Using Continuous Statistical Machine Learning to Enable Performance Prediction in Hybrid Functional/Cycle Accurate Instruction Set Simulators

Daniel Powell, 0347677

Fourth Year Project Report
Computer Science and Electronics
School of Informatics
University of Edinburgh
2009
Abstract

Functional instruction set simulators perform instruction-accurate simulation of benchmarks at high instruction rates. Unlike their slower, but cycle-accurate, counterparts however, they are not capable of providing cycle counts due to the higher level of hardware abstraction. In this report we present a novel approach to performance prediction based on statistical machine learning utilising a hybrid functional and cycle-accurate simulator. We introduce the concept of continuous learning to simulation whereby new training data points are acquired on demand and used for on-the-fly updates of the prediction model. Furthermore, we show how statistical regression can be adapted to reduce the cost of these updates during a performance-critical simulation. For a state-of-the-art simulator modelling the ARC 750D embedded processor core we demonstrate that our approach is highly accurate, with average error $< 2.5\%$ whilst achieving a speed-up of $\approx 1.5$ times that of the baseline cycle-accurate simulation. When extended to use a modern JIT-compiled simulation for the functional simulation, this would enable the full simulation of e.g. complex multimedia applications.
Acknowledgements

Many thanks to my supervisors Björn Franke and Nigel Topham.
Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(Daniel Powell, 0347677)
# Table of Contents

1 Introduction .............................. 1
   1.1 Overview of Problem ......................... 1
   1.2 Aims and Contributions ...................... 2
   1.3 Overview .................................. 3

2 Related Work .............................. 5
   2.1 Fast Instruction Set Simulation ................. 5
   2.2 Sampling Simulation .......................... 5
   2.3 Simulation Based Performance Prediction ......... 6
   2.4 Statistical Simulation ....................... 6

3 Background .............................. 7
   3.1 Functional Simulation ......................... 7
   3.2 Cycle Accurate Simulation ...................... 8
   3.3 Statistical Regression ....................... 9
      3.3.1 Linear Regression ......................... 9
      3.3.2 Locally Weighted Projection Regression .... 10
      3.3.3 Gaussian Process Regression ............... 11
      3.3.4 Confidence Intervals ..................... 12

4 Methodology .............................. 13
   4.1 Overview .................................. 13
   4.2 Incremental Linear Regression .................. 13
   4.3 Feature Selection ............................. 14
      4.3.1 MRMR Toolset ............................. 15
      4.3.2 Variance of a Feature ..................... 15
      4.3.3 Count of Non-Zero Entries of a Feature .... 15
   4.4 Continuous Learning ......................... 16
      4.4.1 Operation ................................ 16
      4.4.2 Dealing With A Less Confident Prediction .... 17
      4.4.3 Altering the Timeslice Length ............... 17
      4.4.4 Transference of Training Data .............. 18

5 Implementation ........................... 19
   5.1 Simulator Overview .......................... 19
   5.2 Prediction Library Overview .................. 20
      5.2.1 Incremental Linear Regression ............. 20
Chapter 1

Introduction

1.1 Overview of Problem

Instruction set simulation is a frequently used method by developers of custom hardware and software. Instruction set simulators (ISS) allow developers to design and test their hardware prior to creating a prototype allowing hardware designers to be much more flexible in their approach to designing a new device. It allows for verification of the design before a large investment is made and allows for the performance of a device to be determined early in the process allowing for incremental design improvements.

For software developers instruction set simulation provides a flexible development platform from which to test operation and performance of a piece of software without access to a physical device. It also allows for many concurrent executions of a program without the need for investment in an excessive quantity of hardware.

Due to these large varying uses of instruction set simulators many different designs exist, ranging from purely functional versions simulating only the operational output of a program and not the underlying hardware, to completely cycle accurate simulators that take exact timing models of the hardware to provide a precise performance metric of a simulation. Understandably, these designs vary broadly in speed and information collected, with functional simulators being extremely fast due to abstracting away all hardware details and only collecting various high-level statistics, and cycle accurate simulators being much slower due to the much finer level of detail that they simulate.

With newly developing embedded technologies with very specific task requirements new application-specific instruction set processors (ASIP) are being created with special instructions sets customised for their purpose. The design of such devices is extremely challenging and requires verification and performance metrics as early as possible in the design process. This requires both fast and timing-accurate simulators for the design process, two mutually exclusive requirements. For this reason much research is being performed in this field to create fast and accurate simulation methods [26, 8, 9], most recent developments include an integrated JIT compiler to directly modify the state of the simulator using native binary code, instead of interpreting the executable compiled for the new hardware device. However this process uses data profiling to control the optimisation process and often requires many executions of a simulation.

This work is driven by the vast difference in speed between cycle-accurate and
Chapter 1. Introduction

*instruction-accurate* instruction set simulators, and their relation to the underlying accuracy of performance metrics. As an example of the large difference in speed cycle-accurate simulators typically operate at $20K - 100K$ instructions per second, compared to $200 - 400$ million instructions per second by recent JIT-translating simulators [5, 21, 26] and greater than $1$ GIPS instructions per second achieved by simple functional simulators collecting no high-level information. This performance gap of several magnitudes makes functional and JIT-translating simulators a very attractive choice for embedded hardware and software developers, however due to their abstraction away from the hardware they can only provide high-level statistical information and not the full performance metrics desired.

1.2 Aims and Contributions

The purpose of this report is to develop and test a novel performance prediction methodology suitable for use in a hybrid functional/cycle accurate simulator. This method is intended to enable the accurate estimation of the performance of a new hardware design whilst retaining the higher simulation speed of a functional simulator. This is achieved through the use of machine learning techniques whereby statistical regression techniques are employed with an initial training period in cycle accurate mode, mapping the high-level statistical information collected by the functional simulator onto the precise cycle count for that period. After this initial training period the cycle count is updated by performing predictions based upon the high-level information collected. This of course will introduce inaccuracies into the provided cycle count due to the nature of statistical regression, hence a confidence metric is monitored whilst performing predictions. Should the confidence metric widen beyond a specific, threshold the simulation falls back into cycle-accurate mode and updates the training data in an attempt to minimise these inaccuracies, before returning to the faster functional simulation.

This reports contributes to current work in the field by further enabling the high speed simulation of new hardware designs whilst still providing accurate performance estimations. It automates the process of machine learning into one easy to use framework, and improves on the accuracy of similar work [8, 9] by predicting across a smaller timeslice, customising the training set for each execution and monitoring the accuracy of the predictions being performed.

For this novel methodology to succeed a number of challenges must be overcome. Due to the performance sensitive nature of benchmark simulation, traditional statistical regression is not very well suited for this purpose due to its high computational complexity. This issue is attempted to be solved with *incremental* linear regression, exploiting the fact that updates of the prediction matrix are rare and only involve adding few data points. Furthermore, we can only detect that a prediction may be unreliable after its simulation and calculation. The actions performed when a prediction is determined to be unreliable can have negative effects on both the performance and accuracy of the simulator. For this we attempt two different methods: a conservative and an aggressive method using *rollback-and-replay* and a *patch scheme*, respectively.

The issue of determining which high-level information is relevant to prediction accuracy is also addressed through the investigation of several feature selection methods. An on-line feature selection method is also integrated with the simulator to automate
this process during simulation as the relevant features are likely to change dependant upon each simulation.

This methodology is integrated with a state-of-the-art hybrid ISS for the ARC 750D embedded processor [26]. Simulations are performed across a vast range of benchmarks to correctly monitor the performance and accuracy of the new methodology in a range of circumstances, with the intention of enabling full simulation of e.g. complex multimedia applications.

The new approach is intended to be portable, fully automated and not require any user interference to enable fast yet simple simulation of a new hardware design.

1.3 Overview

This report is structured as follows. In chapter 2 we discuss the large body of related work. This is followed by a presentation of background material on functional and cycle-accurate simulation and “classic” statistical regression methods in chapter 3. In chapter 4 we introduce our novel continuous learning based performance prediction methodology and its integration into a hybrid functional/cycle-accurate instruction set simulator. A short description detailing of our implementation into a state-of-the-art simulator is covered in chapter 5. The results of our empirical evaluation are presented and discussed in chapter 6 before we summarise and conclude in chapter 7.
Chapter 2

Related Work

2.1 Fast Instruction Set Simulation

Instruction set simulation is an active research area. Researchers have focused in depth on faster simulation to aid hardware and software developers with ensuring functionality and calculating performance metrics. Research has included re-targetable simulators enabling simple exploration of the architecture design space [15, 19], and compiled simulators increasing the speed of simulation dramatically at the expense of an initial compilation stage and possibly further compilation stages should code be modified [20].

More recently, hybrid compiled simulation approaches combining both the standard interpretive and compiled simulation utilising a JIT binary translator, whereby blocks of code are executed in interpretive mode until they have been executed a number of times beyond a threshold. In this circumstance code is recompiled to a native binary and then future executions of these “hot blocks” are performed using the newly compiled code. These methods are presented in e.g. [5, 21, 26].

2.2 Sampling Simulation

Researchers have also investigated sampling simulations, whereby the simulation being ran is profiled, and predictions towards performance are made based upon the profiling. This method requires a hybrid simulator integrating both functional and cycle-accurate methodologies. Most relevant to our work are the SimPoint [11] and SMARTS [31] sampling simulation methodologies. These are two recent developments and, similar to our work, both of the approaches rely on hybrid simulators that combine functional and cycle-accurate simulators in a single framework. Also, both approaches use statistical techniques to maintain CPI predictions during a simulation run.

By including both functional and cycle-accurate modes in the same framework the complexity of the simulator is greatly increased. This poses a large problem whereby ensuring the correct operating state of the micro-architecture prior to sampling is extremely difficult. These simulators are also outside of the well established processor design flows where software development and hardware design simulators are pre-
ferred to be kept separate. This is due to the requirements of fast, functional operation for software development and the slower but highly precise cycle-accurate simulators for hardware testing and synthesis with the usage of synthesisable RTL models [23, 1].

These sampling methods however do not employ any learning techniques however and only rely upon the small sampled section prior to a block for the cycle count of that block. This uniform sampling method ignores any possible state changes within a program allowing for errors to develop should an execution phase change occur between sampling instances. They employ no learning or an incremental model technique forcing them to create a completely new model per sampling session. Due to the uniform sampling this method is inflexible and does not easily fit into a JIT compiled simulation as hot blocks can be of varying length and interrupting a compiled simulation for a sampling period is a performance penalty.

2.3 Simulation Based Performance Prediction

Other approaches such as [17, 8, 9] typically run an entire program in an instruction-accurate simulator to construct a statistical profile which is then used for performance prediction. In [17] an artificial neural network has been trained to estimate cycle counts of a new program, whereas [8, 9] make use of linear regression prediction. These approaches both require separate simulations to generate training data for future simulations removing automation and continuous learning from the process. The latter approach achieves a good average accuracy, however statistical outliers occur relatively frequently. This is likely due to the coarse granularity of the prediction region, i.e. the whole program, whereas our work trains and predicts over much smaller code regions and, thus, has more scope to control the maximum error.

2.4 Statistical Simulation

Statistical simulation [7] is based on the simulation of small, synthetic program traces derived from statistical program profiles rather than the actual program itself. It can yield accurate IPC performance estimates several orders of magnitude faster than full simulation. Our work shares some commonality with statistical simulation in that it also relies on statistical profiles (e.g. various counters maintained by the simulator).

However, different to statistical simulation we do not construct a synthetic trace that is then fed into a cycle-accurate simulator, but compute the cycle count directly based on training data and statistical regression modelling. Hybrid analytical-statistical modelling [6] is mainly used for architecture and workload design space exploration and extends ‘classical’ statistical simulation with two parameters (α and β) characterising the age of register instances. Unfortunately, it is non-trivial to determine the specific value of the β parameter which is defined as the slope of the conditional dependence probability [of two subsequent operations] in a log-log diagram.
Chapter 3

Background

Relevant to our work, this section gives some background information on the theories and technologies used in this report. First information based upon the two simulation methods, functional and cycle-accurate, are described. Following this is a description of the statistical methods that are used or referenced in this report. These consist of Linear Regression, Locally Weighted Projection Regression, and Gaussian Process Regression.

3.1 Functional Simulation

Functional simulation is a very high-level form of Instruction Set Simulation, abstracted away from the hardware of a machine as far as possible. Their design is normally to mimic the exact operational output of a program, working at a purely instruction level. Often they are used for the purposes of ensuring the correct operation of a program during software development without the need for access to physical hardware. They can be used for the purposes of hardware verification however their high level of abstraction makes this difficult to do so beyond ensuring that a newly design instruction set will operate as expected.

Functional simulators can be extended to include a more detailed model of hardware being simulated, becoming semi-functional simulators. These extensions often exclude any timing models or performance calculation beyond the actual operational aspects of the hardware. Such extensions include a more detailed memory model to include cache simulation, or the branch prediction within a core, to more precise pipeline simulation.

From this type of simulation there are no performance calculations involved due to the lack of timing models. However, there is a quantity of high-level statistical information that can be easily collected by these simulators. For the most basic semi-functional simulators, only instruction counts for each distinct assembly instruction can be collected, including the total instructions executed. They can also provide basic memory statistics relating to the quantity of reads and writes in total, and for each specific block.

More complex semi-functional simulators that have been extended to contain cache simulation can provide counters related to the number of read hits and misses and write hits and misses within the cache. Dependant upon the type of cache being simulated
Chapter 3. Background

this could also include translation lookaside buffer hits and misses. Separate statistics can be collected for processors with independent data and instruction caches, or multilevel cache information. Furthermore, information based upon correct/incorrect branch prediction within a pipeline, and dependant upon the design of the branch prediction scheme, further statistics to include multilevel branch predictors with which level of prediction was correct.

Thanks to their higher level of abstraction away from the hardware being simulated, functional simulators are designed to operate much faster than their cycle-accurate counterparts. Basic semi-functional simulators generally operate at 1-2 orders of magnitude faster than RTL simulation, with more complex simulators employing compiled-code can operate at another factor of 10 over this [29]. Modern JIT-translating functional simulators have been shown to operate at 200-400 million instructions per second [5, 21, 26] It is due to these widely differing speeds of simulation that functional simulators are a very attractive option for software and hardware developers, however their high level of hardware abstraction make them difficult to use in a realistic situation, as many circumstances require detailed and accurate performance metrics to be able to determine if an update to a design has a positive influence beyond correct operation.

3.2 Cycle Accurate Simulation

Cycle accurate simulators, unlike their functional counterparts, include a highly precise timing model. Whilst the level of abstraction from hardware of functional simulators can vary wildly depending upon the simulators purpose, cycle accurate simulators must be tied very closely to the hardware that is being simulated. They include a full memory model including any caches within the processor and simulate the full pipeline of a system taking into account any possible causes of pipeline stalls during an execution.

The memory model implemented by cycle accurate simulators keep track of the state of every cache within the processor being simulated. Accesses to data is scrutinised, most importantly the effects of a hit or miss within a cache or the memory itself, keeping track of any stalls due to misses and the precise number of cycles taken for every memory access.

The full pipeline of the processor simulated is accounted for exactly. The timings of each module within the processor, for example the ALU, the floating point unit and the memory access module are all simulated allowing for access to each module as the simulated device would do so normally. This method models the stalls within the pipeline exactly, taking into account those possible at each separate stage within a pipeline. This could include a larger stall later in the pipeline overwriting the effects of a shorter stall in an earlier section, providing a very precise cycle count of a processor’s operation.

This finer degree of simulation allows for more statistics to be collected, namely the quantity and possibly the cause of stalls, and also a very precise overall cycle count for a simulations execution. From this information it is possible to calculate the IPC and CPI of a simulation and, combined with the simulation time, the effective clock speed of the simulator. Given the designed clock speed that the hardware will operate at it is also possible to determine exactly how long a program will take to execute
without any noise interference from an operating system installed on the device.

Understandably to simulate with this extra level of detail vastly increases the complexity of both the simulator and the internal implementation of the hardware being simulated. Not only does this makes it harder to implement re-targetable simulators, but even custom simulators for specific hardware due to the difficulties with ensuring the correct operation and timing models. This added complexity also greatly degrades the performance of a simulator, increasing simulation times by many orders of magnitude. Cycle-accurate simulators typically operate at 20,000 to 100,000 instructions per second.

Whilst it is highly desirable for hardware designers to have access to the metrics provided by cycle-accurate simulators, these lower speeds make it extremely difficult to perform all of the simulations required during hardware design. Their finer degree of simulation also complicates the implementation of JIT-compiled simulators. This is due to the complex timing accuracy and the modelling of stalls within the pipeline, with many JIT-compiled simulators only being able to provide a close approximation of the cycle count [26].

### 3.3 Statistical Regression

Regression analysis is a statistical method to examine the relationship between a dependent variable $y$ and $n$ independent variables $\mathbf{x} = (x_1, \ldots, x_N)$. This relationship is modelled as a function $f$ with $y = f(\mathbf{x})$. The function $f$ chosen depends considerably on the relationship between input vector $\mathbf{x}$ and output $y$. Many different forms of regressional analysis exist to compute $f$, including many variants of linear and non-linear relationships. Such regressional techniques investigated by this report include the standard linear regression technique, a more complex non-linear form called Locally Weighted Projection Regression [28] and also Gaussian process regression [30]. The choice of which regression function to choose is a design decision that greatly influences the accuracy and performance of any predictions made.

#### 3.3.1 Linear Regression

If the relationship between $y$ and $\mathbf{x}$ is linear we would choose a linear regression model like this:

$$ y = \beta_0 + \sum_{i=1}^{N} \beta_i x_i $$

(3.1)

where $\beta = (\beta_0, \ldots, \beta_N)$ is the weights matrix of the calculation.

Often the choice of independent input variable $\mathbf{x}$ are only a subset of the total factors affecting a variable, allowing equation 3.1 to only be an approximation of $y$, creating the predicted value $\hat{y}$. The difference between $y$ and $\hat{y}$ is the error of the prediction $\epsilon = y - \hat{y}$ is caused by an inadequate selection of input variables or potentially unrelated input variables, and should be minimised whenever possible. The error of a prediction is called the *residual*. 
Chapter 3. Background

Given \( m \) training points consisting of \((y_1, x_{1,1}, \ldots, x_{1,N})\) to \((y_m, x_{m,1}, \ldots, x_{m,N})\) we can extend our original design matrix (Equation 3.1) to consist of the equation system:

\[
\begin{align*}
y_1 &= \beta_0 + \beta_1 x_{1,1} + \beta_2 x_{1,2} + \cdots + \beta_N x_{1,N} + \epsilon_1 \\
& \vdots \\
y_m &= \beta_0 + \beta_1 x_{m,1} + \beta_2 x_{m,2} + \cdots + \beta_N x_{m,N} + \epsilon_m
\end{align*}
\]

or rewritten as \( y = X\beta + \epsilon \) with \( \beta \) as the vector of regression coefficients or the weights matrix and \( X \) as the model matrix.

The process of choosing the parameters of the model \( \beta \) must be performed in a manner to minimise \( \epsilon \) such that the regression function produces predictions as close to the actual value as possible. A common method of choosing these parameters is the least-squares method. It does this through minimising the sum of the squares of the prediction errors of the model on the observed data as such:

\[
SSE = \sum_{i=1}^{m} (\epsilon_i)^2, \quad \text{or} \quad SSE = \sum_{i=1}^{m} \left( y_i - \beta_0 - \sum_{j=1}^{N} \beta_j x_{i,j} \right)^2
\]

(3.2)

where \( SSE \) is minimised. The computed values represent estimates of the regression coefficients \( \beta \), thus the calculation of a prediction \( \hat{y} \) at point \( x \) only involves the application of equation 3.1.

From the theory of linear least squares, it is simple to calculate the weights matrix as follows:

\[
\beta = (X^T X)^{-1} X^T y,
\]

(3.3)

Unfortunately this is an expensive operation \( (O(m^3)) \) due to the calculation of the inverse of the variance matrix \( X^T X \). Dependant upon the size of the training data this can severely hinder performance, especially if the weights matrix must be updated frequently due to newly added training points. A more incremental method is preferred in this circumstance, however there exists research that has provided incremental linear learning methods, as described in chapter 4.

The calculation of the weights matrix can also be hindered easily when the variance matrix \( X^T X \) is non-invertible. This occurs when the variance matrix is a singular matrix, i.e. if and only if the matrix has a determinant of zero. This can occur fairly easily, for instance when training data contains a large quantity of columns with zero values among others. In this circumstance either new training data must be introduced, or the features selected for the input matrix \( X \) must be reduced.

### 3.3.2 Locally Weighted Projection Regression

A more complex algorithm for statistical regression is Locally Weighted Projection Regression (LWPR) [28]. This is a non-linear algorithm that employs a number of useful features such as local linear regression, automated feature selection and incremental learning. Given a training data set, LWPR identifies local regions that fit to linear models and splits the data sets accordingly, creating an individual linear model for each set and using the input data to identify which model to use prior to a prediction. For each local region the algorithm automatically identifies the most relevant
features to acquire the most accurate model and calculates the local linear function based upon only the data from those features.

LWPR uses the standard linear regression model for each computed local region or receptive field and calculates the predicted value \( \hat{y}_k \) using equation 3.1. The final prediction \( \hat{y} \) is then calculated using a series of weight functions as follows:

\[
\hat{y} = \frac{\sum_{k=1}^{K} w_k \hat{y}_k}{\sum_{k=1}^{K} w_k}
\]  

(3.4)

The weight function \( w_k \) for each receptive field is calculated from the distance metric of \( D_k \) using a Gaussian kernel:

\[
w_k = \exp \left( -\frac{1}{2} (x - c_k)^T D_k (x - c_k) \right)
\]  

(3.5)

where the vector \( c_k \) is the centre of the \( k \)'th receptive field.

LWPR has the advantages over the standard linear algorithm in that it employs an incremental learning method and automatic feature selection in a high dimensional space. Its prediction methodology is much more complex however, making each prediction much slower than that of the linear model. Due to the non-parametric regression techniques that are used it also requires larger quantities of training data, and hence longer training periods within the simulator. Furthermore, it is also less flexible due to the localised region selection methods, requiring that predictions are performed across similar data input lengths to the training data.

### 3.3.3 Gaussian Process Regression

Previous research into this area has also made use of Gaussian Process Regression [30], providing reasonable accuracy however with the same problem with statistical outliers. This technique employs a Bayesian non-parametric regression model. As with the other statistical methods, we apply a set of training data \( \{(x_i, y_i)\}_{i=1}^m \) to calculate a number of target outputs:

\[y_i = f(x_i) + \varepsilon_i\]  

(3.6)

where \( \varepsilon_i \) is considered to be independent Gaussian noise of variance \( \sigma^2 \). We also assume that the output values of \( f(x_i) \) across the training points \( \{x_i\}_{i=1}^m \) have a joint Gaussian distribution with zero mean and covariance matrix \( K_{i,j} = k(x_i, x_j) \) where \( k \) is the covariance function.

The output \( \hat{y} \) can be calculated using the Bayes Optimal estimator as a set of weightings of the covariance functions:

\[
\hat{y} = \sum_{i=1}^{m} w_i k(x, x_i)
\]  

(3.7)

with \( x \) the input vector, \( x_i \) the \( i \)'th training point. The weights vector \( w = (w_1, \ldots, w_m)^T \) can be calculated using the training output points \( y = (y_1, \ldots, y_m) \) by solving:

\[
(K^m + \sigma I)w = y
\]  

(3.8)
with \( \mathbf{I} \) the unit matrix. The covariance function \( k(x_i, x_j) \) used is the power-law model.

Unfortunately, Gaussian Process Regression would be unsuitable for a continuous learning simulation as it cannot be extended to include an incremental learning pattern. This implies that any update to the training data will require an expensive computation across the whole of the training data set.

### 3.3.4 Confidence Intervals

Regression based statistical models are often reliable methods for predicting an output when given an incomplete training set, or a data set containing features less relevant than others introducing noise into the data. Fortunately, they also have means of providing a description of the “quality” of these predictions, based upon how accurate the prediction is likely to be. From this we define the confidence interval of a prediction describing the upper and lower bounds on the estimated value. This value is based upon a desired confidence level \( \alpha \) chosen at design time. Providing a higher confidence required level will increase the confidence interval output.

In the case of linear regression, the confidence interval for a prediction can be calculated by:

\[
x \beta \pm \left( S \sqrt{x^T (X^T X)^{-1} x} t_{n-p; \frac{\alpha}{2}} \right)
\]

with \( x \) the input vector being estimated, \( \beta \) the weights matrix, \( S \) the estimated variance and \( t_{n-p; \frac{\alpha}{2}} \) the value at \( \frac{\alpha}{2} \) of the cumulative function for the t-distribution.

The Locally Weighted Projection Regression library used provides a similar confidence interval metric. An appropriate alternative also exists for Gaussian Process Regression, however given this method cannot be used in a continuous learning situation it is not listed here.

Appropriate alternatives exist for both Locally Weighted Projection Regression and Gaussian Process Regression.
Chapter 4
Methodology

4.1 Overview

In this chapter we introduce our novel continuous learning based performance prediction methodology and its integration into a hybrid instruction set simulator.

We assume that a hybrid functional and cycle-accurate simulator enhanced with various counters (including distinct assembly instruction counts, cache and memory accesses, and branch prediction statistics) are initially available. The simulator also has basic performance prediction capabilities based upon the available counters, with separate static training, feature selection and predicting across a fixed timeslice.

Our novel method attempts to combine the learning and prediction stages within the same simulation, allowing for automated training and simulation in the same framework. Our new method also monitors the confidence of a given prediction in an attempt to ensure that predictions being calculated are within a certain error margin, and further trains itself when necessary to keep error ratios low.

For us to implement a continuous learning based simulator a faster incremental linear regression model is required. Additionally an on-line feature selection method must be integrated. Finally, the simulator has to be modified to switch between functional and cycle-accurate mode when relevant based upon the calculated confidence and error.

4.2 Incremental Linear Regression

The standard linear regression algorithm has provided good accuracy in previous testing, however its calculation of the weights matrix is a computationally expensive operation involving the inversion of a potentially large matrix. This is to calculate the variance matrix for the training data. On occasions when updates to the training matrix are necessary, this calculation involves the recalculation of the weights matrix using the entire training data set. Alternatively, research has been performed into an incremental version to calculate the weights matrix [16].

The alternative linear regression algorithm still requires the initial calculation of the weights matrix as before, however optimises the process of adding new data points and updating the weights matrix once it has been initially calculated.
The process involves initially updating the variance matrix \((\mathbf{X}^T \mathbf{X})^{-1}\) of the data to include the new training points. The variance matrix will hereafter be referred to as \(\mathbf{A}^{-1}\). This is optimised in two ways dependant upon the number of training points added. One method for a single update to the matrix that does not involve the calculation of the inverse of a matrix, and another method for multiple training points added to the training set.

For a single added training point, the updated variance matrix \(\hat{\mathbf{A}}^{-1}\) is calculated as:

\[
\hat{\mathbf{A}}^{-1} = \mathbf{A}^{-1} - \frac{\mathbf{A}^{-1} \hat{\mathbf{X}} \hat{\mathbf{X}} \mathbf{A}^{-1}}{1 + \hat{\mathbf{X}} \mathbf{A}^{-1} \hat{\mathbf{X}}^T}
\]  

(4.1)

with \(\hat{\mathbf{X}}\) corresponding to the row of the newly added training point. Given \(\mathbf{A}^{-1}\) has already been calculated during the initial fitting of the weights matrix, there is no requirement for recalculation, allowing for further optimisation of this algorithm.

In the case of multiple samples being added to the training data, adding each training point individually using equation 4.1 would use additional computation than necessary, hence the alternative equation shown below can be used:

\[
\hat{\mathbf{A}}^{-1} = \mathbf{A}^{-1} - \mathbf{A}^{-1} \hat{\mathbf{X}}^T (\hat{\mathbf{X}} \mathbf{A}^{-1} \hat{\mathbf{X}}^T + 1)^{-1} \hat{\mathbf{X}} \mathbf{A}^{-1}
\]  

(4.2)

This method still calculates the inverse of a matrix to update the variance matrix, however the size of the matrix being inverted \((\hat{\mathbf{X}} \mathbf{A}^{-1} \hat{\mathbf{X}}^T + 1)^{-1}\) is much smaller and less computationally expensive than if it were to be performed across the entire data set.

Once the updated variance matrix has been calculated, the weights matrix for the regresional function is updated to \(\hat{\beta}\), as shown:

\[
\hat{\beta} = \beta - \hat{\mathbf{A}}^{-1} \hat{\mathbf{X}}^T (\hat{\mathbf{X}} \beta - \hat{y})
\]  

(4.3)

where \(\hat{y}\) represents the column vector of the output variable for the newly added training points. Future predictions use the updated \(\hat{\beta}\) as the weights matrix for equation 3.1.

### 4.3 Feature Selection

The purpose of feature selection is to determine a subset of the most relevant features of the training data. A number of possible feature selection methods are available to help choose which features affect the cycle count the most. For a continuous learning simulator it is necessary to have an on-line feature selector as the relevant features are likely to be different for each benchmark. This also helps to prevent the issue of a singular matrix during the calculation of the weights matrix in the Linear Regression Predictor.

Several possibilities are available. Some could include using the variance or the count of non-zero entries of each feature above a certain threshold. Another option would be to use a library such as the MRMR Toolset [18].

The calculation of relevant features is performed every time the training data is updated, potentially requiring the recalculation of the training matrix. If the list of relevant features does not change then the new training point can update the matrix
using the standard incremental linear regression, however a change in relevant features requires a complete recalculation of the training matrix. For this reason the threshold for deciding which features are relevant must be set such that feature selection changes are infrequent, whilst ensuring that all of the relevant features are kept.

### 4.3.1 MRMR Toolset

The MRMR Toolset is a set of algorithms designed to calculate the most relevant features that affect a dataset's outcome. They involve the calculation of the maximum dependency upon a feature set and its output, followed by a complex calculation of the maximum relevance and minimum redundancy of the feature set. Detailed operation of the method can be found in [18].

Through empirical testing prior to integration with the simulator it becomes obvious that use of the MRMR Toolset is complex and slow. This makes it unsuitable for an on-line feature selector as it would severely degrade performance during the learning process. It would also be complex to integrate with the simulator due to its usage of Matlab[14] packages.

### 4.3.2 Variance of a Feature

An alternative method is to use the variance of the data contained within a feature. For this method the data set must be iterated over to calculate the mean value for information in each feature. This is then followed by a second pass of the data to determine the variance of each feature, using a threshold to decide whether or not to include the feature.

The use of the variance as a feature selector also overly complicates the process and introduces further implementation obstacles. The added calculation of the mean of each feature degrades the performance by requiring several passes through the data on every update to the training data. The final output can also be swayed by one training point that can have an extremely large value whilst the rest of the points remain at zero making it hard to determine a sensible threshold for including a feature or not. This frequently caused singular matrices preventing the training matrix from being calculated.

### 4.3.3 Count of Non-Zero Entries of a Feature

Finally, a much simpler method involves calculating the percentage of non-zero entries within a feature. In this manner features that are used rarely are likely to have little influence on the overall output of the prediction and are hence discarded.

For these reasons the simplest method, the sum of non-zero values within a feature is chosen, allowing for an easily determined threshold to be used. Due to its simplicity, the calculation of the relevant features is much faster than the alternative methods, and the sums can be maintained between training updates removing the need for repeated passes through the data set.
Chapter 4. Methodology

4.4 Continuous Learning

Our novel method employs a continuous learning algorithm in an effort to ensure the accuracy of the predictions being made. By allowing the training data to be updated when confidence decreases, each prediction made will use the most recent and relevant training information. It also allows predictions to be performed over a much smaller timeslice ensuring that any errors that do occur only affect a smaller section of the code than previous work that suffered from statistical outliers due to a very large prediction block [8, 9]. The combined learning and prediction within the same simulation will also assist in customising the training data for the current execution, further reducing any error margins. The new continuous learning method will degrade performance slightly in comparison to the previous work, but this cost is outweighed by the prospect of much more accurate readings, and the full automation of learning and prediction within the same execution.

4.4.1 Operation

For the purposes of continuous learning several statistical metrics are monitored. The accuracy of predictions, and the confidence interval of a prediction. The accuracy of each prediction can only be monitored whilst the simulator is running in cycle accurate mode, and is referred to as a percentage above/below the actual cycle count for a timeslice. The confidence interval of a prediction is the range above/below the prediction made that has been witnessed previously, through performing predictions based across each training point.

Diagram 4.1 displays the operation of the simulator in continuous learning mode. As shown in the diagram the simulator initially begins in cycle accurate mode, with new training points being added to the predictor. Once enough points have been collected such that the initial training matrix can be calculated, the simulation remains in cycle accurate mode performing cycle count predictions once every timeslice. At each prediction the accuracy and confidence is checked. If the prediction is too inaccurate, or the confidence interval of the prediction is too high then the last timeslice is added.

Figure 4.1: The Sequential Operation of the Simulator in Continuous Learning Mode
to the training data and the prediction matrix is updated.

Once the error and confidence interval drop below a pre-programmed threshold the simulation switches to functional mode and relies purely upon the confidence interval of each prediction. This is monitored at every prediction, and should it become too high the simulator will fall back into cycle-accurate mode in one of two methods described in the next section. The simulator continues to run in cycle-accurate mode updating the training data once per timeslice and attempting further predictions. Again, once the error ratio and confidence of the prediction drop below their thresholds the simulator returns to functional mode and continues as above.

Once a prespecified number of confident predictions occur, the timeslice over which the predictions are made is increased in an attempt to gain a further performance gain. Once the confidence interval widens too far the timeslice is reduced again to allow training across a finer grain. This is described further in section 4.4.3.

4.4.2 Dealing With A Less Confident Prediction

When a less confident prediction occurs that is below the threshold the situation can be handled by two distinct methods. A fast patch scheme or a conservative rollback and replay scheme.

**Patch** The simulator switches back to cycle accurate mode and updates the training matrix based upon the next few timeslices.

**Rollback-Replay** The simulation is rolled back to a previously known state, i.e. just before the last timeslice began, and re-runs the simulation in cycle accurate mode, updating the training matrix as necessary.

The first method simply uses the result of the less confident predictions as normal followed by a patch of the training data for future predictions, whereby the second method does not rely upon the less confident prediction, and ensures accuracy for that timeslice. Theoretically the second method should provide more accurate results whilst degrading performance by having to roll back the simulator. This performance degrade may prove to be more expensive than the lack of accuracy of the first method.

4.4.3 Altering the Timeslice Length

To further aid the performance of the simulator, it is possible to make predictions across differing lengths of timeslice. This is thanks to the calculation of the prediction being a simple matrix multiplication, which is distributive:

$$(A \times C) + (B \times C) = (A + B) \times C$$

Using this fact allows the simulator to be trained at a short timeslice, and then predictions to be made at a higher timeslice. To exploit this feature, the simulator initially trains itself at a short timeslice, and then performs a number of predictions at the same length of timeslice. Should all of those predictions stay within the confidence level, then the timeslice can be increased by a factor of ten, making predictions much less frequent and allowing for a greater performance benefit. Again predictions are made across the same number of timeslices with the confidence interval monitored, allowing
for the timeslice to be further increased in a regular manner up to a higher limit, provided predictions remain accurate and confident enough. Should predictions exceed the maximum confidence interval then the timeslice is decreased by a factor of ten (down to a lower limit), and retraining commences.

4.4.4 Transference of Training Data

Another option would be to store the training model between simulations, as with the distinct static training and simulation methods that the simulator featured originally. This would allow the simulator to start making predictions from the start of execution removing the training period, however this could cause further inaccuracies and create inflexibility between differing benchmarks. This is generally only relevant for shorter simulations that execute faster than the training period, therefore removing any performance benefit of the learning simulator.
Chapter 5

Implementation

This chapter gives a brief overview of the structure of the simulator being extended, the prediction libraries being used, and their integration with each other.

5.1 Simulator Overview

The internals of the simulator consist of a system module, overviewing a number of processors (in our case just a single processor however this can be extended). The system model maintains any global information with regards to simulator settings, information collected on the machine that is executing the simulator. The system also stores the main memory model and handles any references to it.

The system module handles any interactions between the user and the processor, passing any relevant interrupts through between timeslices. It also models any system calls that may occur during simulation. The system process has the power to start, halt, pause and restart any simulation occurring due to either user interaction, the use of breakpoints or errors during simulation.

The system module controls the processor through a run call allowing it to run for a specified number of instructions. Depending on the mode of the processor this quantity may be slightly over by allowing a JIT-compiled block to finish execution, or under if the end of the simulation has been reached, however this is a good approximation of how many instructions will be executed before the system module regains control. The quantity of instructions a processor is allowed to execute is called the *timeslice*.

The processor module maintains the exact state of its operation, including the general purpose registers, program counter, ALU result registers and other simulation parameters required. It contains its private caches, both data and instruction, and contains a reference to the system module to allow access to the main memory.

The processor module has three main modes of operation, interpreted, traced, and JIT-compiled.

**Interpreted**  The interpreted mode takes each instruction individually and decodes the binary code compiled for the hardware being simulated. The decoded instruction is then used to update the state of the processor and any attached devices i.e. the caches and memory.
Traced  The traced mode functions just as interpreted except it outputs every operation to the screen/output file.

JIT-compiled  The JIT-compiled mode executes blocks of code at a time. A block of code is effectively a section of code between branch statements, guaranteeing that every instruction within a block will be executed sequentially, however they do have the ability to raise exceptions. With JIT mode, blocks are first executed in interpreted mode until the quantity of times they have executed exceed a threshold, at that point they become a hot block. At regular intervals, a check is made for new hot blocks, which are then translated using the JIT-compiler into native code to modify the state of the processor without having to decode each instruction individually. Every time a hot block is now executed, this newly compiled library is used instead, increasing the speed of execution dramatically.

5.2  Prediction Library Overview

The prediction library used implements several different prediction methodologies whilst using the same library. This allows for the simulator to be extended in simple ways to attempt a new prediction method. There is a base predictor that collects and stores any training data into an unordered map that disallows duplicate entries. The base predictor implements a simple hash look-up into this map on every prediction to determine if there has been an exact training point that represents the prediction. The base predictor also keeps track of the statistics for feature selection, including the methods for calculation.

The other prediction methodologies are then built on top of this module to ensure consistent functionality. The two methodologies investigated in this report are Incremental Linear Regression and Locally Weighted Projection Regression, therefore it is these two that are concentrated on in this section.

5.2.1  Incremental Linear Regression

The incremental linear regression module maintains a number of matrices for the input training points, a variance matrix and a conversion matrix for converting a set of input points to only the relevant selected features. The module also maintains two vectors containing the training data outputs and the calculated weights for use during prediction, and finally a list of the new observations added to the training data since the last update of the weights vector.

When new training points are added, if they are successfully added to the base predictor then they are added to the list awaiting recalculation of the weights vector. This recalculation is not performed until a prediction is attempted as this would cause extra computation if every point updated the vector individually. On the first attempt of a prediction the training matrix is calculated using the non-incremental algorithm, and future updates to the training data utilise the incremental version. Between training updates, the weights vector is left unchanged and predictions are performed through a simple vector multiplication. Any predictions are returned containing their confidence interval.
5.2.2 Locally Weighted Projection Regression

The locally weight projection regression module comes with its own library compiled and ready for static linking. On the addition of a training point, again the point is passed through to the base predictor to check for duplicates. If the addition is successful the point is passed to the LWPR library to update the model immediately. This is distinct from the linear regression model due to the entirely incremental nature of the LWPR library. When a prediction is performed the point is again passed to the LWPR library and the output and confidence interval are returned.

5.3 Integration

Initially the system allows the processor to execute a specific number of instructions, the initial timeslice, in cycle accurate mode then collects the statistics from the last timeslice of the processor into an array and passes this to a linked library containing the predictor. The prediction library adds the new data to its training set and attempts a prediction, reporting the outcome of this prediction back to the system. If the prediction is successful the system determines if it is within correct error margins and confidence levels, and if so the system then instructs the processor to run in functional mode and allows it to run further timeslices, otherwise training continues as before.

Again, statistics from each timeslice are collected and passed through an array to the prediction library, however this time only high level information is available to be passed. The predictor library attempts another prediction and passes this information back to the system where its confidence level is checked again. Too high a confidence interval and the processor is returned to cycle accurate mode for further training, otherwise simulation continues as normal.
Chapter 6
Empirical Evaluation

6.1 Setup
This section describes the methodology for testing our novel new performance prediction scheme. First is a description of the hardware used, including information on the distributed network of computers used and their system parameters. This is followed by a description of the setup of the simulator itself, and a description of the hardware being simulated. Next is a description of the range of benchmarks being executed on the simulator followed by the methodology for varying the parameters of the new prediction scheme in an effort to find the optimal setup.

6.1.1 System
To analyse the benefits our novel method brings, a number of simulations were performed across a range of benchmarks. Simulations were executed to determine both the accuracy of the predictor, and also any reduction in simulation time. Due to the large range in execution times of simulations, and the large quantity of simulations that were required, the simulations were performed distributed across approximately 80 computers, with one benchmark per computer at a time. Table 6.1 lists the system parameters used to run the simulations.

<table>
<thead>
<tr>
<th>System Parameters</th>
<th>Intel Core 2 Quad Q9300 @ 2.50GHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td></td>
</tr>
<tr>
<td>L1 Cache</td>
<td>128 KB</td>
</tr>
<tr>
<td>L2 Cache</td>
<td>6 MB</td>
</tr>
<tr>
<td>Main Memory</td>
<td>8 GB Shared 667 MHz</td>
</tr>
<tr>
<td>Operating System</td>
<td>Scientific Linux 5.2 64-bit (Boron) (2.6.18 kernel)</td>
</tr>
</tbody>
</table>

Table 6.1: System parameters for simulation
### Simulator Configuration

The simulator being extended in this project is a state-of-the-art hybrid function/cycle accurate instruction set simulator, for the ARC 750D embedded processor [26]. It has already been extended to include basic performance prediction functionality using distinct training and prediction simulations. The prediction methods implemented include K-Nearest Neighbour, Linear Regression, Locally Weighted Projection Regression [28] and Gaussian Process Regression [30]. Table 6.2 details the processor being simulated, and the configuration options of the simulator.

Table 6.2: Overview of simulation platform

<table>
<thead>
<tr>
<th>Test</th>
<th>ARC 750D</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Core</strong></td>
<td></td>
</tr>
<tr>
<td>Pipeline</td>
<td>Single, 7-stage (interlocked)</td>
</tr>
<tr>
<td>Execution Order</td>
<td>In-Order</td>
</tr>
<tr>
<td>Branch Prediction</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Memory System</strong></td>
<td></td>
</tr>
<tr>
<td>L1 Cache</td>
<td>8k (Instruction) / 8k (data)</td>
</tr>
<tr>
<td>L2 Cache</td>
<td>None</td>
</tr>
<tr>
<td>MMU</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Simulation</strong></td>
<td></td>
</tr>
<tr>
<td>Simulator</td>
<td>ARC simulator</td>
</tr>
<tr>
<td>Options</td>
<td>Default</td>
</tr>
<tr>
<td>I/O &amp; System Calls</td>
<td>Emulated</td>
</tr>
<tr>
<td><strong>SW Development Tools</strong></td>
<td></td>
</tr>
<tr>
<td>Compiler</td>
<td>GCC 4.2.1</td>
</tr>
<tr>
<td>Compiler Options</td>
<td>-O3 -m32</td>
</tr>
</tbody>
</table>

Previous work [9] has demonstrated that the Linear Regression and Gaussian Process prediction schemes provide good accuracy on average, however Linear Regression is much more flexible as an incremental method to add new data points exists and has a low cost operation to perform predictions. Gaussian Process regression is unsuitable for use in a continuous learning environment as it does not have an incremental learning method. Locally weighted projection regression is as flexible thanks to its incremental nature and claims to be accurate in high dimensional spaces, however it’s accuracy and performance have not yet been tested for this purpose. Locally weighted projection regression also has a more complex prediction technique and requires a larger training data set which could impair performance negating any benefits of our new methodology. For these reasons this report will concentrate mostly on the operation of the Linear Regression prediction scheme. A number of simulations will be performed using locally weighted projection regression as there has been little testing using this method in prior research. The results of prior testing using the Gaussian Process regression scheme will be referred to as an example. Table 6.3 lists the counters maintained by the simulator and provided to the predictor for training and prediction.
### 6.1 Setup

#### Counters

<table>
<thead>
<tr>
<th>Counters</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1 ... x148</td>
<td>Instruction counters for BCC, BR, BRCC, JCC, JCC1, JCC2, LDCC, LDWORD, LD_HALF32, LD_HALF2, LD_HALF1, LD_BYTE, ST_BYTE, ST_HALF, ST_WORD, LR, SR, TST, BTST, CMP, RMP, MOV, ADD, SUB, AND, OR, MOV1, ADD1, SUB1, AND1, OR1, SUB2, AND2, OR2, MOV3, ADD3, SUB3, AND3, OR3, MOV4, ADD4, SUB4, AND4, OR4, MOV5, ADD5, SUB5, AND5, OR5, MOV6, ADD6, SUB6, AND6, OR6, MOV7, ADD7, SUB7, AND7, OR7, MOV8, ADD8, SUB8, AND8, OR8, MOV9, ADD9, SUB9, AND9, OR9, MOV10, ADD10, SUB10, AND10, OR10, MOV11, ADD11, SUB11, AND11, OR11, MOV12, ADD12, SUB12, AND12, OR12, MOV13, ADD13, SUB13, AND13, OR13, MOV14, ADD14, SUB14, AND14, OR14</td>
</tr>
<tr>
<td>x149</td>
<td>Total instructions</td>
</tr>
<tr>
<td>x150, x151</td>
<td>Total I-Cache read hits, read misses</td>
</tr>
<tr>
<td>x152, x153, x154, x155, x156</td>
<td>Total D-Cache write hits, write misses, read hits, read misses, dirty misses</td>
</tr>
<tr>
<td>x157, x158</td>
<td>Total branch prediction hits, misses</td>
</tr>
</tbody>
</table>

Table 6.3: Overview of counters maintained by the instruction set simulator

### 6.1.3 Benchmarks

To gain the best evaluation of the simulators performance, simulations must be performed across a wide variety of benchmarks, covering many common uses. These include a number of embedded applications, multimedia processing applications, digital signal processing applications and kernel simulation benchmarks. Table 6.4 lists the benchmark suites used, and their descriptions. The suites total 318 separate benchmarks. Prior work required this large number of simulations to provide sufficient training data for the predictor [9], however this methodology performs training and prediction at a much finer degree. The large range of simulations is kept however to demonstrate the functionality across a wide range of applications, including simple to extremely complex and most importantly ranging in execution time.

#### Benchmark Suite | Description
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>DSPstone [27]</td>
<td>Small DSP kernels</td>
</tr>
<tr>
<td>UTDSP [3]</td>
<td>Small DSP kernels and applications</td>
</tr>
<tr>
<td>SWEET WCET [10]</td>
<td>Worst case execution time benchmarks</td>
</tr>
<tr>
<td>Mediabench [4, 13]</td>
<td>Multimedia applications</td>
</tr>
<tr>
<td>EEMBC [25]</td>
<td>Automotive, Consumer, Digital Entertainment, Networking, Office Automation &amp; Telecoms Applications</td>
</tr>
<tr>
<td>Pointer-Intensive [24, 2]</td>
<td>Non-trivial pointer-intensive applications</td>
</tr>
<tr>
<td>Benchmarks</td>
<td>StreamIt Benchmarks [22]</td>
</tr>
</tbody>
</table>

Table 6.4: Overview of benchmark suites used to test simulator performance

### 6.1.3 Benchmarks
6.1.4 Experimental Methodology

In our experiments simulations were performed in sequence, with the initial baseline cycle-accurate simulation, followed by our new simulation method. Two parameters were modified inside the simulator and each benchmark was re-simulated across all combinations of these parameters. These were the initial length of the timeslice, and the number of predictions performed at that timeslice prior to increasing its length. These were modified between 100 to 100,000, and 10 to 10,000 respectively, increasing by a factor of 10. The simulations were performed in both patch and rollback-replay modes, totalling 32 separate simulations of the entire benchmark suites.

The LWPR prediction scheme is also tested separately as an investigation into the relevance of a more complex prediction scheme. This is ran at a timeslice length of 1000 instructions across all of the benchmarks to determine both its accuracy and performance.

To ensure the most accurate reading for the execution time difference between standard cycle-accurate simulation and our new simulation method, each simulation was performed with the baseline simulation immediately followed by the new method. This method involves rerunning the baseline simulation in its entirety, however in doing so minimises any noise from differences between the software stack of each computer.

6.2 Results

In this section we discuss our results. Starting with a general overview of the results, we then go on to discuss the effects of modifying the timeslice length and the number of predictions made confidently at each timeslice for the incremental Linear Regression prediction scheme. Next we discuss the characteristics of simulation in both continue and rollback modes. We then examine the performance of the Locally Weighted Projection Regression scheme, followed by a comparison of the results with of prior work in the area [9]. Beyond the statistical information collected for each individual simulation, we calculate several metrics to determine the approximate accuracy of the new methodology. These are the mean absolute error (MAE) and the maximum error of the total simulations at each configuration. These values are calculated only from simulations where predictions were possible, as there will be zero error in those that could not.

6.2.1 Overview

For the standard Linear Regression prediction scheme the simulations displayed performance improvements of up to 1.5 times speedup with a MAE of 2.36% and a maximum error of at best 19.97%. Simulations proved that a shorter original timeslice is effective at allowing shorter benchmarks to receive the performance benefits without adversely affecting longer benchmarks. They also proved that having a variable prediction timeslice posed additional performance benefits without severely affecting error margins, however this cannot increase too quickly, as this prevents the predictor from receiving adequate training data and as such negates any performance benefits. There
is also little to no benefit in rerunning unconfident prediction blocks in cycle accurate mode to ensure accuracy. Full MAE and maximum errors are listed in Appendix A.

Performing simulations using the Locally Weighted Projection regression scheme proved to require a much greater learning period. Predictions were unable to be calculated on simulations shorter than approximately $10^8$ cycles, providing much larger error ratios and unpredictable performance benefit.

The simulator gains adequate performance benefits compared to the prior work in the area, with comparable MAE, and a significantly better maximum error, 52.8% better[9].
6.2.2 Altering the Timeslice Length

With this new method, the training period is the initial bottleneck of a simulation. For shorter benchmarks it can hinder performance, and easily render the new simulation scheme’s benefits useless. By altering the initial timeslice length, we are shortening this training period by allowing the training matrix to be filled with finer grained data and then predictions to be performed across larger blocks. Initially we examine the performance benefit of this method before examining it’s accuracy. The effects of doing so are most noticeable at 100 confident predictions per timeslice increase, with the percentage of useful predictions displayed in Figure 6.1 and the relative speedup shown in Figure 6.2.

From the graphs displaying useful predictions, it is clear that shortening the original timeslice has the desired effect of allowing predictions to begin very early in the simulation process. However, it is also noticeable that there is no performance benefit
from doing so with simulations below $10^6$ cycles.

The results also show that with an original timeslice of 10,000 instructions and above certain simulations start performing worse than their cycle accurate counterparts. Further inspection of these points implies that the predictions being made by the simulator are either outside the error margin or have too low a confidence. This can be solved by altering the thresholds, however this would increase error ratios, whilst still causing performance degrade for simulations below $10^7$ cycles.

From this an original timeslice of 100-1000 instructions provides the best performance benefit. In general the higher this value, the less performance benefit that is apparent. At this level it is clear that the performance of the simulator begins increasing beyond the baseline for simulations above $10^6$ cycles, plateauing at just below a speedup of 1.5 times.

Figure 6.3(i) displays the predicted cycle counts versus the actual cycle count whilst modifying the original timeslice length, with Figure 6.3(ii) displays the distribution of the error ratio. The additional lines in Figure 6.3(ii) show $\pm \sigma$, $\pm 2\sigma$ and $\pm 3\sigma$ intervals representing the expected percentages (68%, 95% and 99.7% respectively) of the quantities between these lines.

The error distribution shows that the majority of errors, lie below 5% in all configurations. Increasing the original timeslice decreases the average error further, with the MAE for a starting timeslice of 10,000 instructions of 0.66%. However due to the performance issues introduced with such a high starting timeslice, the larger MAE and max errors of a starting timeslice of 10-100 instructions is preferable. At a 1,000 instruction timeslice the MAE is still only 2.36% with a maximum error of 19.97%.
Chapter 6. Empirical Evaluation

Figure 6.3: Modifying Original Timeslice Length

(a) 100 instructions

(b) 1,000 instructions

(c) 10,000 instructions

(i) Predicted vs. Actual Cycle Count  (ii) Error Ratio Distribution
Altering the number of confident predictions has a dramatic effect on the performance of the simulator. Figure 6.4 displays the amount of accurate and confident predictions whilst varying the quantity before increasing the timeslice length, with the speedup shown in 6.5.

Altering the number of confident predictions has a dramatic effect on the performance of the simulator. By having too low a quantity, e.g. just 10 before increasing the timeslice, the predictor is not trained accurately enough. Extra training points that would normally be added when the predictor loses confidence are missed causing a low number of valid predictions, rendering the new method useless for shorter simulations ($< 10^7$ cycles).

By increasing this value to 100, the simulator starts receiving a performance increase from $10^6$ cycles, plateauing at just below 1.5 times speedup for longer simula-
tions. Beyond 100 predictions however, it is detrimental to the performance for smaller benchmarks with no overall performance gain for longer simulations.

Figure 6.6(i) displays the predicted versus actual cycle count, with Figure 6.6(ii) displaying the distribution of errors whilst modifying the number of confident predictions. A value of 100 confident predictions provides a MAE of 2.36 % and a low maximum error of 19.97 %. Increasing this value further has very little effect on the prediction errors, whilst hindering performance. Hence a value of 100 confident predictions before increasing the timeslice is optimal.
6.2. Results

(i) Predicted vs. Actual Cycle Count

(ii) Error Ratio Distribution

Figure 6.6: Modifying Number of Predictions
6.2.4 Rollback-Replay vs. Patch

By re-simulating a less confidently predicted block in cycle-accurate mode, accuracy of the predictor is intended to be increased. However, simulations proved that by re-simulating less confident blocks had little effect on the error margins of the predictor, whilst only creating slightly worse performance for the majority of simulations.

Because of this, the rollback-replay mode of the simulator is an unnecessary addition that overly complicates the program, making it much harder to ensure correct operation of a simulation. Figure 6.7 compares the speedup and quantity of useful predictions inside the simulator for an original timeslice of 1000 instructions, and 100 confident predictions, whilst Figure 6.8 compares the predicted versus actual cycle count and the error distribution of the two modes.
6.2. Results

6.2.5 Performance of Locally Weighted Projection Regression

The performance of the Locally Weighted Projection Regression prediction scheme was inadequate compared to that of the standard Linear Regression scheme. Due to the nature of the scheme’s functionality it required a greatly extended learning period, restricting the ability to perform any predictions below executions of approximately $10^8$ cycles. During this training period performance is severely degraded performing worse than the baseline cycle-accurate simulation. Above this level the simulations performed did gain some performance benefit, up to just below 1.5 times speedup as with Linear Regression, however this was only in certain precise circumstances. More frequently than not the simulations performed much worse than that of the baseline, with any performance benefits were extremely sporadic.

With respect to the accuracy of the scheme, the simulation proved that the error margins were much worse than that of the standard Linear Regression scheme, with a MAE of 26.74% and a maximum error of 97.62%. Interestingly, the locally weighted
Previous work [9] has concentrated on training the simulator with prior simulation performed across the entire execution of a program. This training data has then been fed back into the simulator in a Leave-One-Out Cross-Validation method, where all training points except that of the current benchmark being performed is used to calculate the training matrix. A single prediction is then performed at the end of each benchmark using several statistical methods, most importantly Linear Regression and Gaussian Process Regression. Gaussian Process Regression is unsuitable in our simulations as a projection regression scheme always predicted lower than the actual value. Figure 6.9 displays the quantity of useful predictions, the relative speedup and the error margins for the locally weighted projection regression scheme.

### 6.2.6 Comparison with Previous Tests

Previous work [9] has concentrated on training the simulator with prior simulation performed across the entire execution of a program. This training data has then been fed back into the simulator in a Leave-One-Out Cross-Validation method, where all training points except that of the current benchmark being performed is used to calculate the training matrix. A single prediction is then performed at the end of each benchmark using several statistical methods, most importantly Linear Regression and Gaussian Process Regression. Gaussian Process Regression is unsuitable in our simulations as a...
6.2. Results

A single update to the training matrix requires a complete recalculation across the entire training data set, however its results in this circumstance are useful as a comparison of accuracy.

Our new methodology is a vast improvement in accuracy over the previous method mainly thanks to its lack of statistical outliers. This is because of the much finer timeslice across which predictions are performed, and the ability to monitor the confidence of the predictions to ensure they remain within a reasonable error margin.

Our new methodology is slower at performing predictions than the previous work due to the higher quantity of predictions being made, however this is outweighed by the fact that our new method does not require a large amount of cycle-accurate simulations to be performed to acquire the necessary training data. Our new method also improves on the previous work by automating the process of training and prediction into one simple framework, allowing for fast and flexible simulation, enabling a change made to a design to be quickly evaluated. This also removes the need for separate feature selection using slow and complicated methods [18].

Our simulations provided a MAE comparable to that of the Linear Regression model of 2.10%, however thanks to the decreased quantity of statistical outliers there is a 59.5% improvement of the maximum error of 49.4%. With respect to the Gaussian Process Regression, our method provided a 47.3% MAE improvement over 4.47% whilst simulating the same hardware, and a 46.4% MAE improvement over the best performance of 4.40% whilst simulating alternative hardware designs [12]. It also improved the maximum error by 78.1% for the same hardware, and by 62.1% for alternative hardware, with previous values of 91.2% and 52.8% respectively.

It is not possible to quantify the difference in performance of these two methods, however the added simplicity in simulation, and greater accuracy make this a much better candidate for future simulator designs, where complete accuracy is not necessary.
Chapter 7

Summary & Conclusions

7.1 Conclusions

In this report we have developed a new performance prediction methodology suitable for use in a hybrid functional/cycle-accurate instruction set simulator. We have successfully integrated this new methodology and tested it proving that it is suitable for use when both high performance and accuracy are important to the user.

The new methodology is fully automated, enabling both training and prediction in the same simulation, whilst ensuring that the error margins are kept within reason. This allows a hardware designer to make flexible incremental improvements to a design and rapidly find the performance difference of doing so without the need for many repeated simulations involving a separate training session.

We have investigated the usage of incremental regression techniques showing that incremental Linear Regression remains the most accurate and flexible for simulation in these circumstances. We have also investigated the suitability of Locally Weighted Projection Regression as a possible alternative showing that it is too inaccurate whilst providing no stable performance benefit. Linear Regression has been shown to provide good accuracy with a possible MAE of 2.36% and a maximum error of 19.97%.

We have also shown that by using a variable timeslice for training and prediction allows for the benefits of this methodology to become apparent for shorter simulations, for executions from as low as $10^6$ cycles, providing a speedup of up to just below 1.5 times the baseline cycle-accurate performance. It has also been shown that the timeslice cannot increase too quickly to take account of the variability that can occur across a simulations execution that may not become apparent in the first set of timeslices.

The two possible solutions when a less confident prediction occurs have been investigated showing that a patch scheme is slightly faster and generally provides the same accuracy of that of a rollback-replay scheme due to the fine grain at which predictions are performed. Various on-line feature selection methodologies have also been examined showing that for the purposes of high performance simulation a simple threshold of non-zero entries in training data works adequately, without hindering performance.

This new method improves on previous research in the area, providing a comparable MAE whilst dramatically reducing the maximum error by 52.8%.
Chapter 7. Summary & Conclusions

7.2 Future Work

This work has ignored any JIT-functionality within the simulator to provide a more stable baseline estimate, and an easier method for quantifying the benefits of the new methodology. Further research could be investigated into the performance benefits of using this methodology in a JIT-compiled simulation to allow for very high performance simulation of longer benchmarks including media applications and possibly the simulation of a full operating system.

Due to the higher level of abstraction away from the hardware that is provided during JIT-compiled simulations, a method to maintain the accuracy of this methodology could be to run the cycle-accurate training sections in fully interpretive mode, ensuring a correct cycle count, whilst running predictions in JIT-compiled mode, executing as quickly as possible.

Further work could be investigated into the calculation of thresholds for the maximum allowed error during training and the maximum confidence intervals allowed. This could possibly be performed using statistical methods such as the variance of the intervals between each prediction as a method to determine phase changes during an execution.

In an effort to allow for performance benefit in even shorter simulations, further research could be explored into the transference of training data between simulations. This would provide a valid prediction model at the start of the simulations allowing predictions to be performed immediately. This would remove the need for a separate learning period at the start of the simulation and allow the simulation to rely purely on the confidence interval of the data. This method could widen error margins, however would also provide a useful speed-up for shorter simulations.

On the other hand this work also makes no attempt at ageing the training data throughout a simulation which could introduce larger error margins. Ageing training data involves the removal of the oldest training data points when adding new points to the training set, or the use of weightings to each data points, forcing the oldest data to have the least influence on the predictions made. Further work could be performed looking at ageing the training data when updated, allowing for more emphasis to be made on newly added training data over the initial training period. This could allow for greater phase detection during a simulation, attempting to reduce error margins.

The simulations performed were only performed on a single processor simulator. Future work could include extending this methodology across multiple processors. This would include the use of separate statistics for each processor therefore a new methodology for calculating the predictions would need to be developed. This could provide a separate performance estimate for each individual processor, or one global performance estimate based upon the maximum prediction.
# Appendix A

## Error Margin Results

### Table A.1: Mean Absolute Error (%)

<table>
<thead>
<tr>
<th></th>
<th>Original Timeslice</th>
<th>Confident Predictions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>10</td>
</tr>
<tr>
<td><strong>Patch</strong></td>
<td>100</td>
<td>1.29</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>0.96</td>
</tr>
<tr>
<td></td>
<td>10000</td>
<td>0.74</td>
</tr>
<tr>
<td></td>
<td>100000</td>
<td>0.44</td>
</tr>
<tr>
<td><strong>Rollback-Replay</strong></td>
<td>100</td>
<td>1.31</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>1.38</td>
</tr>
<tr>
<td></td>
<td>10000</td>
<td>1.45</td>
</tr>
<tr>
<td></td>
<td>100000</td>
<td>0.95</td>
</tr>
</tbody>
</table>

### Table A.2: Maximum Error (%)

<table>
<thead>
<tr>
<th></th>
<th>Original Timeslice</th>
<th>Confident Predictions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>10</td>
</tr>
<tr>
<td><strong>Patch</strong></td>
<td>100</td>
<td>24.39</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>14.6</td>
</tr>
<tr>
<td></td>
<td>10000</td>
<td>19.12</td>
</tr>
<tr>
<td><strong>Rollback-Replay</strong></td>
<td>100</td>
<td>15.35</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>14.61</td>
</tr>
<tr>
<td></td>
<td>10000</td>
<td>19.12</td>
</tr>
<tr>
<td></td>
<td>100000</td>
<td>11.26</td>
</tr>
</tbody>
</table>
Bibliography


