Università della Svizzera Italiana

Master Thesis

Accelerator for Event-based Failure Prediction

Problem Statement

Author:
Simon Maurer

Supervisor:
Prof. Miroslaw Malek

March 12, 2014

Contents

1 Introduction

This document provides an outline of the planned work for the master thesis "Accelerator for Event-based Failure Prediction". The document addresses the problem considered by the thesis, a state of the art outline as well as important questions and hypothesis that arise or are needed for the thesis. Further the document provides a list of necessary theory bases and introduces the design and experimenting methods applied in the thesis. Finally, the document gives an overview of necessary material in terms of literature and equipment and provides a broad time schedule.

2 Problem and Research Area

In today's live it becomes increasingly important, that computer systems are dependable. The reason being, that computer systems are used more and more in areas where the failure of such a system can lead to catastrophic events. Banking, public transportation and medical engineering are only a few examples of areas employing large and extremely complex systems. The increasing complexity of computer systems has a direct impact on their maintainability and testability. It is simply impossible to guarantee that a piece of software comes without any faults. On top of that, the same problematic arises with the hardware components which also may contain faulty parts but also get increasingly prone to failures due to decay of material.

In the event of a system failure it is of course desirable to fix the system as soon as possible in order to minimize the downtime of the system (maximize the availability). This can be accomplished by using different types of recovery techniques, e.g. Check-pointing (create checkpoints to roll back/forward), system replication (switch to a redundant system), fail over (reboot). All these techniques require a certain amount of time to complete the recovery process, time that is very expensive. In order to minimize this time, techniques have been developed to anticipate upcoming failures. Such a technique is described in [18].

The work presents a new algorithm to predict failures and compares the results with other techniques. The accuracy of the presented algorithm to predict failures proves to be better compared to the other techniques, has however the drawback of increased complexity and hence increased computation time. It is very important to keep the computation overhead very low in order to maximize the time between the prediction of a failure and the actual event of the failure. One way to decrease the computation time is to design a hardware accelerator for the prediction algorithm. The design of such an accelerator is outlined in this document.

3 State of the Art Outline

This section provides a brief outline of the state of the art in the different fields of research that are relevant for the thesis. This includes a small overview of failure prediction methods and existing solutions to accelerate failure prediction algorithms.

3.1 Failure Prediction

A very detailed overview of failure prediction methods is given in [19]. The survey discusses i.a. the techniques used as comparison in the main reference [5, 10, 11, 21] as well as the technique described in the main reference [18].

More recent work uses hardware counters of a general purpose CPU and combines them with software instrumentation to predict failures of single processes (e.g grep, flex, sed) [17, 24]. As industry heads more and more towards cloud computing, it has been proposed to use information of interaction between nodes (instead of analyzing single nodes) in order to analyze and predict failures of a distributed system [15, 20].

3.2 Accelerator

The main goal of this master thesis is to accelerate an adaptation of the forward algorithm. Proposals for a GPU based accelerator for the classic forward algorithm are described in [12, 14]. Further, several proposals to accelerate the Viterbi algorithm (which is closely related to the forward algorithm) have been published: [2] presents an architecture for a lightweight Viterbi accelerator designed for an embedded processor datapath, [6, 13, 16] describe a FPGA based accelerator for protein sequence HHM search and [22] describes i.a. an approach to accelerate the Viterbi algorithm from the HMMER library using GPUs.

Focusing on a more general approach for acceleration, [8] proposes an FPGA implementation of a parallel floating point accumulation and [23] describes the implementation of a vector processor on FPGA.

Quite a few research has been done on the question what type of technology should be used to accelerate certain algorithms: [4] presents a performance study of different applications accelerated on a multicore CPU, on a GPU and on a FPGA, [7] discusses the suitability of FPGA and GPU acceleration for high productivity computing systems (HPCS) without focusing on a specific application and [9] also focuses on HPCS but uses the Basic Linear Algebra Subroutines (BLAS) as comparison and also takes CPUs into account.

It may be interesting to also think about an acceleration of the model training. Similar work has been done by accelerating SVMs (Support Vector Machines): [3]

describes a FPGA based accelerator for the SVM-SMO (support vector machine - sequential minimal optimization) algorithm used in the domain of machine learning and [1] proposes a new algorithm and its implementation on a FPGA for SVMs.

4 Questions and Hypothesis

This section first lists the main questions that will be discussed, answered and verified in the master thesis:

Argumentation for the necessity of the work
The thesis will state why this work is necessary. Problems such as the huge amount of preliminary parametrization or the high computation effort will be discussed.

How to parallelize the algorithm
A theoretical analysis of the algorithm will identify the parts that can be parallelized and by using Amdahl's law the theoretical possible speedup will be computed.

Choice of accelerator technology
Different types of accelerators must be evaluated under the consideration of the properties of the adapted forward algorithm. Parameters like available parallelization, required dataflow, initial parametrization, different cost metrics, etc. need to be taken into account. Possible accelerator technologies are multicore processors, GPUs and FPGAs.

Practical implementation of the accelerator
Using the theoretical analysis of the algorithm as well as the choice of the accelerator type, practical decisions upon the type of acceleration techniques will be taken and discussed.

How to generate test data to verify the accelerator
In order to verify the work, test data needs to be available. Either the data is taken from some public repository, or it must be generated. Problematic is the fact, that algorithm-specific data is necessary. This problem will be discussed.

Second the hypotheses are stated, that will be taken into account to design the accelerator for the adapted forward algorithm:

- The reference algorithm is considered to be verified and tested. The verification in the master thesis will only focus on a valid theoretical and experimental computation of the speedup by comparing the accelerated version of the adapted forward algorithm with the serial computation on a CPU.

- The system using the accelerator provides standardized log events. The master thesis is not discussing the nature of these events. It is supposed, that the events are generated sequentially and fed to the accelerator through a defined interface (to be defined in the thesis).
- Only the online part of the reference algorithm will be accelerated. The acceleration of model training which is computed offline is optional and will most likely be neglected in this master thesis.

5 Theory Base

This section provides a brief overview of the computational steps done by the proposed algorithm. The following description should provide a complete blueprint of the adapted forward algorithm, that allows to implement it, but without any explications or proofs related to the formulation. To be able to understand the formal expression of the algorithm, first a definition of the used parameters is provided.

- N: number of states
- M: number of observation symbols
- L: observation sequence length
- R: number of cumulative probability distributions (kernels)

The delay of the event at time t_k with respect to the event at time t_{k-1} is described as

$$d_k = t_k - t_{k-1} \tag{1}$$

One part of the algorithm is the model training. This part is not described here. The features to be trained by the model training are however important in this context because they are used by the adapted forward algorithm. Following the features:

- π_i, forming the initial state probability vector $\boldsymbol{\pi}$ of size N
- $b_i(o_j)$, forming the emission probability matrix B of size $N \times M$
- p_{ij}, forming the matrix of limiting transmission probabilities P of size $N \times N$
- $\omega_{ij,r}$, the weights of the kernel r
- $\theta_{ij,r}$, the parameters of the kernel r

For simplification reasons, only one kernel is used. Due to this, the kernel weights can be ignored. Choosing the gaussian cumulative distribution results in the kernel parameters μ_{ij} and σ_{ij}. The adapted forward algorithm is defined as follows:

$$\alpha_0(i) = \pi_i b_{s_i}(O_0) \tag{2}$$

$$\alpha_k(j) = \sum_{i=1}^{N} \alpha_{k-1}(i) v_{ij}(d_k) b_{s_j}(O_k); \quad 1 \leq k \leq L \tag{3}$$

where

$$v_{ij}(d_k) = \begin{cases} p_{ij} d_{ij}(d_k) & \text{if } j \neq i \\ 1 - \sum\limits_{\substack{h=1 \\ h \neq i}}^{N} p_{ih} d_{ih}(d_k) & \text{if } j = i \end{cases} \tag{4}$$

with

$$d_{ij}(d_k) = \sum_{r=1}^{R} \omega_{ij,r} \kappa_{ij,r}(d_k | \theta_{ij,r}) \tag{5}$$

forming the matrix of cumulative transition duration distribution functions $D(d_k)$ of size $N \times N \times L$, and

$$\kappa_{ij,gauss}(d_k | \mu_{ij}, \sigma_{ij}) = 0.5 \Big[1 + \frac{2}{\pi} \Big(1 - \exp\big(- \frac{d_k - \mu_{ij}}{\sqrt{2}\sigma_{ij}} \big) \Big) \Big] \tag{6}$$

To prevent α from going to zero very fast, at each step of the forward algorithm a scaling is performed:

$$\alpha_k(i) = c_k \alpha_k(i) \tag{7}$$

with

$$c_k = \frac{1}{\sum\limits_{i=1}^{N} \alpha_k(i)} \tag{8}$$

then the sequence log-likelihood is computed:

$$\log(P(\boldsymbol{o}|\lambda)) = - \sum_{k=1}^{L} \log c_k \tag{9}$$

where $\lambda = \{\boldsymbol{\pi}, P, B, D(d_k)\}$, and finally the classification is performed:

$$\text{class}(s) = F \iff \max_{i=1}^{u} \big[\log P(\boldsymbol{s}|\lambda_i) \big] - \log P(\boldsymbol{s}|\lambda_0) > \log \theta \tag{10}$$

with

$$\theta = \frac{(r_{\bar{F}F} - r_{\bar{F}\bar{F}}) P(c_{\bar{F}})}{(r_{F\bar{F}} - r_{FF}) P(c_F)} \tag{11}$$

To calculate θ, the following parameters need to be set:

- $P(c_{\bar{F}})$: prior of non-failure class
- $P(c_F)$: prior of failure class
- $r_{\bar{F}\bar{F}}$: true negative prediction
- r_{FF}: true positive prediction
- $r_{\bar{F}F}$: false positive prediction
- $r_{F\bar{F}}$: false negative prediction

6 Method, Design and Experiments

In a first step, the proposed algorithm will be implemented in Octave[1]. The main reason for choosing Octave is on one hand the simplicity to implement complex algorithms with only a few lines of code and on the other hand the immediate revelation of parallelization possibilities. This implementation is also meant to help to fully understand the algorithm and possibly provide some ideas about optimizations related to the design of an accelerator.

The next step will consist of generating synthetic data to verify the correctness of the implementation. This data will also be used to verify further implementations and to compute a speedup by comparing the accelerated and the non-accelerated implementation. Knowing this, it is important to generate data that can easily be imported into Octave but also been used as input stream for a C/C++ implementation.

Further, the online part (sequence processing, classification) of the algorithm will be implemented in C++. To be able to run and verify the algorithm, the training data computed with Octave will be used. It is possible that due to optimization decisions it will also be necessary to implement the off-line part of the algorithm (training) in C++. In this case, the training data will be computed anew. To efficiently implement the algorithm in C++ a linear algebra library will be used (e.g. Armadillo[2], Eigen[3], LAPACK[4], etc.). It will be analyzed which library is best fitting for the purposes of this work. This sequential implementation will represent the reference to which the accelerated implementation will be compared to.

Finally, the accelerator will be designed. The design will include discussion concerning the choice of accelerator (e.g. FPGA, GPU, multiple cores). As a

[1] https://www.gnu.org/software/octave

[2] http://arma.sourceforge.net

[3] http://eigen.tuxfamily.org

[4] http://www.netlib.org/lapack

minimum approach, only the online part of the algorithm will be accelerated. However, due to optimization reasons it may be desirable to also accelerate the off-line part and alter the whole algorithm in such away as to compute the learning process online and hence being able to react to model aging problems. Again the generated synthetic data will be used to verify the algorithm as well as to compute a speedup compared to the non-accelerated implementation.

In addition to the steps listed above it would be very desirable to verify the accelerated algorithm with real data. To accomplish this, one idea is to set up the following experiment:

Run high-load procedures (e.g. prime number computation) on a undervolted CPU-core and observe the internal hardware counters of the CPU-core. Whenever a counter overflows, the counter id and the time of the overflow is recorded. Additionally also the time and the type of failures of the CPU are recorded. This data can then be fed to the algorithm:

- each recorded counter id corresponds to an event
- the difference between two consecutive events corresponds to the delay $d_k = t_{k+1} - t_k$
- the failure events are used as oracle to compute F-measure, Precision and Recall

Should the experiment result in a reasonable F-measure, it would not only have been showed that the accelerated algorithm works on real data, but also that it is possible to predict HW failures of a CPU-core by using hardware counters. This experiment must first be performed with a non-optimized version of the algorithm (i.e. the exact same implementation as described in [18]) before it is repeated with an optimized version (if any optimization has been implemented).

Note that there are strong doubts that the speedup of the accelerator will be enough to actually perform the experiment mentioned above. Most likely the events will occur far to fast for the algorithm to compute the result in time.

7 Time Schedule

This section provides a broad overview of the time schedule to finish the master thesis. This schedule is not accurate and serves only as a guideline. It will certainly change due to the fact, that the date of the defense is not yet available.

End of March

The decision on the type of accelerator has been taken. The process to get to this decision involves the study of the state of the art in accelerator technologies and the study of the algorithm to accelerate.

Mid of April
A verification scheme is available as well as the data to verify the accelerator. This also includes the exact choice of the device on which the accelerator will be implemented and tested.

Mid of Mai
The final implementation of the accelerator is available as well as a serial implementation serving as a reference. The verification phase can start. Until this point all acceleration decisions have been discussed and documented.

Mid of June
The first draft of the documentation is available.

End of June
The documentation has been finalized.

References

[1] D. Anguita, A. Boni, and S. RIDELLA, *A digital architecture for support vector machines: theory, algorithm, and fpga implementation*, IEEE Transactions on Neural Networks, 14 (2003), pp. 993–1009.

[2] M. Azhar, M. Sjalander, H. Ali, A. Vijayashekar, T. Hoang, K. Ansari, and P. Larsson-Edefors, *Viterbi accelerator for embedded processor datapaths*, in IEEE International Conference on Application-Specific Systems, Architectures and Processors, ASAP, July 2012, pp. 133–140.

[3] S. Cadambi, I. Durdanovic, V. Jakkula, M. Sankaradass, E. Cosatto, S. Chakradhar, and H. Graf, *A massively parallel FPGA-based coprocessor for support vector machines*, in IEEE Symposium on Field Programmable Custom Computing Machines, FCCM, April 2009, pp. 115–122.

[4] S. Che, J. Li, J. Sheaffer, K. Skadron, and J. Lach, *Accelerating compute-intensive applications with gpus and fpgas*, in Symposium on Application Specific Processors, SASP, June 2008, pp. 101–107.

[5] C. Domeniconi, C.-s. Perng, R. Vilalta, and S. Ma, *A classification approach for prediction of target events in temporal sequences*, in Principles of Data Mining and Knowledge Discovery, T. Elomaa, H. Mannila, and H. Toivonen, eds., vol. 2431 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2002, pp. 125–137.

[6] A. Jacob, J. Lancaster, J. Buhler, and R. Chamberlain, *Preliminary results in accelerating profile hmm search on fpgas*, in IEEE International Parallel and Distributed Processing Symposium, IPDPS, March 2007, pp. 1–8.

[7] D. H. Jones, A. Powell, C.-S. Bouganis, and P. Y. K. Cheung, *Gpu versus fpga for high productivity computing*, in International Conference on Field Programmable Logic and Applications, FPL, Washington, DC, USA, 2010, IEEE Computer Society, pp. 119–124.

[8] E. Kadric, P. Gurniak, and A. DeHon, *Accurate parallel floating-point accumulation*, in IEEE Symposium on Computer Arithmetic (ARITH), ARITH, April 2013, pp. 153–162.

[9] S. Kestur, J. D. Davis, and O. Williams, *Blas comparison on fpga, cpu and gpu*, in IEEE Symposium on VLSI, ISVLSI, Washington, DC, USA, 2010, IEEE Computer Society, pp. 288–293.

[10] T.-T. Lin and D. Siewiorek, *Error log analysis: statistical modeling and heuristic trend analysis*, IEEE Transactions on Reliability, 39 (1990), pp. 419–432.

[11] T.-T. Y. Lin, *Design and evaluation of an on-line predictive diagnostic system*, PhD thesis, Carnegie-Mellon University, Pittsburgh, PA, 1988.

[12] C. Liu, *cuhmm: a cuda implementation of hidden markov model training and classification*, tech. rep., Johns Hopkins University, Mai 2009. Project Report for the Course Parallel Programming.

[13] R. P. Maddimsetty, J. Buhler, R. D. Chamberlain, M. A. Franklin, and B. Harris, *Accelerator design for protein sequence hmm search*, in International Conference on Supercomputing, ICS, New York, NY, USA, 2006, ACM, pp. 288–296.

[14] E. Neumann, *Berechnung von hidden markov modellen auf grafikprozessoren unter ausnutzung der speicherhierarchie*, diploma thesis, Humboldt University of Berlin, Berlin, Germany, Mai 2011.

[15] A. Oliner, A. Kulkarni, and A. Aiken, *Using correlated surprise to infer shared influence*, in IEEE/IFIP International Conference on Dependable Systems and Networks, DSN, June 2010, pp. 191–200.

[16] T. Oliver, L. Yeow, and B. Schmidt, *High performance database searching with hmmer on fpgas*, in IEEE International Parallel and Distributed Processing Symposium, IPDPS, March 2007, pp. 1–7.

[17] B. Ozcelik and C. Yilmaz, *Seer: A lightweight online failure prediction approach.* http://research.sabanciuniv.edu/23359/, January 2014.

[18] F. Salfner, *Event-based Failure Prediction*, PhD thesis, Humboldt-University of Berlin, February 2008.

[19] F. Salfner, M. Lenk, and M. Malek, *A survey of online failure prediction methods*, ACM Comput. Surv., 42 (2010), pp. 10:1–10:42.

[20] F. Salfner and P. Tröger, *Predicting Cloud Failures Based on Anomaly Signal Spreading*, in Dependable Systems and Networks, IEEE, 2012.

[21] R. Vilalta and S. Ma, *Predicting rare events in temporal domains*, in IEEE International Conference on Data Mining, ICDM, December 2002, pp. 474–481.

[22] J. Walters, V. Balu, S. Kompalli, and V. Chaudhary, *Evaluating the use of gpus in liver image segmentation and hmmer database searches*, in IEEE International Parallel and Distributed Processing Symposium, IPDPS, May 2009, pp. 1–12.

[23] H. Yang, S. Ziavras, and J. Hu, *Fpga-based vector processing for matrix operations*, in International Conference on Information Technology, ITNG, April 2007, pp. 989–994.

[24] C. Yilmaz and A. Porter, *Combining hardware and software instrumentation to classify program executions*, in ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE, New York, NY, USA, 2010, ACM, pp. 67–76.