Autotuning under Tight Budget Constraints: A Transparent Design of Experiments Approach

Pedro Bruel*†, Arnaud Legrand*, Brice Videau*, Jean-Marc Vincent*, and Alfredo Goldman†
*University of Grenoble Alpes, CNRS, INRIA, LIG - Grenoble, France
Email: {arnaud.legrand, brice.videau, jean-marc.vincent}@imag.fr
†University of São Paulo - São Paulo, Brazil
Email: {phrb, gold}@ime.usp.br

***Abstract*—A large quantity of resources is spent writing, porting, and optimizing scientific and industrial High Performance Computing applications. Autotuning techniques have become therefore fundamental to lower the costs of leveraging the improvements on execution time and power consumption provided by the latest software and hardware platforms. Despite this, the most popular autotuning techniques still require a large budget of costly experimental measurements to provide good results while still not providing exploitable knowledge about the problem after optimization. In this paper we present a user-transparent autotuning technique based on Design of Experiments that is capable of operating under tight budget constraints by significantly reducing the amount of measurements needed to find good optimizations. Our approach also enable users to make informed decisions on what optimizations to pursue and when to stop optimizing. We present experimental evaluations of our approach and show that, leveraging user decisions, it is capable of finding the global optimum of a GPU Laplacian kernel optimization using half of the measurement budget used by other common autotuning techniques. We also show that our approach is capable of using generic performance models to decrease the measurement budget needed to find speedups of up to 50×, when compared to random sampling, for some applications from a more comprehensive autotuning benchmark.**

I. Introduction

Optimizing code for objectives such as performance and power consumption is fundamental to the success and cost effectiveness of industrial and scientific endeavors in High Performance Computing. A considerable amount of highly specialized time and effort is spent in porting and optimizing code for GPUs, FPGAs and other hardware accelerators. Experts are also needed to leverage bleeding edge software improvements in compilers, languages, libraries and frameworks. The automatic configuration and optimization of High Performance Computing applications, or *autotuning*, is a technique effective in decreasing the cost and time needed to adopt efficient hardware and software. Typical targets for autotuning include algorithm selection, source-to-source transformations and compiler configuration.

Autotuning can be studied as a search problem, where the objective is to minimize single or multiple software of hardware metrics. The exploration of the search spaces defined by configurations and optimizations present interesting challenges to search strategies. These search spaces grow exponentially with the number of considered configuration parameters and their possible values. They are also difficult to extensively explore due to the often prohibitive costs of hardware utilization and program compilation and execution times. Developing autotuning strategies capable of producing good optimizations while minimizing resource utilization is therefore essential. The capability of acquiring knowledge about an optimization problem is also a desired feature of an autotuning strategy, since this knowledge can decrease the cost of subsequent optimizations of the same application or for the same hardware.

It is common and usually effective to use search meta-heuristics such as genetic algorithms and simulated annealing in autotuning. These strategies usually attempt to exploit local properties and are not capable of fully exploiting global search space structures. They are also not much more effective in comparison with a naive uniform random sample of the search space [1], [2], and usually rely on a large number of measurements and frequent restarts to achieve good performance improvements. Search strategies based on gradient descent also are commonly used in autotuning and rely on a large number of measurements. Their effectiveness diminishes additionally in search spaces with complex local structures. Completely automated machine learning autotuning strategies are effective in building models for predicting important optimization parameters, but still rely on a sizable data set for training. Large data sets are fundamental to strategies based on machine learning since they select models from a generally very large class.

Search strategies based on meta-heuristics, gradient descent and machine learning require a large number of measurements to be effective, and are usually incapable of providing knowledge about search spaces to users. At the end of each autotuning session it is difficult to decide if and where further exploration is warranted, and impossible to know which parameters are responsible for the observed improvements. After exploring a search space, it is impossible to confidently deduce its global properties since its was explored with unknown biases.

In this paper we propose an autotuning strategy that leverages existing expert and approximate knowledge about a problem in the form of a performance model, and refines this initial model iteratively using empirical performance evaluations, statistical analysis and user input. Our strategy puts a heavy weight on decreasing the costs of autotuning by using efficient

Design of Experiments strategies to minimize the number of experiments needed to find good optimizations. Each optimization iteration uses *Analysis of Variance* (ANOVA) to help identify the relative significance of each configurable parameter to the performance observations. An architecture- and problem-specific performance model is built iteratively and with user input, enabling informed decisions on which regions of the search space are worth exploring.

We present the performance of our approach on a Laplacian Kernel for GPUs where the search space, global optimum and performance model approximation are known. The experimental budget on this application were tightly constrained. The speedups achieved and the budget utilization of our approach on this setting motivated a more comprehensive performance evaluation. We chose the *Search Problems in Automatic Performance Tuning* (SPAPT) [3] benchmark for this evaluation, where our approach was able to find speedups of over 50× for some SPAPT applications, finding speedups better than random sampling in some scenarios. Despite using generic performance models for every SPAPT application, our approach was able to significantly decrease the budget used to find performance improvements.

The rest of this paper is organized as follows. Section II presents related work on source-to-source transformation, which is the main optimization target in SPAPT problems, on autotuning systems and on search space exploration strategies. Section IV presents a detailed description of the implementation of our approach and its background. It discusses the Design of Experiments concepts we incorporate, and the ANOVA and linear regression algorithms we use in analysis steps. Section V presents our results with the GPU Laplacian Kernel and the SPAPT benchmark. Section VI discusses our conclusions and future work.

II. Background

A. Source-to-source Transformation

B. Autotuning

John Rice's Algorithm Selection framework [4] is the precursor of autotuners in various problem domains. In 1997, the PHiPAC system [5] used code generators and search scripts to automatically generate high performance code for matrix multiplication. Since then, systems approached different domains with a variety of strategies. Dongarra *et al.* [6] introduced the ATLAS project, that optimizes dense matrix multiplication routines. The OSKI [7] library provides automatically tuned kernels for sparse matrices. The FFTW [8] library provides tuned C subroutines for computing the Discrete Fourier Transform. Periscope [9] is a distributed online autotuner for parallel systems and single-node performance. In an effort to provide a common representation of multiple parallel programming models, the INSIEME compiler project [10] implements abstractions for OpenMP, MPI and OpenCL, and generates optimized parallel code for heterogeneous multi-core architectures.

A different approach is to combine generic search algorithms and problem representation data structures in a single system that enables the implementation of autotuners for different domains. The PetaBricks [11] project provides a language, compiler and autotuner, enabling the definition and selection of multiple algorithms for the same problem. The ParamILS framework [12] applies stochastic local search algorithms to algorithm configuration and parameter tuning. The OpenTuner framework [13] provides ensembles of techniques that search the same space in parallel, while exploration is managed by an implementation of a solver of the multi-armed bandit problem.

C. Search Space Exploration Strategies

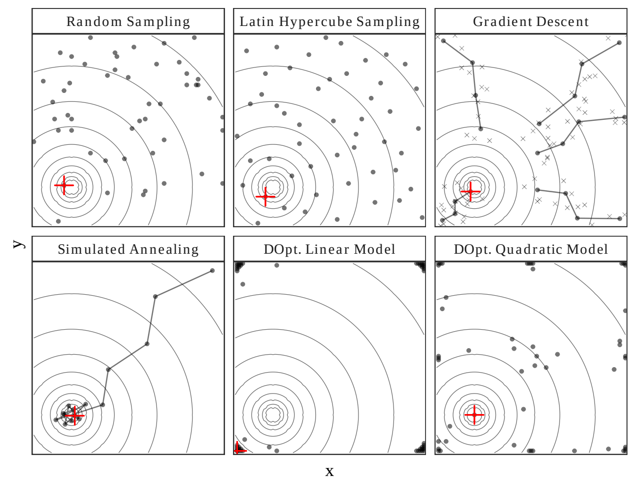

Figure 1: Exploration of the search space defined by $x^2 + y^2$, using a fixed budget of 50 points

III. Design of Experiments

An *experimental design* determines a selection of experiments whose objective is to identify the relationships between *factors* and *responses*. While factors and responses can refer to different concrete entities in other domains, in computer experiments factors can be configuration parameters for algorithms and compilers, for example, and responses can be the execution time or memory consumption of a program. Each possible value of a factor is called a *level*. The *effect* of a factor on the measured response, without its *interactions* with other factors, is the *main effect* of that factor. Experimental designs are constructed with objectives such as identifying the main effects and building an analytical model for the response.

In this Section we use an example of *Screening*, an efficient but limited technique for identifying main effects, to present the assumptions of a traditional Design of Experiments methodology. We also discuss some techniques for the construction of efficient designs for factors with different numbers and types of levels, and present *D-Optimal* designs, the technique we used in the approach presented in this paper.

A. Screening & Plackett-Burman Designs

Screening designs are used to identify the main effects of 2-level factors in the initial stages of studying a problem. Interactions are not considered at this stage, and screening designs are usually small. Identifying main effects early enables focusing on a smaller set of factors on subsequent more detailed experiments. A specially efficient design construction technique for screening designs was presented by Plackett and Burman [14] in 1946. Despite having strong restrictions on the number of factors, Plackett-Burman designs enable the identification of main effects of n factors with $n+1$ experiments.

$$\mathbf{Y} = \boldsymbol{\beta}\mathbf{X} + \epsilon$$

Figure 2: Linear model assumed in main-effect analysis of screening designs

Assuming a linear relationship between factors and the response is fundamental for the analysis of variance using a Plackett-Burman design. For the following example, consider the linear relationship presented in Figure 2, where ϵ is the error term, $\mathbf{Y}$ is the observed response, $\mathbf{X} = (1, x_1, \dots, x_n)$ is the set of n 2-level factors, and $\boldsymbol{\beta} = (\beta_0, \dots, \beta_n)$ is the set with the *intercept* β_0 and the corresponding *model coefficients*.

We now present an example to illustrate the screening methodology. Suppose we wish to minimize a performance metric Y of a problem with factors $x_1, \dots, x_8$ assuming values in $[-1, -0.8, -0.6, \dots, 0.6, 0.8, 1]$. Each $y_i \in Y$ is computed using the formula described in Figure 3, but suppose that, for the purpose of this example, they are computed by a very expensive black-box procedure. To efficiently study this problem we decide to construct a Plackett-Burman design, which minimizes the experiments needed to identify relevant factors. The analysis of this design will enable us to decrease the dimension of the problem.

$$y_i = \begin{array}{c} \boldsymbol{\beta}^\top \\ \begin{pmatrix} 0 \\ -1.5 \\ 1.3 \\ 3.1 \\ -1.4 \\ 1.35 \\ 1.6 \end{pmatrix} \end{array} \cdot \begin{array}{c} \mathbf{X}_i \\ \begin{pmatrix} 1 \\ x_1 \\ x_3 \\ x_5 \\ x_7 \\ x_8^2 \\ x_1 x_3 \end{pmatrix} \end{array} + \epsilon$$

Figure 3: Real model used to obtain the data on Table I

Table I presents the Plackett-Burman design we generated for our problem. In this design we have the 8 2-level factors $x_1, \dots, x_8$, and the observed response $\mathbf{Y}$. As is common when constructing screening designs, we had to add 3 "dummy" factors $d_1, \dots, d_3$ to complete the 12 columns needed to construct a Plackett-Burman design for 8 factors.

Table I: Randomized Plackett-Burman design for factors $x_1, \dots, x_8$, using 12 experiments and "dummy" factors $d_1, \dots, d_3$, and computed response $\mathbf{Y}$

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	d_1	d_2	d_3	Y
1	-1	1	1	1	-1	-1	-1	1	-1	1	13.74
-1	1	-1	1	1	-1	1	1	1	-1	-1	10.19
-1	1	1	-1	1	1	1	-1	-1	-1	1	9.22
1	1	-1	1	1	1	-1	-1	-1	1	-1	7.64
1	1	1	-1	-1	-1	1	-1	1	1	-1	8.63
-1	1	1	1	-1	-1	-1	1	-1	1	1	11.53
-1	-1	-1	1	-1	1	1	-1	1	1	1	2.09
1	1	-1	-1	-1	1	-1	1	1	-1	1	9.02
1	-1	-1	-1	1	-1	1	1	-1	1	1	10.68
1	-1	1	1	-1	1	1	1	-1	-1	-1	11.23
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	5.33
-1	-1	1	-1	1	1	-1	1	1	1	-1	14.79

We use our initial assumption shown in Figure 2 to identify the most relevant factors by performing an ANOVA test. The resulting ANOVA table is shown in Table II, where the *significance* of each factor can be interpreted from the F-test and P($>$ F) values. Table II uses "$*$", as is convention in the `R` language, to represent the significance values for each factor.

We see on Table II that factors (x_3, x_5, x_7, x_8) have at least one "$*$" of significance. For the purpose of this example, this is reason enough to include them in our linear model for the next step. We see that factor x_1 has a significance mark of "$\cdot$", but comparing its F-test and P($>$ F) values we decide that they are fairly smaller than the values of factors that had no significance at all, and we keep this factor. Then, since we want to reduce the dimension of the problem, we decide in this example to not include (x_2, x_4, x_6) in our model due to their low significance.

Table II: Shortened ANOVA table for the fit of the naive model, with significance intervals from the `R` language

	F value	Pr($<$ F)	Significance
x_1	8.382	0.063	$\cdot$
x_2	0.370	0.586	
x_3	80.902	0.003	$**$
x_4	0.215	0.675	
x_5	46.848	0.006	$**$
x_6	5.154	0.108	
x_7	13.831	0.034	$*$
x_8	59.768	0.004	$**$

Moving forward, we will build a linear model using the $(x_1, x_3, x_5, x_7, x_8)$ factors, fit the model using the values of Y we obtained when running our design, and use the coefficients of this fitted model to predict the levels for each factor that minimize the real response. The prediction step will be run using a full factorial combination of the possible values of $(x_1, x_3, x_5, x_7, x_8)$, without running any new experiments. The levels of the selected factors with the best prediction on this data will be the output of this step.

Table III compares the prediction for Y from our linear model with the selected factors $(x_1, x_3, x_5, x_7, x_8)$ with the actual global minimum Y for this problem. Using 12 measurements and a simple linear model, the predicted best value of Y was around $10\times$ larger than the global optimum. Note that the model predicted the correct levels for x_3 and x_5, and almost predicted correctly for x_7. The linear model predicted wrong levels for x_1, perhaps due to this factor's interaction with x_3, and for x_8. Arguably, it would be impossible to predict the correct level for x_8 using this linear model, since we know a quadratic term composes the formula of Y.

Table III: Comparison of the response Y predicted by the linear model and the global minimum Y

	x_1	x_3	x_5	x_7	x_8	Y
Linear Model	-1.0	-1.0	-1.0	1.0	-1.0	-1.046
Global Minimum	1.0	-1.0	-1.0	0.8	0.0	-9.934

We can improve upon this result if we introduce some information about the problem and use a more flexible design construction technique. Next, we will discuss the construction of efficient designs using problem-specific formulas and continue the optimization of our example.

B. *D-Optimal Designs*

The application of Design of Experiments to autotuning problems requires design construction techniques that support factors of different types and number of possible values. Autotuning problems typically combine factors such as binary flags, integer and floating point numerical values, and unordered enumerations of abstract values. Previously, to construct a Plackett-Burman design for our example we had to restrict our factors to the extremes of their levels in the interval $[-1, -0.8, -0.6, \ldots, 0.6, 0.8, 1]$, because such screening designs only support 2-level factors. Doing that makes it impossible to measure the significance of quadratic terms in the model, for example. Next we will continue optimizing our example by constructing *D-Optimal designs* using a more flexible construction technique, that increases the number of levels we can screen for and enables detecting the significance of more complex model terms.

The class of *D-Optimal designs* is the best fit for our requirements of supporting multi-level factors while minimizing the number of experiments. The algorithms for constructing D-Optimal designs are relatively fast, simple, and have few restrictions. To construct a D-Optimal design it is necessary to choose an initial model, which can be done based on previous experiments or on expert knowledge of the problem.

Once a model is selected, algorithmic construction is performed by searching for the set of experiments that minimizes *D-Optimality*, a measure of the *variance* of the *estimators* for the *regression coefficients* associated with the selected model. This search is usually done by swapping experiments from the current candidate set with experiments from a pool of possible experiments, according to certain rules, until some stopping criterion is met. In the example in this Section, as well as in the approach presented in this paper, we use Fedorov's algorithm [15] for constructing D-Optimal designs, implemented in `R` in the `AlgDesign` package.

Going back to our example, suppose that in addition to using our previous screening results we decide to hire an expert in our problem's domain. The expert confirms our initial assumptions that the factor x_1 should be included in our model since it is usually relevant for this kind of problem and has a strong interaction with factor x_3. She also mentions we should replace the linear term for x_8 by a quadratic term for this factor.

Using our previous screening and the domain knowledge provided by our expert, we choose a new performance model and use it to construct a D-Optimal design using Fedorov's algorithm. Since we need enough degrees of freedom to fit our model, we construct the design with 12 experiments shown in Table IV.

Table IV: D-Optimal design constructed for the factors $(x_1, x_3, x_5, x_7, x_8)$ and computed response Y

x_1	x_3	x_5	x_7	x_8	Y
-1.0	-1.0	-1.0	-1.0	-1.0	2.455
1.0	-1.0	-1.0	-1.0	-1.0	-4.881
1.0	-1.0	1.0	-1.0	-1.0	2.128
-1.0	1.0	-1.0	1.0	-1.0	-2.042
-1.0	-1.0	1.0	1.0	-1.0	4.609
1.0	1.0	1.0	1.0	-1.0	4.163
1.0	1.0	-1.0	-1.0	0.0	0.862
-1.0	-1.0	1.0	-1.0	0.0	6.453
-1.0	1.0	1.0	-1.0	0.0	5.703
-1.0	-1.0	-1.0	1.0	0.0	-2.708
1.0	-1.0	-1.0	1.0	0.0	-9.019
1.0	-1.0	1.0	1.0	0.0	-2.187

Our current performance model was constructed by the screening experiment we ran on the previous step, and domain knowledge provided by our hired expert. We are now going to fit this model using the results of the experiments in our D-Optimal design. Table V shows the model fit table and compares the estimated and real model coefficients. This example illustrates that the Design of Experiments approach can achieve close model estimations using few resources, provided are able to use user input to identify relevant factors and knowledge about the problem domain to tweak the model.

Table V: Correct model fit comparing real and estimated coefficients, with significance intervals from the `R` language

	Real	Estimated	t value	Pr(> |t|)	Signif.
Intercept	0.000	0.278	1.192	0.287	
x_1	-1.500	-1.378	-8.116	0.000	***
x_3	1.300	1.283	7.558	0.001	***
x_5	3.100	3.017	18.851	0.000	***
x_7	-1.400	-1.659	-10.365	0.000	***
I(x_8^2)	1.350	1.222	3.816	0.012	*
x_1:x_3	1.600	1.718	10.124	0.000	***

Table VI compares the global minimum in this example with the predictions made by our initial linear model from the screening step and our improved model from this step.

Using screening, D-Optimal designs, and domain knowledge we found an optimization within 10% of the global optimum computing Y only 24 times. We were able to do that by first reducing the dimension of the problem when we eliminated irrelevant factors in the screening step. We then constructed a more careful exploration of this new problem subspace, helped by domain knowledge provided by an expert.

Table VI: Comparison of the response Y predicted by our models and the global minimum Y

	x_1	x_3	x_5	x_7	x_8	Y
Correct Model	1.00	-1.00	-1.00	1.00	0.00	-9.019
Linear Model	-1.00	-1.00	-1.00	1.00	-1.00	-1.046
Global Minimum	1.00	-1.00	-1.00	0.80	0.00	-9.934

We are able to explain the performance improvements we obtained in each step of the process, because we finish steps with a performance model and a performance prediction. Each factor is included or removed using information obtained in statistical tests or expert knowledge. If we need to optimize this problem again, for a different architecture or with larger input, for example, we are able to start exploring the search space with a less naive model.

The process of screening for factor significance using ANOVA and fitting a new model using acquired knowledge is essentially a step in the transparent Design of Experiments approach we present in the next Section.

IV. Autotuning with Design of Experiments

In this Section we discuss in detail our iterative Design of Experiments approach to autotuning. At the start of the process it is necessary to define the factors and levels that compose the search space of the target problem, select an initial performance model, and generate an experimental design. Then, as discussed in the previous Section, we identify relevant factors by running an ANOVA test on the results. This enables selecting and fitting a new performance model, which is used for predicting levels for each relevant factor. The process can then restart, generating a new design for the new problem subspace. Informed decisions made by the user play a central role in each iteration, guiding and speeding up the process. Figure 4 presents an overview of our approach.

The first step of our approach is to define which are the target factors and which levels of each factor are worth exploring. Then, the user must select an initial performance model. Compilers typically expose many 2-level factors in the form of configuration flags. The performance model for a single flag can only be a linear term, since there are only 2 values to measure. Interactions between flags can also be considered in an initial model. Numerical factors are also common, such as block sizes for CUDA programs or loop unrolling amounts. Deciding which levels to include for these kinds of factors requires a more careful analysis. For example, if we suspect the performance model has a quadratic term for a certain factor, we must include at least three of its levels. We can always consider the entire valid range of numerical factors. Other compiler parameters such as `-O(0,1,2,3)` have no clear ordering between their levels. These are categorical factors, and must be treated differently when constructing designs and analyzing the results.

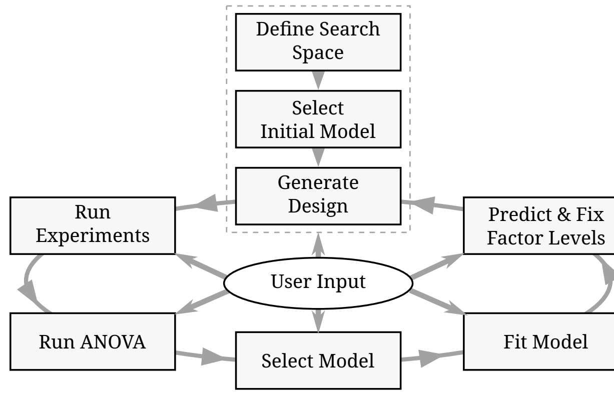

Figure 4: Overview of the Design of Experiments approach to autotuning proposed in this paper

We decided to use D-Optimal designs because their construction techniques enable mixing categorical and numerical factors in the same screening design, while biasing sampling according to a performance model. This enables the autotuner to exploit global search space structures if we use the right model. When constructing a D-Optimal design the user can require that specific points in the search space are included, or that others are not. Algorithms for constructing D-Optimal designs are capable of adapting to these requirements by optimizing a starting design. Before settling on D-Optimal designs, we explored other design construction techniques such as the Plackett-Burman [14] screening designs shown in the previous Section, the *contractive replacement* technique of Addelman-Kempthorne [16] and the *direct generation* algorithm by Grömping and Fontana [17]. These techniques have strong requirements on design size and level mixing, so we opted for a more flexible technique that would enable exploring a more comprehensive class of autotuning problems.

After the design is constructed we run each selected experiment. This step can be done in parallel since experiments are independent. Runtime failures are common in this step due to problems such as incorrect output. The user can decide whether to construct a new design using the successfully completed experiments or to continue to the analysis step if enough experiments succeed.

The next four steps of an iteration, shown in Figure 4, were discussed in detail in the previous Section. User input is fundamental to the success of these steps. After running the ANOVA test, the user should apply domain knowledge to analyze the ANOVA table and determine which factors are relevant. Certain factors might not appear relevant but the user might still want to include them in the model to explore more of its levels, for example. Selecting the model after

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

Figure 5: Initial performance model used by LM and DLMT

the ANOVA test also benefits from domain knowledge. The impact of the number of threads used by a parallel program on its performance is usually modeled using a quadratic term, for example.

A central assumption of ANOVA is the *homoscedasticity* of the response, which can be interpreted as requiring the observed error on measurements to be independent of factor levels and of the number of measurements. Fortunately, up to a point, there are statistical tests and corrections for lack of homoscedasticity, and our approach uses those before every ANOVA step.

After the model is selected and fitted, prediction results will depend on the size of the data set available. If it is feasible to compute the fitted model on all possible factor combinations, we can be sure that the global optimum has a chance of being found. If the search space is too large to be generated, we have to adapt this step and run the prediction on a sample.

The last step on an iteration is fixing factor levels to those predicted to have best performance. The user can also decide the level of trust that will be placed on the model and ANOVA at this step by allowing other levels. This step performs a reduction on the dimension of the problem by eliminating factors and decreasing the size of the search space. If we identify relevant parameters correctly, we will have restricted further search to better regions of the search space. In the next Section we present the performance of our approach in scenarios that differ on search space size, availability and complexity.

V. Performance Evaluation

A. Example on a GPU Laplacian Kernel

Table VII: Algorithms compared in the GPU Laplacian Kernel

	Algorithm
RS	Random Sampling
LHS	Latin Hyper Square Sampling
GS	Greedy Search
GSR	Greedy Search w/ Restart
GA	Genetic Algorithm
LM	Iterative Linear Model
DLMT	D-Optimal Designs

1) Results:

Table VIII: Comparison of slowdown and budget used by 7 optimization methods on the Laplacian Kernel, using a budget of 125 points with 1000 repetitions

	Mean	Min.	Max.	Mean Points	Max Points
RS	1.10	1.00	1.39	120.00	120.00
LHS	1.17	1.00	1.52	98.92	125.00
GS	6.46	1.00	124.76	22.17	106.00
GSR	1.23	1.00	3.16	120.00	120.00
GA	1.12	1.00	1.65	120.00	120.00
LM	1.02	1.01	3.77	119.00	119.00
DLMT	1.01	1.01	1.01	54.84	56.00

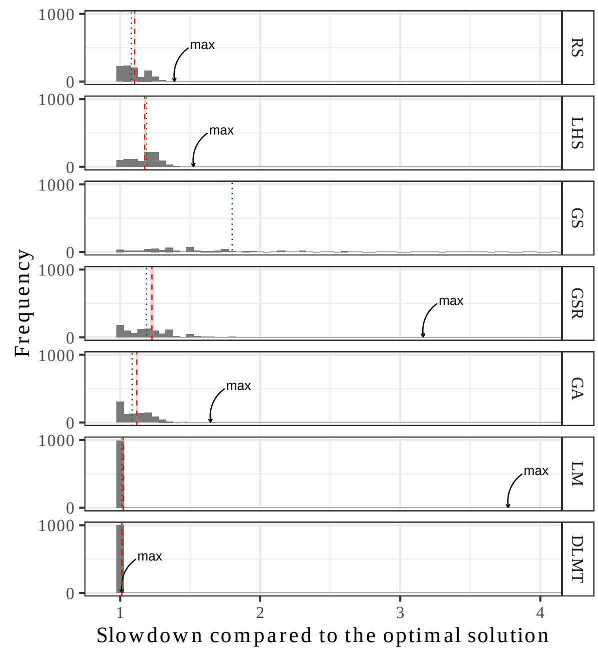

Figure 6: Histograms of 7 optimization methods on the Laplacian Kernel, using a budget of 125 points with 1000 repetitions

2) Experimental Methodology:

1) Effect of Sampling in D-Optimal Designs

B. Results on the SPAPT Benchmark

1) The SPAPT Benchmark:

VI. Conclusion

Acknowledgment

References

[1] K. Seymour, H. You, and J. Dongarra, “A comparison of search heuristics for empirical code optimization.” in *CLUSTER*, 2008, pp. 421–429.

[2] P. M. Knijnenburg, T. Kisuki, and M. F. O’Boyle, “Combined selection of tile sizes and unroll factors using iterative compilation,” *The Journal of Supercomputing*, vol. 24, no. 1, pp. 43–67, 2003.

[3] P. Balaprakash, S. M. Wild, and B. Norris, “Spapt: Search problems in automatic performance tuning,” *Procedia Computer Science*, vol. 9, pp. 1959–1968, 2012.

[4] J. R. Rice, “The algorithm selection problem,” in *Advances in Computers 15*, 1976, pp. 65–118.

Table IX: Set of applications we used from the SPAPT benchmark

Kernel	Operation	Factors	Size
atax	Matrix transp. & vector mult.	18	2.6×10^{16}
dgemv3	Scalar, vector & matrix mult.	49	3.8×10^{36}
gemver	Vector mult. & matrix add.	24	2.6×10^{22}
gesummv	Scalar, vector, & matrix mult.	11	5.3×10^{9}
hessian	Hessian computation	9	3.7×10^{7}
mm	Matrix multiplication	13	1.2×10^{12}
mvt	Matrix vector product & transp.	12	1.1×10^{9}
tensor	Tensor matrix mult.	20	1.2×10^{19}
trmm	Triangular matrix operations	25	3.7×10^{23}
bicg	Subkernel of BiCGStab	13	3.2×10^{11}
lu	LU decomposition	14	9.6×10^{12}
adi	Matrix sub., mult., & div.	20	6.0×10^{15}
jacobi	1-D Jacobi computation	11	5.3×10^{9}
seidel	Matrix factorization	15	1.3×10^{14}
stencil3d	3-D stencil computation	29	9.7×10^{27}
correlation	Correlation computation	21	4.5×10^{17}

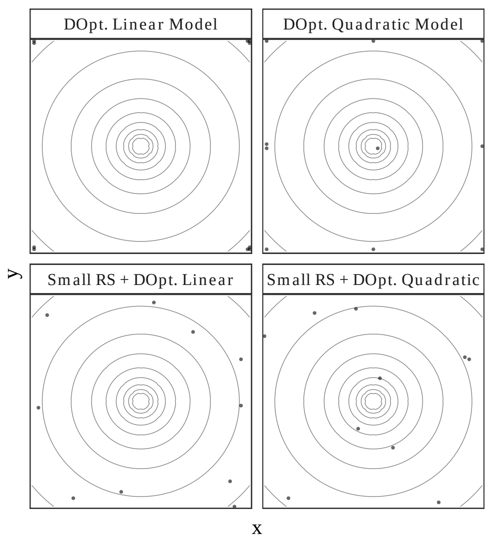

Figure 7: Effect of a small initial candidate set on constructing D-Optimal designs

[5] J. Bilmes, K. Asanovic, C.-W. Chin, and J. Demmel, "Optimizing matrix multiply using phipac: a portable, high-performance, ansi c coding methodology," in *Proceedings of International Conference on Supercomputing, Vienna, Austria*, 1997.

[6] J. J. Dongarra and C. R. Whaley, "Automatically tuned linear algebra software (atlas)," *Proceedings of SC*, vol. 98, 1998.

[7] R. Vuduc, J. W. Demmel, and K. A. Yelick, "Oski: A library of automatically tuned sparse matrix kernels," in *Journal of Physics: Conference Series*, vol. 16, no. 1. IOP Publishing, 2005, p. 521.

[8] M. Frigo and S. G. Johnson, "Fftw: An adaptive software architecture for the fft," in *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*, vol. 3. IEEE, 1998, pp. 1381–1384.

[9] M. Gerndt and M. Ott, "Automatic performance analysis with periscope," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 6, pp. 736–748, 2010.

[10] H. Jordan, P. Thoman, J. J. Durillo, S. Pellegrini, P. Gschwandtner, T. Fahringer, and H. Moritsch, "A multi-objective auto-tuning framework for parallel codes," in *High Performance Computing, Networking, Storage and Analysis (SC), 2012 International Conference for*. IEEE, 2012, pp. 1–12.

[11] J. Ansel, C. Chan, Y. L. Wong, M. Olszewski, Q. Zhao, A. Edelman, and S. Amarasinghe, *PetaBricks: a language and compiler for algorithmic choice*. ACM, 2009, vol. 44, no. 6.

[12] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle, "Paramils: an automatic algorithm configuration framework," *Journal of Artificial Intelligence Research*, vol. 36, no. 1, pp. 267–306, 2009.

[13] J. Ansel, S. Kamil, K. Veeramachaneni, J. Ragan-Kelley, J. Bosboom, U.-M. O'Reilly, and S. Amarasinghe, "Opentuner: An extensible framework for program autotuning," in *Proceedings of the 23rd international conference on Parallel architectures and compilation*. ACM, 2014, pp. 303–316.

[14] R. L. Plackett and J. P. Burman, "The design of optimum multifactorial experiments," *Biometrika*, vol. 33, no. 4, pp. 305–325, 1946.

[15] V. V. Fedorov, *Theory of optimal experiments*. Elsevier, 1972.

[16] S. Addelman and O. Kempthorne, "Some main-effect plans and orthogonal arrays of strength two," *The Annals of Mathematical Statistics*, pp. 1167–1176, 1961.

[17] U. Grömping and R. Fontana, "An algorithm for generating good mixed level factorial designs," Beuth University of Applied Sciences, Berlin, Tech. Rep., 2018.