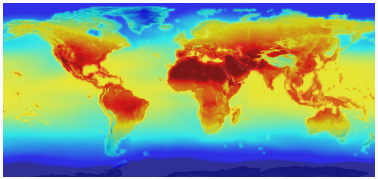
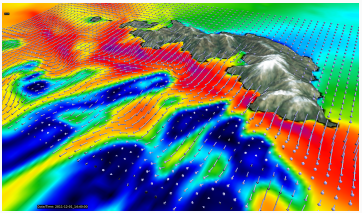
Toward Transparent and Parsimonious Methods for Automatic Performance Tuning

Pedro Bruel
phrb@ime.usp.br
July 9 2021

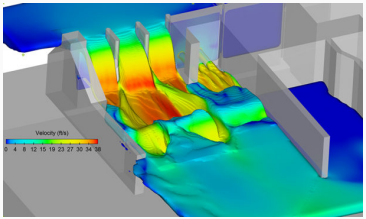
High Performance Computing is Needed at Multiple Scales, . . .

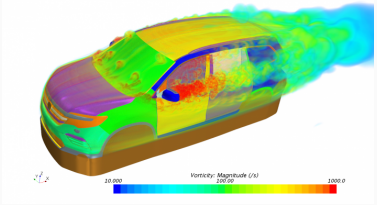
Climate simulation for policies to fight climate change



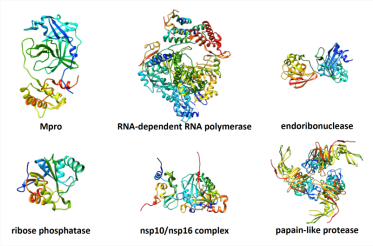


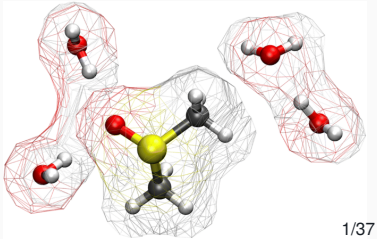
Fluid dynamics for stronger infrastructure and fuel efficiency



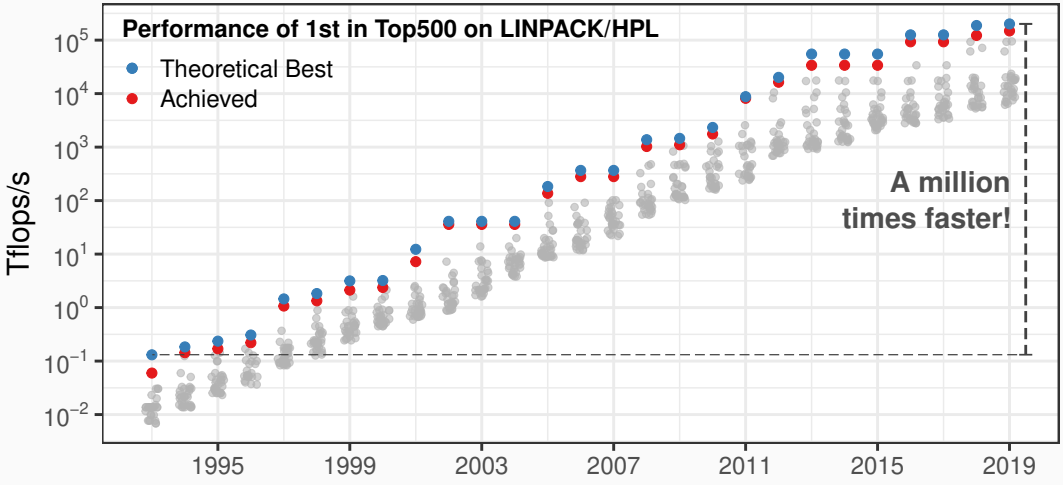


Molecular dynamics for virtual testing of drugs and vaccines

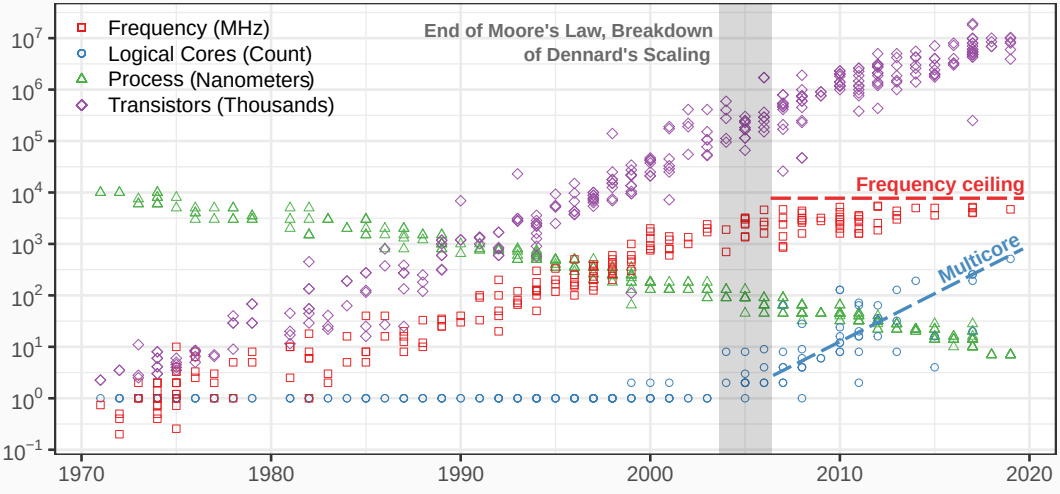




. . . and the Performance of Supercomputers has so far Improved Exponentially



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[][j] 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, y) 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?

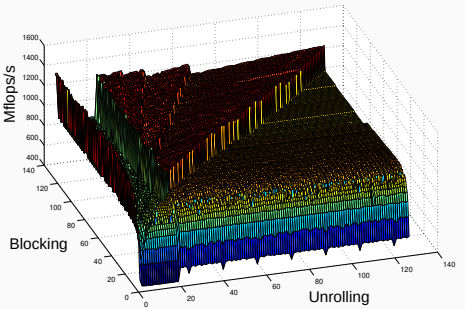
```
int N = 256;
int B_size = 4;
float A[N][N], B[N][N], C[N][N];
int i, j, k, x; // int U_size = 4;
// Initialize A, B, C
for(i = 0; i < N; i += B_size){
  for(j = 0; j < N; j += B_size){
    // Load block (i, j) of C to fast memory
    for(k = 0; k < N; k++){
      // Load block (i, k) of A to fast memory
      // Load block (k, y) of B to fast memory
      for(x = i; x < min(i + B_size, N); x++){
        C[i][j + 0] += A[i][k] * B[k][j + 0];
        C[i][j + 1] += A[i][k] * B[k][j + 1];
        C[i][j + 2] += A[i][k] * B[k][j + 2];
        C[i][j + 3] += A[i][k] * B[k][j + 3];
      }
    } // 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

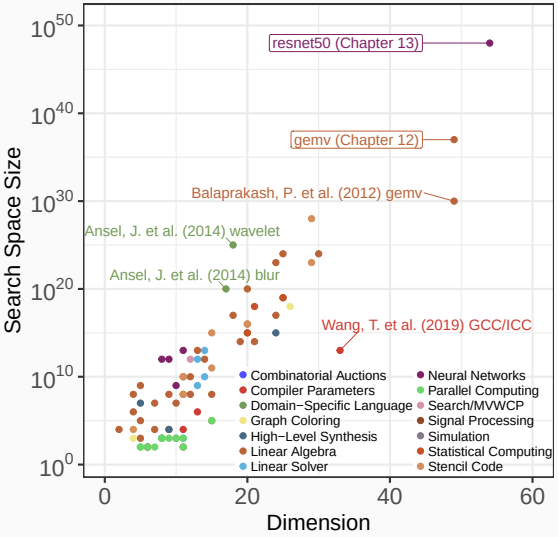
How to restructure memory accesses in loops to increase throughput by leveraging cache locality?

Resulting Search Space

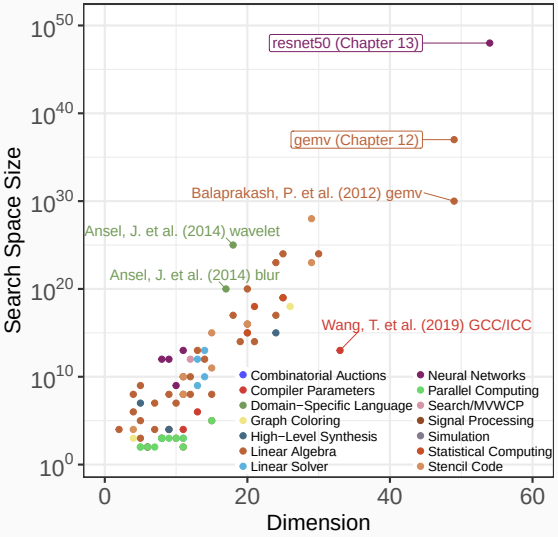


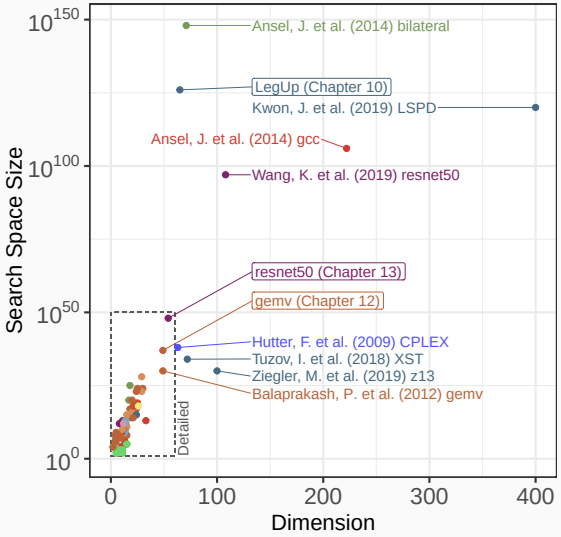

```
int N = 256;
int B_size = 4;
float A[N][N], B[N][N], C[N][N];
int i, j, k, x; // int U_size = 4;
// Initialize A, B, C
for(i = 0; i < N; i += B_size){
  for(j = 0; j < N; j += B_size){
    // Load block (i, j) of C to fast memory
    for(k = 0; k < N; k++){
      // Load block (i, k) of A to fast memory
      // Load block (k, y) of B to fast memory
      for(x = i; x < min(i + B_size, N); x++){
        C[i][j + 0] += A[i][k] * B[k][j + 0];
        C[i][j + 1] += A[i][k] * B[k][j + 1];
        C[i][j + 2] += A[i][k] * B[k][j + 2];
        C[i][j + 3] += A[i][k] * B[k][j + 3];
      }
    } // 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

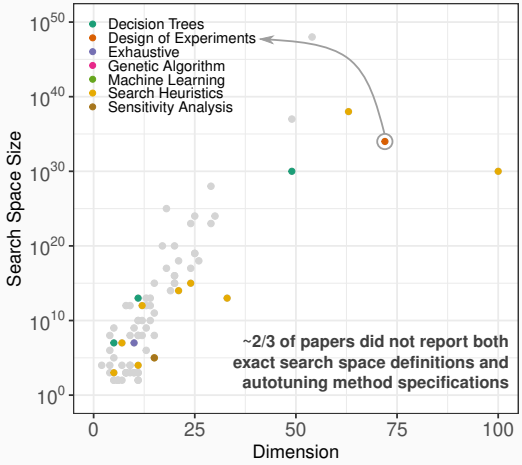
Performance as a Function

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

- Parameters: $\mathbf{x} = [x_1 \ \ldots \ x_n]^\mathsf{T} \in \mathcal{X}$
- 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

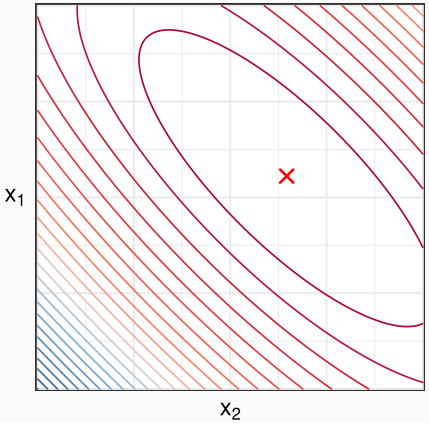
Applying Methods for Function Minimization

Minimizing Functions using Derivatives and Heuristics

We know or can compute information about f

- Directly measure new $\mathbf{x}_1, \ldots, \mathbf{x}_k, \ldots, \mathbf{x}_n$
- Search for the global optimum

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

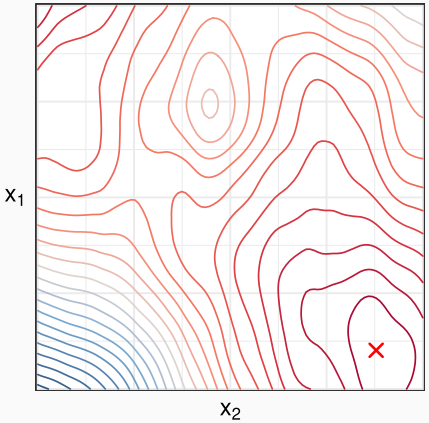


Minimizing Functions using Derivatives and Heuristics

We know or can compute information about f

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

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



Minimizing Functions using Derivatives and Heuristics

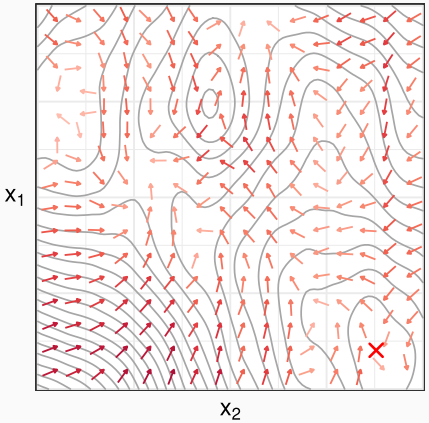
We know or can compute information about f

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

Strong Hypothesis: we can Compute Derivatives

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

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



Minimizing Functions using Derivatives and Heuristics

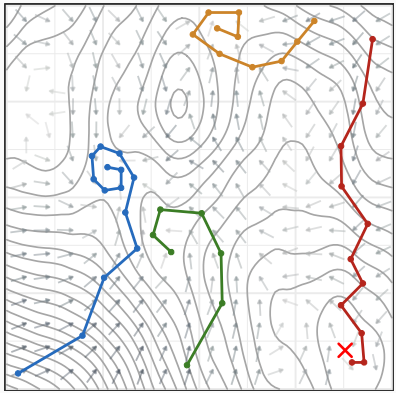
We know or can compute information about f

- Directly measure new $\mathbf{x}_1, \ldots, \mathbf{x}_k, \ldots, \mathbf{x}_n$
- 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 best point in a neighborhood

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



Minimizing Functions using Derivatives and Heuristics

We know or can compute information about f

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

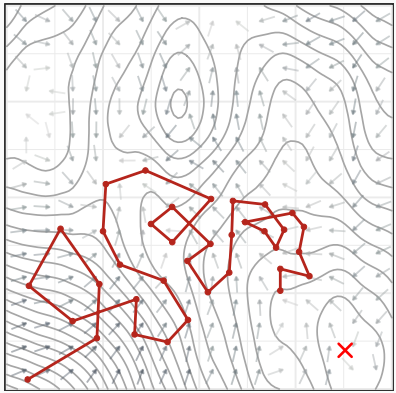
Strong Hypothesis: we can Compute Derivatives

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

Hard to State Hypotheses: Search Heuristics

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

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



Minimizing Functions using Derivatives and Heuristics

We know or can compute information about f

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

Strong Hypothesis: we can Compute Derivatives

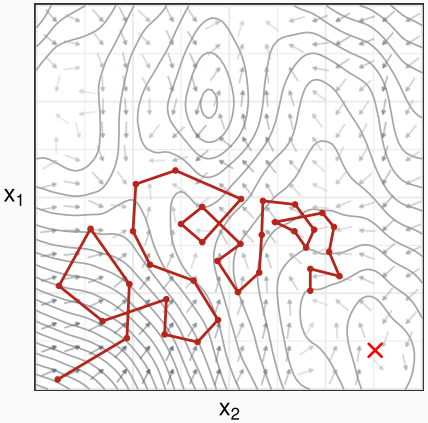
- $\mathbf{x}_k = \mathbf{x}_{k-1} - \mathbf{H}f(\mathbf{x}_{k-1})\nabla f(\mathbf{x}_{k-1})$
- Move to best point in a neighborhood

Hard to State Hypotheses: Search Heuristics

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

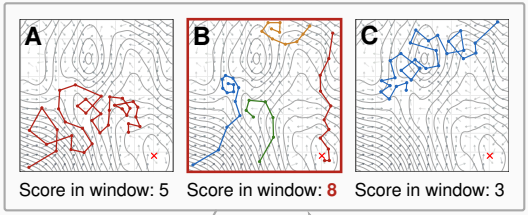
How to choose a method?

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



Choosing Methods from an Ensemble

Example of an Ensemble of Methods



Methods in this Ensemble

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

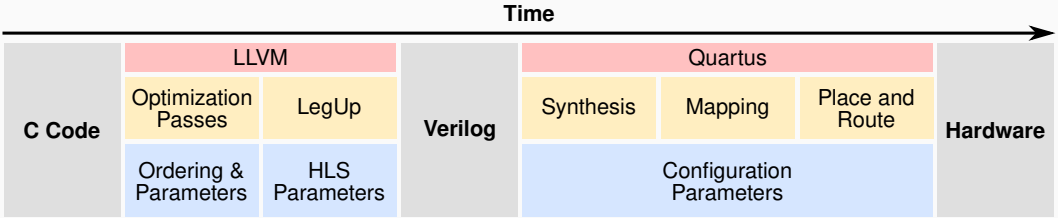
Minimization of f using OpenTuner

- 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 $\mathbf{x}$ in a time window
- The best $\mathbf{x}$ over all methods is reported

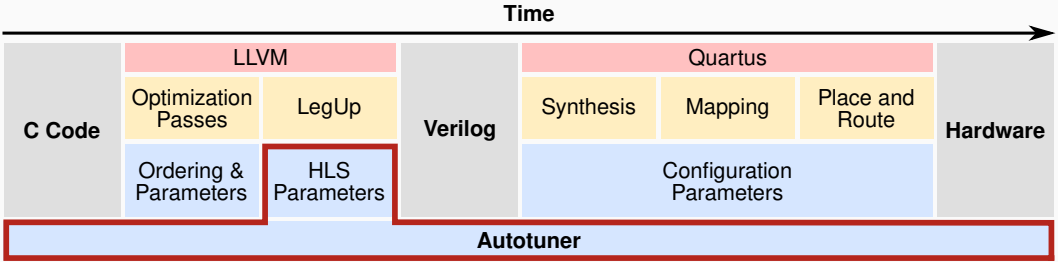
Application: High-Level Synthesis for FPGAs

Application: High-Level Synthesis for FPGAs

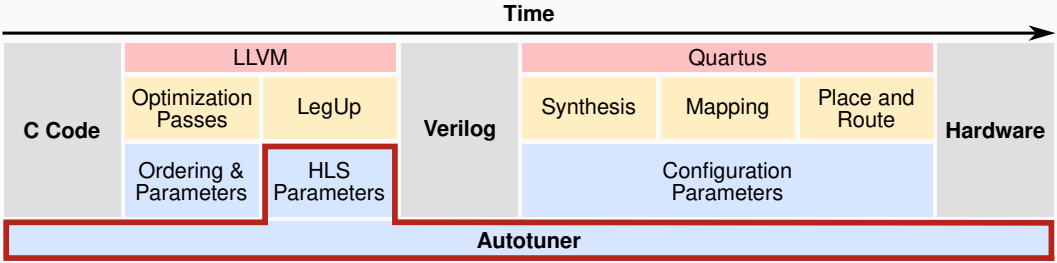
Application: High-Level Synthesis for FPGAs



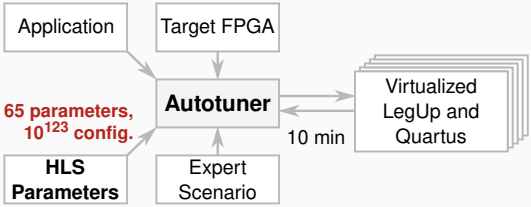
Application: High-Level Synthesis for FPGAs



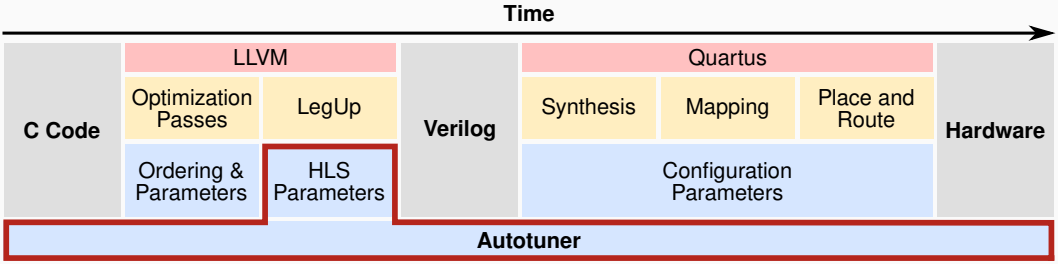
Application: High-Level Synthesis for FPGAs



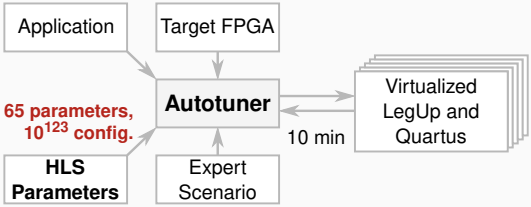
Search Space



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

	WNS	LUTs	Pins	BRAM	Regs	Blocks	Cycles	DSP	FMax
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
dfsin	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

Weighted Average for all Scenarios

	Balanced	Area	Performance	Perf. & Lat.
aes	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
dfsin	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

Discussion

Autotuning with Heuristics

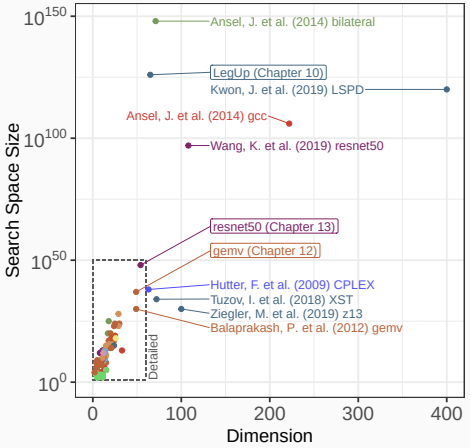
- Lack of structured exploration prevented statistical analyses and interpretation

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

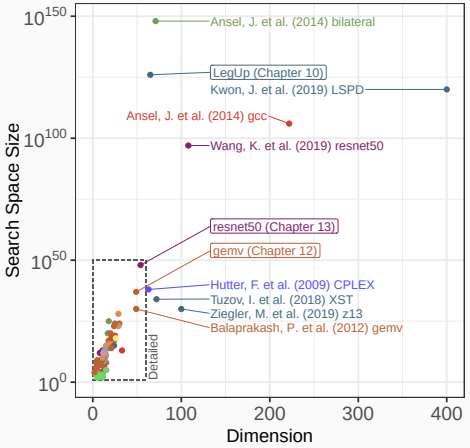
- Lack of structured exploration prevented statistical analyses and interpretation

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

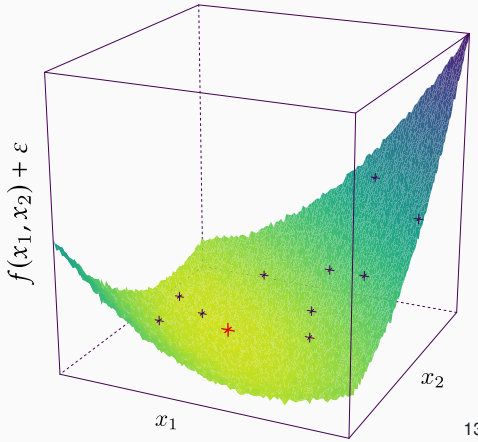
Search Space Hypotheses with Linear Models

Learning: Building Surrogates

- $f : \mathcal{X} \to \mathbb{R}$
- Model: $f(\mathbf{x}) = \mathbf{x}^\mathsf{T}\boldsymbol{\theta} + \varepsilon$, 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

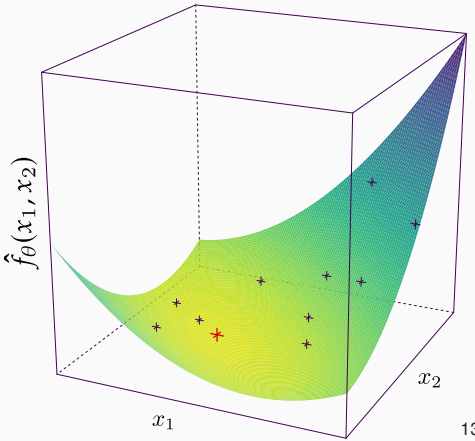
- $f : \mathcal{X} \to \mathbb{R}$
- Model: $f(\mathbf{x}) = \mathbf{x}^\intercal \boldsymbol{\theta} + \varepsilon$, with $\varepsilon \sim \mathcal{N}(0, \sigma^2)$
- Data: $(\mathbf{x}_k, y_k = f(\mathbf{x}_k))$
- $\mathbf{x}_{1,\dots,n}$ in a given design $\mathbf{X}$

Minimize a Surrogate instead of f

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

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

- $f : \mathcal{X} \to \mathbb{R}$
- Model: $f(\mathbf{x}) = \mathbf{x}^\intercal \boldsymbol{\theta} + \varepsilon$, 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

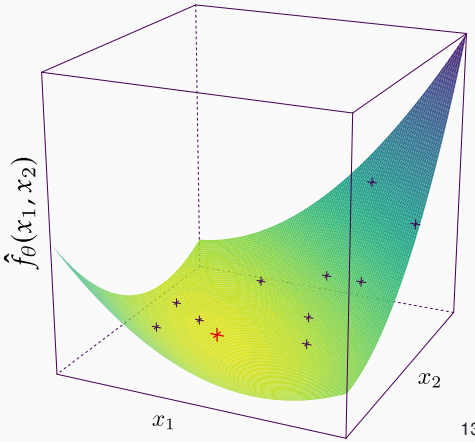
- Surrogate: $\hat{f}_\theta(\mathbf{x}') = \mathbf{x}'^\intercal \hat{\boldsymbol{\theta}}$
- Estimator: $\hat{\boldsymbol{\theta}} = (\mathbf{X}^\intercal \mathbf{X})^{-1} \mathbf{X}^\intercal \mathbf{y}$

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

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

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 $\mathbf{X}$

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

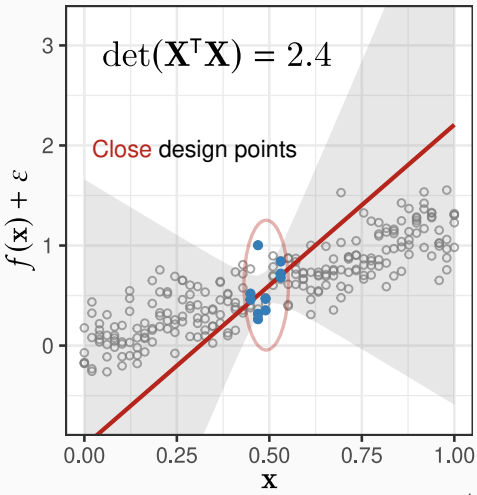
Components

- $\mathbf{X}_{n \times p}$: design matrix
- $\mathbf{x}_{1 \times n} \in \mathbf{X}$: factor columns
- $x_1, \ldots, x_p \in \mathbf{x}$: chosen factor levels

Examples

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

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



Design of Experiments

Choosing the Design $\mathbf{X}$

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

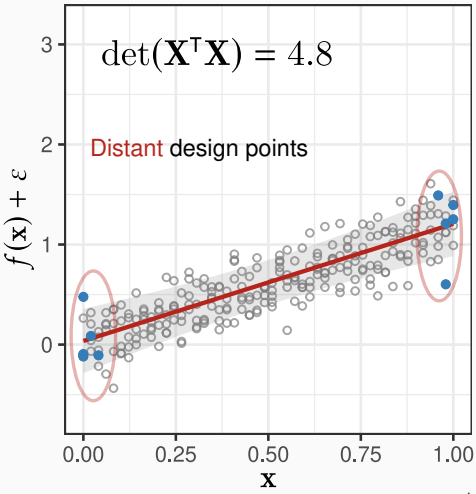
Components

- $\mathbf{X}_{n \times p}$: design matrix
- $\mathbf{x}_{1 \times n} \in \mathbf{X}$: factor columns
- $x_1, \ldots, x_p \in \mathbf{x}$: chosen factor levels

Examples

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

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



Optimal Design: Parsimony

Designs

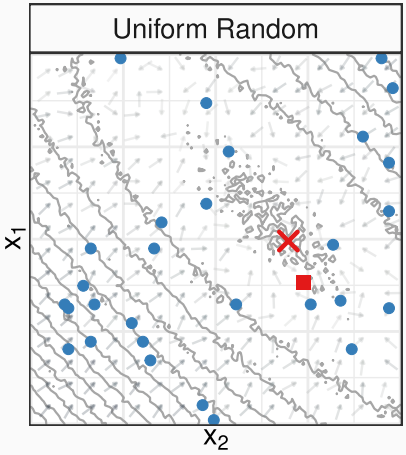
- Building surrogates within a constrained budget
- Exploiting known search space structure
- Testing modeling hypotheses

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

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

Sampling with Different Models

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



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

Optimal Design: Parsimony

Designs

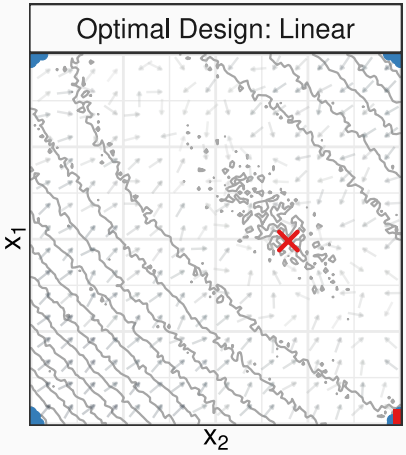
- Building surrogates within a constrained budget
- Exploiting known search space structure
- Testing modeling hypotheses

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

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

Sampling with Different Models

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



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

Optimal Design: Parsimony

Designs

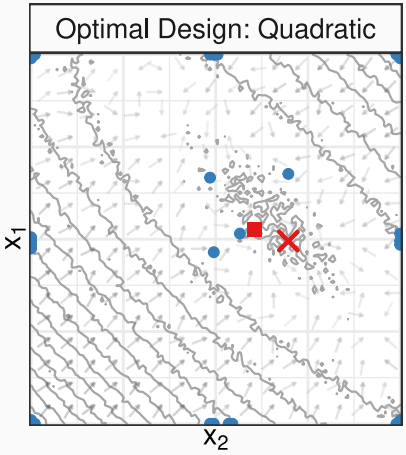
- Building surrogates within a constrained budget
- Exploiting known search space structure
- Testing modeling hypotheses

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

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

Sampling with Different Models

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



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

Optimal Design: Parsimony

Designs

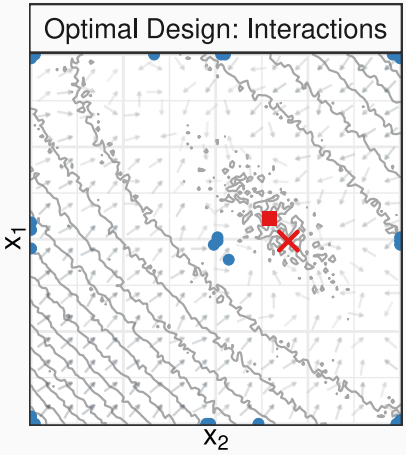
- Building surrogates within a constrained budget
- Exploiting known search space structure
- Testing modeling hypotheses

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

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

Sampling with Different Models

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



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

Optimal Design: Parsimony

Designs

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

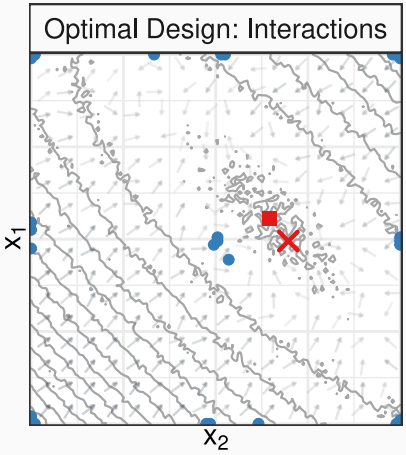
Maximizing $\det(\mathbf{X}^T\mathbf{X})$ by Swapping Rows

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

Best design is independent of measurements

Sampling with Different Models

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



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

Interpreting Significance with Analysis of Variance

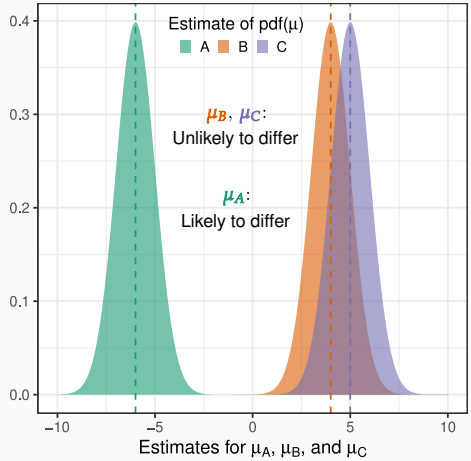
Analysis of Variance (ANOVA)

- Identify which factors and levels are significant

Steps

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

One-Way ANOVA for Levels A, B, C



Interpreting Significance with Analysis of Variance

Analysis of Variance (ANOVA)

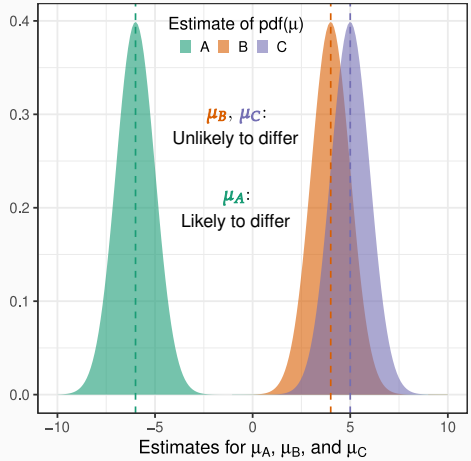
- Identify which factors and levels are significant

Steps

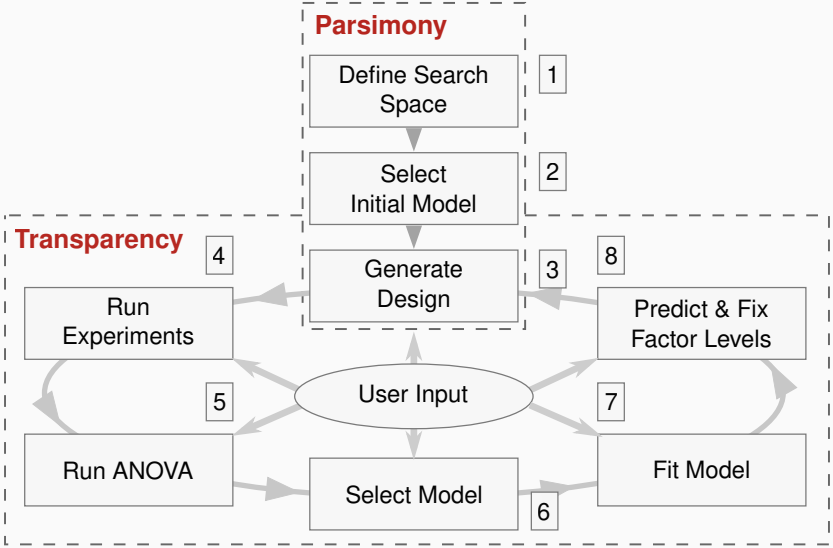
- Group observations by factor and factor levels
- Estimate distributions for each group mean μ
- Run F-tests for significance of differences between means

Enables refining initial hypotheses

One-Way ANOVA for Levels A, B, C



A Transparent and Parsimonious Approach to Autotuning



Application: OpenCL GPU Laplacian Kernel

Edge Detection with the Laplacian

The OpenCL Kernel

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

Search Space with 10^4 Valid Configurations

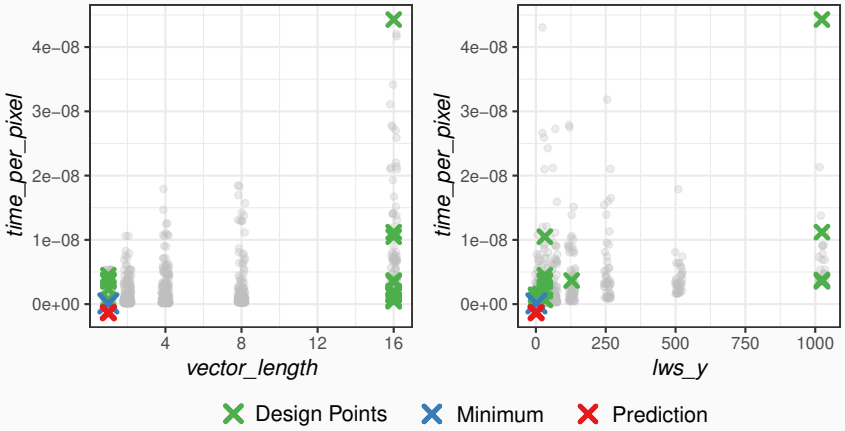
Factor	Levels	Short Description
vector_length	$2^0, \ldots, 2^4$	Size of vectors
load_overlap	*true*, *false*	Load overlaps in vectorization
temporary_size	$2, 4$	Byte size of temporary data
elements_number	$1, \ldots, 24$	Size of equal data splits
y_component_number	$1, \ldots, 6$	Loop tile size
threads_number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

Performance Metric and Starting Model

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

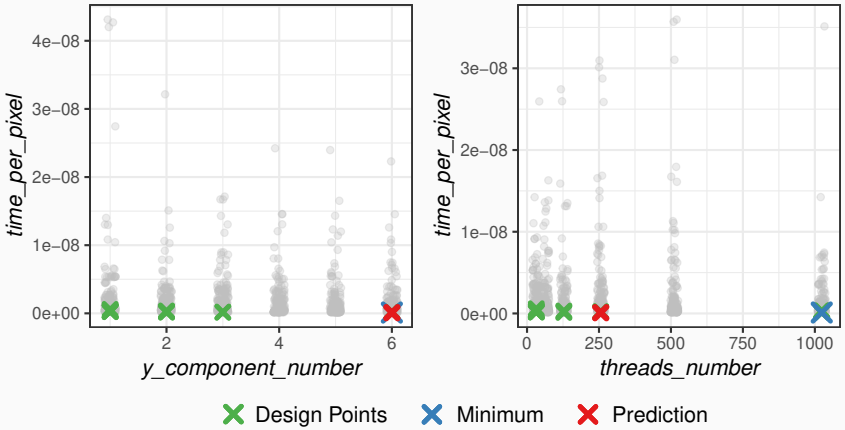
Sequential Approach to Optimization: Refining the Model

Step 1 Fix ***vector*** and ***lws_y***

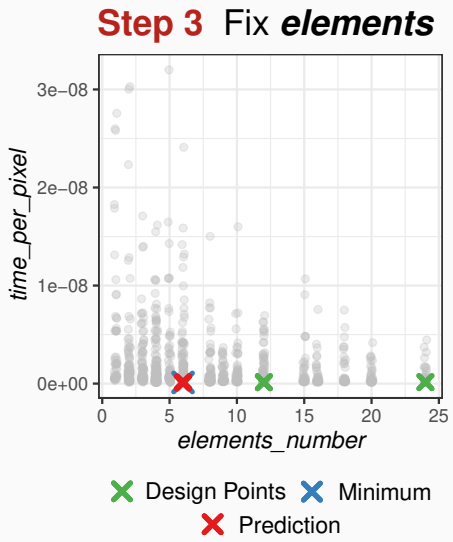


Sequential Approach to Optimization: Refining the Model

Step 2 Fix *y_cmp.* and *threads*

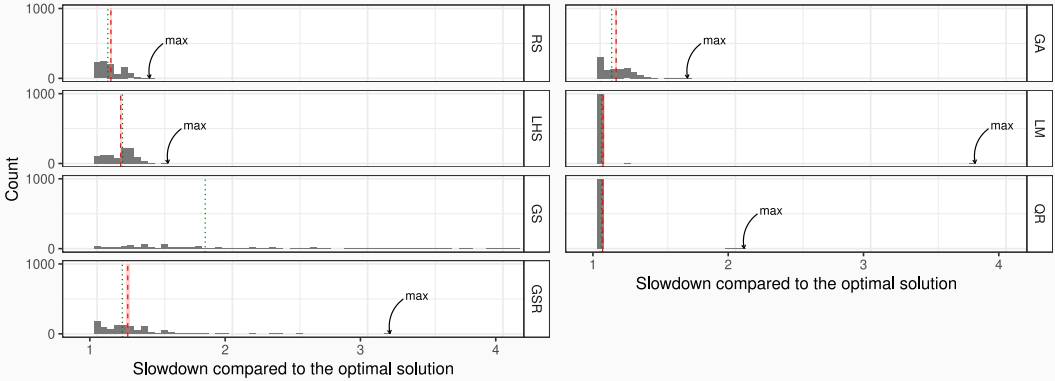


Sequential Approach to Optimization: Transparency



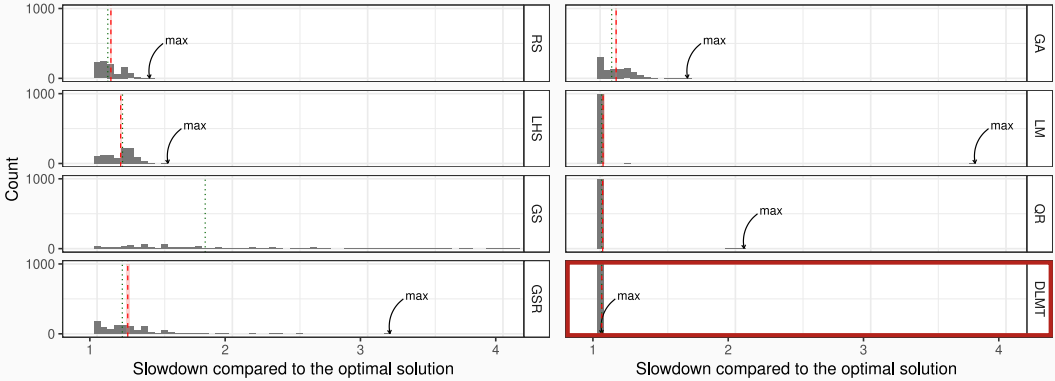
Step	Term	Sum Sq.	F-value	p(>F)
1st	*y_component_number*	2.1×10^{-18}	7.3×10^{-1}	4.1×10^{-1}
	1/y_component_number	4.4×10^{-18}	1.6×10^{0}	2.4×10^{-1}
	vector_length	1.3×10^{-17}	4.4×10^{0}	4.7×10^{-2}
	lws_y	6.9×10^{-17}	2.4×10^{1}	3.5×10^{-4}
	1/lws_y	1.8×10^{-17}	6.2×10^{0}	2.8×10^{-2}
	load_overlap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

Results: 1000 Repetitions with a Budget of 120 Measurements



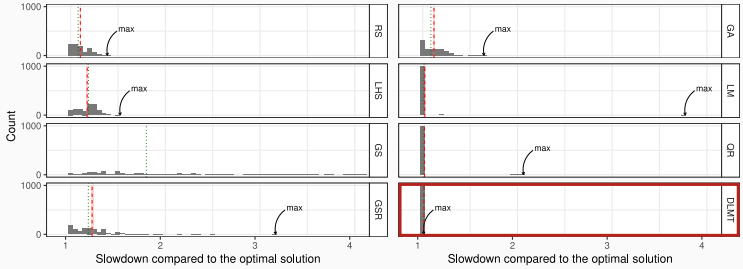
RS: Random Sampling, LHS: Latin Hypercube Sampling, GS: Greedy Search,
GSR: Greedy Search w. Restart, GA: Genetic Algorithm, LM: Linear Model, QR: Quantile Regression

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



	Slowdown			Budget	
Method	*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
D-Opt., Linear Model w. Transform (DLMT)	**1.01**	**1.01**	**1.01**	**54.84**	**56**

Application: Search Problems in Automatic Performance Tuning (SPAPT)

SPAPT Kernels

- 16 problems on multiple HPC domains
- Generated by ORIO
- 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

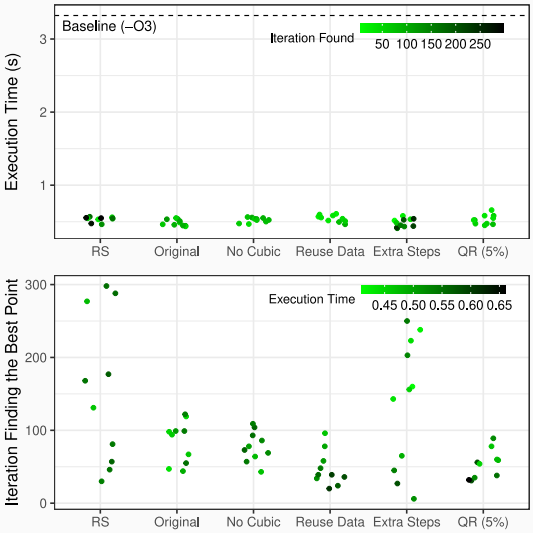
Search Spaces

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

Performance Metric and Starting Model

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

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 1st and 2nd 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
- What is the most effective?

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
- What is the most effective?

Sequential and Incremental Approach

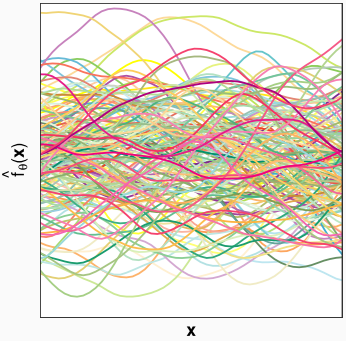
- Definitive search space restrictions
- Experiments and improvements by batch
- Rigid models

Active Learning with Gaussian Processes

Balance exploitation of structure with unrestricted exploration

Applying Active Learning with Gaussian Processes

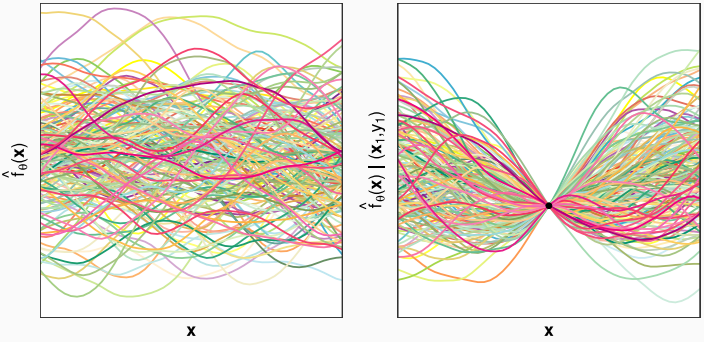
Sampling Functions with Gaussian Process Regression



Gaussian Process Surrogates

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

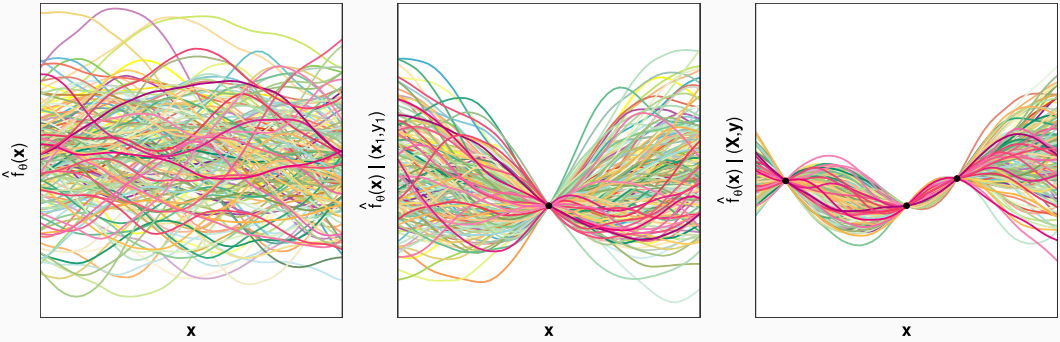
Sampling Functions with Gaussian Process Regression



Gaussian Process Surrogates

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

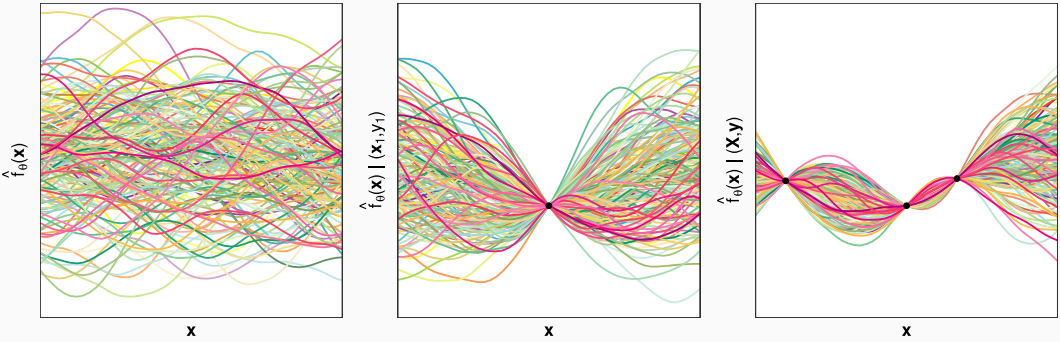
Sampling Functions with Gaussian Process Regression



Gaussian Process Surrogates

- $f : \mathcal{X} \to \mathbb{R}$
- Model: $f(\mathbf{x}) \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$
- 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



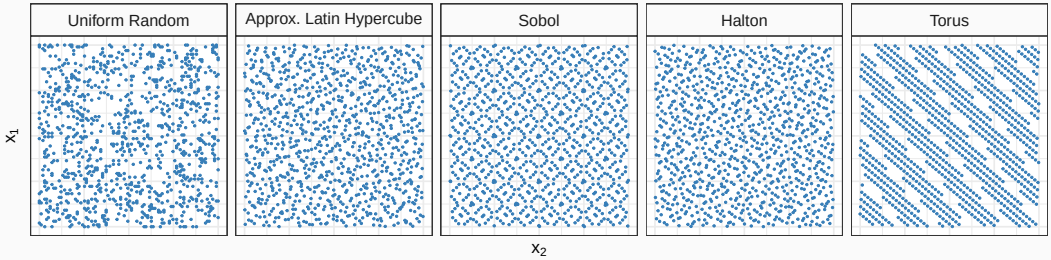
Gaussian Process Surrogates

- $f : \mathcal{X} \to \mathbb{R}$
- Model: $f(\mathbf{x}) \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$
- 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 X?

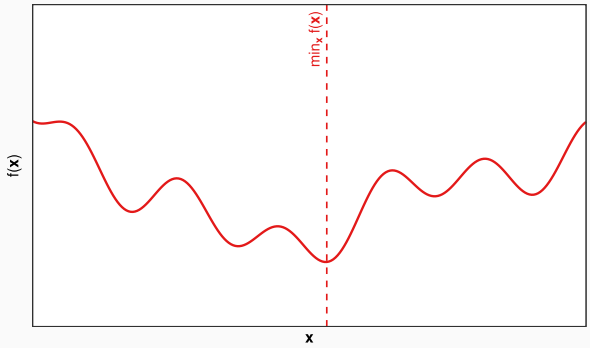
- Minimizing f
- Building an accurate surrogate
- Doing both?

Space-filling Designs



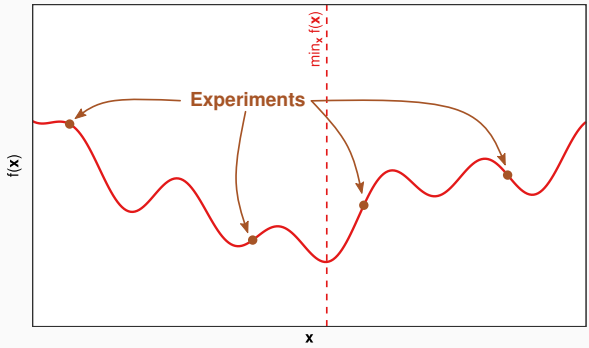
- Curse of dimensionality for sampling:
  - Most sampled points will be on the "shell"
- LHS: Partition and then sample, need to optimize later
- Low-discrepancy: deterministic space-filling sequences

Expected Improvement: Balancing Exploitation and Exploration



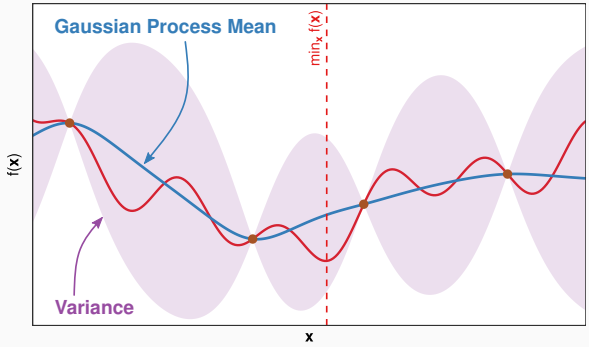
How to decide where to measure next?

Expected Improvement: Balancing Exploitation and Exploration



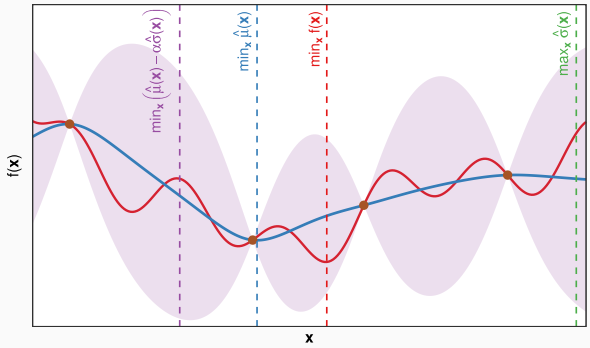
How to decide where to measure next?

Expected Improvement: Balancing Exploitation and Exploration



How to decide where to measure next?

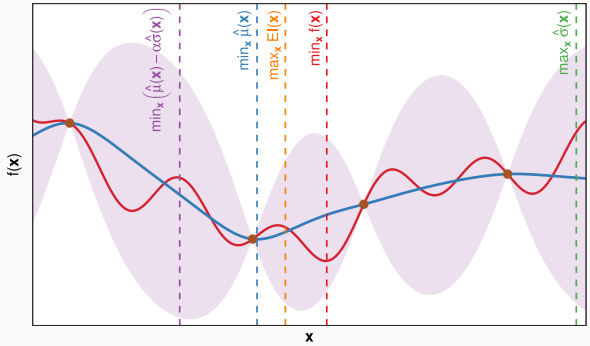
Expected Improvement: Balancing Exploitation and Exploration



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)

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

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: GPU Laplacian and SPAPT

- GPR was applied to these problems too
- Quickly Present Results Here Too
- GPR is good too, but the simpler model is more consistent

Application: Quantization for Convolutional Neural Networks

Search Space, Constraints, and Performance Metrics

- Comparing with a Reinforcement Learning approach in the original paper
- ImageNet

Results

Interpreting the Optimization

- Sobol indices, inconclusive

Discussion

- Low-discrepancy sampling in high dimension
- Constraints complicate exploration
- Multi-objective optimization
- A more complex method usually produces less interpretable results, but not always achieves better optimizations

Conclusion

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

Reproducibility of Performance Tuning Experiments

- Redoing all the work for different problems
- Complementary approaches:
  - Completely evaluate small sets of a search space
  - Collaborative optimizing for different architectures, problems

Key Discussions

Curse of Dimensionality for Autotuning Problems

- Implications for Sampling and Learning
- Space-filling helps, but does not solve
- Constraints

Which method to use?

- Design of Experiments for transparency and parsimony when building and interpreting statistical models
- Linear models for simpler spaces and problems
- Gaussian Process Surrogates for more complex situations

It is often unclear if there is something to find

- Abundance of local optima
- Is there a global optimum, is it "hidden"?
  - How to find it, if so? (can learning do it?)
- What is the most effective level of abstraction for optimizing a program?
  - Compiler, kernel, machine, model, dependencies?
- When to stop?

Conclusion

Toward Transparent and Parsimonious Methods for Automatic Performance Tuning

Pedro Bruel
phrb@ime.usp.br
July 9 2021