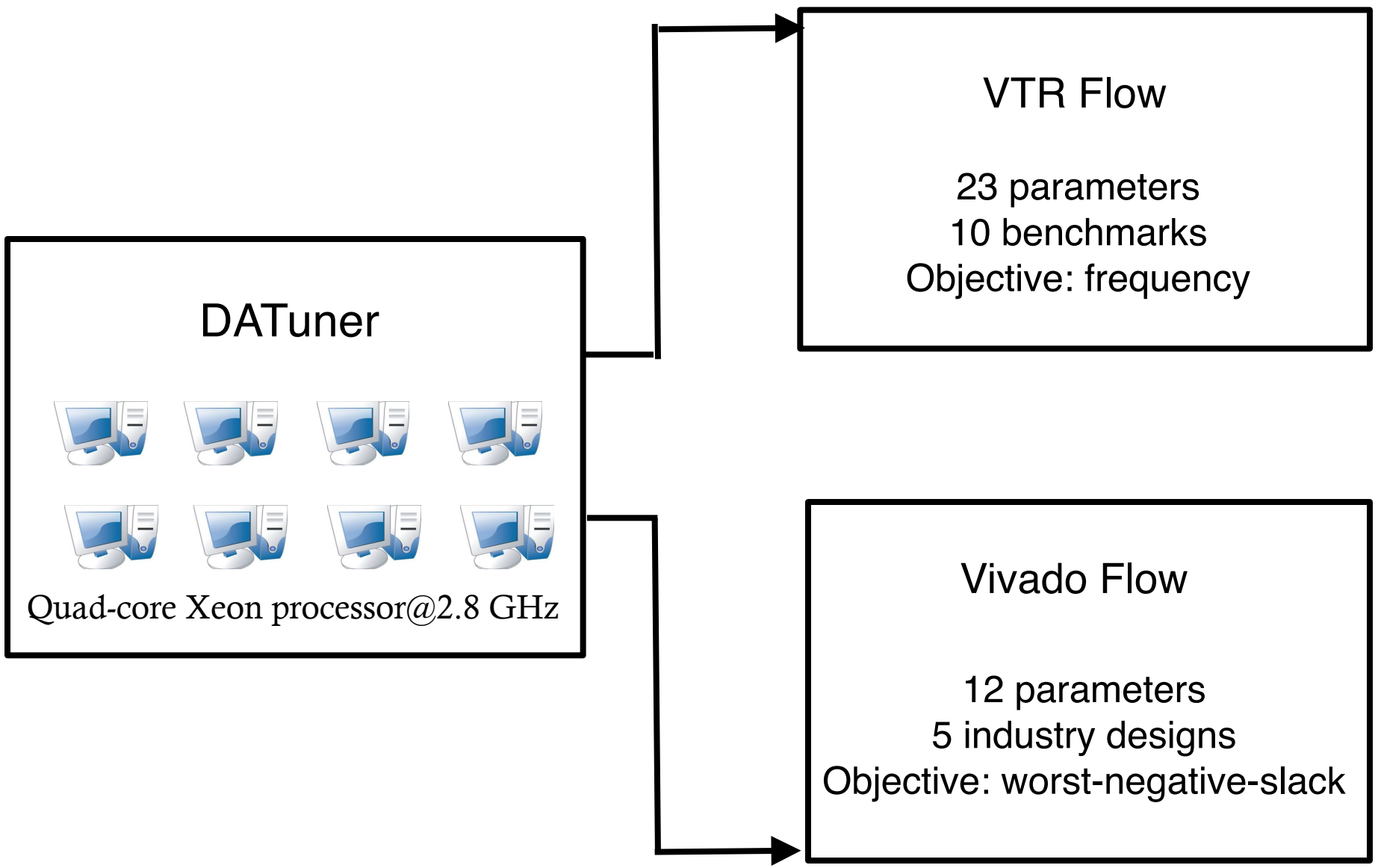
MAB-directed Resource Allocation

- Bandit-based score assignment
  - **Exploitation :** average QoR of samples
  - **Exploration:** inversely related to the number of times (H_t) subspace has been explored

$$\mathrm{Score}(s_t) = \mathrm{avg\ QoR} + C * \sqrt{\frac{2\lg|H|}{H_t}}$$

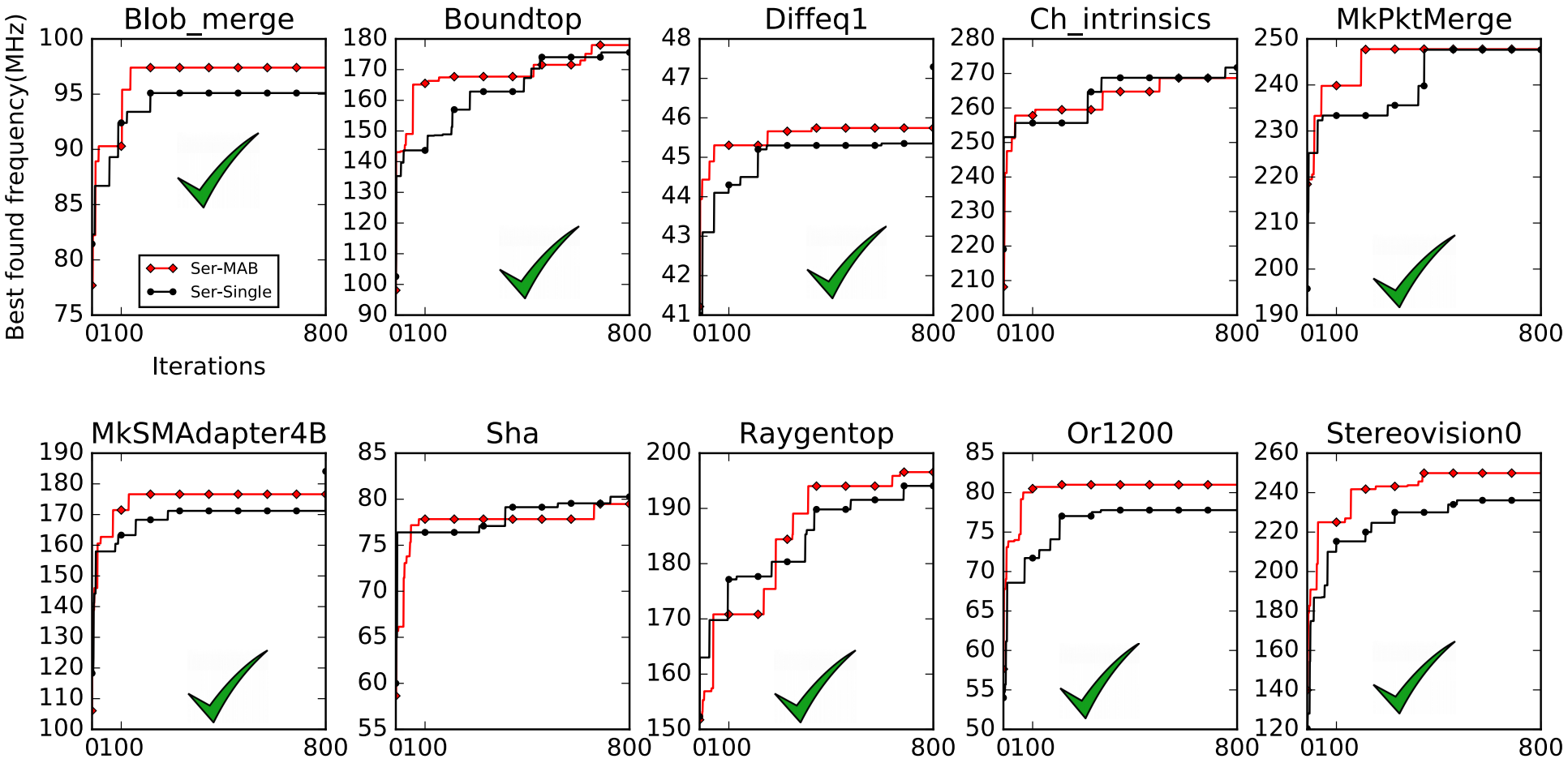


Experimental Results: Environment Setting



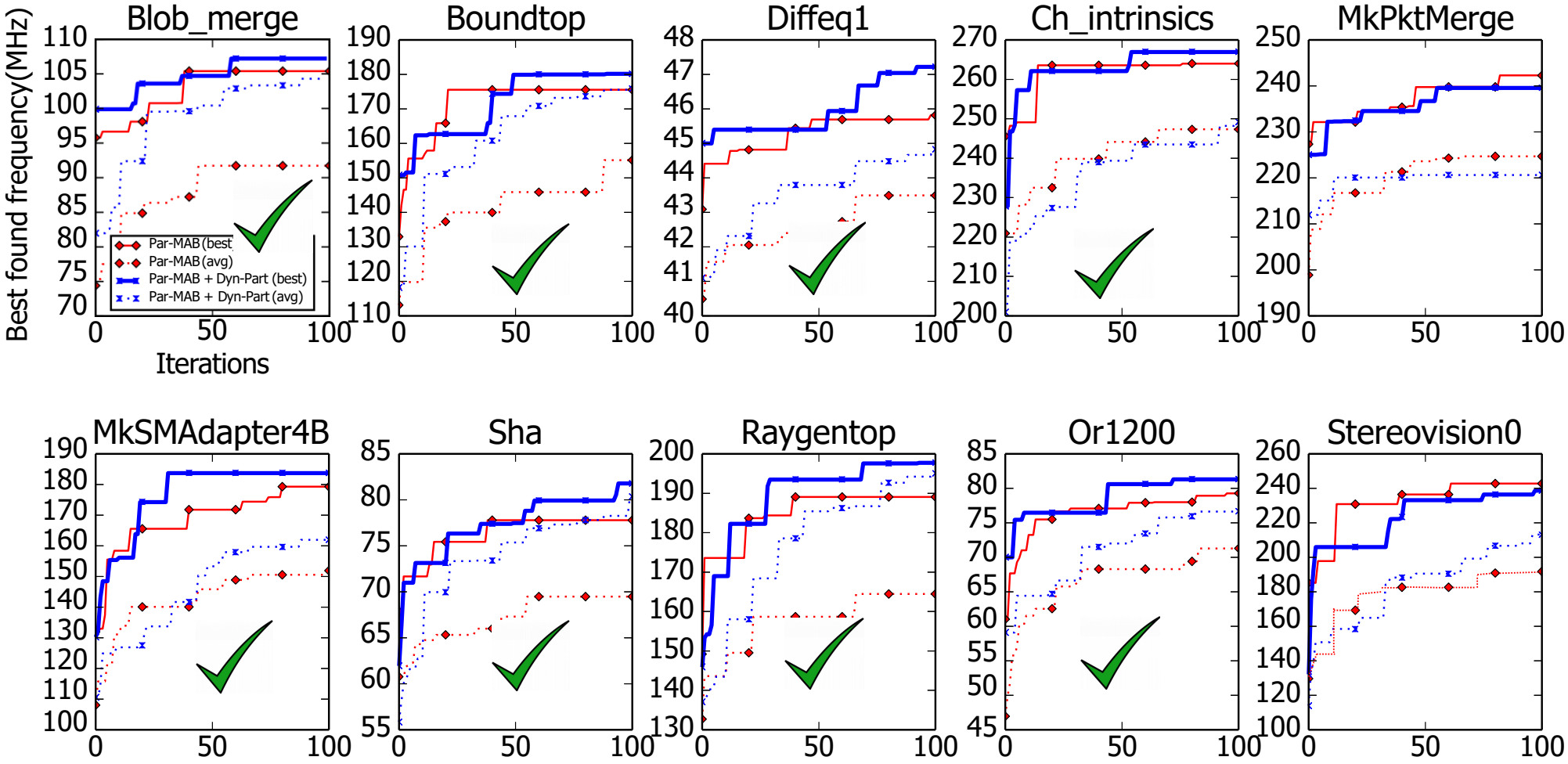
VTR Tuning: Bandit-based vs. Single Search

- Bandit-based search (*Ser-MAB*) outperforms single search (*Ser-Single*) on multiple designs => 2nd level bandit is effective



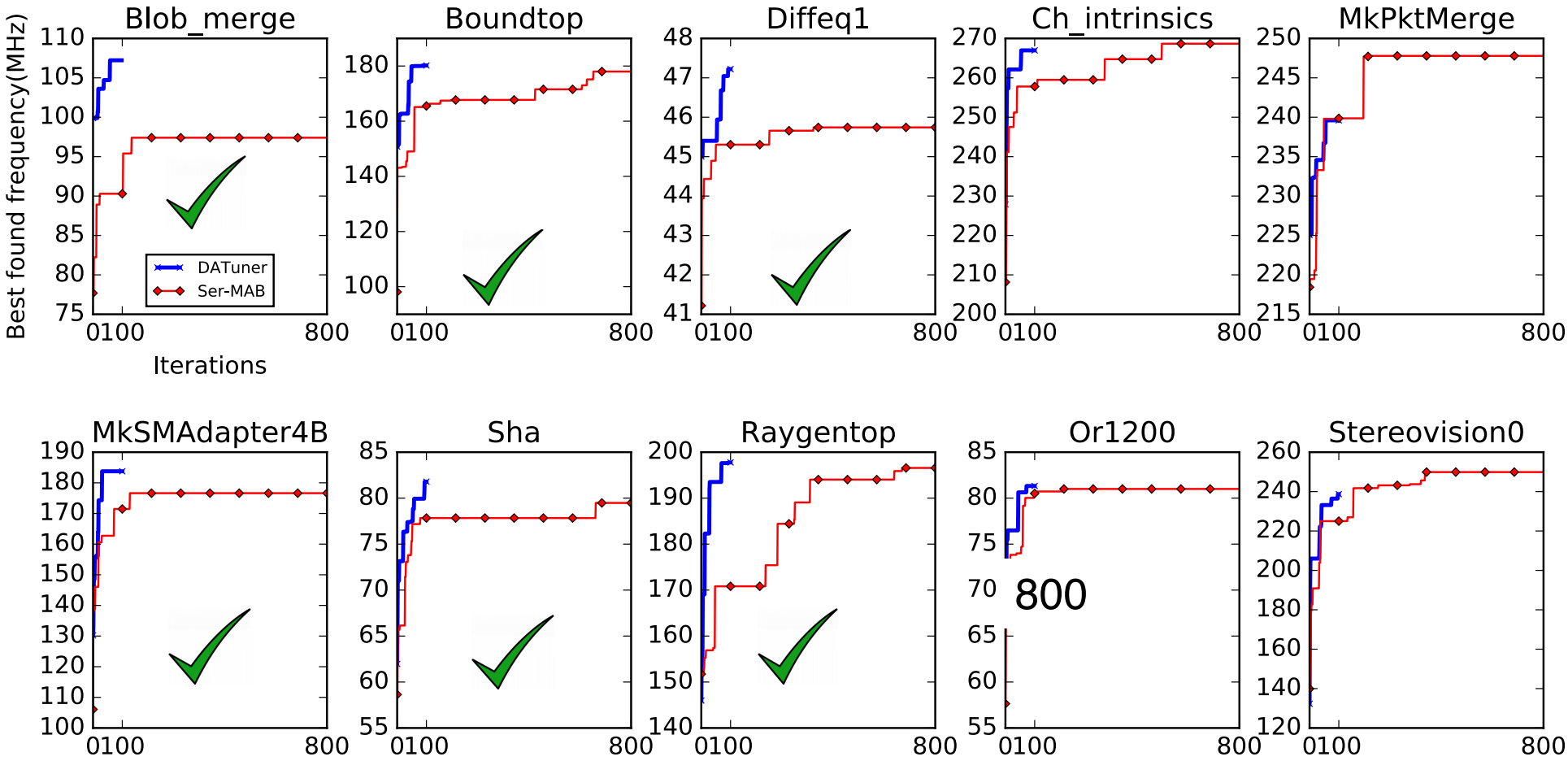
VTR Tuning: Dynamic vs. Static Space Partitioning

- Dynamic partitioning (*Par-MAB+Dyn-Part*) outperforms static partitioning (*Par-MAB*) on multiple designs



VTR Tuning: Parallel vs. Sequential Search

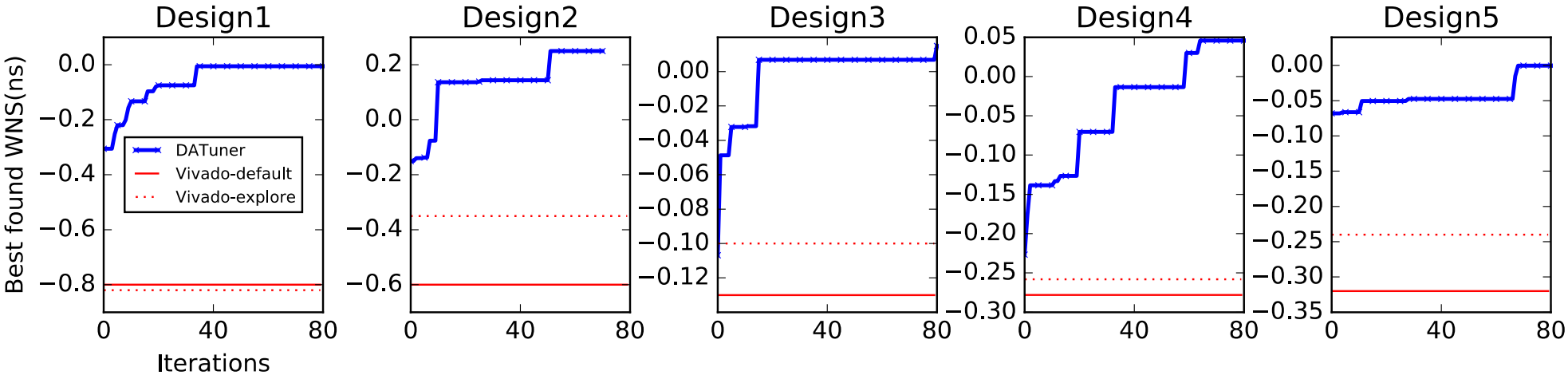
- Parallel searching (*DATuner*) outperforms sequential search (*Ser-MAB*) on multiple designs => 1st level bandit is effective



- Speedup: compare the runtime to reach QoR target
  - 11X faster with 8 parallel searches over sequential search

Vivado Tuning

- Resolve timing problem for industry designs



Compare with default setting, best-found results improve WNS by 11%

Circuit	FF	LUT	Constraint (ns)	Device
Design1	14545	14122	2.60	Virtex 7K160T
Design2	17847	29012	6.55	Virtex 7K70T
Design3	18204	28361	2.00	Virtex 7K160T
Design4	26098	17242	2.65	Virtex 7VX330T
Design5	27873	38261	4.30	Virtex 7VX330T

Circuit	WNS ratio	RT ratio	LUT ratio	FF ratio
Design1	0.77	2.38	0.83	1.00
Design2	0.93	6.80	1.00	1.00
Design3	0.94	3.26	1.00	1.00
Design4	0.88	4.30	1.03	1.00
Design5	0.90	1.62	1.00	1.00
Avg.	0.89	3.67	0.97	1.00

Conclusions

- Bandit-based autotuning method is quite promising
  - 1st level: Dynamic partitioning + bandit-based resource allocation
  - 2nd level: Bandit-based meta-heuristics outperform a single search technique
- Parallelization is key to enable feasible runtime
- DATuner: Open-source release coming soon
- Future improvements: Scale up to cloud platforms
  - Related work: Plunify InTime [N. Kapre et al., FPGA'15] – Cloud platform to tune FPGA timing performance