A Sampling-Based Approach to GPGPU Performance Auto-Tuning

Wilson Feng and Tarek S. Abdelrahman

The Edward S. Rogers Department of
Electrical and Computer Engineering
University of Toronto

wilson.feng@mail.utoronto.ca tsa@ece.utoronto.ca

Outline

- Motivation
- Regression Trees
- Auto-Tuning Strategy
- Evaluation
- Related Work
- Conclusions and Potential Extensions

GPU Computing

- Graphics Processing Units (GPUs) have increased in popularity over the past decade

- GPU programmers need to *optimize* their code
  - Restructure the GPU code to exploit the underlying GPU hardware
  - Several optimizations whose impact is sometimes non-intuitive
  - Programmers have to explore many combinations
  - Number of *optimization configurations* to examine can be prohibitively large to manually explore
  - Manual exploration of optimization configurations is challenging

Auto-Tuners

- Interest in frameworks to *automatically* and *efficiently* explore a large space of optimization configurations to determine good performing ones
  - Analytic based: uses a model of the program and target machine
  - Machine Learning based: uses training programs to build machine learning model, then uses mode to predict optimizations for new programs
  - ✓ Heuristic based: uses a heuristic to search and prune the space of all possible configurations (*optimization space*) starting from some initial configuration

This Work

- We propose a novel heuristic search strategy for GPU auto-tuners
- Informed by a machine learning model
  - A *regression tree* used in a non-conventional way
  - The tree is used to prune the optimization space
- Our strategy is able to obtain in less time better performing configurations compared to state-of-the-art techniques
  - As implemented in OpenTuner

Outline

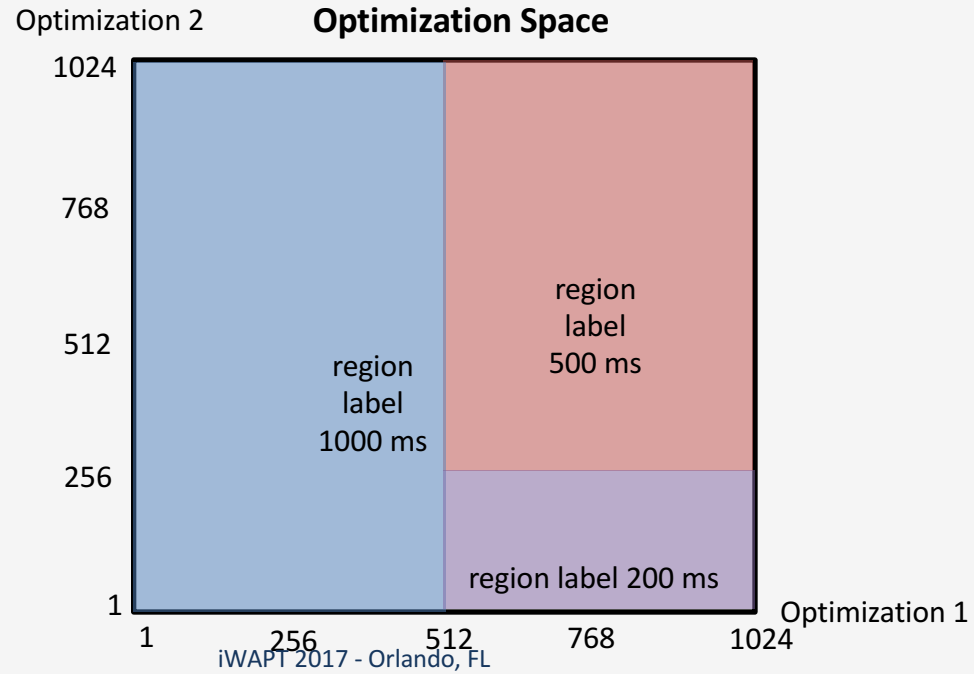
- Motivation
- Regression Trees
- Auto-Tuning Strategy
- Evaluation
- Related work
- Conclusions and Potential Extensions

Regression Trees

- ML model that divides the configuration space into rectangular regions, where configurations in each region are predicted with one label

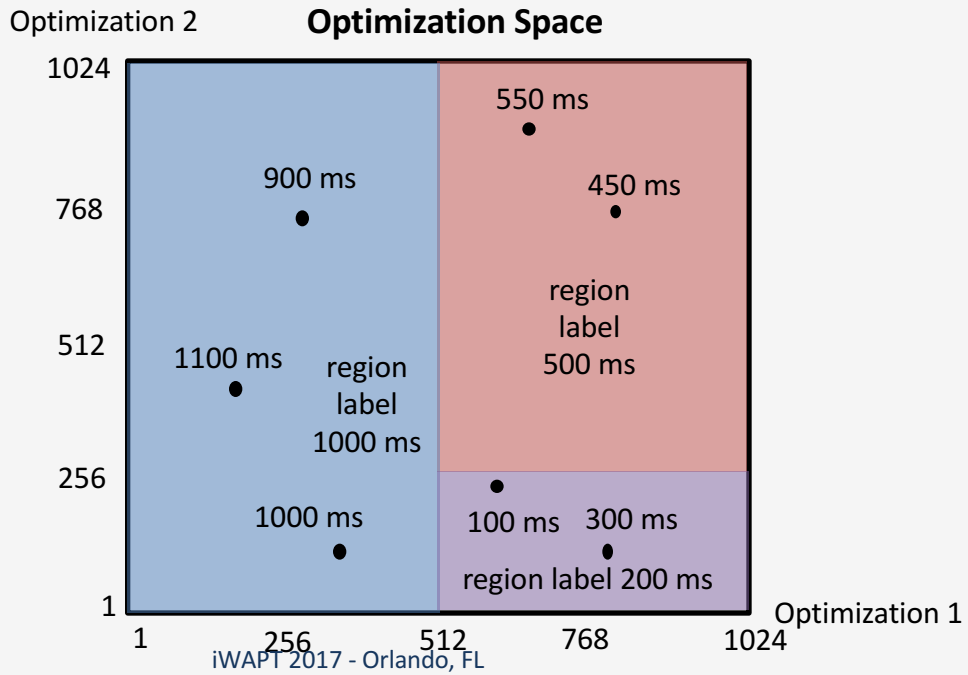
Regression Trees

- ML model that divides the configuration space into rectangular regions, where configurations in each region are predicted with one label



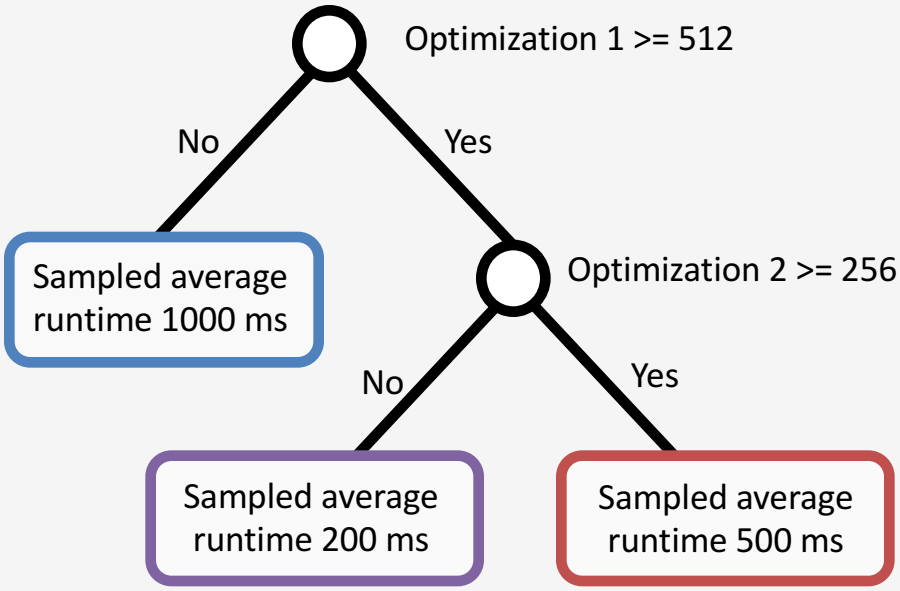
Regression Trees

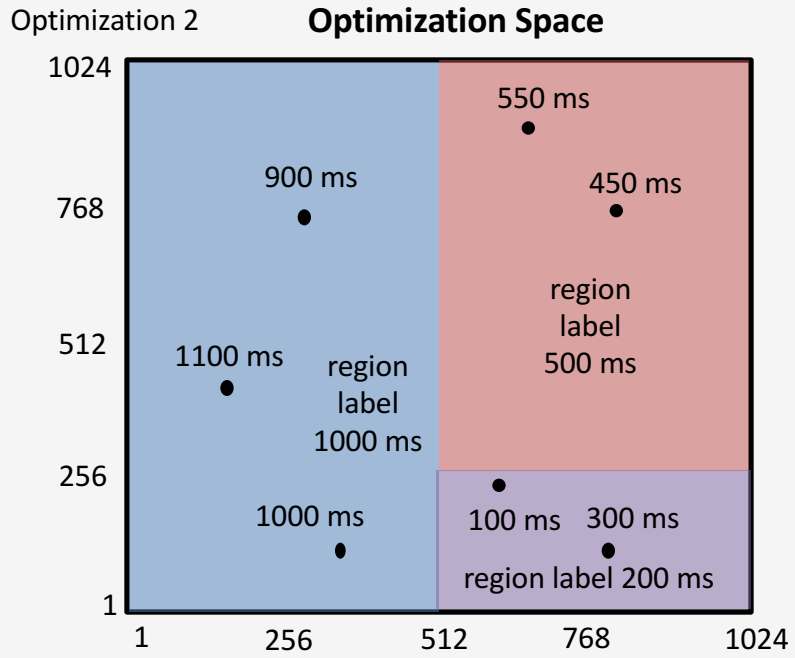
- Training data is used to partition the space so the mean square error between label and data in region is minimized



Regression Trees

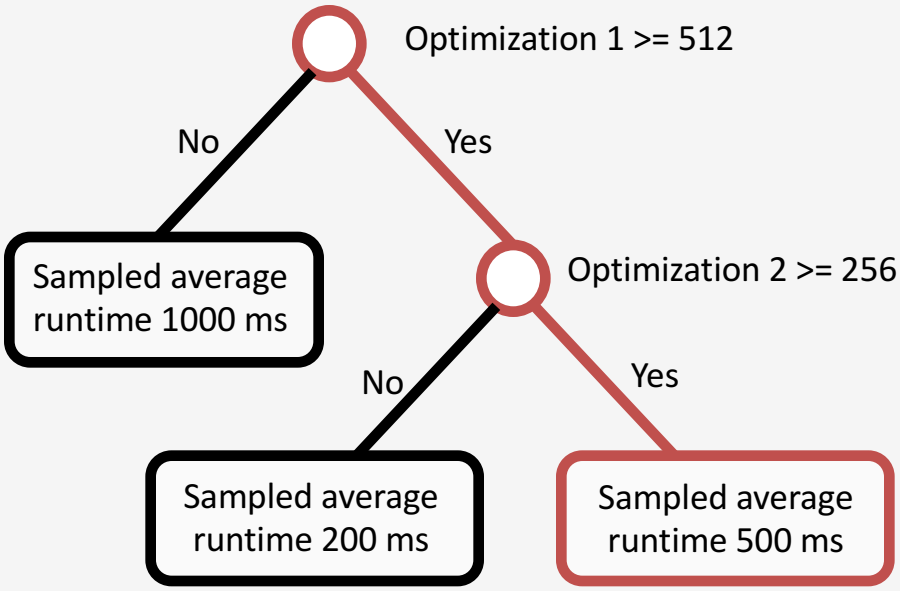
- The model can also be represented as a tree structure
  - Paths form constraints on optimization values

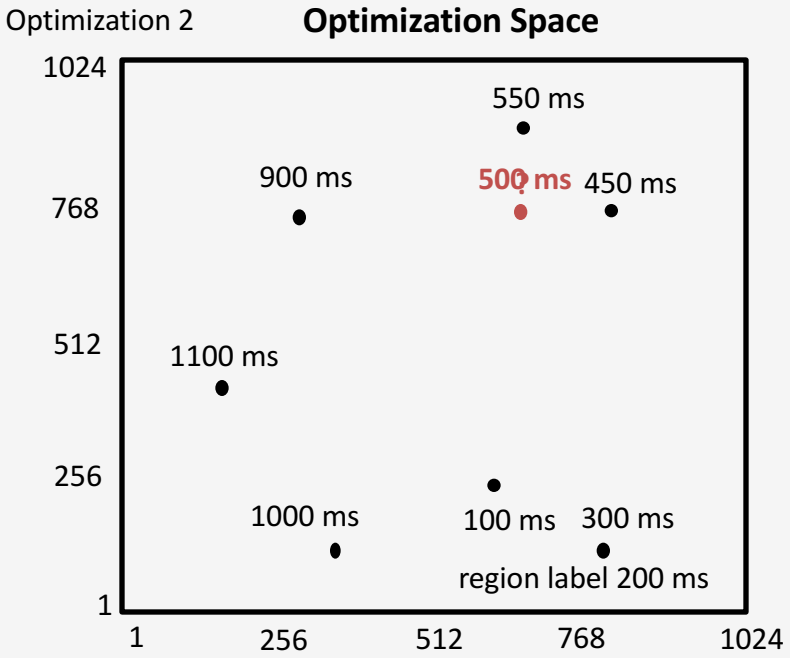




Regression Trees

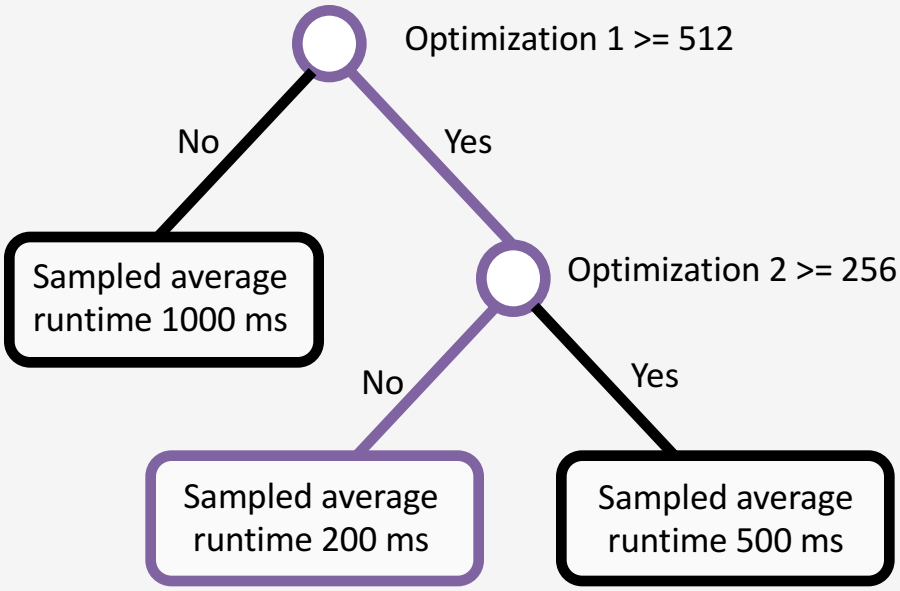
- The model is used to predict the performance of an optimization configuration

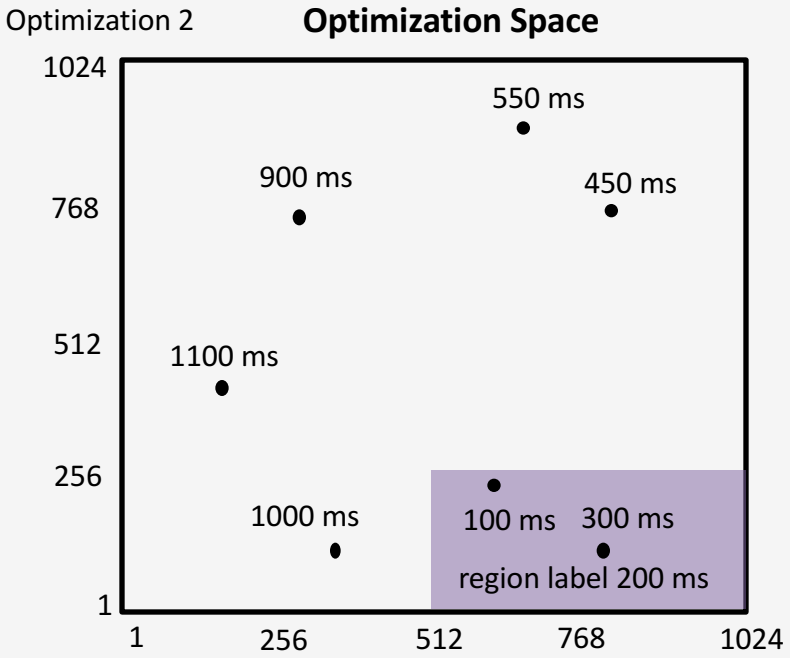




Regression Trees

- Traversing the tree and using the constraints can be used to prune the space



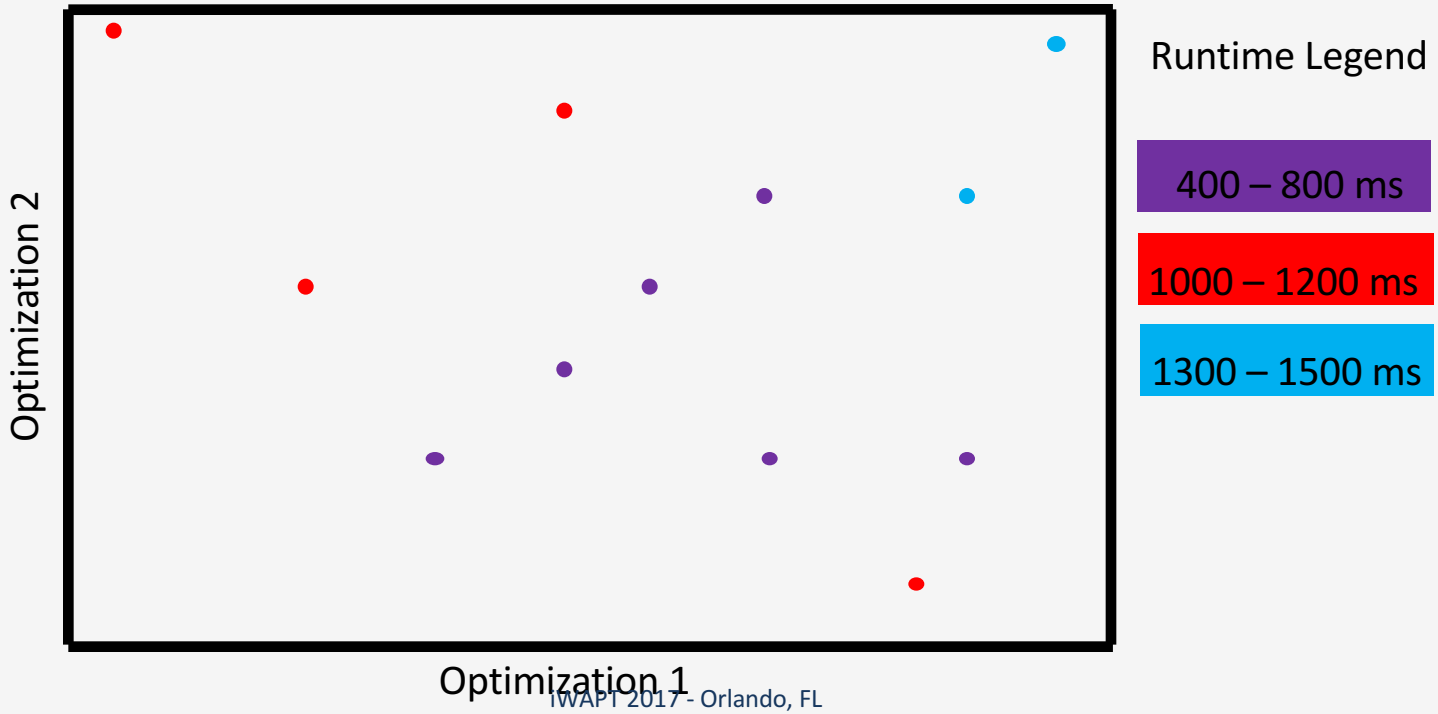


Outline

- Motivation
- Regression Trees
- Auto-Tuning Strategy
- Evaluation
- Related work
- Conclusions and Potential Extensions

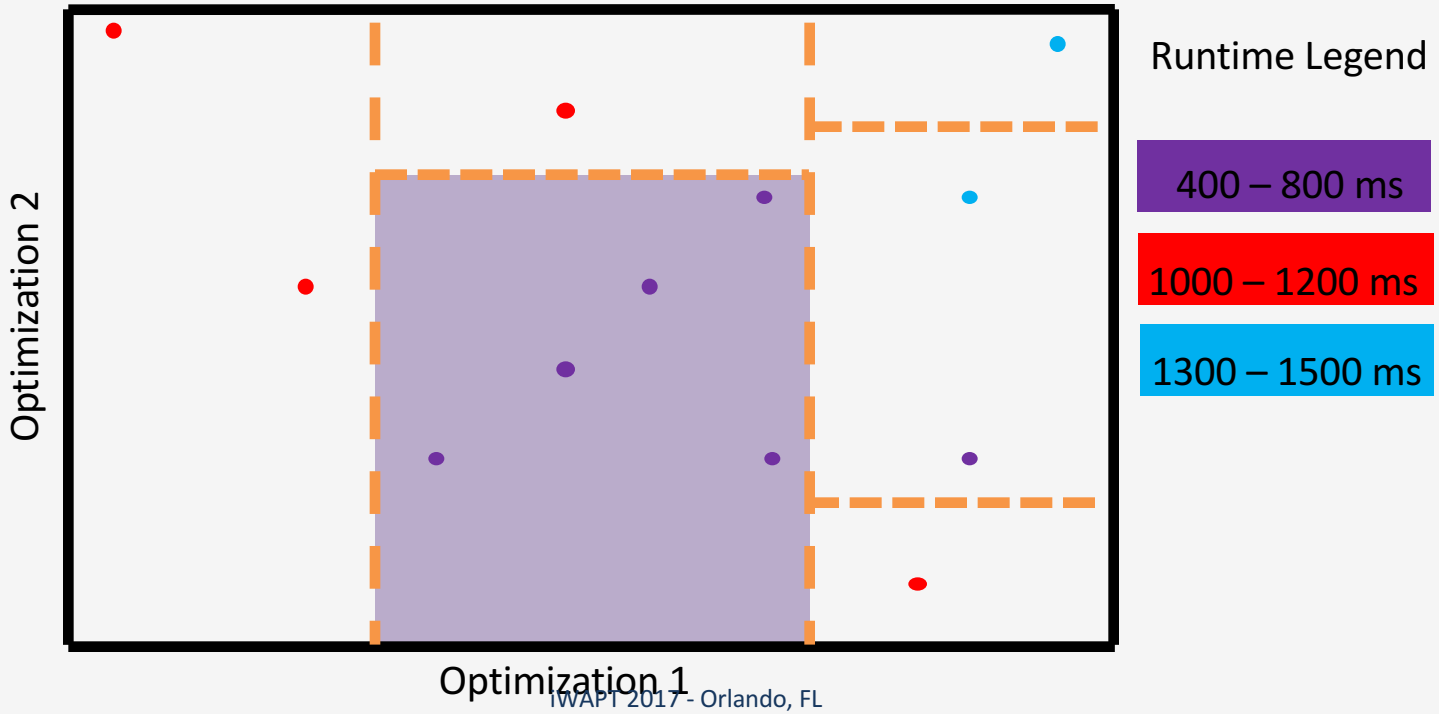
Auto-Tuning Strategy

- Sample configurations from the full space



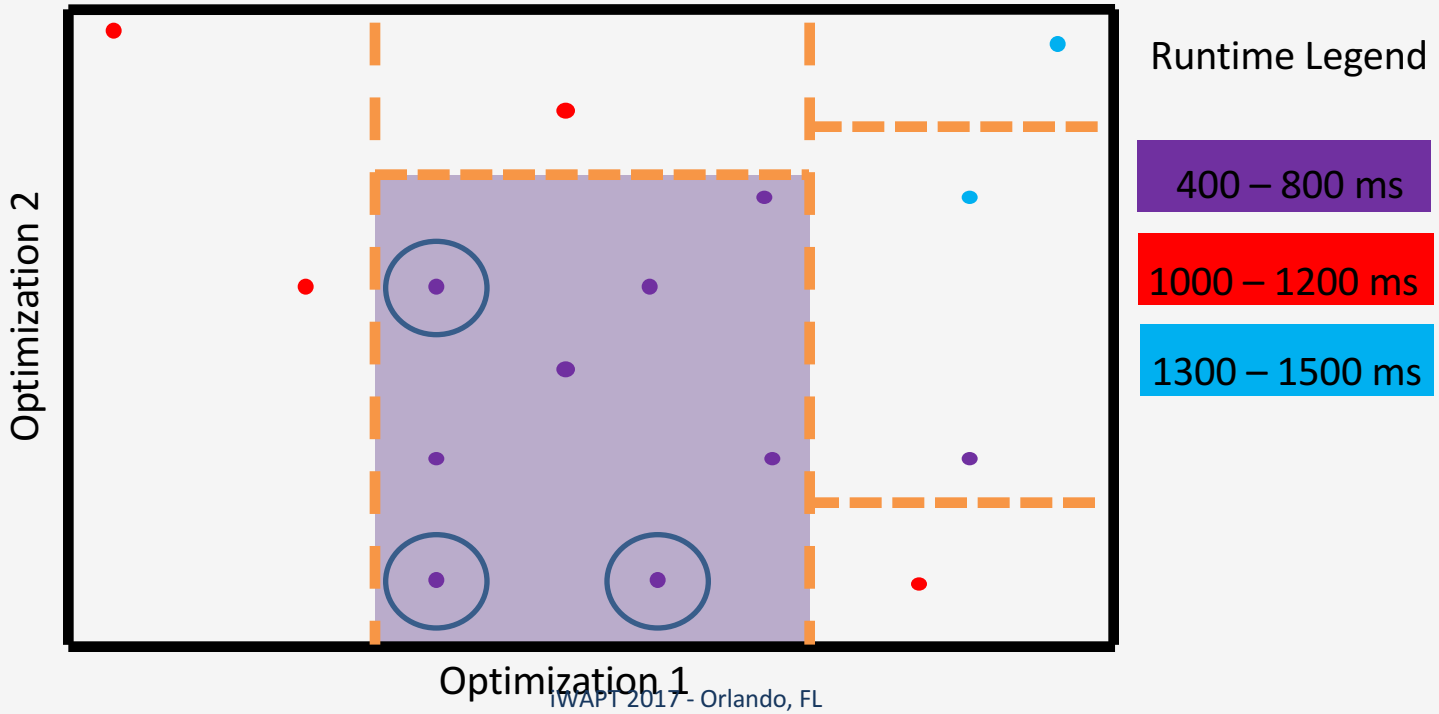
Auto-Tuning Strategy

- Train regression tree and focus search on region with lowest predicted runtime



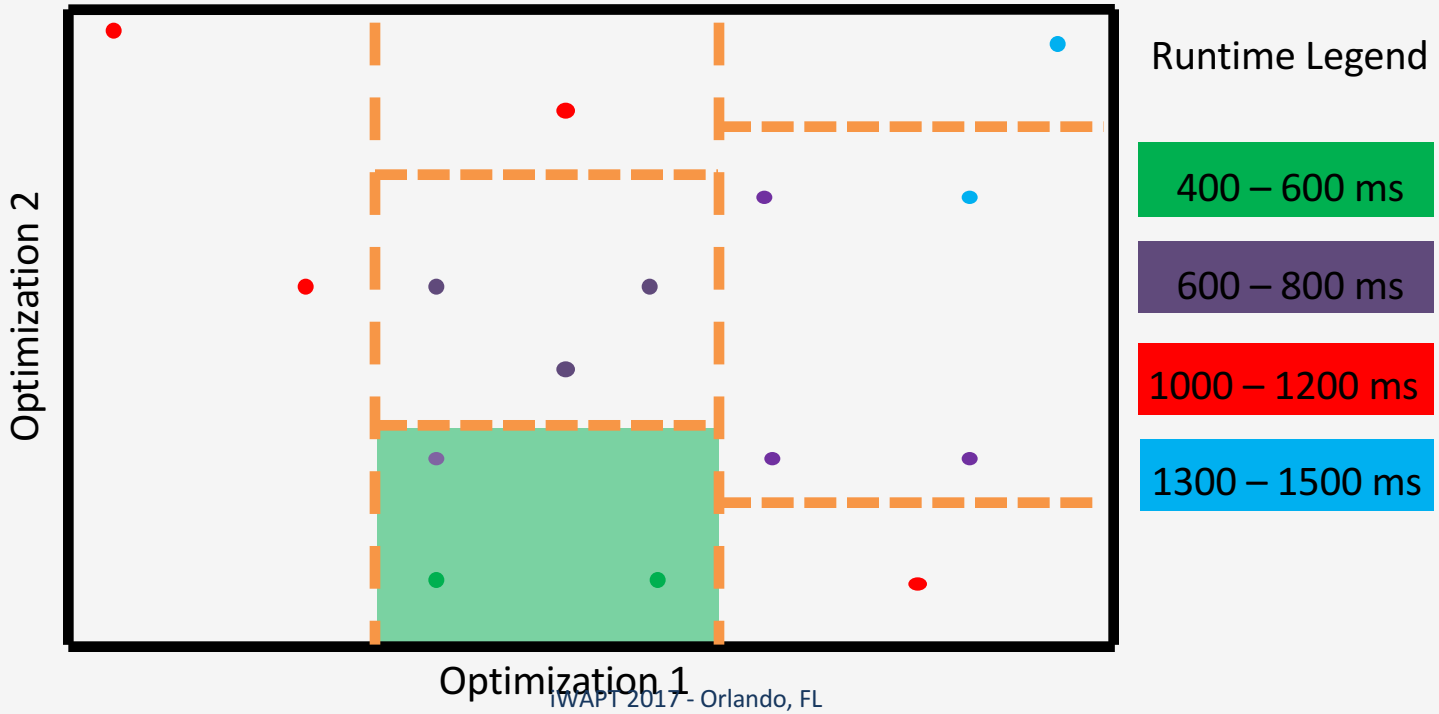
Auto-Tuning Strategy

- Sample more configurations within region and retrain a new regression tree with all samples



Auto-Tuning Strategy

- Region size is smaller and its predicted runtime is lower



Auto-Tuning Strategy

- Repeat the process of sampling, training and focusing on the "best" region until an auto-tuning time budget is exhausted

- The hypotheses of the approach:
  - The region with the lowest predicted runtime contains high performing configurations, ideally the best one
  - This region can be further pruned by iteratively sampling more configurations in the region and building additional regression trees
    - Predicted time gets lower

Sampling-Based Startegy Formulation

$\mathcal{R} = R_o$
$S = \{\ \}$
$n = \sigma \cdot k + b$

k: number of optimizations
σ : scaling factor
b: base offset

while *elapsed time* $< \tau$ **do**
  while *count* $< n$ **do**
    s = sample a configuration from $\mathcal{R}$
    count $++$
    Execute s and record its runtime
    $S = S + s$
    C = best performing configuration in S
  end
  Train regression tree $\mathcal{T}$ with S
  $\mathcal{N}$ = $\mathcal{T}$'s leaf node with best predicted performance
  $\mathcal{P}$ = path from $\mathcal{T}$'s root node to $\mathcal{N}$
  $\mathcal{R}$ = narrowed down R_o using $\mathcal{P}$'s constraints
end

Outline

- Motivation
- Regression Trees
- Auto-Tuning Strategy
- Evaluation
- Related work
- Conclusions and Potential Extensions

Benchmarks

Benchmark	Optimizations	# of Configs	Days to Explore
Hotspot	Thread block size, transpose temperature/power/buffer arrays	57424	1/2
K-means	Thread block size, # of threads working on each point, coalesce features to be processed	2293504	166
BFS	Thread block size, # of nodes processed per thread, thread process consecutive nodes, store node info in parallel arrays, max registers per thread, use of cache	10158080	1/8
Streamcluster	Thread block size, points per thread, store points in parallel arrays, use shared memory, max registers per thread, use of cache	1310720	50
lavaMD	Thread block size, local memory size	573376	30
GEMM	Thread block size, tiling length, use shared memory	2745216	302
Covariance	Thread block size, loop tiling factor	33056976	1627
Correlation	Thread block size, loop tiling factor	33056976	1531

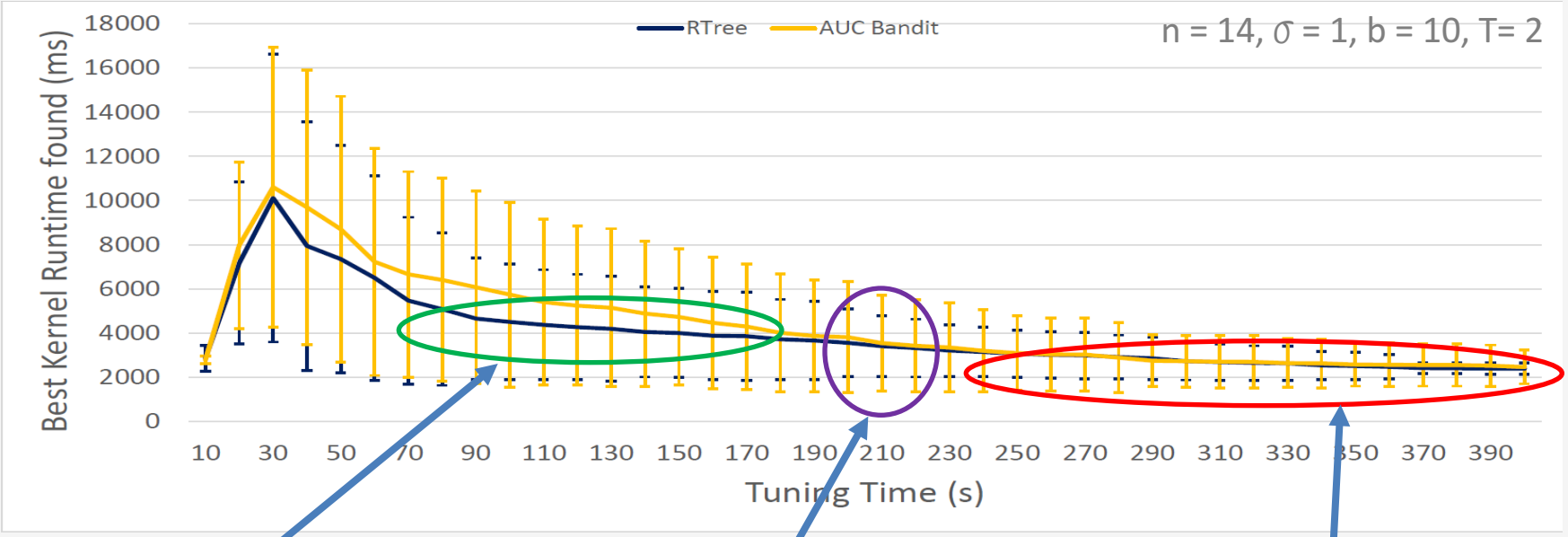
Platforms

- Implemented our strategy the OpenTuner (version 0.5.0) framework developed at MIT (Ansell et al., 2014)
  - State-of-the-art framework for auto-tuning
  - Implements several heuristics
  - Lacks handling optimization inter-dependence
- Scikit-learn v. 0.18 with default parameters for regression trees
- Run OpenTuner on an Intel i5-6400, 8GB RAM, Nvidia GTX1060

Evaluation Experiments

- Run OpenTuner with different auto-tuning time budgets
  - Collect best kernel configuration/runtime found
  - Repeat experiment 100 times to address sampling randomness
  - Report the average of best kernel runtimes and the standard deviation
- Compare our strategy's performance to OpenTuner's Area Under the Curve (AUC) Bandit strategy
  - Determines which of Simulated Annealing, Differential Evolution and Nelder Mead is best at a given step and allocates it more time
  - Shown effective in practice (Ansell et al., 2014)

Strategy Performance – K-means

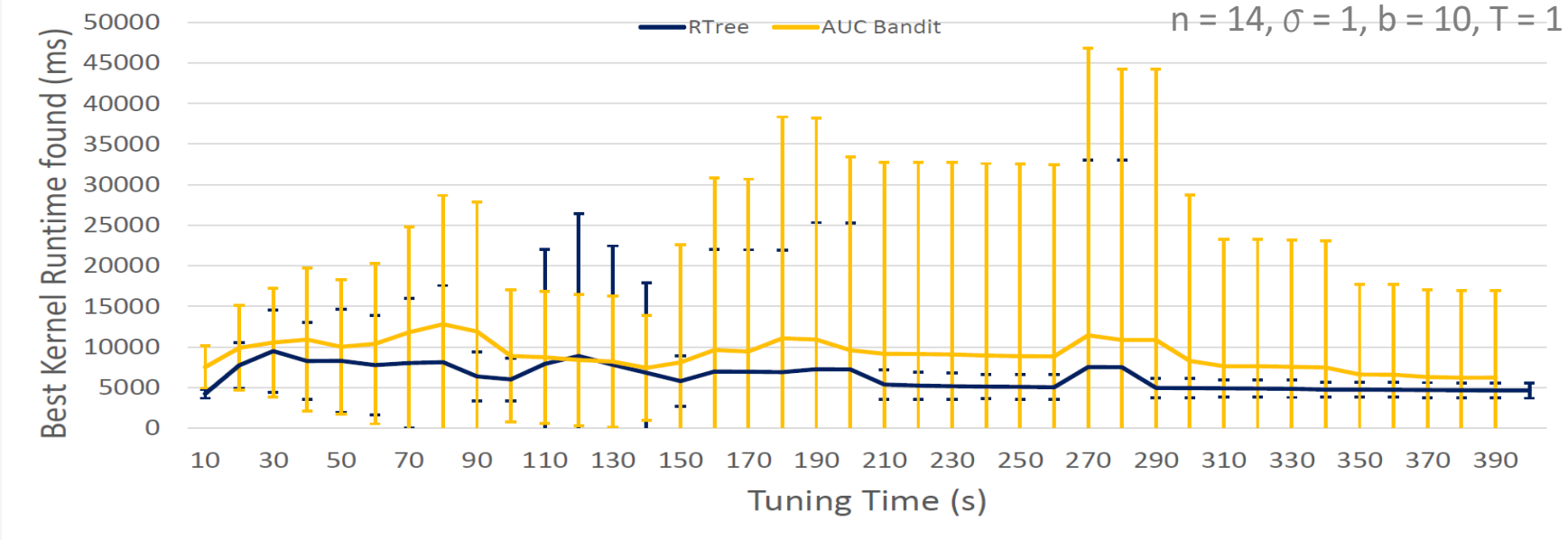


Our strategy is able to converge on good performing configurations faster than OpenTuner's strategy

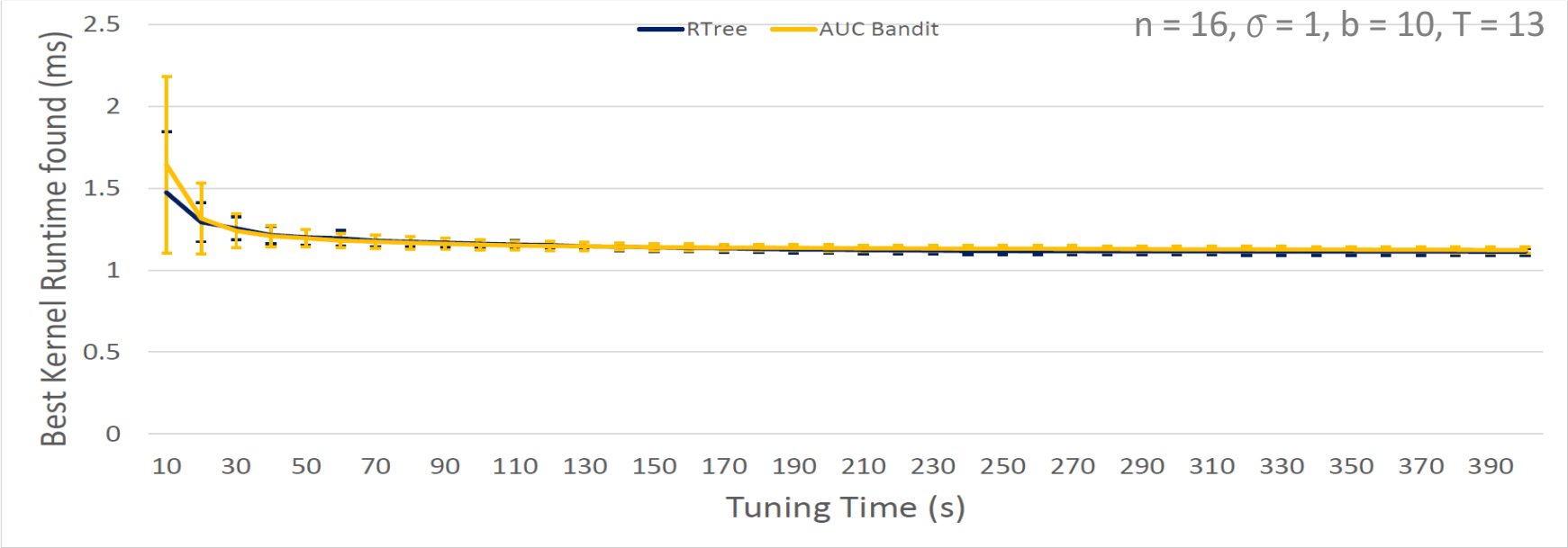
Our strategy is more robust as evident by smaller standard deviation

Both strategies eventually converge to the same kernel runtime

Strategy Performance - GEMM



Strategy Performance - BFS



Quality of Configurations

Benchmark	AUC Bandit Total # of Samples Taken	R. Tree Total # of Samples Taken	AUC Bandit Total Kernel Runtime (s)	R. Tree Total Kernel Runtime(s)
Hotspot	221	236	218	209
K-means	33	31	224	230
BFS	245	246	0.42	0.42
Streamcluster	47	47	144	144
lavaMD	57	96	326	267
GEMM	17	16	305	318
Covariance	82	120	401	373
Correlation	81	116	402	371

Auto-tuning time budget of 200 seconds

Related Work

- Several auto-tuning frameworks already exists
  - Examples: OpenTuner, Patus, Nitro, etc.
  - Use various heuristics: Nelder-Mead, Differential Evolution, Gradient Descent, etc.
  - Our work focuses on the design of an optimization heuristic instead of an auto-tuning framework
- Regression trees are used as a prediction model for GPU kernel performance
  - Starchart is a framework that takes samples and trains a regression tree model
  - Characterizes optimization space and predicts performance

Concluding Remarks

- A search strategy based on sampling and non-traditional use of regression trees

- Experimental evaluation:
  - Delivers better performing configurations than existing strategies
  - Is more robust
  - Validates our hypothesis that region of interest prunes the space and contains high performing configurations
  - Explores strategy parameters

Potential Extensions

- Incorporate regression tree strategy into OpenTuner's AUC method
  - Evaluate the performance of the combined strategies
- Heuristics for determining good auto-tuning time
  - Many scenarios
- Further explore parameters of strategy

Questions?