



LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

Detecting Soft Errors in Stencil based Computations

V. Sharma, G. Gopalkrishnan, G. Bronevetsky

May 7, 2015

Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

Detecting Soft Errors in Stencil based Computations

Vishal C. Sharma and Ganesh Gopalakrishnan
School of Computing, University of Utah, USA
Email: {vishal,ganesh}@cs.utah.edu

Greg Bronevetsky
Lawrence Livermore National Laboratory
Email: bronevetsky1@llnl.gov

Abstract—Given the growing emphasis on system resilience, it is important to develop software-level error detectors that help trap hardware-level faults with reasonable accuracy while minimizing false alarms as well as the performance overhead introduced. We present a technique that approaches this idea by taking stencil computations as our target, and synthesizing detectors based on machine learning. In particular, we employ linear regression to generate computationally inexpensive models which form the basis for error detection. Our technique has been incorporated into a new open-source library called **SORREL**. In addition to reporting encouraging experimental results, we demonstrate techniques that help reduce the size of training data. We also discuss the efficacy of various detectors synthesized, as well as our future plans.

I. INTRODUCTION

Soft errors (also called *single-event-upsets*) are one of the most serious of impediments to the rapid attainment of large-scale (especially exa-scale) computing capabilities. Such errors are typically caused by radiation from chip packaging, cosmic rays [1], [2] or even circuit noise due to low-power operation [3], [4]. They can introduce silent data corruptions (SDC) into the computational state [5], [6], and as such are a huge concern in extreme-scale computing [7], [8].

In this paper, we focus on the synthesis of software-level error detectors for soft-errors. In a nutshell, these detectors are nothing but specific `assert` statements introduced into the user code, with the expectation that the assertion fail whenever there is a fault. Naturally, they must also not fail when there is no fault (“false alarms”), and there must be the least added computational burden. Such detectors underlie any system resilience solution—whether it be checkpointing and restart [7], [9], [10] or more localized containment and repair methods.

Our specific contributions are toward an almost fully automated synthesis of soft-error detectors for applications that perform time-stepped stencil computations, such as in solvers for finite-difference discretization of ordinary and partial differential equations (ODEs and PDEs). We focus on detecting SDCs caused by transient soft-errors in CPU operations and registers. We do not consider permanent errors (caused due to transistor aging [4], [11]). We also do not consider soft-errors occurring in other memory elements such as DRAM

Supported in part by NSF Award CCF 1255776, SRC Contract 2013-TJ-2426, and Scientific Discovery through Advanced Computing (SciDAC) program funded by U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research (and Basic Energy Sciences/Biological and Environmental Research/High Energy Physics/Fusion Energy Sciences/Nuclear Physics). See <http://super-sciDAC.org/>.

and data caches, as these subsystems are very effectively protected via error correcting codes (ECC) [12], and other methods [13], [14], including triple modular redundancy [15]. While we evaluate our approach using a numerical solution to a real world example called reverse time migration (RTM) [16], our approach is more broadly applicable to all forms of time-stepped stencil based codes.

Our key idea is to use machine learning techniques to train a cost-effective regression model that predicts the output of the target stencil’s kernel given its input. The model will be trained on values observed in real stencil executions and will declare an error when its predictions significantly disagree with the value computed by the stencil. Our specific contributions are:

- A novel approach that uses regression techniques to compute an efficient approximation of the computational kernels used in a given stencil,
- A systematic way to reduce the size of a training data used for generating a regression model,
- Application of cross validation to estimate the sample size and for feature selection for building efficient detectors; and
- A new open-source soft-error detection library **SORREL** that helps evaluate/extend our technique. It could be compiled as a static library or as a dynamically linked shared object library. A detailed usage documentation has been made available with the public release of **SORREL** [17].

II. RELATED WORK

Recently, there has been a considerable interest in developing efficient error detectors for time-stepping applications. The work by Benson et. al. [18] proposes running a cost-effective but unstable solver alongside the main solver and declaring an error when the results of these solvers disagree beyond a given threshold. Our approach is very similar to this work but derives the secondary solver automatically rather than selecting and implementing it manually for each main solver.

Another recent work by Berrocal et. al. [19] uses regression functions to detect run-time anomalies caused due to soft errors. They use execution data to learn these regression functions at run-time. The key differences between our and their approach are following:

- We train the approximate function in a separate phase. This allows us to use more data to produce a more accurate model but also places constraints on our technique.
- Whereas Berrocal et. al. train functions that specifically detect anomalies, **SORREL** computes functions that ap-

proximate the original computation. This makes them useful for additional purposes, such as optimization via approximate computing, and code understanding for developers.

Several other promising directions have been pursued towards building soft error detectors. While hardware and architectural-level protection [15], [20], [21] serves as a first line of defense, they must be complemented with more flexible software-level techniques. Several software-level techniques employ control-flow based detectors [22], [23], which rely on detecting illegal control transitions. These detectors are more suitable for control-flow rich applications, and less effective for data-intensive applications. Finally, Algorithm Based Fault Tolerance (ABFT) exploits algorithmic properties of a program to detect errors [24]–[26] but these solutions are problem-specific.

III. AN OVERVIEW

Our approach works with arbitrary stencils; in this paper it is evaluated on a specific important method, the reverse time migration (RTM) [16] algorithm, as implemented by McCool et al [27]. This example RTM stencil kernel shown in eq. 1 uses a finite-difference approximation of the PDE described in eq. 2. This scheme is second-order accurate in time and eighth-order accurate in space [27]. Here, P is a three-dimensional array and $P_n(x, y, z)$ denotes the value of the pressure wave at coordinates (x, y, z) at time n , and v is the velocity of pressure wave which is constant for a given medium. As shown in figure 1, to calculate $P_{n+1}(x, y, z)$ the RTM stencil kernel uses the values of 25-point RTM stencil from the previous time-step.

$$\begin{aligned}
 P_{n+1}(x, y, z) = & 2 * P_n(x, y, z) - P_{n-1}(x, y, z) + v^2 \left\{ C_0 \right. \\
 & * P_n(x, y, z) + \sum_{k=4} \left\{ C_k * (P_n(x+k, y, z) \right. \\
 & + P_n(x-k, y, z) + P_n(x, y+k, z) \\
 & + P_n(x, y-k, z) + P_n(x, y, z+k) \\
 & \left. \left. + P_n(x, y, z-k) \right) \right\} \left. \right\} \quad (1)
 \end{aligned}$$

Our goal is to train a regression model that predicts the output $P_{n+1}(x, y, z)$ of a stencil evaluation given one or more of its inputs. The choice of which inputs to use for model training affects its accuracy and cost. Table I lists the choices we evaluate in this study. The output and the selected inputs for a given stencil evaluation form a feature vector and the regression model is trained in a set of such vectors collected over multiple application runs with different inputs. The resulting regression model computes an approximation of the RTM stencil kernel, and is used to verify that whether the output of the original kernel is likely to be correct.

$$\frac{\partial^2 P}{\partial n^2} = v^2 \left(\frac{\partial^2 P}{\partial x^2} + \frac{\partial^2 P}{\partial y^2} + \frac{\partial^2 P}{\partial z^2} \right) \quad (2)$$

When selecting the features on which to train the regression model our goal is to use as few inputs of the RTM stencil

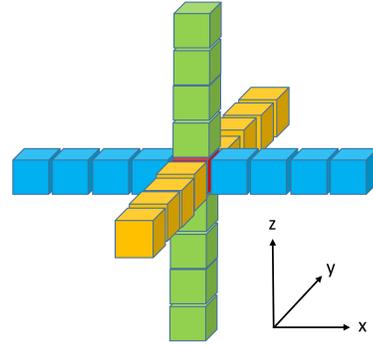


Fig. 1: 25-point RTM stencil

as possible while accurately approximating its output. Feature vectors f_1, f_4, f_5, f_6 include different subsets of the stencil's inputs, while vectors f_2 and f_3 include the stencil's output in the preceding time-step. In contrast, vector f_7 uses all 25 points in the RTM stencil and serves as an upper bound on the accuracy of a regression model. Our hypothesis, which is evaluated in the following sections, is that including more input data will produce a model that is more accurate but more expensive.

Feature Vector	Feature List	Comments
f_1	$P_n(x, y, z)$	
f_2	$P_{n-1}(x, y, z)$	
f_3	$P_n(x, y, z), P_{n-1}(x, y, z)$	
f_4	$P_n(x, y, z),$ $P_n(x+1, y, z), P_n(x-1, y, z),$ $P_n(x, y+1, z), P_n(x, y-1, z),$ $P_n(x, y, z+1), P_n(x, y, z-1)$	
f_5	$P_n(x, y, z),$ $P_n(x+s_1, y, z), P_n(x-s_1, y, z),$ $P_n(x, y+s_1, z), P_n(x, y-s_1, z),$ $P_n(x, y, z+s_1), P_n(x, y, z-s_1)$	$1 \leq s_1 \leq 2$
f_6	$P_n(x, y, z),$ $P_n(x+s_2, y, z), P_n(x-s_2, y, z),$ $P_n(x, y+s_2, z), P_n(x, y-s_2, z),$ $P_n(x, y, z+s_2), P_n(x, y, z-s_2)$	$1 \leq s_2 \leq 3$
f_7	$P_n(x, y, z),$ $P_n(x+s_3, y, z), P_n(x-s_3, y, z),$ $P_n(x, y+s_3, z), P_n(x, y-s_3, z),$ $P_n(x, y, z+s_3), P_n(x, y, z-s_3)$	$1 \leq s_3 \leq 4$

TABLE I: Feature Vectors based on 25-Point RTM Stencil

IV. APPROXIMATE FUNCTION GENERATION

The first step in training a regression model is to collect the data on which it will be trained. We generate the training dataset by running the RTM application on randomly-generated inputs. The inputs are randomly-generated by varying fields such as array size, number of time-steps, the value chosen to initialize $P_n(x, y, z)$ at time $n = 0$, and the point of origin for the pressure wave denoted as a point in $P_n(x, y, z)$ at time $n = 0$. During an execution of the stencil program, each point in time and space produces a unique observation. During training phase, these observations are used to generate training data.

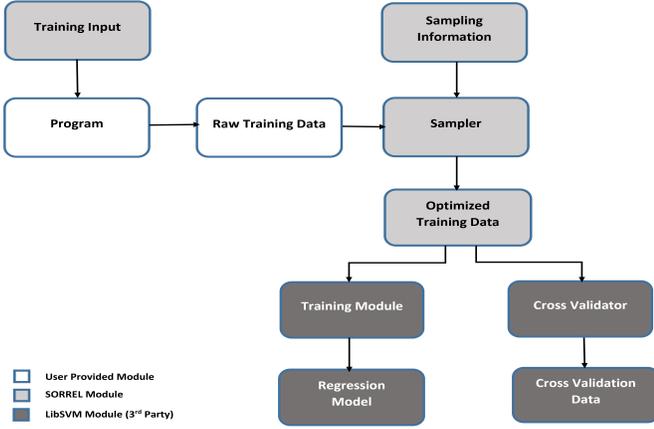


Fig. 2: Sampler workflow during training phase

A. Sampling Technique

Given the very large number of observations generated in the training phase, it is crucial that we perform sampling to reduce the training data size. As such, we employ stratified sampling [28] to collect a representative subset of all the collected data that captures the breadth of the full dataset but is tractable to train on. To sample we divide the time-stepped computational data into a finite number of (preferably equal sized) strata. We then perform a simple random sampling on the computational data belonging to each stratum having a sample size expressed in eq. 3.

$$S_m = k * P_m \quad (3)$$

Here, S_m is the sample size for a stratum with index m , and P_m is the corresponding population size. The population size P_m is the size of the complete computational data which belongs to the stratum at index m . The constant value k is the fraction of the total population size that is sampled.

B. Sample Size Estimation

To determine the number of samples necessary to compute an accurate model, we use a cross-validation driven approach [29] that is illustrated in figure 2. A target program is executed under a set of randomly-generated training inputs. Using an initial guess value for k , the raw computational data is sampled to produce an optimized training dataset. We perform n-fold cross-validation [30] on the training dataset by training LibSVM [31] on the training subsets and evaluating it on the test subsets to quantify accuracy. We iterate this process with the larger values of k until the respective cross-validation accuracy stabilizes. The smallest possible value of k that provides reasonable cross-validation accuracy is chosen for generating the regression model.

C. Regression Analysis

In order to generate approximate functions, we perform linear regression [32] using each feature vector listed in table I. Specifically, we use a linear regression kernel *epsilon*-SVR [33]–[35] implemented in LibSVM software [31]. The

generated regression models are then used by SORREL to compute their corresponding approximate functions.

V. DETECTOR SYNTHESIS

In eq. 4, $\mathcal{K}^n(x, y, z)$ represents a stencil kernel that computes $P_n(x, y, z)$, and $\mathcal{A}_{f_r}^n(x, y, z)$ is an approximation for $\mathcal{K}^n(x, y, z)$, generated through regression analysis using a feature vector f_r as described in section IV. As shown in eq. 4, $\mathcal{D}_{f_r}^n(x, y, z)$ is a boolean detector function. It returns *true* if the absolute difference between the observed and the predicted value for a given stencil computation, represented as $|\mathcal{K}^n(x, y, z) - \mathcal{A}_{f_r}^n(x, y, z)|$ exceeds a threshold value τ .

$$\mathcal{D}_{f_r}^n(x, y, z) = \begin{cases} \text{true}, & \text{if } |\mathcal{K}^n(x, y, z) - \mathcal{A}_{f_r}^n(x, y, z)| > \tau. \\ \text{false}, & \text{otherwise.} \end{cases} \quad (4)$$

Table II quantifies the additional number of operations performed by the detectors as compared to the *native* version of the RTM stencil kernel shown in eq. 1. The detectors D_{f_1} through D_{f_7} are synthesized using feature vectors f_1 through f_7 respectively as listed in table I. As shown later in section VI-D, the higher cost of detectors D_{f_4} through D_{f_7} relates directly to the larger numbers of RTM stencil points they use as input.

A. Threshold Estimation & Detector Accuracy

We now explain the method to determine an *optimal* or *near-optimal* value of the threshold (hereafter denoted as τ_{opt}) which yields a high error detection rate (*true-positives*) with few *false-positives*. To quantify the impact of τ on these two metrics we plot receiver operator characteristic (ROC) curve [36] for the detectors synthesized using feature vectors listed in table I. This curve shows the *true-positive* and *false-positive* rates achievable by each detector across a range of values for τ . For a given number of observations, the *true-positive rate* R_{tp} is calculated using number of observations with *true-positives* (N_{tp}) and *false-negatives* (N_{fn}) as shown in eq. 5.

$$R_{tp} = \frac{N_{tp}}{N_{tp} + N_{fn}} \quad (5)$$

The *false-positive rate* R_{fp} is calculated using number of observations with *false-positives* (N_{fp}) and *true-negatives* (N_{tn}) as shown in eq. 6.

$$R_{fp} = \frac{N_{fp}}{N_{fp} + N_{tn}} \quad (6)$$

In the current context, an observation represents an instance of a program execution during which a soft-error may or may not occur. If a soft-error is witnessed during an observation and the soft-error is successfully flagged by a detector then the observation is regarded as a *true-positive* instance. However, if the detector fails to catch the soft-error then the observation is a *false-negative* instance. Conversely, during an error-free observation, if a detector falsely reports the detection of a soft-error then we treat this observation as a *false-positive* instance.

If the detector does not flag any error during an error-free observation then we call it a *true-negative* instance.

A ROC curve is obtained by plotting R_{fp} and R_{tp} against each other using x and y axes respectively across a range values for τ . We select the point on the ROC curve that gives a high value for R_{tp} and a low value for R_{fp} . Note that the acceptable values for R_{tp} and R_{fp} depend on the magnitude of errors a given application can tolerate, with some applications being inherently resilient to errors of low magnitude [26], [37].

Operation Type	Operation Count ¹							
	Native Computation	D_{f_1}	D_{f_2}	D_{f_3}	D_{f_4}	D_{f_5}	D_{f_6}	D_{f_7}
{*}	19	1	1	2	7	13	19	25
{+}	25	0	0	1	6	12	18	24
{-}	1	1	1	1	1	1	1	1
{>}	0	1	1	1	1	1	1	1
{!=}	0	1	1	1	1	1	1	1
Total	45	4	4	6	16	28	40	52

TABLE II: A comparison of operation count

VI. EXPERIMENTAL RESULTS

We generate our error detectors in two phases. During the initial training phase we generate the approximate stencil function. During the test phase we select τ_{opt} using ROC curve analysis and evaluate the detector’s effectiveness.

A. Training Phase

We generate 1000 unique program inputs for the RTM program using SORREL. We use 1% of these inputs in the training phase and the rest are used during the testing phase. Each of these training inputs leads to a huge amount of training data underlining the need for sampling as explained in section IV. The next step is to quantify the distribution of errors made by each model relative to the real values computed by the RTM stencil. To this end we used n -fold cross-validation, where the set of observations (sampled using stratified sampling as explained in section IV-B) is divided into n non-overlapping sub-sets. For each sub-set i , we train a model using the remaining $n - 1$ sub-sets and compute the error of the model using sub-set i as the test data. Finally, the overall-error (hereafter referred as e_t) is computed by applying the mean-squared-error (MSE) metric on the individual model errors obtained using each of the n sub-sets. Figures 3 and 4 show the distribution of instances of e_t obtained using 10 different observation samples, when using either $n = 2$ or $n = 10$ and $k = 4e - 5$. The data shows that the instances of e_t are distributed according to a skew normal distribution, where all values lie within 3 standard deviations of the mean ($\mu \pm 3\sigma$). This distribution has the same shape regardless of the number of folds (n) and other values of k also produce errors with similar distributions.

Having observed a skew normal distribution for the instances of e_t , we now represent the accuracy of the corresponding regression model as an average of the individual

¹Operation count mentioned for the detectors are in addition to the number of operations required by the *native computation*.

values of e_t in the distribution (hereafter referred as e_a). Next, we determine the appropriate value of k , which controls how sparsely the training data is sampled. Figures 5 and 6 show the different values of e_a calculated for the regression models based on feature vectors f_1 through f_7 with values of k ranging from $1e-6$ to $1e-4$. As k increases (i.e. more observations are used to train the model) the regression model’s accuracy (e_a) improves until k reaches $4e - 5$, after which point it stabilizes. We thus use $k = 4e - 5$ for training our models.

Another important point to note that in our experiments, we fix the stratum size to 60 while performing stratified sampling, which means each stratum represents data from 60 consecutive time-steps of the RTM program. We fix the stratum size in order to limit the volume of our experiments. In general, it is expected that smaller the size of the stratum, the finer will be the representation of the individual time-steps leading to a better cross validation accuracy. In future, we plan to further augment our current experimental strategy (which already includes extensive set of experiments to determine sample size) to also determine the optimal stratum size.

B. Error Model

We consider an error model involving a soft-error occurring in a CPU operation or register. Further, we only consider a subset of the soft-errors which cause SDC in the program output. Therefore, we do not consider all possible program locations in the RTM program for error injections but rather only inject single-bit errors into a single randomly-selected location in an array used by the stencil. The time-step during which the error is injected is also chosen at random. The error is injected in a value after it is loaded from a memory location into a CPU register and before it is stored back to the memory after a computation. This approach helps us to focus on our primary goal of studying the efficacy of our detectors in detecting SDC causing soft-errors.

C. Threshold Selection

To determine τ_{opt} , we plot the ROC-curve using the values of *true-positive* rate (R_{tp}) and *false-positive* rate (R_{fp}) computed by running two independent experiments. In the first experiment, we run the RTM program under each test input and a soft-error is injected during each run. We obtain the values for N_{tp} and N_{fn} from this experiment and use these values to compute R_{tp} using eq. 5. In the second experiment, we repeat all the steps followed in the first experiment without injecting any error. Using the result of the second experiment, we obtain N_{fp} and N_{tn} and use them to compute R_{fp} following eq. 6. We repeat these experiments for different threshold values for each of the detectors synthesized using feature vectors listed in table I.

The threshold value chosen for our experiments starts with a very small value, and is gradually increased until we achieve reasonable *true-positive* and *false-positive* rates. Figure 7 shows that a very high value of the *true-positive* and the *false-positive* rates are observed (*top-right* area) for a low threshold

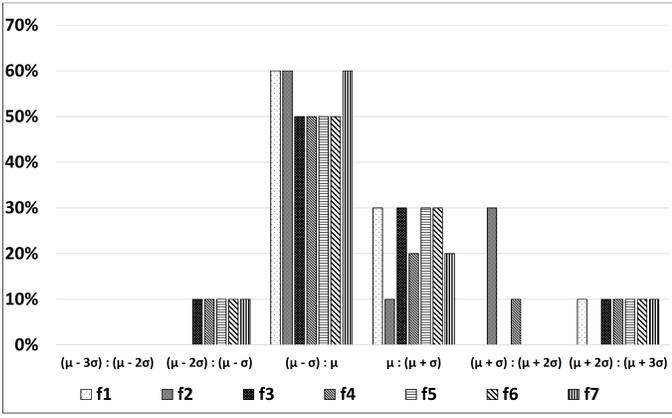


Fig. 3: Distribution of e_t for 2-fold Cross Validation

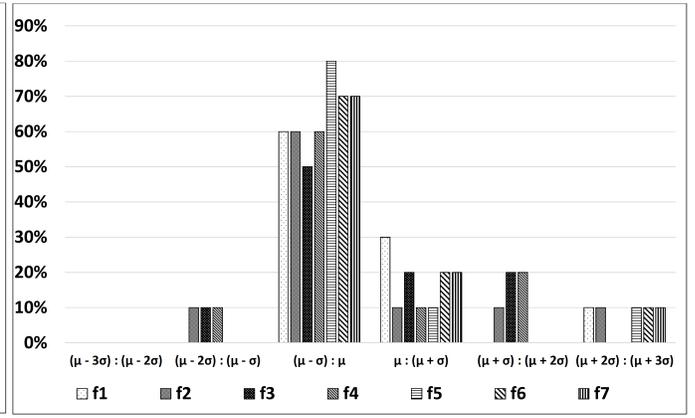


Fig. 4: Distribution of e_t for 10-fold Cross Validation

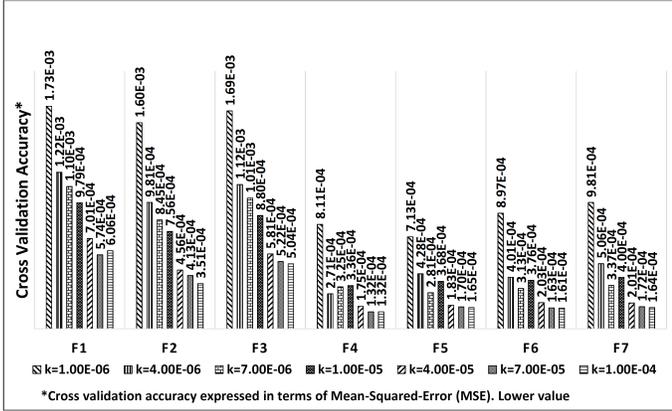


Fig. 5: Sample size estimation using 2-fold Cross Validation

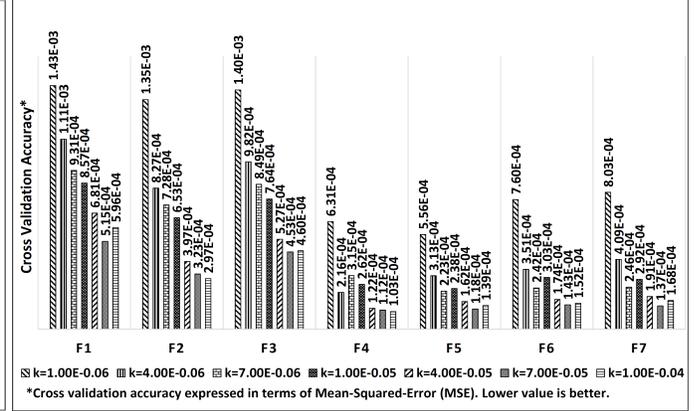


Fig. 6: Sample size estimation using 10-fold Cross Validation

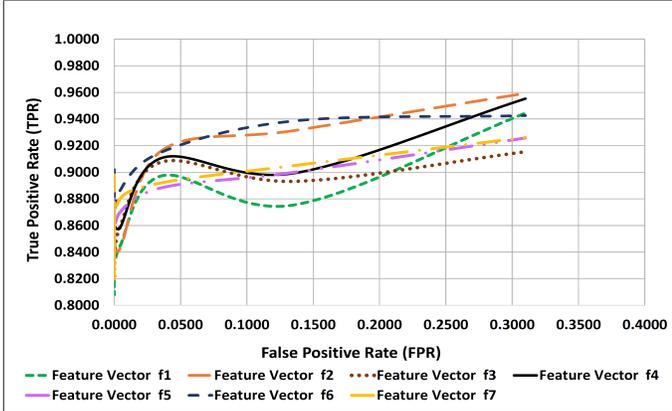


Fig. 7: ROC curve for the detectors

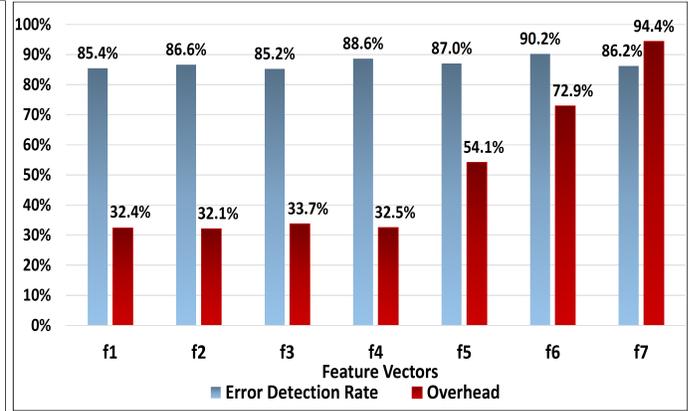


Fig. 8: Error detection rate and overhead data with $\tau_{opt} = 30$

value of 5. As the threshold value is increased in a fixed step-size of 5, the *true-positive* rate decreases at a much slower rate as compared to the *false-positive* rate, which demonstrates the effectiveness of the detectors. For a threshold value of 30, the *false-positive* rate shrinks to zero, while still providing a true positive rate of $> 85\%$ for all the detectors.

D. Error Detection Rate & Overhead Data

Figure 8 presents the *true-positive* error detection rate and overhead of each detector. With a threshold value of $\tau_{opt} = 30$, a high error-detection rate ($> 85\%$) is observed for all

the detectors. Detectors which use features f_1 , f_2 , f_3 and f_4 observe an average overhead of approximately 33%, whereas detectors with higher feature counts have an average overhead between 54% to 94%. This suggests that the former are the best choice for making applications resilient to errors. Further that fact that the addition of features between f_1 and f_4 has no effect on overhead suggests that their primary cost may not be the computations they perform, but rather other effects such as interference with the main computation's use of the memory hierarchy. We will examine these performance properties in more detail in future work.

VII. CONCLUSION AND FUTURE WORK

In this paper, we presented a novel approach using linear regression to efficiently approximate stencil kernels, validating it on the RTM stencil. We showed how the cost of training the model can be reduced via cross-validation driven stratified sampling to systematically reduce the training set size. We also showed how to find a *near-optimal* threshold value (τ_{opt}) for the detectors using ROC curve analysis and presented the error detection rate and the overhead data for the detectors. A high error detection rate reported by our detectors demonstrates the effectiveness of our approach. As a part of our future work, we'll analyze the error detection rate of our detectors against a subset of SDC-causing soft-errors which affect the convergence and stability of the solvers. Finally, in future, we plan to evaluate our approach on other ODE and PDE solvers based on explicit finite-difference method, such as problems related to computational fluid dynamics and electromagnetism.

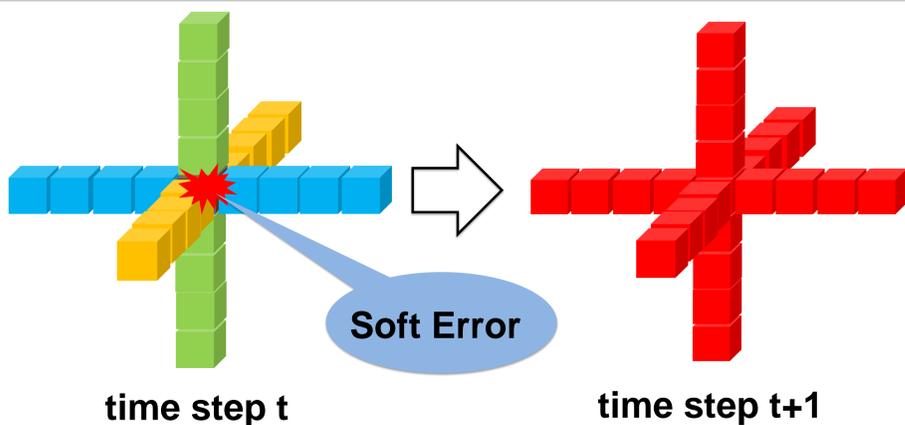
VIII. ACKNOWLEDGEMENT

The authors would like to thank the anonymous reviewers for their feedback and comments.

REFERENCES

- [1] D. F. Heidel, K. P. Rodbell, E. H. Cannon, C. Cabral, M. S. Gordon, P. Oldiges, and H. H. Tang, "Alpha-particle-induced upsets in advanced cmos circuits and technology," *IBM Journal of Research and Development*, vol. 52, no. 3, pp. 225–232, 2008.
- [2] P. N. Sanda, J. W. Kellington, P. Kudva, R. N. Kalla, R. B. McBeth, J. Ackaret, R. Lockwood, J. Schumann, and C. R. Jones, "Soft-error resilience of the IBM POWER6 processor," *IBM Journal of Research and Development*, vol. 52, no. 3, pp. 275–284, 2008.
- [3] S. Borkar, "Design Challenges of Technology Scaling," in *IEEE Micro*, vol. 19, no. 4, pp. 23–29, 1999.
- [4] S. Borkar, "Designing reliable systems from unreliable components: the challenges of transistor variability and degradation," in *IEEE Micro*, vol. 25, no. 6, pp. 10–16, 2005.
- [5] S. K. S. Hari, S. V. Adve, H. Naeimi, and P. Ramachandran, "Relyzer: Exploiting application-level fault equivalence to analyze application resiliency to transient faults," in *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2012.
- [6] V. C. Sharma, A. Haran, Z. Rakamaric, and G. Gopalakrishnan, "Towards Formal Approaches to System Resilience," in *Pacific Rim International Symposium on Dependable Computing (PRDC)*, 2013.
- [7] M. Snir, R. W. Wisniewski, J. A. Abraham, S. V. Adve, S. Bagchi, P. Balaji, J. Belak, P. Bose, F. Cappello, B. Carlson, A. A. Chien, P. Coteus, N. A. DeBardeleben, P. C. Diniz, C. Engelmann, M. Erez, S. Fazzari, A. Geist, R. Gupta, F. Johnson, S. Krishnamoorthy, S. Leyfer, D. Liberty, S. Mitra, T. Munson, R. Schreiber, J. Stearley, and E. V. Hensbergen, "Addressing failures in exascale computing," *International Journal of High Performance Computing Applications*, vol. 28, no. 2, pp. 129–173, 2014.
- [8] S. E. Michalak, W. N. Rust, J. T. Daly, A. J. DuBois, and D. H. DuBois, "Correctness field testing of production and decommissioned high performance computing platforms at los alamos national laboratory," in *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2014.
- [9] F. Cappello, A. Geist, B. Gropp, L. Kale, B. Kramer, and M. Snir, "Toward Exascale Resilience," *International Journal of High Performance Computing Applications*, vol. 23, no. 4, pp. 374–388, 2009.
- [10] F. Cappello, A. Geist, W. Gropp, S. Kale, B. Kramer, and M. Snir, "Toward exascale resilience: 2014 update," *Supercomputing frontiers and innovations*, vol. 1, no. 1, pp. 5–28, 2014.
- [11] M. Agarwal, B. Paul, M. Zhang, and S. Mitra, "Circuit failure prediction and its application to transistor aging," pp. 277–286, 2007.
- [12] C. Chen and M. Hsiao, "Error-correcting codes for semiconductor memory applications: A state-of-the-art review," *IBM Journal of Research and Development*, vol. 28, no. 2, pp. 124–134, 1984.
- [13] T. J. Dell, "A white paper on the benefits of chipkill-correct ecc for pc server main memory," *IBM Microelectronics Division*, pp. 1–23, 1997.
- [14] C. Slayman, "Cache and memory error detection, correction, and reduction techniques for terrestrial servers and workstations," *IEEE Transactions on Device and Materials Reliability*, vol. 5, no. 3, pp. 397–404, 2005.
- [15] S. Mitra, N. Seifert, M. Zhang, Q. Shi, and K. S. Kim, "Robust system design with built-in soft-error resilience," *Computer*, vol. 38, no. 2, pp. 43–52, 2005.
- [16] E. Baysal, D. D. Kosloff, and J. W. Sherwood, "Reverse time migration," *Geophysics*, vol. 48, no. 11, pp. 1514–1524, 1983.
- [17] "SORREL: A Soft-Error Detection Framework," http://www.cs.utah.edu/formal_verification/fmr/#sorrel.
- [18] A. R. Benson, S. Schmit, and R. Schreiber, "Silent error detection in numerical time-stepping schemes," *International Journal of High Performance Computing Applications*, 2014.
- [19] E. Berrocal, L. Bautista-Gomez, S. Di, Z. Lan, and F. Cappello, "Lightweight silent data corruption detection based on runtime data analysis for hpc applications," 2014.
- [20] P. Meaney, S. Swaney, P. Sanda, and L. Spainhower, "IBM z990 soft error detection and recovery," *IEEE Transactions on Device and Materials Reliability*, vol. 5, no. 3, pp. 419–427, 2005.
- [21] N. Wang and S. Patel, "Restore: symptom based soft error detection in microprocessors," in *International Conference on Dependable Systems and Networks (DSN)*, 2005.
- [22] N. Oh, P. P. Shirvani, and E. J. McCluskey, "Control-flow checking by software signatures," *IEEE Transactions on Reliability*, vol. 51, pp. 111–122, 2002.
- [23] R. Venkatasubramanian, J. Hayes, and B. Murray, "Low-cost on-line fault detection using control flow assertions," in *On-Line Testing Symposium (IOLTS)*, 2003.
- [24] C. Ding, C. Karlsson, H. Liu, T. Davies, and Z. Chen, "Matrix multiplication on gpus with on-line fault tolerance," in *International Symposium on Parallel and Distributed Processing with Applications (ISPA)*, 2011.
- [25] P. Du, A. Bouteiller, G. Bosilca, T. Herault, and J. Dongarra, "Algorithm-based Fault Tolerance for Dense Matrix Factorizations," in *Symposium on Principles and Practice of Parallel Programming (PPoPP)*, 2012.
- [26] J. Sloan, R. Kumar, and G. Bronevetsky, "An algorithmic approach to error localization and partial recomputation for low-overhead fault tolerance," in *International Conference on Dependable Systems and Networks (DSN)*, 2013.
- [27] M. McCool, A. D. Robison, and J. Reinders, "Chapter 10: Forward Seismic Simulation," in *Structured Parallel Programming: Patterns for Efficient Computation*, 1st ed., 2012, pp. 265–277.
- [28] J. Neyman, "On the two different aspects of the representative method: The method of stratified sampling and the method of purposive selection," *Journal of the Royal Statistical Society*, vol. 97, no. 4, pp. 558–625, 1934.
- [29] C. N. Park and A. L. Dudycha, "A cross-validation approach to sample size determination for regression models," *Journal of the American Statistical Association*, vol. 69, no. 345, pp. 214–218, 1974.
- [30] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *International Joint Conference on Artificial Intelligence (IJCAI)*, 1995.
- [31] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 1–27, 2011, software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>.
- [32] L. S. Aiken, S. G. West, and S. C. Pitts, *Multiple Linear Regression*. John Wiley & Sons, Inc., 2003.
- [33] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.
- [34] V. N. Vapnik, "The nature of statistical learning theory," 1995.
- [35] V. Vapnik, "An overview of statistical learning theory," *IEEE Transactions on Neural Networks*, vol. 10, no. 5, pp. 988–999, 1999.
- [36] A. P. Bradley, "The use of the area under the ROC curve in the evaluation of machine learning algorithms," *Pattern Recognition*, vol. 30, no. 7, pp. 1145–1159, 1997.
- [37] A. Thomas and K. Pattabiraman, "Error detector placement for soft computation," in *International Conference on Dependable Systems and Networks (DSN)*, 2013.

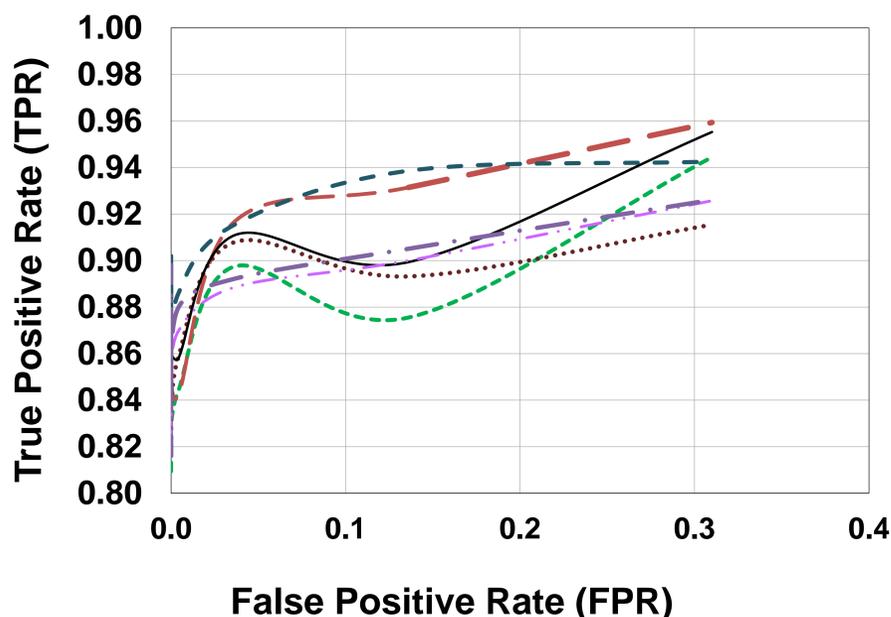
1. Motivation



Soft-error Propagation in 25-point RTM Stencil

- Protect stencil computations against soft errors
- Stencil based computations have high arithmetic intensity
- Target PDE solvers using stencil based computations
- Main memory and data cache often protected using ECC
- Focus on soft errors affecting CPU registers and its ALU
- Key idea is to use a variant of software level DMR
- Use approximate function for redundant computing
- Approximate function derived using machine learning

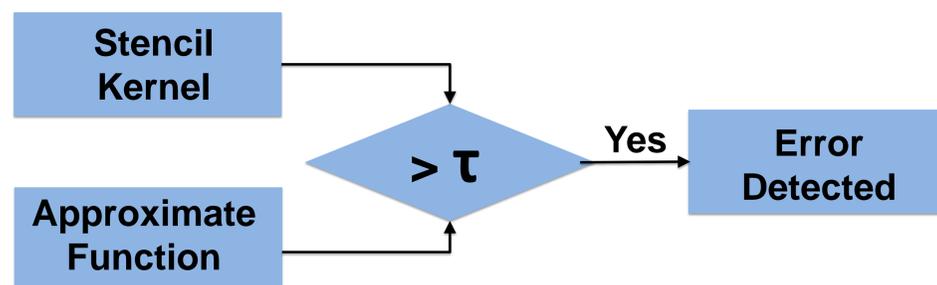
4. ROC Curve Analysis



- Feature Vector f1
- Feature Vector f2
- ... Feature Vector f3
- Feature Vector f4
- Feature Vector f5
- Feature Vector f6
- Feature Vector f7

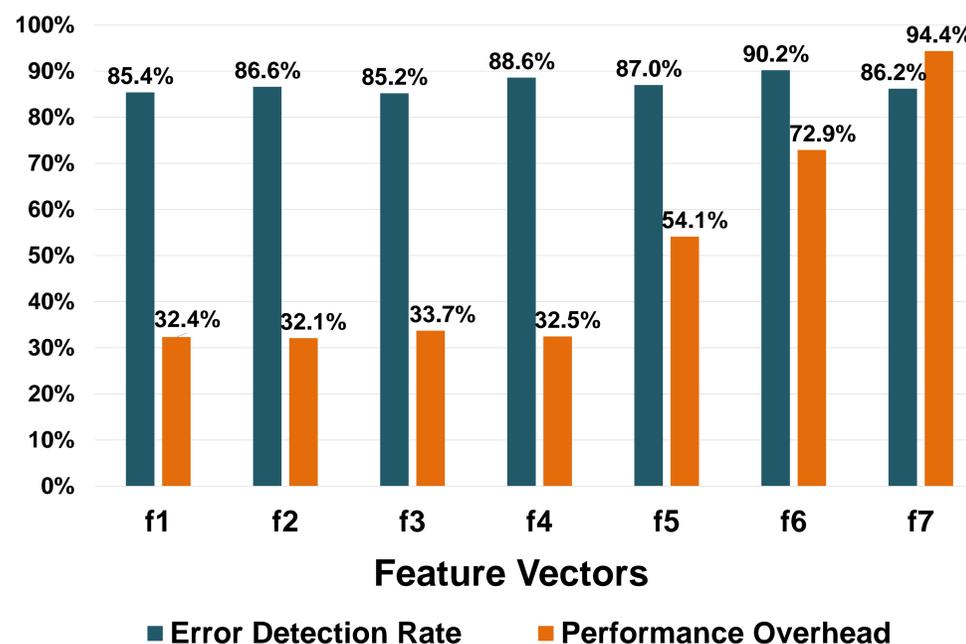
2. Proposed Error Detection Scheme

- Derive an approximate function for a stencil kernel
- Use SVM based regression analysis for this purpose [3]
- Use stratified sampling to reduce training data size [6]
- Approximate function requires fewer arithmetic operations
- Detector synthesis uses a software level DMR scheme
- Approximate function used for the redundant computation
- Perform ROC analysis for near optimal threshold estimation
- Estimated threshold should minimize false positive cases



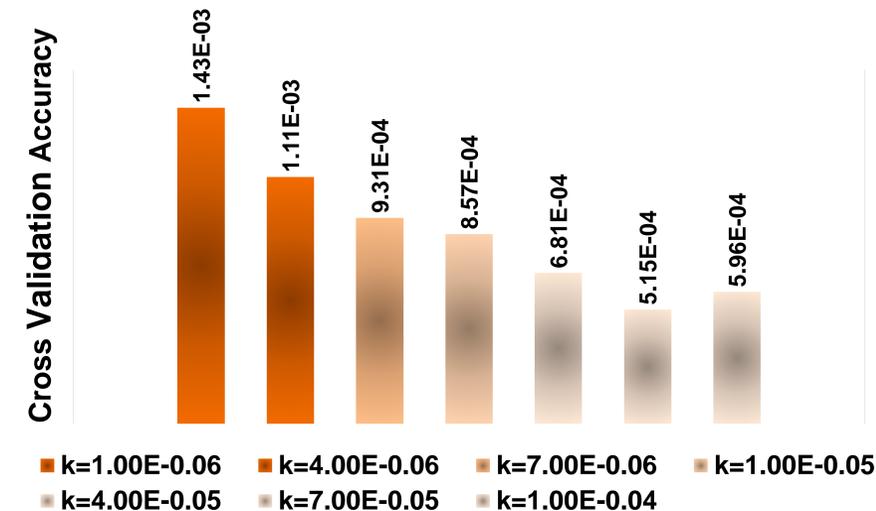
- A recent work uses regression analysis to detect runtime anomalies [5]
- Another work uses secondary lesser accurate solver [4]
- Our approach relies on deriving and using approximate functions for detector synthesis

5. Experimental Result



- A high error detection rate between 85%-90% is witnessed
- An overhead around 32%-33% for detectors based on f1-f4
- Detectors based on feature vectors f1-f4 are the feasible ones

3. Sample Size Estimation



6. Conclusions & Future Work

- An efficient approximation of RTM stencil using linear regression
- A cross validation driven stratified sampling approach to systematically reduce training data size
- A near optimal threshold estimation through ROC curve analysis
- In future, consider other applications which use finite difference explicit methods

7. Acknowledgements

- Supported in part by NSF Award CCF 1255776 and SRC contract 2013-TJ-2426

8. Closely Related References

- [1] C. N. Park and A. L. Dudycha, "A cross-validation approach to sample size determination for regression models," *Journal of the ASA*, vol. 69, no. 345, pp. 214–218, 1974.
- [2] E. Baysal, D. D. Kosloff, and J. W. Sherwood, "Reverse time migration," *Geophysics*, vol. 48, no. 11, pp. 1514–1524, 1983.
- [3] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," in *ACM Transaction on IST*, vol. 2, pp. 1–27, 2011, software available at <http://www.csie.ntu.edu.tw/~cjlin/libsvm>.
- [4] A. R. Benson, S. Schmit, and R. Schreiber, "Silent error detection in numerical time-stepping schemes," *International Journal in HPCA*, 2014.
- [5] E. Berrocal, L. Bautista-Gomez, S. Di, Z. Lan, and F. Cappello, "Lightweight silent data corruption detection based on runtime data analysis for hpc applications," 2014.
- [6] J. Neyman, "On the two different aspects of the representative method: The method of stratified sampling and the method of purposive selection," in *Journal of the RSS*, vol. 97, no. 4, pp. 558–625, 1934.