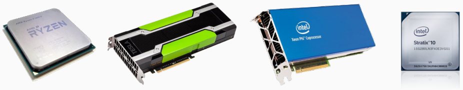
Autotuning: A Design of Experiments Approach

Pedro Bruel
phrb@ime.usp.br

July 3rd, 2018

Autotuning: Optimizing Program Configurations

Architectures for High Performance Computing



How to write efficient code for each of these?

Autotuning

The process of automatically finding a configuration of a program that optimizes an objective

Configurations

- Program Configuration
  - Algorithm, block size, . . .
- Source code transformation
  - Loop unrolling, tiling, rotation . . .
- Compiler configuration
  - −O2, vectorization, . . .
- . . .

Objectives

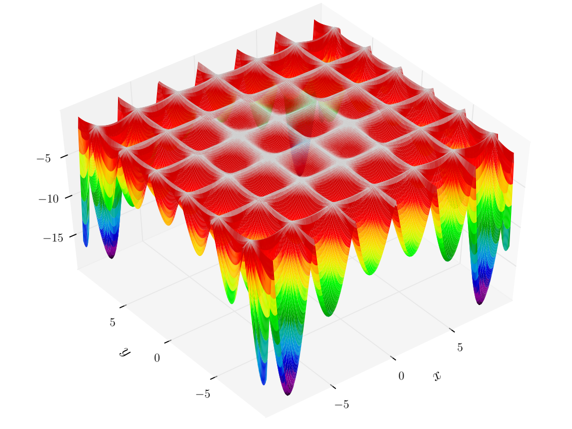
- Execution time
- Memory & power consumption
- . . .

Search Spaces

Represent the effect of all possible configurations on the objectives

Issues

- Exponential Growth
- Geometry
- Measurement Time



Hölder Table Function

Autotuning: Multiple Approaches

Popular Approaches

- Exhaustive
- Meta-Heuristics
- Machine Learning

System	Domain	Approach
ATLAS	Dense Linear Algebra	Exhaustive
INSIEME	Compiler	Genetic Algorithm
Active Harmony	Runtime	Nelder-Mead
ParamILS	Domain-Agnostic	Stochastic Local Search
OPAL	Domain-Agnostic	Direct Search
OpenTuner	Domain-Agnostic	Ensemble
MILEPOST GCC	Compiler	Machine Learning
Apollo	GPU kernels	Decision Trees

Main Issues

These approaches assume:

- A large number of function evaluations
- Seach space "smoothness"
- Good solutions are reachable

After optimizing:

- Learn nothing about the search space
- Can't explain why optimizations work

Applying Design of Experiments to Autotuning

Our Approach

Using efficient experimental designs to overcome issues related to exponential growth, geometry, and measurement time

Design Requirements

- Support a large number of factors (Exponential Growth)
- Support numerical and categorical factors (Geometry)
- Minimize function evaluations (Measurement Time)

Main Candidate: D-Optimal Designs

- Require an initial model
- Minimize variance of estimators
- Support mixed-level factors
- Constructed using search algorithms

Example

- Factors & Levels:
  $\mathbf{X} = \{x_1 = \{1, 2, 3\}, x_2 = \{1, 2, 3\}\}$
- Model: $\mathbf{Y} = \mathbf{X}\beta + \eta$
- Minimize: D-optimality
- Candidate set: Full factorial
- Construction method: Fedorov's algorithm

Source code

```
library(AlgDesign)
full_factorial <- gen.factorial(c(3, 3),
                    factors = c(1, 2))
output <- optFederov(~., full_factorial,
                   nTrials = 5)
```

Output

```
$D
[1] 0.2

$A
[1] 15

$Ge
[1] 0.2

$Dea
[1] 0.018

$design
   2   3   4   6   7
X1 "2" "3" "1" "3" "1"
X2 "1" "1" "2" "2" "3"

$rows
[1] 2 3 4 6 7
```

Example: A Laplacian GPU Kernel

Search Space

Parameters	Values
vector_length	1, 2, 4, 8, 16
load_overlap	true, false
temporary_size	2, 4
elements_number	1 - 24
y_component_number	1 - 6
threads_number	32, 64, 128, 256, 512, 1024
lws_y	1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024

Objective

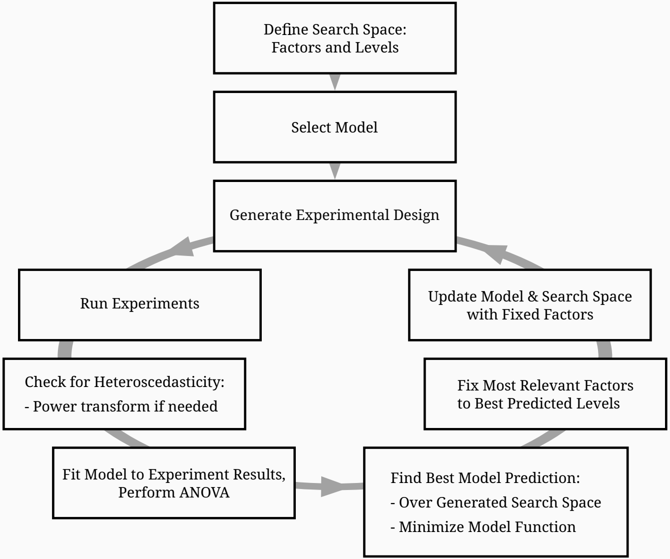
Minimize the time to compute each pixel:

- *time_per_pixel*

Initial Model

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

Strategy



Define Search Space:
Factors and Levels
Select Model
Generate Experimental Design
Run Experiments
Check for Heteroscedasticity:
- Power transform if needed
Fit Model to Experiment Results,
Perform ANOVA
Find Best Model Prediction:
- Over Generated Search Space
- Minimize Model Function
Fix Most Relevant Factors
to Best Predicted Levels
Update Model & Search Space
with Fixed Factors

Loading Data

```
library(AlgDesign)
library(car)
library(dplyr)

complete_data = read.csv("../data/search_space.csv", header = TRUE)
str(complete_data)
```

```
'data.frame':^^I23120 obs. of  9 variables:
 $ elements_number   : int  3 2 4 2 2 2 2 4 4 3 ...
 $ y_component_number: int  3 2 1 1 1 2 2 2 4 1 ...
 $ vector_length     : int  4 1 4 1 8 2 1 8 16 4 ...
 $ temporary_size    : int  4 2 2 2 2 2 4 4 2 4 ...
 $ vector_recompute  : Factor w/ 1 level "true": 1 1 1 1 1 1 1 1 1 1 ...
 $ load_overlap      : Factor w/ 2 levels "false","true": 2 1 2 1 2 2 1 2 2 2 ...
 $ threads_number    : int  64 128 64 256 128 128 128 64 128 32 ...
 $ lws_y             : int  64 1 32 64 32 8 2 2 128 32 ...
 $ time_per_pixel    : num  1.11e-08 1.58e-10 2.34e-09 1.39e-09 3.40e-09 ...
```

Configuration

```
used <- 0
budget <- 120

iterations <- 1

factors = c("elements_number", "y_component_number",
            "vector_length", "temporary_size",
            "load_overlap", "threads_number",
            "lws_y")

data <- complete_data[, c(factors, "time_per_pixel")]
```

Step 1: D-Optimal Design

```
output <- optFederov(~ y_component_number + I(1 / y_component_number) +
                       vector_length + lws_y + I(1 / lws_y) +
                       load_overlap + temporary_size +
                       elements_number + I(1 / elements_number) +
                       threads_number + I(1 / threads_number),
                     data,
                     nTrials = 24)

federov_design <- data[output$rows, ]
experiments <- output$rows

str(federov_design)
```

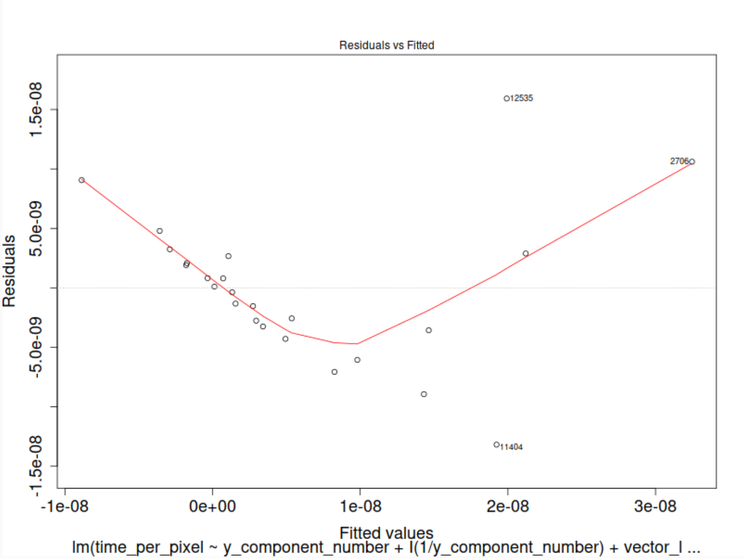
```
'data.frame':^^I24 obs. of  8 variables:
 $ elements_number   : int  1 4 4 1 4 3 6 1 2 24 ...
 $ y_component_number: int  1 1 1 1 1 3 6 1 2 6 ...
 $ vector_length     : int  1 1 1 16 1 16 16 1 1 16 ...
 $ temporary_size    : int  2 4 2 4 2 4 2 4 2 2 ...
 $ load_overlap      : Factor w/ 2 levels "false","true": 2 2 1 2 1 2 1 1 2 2 ...
 $ threads_number    : int  256 128 32 32 128 256 128 1024 1024 32 ...
 $ lws_y             : int  1 32 1 32 32 1 1 1024 32 32 ...
 $ time_per_pixel    : num  2.31e-10 1.21e-09 3.48e-10 4.31e-08 1.21e-09 ...
```

Step 1: Regression

```
regression <- lm(time_per_pixel ~ y_component_number + I(1 / y_component_number) +
                                  vector_length + lws_y + I(1 / lws_y) +
                                  load_overlap + temporary_size +
                                  elements_number + I(1 / elements_number) +
                                  threads_number + I(1 / threads_number),
                 data = federov_design)
summary.aov(regression)
```

```
                        Df    Sum Sq   Mean Sq F value Pr(>F)
y_component_number       1 3.510e-17 3.510e-17   0.465 0.5082
I(1/y_component_number)  1 7.500e-18 7.500e-18   0.100 0.7574
vector_length            1 6.135e-16 6.135e-16   8.119 0.0146 *
lws_y                    1 2.899e-16 2.899e-16   3.836 0.0738 .
I(1/lws_y)               1 3.175e-16 3.175e-16   4.202 0.0629 .
load_overlap             1 7.000e-17 7.000e-17   0.926 0.3549
temporary_size           1 6.770e-17 6.770e-17   0.896 0.3626
elements_number          1 9.380e-17 9.380e-17   1.242 0.2870
I(1/elements_number)     1 1.707e-16 1.707e-16   2.259 0.1587
threads_number           1 2.756e-16 2.756e-16   3.648 0.0803 .
I(1/threads_number)      1 2.321e-16 2.321e-16   3.072 0.1051
Residuals               12 9.067e-16 7.560e-17
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Step 1: Heteroscedasticity



Step 1: Power Transform

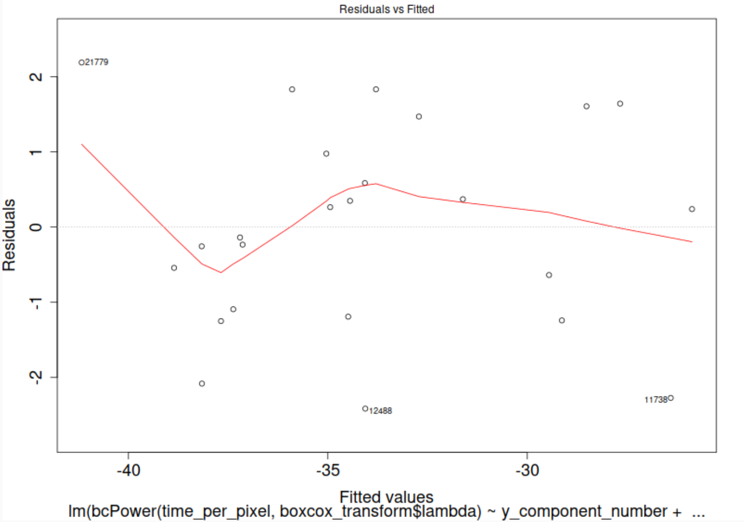
```
boxcox_transform <- powerTransform(time_per_pixel ~ y_component_number +
                                      I(1 / y_component_number) +
                                      vector_length + lws_y + I(1 / lws_y) +
                                      load_overlap + temporary_size +
                                      elements_number + I(1 / elements_number) +
                                      threads_number + I(1 / threads_number),
                                   data = federov_design)

regression <- lm(bcPower(time_per_pixel, boxcox_transform$lambda) ~ y_component_number +
                                I(1 / y_component_number) +
                                vector_length + lws_y + I(1 / lws_y) +
                                load_overlap + temporary_size +
                                elements_number + I(1 / elements_number) +
                                threads_number + I(1 / threads_number),
                             data = federov_design)
```

Step 1: Power Transform Results

```r
summary.aov(regression)
```

```
                         Df Sum Sq Mean Sq F value   Pr(>F)    
y_component_number        1   5.28    5.28   1.489  0.24574    
I(1/y_component_number)   1   6.83    6.83   1.927  0.19031    
vector_length             1 153.99  153.99  43.466 2.56e-05 ***
lws_y                     1 144.51  144.51  40.791 3.47e-05 ***
I(1/lws_y)                1  37.32   37.32  10.534  0.00701 ** 
load_overlap              1   0.11    0.11   0.030  0.86442    
temporary_size            1   3.52    3.52   0.993  0.33878    
elements_number           1   2.80    2.80   0.789  0.39187    
I(1/elements_number)      1   5.07    5.07   1.432  0.25462    
threads_number            1  40.03   40.03  11.299  0.00566 ** 
I(1/threads_number)       1   6.72    6.72   1.898  0.19349    
Residuals                12  42.51    3.54                     
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Step 1: Power Transform Results



Step 1: Predicting Best Point and Pruning Data

```
predicted_best <- data[predict(regression, data) == min(predict(regression, data)), ]
best <- complete_data[complete_data$time_per_pixel == min(complete_data$time_per_pixel), ]
best_row <- rownames(best)

predicted_best$slowdown <- predicted_best$time_per_pixel / best$time_per_pixel
predicted_best$method <- rep("DOPTaov_t", nrow(predicted_best))
predicted_best$point_number <- rep(used, nrow(predicted_best))
predicted_best$vector_recompute <- rep("true", nrow(predicted_best))

data <- complete_data[complete_data$vector_length == predicted_best$vector_length &
                      complete_data$lws_y == predicted_best$lws_y, c(factors,
                             "time_per_pixel")]
scaled_data <- data[, factors]
```

Step 1: Predicting Best Point and Pruning Data

```
predicted_best
str(data)
```

```
      elements_number y_component_number vector_length temporary_size
15827              12                  3             1              2
      load_overlap threads_number lws_y time_per_pixel slowdown    method
15827        false           1024     1   2.380298e-10 2.043151 DOPTaov_t
      point_number vector_recompute
15827           24             true
'data.frame':^^I576 obs. of  8 variables:
 $ elements_number   : int  2 4 4 1 3 3 3 4 4 4 ...
 $ y_component_number: int  2 1 1 1 1 3 1 2 2 1 ...
 $ vector_length     : int  1 1 1 1 1 1 1 1 1 1 ...
 $ temporary_size    : int  2 4 2 4 4 2 2 4 4 4 ...
 $ load_overlap      : Factor w/ 2 levels "false","true": 1 1 1 1 1 1 2 1 2 1 ...
 $ threads_number    : int  128 64 128 256 256 128 512 64 64 512 ...
 $ lws_y             : int  1 1 1 1 1 1 1 1 1 1 ...
 $ time_per_pixel    : num  1.58e-10 3.03e-10 3.01e-10 2.36e-10 3.33e-10 ...
```

We can now continue with the other steps:

```
predicted_best


      elements_number y_component_number vector_length temporary_size
17258               6                  6             1              2
      vector_recompute load_overlap threads_number lws_y time_per_pixel
17258             true         true            256     1     1.1792e-10
      point_number    method slowdown
17258           55 DOPTaov_t 1.012177
```

Comparing Strategies

	Min.	1st Qu.	Median	Mean	3rd Qu.	Max.	Mean Pt.	Max Pt.
RS	1.00	1.03	1.08	1.10	1.18	1.39	120.00	125.00
LHS	1.00	1.09	1.19	1.17	1.24	1.52	98.92	125.00
GS	1.00	1.35	1.80	6.46	6.31	124.76	22.17	106.00
GSR	1.00	1.07	1.19	1.23	1.33	3.16	120.00	120.00
GA	1.00	1.02	1.09	1.12	1.19	1.65	120.00	120.00
LM	1.01	1.01	1.01	1.02	1.01	3.77	119.00	119.00
LMB	1.01	1.01	1.03	1.03	1.03	3.80	104.81	106.00
LMBT	1.01	1.01	1.03	1.03	1.03	1.98	104.89	106.00
RQ	1.01	1.01	1.01	1.02	1.01	2.06	119.00	119.00
DOPT	1.38	1.64	1.64	1.68	1.64	2.91	120.00	120.00
DLM	1.01	1.01	1.01	1.01	1.01	1.08	54.85	56.00
DLMT	1.01	1.01	1.01	1.01	1.01	1.01	54.84	56.00

Table 1: Summary statistics

Resources

The code, slides and images are hosted at GitHub:

`github.com/phrb/presentations/tree/master/demo_doptanova_lig`

Autotuning: A Design of Experiments Approach

Pedro Bruel
phrb@ime.usp.br

July 3rd, 2018