

# Eiger: A Framework for the Automated Synthesis of Statistical Performance Models

Andrew Kerr\*, Eric Anger\*, Gilbert Hendry† and Sudhakar Yalamanchili\*

\*School of Electrical and Computer Engineering

Georgia Institute of Technology

{akerr, eanger, sudha}@gatech.edu

†Sandia National Laboratories

ghendry@sandia.gov

**Abstract**—As processor architectures continue to evolve to increasingly heterogeneous and asymmetric designs, the construction of accurate performance models of execution time and energy consumption has become increasingly more challenging. Models that are constructed, are quickly invalidated by new features in the next generation of processors while many interactions between application and architecture parameters are often simply not obvious or even apparent. Consequently, we foresee a need for an automated methodology for the systematic construction of performance models of heterogeneous processors. The methodology should be founded on rigorous mathematical techniques yet leave room for the exploration and adaptation of a space of analytic models. Our current effort toward creating such an extensible, targeted methodology is *Eiger*. This paper describes the methodology implemented in *Eiger*, the specifics of *Eiger*'s extensible implementation and the results of one scenario in which *Eiger* has been applied - the synthesis of performance models for use in the simulation-based design space exploration of Exascale architectures.

## I. INTRODUCTION

The transition to many-core computing has been accompanied by the emergence of heterogeneous architectures combining mainstream multithreaded cores with accelerators such as vector units (Intel AVX) or graphics processor units (GPU), e.g., from AMD and NVIDIA. Critically important are performance models which are necessary for the efficient management of these architectures, e.g., in task scheduling, as well being central to the design space exploration of future heterogeneous processors. However, the increased complexity of these architectures challenges the construction of accurate performance models. In addition, the design-space exploration of the interaction of future hardware and software that is necessary to advance computing into the Exascale regime is highly dependent on simulation and analytical modeling. Coarse-grained simulators such as SST/macro [1] can be used to predict the performance of many thousands of tasks running on millions of heterogeneous cores and the communication between them using models of varying degrees of fidelity, often trading off simulation performance for accuracy. The periodic construction of accurate analytic models is central to such analyses.

In this paper, we describe an automated statistical approach for modeling program behaviors on diverse architectures and present an evaluation of the application of the infrastructure to the problem of generating performance models for graphics processing unit (GPU) accelerators. Our objective is to *design and implement a methodology* for discovering and synthesizing analytic performance models of the runtimes and energy consumption of applications executing on target heterogeneous processors.

Our approach discovers analytic relationships between the static and dynamic parameters of an application and performance metrics. For example, we may wish to capture the impact of the sparsity of input data structures, dynamic execution count, and number of function calls on the execution time. Or we may wish to discover a relationship between the number of double precision load operations, number of DMA calls, and occurrence of unconditional branches

on energy consumption. These are usually complex relationships that elude manual discovery or effective application of off the shelf models and mathematical techniques. This complexity is magnified in modern and emerging heterogeneous processors. Broadly, our methodology is comprised of **1.**) experimental data acquisition and database construction, **2.**) a series of data analysis passes over the database (possibly creating new higher order data), and **3.**) model selection and construction. The last phase automates the construction of the software implementations of the models while the analysis passes can utilize a rich source of existing data analytics techniques.

Our current implementation is *Eiger*, which achieves our goals via statistical methods for dimensionality reduction and automated model selection. *Eiger* constructs coarse-grain predictive models when trained with results of similar applications on similar machines. The regression models may be as simple or complex as desired using metrics ranging from fine grained counts of instruction distributions to coarse grain estimates of computation working set size. The infrastructure is easily extensible to new sources of measurement data and supports the incremental addition of new experimental data. Its modular structure supports the easy addition of new analysis and model construction passes. Consequently, we hope the infrastructure and framework will benefit other explorations in the community by lowering the barriers to entry in performance model generation or synthesis.

## II. MODELING TECHNIQUE

This section describes elements of the proposed performance modeling infrastructure as well as a formal definition of analyses and the model selection algorithm.

### A. *Eiger* Framework

A detailed illustration of the *Eiger* framework is provided in Figure 1. The following components constitute an automated process in which application profiling data is collected via a standardized interface and ultimately used to construct a model of runtimes, energy, or any other dependent result metric. The resulting statistical model may be then composed with other tools and applications such as simulation environments, heterogeneity-aware process schedulers, and reporting tools. This paper describes the construction of models of execution time. An application is executed on the target processor, multiple parameters recorded, and execution time measured. This is defined as one trial. Multiple trials for an application typically cover different parameter values.

**Measurements.** Due to the mutually independent nature of each trial, the individual runs required to accumulate this information can intrinsically be run independently. We use a relational database to manage the storage of all data accumulated during profiling runs as it allows asynchronous insertions while allowing for rigorous relational specifications. Additionally, we provide support for the

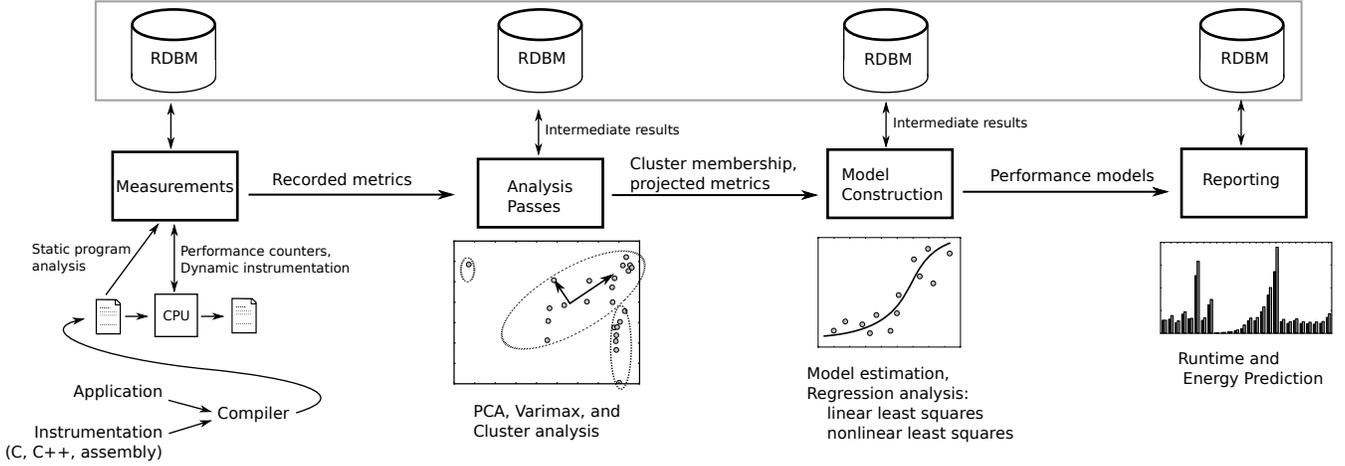


Fig. 1: Implementation details of Eiger Statistical Model Creation framework.

scenario where execution runs of multiple tools are required to construct a single data point (trial).

**Analysis.** The design supports the addition of analysis passes. We will start with Principal Component Analysis (PCA) which is a well-known dimensionality reduction technique. The major computations are construction of a correlation matrix and computing the eigenvectors of this correlation matrix via Singular Value Decomposition (SVD) [2]. Implementations of SVD are available in LAPACK implementations, notably SciPy [3]. Benefits of reducing the dimensionality of the input dataset are manifold; it speeds up the model generation process, improves the clarity of the resulting model, and allows for intuition into the correlations between input metrics.

It is important to note that PCA is an *unsupervised* learning technique in that it does *not* take the performance metric into account when choosing dimensions to eliminate. It is entirely possible that a dimension with low variance may have a larger affect on application performance than one with high variance. For example, number of cores may not have as large a variance as memory bandwidth for a range of machines but a greater impact on runtime for compute-bounded applications. PCA does not explicitly specify how many dimensions to retain; rather it relies upon the user to make the final decision.

**Model Construction.** Designing processors to accelerate general sets of workloads would be significantly simpler if all workloads exhibited similar performance characteristics. In contrast, real applications demonstrate varied performance behavior. Dense linear algebra workloads with regular control properties and compute-intensive inner loops differ significantly from irregular workloads with data-dependent branch behavior and load imbalance across threads. Goswami et al. [4] analyze the diversity of CUDA workloads and present a prioritized tree of benchmark applications sorted by how much each increases total variance of a given set of metrics. Applications exhibiting high correlation among principal components are clustered together. A single model trained from profiling data from all applications in a comprehensive benchmark is unlikely to yield high accuracy. Rather, the best model is likely to be obtained from training data gathered by a set of applications that are “similar” to the experimental application. Kerr et al. [5] provide empirical evidence that clustering and partitioning improves model accuracy.

**Model Pool** The model pool defines a set of possible basis functions which may be mapped to principal components and whose linear

Functions							
$x_i^{-2}$	$x_i^{-1}$	$x_i^{-1/2}$	$\log_2(x_i)$	$x_i^{1/2}$	$x_i^1$	$x_i^2$	$x_i * x_j$

TABLE I: Model pool.

combination yields the resulting performance model. The model pool must be selected by the experimenter and should offer sufficient variety for maximizing goodness of fit of the resulting model. The model pool should include functions that closely model the space and time complexity of dominant algorithms within the applications of interest as well as non-linear combinations of several metrics. For example, compute-bound applications may demonstrate a very strong correlation between the product of clock frequency and dynamic instruction counts. Table I describes the basis functions used for this work.

**Model Selection and Training.** Model construction is a pass over the data to produce a model. Our first model construction pass will be automated regression analysis [6] which constructs an analytic model determined from a set of training samples. In this case, the samples are taken from data projected onto the principle component basis vectors. Regression modeling yields an analytic formula for computing runtime and energy from additional signals. PCA and varimax yield orthogonal and uncorrelated principal components, crucial assumptions enabling classic regression analysis techniques. We propose the use of parametric regression models.

Each step in the stepwise procedure, shown in Algorithm 1, considers the function from the model pool that, if added to the current model, would increase the fit the most. If the  $\bar{R}^2$  of the this new model, including the candidate function, surpasses the  $\bar{R}^2$  of the model without the new function by more than the provided *threshold*, the function is added to the model, removed from the model pool, and the algorithm starts again. When there are no more functions remaining in the model pool that would pass this threshold, the algorithm completes and returns the final model. This final model may not have maximally reduced squared error for the training data, but the formulation of  $\bar{R}^2$  provides for a model that is more likely to predict values not present in the training set.

**Reporting** Completed models consist of a set of transformation matrices from dimensionality reduction and cluster analysis as well as a vector of functions and their associated weights. Reporting passes over this data will format the information in a method easily consumed by the user, including plotting and statistical results.

This phase also allows for the serialization and memoization of the finished models for later consumption. Finally, this phase produces model descriptions in a format that can be imported by system simulation tools and software modules such as run-time schedulers.

### B. Formal Specification

Let  $m \in \mathbb{Z}$  refer to the number of *trials* executed for a multiplicity of applications, datasets, and machine configurations. Let  $n \in \mathbb{Z}$  refer to the total number of metrics (measurements) acquired per trial. These may include static application metrics, dynamic metrics acquired during the execution of the trial, and machine configuration parameters.

We define  $X \in \mathbb{R}^{m \times n}$  as an input dataset and  $R \in \mathbb{R}^{m \times 1}$  as a result set. Each row in  $X$  corresponds to a trial instance with result (e.g., execution time) in the corresponding row of the column vector  $R$ . Together,  $(X, R)$  captures sufficient data describing the application, machine, and performance characteristics to construct a model for  $R$ .

$$X = \begin{bmatrix} \vdots & & \\ m \text{ trials} & & \\ \vdots & & \\ \dots & n \text{ metrics} & \dots \end{bmatrix} \quad R = \begin{bmatrix} \vdots \\ m \text{ results} \\ \vdots \end{bmatrix}$$

Principle component analysis (PCA) yields a projection  $P$  from  $X$  onto  $U \in \mathbb{R}^{m \times p}$ , where  $p \in \mathbb{Z}, p \leq n$  is the number of principle components, such that all axes are orthogonal and are sorted in decreasing amount of variance.

Clustering analysis enables a down-selection of trials used in model computation. This analysis yields a subset of rows such that  $U' = SU$  where  $U' \in \mathbb{R}^{m' \times p}, m' \leq m$  and  $S$  is a selection matrix. This work applies  $k$ -means clustering which partitions a set of points into  $k$  clusters such that each point in a cluster  $k_i$  is closest to the mean of  $k_i$  than any other mean. The distance metric used in this work is *squared Euclidean distance*, which gives increasingly greater weight to the distance between two elements.

Model selection yields a function  $f : \mathbb{R}^{1 \times p} \rightarrow \mathbb{R}$  that maps individual trials onto a predicted result value.  $f$  is the performance model, and this work yields one model per cluster. Model selection leverages linear regression, a commonly-used and well-behaved form of regression that evaluates the dependent variable  $y$  as the weighted linear combination of independent variables  $x$  plus an error term  $\epsilon$ , representing any deviation of the expected value of the model from the real value.

$$y = \beta_0 + \sum_{i=1}^n \beta_i x_i + \epsilon \quad (1)$$

The method of model estimation is least squares, in which the set of coefficients  $\beta$  is chosen to minimize the residual sum of squares

$$RSS(\beta) = \sum_{i=1}^n (y_i - \beta_0 - \sum_{j=1}^p x_{ij} \beta_j)^2 \quad (2)$$

Model selection itself composes a performance model as the linear combination of a set of non-linear functions on the projected profiling data. The set of possible functions is known as a *model pool*, which can include basis expansions, mathematical transformations, and variable interactions, among others. To allow for greater generalizability, this work can iterate over a set of model pools and select the one that minimizes squared error.

In order to manage model complexity and optimize training error rate, a forward-stepwise procedure is used to aggregate basis functions based upon adjusted coefficient of determination, a modification of the coefficient of determination that adjusts for the number of terms in the model [7].

$$AdjustedR^2 = \bar{R}^2 = 1 - (1 - R^2) \frac{n - 1}{n - p - 1} \quad (3)$$

It is important to note that  $\bar{R}^2$  does not have the same interpretation as  $R^2$ ; while  $R^2$  is in the range  $0 \rightarrow 1$ ,  $\bar{R}^2$  is in the range  $-\infty \rightarrow 1$ . The intuition is that any value of  $\bar{R}^2$  less than zero implies a fit worse than could be expected by chance.

```

profile, performance, modelPool = ... // initialize training data
threshold = ... // specification from user
finalModel = [] // empty set
currentRsquaredAdj = -∞
begin
  while not done do
    maximum = -∞
    foreach each function left in modelPool do
      add function to finalModel
      U = apply finalModel to profile
      beta = leastSquares(U, performance)
      RsquaredAdj = ... // calculate adjusted rsquared
      if RsquaredAdj  $\zeta$  maximum then
        maximum = RsquaredAdj
        newFunction = function
        newBeta = beta
      end
      remove current function from finalModel
    end
    if maximum - currentRsquaredAdj  $\zeta$  threshold then
      add newFunction to finalModel
      remove newFunction from modelPool
      currentRsquaredAdj = maximum
    end
    else
      done = True // there are no more useful functions
    end
  end
end
return finalModel, beta, currentRsquaredAdj

```

**Algorithm 1:** Selects a model that minimizes error over a cluster

### III. EVALUATION

This section describes the profiling infrastructure developed for Eiger including a normalized schema for storing profiling data in a relational database.

**Application Interfaces.** GPU Ocelot defines a functional simulator [8] for NVIDIA's PTX [9], a low-level RISC-like assembly language for GPU computing. We extended this functional simulator to capture detailed traces of dynamic instructions, SIMD utilization, and effective memory bandwidth. We leverage the compiler analysis functionality of GPU Ocelot to measure the metrics such as number of live registers, frequency of synchronization, and registers per thread.

Statistical performance modeling requires significant profiling to properly characterize workloads and yield enough data points at various configurations to adequately capture relationships among metrics and results. Consequently, this work focuses on providing a durable and centralized data store supporting concurrent access from multiple data sources. We selected MySQL, a commercially

available open-source relational database management system, to serve as the centralized data store. Eiger’s profiling database schema supports flexible, user-defined metrics to capture application behavior and machine configuration. At the top level, this work defines a *trial* as one execution of a GPU compute kernel. The trial references a particular application, machine configuration, and dataset and is annotated with additional metadata capturing details about the trial’s execution environment and purpose. The machine configuration is a vector of real numbers describing characteristics of the target processor.

### A. Metrics

The set of metrics under consideration are partitioned into two halves: application metrics and machine metrics. Application metrics are independent of the device upon which they are run where as machine metrics describe the execution hardware itself. Many of these application metrics are consistent with metrics identified in other workload characterization studies such as [10] which describes a set of microarchitecture-independent metrics.

Application metrics characterize the runtime behavior of the trial and are partitioned into deterministic and non-deterministic metrics. Deterministic metrics are invariant across machines and are based on profiling results obtained from executing the application on a deterministic functional simulator. This work assumes that applications are race free, or that data races may be safely ignored without impacting results. Non-deterministic metrics contain the results of hardware performance counters, runtimes, and other metrics which may vary non-deterministically across trials, even while all inputs and machine configuration remain constant. This partitioning of metrics enables profiling runs to capture strictly necessary information for each trial, as the deterministic metrics, by definition, do not vary across executions. A summary of deterministic application metrics are listed in Table II.

Metric	Units	Collection method
Memory Efficiency	percentage	instrumentation
Memory Intensity	instructions	instrumentation
Memory Sharing	percentage	instrumentation
Activity Factor	percentage	instrumentation
MIMD	speedup	instrumentation
SIMD	speedup	instrumentation
DMA Size	bytes	static analysis
Static Integer	instructions	static analysis
Static Memory	instructions	static analysis
Static Control	instructions	static analysis
Static Parallelism	instructions	static analysis
Static Float	instructions	static analysis
Static Special	instructions	static analysis
Dynamic Integer	instructions	emulation
Dynamic Memory	instructions	emulation
Dynamic Control	instructions	emulation
Dynamic Parallelism	instructions	emulation
Dynamic Float	instructions	emulation
Dynamic Special	instructions	emulation

TABLE II: Application metrics.

This work considers machine metrics, summarized in Table III which capture the performance characteristics of processors under consideration and define the dimensions of possible design space explorations.

### B. Validation Procedure

While there are many options for validation of regression models, there is no universally agreed upon method.  $R^2$ , known as the *coefficient of determination*, is often used in regression analysis,

	GTX 480	GTX 560Ti	C2070
Issue width	2	3	2
Shader frequency (MHz)	1401	1645	1150
Memory frequency (MHz)	3696	4008	3696
Cores per Streaming Multiprocessor	32	48	32
Streaming Multiprocessors	8	8	8
Bandwidth (GB/s)	177.4	128	144
L2 cache (kB)	640	512	640

TABLE III: GPU processor metrics.

Suite	Application	Description
Parboil	fft	Fast Fourier transformation
	mm	Dense matrix-matrix multiply
	mri-q	Magnetic resonance image reconstruction in non-Cartesian space
Rodinia	hotspot	Microprocessor thermal modeling
	gaussian	Linear system solver using Gaussian elimination
	bfs	Breadth-first graph traversal
CUDA SDK	scalarprod	Scalar product of vector pairs
	eigenvalues	Bisection algorithm for calculating eigenvalues of tridiagonal matrices
	binomialoptions	Fair call price evaluator for European options using binomial model
	scan	Parallel prefix sum over large vectors
	boxfilter	Box convolution filter for image processing
	mersennetwister	Random number generator

TABLE IV: Applications.

but it can only indicate how close the model fits the trained data, known as *training error*. Instead we would ideally like to know how well the model performs for test data independent from the training data, known as *test error*; however, this is complicated by statistically unstable methods for test error estimation. For this paper we will use *K-folds cross-validation* [11], a popular technique for estimating prediction error. In *K-fold cross-validation*, the input data set is randomly partitioned into *K* equally sized parts, denoted  $K_1, K_2, \dots, K_K$ . For all  $i \in 1 \dots K$ , parts  $K_{j \neq i} \forall j \in 1 \dots K$  are used to train the model, and part  $K_i$  is used for testing. The performance of the model is then the average of all *K* runs. The special case where  $K = N$  is known as *leave-one-out cross-validation*.

To evaluate Eiger’s capacity for performance prediction, 12 benchmark applications, listed in Table IV, were selected and profiled in detail using GPU Ocelot’s PTX functional simulator. This gathers deterministic application-dependent profiling information using the metric collection methodology laid out in Section II-B. Subsequently, the same set of applications were executed on NVIDIA GPUs via GPU Ocelot’s NVIDIA backend. Light-weight profiling tools attached to GPU Ocelot capture kernel execution times. Timing measurements were made for each of the devices described in Table III. The host machine is an Intel Core-i7 at 3.2 GHz.

### C. Experiments

**Varying Dimensionality.** This experiment reduces the number of dimensions retained after principal component analysis (PCA). Reducing dimensionality is a lossy compression technique, which has the potential to alias important metrics which may have a great impact on the performance of the model. However, increasing the dimensionality of the input data has the potential to complicate the model, resulting in poor generalizability due to over fitting. As seen in Figure 2, there is indeed a minimum where the benefit of reduced

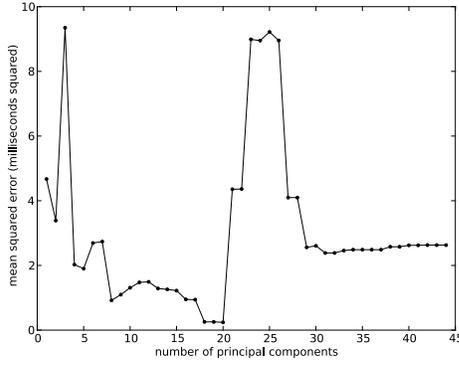


Fig. 2: Error as dimensions are varied.

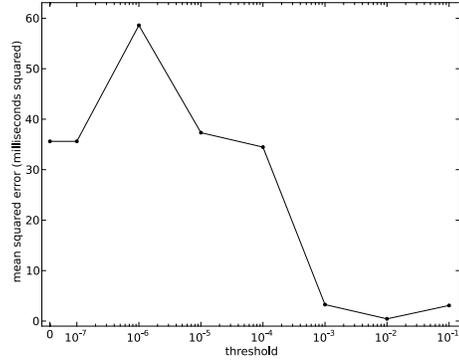


Fig. 4: Error as convergence threshold varies.

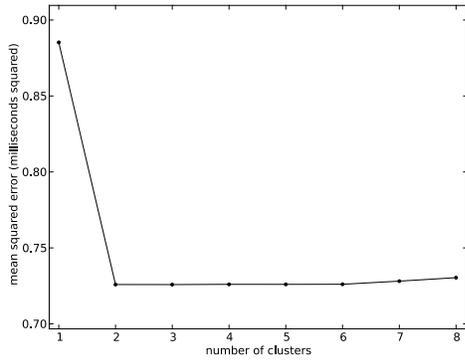


Fig. 3: Error as number of clusters are varied.

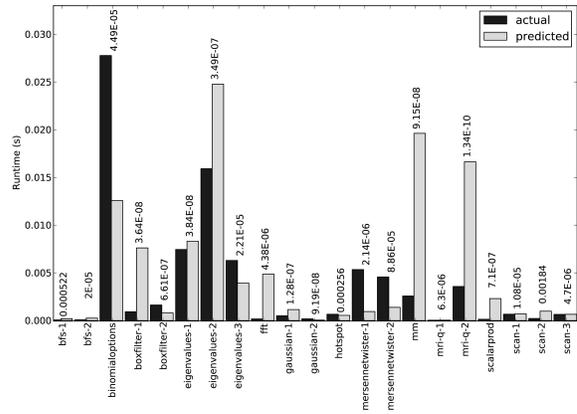


Fig. 5: Predicted versus actual runtimes for each application for models trained from all other applications, annotated by mean squared error.

dimensions on model complexity are balanced out by the loss of data.

**Varying Cluster Count.** This experiment increases the number of clusters to demonstrate the performance benefits of generating models from only like data points. Each data point in the experiment set is predicted by the model whose cluster’s centroid is the closest. The results, shown in Figure 3, demonstrate how the segmentation into separate clusters for modeling can increase the quality of the models, although at a certain point the quality of the individual models decays due to the low number of data points in the cluster.

**Varying New Function Threshold.** This experiment reduces the threshold for adding new functions to the final model. As the threshold decreases, more functions that only marginally increase the quality of the regression are included. This increases the dependence on the training data and therefore the variance in the final model. The effect of varying this threshold is demonstrated in Figure 4.

**Varying Applications.** In this experiment each application is predicted using models created from all of the other applications. This experiment demonstrates how generalized the application metrics are; instead of relying upon previous runs of the same execution of the application to train the model, which may obscure some of the application characteristics (e.g. algorithmic complexity, communication patterns, etc.), other applications with potentially widely different algorithmic implementations are used. Results are shown in Figure 5. Each application kernel is listed separately, indicated by the number concatenated to the end of the application name.

**Varying Machine Parameters.** In this experiment the GTX 480 is predicted by models trained on only the GTX 560Ti and the Tesla C2070. This experiment demonstrates how well hardware metrics

can describe the performance of a given application. Results from this experiment, shown in Figure 6, indicate accurate predictions are possible simply by varying machine parameters.

#### D. Discussion

Let us consider the structure of one of these models, where a model is built from every trial except those from the second `eigenvalues` kernel. All but nine principal components are removed and all trials belong to a single cluster. The threshold used in the model building algorithm is 0.01. The final model built takes the form  $runtime = \beta_0 * \log_2(PC_0) + \beta_1 * \log_2(PC_8) + \beta_2 * \log_2(PC_1) + \beta_3 * (PC_2 * PC_3)$  where  $\beta_0, \beta_2$  are positive and  $\beta_1, \beta_3$  are negative. Here  $PC_0$  represents problem size, including contributions from dynamic instruction counts, function call depth, and branches;  $PC_8$  represents SIMD utilization, i.e. parallelization and control flow divergence;  $PC_1$  represents program size, contributed to most by static instruction counts;  $PC_2 * PC_3$  represents throughput as the product of threading and efficiency. Intuitively, this set of principal components makes sense: as problem size increases, so does execution time, and as SIMD utilization and throughput increase, execution time *decreases*. This resulted in a 29.28% average mean absolute percent error when predicting the second `eigenvalues` kernel trials.

It is surprising that Eiger selects base-2 logarithms to construct the performance model, however. This is a consequence of automated model selection that attempts to find the best fit possible given the

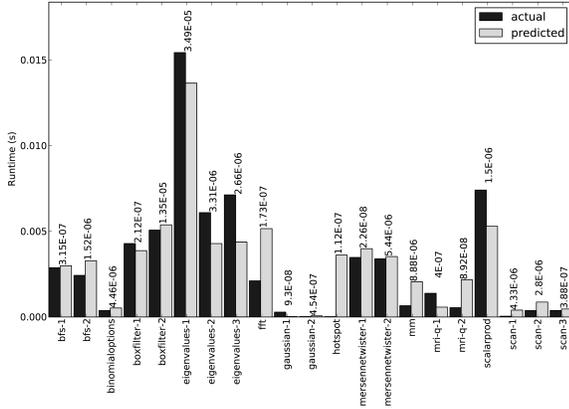


Fig. 6: Predicted versus actual runtimes on the GTX 480 when model is trained from the GTX 560Ti and the Tesla C2070, annotated by mean squared error.

available model pool and training data. For the given problem sizes, this performance model yields the best-fit model of runtimes. Careful analysis of the algorithm, implementation, and target hardware would almost certainly not express runtime as proportional to the logarithm of dynamic instruction counts. And yet, doing so yields the closest fitting performance model for the problem sizes in the training data. Thus, Eiger is able to obtain the best performance model given training data and may out perform analytic models.

#### IV. RELATED WORK

Jia et al. [12] present a design space exploration technique which simulates a random subset of GPU designs from a very large design space then applies a stepwise regression modeling algorithm to construct a performance estimator. Our work presents a flexible framework for the *composition and use* of such existing and future regression techniques to facilitate the construction and application of performance models.

Genbrugge et al. [13] describe a method for constructing a synthetic trace of a program execution bearing the same statistical properties as a complete execution but of much shorter overall length. Both approaches are focused on modeling specific phenomena rather than enabling the construction of general models. Hong et al. [14] propose a predictive analytical performance model for GPUs. Our approach, on the other hand, does not assume particular processor architecture or machine model and instead attempts to determine them based on measurable statistics that may change substantially as microarchitectures evolve.

PCA-based approach to modeling GPU workloads executing on either GPUs or CPUs. That work inspired the design of Eiger which formalizes and automates the steps of application profiling, dimensionality reduction, cluster analysis, and model selection. While much of the cluster analysis and model selection performed in that work was performed in a human-guided *ad hoc* manner, the Eiger framework provides a automated implementation for i) importing instrumentation data, ii) exercising the analysis passes, and iii) searching the (extensible) space of performance models. The models are exported in a format that can be used by simulators and software systems. Goswami et al. [4] perform detailed characterization studies of GPU applications using a detailed cycle-accurate simulator and identify redundancies in characterization metrics among kernels from different CUDA benchmark applications. The authors decompose kernel executions to provide recommendations for prioritizing

the execution of benchmarks to exercise the most complete set of program behaviors in as few trial executions as possible.

#### V. CONCLUSION

This study demonstrates the Eiger modeling framework for automating the generation of statistical performance models. Eiger provides an automated method in which designers may profile and characterize workloads, automatically construct performance models, and evaluate performance sensitivity to processor configurations. Our results show models based on as few as 5-7 principal components achieve fairly low mean squared error, but that adding principal components can increase squared error. We also verify cluster analysis has a strong impact on model accuracy due to significant differences in workload characteristics among benchmark suites. Finally, this empirical evaluation shows an automated statistical performance modeling framework such as Eiger can provide an effective approach to design space exploration of candidate microarchitectures. The next step for us is the application to the modeling of energy and power.

#### VI. ACKNOWLEDGMENTS

This research was supported in part by the National Science Foundation under grant CCF-0905459 and Sandia National Laboratories. Sandia National Laboratories is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

#### REFERENCES

- [1] C. L. Janssen, H. Adalsteinsson, S. Cranford, J. P. Kenny, A. Pinar, D. A. Evensky, and J. Mayo, "A simulator for large-scale parallel computer architectures." *IJDS*, vol. 1, no. 2, pp. 57–73, 2010.
- [2] G. Golub and C. V. Loan, *Matrix Computations*, 3rd ed. Baltimore, MD.: Johns Hopkins University Press, 1996.
- [3] "Scipy: Scientific tools for python," July 2012, <http://www.scipy.org/>.
- [4] N. Goswami, R. Shankar, M. Joshi, and T. Li, "Exploring gpgpu workloads: Characterization methodology, analysis and microarchitecture evaluation implications," in *Workload Characterization (IISWC), 2010 IEEE International Symposium on*, dec. 2010, pp. 1–10.
- [5] A. Kerr, G. Damos, and S. Yalamanchili, "Modeling gpu-cpu workloads and systems," in *Third Workshop on General-Purpose Computation on Graphics Processing Units*, Pittsburg, PA, USA, March 2010.
- [6] H. Samet, *Foundations of Multidimensional and Metric Data Structures*, 1st ed. Morgan Kaufmann, 2006.
- [7] M. H. Kutner, C. J. Nachtsheim, and J. Neter, *Applied Linear Regression Models*, fourth international ed. McGraw-Hill/Irwin, Sep. 2004.
- [8] A. Kerr, G. Damos, and S. Yalamanchili, "A characterization and analysis of ptx kernels," in *IISWC09: IEEE International Symposium on Workload Characterization*, Austin, TX, USA, October 2009.
- [9] NVIDIA, *NVIDIA Compute PTX: Parallel Thread Execution*, 1st ed., NVIDIA Corporation, Santa Clara, California, October 2008.
- [10] K. Hoste and L. Eeckhout, "Microarchitecture-independent workload characterization," *Micro, IEEE*, vol. 27, no. 3, pp. 63–72, may-june 2007.
- [11] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection." Morgan Kaufmann, 1995, pp. 1137–1143.
- [12] W. Jia, K. Shaw, and M. Martonosi, "Stargazer: Automated regression-based gpu design space exploration," in *Performance Analysis of Systems and Software (ISPASS), 2012 IEEE International Symposium on*, april 2012, pp. 2–13.
- [13] D. Genbrugge and L. Eeckhout, "Chip multiprocessor design space exploration through statistical simulation," *Computers, IEEE Transactions on*, vol. 58, no. 12, pp. 1668–1681, dec. 2009.
- [14] S. Hong and H. Kim, "An integrated gpu power and performance model," in *Proceedings of the 37th annual international symposium on Computer architecture*, ser. ISCA '10. New York, NY, USA: ACM, 2010, pp. 280–289.