Accelerator for Event-based Failure Prediction

Subtitle

Master's Thesis submitted to the
Faculty of Informatics of the *Università della Svizzera Italiana*
in partial fulfillment of the requirements for the degree of
Master of Science in Informatics
Embedded Systems Design

presented by
Simon Maurer

under the supervision of
Prof. Miroslaw Malek

Janaury 2014

I certify that except where due acknowledgement has been given, the work presented in this thesis is that of the author alone; the work has not been submitted previously, in whole or in part, to qualify for any other academic award; and the content of the thesis is the result of work which has been carried out since the official commencement date of the approved research program.

Simon Maurer
Lugano, 29. Janaury 2014

Abstract

Acknowledgements

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue.

Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

Contents

Figures

Tables

Chapter 1

Introduction

1.1 Problem Statement

1.2 Motivation

The email of Felix left some doubts to whether the acceleration of the algorithm is useful. The following list will give some arguments to justify the work.

Too many parameters to be identified, estimated and set
Considering an embedded system, this is usually not a problem because the parameters are defined during the design phase and will never be changed afterwards.

Limited performance scalability
There are studies available claiming otherwise. The discussion of Neumanns work will provide some arguments against this statement.

Industry trends point towards cloud
In embedded systems it will still be beneficial to predict failures of single nodes. It is however important to keep the power and computational footprint low. This will be one of the major challenges. On the other hand, I think it would also be possible to also use this algorithm to monitor a distributed system and predict failures. It is only a matter of getting defining the events to feed to the algorithm.

1.3 Contributions

1.4 Document Structure

Chapter 2

State of the Art

2.1 Failure Prediction

2.2 Accelerator

Chapter 3

Event-based Failure Prediction

This section provides a brief overview of the computational steps done by the proposed algorithm [20].

brief description of the idea behind the algorithm, HSMM, Events, etc

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

$$d_k = t_k - t_{k-1} \tag{3.1}$$

3.1 Data Processing

3.2 Model Training

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 π 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

3.3 Sequence Processing

The following description will provide a complete blueprint of the adapted forward algorithm, that allows to implement it, but without any explanations or proofs related to the formulation. The adapted forward algorithm is defined as follows:

$$\alpha_0(i) = \pi_i b_{s_i}(O_0) \tag{3.2}$$

$$\alpha_k(j) = \sum_{i=1}^{N} \alpha_{k-1}(i) v_{ij}(d_k) b_{s_j}(O_k); \quad 1 \le k \le L \tag{3.3}$$

where

$$v_{ij}(d_k) = \begin{cases} p_{ij} d_{ij}(d_k) & \text{if } j \neq i \\ 1 - \sum\limits_{\substack{h=1 \\ h \neq i}}^{N} p_{ih} d_{ih}(d_k) & \text{if } j = i \end{cases} \tag{3.4}$$

with

$$d_{ij}(d_k) = \sum_{r=1}^{R} \omega_{ij,r} \kappa_{ij,r}(d_k | \theta_{ij,r}) \tag{3.5}$$

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

For simplification reasons, only one kernel is used. Due to this, the kernel weights can be ignored. Equation 3.5 can then be simplified:

$$d_{ij}(d_k) = \kappa_{ij}(d_k | \theta_{ij}) \tag{3.6}$$

Choosing the gaussian cumulative distribution results in the kernel parameters μ_{ij} and σ_{ij}:

$$\kappa_{ij,gauss}(d_k | \mu_{ij}, \sigma_{ij}) = \frac{1}{2}\left[1 + \text{erf}\left(\frac{d_k - \mu_{ij}}{\sqrt{2}\sigma_{ij}}\right)\right] \tag{3.7}$$

maybe use sequence likelihood and then explain about scaling

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

$$\alpha_k(i) = c_k \alpha_k(i) \tag{3.8}$$

with

$$c_k = \frac{1}{\sum\limits_{i=1}^{N} \alpha_k(i)} \tag{3.9}$$

begin new sentence and explain log likelihood

then the sequence log-likelihood is computed:

$$\log(P(\mathbf{o}|\lambda)) = -\sum_{k=1}^{L} \log c_k \tag{3.10}$$

where $\lambda = \{\pi, P, B, D(d_k)\}$.

3.4 Classification

explain classification

and finally the classification is performed:

$$\text{class}(s) = F \iff \max_{i=1}^{u} \left[\log P(\boldsymbol{s}|\lambda_i) \right] - \log P(\boldsymbol{s}|\lambda_0) > \log \theta \tag{3.11}$$

with

$$\theta = \frac{(r_{\bar{F}F} - r_{\bar{F}\bar{F}})P(c_{\bar{F}})}{(r_{F\bar{F}} - r_{FF})P(c_F)} \tag{3.12}$$

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

Chapter 4

Acceleration

Challanges of the acceleration

- implementation of exp and log function (LUT, Taylor, ...)
- floating points vs fixed points
- precision
- choice of accelerator (Type, Model)
- find avaliable options for parallelization

Ideas on how to accelerate the online part of the algorithm

- use high speed multiplier-accumulator (MAC) devices on a FPGA
- use MACs only on integer numbers and compute FP later manually
- minimize division (compute scaling factor once and then multiply)
- if precision allows use pipelining to precompute the factor $b(j, o[k]) * v(i, j, k)$
- precompute known factors and store them in order to simlify online computation (e.g. parts of the kernel, classification thershold, ...).
- ...

Possible optimizations of the algorithm:

- use a regularization term in the cost function to prevent overfitting
- incorporate the offline part of the algorithm into the online part in order to deal with model aging
- ...

4.1 Theoretical Analysis

4.1.1 Pre-Computation of Known Parameters

To reduce the computation effort of the algorithm during runtime, it may be desirable to pre-compute factors composed of known parameters and store them in the ROM of the accelerator. This may increase the necessary ROM on the accelerator but reduce the number of costly computations.

Considering this, the equation 3.7 can be rewritten in the following matter, by using the exponential properties:

$$\kappa_{ij,gauss}(d_k|\mu_{ij},\sigma_{ij}) = c_{ij,1} - c_{ij,2}\exp(c_{ij,3}d_k) \tag{4.1}$$

with the factors

$$c_{ij,1} = 0.5 + \frac{1}{\pi} \tag{4.2}$$

,

$$c_{ij,2} = \frac{1}{\pi}\exp(\frac{\mu_{ij}}{\sqrt{2}\sigma_{ij}}) \tag{4.3}$$

and

$$c_{ij,3} = -\frac{1}{\sqrt{2}\sigma_{ij}} \tag{4.4}$$

By using the equations 3.6 and 4.1, 3.4 can now be written as

$$v_{ij}(d_k) = \begin{cases} c'_{ij,1} - c'_{ij,2}\exp(c_{ij,3}d_k) & \text{if } j \neq i \\ 1 - \sum\limits_{\substack{h=1 \\ h\neq i}}^{N} \left[c'_{ih,1} - c'_{ih,2}\exp(c_{ih,3}d_k)\right] & \text{if } j = i \end{cases} \tag{4.5}$$

with the factors

$$c'_{ij,1} = p_{ij}c_{ij,1} = p_{ij}(0.5 + \frac{1}{\pi}) \tag{4.6}$$

and

$$c'_{ij,2} = p_{ij}c_{ij,2} = \frac{p_{ij}}{\pi}\exp(\frac{\mu_{ij}}{\sqrt{2}\sigma_{ij}}) \tag{4.7}$$

The factors 4.6, 4.7 and 4.4 can be precomputed and stored in memory. Upon the computation of 4.5 the factors are read out of the ROM on the accelerator. Without pre-computation, the necessary ROM storage to provide the model parameters on the accelerator is

$$N\pi_{\#bit} + NMb_{\#bit} + N^2(p_{\#bit} + \mu_{\#bit} + \sigma_{\#bit})\,\text{bit} \tag{4.8}$$

With the pre-computation described above

$$N\pi_{\#bit} + NMb_{\#bit} + N^2(c1'_{\#bit} + c2'_{\#bit} + c3_{\#bit})\,\text{bit} \tag{4.9}$$

of available ROM storage is necessary.

Check speed and memory difference

The log of equation 3.12 can also be precomputed as it is used in equation 3.11 and all parameters are known before runtime.

4.1.2 Serial Implementation

```
% computation of the extended forward algorithm
% @param N:             number of states
% @param L:             number of observation symbols
% @param PI:            initial state probability vector. size N
% @param B:             matrix of emission probabilities. size N, L
% @param cdf_param:     parameters for the cdf
% @return alpha:        forward variables. size N, L
% @return scale_coeff:  coefficients. size L
function [alpha scale_coeff] = forward_s(N, L, PI, B, mu, cdf_param)
    k = 1;
    % initialize forward variables
    % read o0: index of first observation symbol
    for i=1:N,
        alpha(i, 1) = PI(i)*B(i, o0);
    end
    % scaling
    alpha_sum = 0;
    for i=1:N,
        alpha_sum += alpha(i, 1);
    end
    scale_coeff(1) = 1 / alpha_sum;
    for i=1:N,
        alpha(i, 1) *= scale_coeff(1);
    end
    % forward algorithm
    while (k < L),
        % read ok: index of k-th observation symbol
        % read dk: delay of k-th observation symbol
        % compute one step of forward algorithm
        [alpha(:, k+1) scale_coeff(k+1)] = ...
            forward_step_s(N, dk, alpha(:, k), B(:, ok), cdf_param);
        k++;
    end
end
```

```
% computation of one step of the extended forward algorithm
% @param N:             number of states
% @param dk:            delay of k-th observation symbol
% @param alpha:         forward variables of step k-1. size N
% @param B:             emission probabilities of step k. size N
% @param cdf_param:     parameters for the cdf
% @param alpha_new:     forward variables of step k. size N
% @return scale_coeff:  coefficient of step k
function [alpha_new scale_coeff] = forward_step_s.m(N, dk, alpha, B, cdf_param)
    % compute transistion probabilities
    tp = compute_tp(N, dk, cdf_param);
    % compute forward algorithm
    for j=1:N,
        alpha_new(j) = 0;
        for i=1:N,
            alpha_new(j) += alpha(i) * tp(i, j);
        end
        alpha_new(j) *= B(j);
    end
    % scaling
    alpha_sum = 0;
    for i=1:N,
        alpha_sum += alpha_new(i);
    end
    scale_coeff = 1 / alpha_sum;
    for i=1:N,
        alpha_new(i) *= scale_coeff;
    end
end
```

```
% computation of the extended transition probabilities
% @param N:         number of states
% @param dk:        delay of k-th observation symbol
% @param cdf_param: parameters for the cdf
% @return v:        extended transition probabilities
function [v] = compute_tp(N, dk, cdf_param)
    % compute all elements of v
    for i=1:N,
        for j=1:N,
            v(i, j) = normcdf(dk, cdf_param.mu(i, j), cdf_param.sigma(i, j));
        end
    end
    % correct diagonal elemnts of v
    for i=1:N,
        for j=1:N,
            v_sum(i) += v(i, j);
        end
    end
    for i=1:N,
        v_sum(i) -= v(i, i);
        v(i, i) = 1 - v_sum(i);
    end
end
```

4.1.3 Parallelization

4.1.4 Precision

4.2 Choice of Accelerator Type

4.2.1 CPU

- pro
  - fast and easy implementation
  - high precision
  - high frequency
- contra
  - high power consumption
  - limited parallelization
  - large overhead for simple instructions
  - fixed architecture (memory, computation units)

4.2.2 GPU

- pro
  - parallelization options for a low price
  - fast onboard memory
  - high frequency
  - high precision
  - simple implementation
- contra
  - high power consumption
  - overhead for simple instructions
  - fixed architecture (memory, computation units)

4.2.3 FPGA

- pro
  - low power consumption
  - low overhead
  - flexibility
- contra
  - low frequency
  - parallelization is expensive
  - precision is expensive
  - complex implementation

4.2.4 ASIC

- pro
  - very low power consumption
  - no overhead
  - very flexible
- contra
  - very expensive
  - very complex implementation

4.2.5 Conclusion

4.3 Implementation

Chapter 5

Testing and Verification

5.1 Log Standard

5.2 Metrics

5.3 Automated Log Generation

5.4 Online Log Generation

Chapter 6

Results

6.1 Speedup

6.2 Accuracy

Chapter 7

Conclusion

7.1 Achievements

7.2 Future Work

Appendix A

Some material

Bibliography

[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] J. Detrey and F. de Dinechin, *A parameterized floating-point exponential function for FPGAs*, in IEEE International Conference on Field-Programmable Technology, ICFPT, Dec 2005, pp. 27–34.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

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

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

[13] 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.

[14] 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.

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

[16] 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.

[17] 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.

[18] B. OZCELIK AND C. YILMAZ, *Seer: A lightweight online failure prediction approach*. http://research.sabanciuniv.edu/23359/, January 2014.

[19] R. POTTATHUPARAMBIL AND R. SASS, *Implementation of a cordic-based double-precision exponential core on an FPGA*, Proceedings of RSSI, (2008).

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

[21] F. SALFNER, M. LENK, AND M. MALEK, *A survey of online failure prediction methods*, ACM Comput. Surv., 42 (2010), pp. 10:1–10:42.

[22] F. SALFNER AND P. TRÖGER, *Predicting Cloud Failures Based on Anomaly Signal Spreading*, in Dependable Systems and Networks, IEEE, 2012.

[23] R. VILALTA AND S. MA, *Predicting rare events in temporal domains*, in IEEE International Conference on Data Mining, ICDM, December 2002, pp. 474–481.

[24] 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.

[25] 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.

[26] 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.