

# Toward Transparent and Parsimonious Methods for Automatic Performance Tuning

---

Pedro Bruel

July 9 2021

## Advisors

Alfredo Goldman (USP)  
Arnaud Legrand (CNRS)  
Brice Videau (ANL)

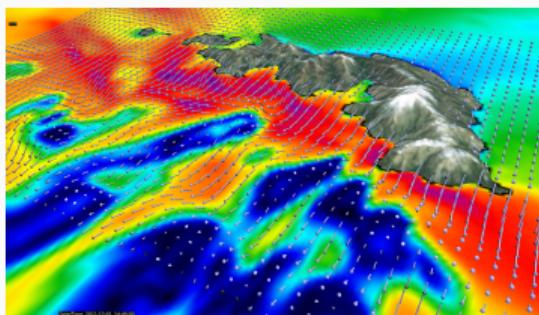
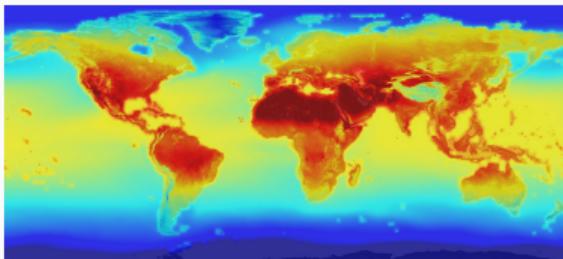
## Jury

Stefan M. Wild (ANL)  
Albert Cohen (Google)  
Boyana Norris (UO)  
Lucia Drummond (UFF)

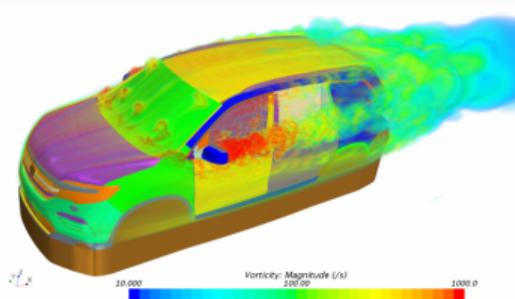
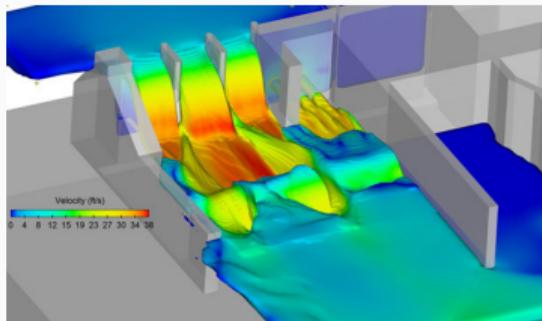


# High Performance Computing is Needed at Multiple Scales, ...

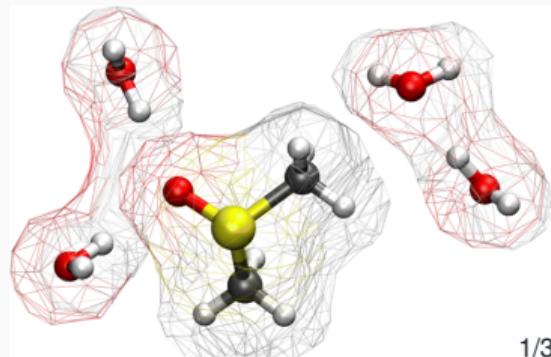
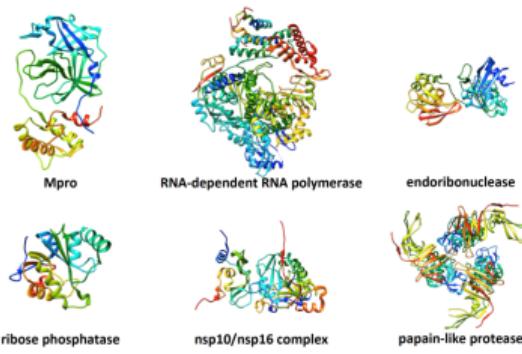
Climate simulation for policies  
to fight climate change



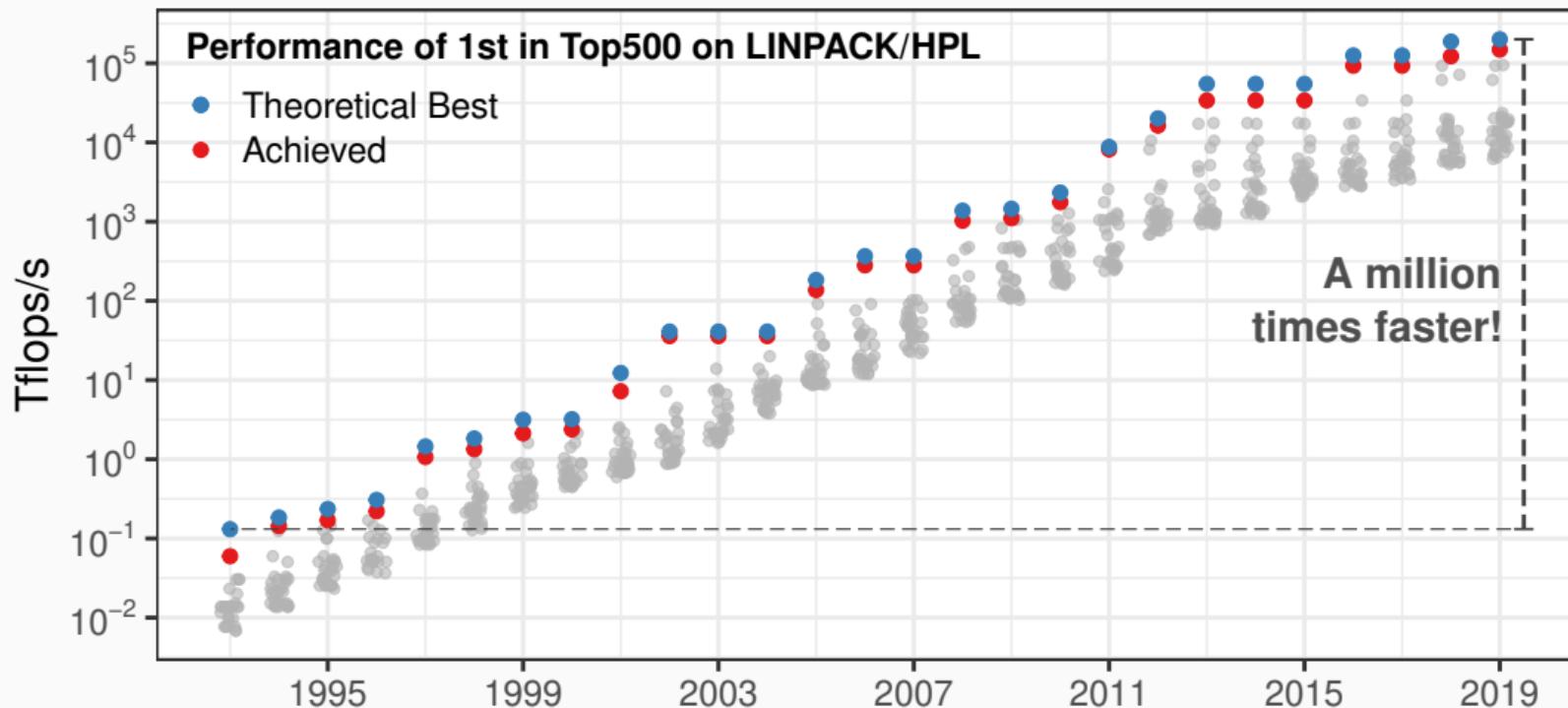
Fluid dynamics for stronger  
infrastructure and fuel efficiency



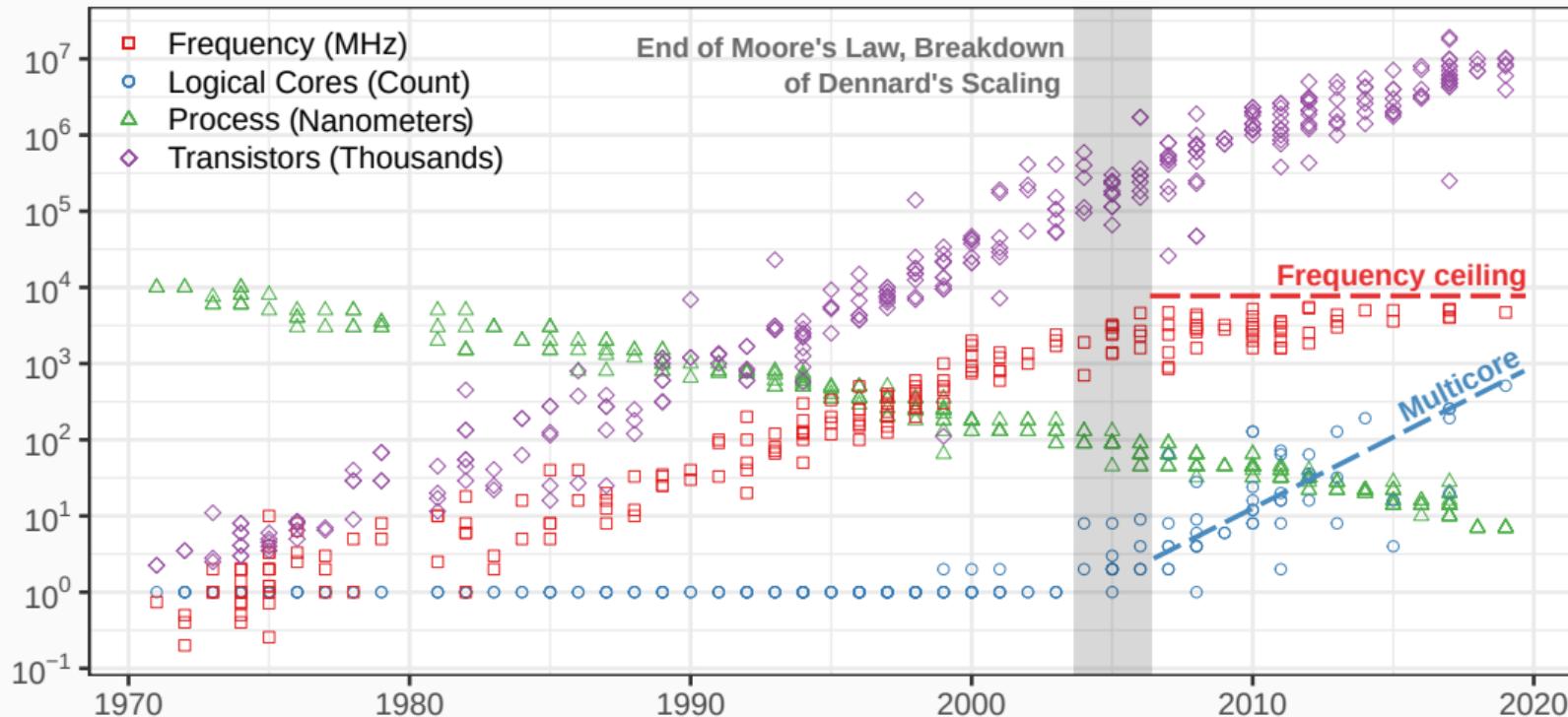
Molecular dynamics for virtual  
testing of drugs and vaccines



... and the Performance of Supercomputers has so far Improved Exponentially



# Software must Improve to Leverage Complexity, and Autotuning can Help



# An Autotuning Example: Loop *Blocking* and *Unrolling* for Matrix Multiplication

## Optimizing Matrix Multiplication

How to **restructure memory accesses** in loops to increase throughput by leveraging **cache locality**?

```
int N = 256;  
  
float A[N][N], B[N][N], C[N][N];  
int i, j, k;  
// Initialize A, B, C  
for(i = 0; i < N; i++){ // Load A[i][]  
    for(j = 0; j < N; j++){  
        // Load C[i][j], B[][], to fast memory  
        for(k = 0; k < N; k++){  
  
            C[i][j] += A[i][k] * B[k][j];  
        }  
  
        // Write C[i][j] to main memory  
    }  
}
```

# An Autotuning Example: Loop *Blocking* and *Unrolling* for Matrix Multiplication

## Optimizing Matrix Multiplication

How to **restructure memory accesses** in loops to increase throughput by leveraging **cache locality**?

```
int N = 256;
int B_size = 4;
float A[N][N], B[N][N], C[N][N];
int i, j, k, x, y;
// Initialize A, B, C
for(i = 0; i < N; i += B_size){
    for(j = 0; j < N; j += B_size){
        // Load block (i, j) of C to fast memory
        for(k = 0; k < N; k++){
            // Load block (i, k) of A to fast memory
            // Load block (k, j) of B to fast memory
            for(x = i; x < min(i + B_size, N); x++){
                for(y = j; y < min(j + B_size, N); y++){
                    C[x][y] += A[x][k] * B[k][y];
                }
            }
        }
        // Write block (i, j) of C to main memory
    }
}
} // One parameter: B_size
```

# An Autotuning Example: Loop Blocking and Unrolling for Matrix Multiplication

## Optimizing Matrix Multiplication

How to **restructure memory accesses** in loops to increase throughput by leveraging **cache locality**?

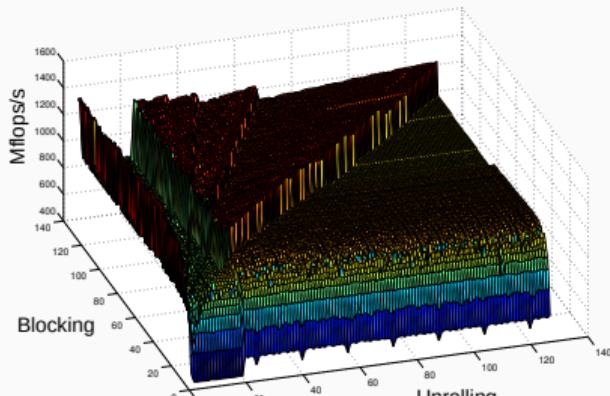
```
int N = 256;
int B_size = 4; int U_size = 2;
float A[N][N], B[N][N], C[N][N];
int i, j, k, x, y;
// Initialize A, B, C
for(i = 0; i < N; i += B_size){
    for(j = 0; j < N; j += B_size){
        // Load block (i, j) of C to fast memory
        for(k = 0; k < N; k++){
            // Load block (i, k) of A to fast memory
            // Load block (k, j) of B to fast memory
            for(x = i; x < min(i + B_size, N); x++){
                for(y = j; y < min(j + B_size, N); y += U_size){
                    C[i][y + 0] += A[i][k] * B[k][y + 0];
                    C[i][y + 1] += A[i][k] * B[k][y + 1];
                }
            }
        } // Write block (i, j) of C to main memory
    }
} // Two parameters: B_size and U_size
```

# An Autotuning Example: Loop *Blocking* and *Unrolling* for Matrix Multiplication

## Optimizing Matrix Multiplication

How to **restructure memory accesses** in loops to increase throughput by leveraging **cache locality**?

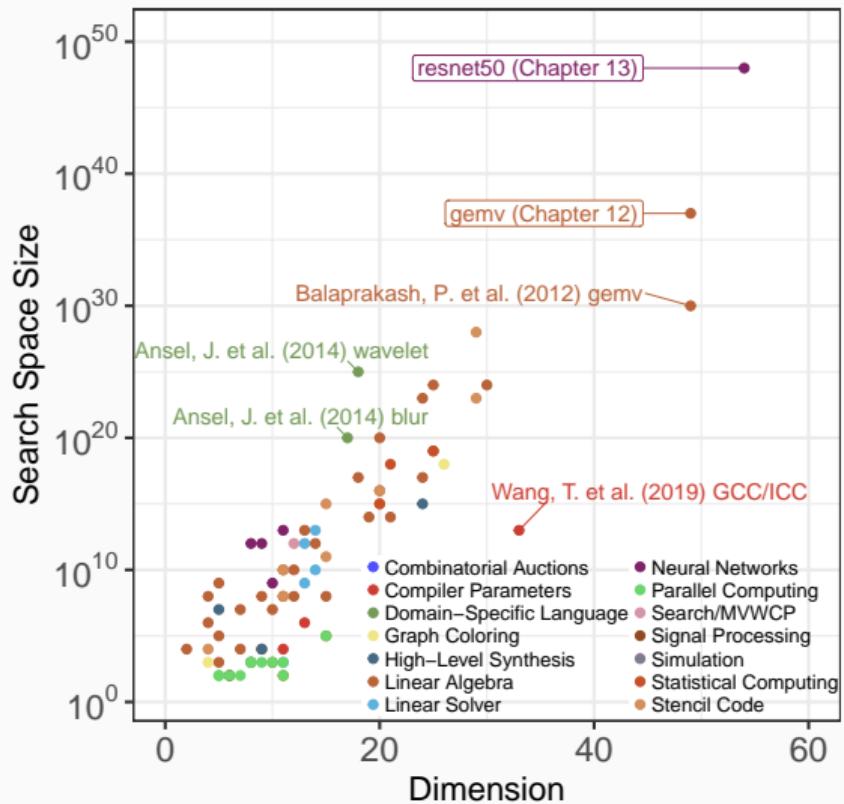
## Resulting Search Space



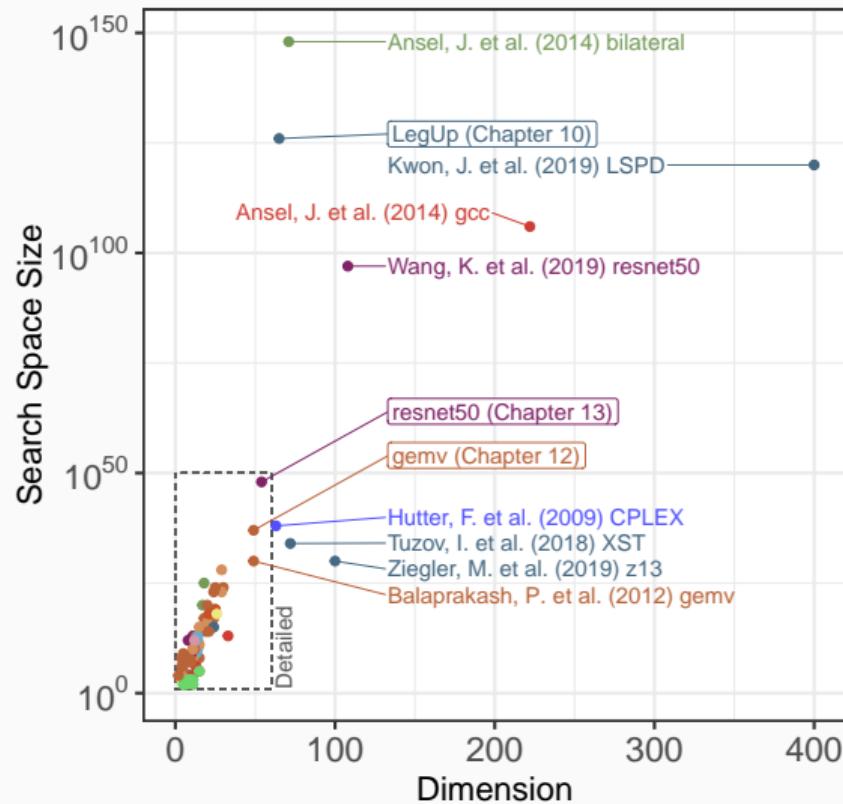
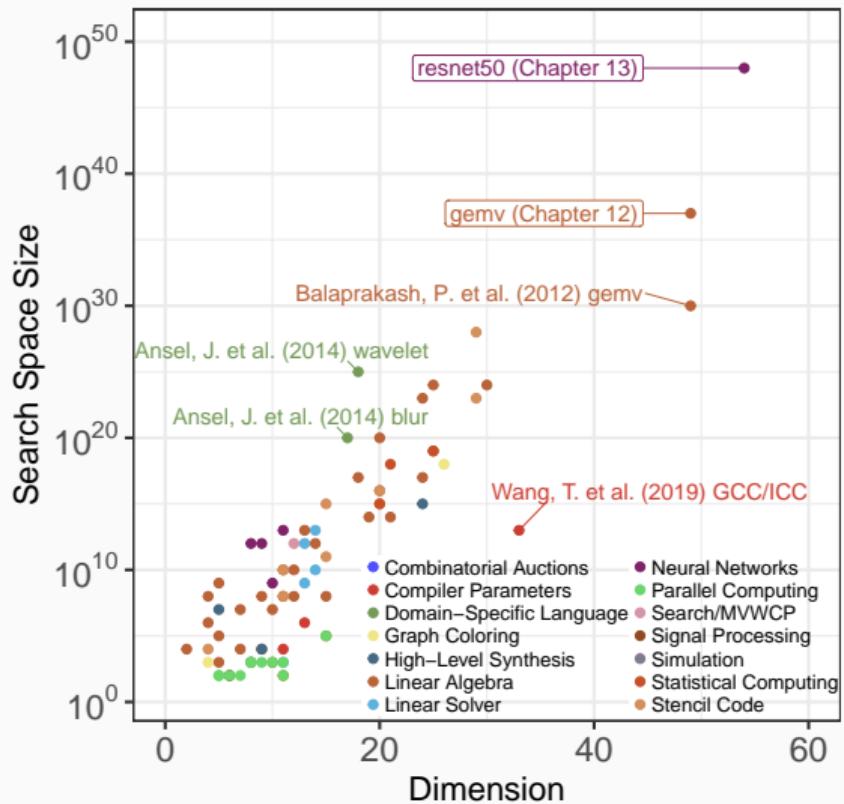
Seymour et al. (2008)

```
int N = 256;
int B_size = 4; int U_size = 2;
float A[N][N], B[N][N], C[N][N];
int i, j, k, x, y;
// Initialize A, B, C
for(i = 0; i < N; i += B_size){
    for(j = 0; j < N; j += B_size){
        // Load block (i, j) of C to fast memory
        for(k = 0; k < N; k++){
            // Load block (i, k) of A to fast memory
            // Load block (k, j) of B to fast memory
            for(x = i; x < min(i + B_size, N); x++){
                for(y = j; y < min(j + B_size, N); y += U_size){
                    C[i][y + 0] += A[i][k] * B[k][y + 0];
                    C[i][y + 1] += A[i][k] * B[k][y + 1];
                }
            }
        } // Write block (i, j) of C to main memory
    }
} // Two parameters: B_size and U_size
```

## Autotuning Problems in Other Domains: Dimension Becomes an Issue



# Autotuning Problems in Other Domains: Dimension Becomes an Issue



# Autotuning as an *Optimization or Learning* Problem

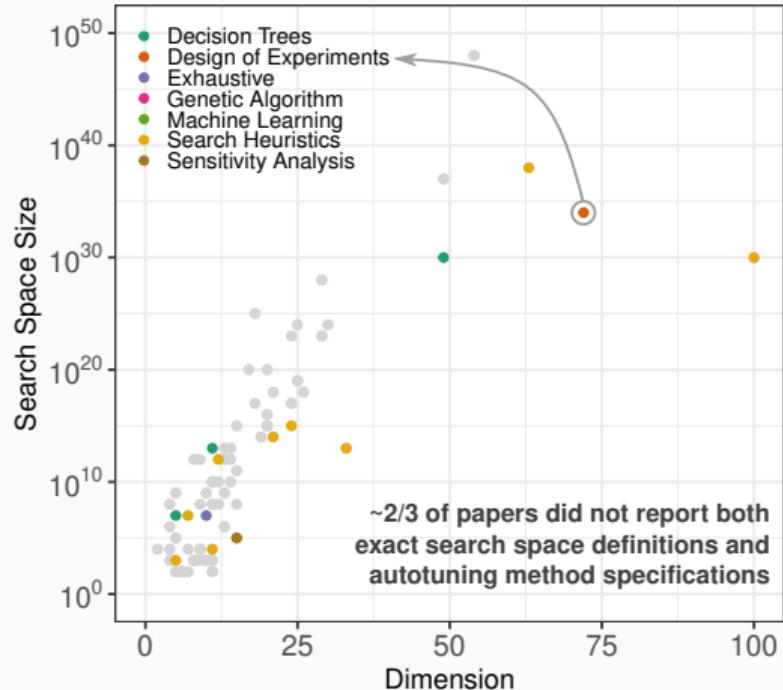
## Performance as a Function

Performance:  $f : \mathcal{X} \rightarrow \mathbb{R}$

- Parameters:  $\mathbf{x} = [x_1 \dots x_n]^T \in \mathcal{X}$
- Performance metric:  $y = f(\mathbf{x})$

To minimize  $f$ , we can adapt proven methods from other domains:

- Function minimization, Learning:** not necessarily parsimonious and transparent
- Design of Experiments:** can help, but not widely used for autotuning



# Toward Transparent and Parsimonious Autotuning

## Contributions of this Thesis

- Developing transparent and parsimonious autotuning methods based on the Design of Experiments
- Evaluating different autotuning methods in different HPC domains

### Transparent

- Use statistics to justify code optimization choices
- Learn about the search space

### Parsimonious

- Carefully choose which experiments to run
- Minimize  $f$  using as few measurements as possible

| Domain                   | Method  |
|--------------------------|---------|
| CUDA compiler parameters | F, D    |
| FPGA compiler parameters | F       |
| OpenCL Laplacian Kernel  | F, L, D |
| SPAPT Kernels            | L, D    |
| CNN Quantization         | L, D    |

F: Function Minimization, L: Learning,

D: Design of Experiments

# Toward Transparent and Parsimonious Autotuning

## Contributions of this Thesis

- Developing transparent and parsimonious autotuning methods based on the Design of Experiments
- Evaluating different autotuning methods in different HPC domains

### Transparent

- Use statistics to justify code optimization choices
- Learn about the search space

### Parsimonious

- Carefully choose which experiments to run
- Minimize  $f$  using as few measurements as possible

| Domain                   | Method  |
|--------------------------|---------|
| CUDA compiler parameters | F, D    |
| FPGA compiler parameters | F       |
| OpenCL Laplacian Kernel  | F, L, D |
| SPAPT Kernels            | L, D    |
| CNN Quantization         | L, D    |

F: Function Minimization, L: Learning,

D: Design of Experiments

: In this presentation

## Applying Methods for Function Minimization

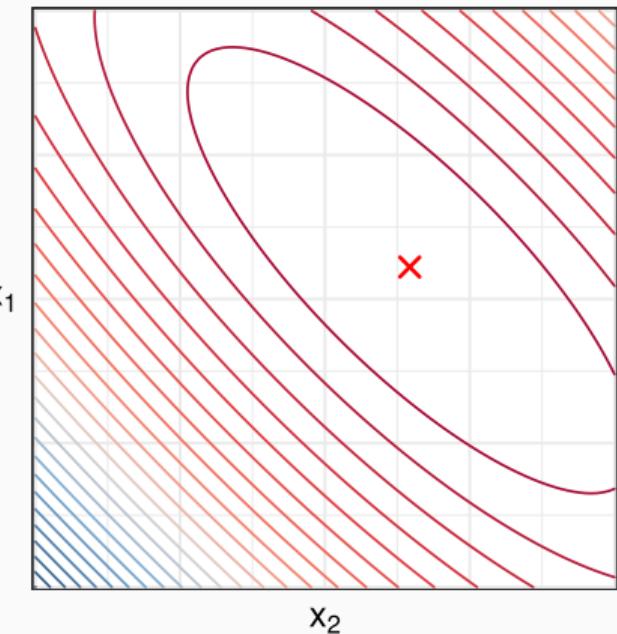
---

# Minimizing Functions using Derivatives and Heuristics

We know or can compute **information** about  $f$

- Directly measure **new**  $x_1, \dots, x_k, \dots, x_n$
- Search for the **global optimum**

$$f(\mathbf{x}) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2,$$
$$x_1, x_2 \in [-10, 10]$$

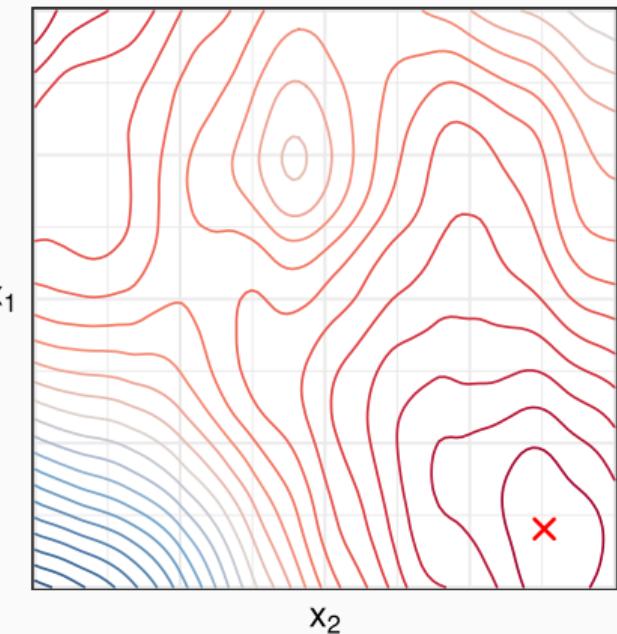


# Minimizing Functions using Derivatives and Heuristics

We know or can compute **information** about  $f$

- Directly measure **new**  $\mathbf{x}_1, \dots, \mathbf{x}_k, \dots, \mathbf{x}_n$
- Search for the **global optimum**
- Try to escape **local optima**

$$f(\mathbf{x}) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2 + z,$$
$$x_1, x_2 \in [-10, 10], \text{ and } z \sim \mathcal{N}(\mathbf{0}, K(\boldsymbol{\theta}, \mathbf{x}, \mathbf{x}'))$$



# Minimizing Functions using Derivatives and Heuristics

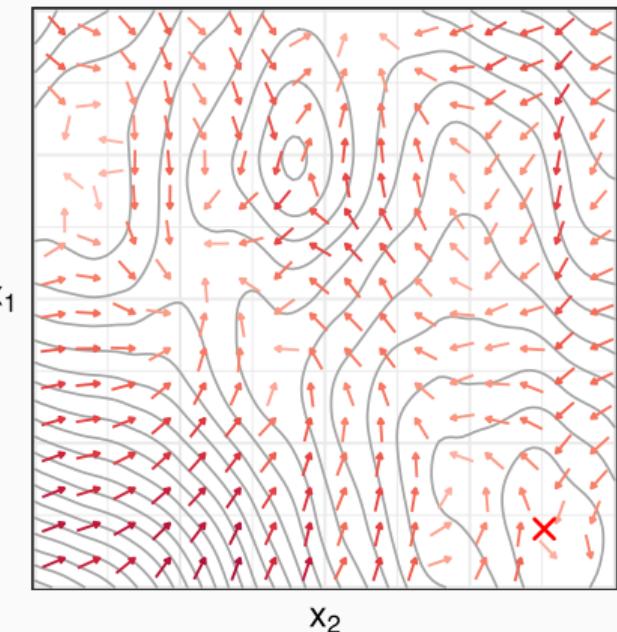
We know or can compute **information** about  $f$

- Directly measure **new**  $\mathbf{x}_1, \dots, \mathbf{x}_k, \dots, \mathbf{x}_n$
- Search for the **global optimum**
- Try to escape **local optima**

**Strong Hypothesis:** we can Compute **Derivatives**

$$\mathbf{x}_k = \mathbf{x}_{k-1} - \mathbf{H}f(\mathbf{x}_{k-1})\nabla f(\mathbf{x}_{k-1})$$

$$f(\mathbf{x}) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2 + z,$$
$$x_1, x_2 \in [-10, 10], \text{ and } z \sim \mathcal{N}(0, K(\theta, \mathbf{x}, \mathbf{x}'))$$



# Minimizing Functions using Derivatives and Heuristics

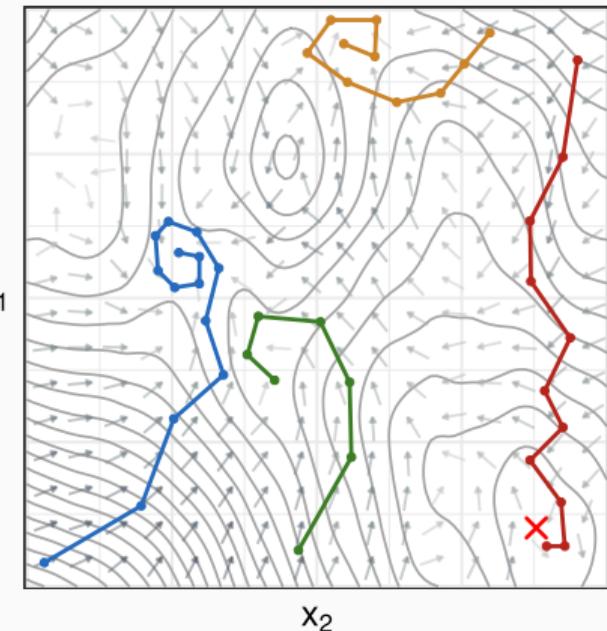
We know or can compute **information** about  $f$

- Directly measure **new**  $\mathbf{x}_1, \dots, \mathbf{x}_k, \dots, \mathbf{x}_n$
- Search for the **global optimum**
- Try to escape **local optima**

**Strong Hypothesis:** we can Compute **Derivatives**

- $\mathbf{x}_k = \mathbf{x}_{k-1} - \mathbf{H}f(\mathbf{x}_{k-1})\nabla f(\mathbf{x}_{k-1})$
- Move to the best point in a **neighborhood**

$$f(\mathbf{x}) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2 + z,$$
$$x_1, x_2 \in [-10, 10], \text{ and } z \sim \mathcal{N}(0, K(\theta, \mathbf{x}, \mathbf{x}'))$$



# Minimizing Functions using Derivatives and Heuristics

We know or can compute **information** about  $f$

- Directly measure **new**  $x_1, \dots, x_k, \dots, x_n$
- Search for the **global optimum**
- Try to escape **local optima**

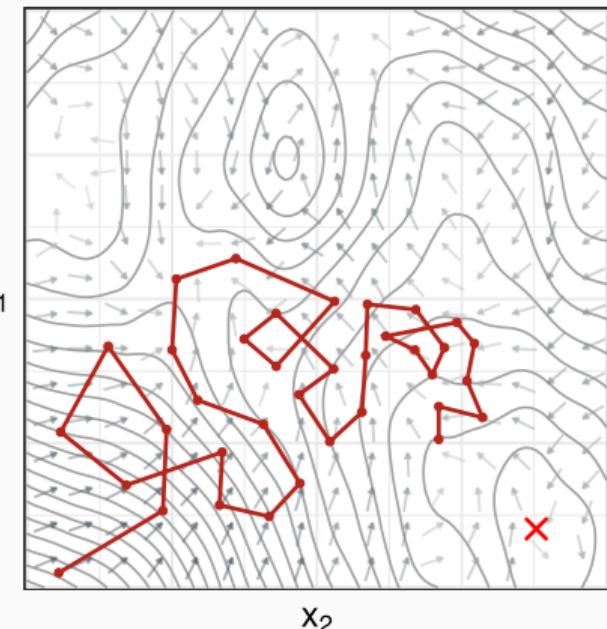
**Strong Hypothesis:** we can Compute **Derivatives**

- $x_k = x_{k-1} - Hf(x_{k-1})\nabla f(x_{k-1})$
- Move to the best point in a **neighborhood**

**Hard to State Hypotheses:** **Search Heuristics**

- **Random Walk**, Simulated Annealing, Genetic Algorithms, and many others

$$f(\mathbf{x}) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2 + z,$$
$$x_1, x_2 \in [-10, 10], \text{ and } z \sim N(0, K(\theta, \mathbf{x}, \mathbf{x}'))$$



# Minimizing Functions using Derivatives and Heuristics

We know or can compute **information** about  $f$

- Directly measure **new**  $x_1, \dots, x_k, \dots, x_n$
- Search for the **global optimum**
- Try to escape **local optima**

**Strong Hypothesis:** we can Compute **Derivatives**

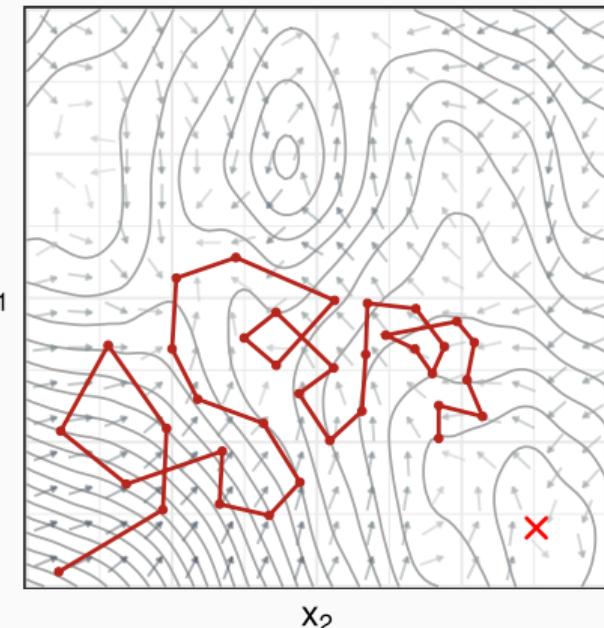
- $x_k = x_{k-1} - Hf(x_{k-1})\nabla f(x_{k-1})$
- Move to the best point in a **neighborhood**

**Hard to State Hypotheses:** **Search Heuristics**

- **Random Walk**, Simulated Annealing, Genetic Algorithms, and many others

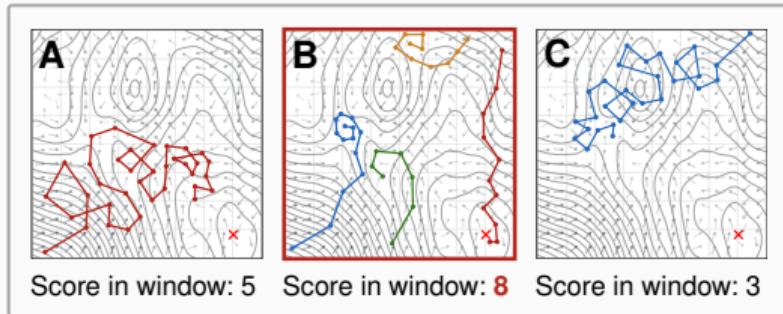
How to choose a method?

$$f(\mathbf{x}) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2 + z,$$
$$x_1, x_2 \in [-10, 10], \text{ and } z \sim N(0, K(\theta, \mathbf{x}, \mathbf{x}'))$$



# Choosing Methods from an Ensemble

## Example of an Ensemble of Methods



Best  $x$  from a shared results pool

## Methods in this Ensemble

- A, C: Simulated Annealing with different temperature, B: Gradient Descent

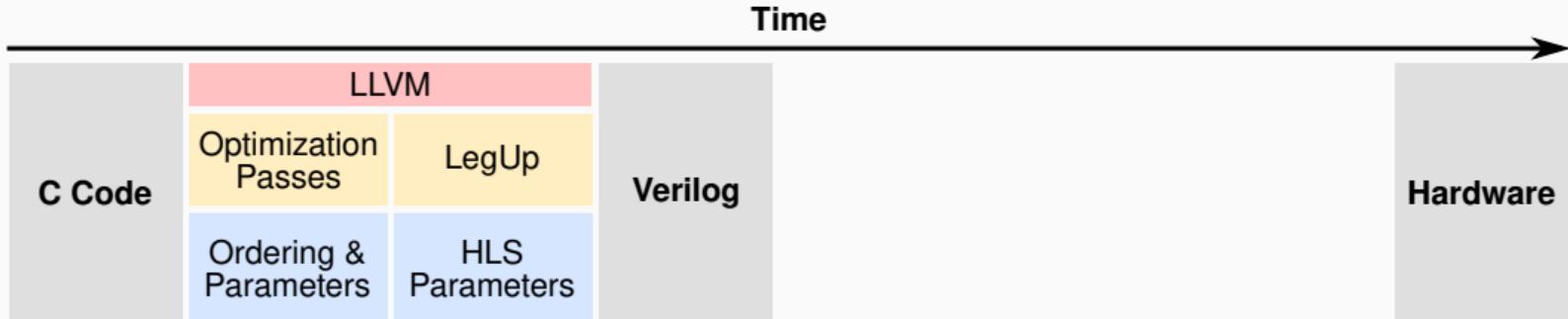
## Minimization of $f$ using OpenTuner

- Coordinated by a Multi-Armed Bandit (MAB) algorithm
- Methods perform measurements proportionally to their score
- Score: the number of times a method found the best  $x$  in a time window
- The best  $x$  over all methods is reported

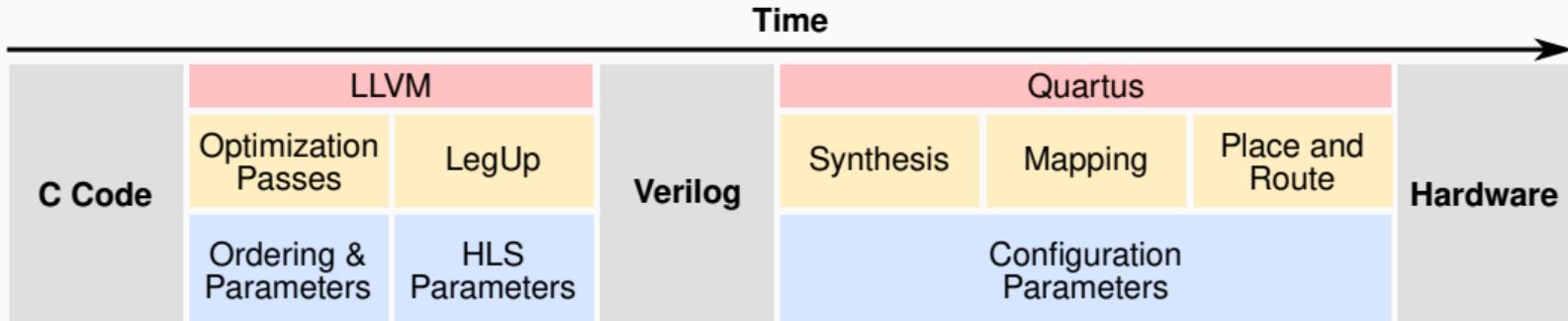
## Application: High-Level Synthesis for FPGAs



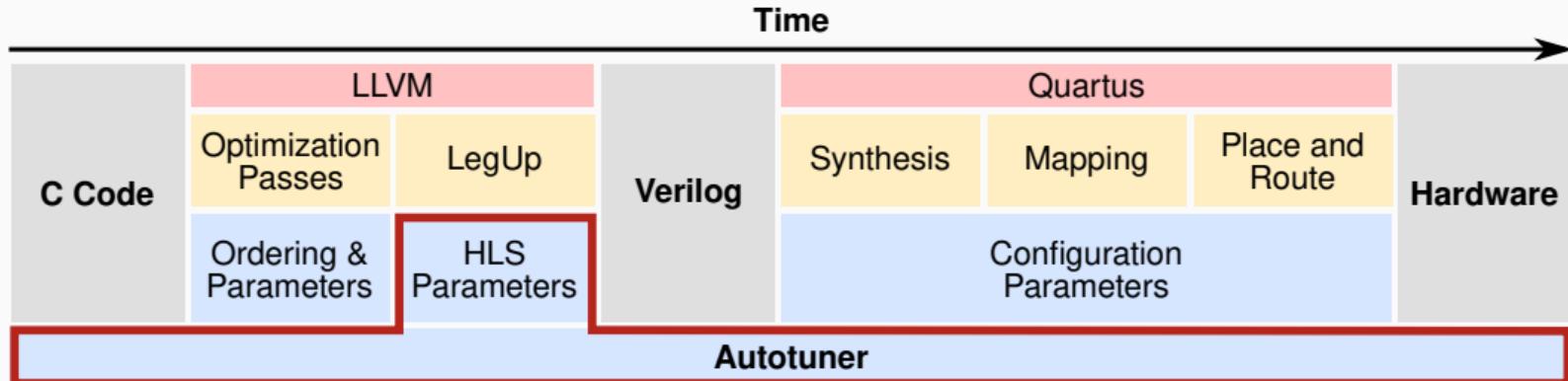
# Application: High-Level Synthesis for FPGAs



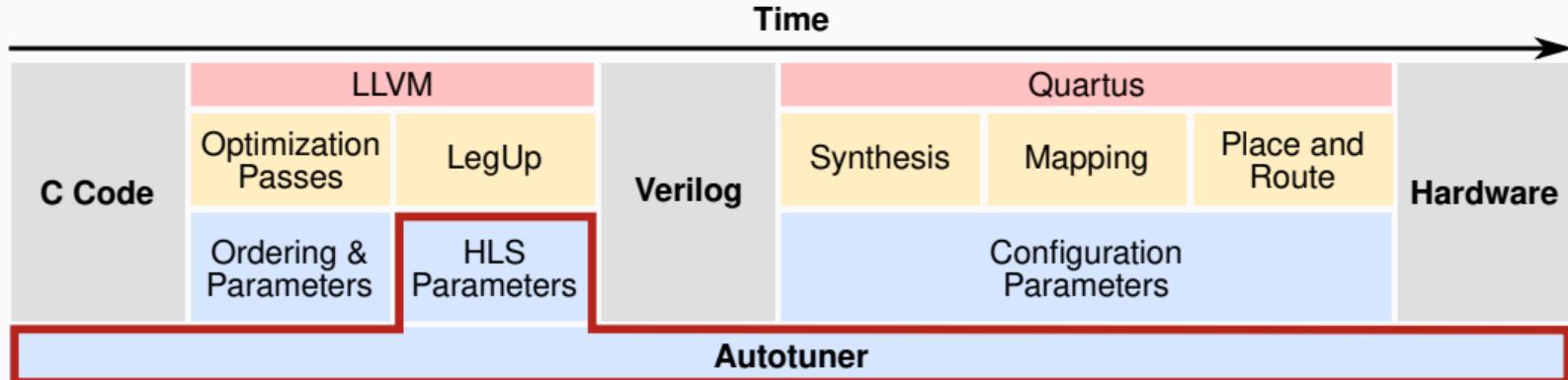
# Application: High-Level Synthesis for FPGAs



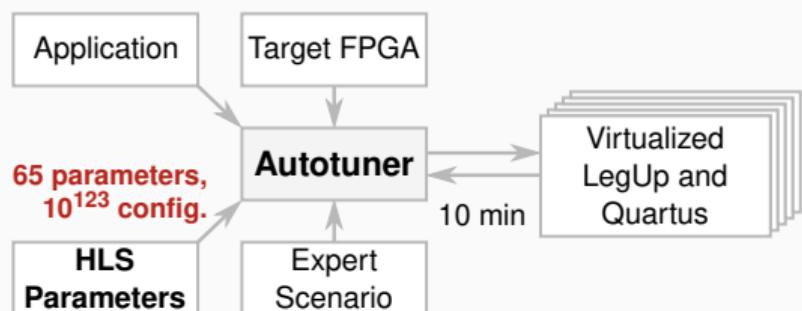
# Application: High-Level Synthesis for FPGAs



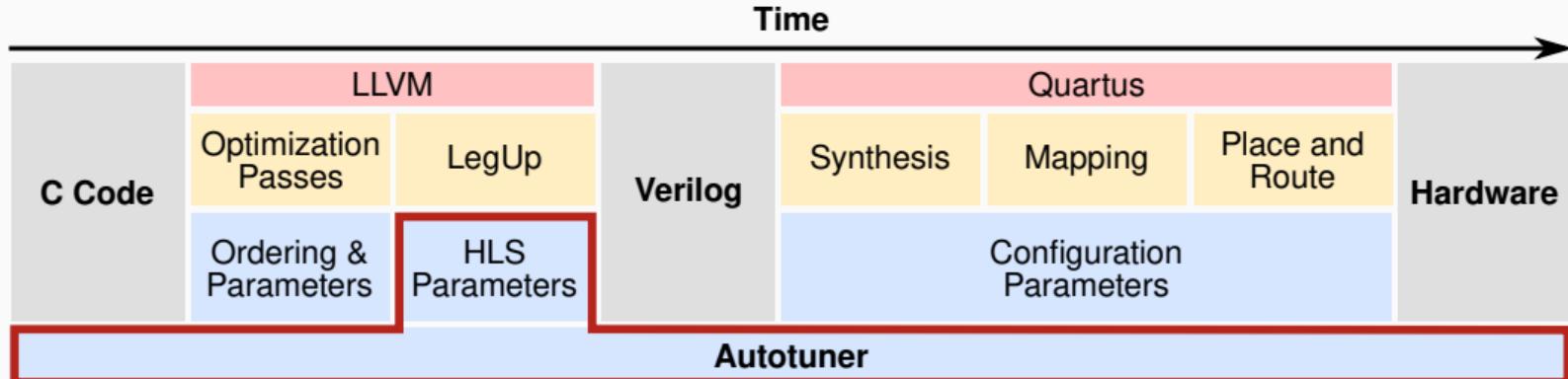
# Application: High-Level Synthesis for FPGAs



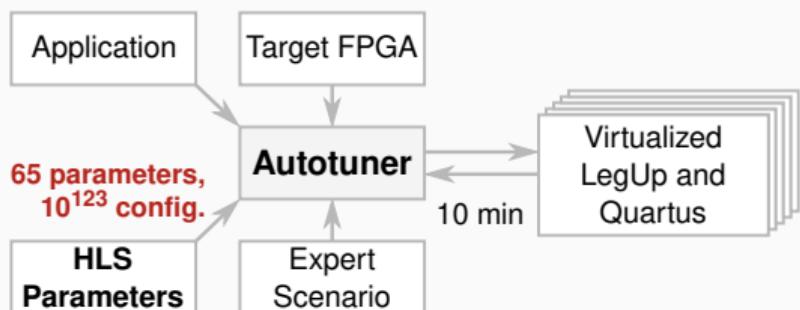
## Search Space



# Application: High-Level Synthesis for FPGAs



## Search Space



## Performance Metrics

- Weighted average of **8 hardware metrics**
- Metrics for the usage of registers, memory, DSP units, frequency and clock speed
- An **expert** devised weights for optimizing for area, latency, performance, or for multiple criteria

# Results

## Experimental Settings

- 11 problems
- Up to 300 measurements per problem
- Compared to optimized LegUp configurations for the target FPGA

## Improvements

- 10% improvement on weighted average
- 2 and 5 times improvements for some metrics and scenarios

## Implementation in OpenTuner

- Ensemble with Simulated Annealing, Genetic Algorithms, and Nelder-Mead

Performance: darker blues are better

|          | aes | 0.82 | 0.75 | 1.00 | 1.00 | 0.92   | 1.00   | 1.05 | 1.00 | 0.63 |
|----------|-----|------|------|------|------|--------|--------|------|------|------|
| adpcm    |     | 0.84 | 0.94 | 1.17 | 1.00 | 0.96   | 1.00   | 0.94 | 0.69 | 0.74 |
| sha      |     | 0.92 | 1.00 | 1.06 | 1.00 | 0.92   | 1.00   | 0.88 | 1.00 | 0.96 |
| motion   |     | 0.99 | 1.00 | 1.00 | 1.00 | 1.01   | 1.00   | 1.00 | 1.00 | 0.98 |
| mips     |     | 0.90 | 1.00 | 1.06 | 1.00 | 0.93   | 1.00   | 1.03 | 0.83 | 0.80 |
| gsm      |     | 0.95 | 0.92 | 1.00 | 1.00 | 0.95   | 1.00   | 0.90 | 1.00 | 1.02 |
| dfsinv   |     | 0.87 | 1.11 | 1.06 | 1.00 | 1.27   | 1.00   | 1.25 | 0.53 | 0.56 |
| dfmul    |     | 0.77 | 1.00 | 1.17 | 0.83 | 0.85   | 0.83   | 1.04 | 0.21 | 0.70 |
| dfdiv    |     | 0.81 | 1.00 | 1.06 | 1.00 | 1.03   | 1.00   | 1.25 | 0.50 | 0.59 |
| dfadd    |     | 0.97 | 1.00 | 1.00 | 1.00 | 0.97   | 1.00   | 1.00 | 1.00 | 0.94 |
| blowfish |     | 0.94 | 1.00 | 1.00 | 1.00 | 0.98   | 1.00   | 0.91 | 1.00 | 0.95 |
|          | WNS | LUTs | Pins | BRAM | Regs | Blocks | Cycles | DSP  | FMax |      |

Weighted Average for all Scenarios

|          | 0.94     | 0.96 | 0.82        | 0.83         |
|----------|----------|------|-------------|--------------|
| adpcm    | 0.84     | 0.83 | 0.84        | 0.76         |
| sha      | 0.98     | 0.96 | 0.92        | 0.88         |
| motion   | 0.98     | 0.97 | 0.99        | 0.98         |
| mips     | 0.95     | 0.91 | 0.90        | 0.95         |
| gsm      | 0.95     | 0.89 | 0.95        | 0.92         |
| dfsinv   | 0.93     | 0.94 | 0.87        | 0.97         |
| dfmul    | 0.85     | 0.82 | 0.77        | 0.86         |
| dfdiv    | 0.84     | 0.77 | 0.81        | 0.82         |
| dfadd    | 1.00     | 0.99 | 0.97        | 0.98         |
| blowfish | 0.99     | 0.99 | 0.94        | 0.96         |
| Average  | 0.93     | 0.91 | 0.89        | 0.90         |
|          | Balanced | Area | Performance | Perf. & Lat. |

# Discussion

## Autotuning with Heuristics

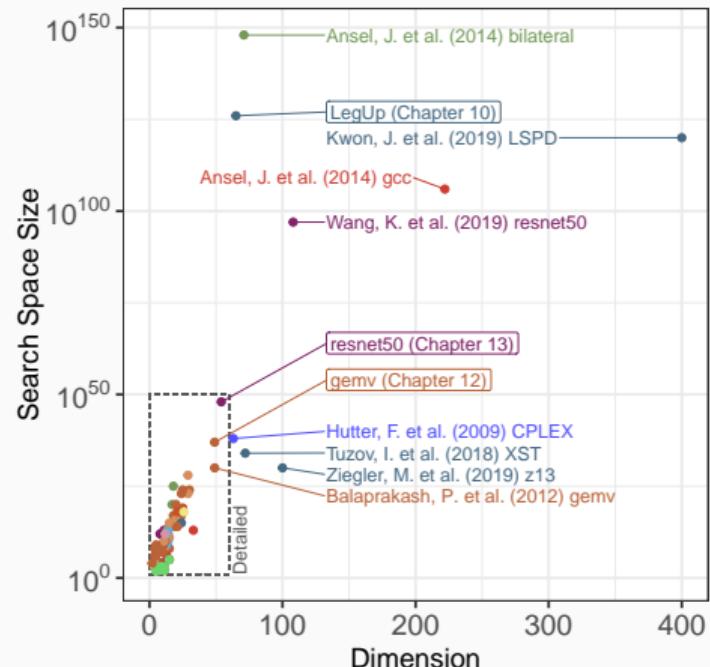
- Lack of **structured exploration** prevented statistical analyses and interpretation

## Needle in a Haystack

- Global optimum in  $10^{123}$  configurations?
- Are there **better configurations** to find?
- For how long should we continue **exploring**?

## Proprietary Software Stack for FPGAs

- LegUp is now proprietary software



# Discussion

## Autotuning with Heuristics

- Lack of **structured exploration** prevented statistical analyses and interpretation

## Needle in a Haystack

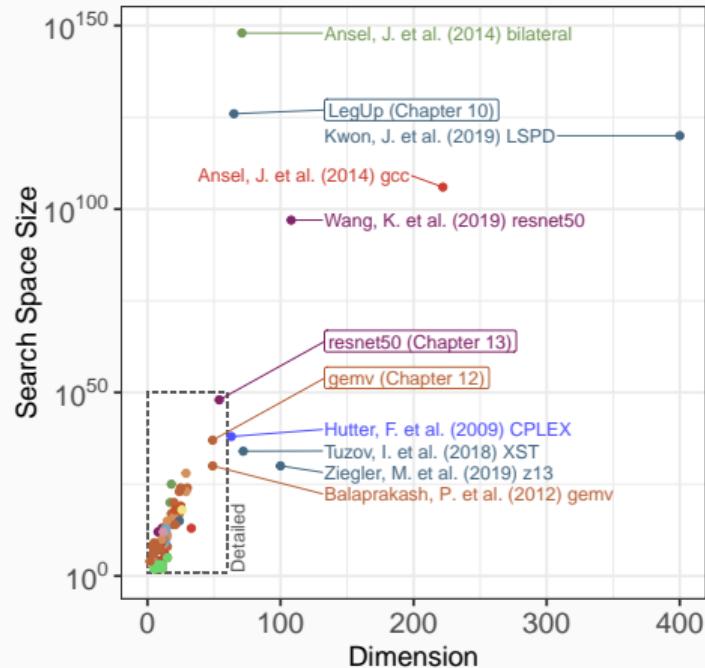
- Global optimum in  $10^{123}$  configurations?
- Are there **better configurations** to find?
- For how long should we continue **exploring**?

## Proprietary Software Stack for FPGAs

- LegUp is now proprietary software

## Sequential Design of Experiments

Structure explorations using modeling hypotheses to guide sampling and optimization



## **Applying Sequential Design of Experiments**

---

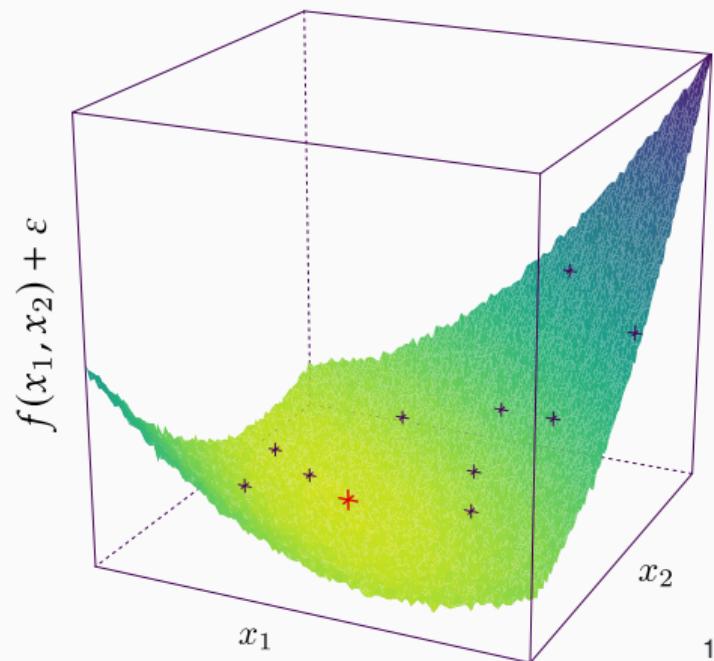
# Search Space Hypotheses with Linear Models

## Learning: Building Surrogates

- $f : \mathcal{X} \rightarrow \mathbb{R}$
- Model:  $f(\mathbf{x}) = \mathbf{x}^\top \boldsymbol{\theta} + \varepsilon$ , with  $\varepsilon \sim \mathcal{N}(0, \sigma^2)$
- Data:  $(\mathbf{x}_k, y_k = f(\mathbf{x}_k))$
- $\mathbf{x}_{1,\dots,n}$  in a **given** design  $\mathbf{X}$

## 10 Measurements of Booth's Function

$$f(\mathbf{x}) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2 + \varepsilon$$



# Search Space Hypotheses with Linear Models

## Learning: Building Surrogates

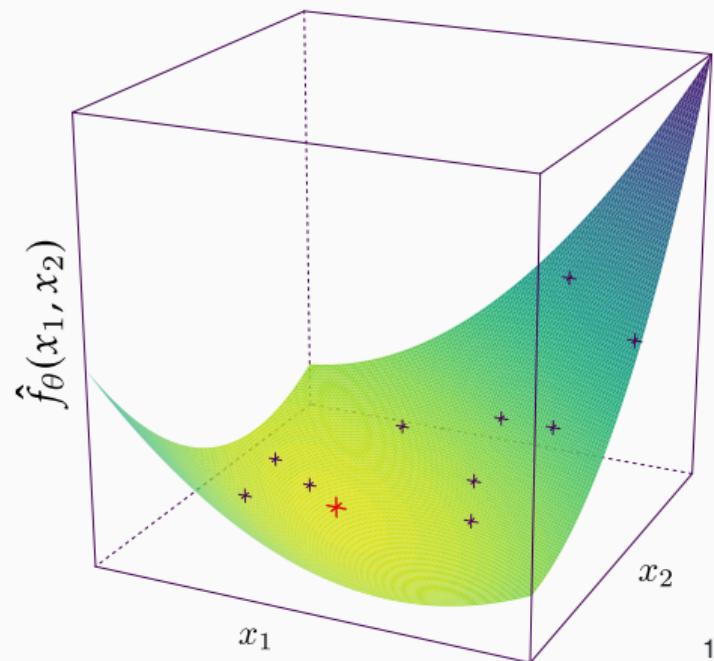
- $f : \mathcal{X} \rightarrow \mathbb{R}$
- Model:  $f(\mathbf{x}) = \mathbf{x}^\top \boldsymbol{\theta} + \varepsilon$ , with  $\varepsilon \sim \mathcal{N}(0, \sigma^2)$
- Data:  $(\mathbf{x}_k, y_k = f(\mathbf{x}_k))$
- $\mathbf{x}_{1,\dots,n}$  in a **given** design  $\mathbf{X}$

Minimize a **Surrogate** instead of  $f$

- Surrogate:  $\hat{f}_\theta(\mathbf{x}') = \mathbf{x}'^\top \hat{\boldsymbol{\theta}}$
- Estimator:  $\hat{\boldsymbol{\theta}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$

## 10 Measurements of Booth's Function

$$\hat{f}_\theta(\mathbf{x}) = \hat{\theta}_0 + \hat{\theta}_1 x_1 + \hat{\theta}_2 x_2 + \hat{\theta}_3 x_1^2 + \hat{\theta}_4 x_2^2 + \hat{\theta}_5 x_1 x_2$$



# Search Space Hypotheses with Linear Models

## Learning: Building Surrogates

- $f : \mathcal{X} \rightarrow \mathbb{R}$
- Model:  $f(\mathbf{x}) = \mathbf{x}^\top \boldsymbol{\theta} + \varepsilon$ , with  $\varepsilon \sim \mathcal{N}(0, \sigma^2)$
- Data:  $(\mathbf{x}_k, y_k = f(\mathbf{x}_k))$
- $\mathbf{x}_{1,\dots,n}$  in a **given** design  $\mathbf{X}$

Minimize a **Surrogate** instead of  $f$

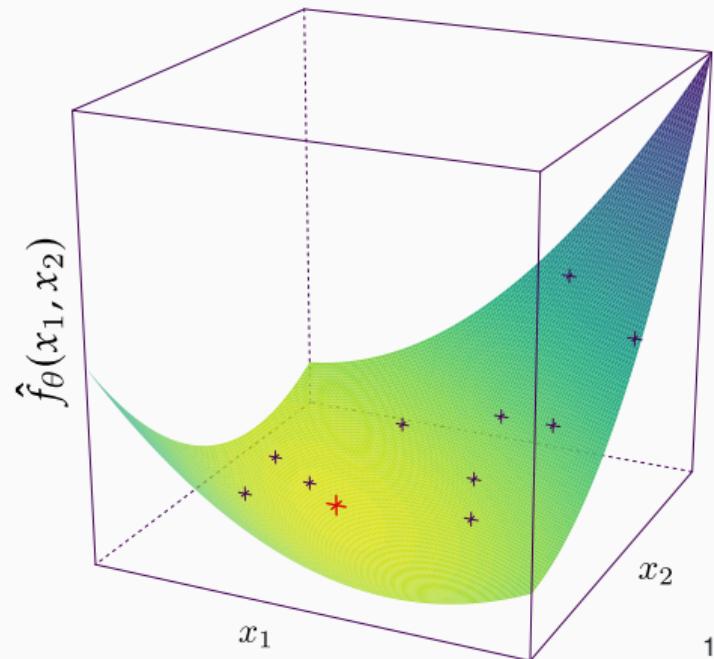
- Surrogate:  $\hat{f}_\theta(\mathbf{x}') = \mathbf{x}'^\top \hat{\boldsymbol{\theta}}$
- Estimator:  $\hat{\boldsymbol{\theta}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$

Variance of  $\hat{\boldsymbol{\theta}}$  is independent of  $\mathbf{y}$

$$\text{Var}(\hat{\boldsymbol{\theta}}) = (\mathbf{X}^\top \mathbf{X})^{-1} \sigma^2$$

## 10 Measurements of Booth's Function

$$\hat{f}_\theta(\mathbf{x}) = \hat{\theta}_0 + \hat{\theta}_1 x_1 + \hat{\theta}_2 x_2 + \hat{\theta}_3 x_1^2 + \hat{\theta}_4 x_2^2 + \hat{\theta}_5 x_1 x_2$$



# Design of Experiments

## Choosing the Design X

- Minimizes  $\text{Var}(\hat{\theta})$
- Decreases number of experiments in X
- Enables testing hypotheses

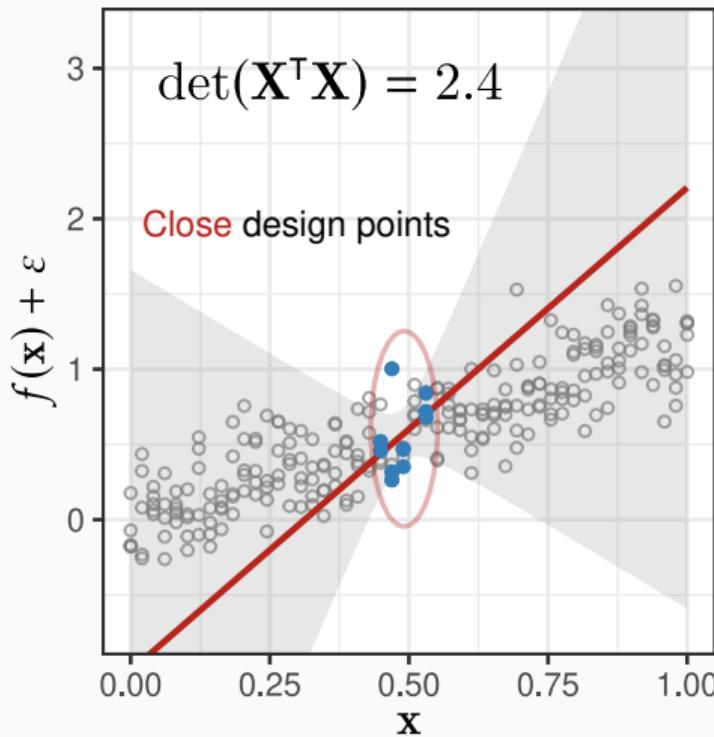
## Components

- $X_{n \times p}$ : design matrix
- $x_{1 \times n} \in X$ : factor columns
- $x_1, \dots, x_p \in x$ : chosen factor levels

## Examples

- Factorial designs, screening, Latin Hypercube and low-discrepancy sampling, **optimal design**

## Distance of Experiments Impacts $\text{Var}(\hat{\theta})$



# Design of Experiments

## Choosing the Design X

- Minimizes  $\text{Var}(\hat{\theta})$
- Decreases number of experiments in X
- Enables testing hypotheses

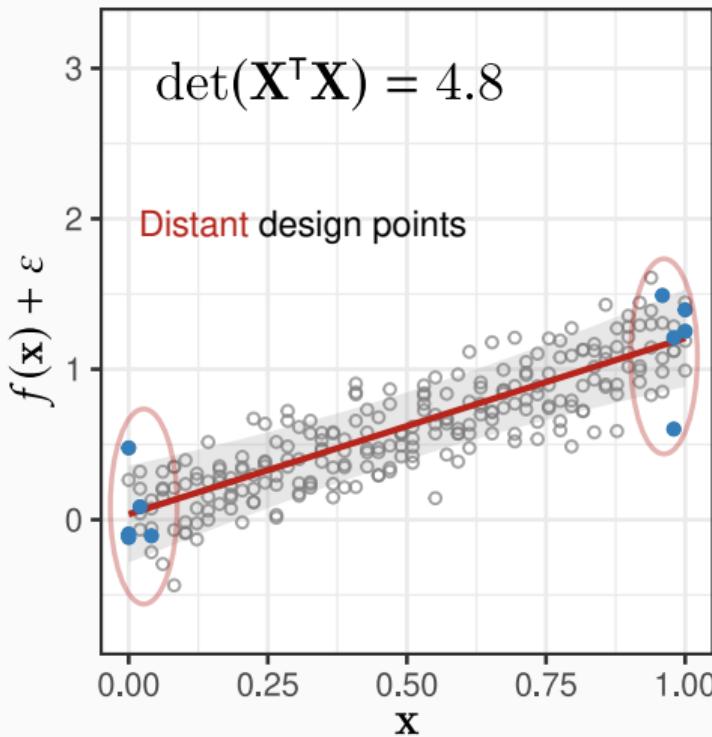
## Components

- $X_{n \times p}$ : design matrix
- $x_{1 \times n} \in X$ : factor columns
- $x_1, \dots, x_p \in x$ : chosen factor levels

## Examples

- Factorial designs, screening, Latin Hypercube and low-discrepancy sampling, optimal design

## Distance of Experiments Impacts $\text{Var}(\hat{\theta})$



## Optimal Design: Parsimony

## Sampling with Different Models

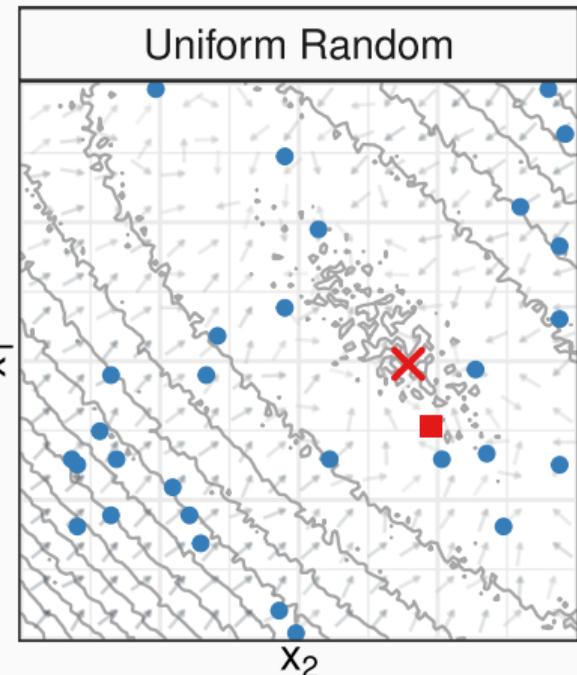
$$f(\mathbf{x}) = (x_1 + 2x_2 - 7)^2 + (2x_1 + x_2 - 5)^2 + \varepsilon$$

### Designs

- Building surrogates within a **constrained budget**
- Exploiting known search space structure
- **Testing** modeling hypotheses

### Maximizing $\det(\mathbf{X}^T \mathbf{X})$ by Swapping Rows

- Requires an initial **model**
- Choose best rows for  $\mathbf{X}$  from a **large set**
- $D(\mathbf{X}) \propto \det(\mathbf{X}^T \mathbf{X})$



✖: global optimum, ■: best point found,  
●: measurements

## Optimal Design: Parsimony

### Designs

- Building surrogates within a **constrained budget**
- Exploiting known search space structure
- **Testing** modeling hypotheses

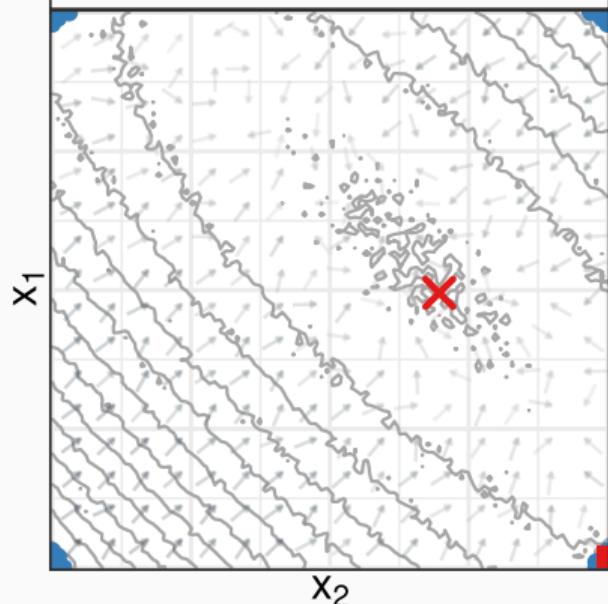
### Maximizing $\det(X^T X)$ by Swapping Rows

- Requires an initial **model**
- Choose best rows for  $X$  from a **large set**
- $D(X) \propto \det(X^T X)$

## Sampling with Different Models

$$\hat{f}_\theta(\mathbf{x}) = \hat{\theta}_0 + \hat{\theta}_1 x_1 + \hat{\theta}_2 x_2$$

### Optimal Design: Linear



✖: global optimum, ■: best point found,  
●: measurements

## Optimal Design: Parsimony

### Designs

- Building surrogates within a **constrained budget**
- Exploiting known search space structure
- **Testing** modeling hypotheses

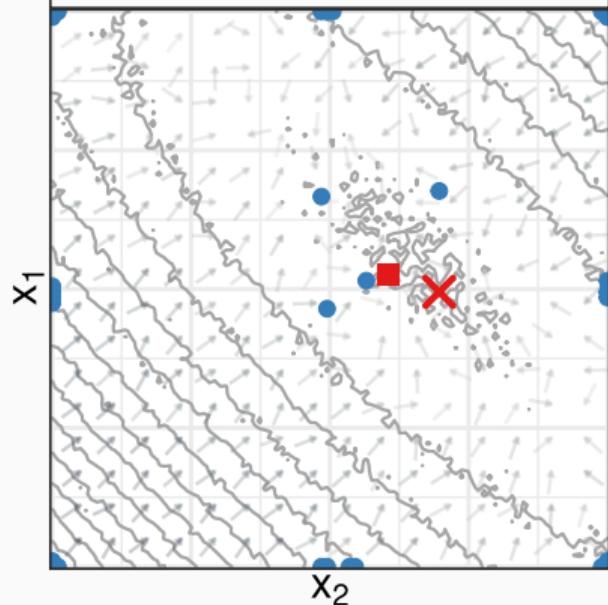
### Maximizing $\det(X^T X)$ by Swapping Rows

- Requires an initial **model**
- Choose best rows for  $X$  from a **large set**
- $D(X) \propto \det(X^T X)$

## Sampling with Different Models

$$\hat{f}_\theta(\mathbf{x}) = \hat{\theta}_0 + \hat{\theta}_1 x_1 + \hat{\theta}_2 x_2 + \hat{\theta}_3 x_1^2 + \hat{\theta}_4 x_2^2$$

### Optimal Design: Quadratic



$\times$ : global optimum,  $\blacksquare$ : best point found,  
●: measurements

## Optimal Design: Parsimony

### Designs

- Building surrogates within a **constrained budget**
- Exploiting known search space structure
- **Testing** modeling hypotheses

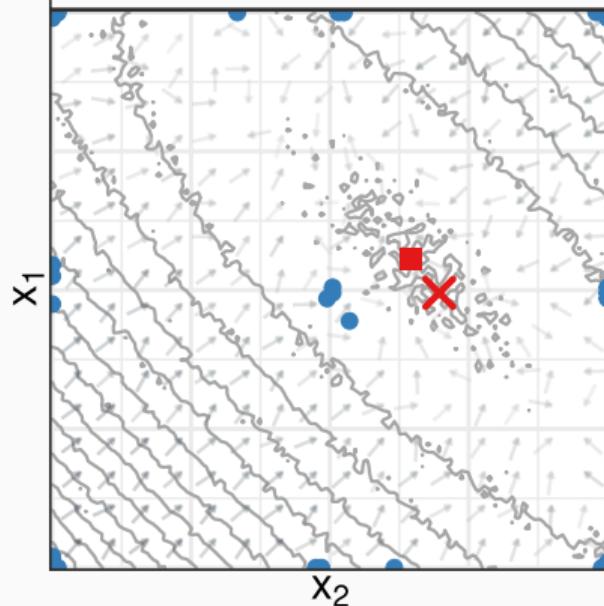
### Maximizing $\det(X^T X)$ by Swapping Rows

- Requires an initial **model**
- Choose best rows for  $X$  from a **large set**
- $D(X) \propto \det(X^T X)$

## Sampling with Different Models

$$\hat{f}_\theta(\mathbf{x}) = \hat{\theta}_0 + \hat{\theta}_1 x_1 + \hat{\theta}_2 x_2 + \hat{\theta}_3 x_1^2 + \hat{\theta}_4 x_2^2 + \hat{\theta}_5 x_1 x_2$$

### Optimal Design: Interactions



✗: global optimum, ■: best point found,

●: measurements

## Optimal Design: Parsimony

### Designs

- Building surrogates within a **constrained budget**
- Exploiting known search space structure
- **Testing** modeling hypotheses

### Maximizing $\det(X^T X)$ by Swapping Rows

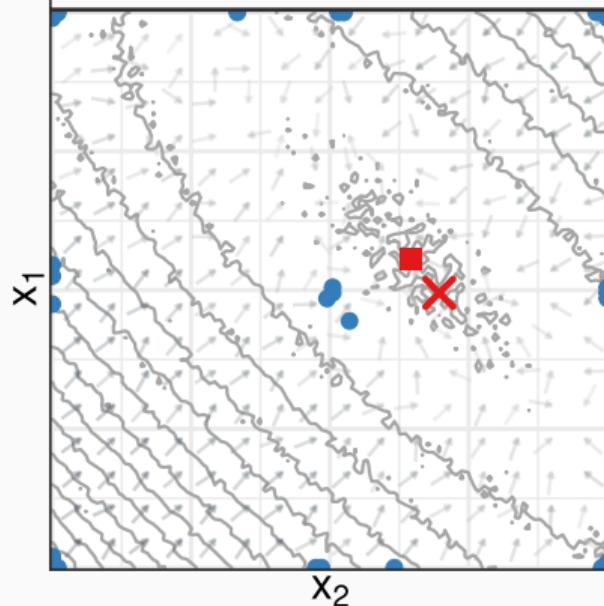
- Requires an initial **model**
- Choose best rows for  $X$  from a **large set**
- $D(X) \propto \det(X^T X)$

Best design is **independent of measurements**

## Sampling with Different Models

$$\hat{f}_\theta(\mathbf{x}) = \hat{\theta}_0 + \hat{\theta}_1 x_1 + \hat{\theta}_2 x_2 + \hat{\theta}_3 x_1^2 + \hat{\theta}_4 x_2^2 + \hat{\theta}_5 x_1 x_2$$

### Optimal Design: Interactions



✗: global optimum, ■: best point found,

●: measurements

# Interpreting Significance with Analysis of Variance

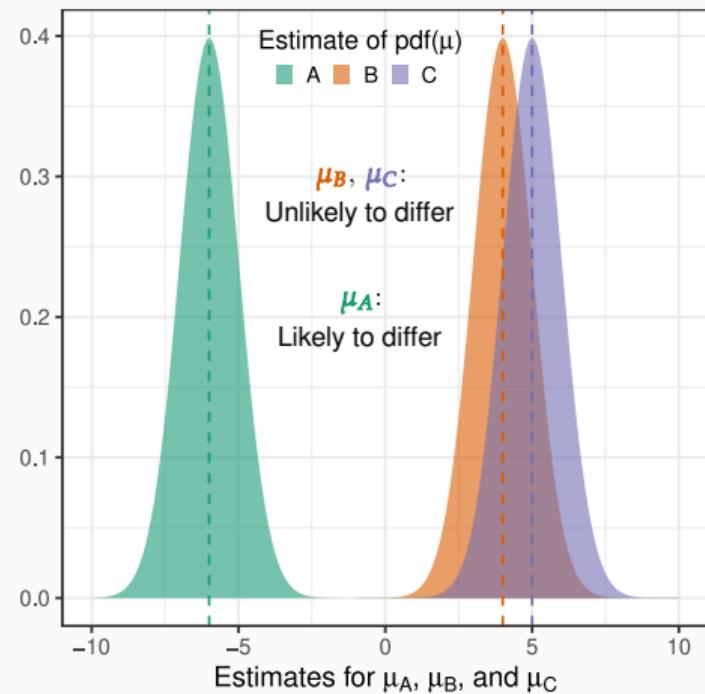
## One-Way ANOVA for Levels A, B, C

### Analysis of Variance (ANOVA)

- Identify which factors and levels are **significant**

### Steps

- Group observations by factor and factor levels
- Estimate distributions for each group mean  $\mu$
- Run **F-tests** for significance of differences between means



# Interpreting Significance with Analysis of Variance

## One-Way ANOVA for Levels A, B, C

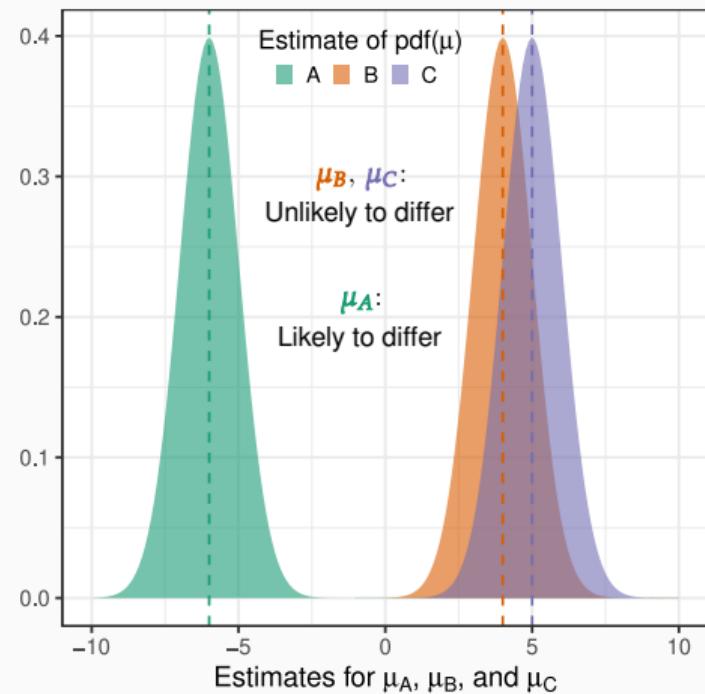
### Analysis of Variance (ANOVA)

- Identify which factors and levels are **significant**

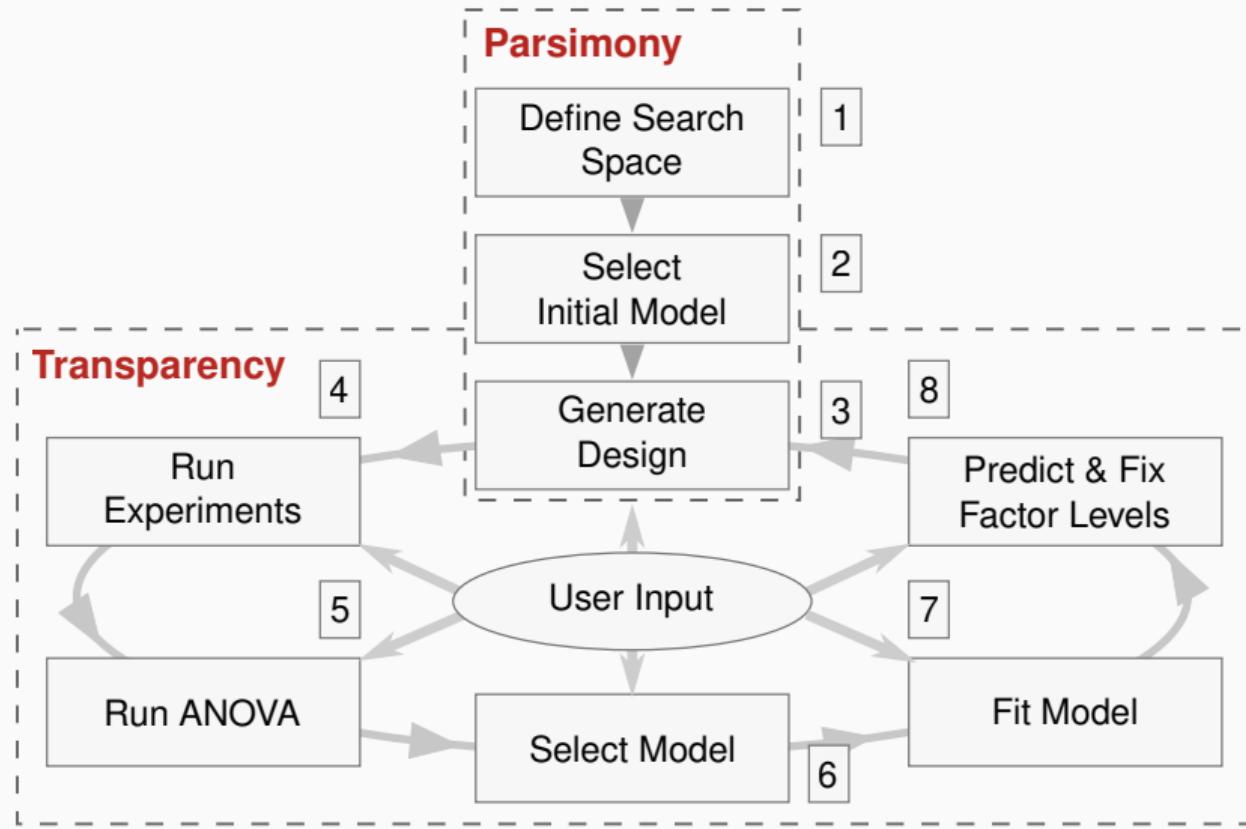
### Steps

- Group observations by factor and factor levels
- Estimate distributions for each group mean  $\mu$
- Run **F-tests** for significance of differences between means

Enables **refining** initial hypotheses



# A Transparent and Parsimonious Approach to Autotuning



# Application: OpenCL GPU Laplacian Kernel

## Edge Detection with the Laplacian



## Search Space with $10^4$ Valid Configurations

| Factor                    | Levels               | Short Description                |
|---------------------------|----------------------|----------------------------------|
| <i>vector_length</i>      | $2^0, \dots, 2^4$    | Size of vectors                  |
| <i>load_overlap</i>       | <i>true, false</i>   | Load overlaps in vectorization   |
| <i>temporary_size</i>     | 2, 4                 | Byte size of temporary data      |
| <i>elements_number</i>    | $1, \dots, 24$       | Size of equal data splits        |
| <i>y_component_number</i> | $1, \dots, 6$        | Loop tile size                   |
| <i>threads_number</i>     | $2^5, \dots, 2^{10}$ | Size of thread groups            |
| <i>lws_y</i>              | $2^0, \dots, 2^{10}$ | Block size in <i>y</i> dimension |

## Performance Metric and Starting Model

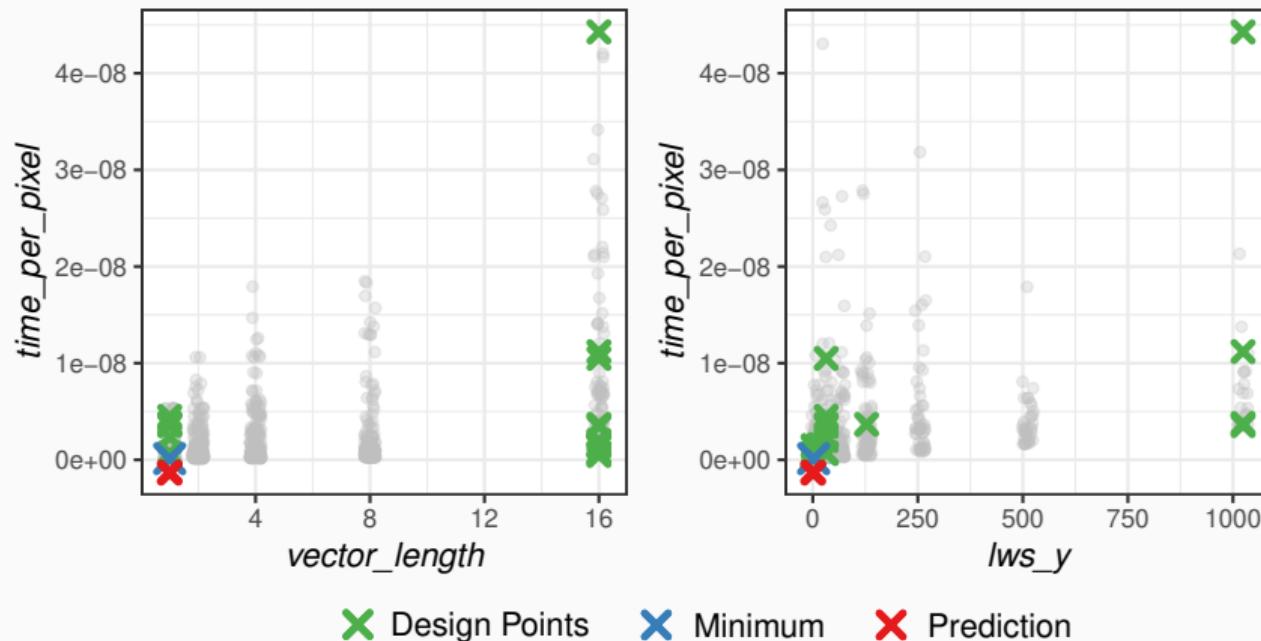
$$\begin{aligned} \text{time\_per\_pixel} \sim & y\_component\_number + \frac{1}{y\_component\_number} + \\ & temporary\_size + vector\_length + load\_overlap + \\ & lws\_y + \frac{1}{lws\_y} + elements\_number + \frac{1}{elements\_number} + \\ & threads\_number + \frac{1}{threads\_number} \end{aligned}$$

## The OpenCL Kernel

- Highly optimized
- Efficiently parametrized
- Generated by BOAST
- Completely evaluated previously

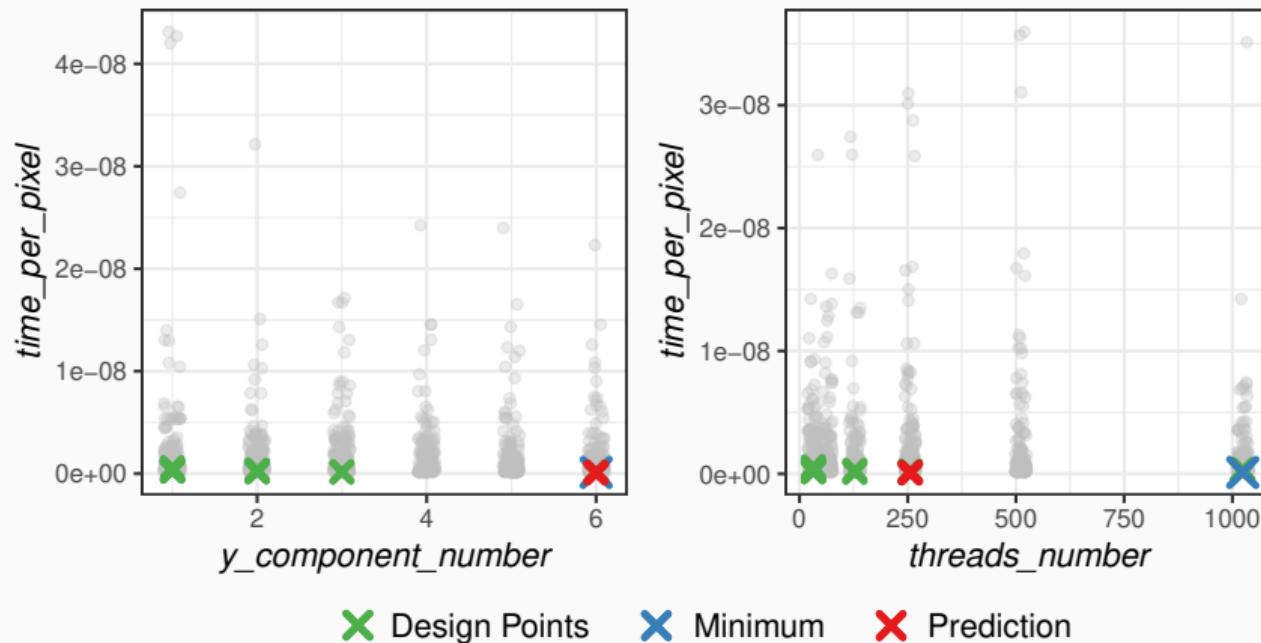
## Sequential Approach to Optimization: Refining the Model

### Step 1 Fix $\text{vector}$ and $\text{lws\_y}$



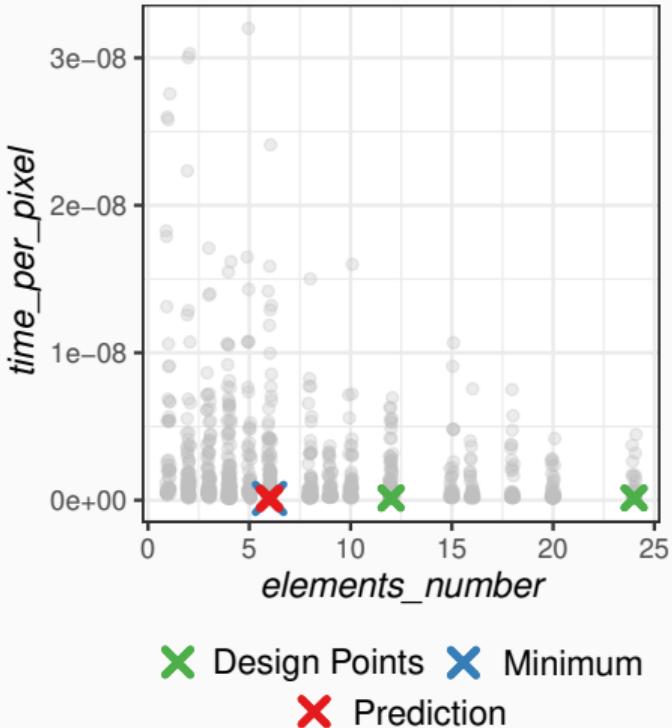
## Sequential Approach to Optimization: Refining the Model

### Step 2 Fix $y\_cmp.$ and $threads$



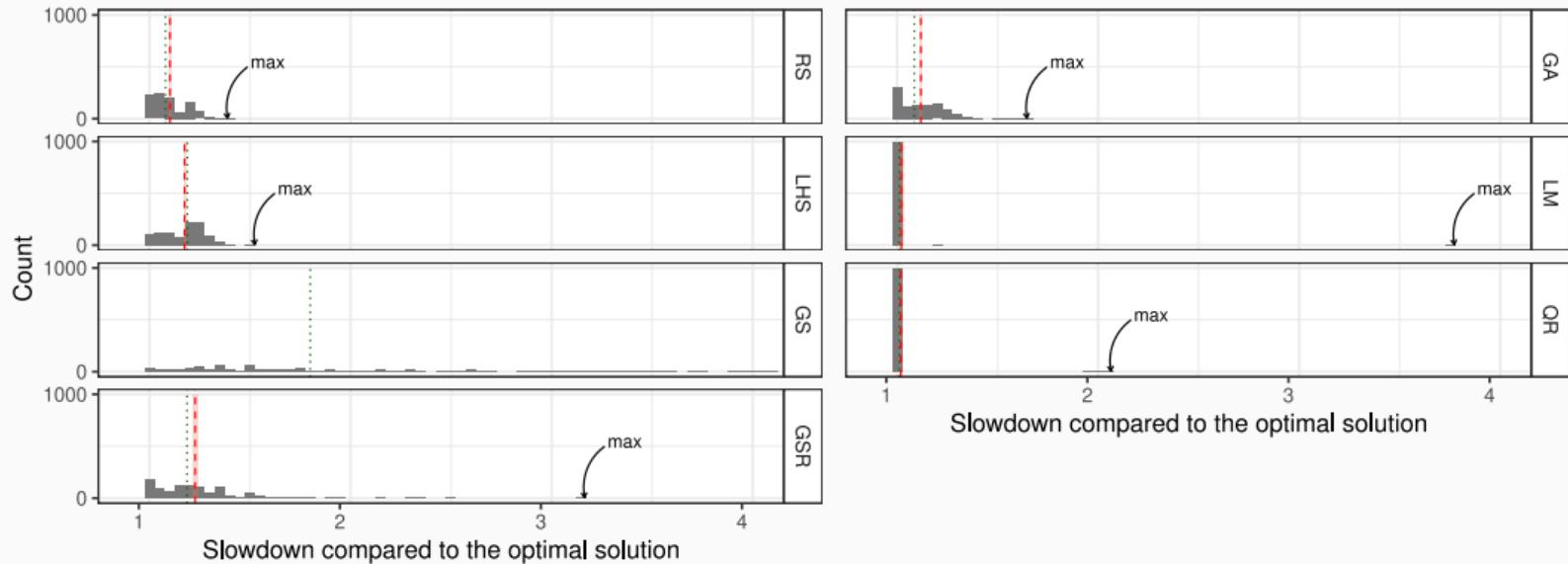
# Sequential Approach to Optimization: Transparency

## Step 3 Fix *elements*



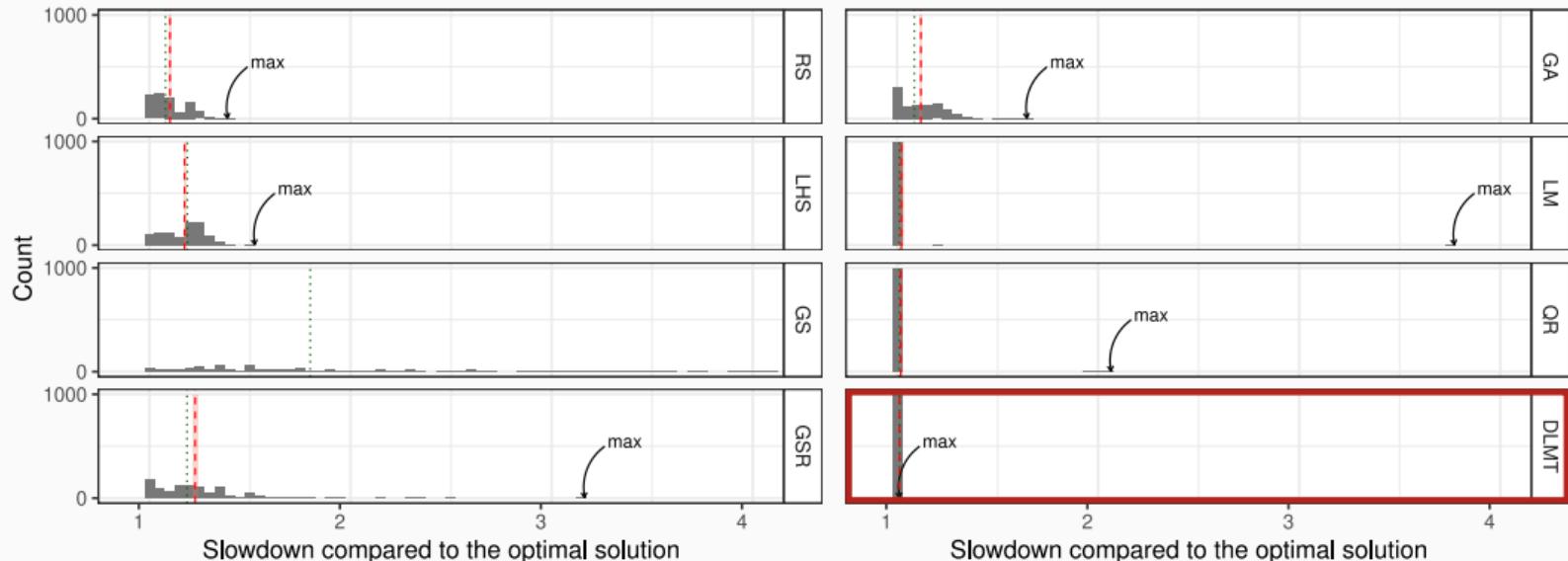
| Step            | Term                        | Sum Sq.               | F-value              | p(>F)                |
|-----------------|-----------------------------|-----------------------|----------------------|----------------------|
| 1 <sup>st</sup> | <i>y_component_number</i>   | $2.1 \times 10^{-18}$ | $7.3 \times 10^{-1}$ | $4.1 \times 10^{-1}$ |
|                 | <i>1/y_component_number</i> | $4.4 \times 10^{-18}$ | $1.6 \times 10^0$    | $2.4 \times 10^{-1}$ |
|                 | <i>vector_length</i>        | $1.3 \times 10^{-17}$ | $4.4 \times 10^0$    | $4.7 \times 10^{-2}$ |
|                 | <i>lws_y</i>                | $6.9 \times 10^{-17}$ | $2.4 \times 10^1$    | $3.5 \times 10^{-4}$ |
|                 | <i>1/lws_y</i>              | $1.8 \times 10^{-17}$ | $6.2 \times 10^0$    | $2.8 \times 10^{-2}$ |
|                 | <i>load_overlap</i>         | $9.1 \times 10^{-20}$ | $3.2 \times 10^{-2}$ | $8.6 \times 10^{-1}$ |
|                 | <i>temporary_size</i>       | $7.1 \times 10^{-18}$ | $2.5 \times 10^0$    | $1.4 \times 10^{-1}$ |
|                 | <i>elements_number</i>      | $3.1 \times 10^{-19}$ | $1.1 \times 10^{-1}$ | $7.5 \times 10^{-1}$ |
|                 | <i>1/elements_number</i>    | $1.3 \times 10^{-18}$ | $4.4 \times 10^{-1}$ | $5.2 \times 10^{-1}$ |
|                 | <i>threads_number</i>       | $7.2 \times 10^{-18}$ | $2.5 \times 10^0$    | $1.4 \times 10^{-1}$ |
|                 | <i>1/threads_number</i>     | $4.3 \times 10^{-18}$ | $1.5 \times 10^0$    | $2.4 \times 10^{-1}$ |
| 2 <sup>nd</sup> | <i>y_component_number</i>   | $1.2 \times 10^{-19}$ | $2.1 \times 10^1$    | $1.4 \times 10^{-3}$ |
|                 | <i>1/y_component_number</i> | $1.4 \times 10^{-20}$ | $2.4 \times 10^0$    | $1.5 \times 10^{-1}$ |
|                 | <i>load_overlap</i>         | $4.1 \times 10^{-21}$ | $7.3 \times 10^{-1}$ | $4.1 \times 10^{-1}$ |
|                 | <i>temporary_size</i>       | $1.4 \times 10^{-21}$ | $2.6 \times 10^{-1}$ | $6.2 \times 10^{-1}$ |
|                 | <i>elements_number</i>      | $6.0 \times 10^{-22}$ | $1.1 \times 10^{-1}$ | $7.5 \times 10^{-1}$ |
|                 | <i>1/elements_number</i>    | $2.7 \times 10^{-21}$ | $4.8 \times 10^{-1}$ | $5.0 \times 10^{-1}$ |
|                 | <i>threads_number</i>       | $7.2 \times 10^{-21}$ | $1.3 \times 10^0$    | $2.9 \times 10^{-1}$ |
|                 | <i>1/threads_number</i>     | $2.9 \times 10^{-20}$ | $5.1 \times 10^0$    | $4.0 \times 10^{-2}$ |
| 3 <sup>rd</sup> | <i>load_overlap</i>         | $7.4 \times 10^{-25}$ | $3.8 \times 10^0$    | $1.1 \times 10^{-1}$ |
|                 | <i>temporary_size</i>       | $1.1 \times 10^{-22}$ | $5.7 \times 10^2$    | $2.4 \times 10^{-1}$ |
|                 | <i>elements_number</i>      | $9.3 \times 10^{-22}$ | $4.7 \times 10^3$    | $1.2 \times 10^{-8}$ |
|                 | <i>1/elements_number</i>    | $3.1 \times 10^{-22}$ | $1.6 \times 10^3$    | $1.9 \times 10^{-7}$ |

## Results: 1000 Repetitions with a Budget of 120 Measurements



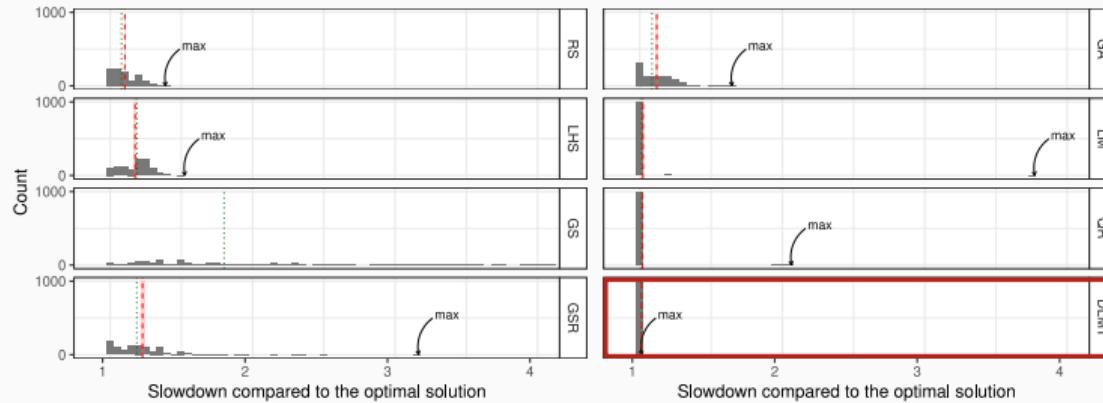
RS: Random Sampling, LHS: Latin Hypercube Sampling, GS: Greedy Search,  
GSR: Greedy Search w. Restart, GA: Genetic Algorithm, LM: Linear Model, QR: Quantile Regression

## Results: 1000 Repetitions with a Budget of 120 Measurements



RS: Random Sampling, LHS: Latin Hypercube Sampling, GS: Greedy Search,  
GSR: Greedy Search w. Restart, GA: Genetic Algorithm, LM: Linear Model, QR: Quantile Regression,  
DLMT: D-Optimal Designs, Linear Model w. Transform

# Parsimony under Tight Budget Constraints



| Method  | Slowdown    |             |             | Budget       |           |
|---|-------------|-------------|-------------|--------------|-----------|
|   | Mean        | Min.        | Max.        | Mean         | Max.      |
| Random Sampling (RS)                            | 1.10        | 1.00        | 1.39        | 120.00       | 120       |
| Latin Hypercube Sampling (LHS)                  | 1.17        | 1.00        | 1.52        | 98.92        | 125       |
| Greedy Search (GS)                              | 6.46        | 1.00        | 124.76      | 22.17        | 106       |
| Greedy Search w. Restart (GSR)                  | 1.23        | 1.00        | 3.16        | 120.00       | 120       |
| Genetic Algorithm (GA)                          | 1.12        | 1.00        | 1.65        | 120.00       | 120       |
| Linear Model (LM)                               | 1.02        | 1.01        | 3.77        | 119.00       | 119       |
| Quantile Regression (QR)                        | 1.02        | 1.01        | 2.06        | 119.00       | 119       |
| <b>D-Opt., Linear Model w. Transform (DLMT)</b> | <b>1.01</b> | <b>1.01</b> | <b>1.01</b> | <b>54.84</b> | <b>56</b> |

# Application: Search Problems in Automatic Performance Tuning (SPAPT)

## SPAPT Kernels

- 16 problems on multiple HPC domains
- Generated by ORIO
- Sets of constraints for each kernel
- Too large to completely evaluate
- Same starting model for all kernels

## Numeric Parameters

- Unrolling, blocking, for multiple loops

## Binary Categorical Parameters

- Parallelization, vectorization, scalar replacement

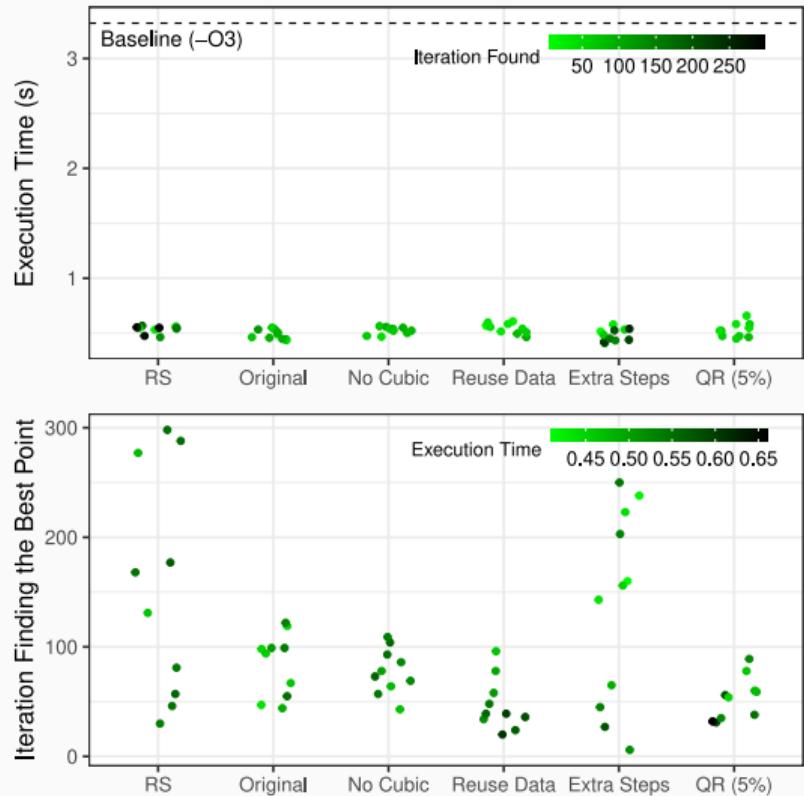
## Search Spaces

| Kernel             | Short Description               | Factors | Size      |
|--------------------|---------------------------------|---------|-----------|
| <i>dgemv3</i>      | Scalar, vector & matrix mult.   | 49      | $10^{36}$ |
| <i>stencil3d</i>   | 3-D stencil computation         | 29      | $10^{27}$ |
| <i>trmm</i>        | Triangular matrix operations    | 25      | $10^{23}$ |
| <i>gemver</i>      | Vector mult. & matrix add.      | 24      | $10^{22}$ |
| <i>tensor</i>      | Tensor matrix mult.             | 20      | $10^{19}$ |
| <i>correlation</i> | Correlation computation         | 21      | $10^{17}$ |
| <i>atax</i>        | Matrix transp. & vector mult.   | 18      | $10^{16}$ |
| <i>adi</i>         | Matrix sub., mult., & div.      | 20      | $10^{15}$ |
| <i>seidel</i>      | Matrix factorization            | 15      | $10^{14}$ |
| <i>mm</i>          | Matrix multiplication           | 13      | $10^{12}$ |
| <i>lu</i>          | LU decomposition                | 14      | $10^{12}$ |
| <i>bicg</i>        | Subkernel of BiCGStab           | 13      | $10^{11}$ |
| <i>gesummv</i>     | Scalar, vector, & matrix mult.  | 11      | $10^9$    |
| <i>mvt</i>         | Matrix vector product & transp. | 12      | $10^9$    |
| <i>jacobi</i>      | 1-D Jacobi computation          | 11      | $10^9$    |
| <i>hessian</i>     | Hessian computation             | 9       | $10^7$    |

## Performance Metric and Starting Model

$$\text{run\_time} \sim \sum_{i=1, \dots, p} x_i + x_i^2 + x_i^3$$

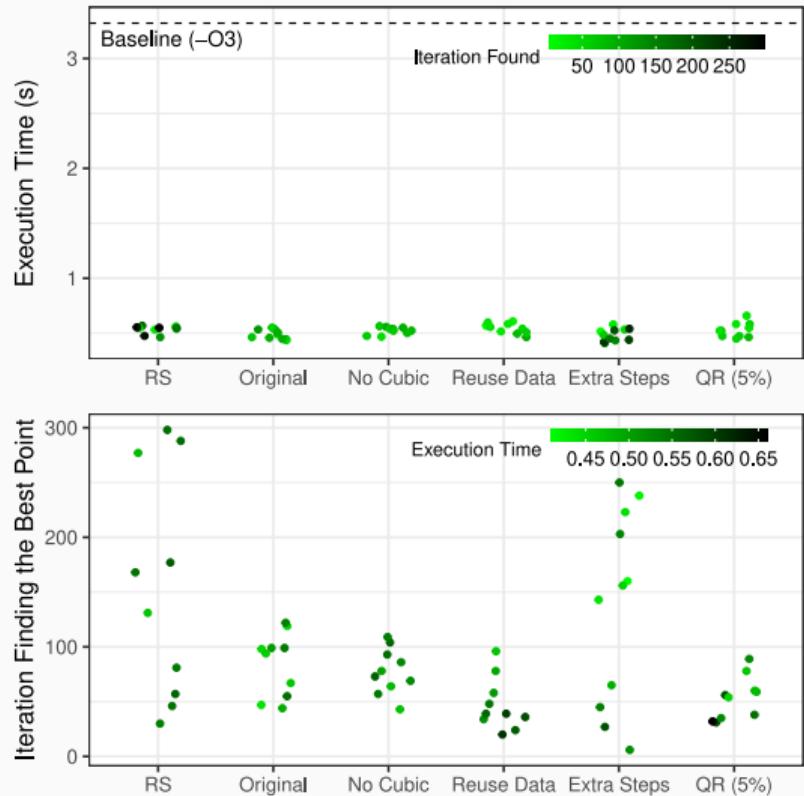
# Summarizing Results: *bicg* Kernel



## Interpreting the Optimization

- 4 steps, budget of 300 measurements
- Improvements compared to  $-O3$ , not so much compared to Random Sampling
- 2 most practically significant parameters detected at 1<sup>st</sup> and 2<sup>nd</sup> steps
- Other factors were statistically significant, but not practically

# Summarizing Results: *bicg* Kernel



## Interpreting the Optimization

- 4 steps, budget of 300 measurements
- Improvements compared to  $-O3$ , not so much compared to Random Sampling
- 2 most practically significant parameters detected at 1<sup>st</sup> and 2<sup>nd</sup> steps
- Other factors were statistically significant, but not practically

Is there anything else to find?

- How far are we from the global optimum?

# Applying Design of Experiments and Learning to Autotuning

## Random Sampling has Good Performance

- Abundance of local optima?

## Motivating Results with the Laplacian Kernel

- Knowledge of the search space
- Good starting model

## Broader Evaluation with SPAPT Kernels

- Is there something else to find?
- Can we find it by exploiting structure?

## Different Abstraction Levels

- Algorithm, implementation, dependencies, compiler, OS, hardware
- How to combine them effectively?

## Sequential and Incremental Approach

- Definitive search space restrictions
- Experiments and improvements by batch
- Rigid models

# Applying Design of Experiments and Learning to Autotuning

## Random Sampling has Good Performance

- Abundance of local optima?

## Motivating Results with the Laplacian Kernel

- Knowledge of the search space
- Good starting model

## Broader Evaluation with SPAPT Kernels

- Is there something else to find?
- Can we find it by exploiting structure?

## Different Abstraction Levels

- Algorithm, implementation, dependencies, compiler, OS, hardware
- How to combine them effectively?

## Sequential and Incremental Approach

- Definitive search space restrictions
- Experiments and improvements by batch
- Rigid models

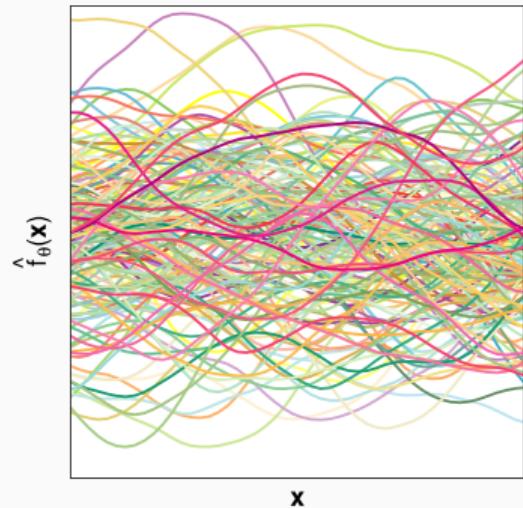
## More Flexibility with Gaussian Processes

Balance exploitation of structure  
with unrestricted exploration

# Applying Active Learning with Gaussian Processes

---

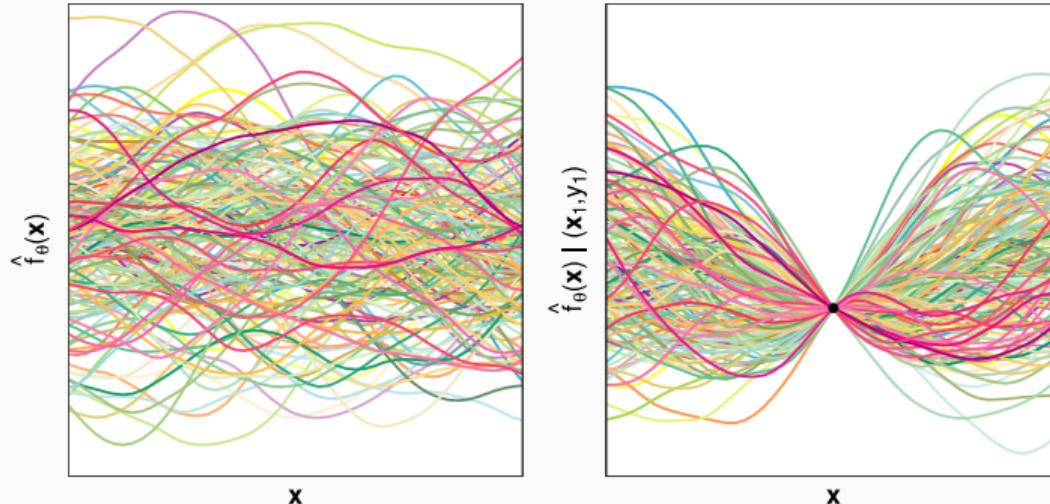
## Sampling Functions with Gaussian Process Regression



### Gaussian Process Surrogates

- $f : \mathcal{X} \rightarrow \mathbb{R}$
- Model:  $f(\mathbf{x}) \sim \mathcal{N}(\mu, \Sigma)$
- Data:  $(\mathbf{x}_k, y_k = f(\mathbf{x}_k))$
- Surrogate  $\hat{f}_\theta(\mathbf{x}) \sim f(\mathbf{x}) \mid \mathbf{X}, \mathbf{y}$

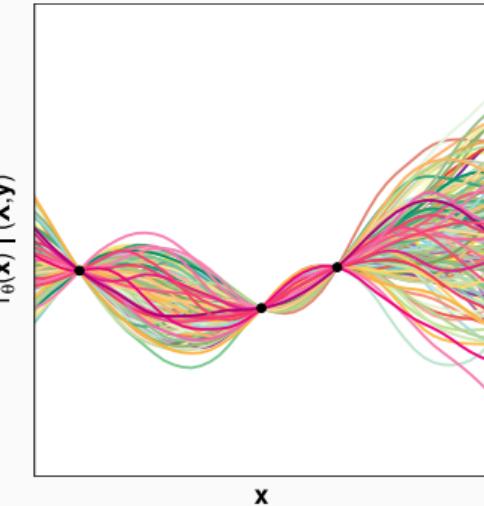
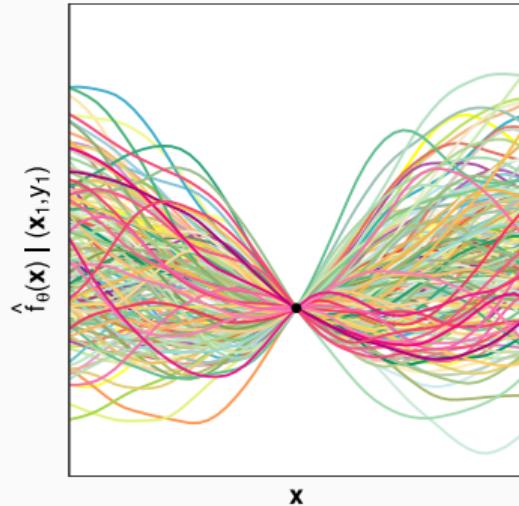
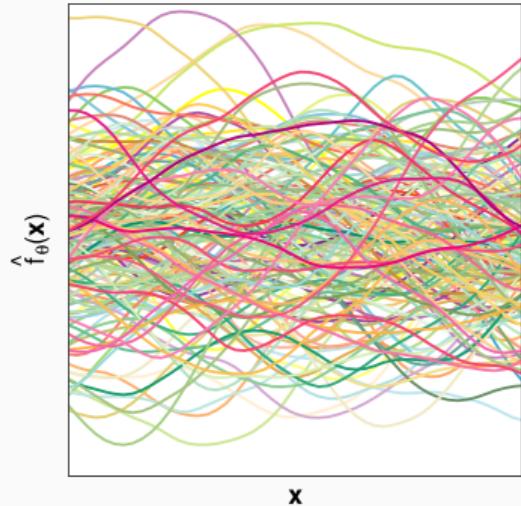
## Sampling Functions with Gaussian Process Regression



## Gaussian Process Surrogates

- $f : \mathcal{X} \rightarrow \mathbb{R}$
- Model:  $f(\mathbf{x}) \sim \mathcal{N}(\mu, \Sigma)$
- Data:  $(\mathbf{x}_k, y_k = f(\mathbf{x}_k))$
- Surrogate  $\hat{f}_\theta(\mathbf{x}) \sim f(\mathbf{x}) \mid \mathbf{X}, \mathbf{y}$

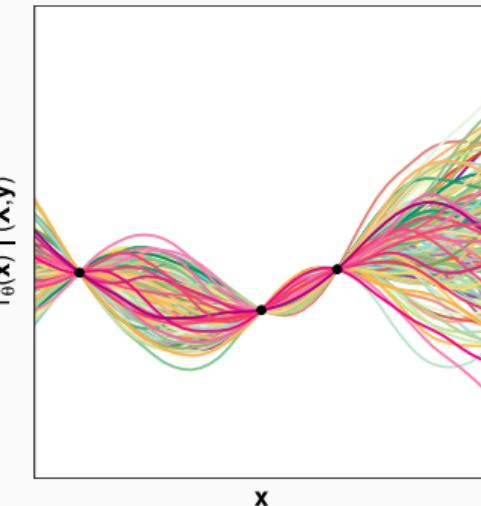
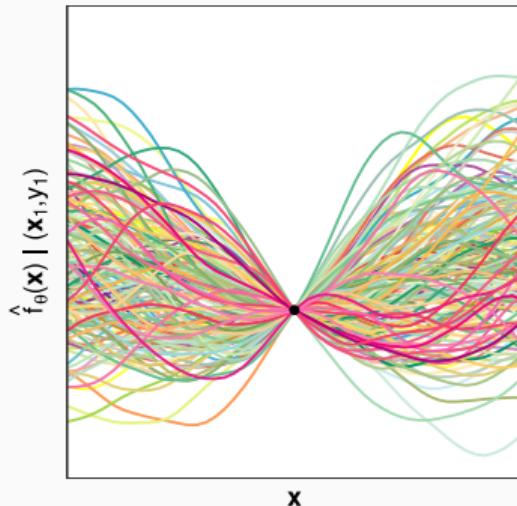
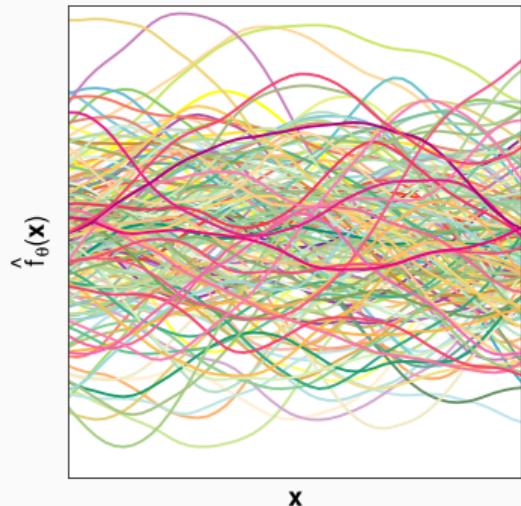
## Sampling Functions with Gaussian Process Regression



## Gaussian Process Surrogates

- $f : \mathcal{X} \rightarrow \mathbb{R}$
- Model:  $f(\mathbf{x}) \sim \mathcal{N}(\mu, \Sigma)$
- Data:  $(\mathbf{x}_k, y_k = f(\mathbf{x}_k))$
- Surrogate  $\hat{f}_\theta(\mathbf{x}) \sim f(\mathbf{x}) \mid \mathbf{X}, \mathbf{y}$

# Sampling Functions with Gaussian Process Regression



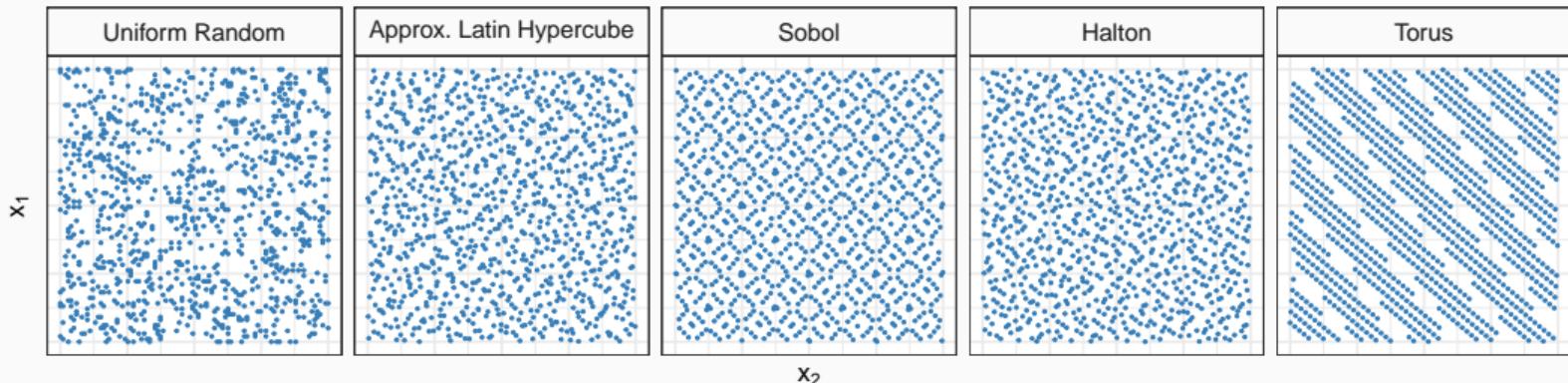
## Gaussian Process Surrogates

- $f : \mathcal{X} \rightarrow \mathbb{R}$
- Model:  $f(\mathbf{x}) \sim \mathcal{N}(\mu, \Sigma)$
- Data:  $(\mathbf{x}_k, y_k = f(\mathbf{x}_k))$
- Surrogate  $\hat{f}_\theta(\mathbf{x}) \sim f(\mathbf{x}) \mid \mathbf{X}, \mathbf{y}$

### How to choose $\mathbf{X}$ ?

- Minimizing  $f$
- Building an accurate surrogate
- Doing both?

# Space-filling Designs: Sampling for High Dimension



## Curse of Dimensionality: Sampling

- In high dimension, most sampled points are on the shell

## Strategies

- Latin Hypercube Sampling: Partition and then sample, might need to optimize later
- Low-discrepancy:** deterministic space-filling sequences

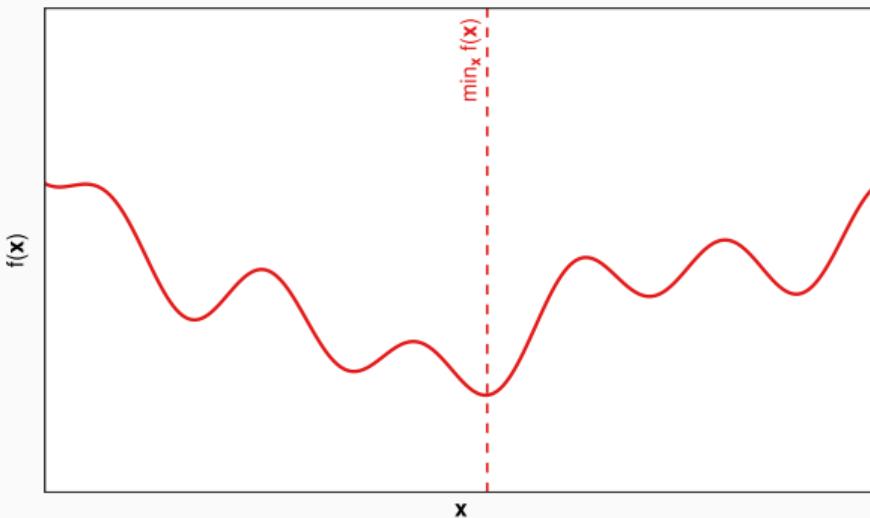
## Optimization

- More uniform starting samples

## Interpretation

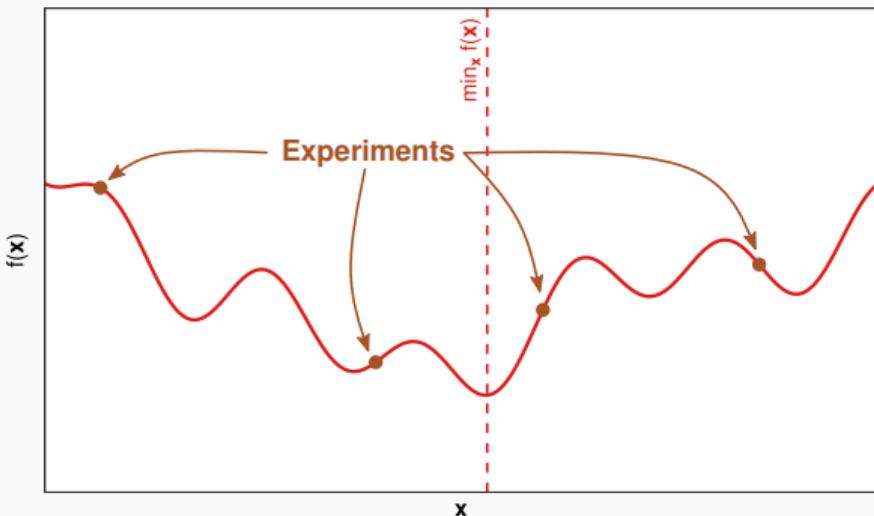
- Sobol indices
- Still need many samples

## Expected Improvement: Balancing Exploitation and Exploration



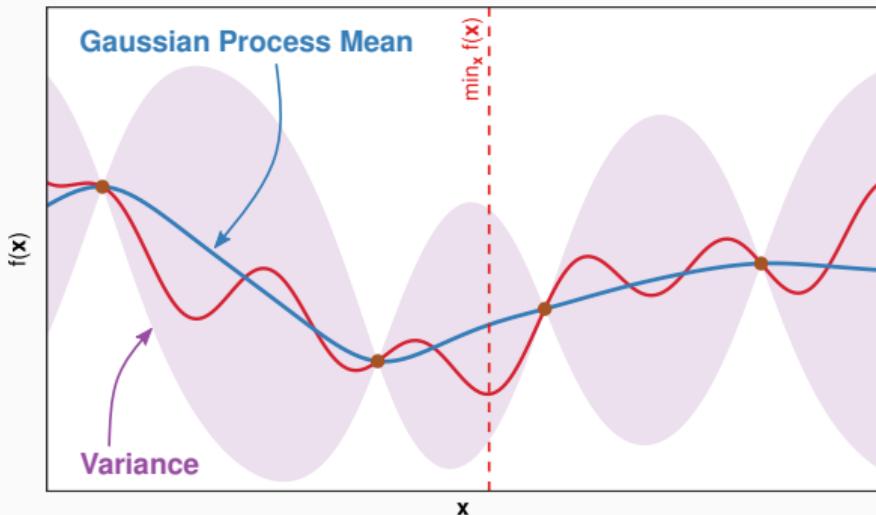
How to decide where to measure next?

## Expected Improvement: Balancing Exploitation and Exploration



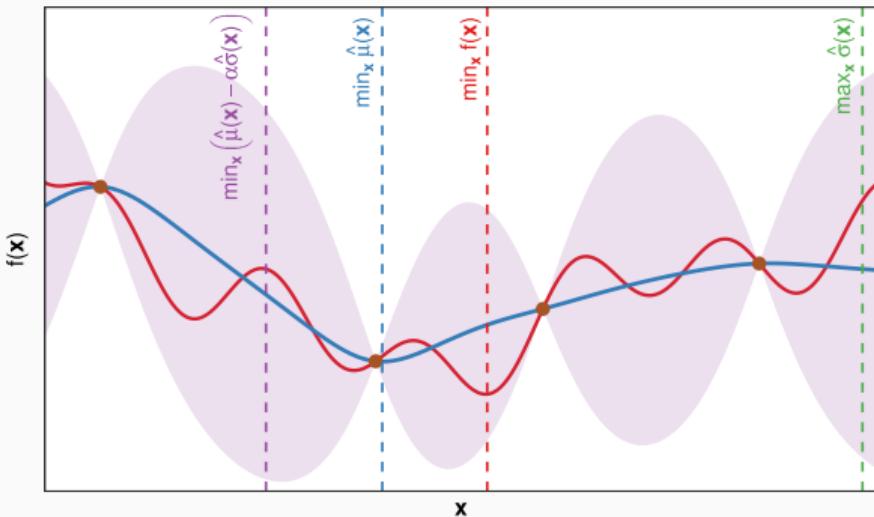
How to decide where to measure next?

## Expected Improvement: Balancing Exploitation and Exploration



How to decide where to measure next?

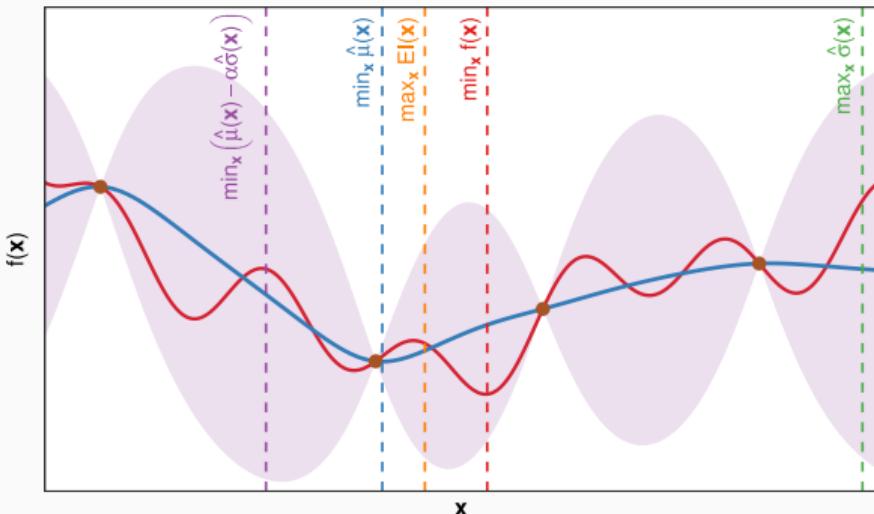
## Expected Improvement: Balancing Exploitation and Exploration



How to decide where to measure next?

- Explore: maximum **variance**
- Exploit: minimum **mean**
- Balance: minimum **mean** minus **confidence interval lower bound**

# Expected Improvement: Balancing Exploitation and Exploration



## Computing the Expected Improvement (EI)

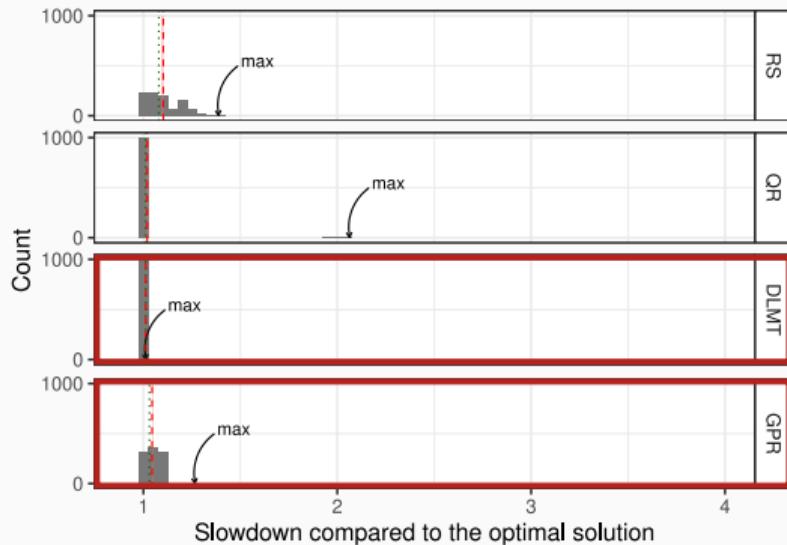
$$\mathbb{E}[I(x)] = (y^* - \hat{\mu}(x)) \Phi\left(\frac{y^* - \hat{\mu}(x)}{\hat{\sigma}(x)}\right) + \hat{\sigma}(x) \phi\left(\frac{y^* - \hat{\mu}(x)}{\hat{\sigma}(x)}\right)$$

How to decide where to measure next?

- Explore: maximum **variance**
- Exploit: minimum **mean**
- Balance: minimum **mean** minus **confidence interval** lower bound
- Balance: maximum **Expected Improvement**

# Application: OpenCL Laplacian and SPAPT Kernels

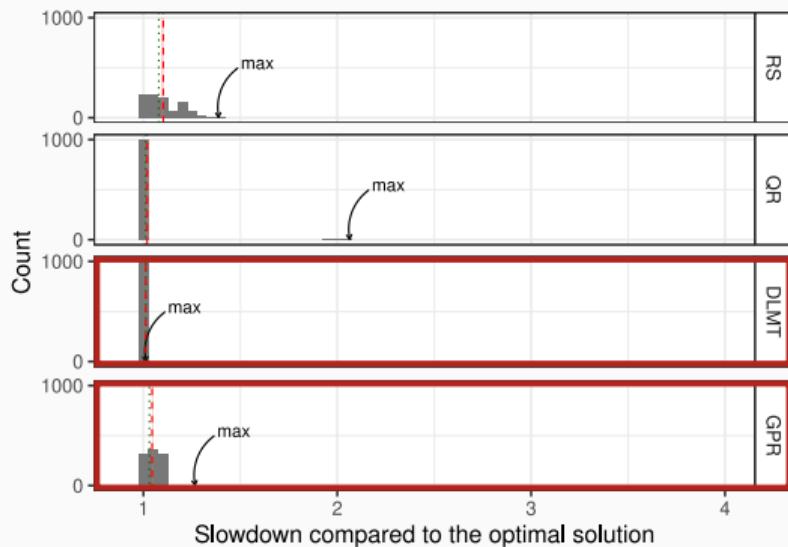
## OpenCL Laplacian Kernel



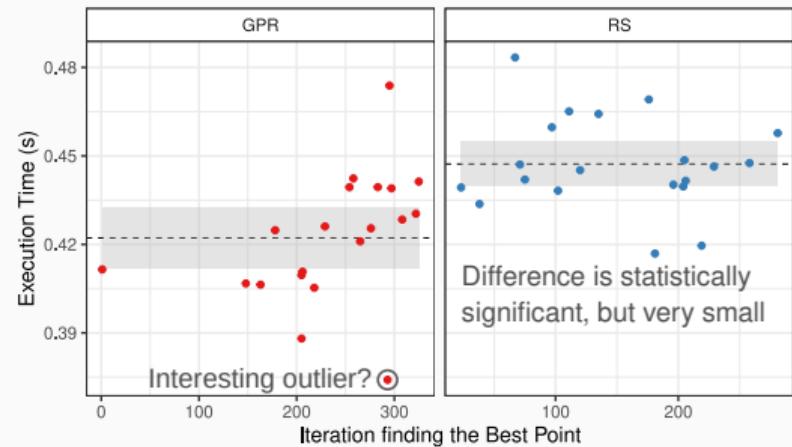
- The more **informed** model performed better

# Application: OpenCL Laplacian and SPAPT Kernels

## OpenCL Laplacian Kernel



## SPAPT *bicg* Kernel

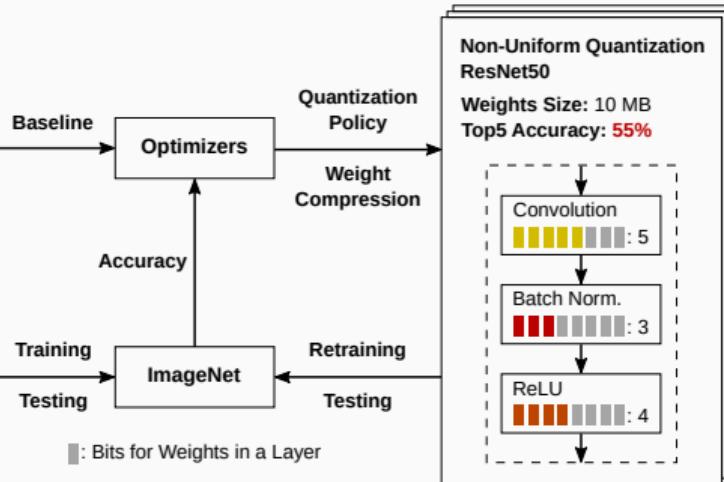
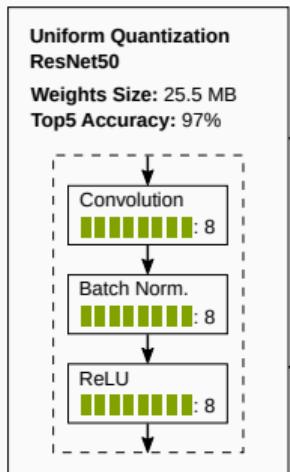


- The more **informed** model performed better

- Consistent, small improvements on RS
- Found outliers, but did not explore further

# Application: Quantization for Convolutional Neural Networks

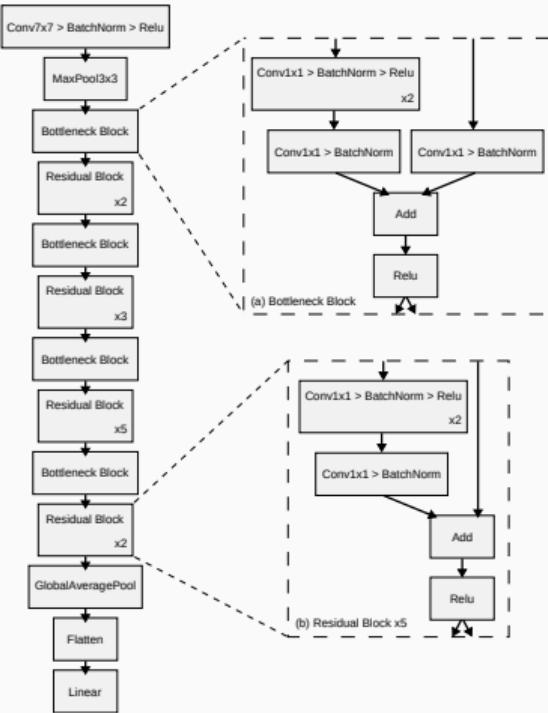
Keep Weights  $\leq 10\text{MB}$  with Mixed-Precision Quantization



Maintain Original Accuracy on ImageNet



ResNet50:  $10^{48}$  Configurations



# Results

## Accuracy Metrics

- Top1: correct class is the network's **most probable** prediction
- Top5: correct class is in the **five most probable** predictions

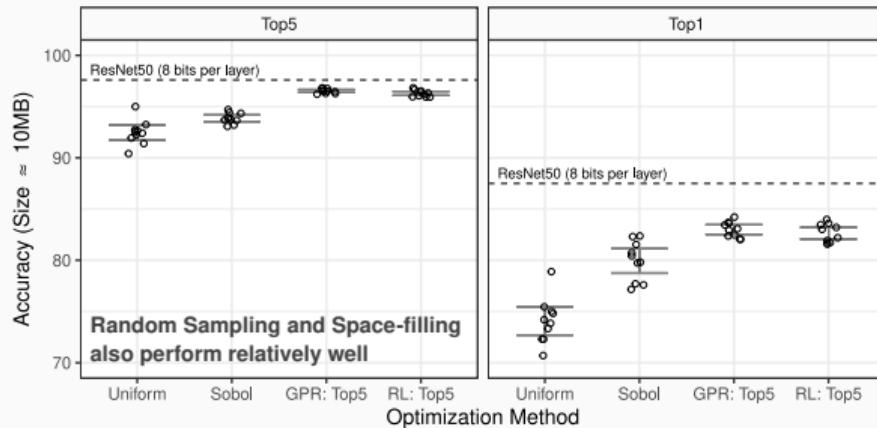
## Constraints

- Weight size must be  $\leq 10\text{MB}$

## Optimizing for Top5

- Compared with Random Sampling, Space-filling designs, and Reinforcement Learning
- GPR was more consistent

## Comparison with Reinforcement Learning



# Results

## Accuracy Metrics

- Top1: correct class is the network's **most probable** prediction
- Top5: correct class is in the **five most probable** predictions

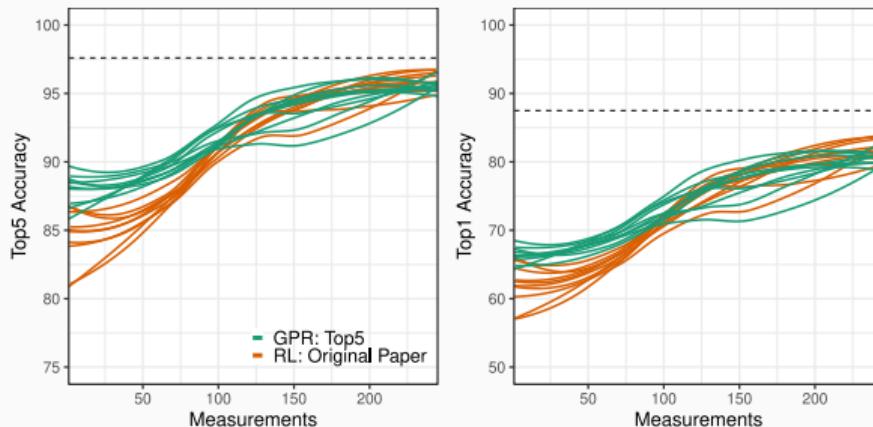
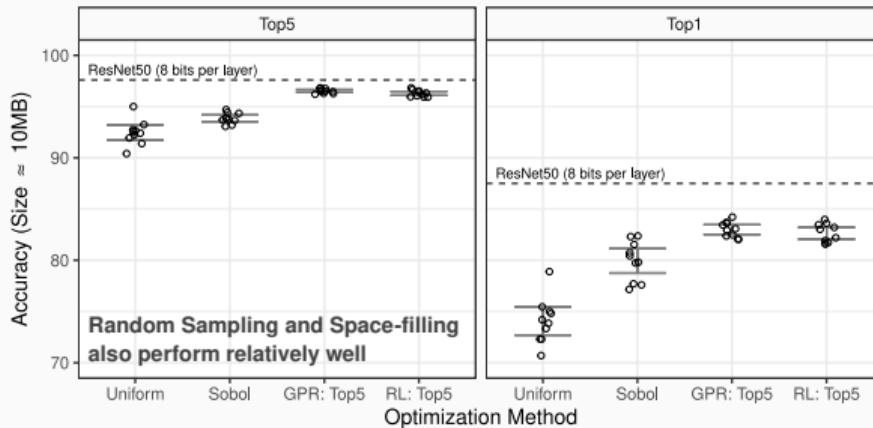
## Constraints

- Weight size must be  $\leq 10\text{MB}$

## Optimizing for Top5

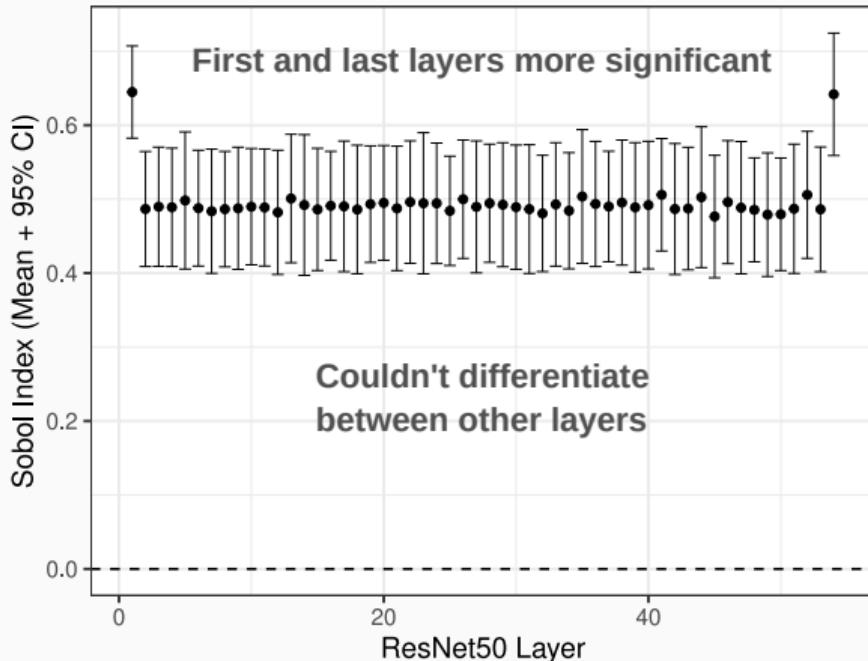
- Compared with Random Sampling, Space-filling designs, and Reinforcement Learning
- GPR was more consistent

## Comparison with Reinforcement Learning



# Interpreting Results

## Computing Sobol Indices



- Using all data, and only Sobol samples:  
inconclusive

## Optimizing for Size and Top5

- Adapted the RL algorithm
- GPR was more stable

## Fitting a GP to All Data

- Enables leveraging entire dataset
- Extremely time-consuming
- Unclear if viable with RL

## Expressing Structure with Kernels

- Could recover interpretability

## Discussion

### More Flexibility with Gaussian Processes

- No accurate modeling hypotheses needed
- Harder to interpret
- Not always achieves better optimizations
- Effort to build a good model can pay off

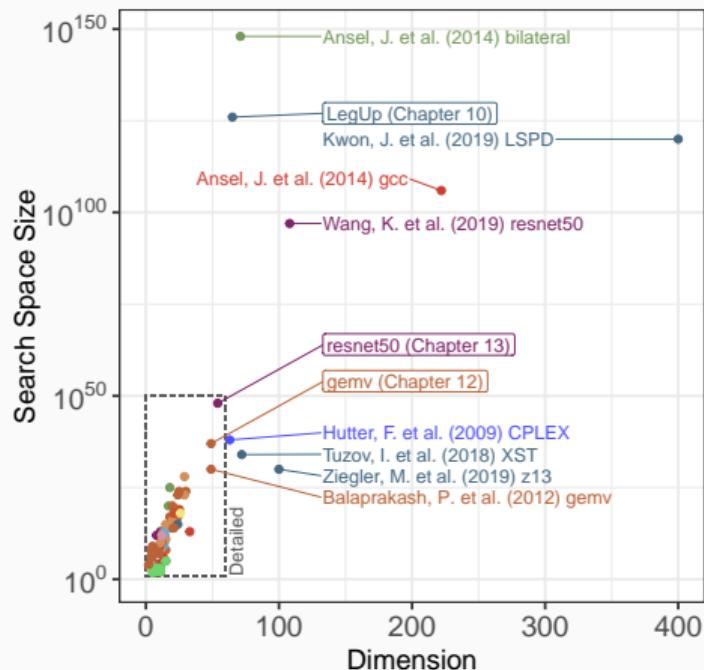
### Online Learning

- Deciding where to measure at each new experiment
- Balancing exploitation and exploration
- No restriction to subspaces

### Space-filling Designs

- Sampling in high dimension
- Filter to go around constraints

### Context: Size of the Search Space

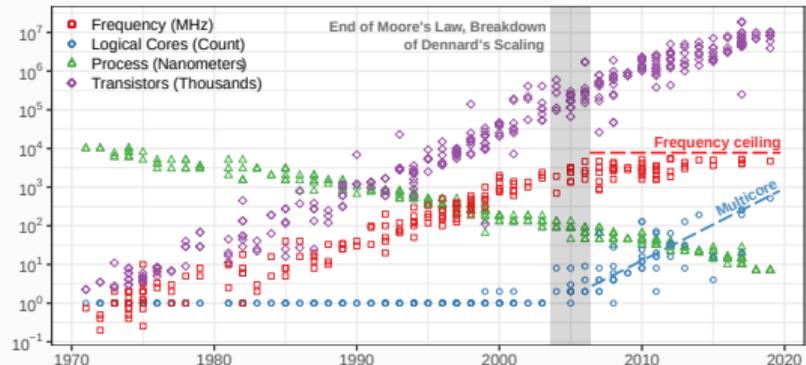
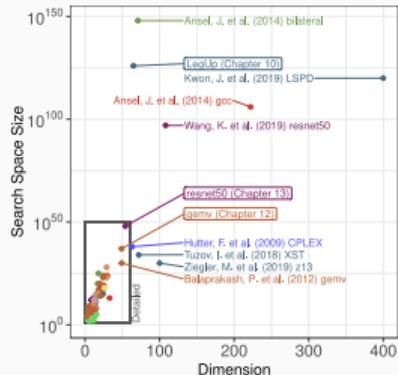
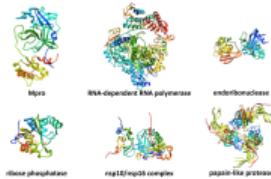
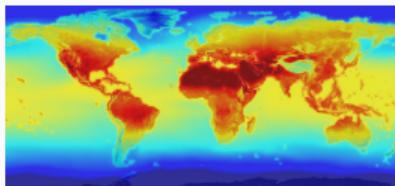


## Conclusion

---

# Toward Transparent and Parsimonious Autotuning

## Autotuning for High Performance Computing



## Contributions of this Thesis

- Developing transparent and parsimonious autotuning methods based on the Design of Experiments
- Evaluating different autotuning methods in different HPC domains

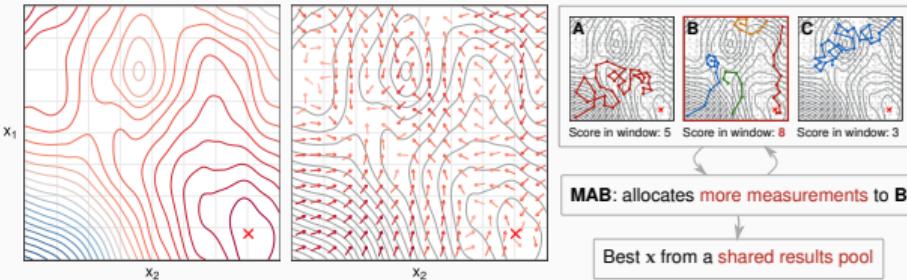
| Domain                   | Method  |
|--------------------------|---------|
| CUDA compiler parameters | F, D    |
| FPGA compiler parameters | F       |
| OpenCL Laplacian Kernel  | F, L, D |
| SPAPT Kernels            | L, D    |
| CNN Quantization         | L, D    |

F: Function Minimization, L: Learning,

D: Design of Experiments

: In this presentation

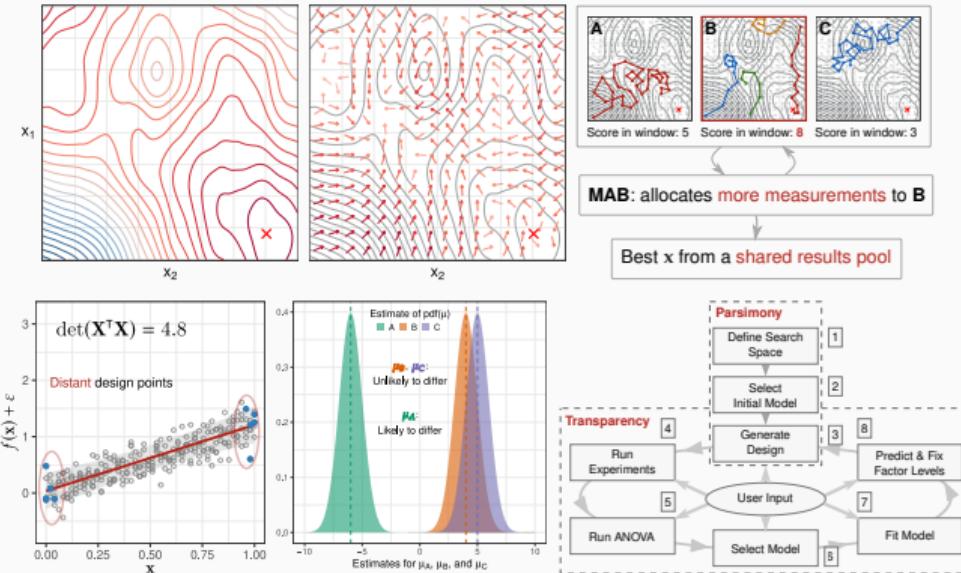
# Autotuning Methods Best Suited for Different Contexts



## Function Minimization

- Optimize  $f$  directly
- Hypotheses not always clear
- Use when modeling is hard

# Autotuning Methods Best Suited for Different Contexts



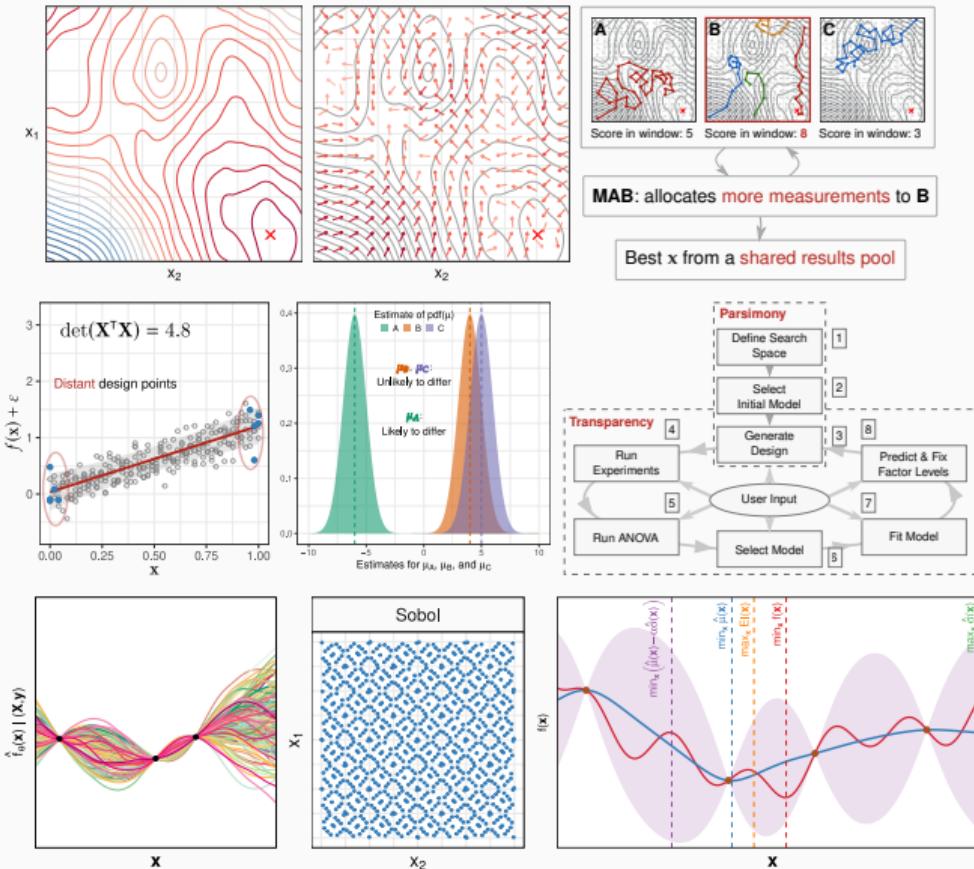
## Function Minimization

- Optimize  $f$  directly
- Hypotheses not always clear
- Use when modeling is hard

## Design of Experiments

- Optimize informed surrogates
- Sequential and incremental
- Use when a model is available

# Autotuning Methods Best Suited for Different Contexts



## Function Minimization

- Optimize  $f$  directly
- Hypotheses not always clear
- Use when modeling is hard

## Design of Experiments

- Optimize informed surrogates
- Sequential and incremental
- Use when a model is available

## Learning with GPs

- Exploration and exploitation
- Choose experiments online
- Use to build flexible surrogates

# Improving Autotuners: Collaborative, Exhaustive, and Reproducible Experiments

## Solving Similar Problems

- Each application required redoing work
- Benchmarks such as SPAPT are rare

## Exhaustive Measurements

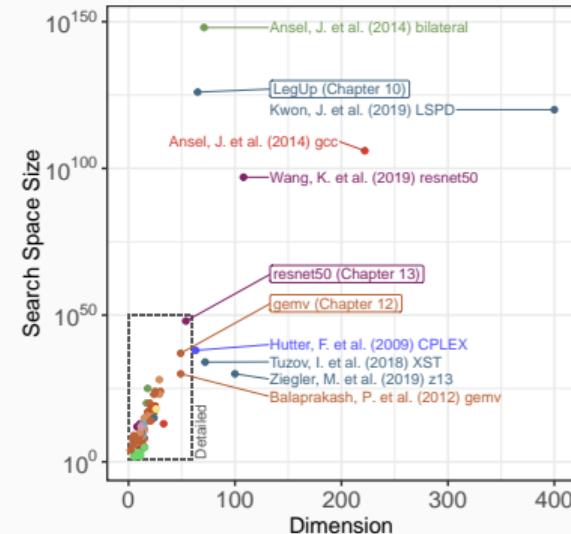
- Completely evaluate a few search spaces

## Collaborative Experiments

- Leverage community efforts

## Reproducibility

- Notebooks, workflows, archival, and sharing
- Target diverse domains, hardware, software, abstraction levels



Exhaustive and collective experiments are **interdependent approaches** and can help achieve reproducible autotuning

# Toward Transparent and Parsimonious Methods for Automatic Performance Tuning

---

Pedro Bruel

July 9 2021

## Advisors

Alfredo Goldman (USP)  
Arnaud Legrand (CNRS)  
Brice Videau (ANL)

## Jury

Stefan M. Wild (ANL)  
Albert Cohen (Google)  
Boyana Norris (UO)  
Lucia Drummond (UFF)

