Deploying ML Models for Faster Micro-Architectural Simulation

M.Tech Major Project Report Submitted
in Partial Fulfillment of the Requirements
for the Degree of

M. TECH. System-on-Chip Design (SOCD)

by

PARISA SURYA TEJA
(152002014)

under the guidance of

Internal Guide
Dr. SANDEEP CHANDRAN
Assistant Professor
IIT Palakkad

External Guide
Dr. VENKATARAMANI VANCHINATHAN
Sr. Silicon Design Engineer
AMD Bangalore

IIT PALAKKAD

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY PALAKKAD

CERTIFICATE

*This is to certify that the work contained in this project entitled "**Deploying ML Models for Faster Micro-Architectural Simulation**" is a bonafide work of **PARISA SURYA TEJA** (**Roll No. 152002014**), carried out in the Department of Computer Science and Engineering, Indian Institute of Technology Palakkad under my supervision and that it has not been submitted elsewhere for evaluation.*

Dr. SANDEEP CHANDRAN

Assistant Professor

Department of Computer Science & Engineering

Indian Institute of Technology Palakkad

Acknowledgements

I would like to express my sincere gratitude to Dr. Venkataramani Vanchinathan, Sr. Silicon Design Engineer, Advanced Micro Devices, Bangalore for providing impeccable guidance in the work.

I would like to express my sincere thanks to Dr. Sandeep Chandran, Assistant Professor, Department Of Computer Science, for crucial support and the learning's learned from the previous project were immensely helpful in the current project.

I would like to thank Mr. Shivam Potdar, Silicon Design Engineer 1, Advanced Micro Devices, Bangalore for providing the required support in the project whenever necessary.

I would like to thank Mrs. Anasua Bhowmik, Fellow Silicon Design Engineer, Advanced Micro Devices, Bangalore for managing and guiding entire team.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this modern world, there is so much need for high performance computing to accomplish human endeavours. The tremendous growth in the information technology field in recent years has impacted enormously in many fields of society for various purposes such as space exploration, weather forecasting, new molecular compounds research, quantum mechanics, etc. Increasing the system performance decreases the time to accomplish a task which thereby enhances productivity.

The performance can be achieved with the systems which possess following things faster operational frequency processing units, fewer instruction counts for operations, more instructions per cycle, more cache, ram sizes and also it can be achieved by the execution of the tasks in more parallelism and concurrent way. Parallelism can be achieved through threadlevel parallelism, instruction-level parallelism and data-level parallelism with the use of CPUs, GPUs and APUs.

Present computing systems possess multicore CPUs that offer parallelism with different methods that have their own advantages and associated challenges. One of the methods is to greatly improve the turnaround time of specific applications by running the different parts of it in parallel. Which requires careful attention by the programmers to follow the

parallel programming constructs such as coordinating and synchronising different parts of the program execution.

This synchronisation of programs among the cooperative processors is overhead in achieving maximum performance from each individual processor core[1]. The second method involves scheduling more than one application at the same time to run in different cores, which improves the responsiveness of the system substantially by increasing the use of networking, input and output operations, storage and other subsystems. The third method uses the fact that the degree of synchronisation and communication needed by the programs which execute concurrently is much less than the applications which execute in parallel.

This takes advantage of the more CPU cores in a system to improve the throughput of an application. This happens if the system works to improve the rate at which data sets collection can be processed by an application instead of focusing on speeding up the application's task on a single piece of data. Effectively utilising each and every individual system resources like cache, memory, processor cores etc improves systems performance.

1.1 Overview

The simulator of the processor core used in the project can predict all the memory latencies accurately whatever happens beyond the L3 cache. In fact the simulator excludes both the L1 cache and L2 cache models. The things beyond L3 cache involve L3 cache misses, L3 cache evicts, system probes, cache coherence etc. If we can predict the memory latencies that happen beyond L3 cache then we do not need to do the modelling, we can simply add the latencies and get the simulation done instead of modelling different scenarios i.e transactions beyond L3 cache. If the L3 cache miss occurs it adds a request to the entire pipeline stage and then raises the request for the Dram which are both huge overheads.

The Accurate SoC Simulation model is the model which predicts the memory latencies accurately but the problem with this model is it has the worst simulation run time overhead. So we want alternate solutions to the models which run with very less time and should

match the accuracy of an accurate SoC simulation model.

Already existing model is the Fixed Latency Model, which is a naive way of calculating latencies with fixed values, so whenever beyond L3 cache happens it just uses fixed values instead of modelling the L3 cache beyond. As the values are fixed one these simulations happen very fast and are not accurate. So we go for the other model, which is the Machine Learning Model.

As in different performance evaluation studies, it's clear that the L3 cache beyond things are modelled accurately, now instead we wanted to do accurate simulation once with accurate SoC simulation model and from the other time we are not modelling it using the accurate model instead we use machine learning model to predict L3 cache beyond latencies accurately.

So the main reason for shifting to the machine learning model is we don't want the latencies of the Accurate SoC simulation model and the expectation from the machine learning model is the run time would be similar to the fixed latency model and accuracy will be closer to the proper simulation of L3 cache beyonds.

1.2 Problem Statement

To improve the performance of a Micro-Architectural Simulation, by reducing the memory latency with Performance Modelling techniques, deploying Machine Learning models and parallelising the required components by using more computer resources.

Chapter 2

Background

2.1 Performance Modelling

Modelling is the process of representing a model which has a detailed process of the system's working principles and its construction, which will be similar to the real system. The data obtained in these models help to predict and analyse the changes that happen in the real system, considering various improvement factors with the motivation of maintaining the performance improvement.[2]

So the performance Modelling involves analytical methods, tools and some advanced techniques that leads to improvement of the productivity of an existing or planned system, along with efficiency. The various phases where the performance Modelling is applied are:

- In Requirement gatherings
- In Design
- In Performance Test
- In the Production Environment.

Performance Modelling in Requirement Gathering:

This stage's main objective is to identify Infrastructure workload and key business units across the various tiers and also to determine the total non-functional requirement across the system.

Performance Modelling in Design:

At the design stage, we have design specifications and detailed high level views. We can now choose simulation and analytical Modelling techniques to authenticate the performance of the systems underlying infrastructure and architecture.

Performance Modelling in Performance Test:

At this stage, the system has generated huge amounts of the data, which is a good input for getting a lot of statistical data through various performance runs. The statistical Modelling data are useful when there's a huge difference between the size of the production environment and performance testing and based on the results of the performance testing the scalability and the performance of the system can be extrapolated.

Performance Modelling in Production:

In the actual environment, the statistical Modelling techniques identifies relation between infrastructure workloads and business workloads for very critical applications. The statistical Modelling techniques are used to predict when there's a change in application performance if baseline configurations have not changed. Different combinations of Analytical and Simulation Modelling techniques can be used in scenarios like what-if analysis of different hardware configurations.

2.1.1 Significance of Performance Modelling

Performance Modelling delivers a group of techniques that sync to the modification in application performances for different workloads to predict system performance based on the key design and infrastructure decisions.[3]

1. For validating the design decisions.
2. To meet the user end user expectations in all levels of design.
3. To validate infrastructure specifications and also to get guidelines for modifying them.
4. To extrapolate the performance of the application from the data obtained by the performance test.
5. To forecast changes in performance due to the application of different configurations.
6. To forecast changes in performance due to the application of different architectural specifications.
7. To estimate the load balancing capabilities of the system in real time conditions.

2.1.2 Tasks involved in Performance Modelling

- To understand System and applications Architecture.
- possible obtain performance data from existing applications.
- To validate results using a combination of Modelling techniques.
- To understand business goals and objectives.
- To know which modelling techniques are applicable.
- To determine the non functional requirements and business workload.

2.1.3 Difficulties Involved in Performance Modelling

- To visualise and analyse data from performance testing it becomes difficult without the analytical tools.
- Lack of information around the System specifications and application architecture due to poor documentation

- Lack of understanding the details about the performance modelling across various stakeholders creates many difficulties.

2.1.4 Goal and Applications of Performance Modelling

By measuring and analysing the various applications in the system then grouping these characteristics in a compact formula to get the complete understanding of the system is the main aim of the performance modelling and these results are also used to get more performance gain when used with other combinations of the applications and hardware. This leads to a complete understanding of the performance phenomenon and also huge performance gain.[4]
There are unlimited ways of using performance modelling.

- Run-Time Estimation
- System Tuning
- Application Tuning
- System Procurement
- System Design

Run Time Estimation:

It's one of the most common applications of performance modelling. When the different number of parallel processors or different architectures and configs and also when input parameters of the existing system changed then by using Performance Modelling we can estimate the run time of the task.

System and Application Tuning:

When some simple performance models were introduced into the code then they can improvise a huge performance gain. For example when application parameters are combined with

the memory performance model then predicting the cache hit rates becomes easy when the different blocking factor of cache is used in the system.[5]

System Design and Procurement:

To make the future technologies standardised in the current system, generally engineers construct a performance model for one or two applications. Now they will test all the what-if cases with different application parameters and different investigations, if the all results are satisfactory then they will adopt them to the new technologies and also target many more applications. In the same way many other parameters of the system like networks, parallelisation etc are explored. Currently, most large system procurement involves exhaustive testing of the used systems with a bunch of system and application benchmarks. Once we reach a reasonable performance modelling facility, then we can remove those costly benchmarking tests and replace them with the specific parameters which targets the exact areas of the targeted applications and systems.

2.2 Computing Benchmarking

Benchmarking is the process of accessing the performance characteristics of the object by running some programs or other tasks. These benchmarks allow us to compare the performance of different processor architectures which are very complex in nature. There are two types of benchmarks 1. Application benchmark. 2. Synthetic benchmark. The application benchmarks usually run on the real world applications on the system, whereas the synthetic benchmarks perform the tests on the selected components which will be curated especially for applications like networking and storage devices.

The benchmarks give architecture designers the ability to make the tradeoffs in architecture design and also they extract the most sensitive part in an application where the application behaves differently for different inputs. Generally, CPUs like Very Long Instruction Word, reconfigurable CPUs and superscalar CPUs have very slower clock rates

than the normal sequential CPUs. So such multi-execution unit CPUs generally complete the benchmark tests faster than the expected time.[6]

There are several important features of the benchmarks such as.

1. The benchmarks should equally treat all the systems with which testing is going on.
2. Benchmarks should be cost effective and should be able to understood easily.
3. Benchmarks should maintain the Industry standard norms.
4. Benchmarks should measure important features relatively.
5. The performed benchmarks should be re verified.
6. The benchmarks should have the ability to work for different ranges of resources.

Common Benchmarks

There are several kinds of Benchmarks.

- Industry Standard
- Open Source Benchmarks
- Microsoft Windows Benchmarks
- Others

Industry Standard Benchmarks

SPEC - Standard Performance Evaluation Corporation

BAPCo - Business Applications Performance Corporation

TPC - Transaction Processing Performance Council

EEMBC - Embedded Microprocessor Benchmark Consortium

Open source benchmarks

- AIM Multiuser Benchmark - AIM7 or AIM multiuser benchmark suite mostly used by the UNIX computer system vendor and is composed of a list of tests which creates a load mix that simulates a specific computer function.
- DiskSpd - Its a command line tool for storage benchmarking that generates a variety of requests against system files, storage devices or partitions.
- Iometer - Its a benchmarking and troubleshooting tool and is used to replicate behaviour of many applications also used for single and clustered systems for characterization and I/O subsystem measurement.
- NBench - Currently known as BYTEmark and peviously known as NAtive mode benchmark which is synthetic benchmark suite used for measuring performance of floating point arithmetic, memory operations and Integer arithmetic.
- Whetstone - Its mainly used for floating point arithmetic performance, mostly the units are represented in MWIPS which is Millions of Whetstone Instructions per Second.

Microsoft and Other Benchmarks

SuperPosition benchmark, SuperPrime, SuperPI, PiFast, CrystalDiskMark etc are some mostly used benchmarks from Microsoft and AnTuTu moslty found testing for ARM based devices, Geekbench almost used for all platforms such as MacOS, Linux, Android, iOS, Windows, VMmark used for testing virtualization etc are some more most popular benchmarks.

2.2.1 SPEC Benchmarks

SPEC is a non-profit American Corporation found in 1988, released first set of their benchmarks for UNIX Systems version 1.0, which maintains and produce a standardized set of performance benchmarks for computers that are reasonably scientific, relevant, unbiased,

meaningful.
The suite consists of 10 programs which could be run and measured to produce three scores. Overall SPECmark, floating point SPECmark, integer SPECmark. For a program the score was measured by comparing the speed of running a program on a test machine relative to speed of the same program running on the reference machine.[7]

To measure integer performance the SPECint score which is geometric mean of speeds of the four programs, Similarly SPECfp was measured by taking geometric mean of the other six programs and finally mean of all the ten programs yield SPECmark score. Previously the SPECint, SPECfp were called as integer SPECmarks and floating SPECmarks respectively.

Later SPEC used a larger number of programs to measure performance and came up with the SPECrate metric for multi-CPU machines. The formula is bit more complicated because it needs to include an extra variable for number of copies that were running simultaneously, The SPECrate measurement includes running many copies of a benchmark program simultaneously.

In later year with offset of five years of span SPECC released new versions of the benchmarks with larger data sets, each with larger amounts of the code, greater number of programs than its predecessors.
These SPECC benchmarks are continuously updated because of two problems that mainly affect most of the benchmarks but not all.
1.As newer better machines arrive to the market, newer tasks becomes feasible and these newer tasks are composed of several mix of resources, such as caches, total memory, registers, multiply or divide, etc.
2. Lack of scalability - Most synthetic benchmarks do not auto-scale, a benchmark should be task that should be of same size which corresponds on the machine where benchmarking

is done.

SPEC grown into a umbrella organization encompassing four diverse groups.

1. GWPG - Graphics and Workstation Performance Group
2. HPG - High Performance Group
3. OSG - Open Systems Group
4. RG - Research Group

The benchmarks aims to test real life situations. The SPEC discourages comparing results from one version to other version of SPECC metric. Its mainly due to the following reason. The newer metrics or new suite involves larger memory usage, greater number of programs and greater running time when measured with the programs from older benchmark suite running on same test suite.

For example when compared SPECint89 with SPECint95 programs, as SPECC89 version uses less memory than SPECC95 then the system with large amount of RAM and small CPU data cache will do worse comparatively worse on SPECC89 and comparatively better on SPECC95. But mostly real machines have balanced amount of memory, cache and other necessary system components. Due to this, SPEC benchmarks of consecutive releases are correlated quite closely in real.
In the current project SPEC CPU 2017 is used for performance modelling, The Sun Fire V490 server53 with a four 2.1-GHz UltraSPARC-IV+ processor chip is used as a reference machine.
SPEC CPU 2017 Benchmark suite consists of four suites. The SPECrate 2017 Floating Point and SPECrate 2017 Integer suites measures work per unit of time or the throughput. The SPECspeed 2017 Floating Point and SPECspeed 2017 Integer suites are meant for

comparing time to complete single tasks for a computer.

SPEC distributes source code files to users to analyse the systems. Those files are written in general programming language, which are then compiled for each particular operating system and CPU architecture. Thus performance measured is that of the compiler, RAM, CPU, and does not test I/O, graphics, or networking.

Two metrics are listed for a each and every benchmark, base and peak. Mainly the compiler options accounts for the difference. Peak has a less strict set of compilation rules than Base. Less optimization can be done, the compiler flags must be the same for each benchmark, in the same order and there must be a less number of flags. The peak metric can be performed with maximum compiler optimization to the extent of various optimizations for each benchmark. This number mentions maximum system performance, achieved by full compiler optimization.

Some of the benchmark programs are xalancbmk which is XML Processing which transforms XML documents to other document types, mcf Combinatorial Optimization which is vehicle scheduling uses a network simplex algorithm to schedule public transport, deepsjeng Artificial Intelligence chess playing which is a highly ranked chess program that also plays several chess variants, Omnetpp which is a Discrete Event Simulation type and uses the OMNet++ discrete event simulator to model a large Ethernet campus network etc programs are included in SPEC benchmark suite.

2.3 Recurrent Neural Networks

All above mentioned ML models are Recurrent Neural Networks. The problem with traditional neural networks is that they erases all its information after once a input was processed, but those information are vital in predicting outputs. This was addressed by

the Recurrent Neural Networks by the loops in the networks which allows the information to persist even after next input sequence.

A group of NN A, takes few x_t input and single value h_t is outputted . A loop makes information passes through one network to other network. These RNNs are different copies of same network, where each network passes message to its successor network. An unrolled recurrent neural network. The architecture of neural network is designed such a way to predict data which are temporal related sequences and lists.

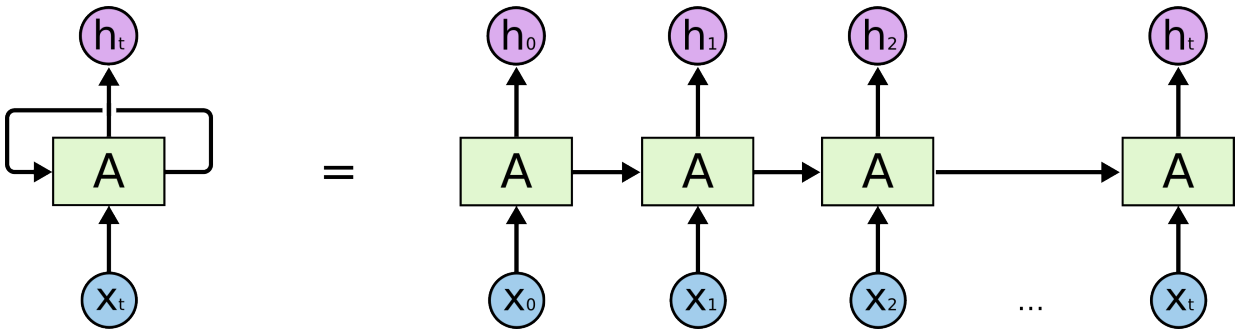


Fig. 2.1 An unrolled recurrent neural network

Most important feature in Recurrent Neural Networks is to connect previous information to present task, the Recurrent Neural Networks are functioning properly i.e to learn from the past information when the gap between the current place thats needed and relevant information is less but the places where the gap between where its needed and the relevant information is very large the RNNs struggle to learn to connect the information. Fortunately RNNs dont have this long term dependencies problems. [8]

Long Short Term Memory networks a special kind of Recurrent Neural Networks also called LSTMs were introduced by Hochreiter and Schmidhuber in 1997 which are capable of learning long term dependencies. Repeating modules of neural network are present in every RNNs. Its a very simple structure with only tanh layer present in it.

LSTMs alike traditional RNNs have chain like structure but has different repeating module structure, here LSTM has one neural network layer instead of four interacting layers.

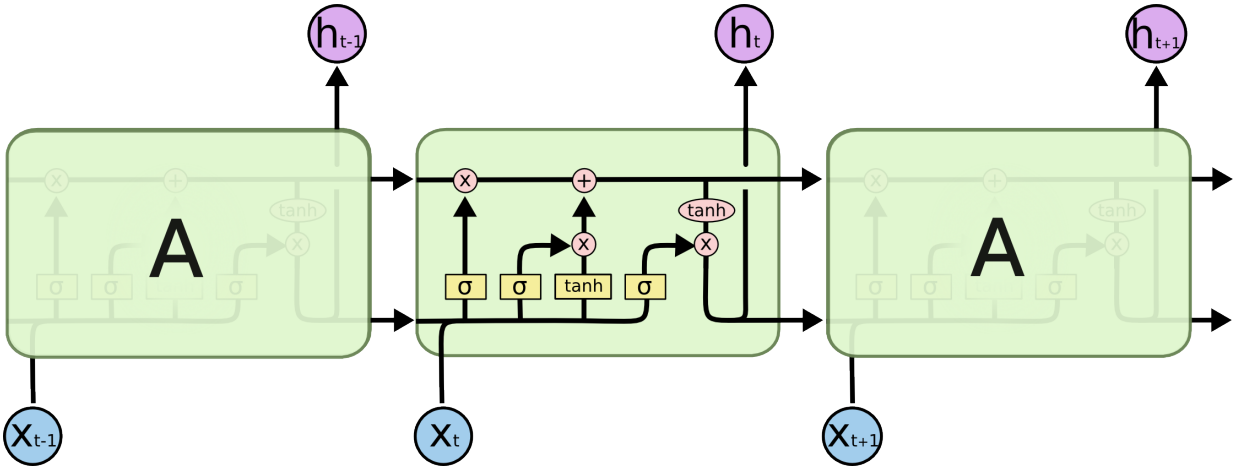


Fig. 2.2 Four interacting layers in LSTM

These are the notations used

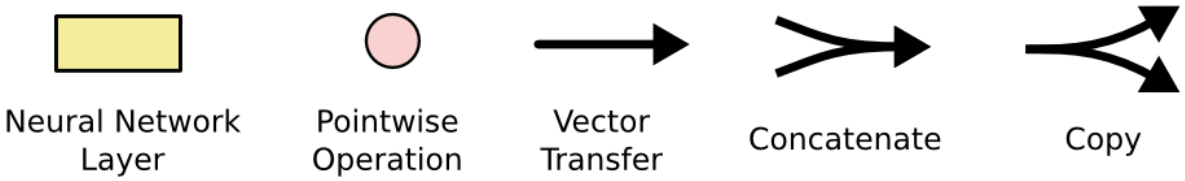


Fig. 2.3 Notations

Each line takes an full vector, from the output of one node to inputs of others, the rectangular boxes are neural network layers that learned the data. Point-wise operations are represented by the dots, lines splitting represent copying and lines merging represent concatenation.

Main Idea Behind LSTM

Cell state is the horizontal line in neural network passing through the top of the module, which is a key element to the LSTM. Its easy for information to just flow and unchanged only few linear iterations are present.

Gates are structures which has ability to add or remove the information to the cell state, They are composed of multiplication operation and sigmoid neural net layer. Gates

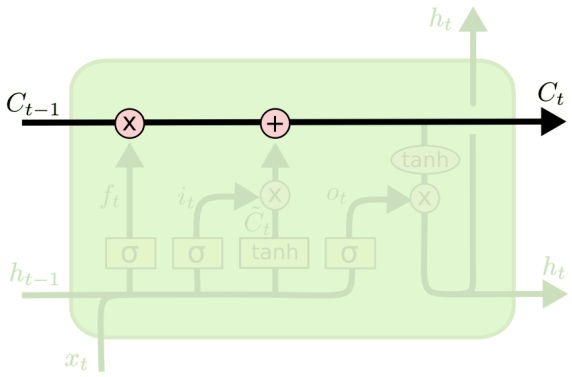


Fig. 2.4 Cell State

are a way to optionally allow information pass through.

Sigmoid layer outputs values between 0 and 1. Which controls how much of each component should be allowed through. If value is 0 then it means allow nothing and if the value is 1 then it means allow everything pass through it. To protect and control cell state Long Short Term Memory has three of the sigmoid layers.

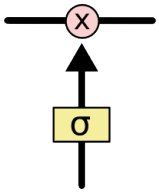
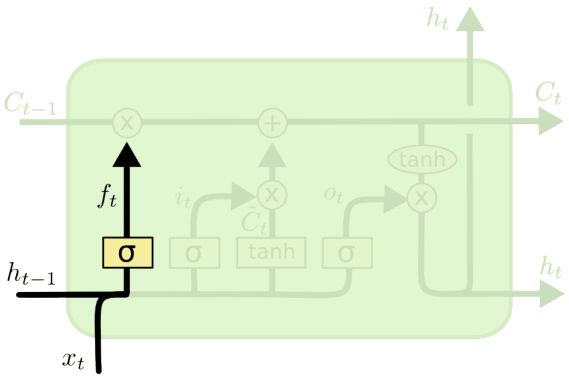


Fig. 2.5 Sigmoid

The first step in LSTM is made by sigmoid layer called forget gate layer which makes decision of what information was going to throw away from cell state. The inputs to the forget gate are previous output state h_{t-1} and current input state x_t, for each number in cell state C_{t-1}, outputs a number between 0 and 1.

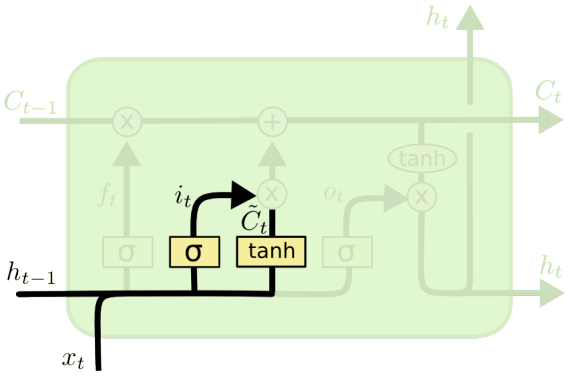
The next step is to unite these two updates to create a new update to the state. first is input gate layer which is a sigmoid layer that decides which values are to be updated. Other is tanh layer which creates a vector of new candidate values, C_t that can be added



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Fig. 2.6 Forget Gate Layer

to the state. All these modification decides which new content is going to present in the cell state.

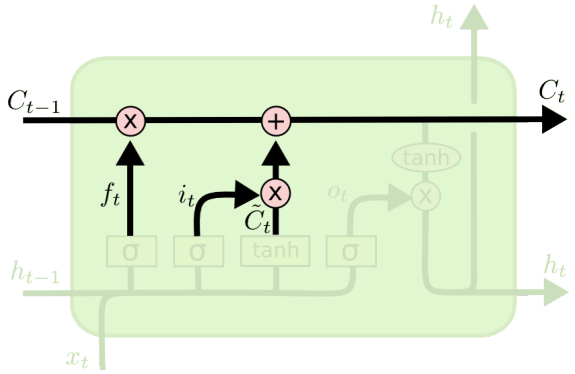


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Fig. 2.7 Input Gate Layer

Next step is to update the old cell state, C_{t-1}, into the new cell state C_t. The previous steps are already decided, we just need to actually do it. We multiply the old state by ft to forget the things we decided to forget earlier. Then we add $i_t * C_t$. This is the new

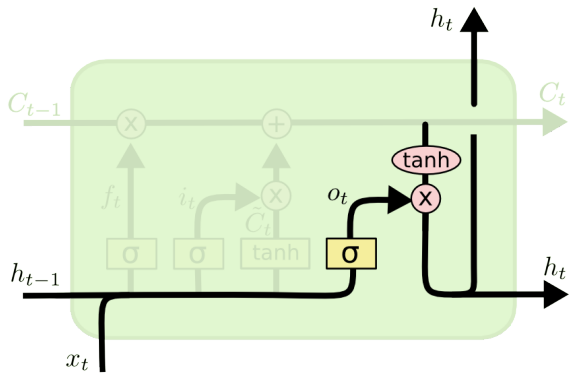
candidate value, scaled by how much we decided to update each state value.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Fig. 2.8 Updated State

Finally, we decide, what we are going to output. This output will be based on our cell state, but will be a filtered version. First, we run sigmoid layer that decides what parts of the cell state are going to the output. Then, we put the cell state through *tanh* to push the values in range of -1,1 and multiply it with the output of sigmoid gate, so that we only output the parts that we decided to.



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Fig. 2.9 Final Output Step

There are many variants of LSTM like peephole connections here main modification is gate layers look at the cell state. Bi-LSTM:(Bi-directional long short term memory):

Bidirectional recurrent neural networks are truly just placing two independent RNNs together. This structure allows the networks to have both forward and backward information about the sequence at every time step. BIDs will evaluate inputs in two ways,

one from future to past and one from past to future and what differs this approach from unidirectional is that in the LSTM that runs backward which preserves information from the future and using the two hidden states combined we are able in any point in time to preserve information from both future and past.[9] Other more interesting one is GRU or Gated Recurrent Unit introduced by Cho, et al in 2014. It merges the cell state and hidden state and also unites the input and forget gates into a single "update gate." GRU is less complex than LSTM models and has increased its popularity in recent times.

2.4 Clustering

Clustering is the process of separating the data points or the population into several groups, such that the members of one group are much more alike to the other members of the same group.

Clustering can be classified into the two types:

Soft Clustering: A likelihood or probability of assigning a data point to a particular cluster is assigned.

Hard Clustering: In hard clustering each data point does not have the approximation or chance to be a particular cluster but it has definite information either the data point belongs to a cluster completely or not.

Different Types of Clustering algorithms

Density models: The data space for the areas of different density of data points are searched by the model. The data points within various different regions are assigned to the these regions in the same cluster by separating various different density regions. DBscan, Optics are popular examples of the density model.

Centroid models: The closeness of a data point to the centroid of the cluster with various iterations to find local optima, decides the data point which cluster it belongs to. K-means clustering algorithm is the most popular clustering algorithm that belongs to this centroid model. In this type of models, user needs to mention the number of clusters required beforehand hence to use this model prior knowledge of the data set is required.

Distribution models: These models often suffer with over fitting problem, this model depends on on how probable is it that all member point in the cluster belongs to the same distribution like Gaussian, Normal distribution etc.

2.4.1 DBScan Clustering Algorithm

Density-based spatial clustering of applications with noise. Two parameters are mainly important in the DBScan algorithm.

1. eps: Two points are considered as neighbors if the distance between them is less than or equal to eps parameter. eps defines the neighborhood around a data point. This value needs to chosen carefully. If eps value is choosen very large then most of the data points are will be in the same cluster. Most part of the data will be considered as outliers if the eps value is too small.

2. MinPts: Minimum number of neighbors or the data points within radius eps. General rule is that Minpts can be derived from the numbers of parameters or dimensions. For larger dataset, larger value of the MinPts is choosen.

Core Point: A point is said to be core point if a point has atleast Minpts within the eps.

Non Core Point: All points other than the non core points are treated as core point.

Working of DBScan

1. Randomly pick a point from the all points and check whether its a core point or not.[10]
2. Complete the above step for all the points.
3. Next step is randomly pick a core point, and name it as cluster and the non-core points lie within the eps are merged into the same cluster and the same step is done for the core points within the eps.
4. The core points which are away from the cluster forms the other cluster.

2.4.2 K-Means Clustering Algorithm

The data points are assigned to clusters by using the Euclidean distance which is sum of the squares of the distances between the data points. The K in the K-means clustering

algorithm means number of clusters, This algorithm repeatedly computes the centroids until the optimal centroid is found. In this algorithm the number of clusters is known presumptively known. This algorithm is also known as flat clustering algorithm. The quality of the clustering can be assessed by adding up the variation within each cluster.

Working of K-Means Algorithm

Firstly one needs to analyse the data carefully as number of clusters needs to be decided priorly

1. From the set all data point randomly pick k points and assign them as the clusters.[11]
2. Measure the distance using the Euclidean distance formulae between the first point and the all the clusters.
3. Assign the point to the cluster on which the distance is minimum among the all clusters.
4. Now repeat the step three for all the data points.
5. Now calculate the mean of each cluster.
6. Repeat the above step i.e measure and cluster till the means of the clusters do not change.

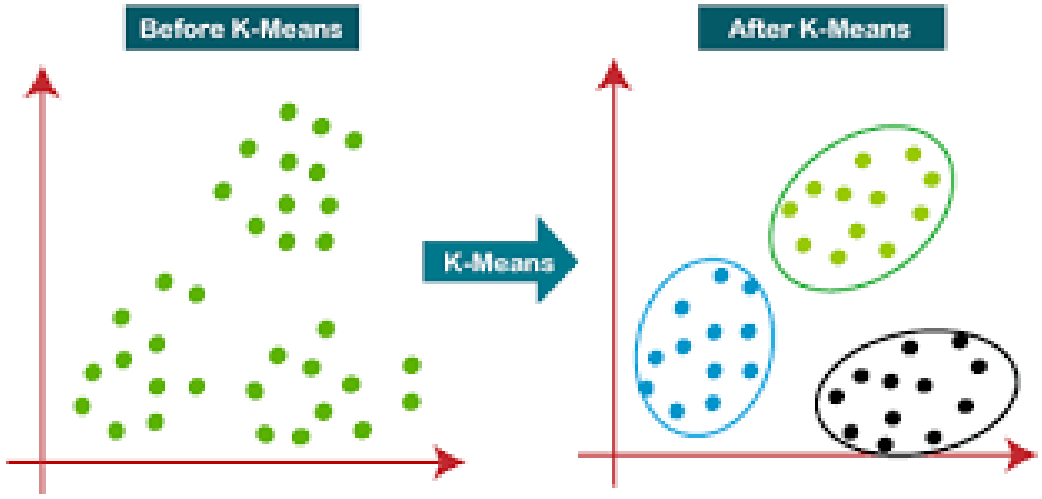


Fig. 2.10 K-Means Clustering

Chapter 3

Environments Used

3.1 VS Code

VS Code is an Integrated Development Environment made by Microsoft for most user operating systems. It's a boon for the programmers as it provides excellent features like syntax highlighting, auto completion, code refactoring, many keyboard shortcuts, supports many keymap extensions to replicate other editor, extension market.

Which again includes many applications that can be used within VScode like embedded Git, Jupyter notebook, C, C++, Python etc languages support, Remote SSH, debugging support etc. Unlike normal text editors, VScode uses a workspace that contains directory structure and thus it's easy to traverse the paths and apply necessary actions using extensions.

3.2 Sub Version Control System

Svn is a open source centralized version control system developed as a project of Apache Software Foundation, which is used to manage different temporal versions of several types of files that includes source code, documents etc files. Svn life cycle involves the follow-

ing operations such as create, checkout, update, perform changes, review changes, resolve conflicts, commit changes in repositories.

3.3 Load Sharing Cloud

Load Sharing Cloud(LSC) is suite of software which provides a framework to a cluster of systems for dispatching the jobs on cloud with intelligent scheduling algorithms.

In order to use LSC, User Environment must contain various configuration files location and port numbers etc. These compute systems are identical to normal workstations. We need to submit the jobs in the workstations to run it properly where runs expected by cluster. Everything like project work directories, user accounts that are accessible from workstation is also accessible from these server nodes.

The LSC server nodes gives enormous pool of compute resources to get a normal user accomplish their tasks by distributing all the workload across as many cpu cores as needed by the task where as the normal workstation would slow down other processes and also which takes huge amount of time compared to LSC for the work to get done.

Job Submission

The following are the basic components for a valid LSC Job submission.

- Which queue the job should be submitted into.
- What resource to request.
- What project the job belongs to.
- What limits to impose on the job.
- Which command to execute.
- Is the job require to be interactive.

Queue selection

Queue selection is based on the priority and also aside form priority number of jobs per user also plays key role. If a lots a slots needed like in this project to do some experiments with benchmark suite queue regressions are used where allocation of the jobs is controlled by fairshare.

Resource Selection and Reservation

The main common resource types here LSC concerned with are total available memory, OS type the system is running, named resources i.e alias for a group of systems.

LSC also provides support on project basis which means there can be more job slots in certain queues. Jobs can also be submitted in interactive mode if needed. LSC has a mechanism to automatically flag and terminate the jobs after some amount of time or jobs which disobeys long-running job policy. Hence it has a feature to specify estimated time and also to check job efficiency.

Finally the new RTM website provides various details of job running information such as jobid, clustername, jobname jobDescription, user, queue, projectName, command, errFile, inFile, outFile, from_host, exec_host, num_nodes, num_cpus, maxNumProcessors, exitcode, mem_requested, mem_reserved, wasted_memory, res_requirements, start_time, end_time, pend_time, run_time, runtimeEstimation, userGroup, parentGroup, mem_used, swap_used, max_memory, max_swap, cpu_used, core_eff.

Chapter 4

Overview of the Model

There are individual models of both cache, core hierarchy and detailed memory model(Data fabric - Control/Data path, Umc) combined to evaluate performance. Access Latency to main memory is typically dependent on static parameters - the DRAM topology and timing parameters and dynamic parameters - the application behavior which vary with time. The detailed memory model simulates the DRAM along with several components like Data Fabric, Coherent Master/Slave etc. The model is highly accurate but takes a considerable simulation time.

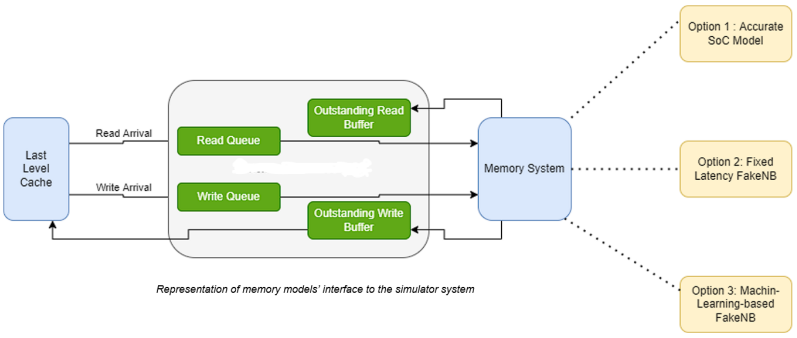


Fig. 4.1 ML Model

The memory requests are primarily generated by L3 miss (reads) and L3 victims (writes) that are serviced by the main memory. Each memory request arrives the memory interface

and gets queued into the request queue and is serviced by the main memory before it is sent back to the caches.The latency is calculated as the difference between time when the request gets serviced and to the time when it arrived at the queue.

If the output space of the problem is vast and extremely sparse, it makes the problem poor fit for standard regression model. The output space of the memory latency prediction problem is spread across 100 to 1000 cycles range. Therefore its good for solving using regression method.

Here inference is to predict the memory access latency of Nth instruction, an input sequence of length N is considered. It contains the request information including memory access latency of N1 previous requests a long with the information of the current Nth request. The main idea is to collect the details about the memory transactions in memdump file, the physical address, read/write information, etc. and design a machine learning model to predict latency.

4.1 Latency Models

As said in the Introduction part, we have three different models to predict memory latency. which are:

1. Accurate SoC Simulation Latency Model
2. Fixed Latency Model
3. Machine Learning Model

Actual SoC Simulation Latency Model:

This memory latency model gives the exact values of the applied configuration. It will be the most accurate model, but the actual problem is that for getting the values of different memory latencies we cannot actually run the tests with actual SoC as its most time taking

model. That is the reason we are coming to other models which can give the results with lower delay and most accuracy.

Fixed Latency Model:

This model is a fixed Latency Model. Here instead of taking the values from the actual SoC we actually generate the random or approximate values to get the memory latency values. As it does not consider the original model, when it comes to the accuracy this model gives the least accuracy and also this will be the model with the least computation effort required than all the other existing models.

Machine Learning Model:

The main aim of this model is to estimate the memory latency in the SoC models with different configurations available without actually using the actual SoC simulation. When it comes to accuracy. If the ML models are quite good then the best possible outcome can be delivered that is as par with the main SoC model and when it comes with execution time it runs at par with the approximate latency model, which is best of both the models and that is what we wanted here to achieve.

In the current version of the ML model three models were implemented which are:

a. LSTM - Long Short Term Model

b. BID LSTM - Bi-Directional LSTM

c. GRU - Gated Recurrent Unit

4.2 Memory Log Dump File

By using the Memlog module we are generating a memory log dump file of all the transactions that go beyond L3 Cache. Our model has capability to capture all the Cache lines that are evicted from the L3 Cache. As discussed in the introduction part, the Accurate

SoC Simulation model can accurately model the transactions that go beyond L3 Cache, now we can find the actual memory latency from the simulation.

So for each benchmark trace, we are connecting the Memlog module and then we generate a model out of this memory log dump file and that model is used for predicting the latency of the same benchmark trace. Below is the example of the memory log dump file of a random benchmark trace.

	PA	R/W	PF?	CORE	REQ_TIME		RESP_TIME		ReqtoReq_In		RtoR_In	RtoW_In	WtoR_In	WtoW_In				
	WRPEND	OUTSTAND		CUMREFRESH		CUMRBH		IsClean										
2	10000aa414140	1	0	0	127	193	28	6	28	99	127	0	0	0	0	3	0	1	1
3	10000adc17d80	1	0	0	138	202	11	6	39	99	11	0	0	0	0	4	0	1	1
4	1e00000000150	0	0	0	86	290	86	86	0	86	0	0	0	0	0	0	0	1	0
5	1e000000007f8	0	0	0	93	299	7	7	0	93	0	1	0	0	0	1	0	1	0
6	1e00000000180	0	1	0	99	308	6	6	0	99	0	1	0	0	0	2	0	1	0
7	10000a52199c0	1	0	0	183	362	45	6	84	99	45	0	0	0	0	5	0	1	0
8	10000ad812680	1	0	0	395	461	7	4	7	205	212	0	0	0	0	3	0	1	1
9	10000bce1c3c0	1	0	0	435	500	40	4	47	205	40	0	0	0	0	4	0	1	1
10	1e00000001aa8	0	0	0	368	592	185	269	84	185	45	0	0	0	0	0	0	1	0
11	1e00000002ff8	0	0	0	388	628	4	4	84	205	45	0	0	0	0	2	0	1	0
12	1e00000001ac0	0	1	0	384	637	16	16	84	201	45	1	0	0	0	1	0	1	0
13	10000baa19000	1	0	0	686	757	5	15	5	246	251	0	0	0	0	2	0	1	1
14	10000baa19040	1	0	0	710	775	24	15	29	246	24	1	0	0	0	3	0	1	1
15	1e00000004ff8	0	0	0	717	919	7	36	29	7	24	0	0	0	0	4	0	1	0

Fig. 4.2 Example of Random Memory Dump File

The memory dump can be generated with different Cache configurations, SoC configurations and also with different latency models. All these can be controlled by running the processor core simulator with different options such as -simulator configs, -config etc. There are several other options that can be used with the simulator which includes fast warmup which means initialising the memory modules when the simulator run begins, next the core warmup which initialises the core modules like branch predictor in the processor core simulator, next is the pre-warm-up which used for the initialising of the cache systems in the SoC.

The -config option can be used to give configurations for the core of the system. The -simxconfig option can be used to give different configurations for the entire SoC.

Attribute Name	Definition
PA	Physical Address
I/D	Instruction or Data
R/W	1 if request is for write else 0
PF?	Is Prefetch?
ReqtoReq_In	Time delta between two memory requests
RtoR_In	Time delta between read in and last read request (holds till read is seen again)
RtoW_In	Time delta between write in and last read request
WtoR_In	Time delta between read in and last write request
WtoW_In	Time delta between write in and last write request
OUTSTAND	Number of outstanding transactions beyond L3
IsClean	Is request == CpuEvictClean
Latency	Calculated as response time - request time

Table 4.1 Attributes Details of Memory Log Dump File

4.3 Features in Memory Log Dump

Physical Address

It is the address of the data, in the main memory, which is needed by the incoming memory request. Physical address can be translated into channel, rank, bank row, and column address. Therefore, a physical address can give the information about the DRAM command that could be possibly executed later.

For example, the new request might need the data from a particular row or column that is already there in the buffer which leads to a buffer hit so requests are serviced faster. Physical address of requests in the queue can also give us the information of any parallelism the request utilizes in the DRAM. This feature targets row buffer locality and parallelism.

Read or Write Request

These requests differ in the sequence, type of commands sent, also differ in scheduling differently by the DRAM controller, therefore they have different access latencies. Generally, reads are critical so they are preferred over the writes. Hence, the latencies of processing

reads would be different from the writes. This feature distinguishes how reads and writes are processed differently inside the DRAM.

Prefetch or Demand Request

It can vastly differ in the latency. But prefetch requests are already following some pattern of memory access. So, the predictability of the latency is expected to improve with the information based on the current request is a prefetch or demand. This feature indicates the implicit locality that could be potentially present in the sequence of requests.

Num Pending Requests

It is the number of requests that are waiting to be dispatched in the request queue of the cores. This feature gives information of the utilization of the queue and also average time the requests spend in the queue.

Num Outstanding Requests

It is the number of requests that have been dispatched controller but have not yet been serviced by the main memory. This feature gives the information regarding the level of concurrency or potential parallelism in servicing the requests inside the main memory and also, its utilization.

Request to Request Gap

This tracks the arrival rate of each type of request. There is a separate feature for each kind of request gap, they are Read to Read, Read to Write, Write to Write and Write to

read. This feature captures the benchmark's and the core's behavior. This also gives the timing parameters that typically indicates the read/write switching modes inside DRAM.

Instruction or Data

Generally, instructions follow a better locality than data. Hence, distinguishing instructions from data would provide better predictions for latency of instruction requests.

Victim Type

Mostly clean victim requests are processed early. Hence distinguishing the victim type as dirty or clean helped better in predicting the low latencies accurately. So, this feature has direct impact on the low latency predictions.

Latency

The time taken for the request to be serviced by the main memory. As request arrival at the queue is the feature that is being predicted. Hence history of latency information helps the ML models to detect patterns in the prediction.

4.4 Correlation of the features

Clean or Dirty victims are less correlated with latency history. The main reason for this is latency would be less for clean victims. The arrival rates show lesser but yet some correlation. Read to read arrival time shows the maximum correlation.

Read /Write information shows less correlated i.e latency is inversely correlates with number of writes. If the number of writes are more, the latency is less, as the number of writes are high, the DRAM is expected to switch to write mode and start flushing writes.

Finally history of latency information, a request as a prefetch or not, Physical address has direct correlation with the current latency.

Chapter 5

Implementation

The project involves a complete architecture of SoC simulator where it contains all the modules like memory, caches, integer unit etc all the basic blocks which are of real hardware and in this system, there are totally 2 types of the Cache Hierarchies. Let's say them as Cache Hierarchy1 and Cache Hierarchy2. Also, there are different versions of memory configurations such as Memory Config1, Memory Config2, Memory Config3.
Now The testing is going on the evaluation of memory latencies in each of the configurations mentioned above.

From all the available cache hierarchies and SoC configurations. Only a few combinations of configurations are used in the project.

	Cache Hierarchy	SoC Config
Configuration1	Cache Heirarchy1	MemoryConfig1
Configuration2	Cache Heirarchy2	MemoryConfig2
Configuration3	Cache Heirarchy3	MemoryConfig2

Table 5.1 Different SoC Configurations

Simulator Built:

After making sure that all the files in the entire project code base are up to date by running the command svn up which updates all the files to the current version or just maintaining all files in the same revision. We can compile the simulator core by using the necessary make command.

Benchmarking:

Each Trace is classified as a region of the application. The performance study is done by executing multiple suites which consist of multiple benchmarks and again each benchmark consists of multiple traces or a list of instructions on a processor core.

The benchmark suite is classified into Integer Benchmark and Floating point Benchmark. In this project, we are only performing the tests using SPEC benchmarks- Standard Performance Evaluation Corporation, which again consists of both Integer benchmarks and floating point benchmarks.

As benchmarking is done on the huge parts of the application. The execution time of evaluating the benchmarks takes a very long time. Hence we divide the application into selected small representative regions called traces. These regions are chosen in such a way that they capture the entire benchmark accurately. Hence they are called representative regions of the benchmarks.

5.1 Methods to Reduce Training Time

Main objective of the project is to reduce training and evaluation time of existing Machine learning models. Below are some of few ideas to reduces training time.

1. Decrease the number of parameters in the model.
2. Change ratios of the test and training division.
3. Reduce the number of layers for reducing the training error and increase number of layers for the reducing the test error.
4. Early stopping instead of setting large number of epochs use minimum number of epochs where error rate reaches its saturation value.

For simpler analysis in current system the traces in benchmark are divided based on the number of memory transactions count. i.e

1. If the memory transactions are in the range of few thousands - Small.
2. If counts are in range of above sixty thousands - Medium.
3. And rest in range of few million transactions - Large.

Initially the ML model contains the 12 features ('PA', 'R/W', 'PF?', 'CORE', 'REQ_TIME', 'ReqtoReq_In', 'RtoR_In', 'RtoW_In', 'WtoR_In', 'WtoW_In', 'OUTSTAND', 'isClean') for training the data. Assumption is that these 4 parameters ('RtoR_In', 'RtoW_In', 'WtoR_In', 'WtoW_In') confuses the model instead helping gain more accuracy. So, the study is done on 8 features excluding above mentioned 4 parameters.
To test it 10 medium level transaction count traces are chosen and ML script has been executed on them and generated three logs per trace each correspond to BID, LSTM, GRU.

Below are graphs comparing three models root mean square error and absolute error values.

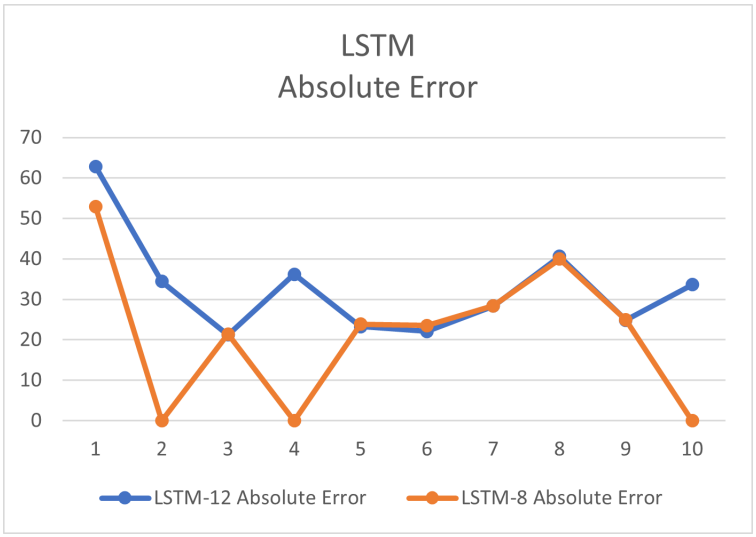


Fig. 5.1 LSTM Absolute Error 12 VS 8 Parameters

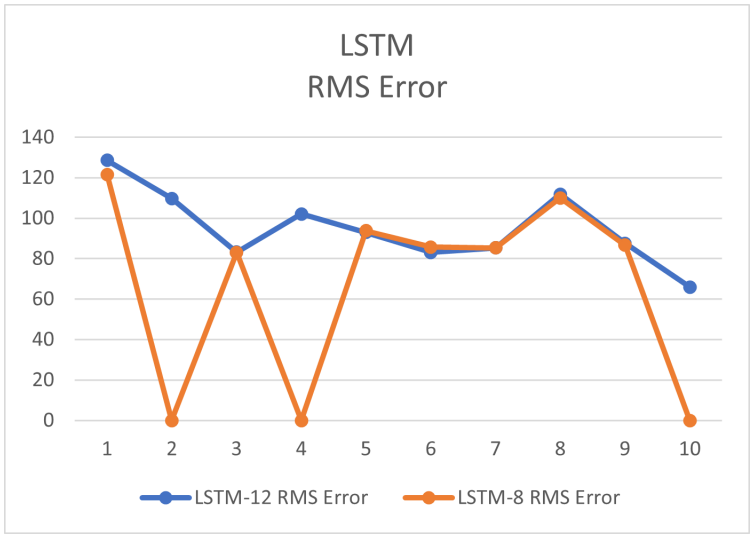


Fig. 5.2 LSTM RMS Error 12 VS 8 Parameters

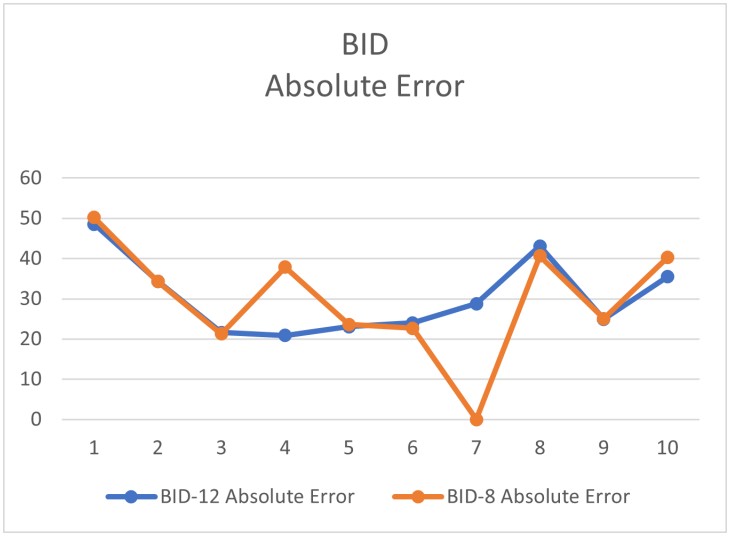


Fig. 5.3 BID Absolute Error 12 VS 8 Parameters

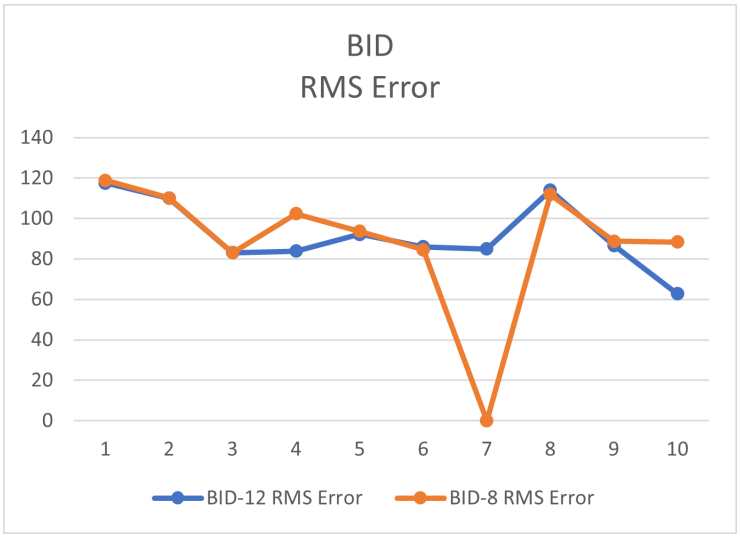


Fig. 5.4 BID RMS Error 12 VS 8 Parameters

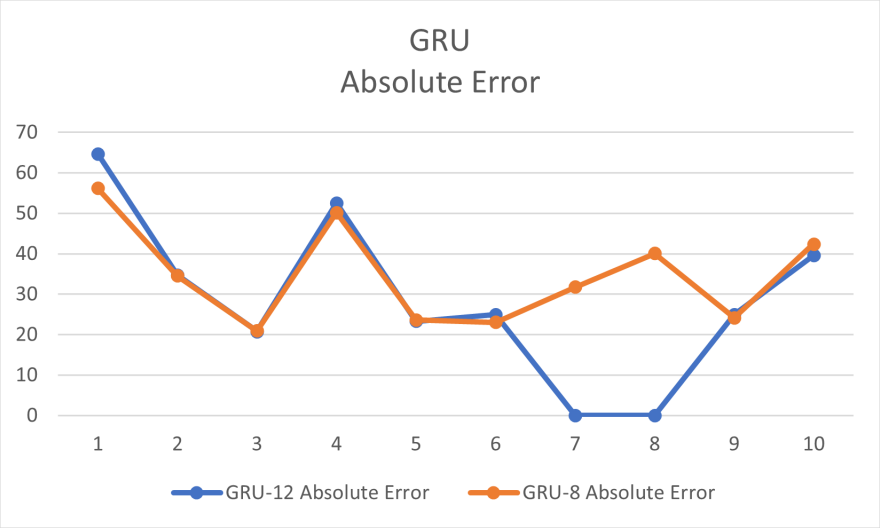


Fig. 5.5 GRU Absolute Error 12 VS 8 Parameters

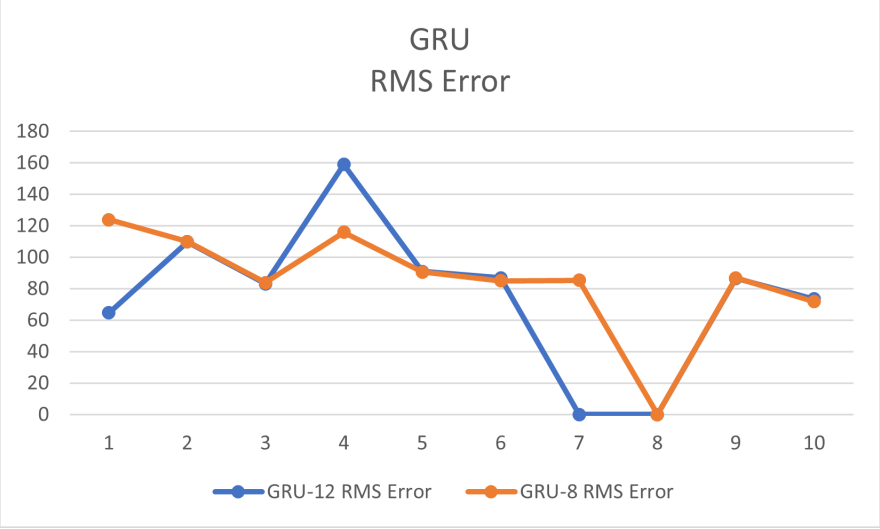


Fig. 5.6 GRU RMS Error 12 VS 8 Parameters

In all above cases its empirically observed that almost every time in three models the 8 parameter model has less absolute error and root mean square errors values than that of the 12 parameter one. Training time is very large almost it takes 2-3 days for the traces with very high transaction count. So possible methodologies to reduce the training time is to reduce number of inputs in a trace by grouping into smaller sets. so that training time taken for smaller sets are less than that of the larger ones. Currently for each benchmark trace we are building one model, so its n models per n benchmark traces and this type of approach is not scalable.

So we want to see how we can reduce or group traces by using clustering method or some other approach, so that we can reduce no of models, As we know approximately for a benchmark we have 30 traces, so by this new method we may have single model for two or more traces or entire benchmark, we can unite two traces by identifying similar properties between them, but finding the similarity is the challenge but metrics like below examples helps to achieve similar properties.

a. Transaction count.

b. Avg read and write LatencyCycles.

c. Number of same type of instructions(arithmetic, condition or addressing modes).

etc that would help us decide traces are alike or not.

5.2 Parallelising the Existing Machine Learning Model Scripts

Here the above mentioned Machine learning models already exist in the code base. We are generating a model per each benchmark trace by training the output of the actual SoC run which is a memory dump file. Here the ML models are executing in the sequential fashion which takes more delay per trace.

In an application there can be large number of such benchmark traces so which leads to huge latency overhead. Instead of the Machine learning models LST, BID, GRU all these models can be executed parallelly as there are no dependencies among the three models

and it can be achieved by running the script with command line arguments specifying the memory trace dump file, the machine learning model and launching them on a computer grid. Here the execution takes place on different clients in the widely distributed computer resource.

The train python script does the necessary preprocessing steps, say dropping some columns, converting the physical address from hexadecimal to the integer data type, sorting the data by the required time and calculating the memory latency from the response time and required time. Next, it performs the necessary scaling operations of the data using the pandas data framework with a fit transform function.

Then trains the three models and saves the three models in .h5 and .pb formats and finally generates the RMS error for all the models. Now the final evaluate script takes the generated RMS errors of the three models and among all the models it chooses the model with the least memory latency and saves that model and deletes all other models. Thus by executing parallelly on different clients we can reduce the computational delay by almost greater than 50%.

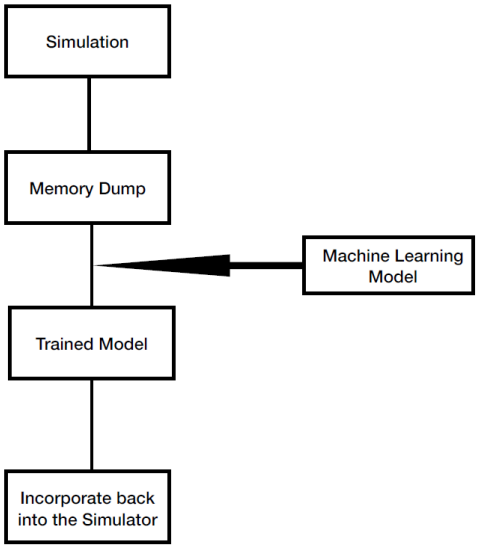


Fig. 5.7 The Behavior of Machine Learning Model

5.2.1 Outputs of Latency Prediction using three models

Below are the memory latency prediction outputs i.e absolute error and also root mean square error values of three models in which the execution of training the models took parallelly on the grid system:

```
scaled_data:  (14, 12)
train, test, validate, reframed:  (8, 84) (8, 84) (8, 84) (8, 84)
scaled_data:  (14, 12)
train, test, validate, reframed:  (8, 84) (8, 84) (8, 84) (8, 84)
scaled_data:  (14, 12)
train, test, validate, reframed:  (8, 84) (8, 84) (8, 84) (8, 84)
scaled_data:  (14, 12)
train, test, validate, reframed:  (8, 84) (8, 84) (8, 84) (8, 84)
scaled_data:  (14, 12)
train, test, validate, reframed:  (8, 84) (8, 84) (8, 84) (8, 84)
scaled_data:  (14, 12)
train, test, validate, reframed:  (8, 84) (8, 84) (8, 84) (8, 84)
scaled_data:  (14, 12)
train, test, validate, reframed:  (8, 84) (8, 84) (8, 84) (8, 84)
scaled_data:  (14, 12)
train, test, validate, reframed:  (8, 84) (8, 84) (8, 84) (8, 84)
scaled_data:  (14, 12)
train, test, validate, reframed:  (8, 84) (8, 84) (8, 84) (8, 84)
scaled_data:  (14, 12)
train, test, validate, reframed:  (8, 84) (8, 84) (8, 84) (8, 84)
scaled_data:  (14, 12)
train, test, validate, reframed:  (8, 84) (8, 84) (8, 84) (8, 84)
scaled_data:  (14, 12)
train, test, validate, reframed:  (8, 84) (8, 84) (8, 84) (8, 84)
scaled_data:  (14, 12)
train, test, validate, reframed:  (8, 84) (8, 84) (8, 84) (8, 84)
scaled_data:  (14, 12)
train, test, validate, reframed:  (8, 84) (8, 84) (8, 84) (8, 84)
scaled_data:  (14, 12)
train, test, validate, reframed:  (8, 84) (8, 84) (8, 84) (8, 84)
scaled_data:  (14, 12)
train, test, validate, reframed:  (8, 84) (8, 84) (8, 84) (8, 84)
scaled_data:  (14, 12)
train, test, validate, reframed:  (8, 84) (8, 84) (8, 84) (8, 84)
scaled_data:  (14, 12)
train, test, validate, reframed:  (8, 84) (8, 84) (8, 84) (8, 84)
predicted_values shape:  (5330,)
actual_values shape:  (5330,)
finished prediction
time taken for prediction 327.71132493019104
LSTM:
Mean Absolute Error: 18.6721
Root Mean Square Error: 43.6070
```

Fig. 5.8 Latency Prediction Using LSTM

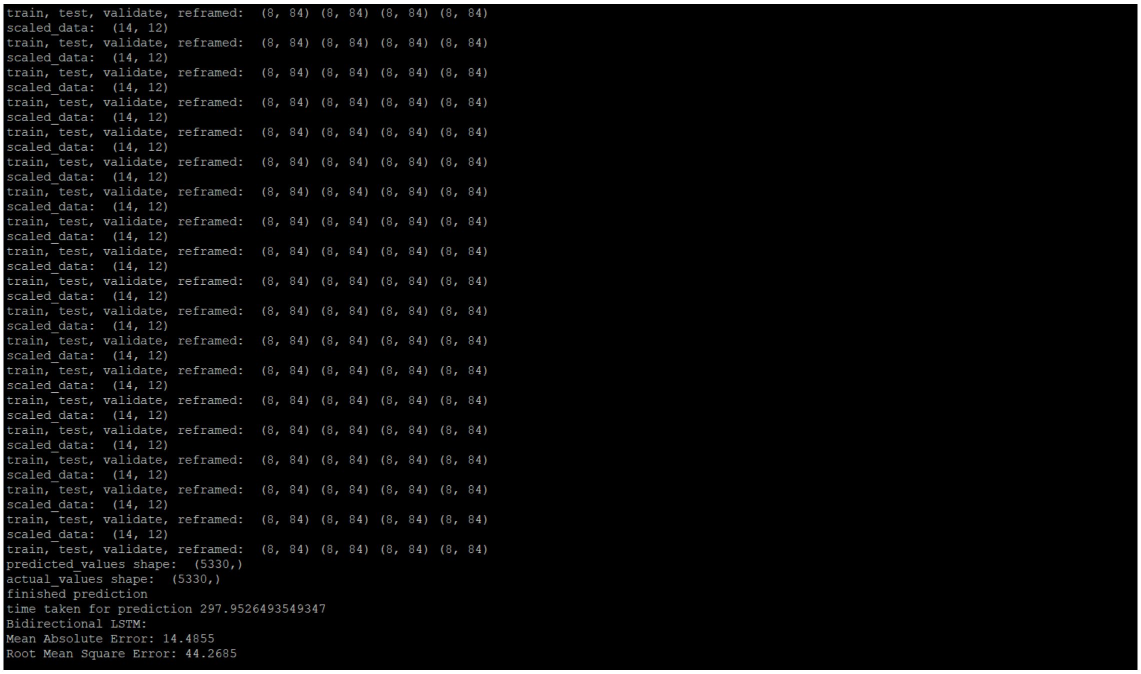


Fig. 5.9 Latency Prediction Using BID

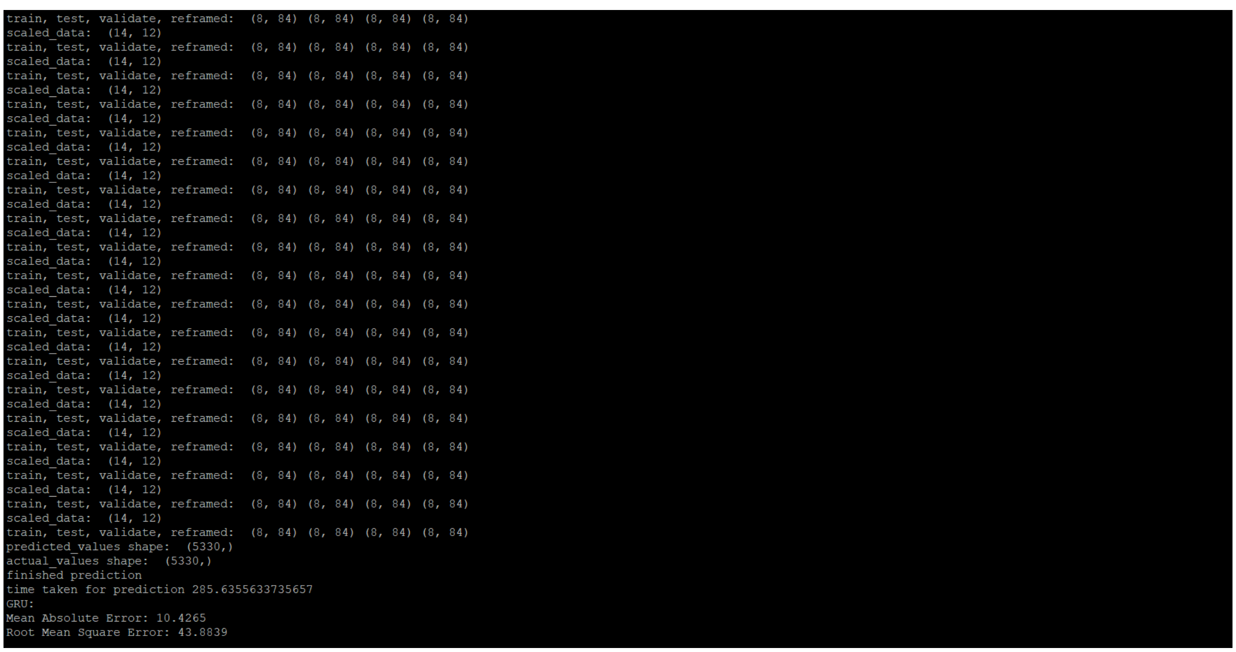


Fig. 5.10 Latency Prediction using GRU

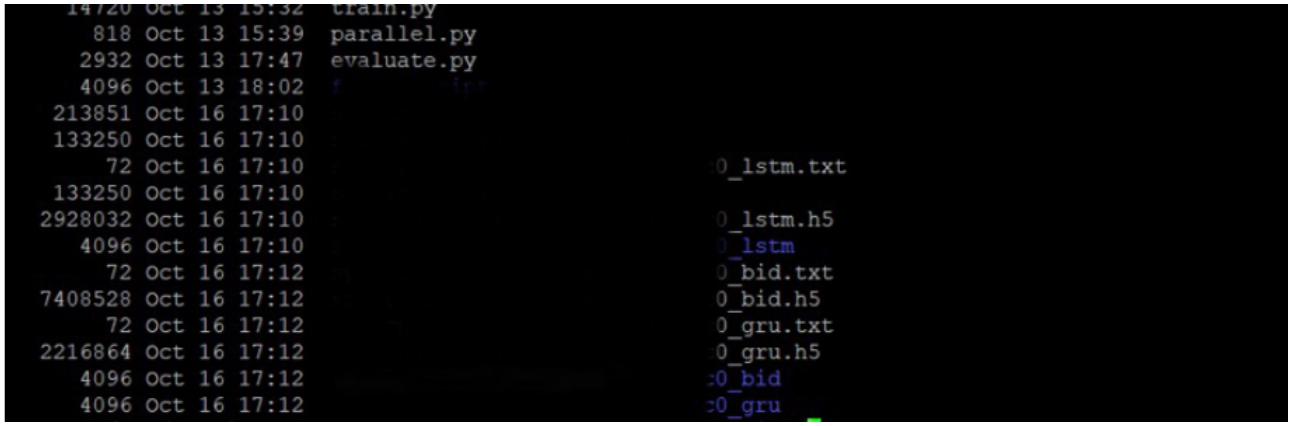


Fig. 5.11 ML models in different file formats

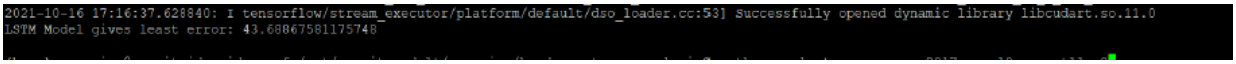


Fig. 5.12 Picking up the model with Least Memory Latency

5.3 Comparison of Measured Latency and Predicted Latency

Latency is the time taken by the main memory to get request serviced as request arrived at the queue is the feature that is predicted. The measured latencies are the actual latency values which are obtained from the memory log dump file, that are generated by the actual simulator, in which the testing is happening with different cache and memory configurations.

Where as the machine learning model used in the project predicts the latency from the input data by looking past behaviour or already learnt data.

As discussed earlier all the SPECCpu suite traces are classified into three types based on memory transaction counts as small, medium and large. For the comparison of measured and predicted latency experiment, the ten medium memory transaction count range traces are picked up and ran the ML script and collected predicted memory latency values form the result of the script and measured latency values from the memory dump file.

5.3.1 Observation from Latency Comparision plots

1. All the memory transactions which have a huge difference in latency values are mostly having read-only enabled.

2. When measured latencies are very high, the ML model struggles to learn those transactions.

3. For a transaction, if all the values are almost in the same range except attribute value WtoR_In, then the difference in latency values changes enormously.

Below are the comparison plots for some random traces from specc benchmark suite: GCC, Nab, Parest and Xalancbmk.

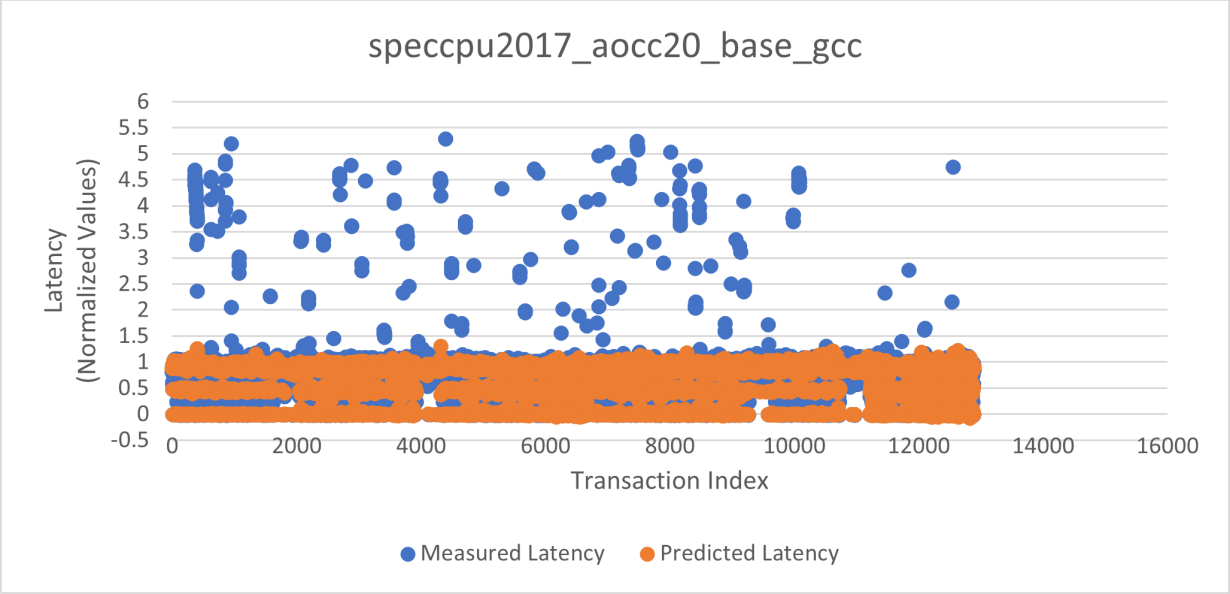


Fig. 5.13 Actual Memory Latency vs Predicted Memory Latency in GCC trace

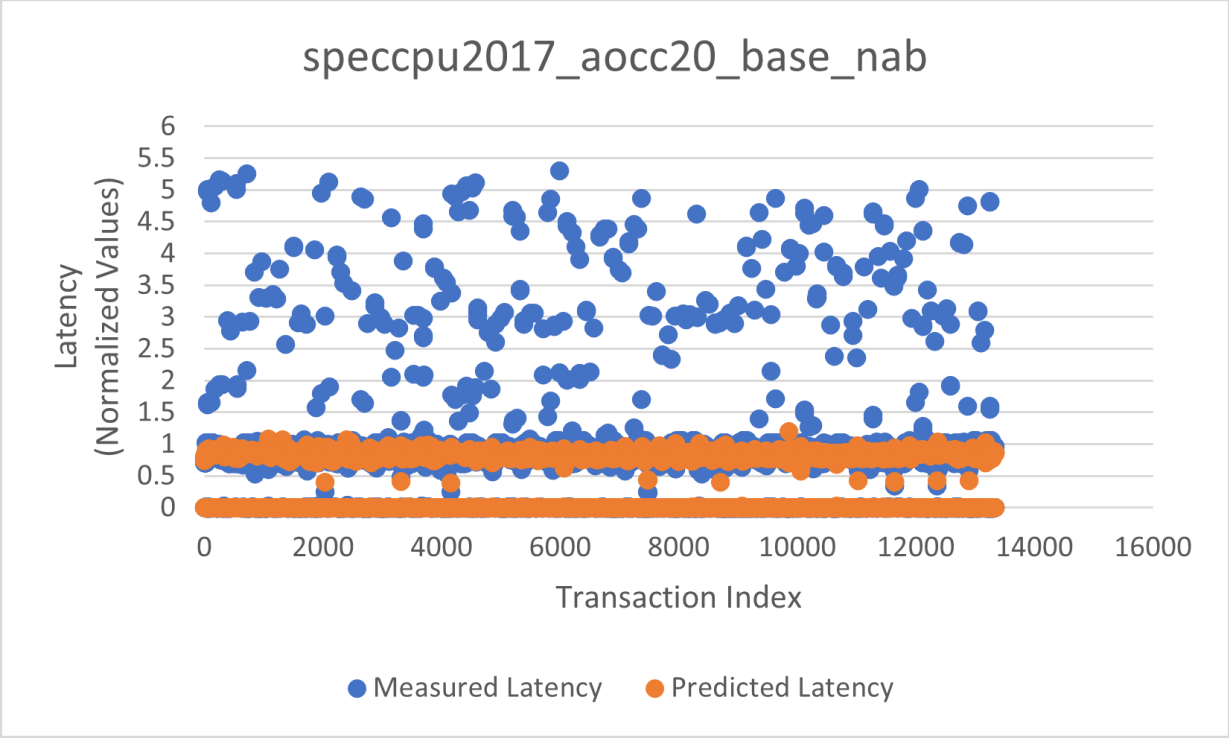


Fig. 5.14 Actual Memory Latency vs Predicted Memory Latency in Nab trace

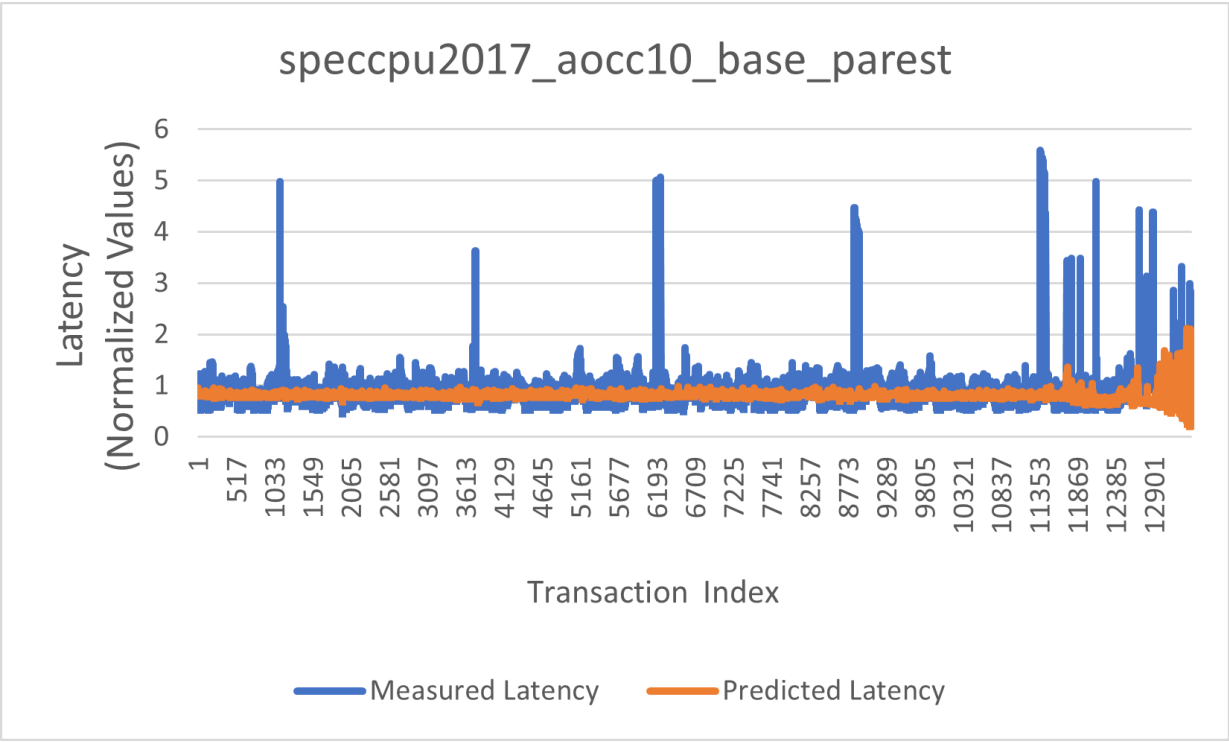


Fig. 5.15 Actual Memory Latency vs Predicted Memory Latency in Parest trace

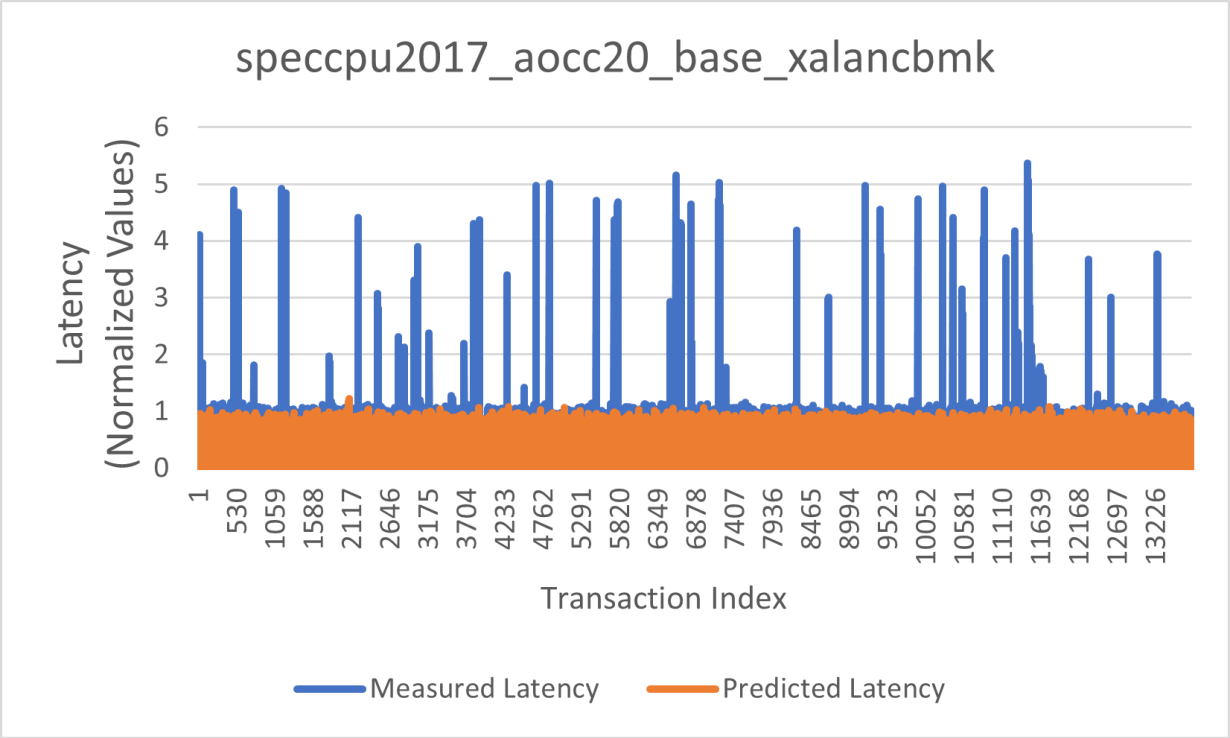


Fig. 5.16 Actual Memory Latency vs Predicted Memory Latency in Xalancbmk trace

5.4 Investigation on Very High Latency Values

From the previous study of the plots its evident that the ML model struggles to predict the higher memory latency values when compared to the measured memory latency of Soc. So now the question is how these high memory latency values are appeared in the Soc? Upon further investigation its found out that these two scenarios might effects the latency values.

1. Disabling Dram refresh in the Soc and
2. The other one is enabling priority feature demand vs prefetch.

As many of the demand requests have higher priorities over prefetchers, the data fabric lets higher priority pass through each of memory components and bypass the low priority with ease, the prefetchers occurs in the higher latencies, So turning off the prefetchers and enabling priority feature demand requests might be beneficial to reduce the peaks.

5.4.1 Bucketizing the Latency Values

As per the above mentioned reasons a study was made to observe the number of transactions in particular latency ranges or the spread of memory latency values by comparing with a Soc model with different configurations such as Soc model with no dram refresh feature and the other is with priority feature enabled.

After enabling the both the features and bucketize the values into seven sets in ranging from 0-100, 101-200, 201-300, 301-400, 401-500, 501-600 and 600 below observations are found from it.

1. Soc model vs Soc dram refresh disabled feature.

When compared the Soc dram refresh disabled feature values with the Soc, Overall the number of transactions count of latency values in the range 301-400 is decreasing and 401-500, 501-600 and above has drastic change in count(i.e average difference is 700).

2. Soc model vs Soc dram refresh disabled no priority feature.

When compared the Soc dram refresh disabled no priority feature values with the Soc similar trend as above was observed here.

3. Soc dram refresh disabled feature vs Soc dram refresh disabled with no priority feature.

Theres no significant difference between above mentioned features in the latency range transaction count but dram refresh disabled feature values are slightly increasing every where.(mostly in the middle region)

Summary is that the memory latency values when compared with Soc its found out that out of two configurations the Soc dram refresh disabled is reducing the higher memory latency values better than the Soc dram refresh disabled with no priority.

Below are the graphs mentioning about the summary.

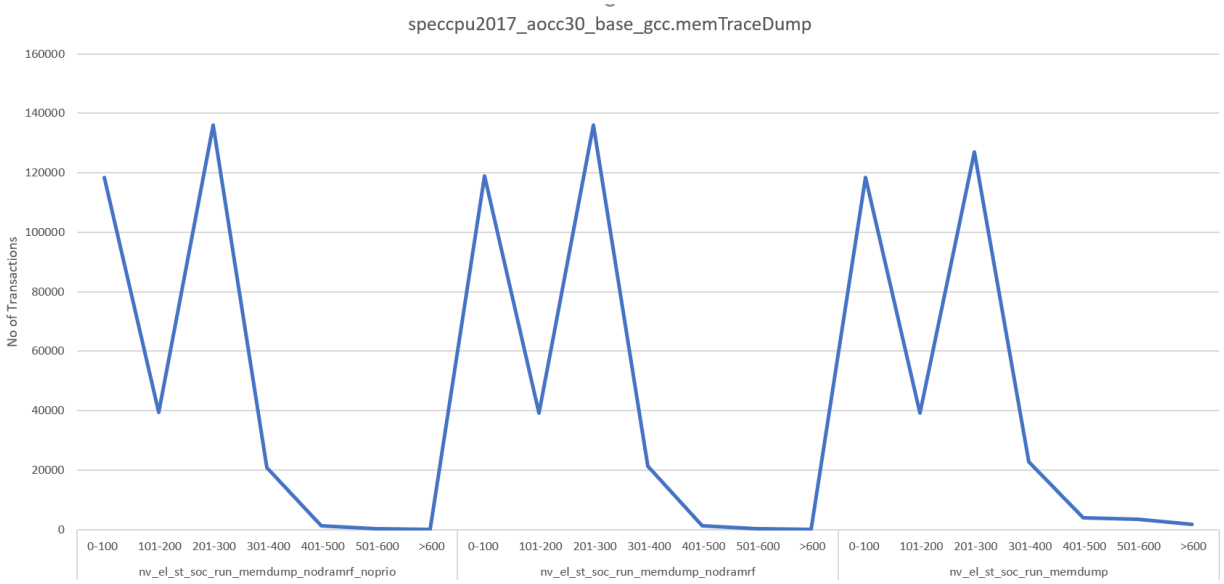


Fig. 5.17 Latency Buckets value comparision for Base GCC trace

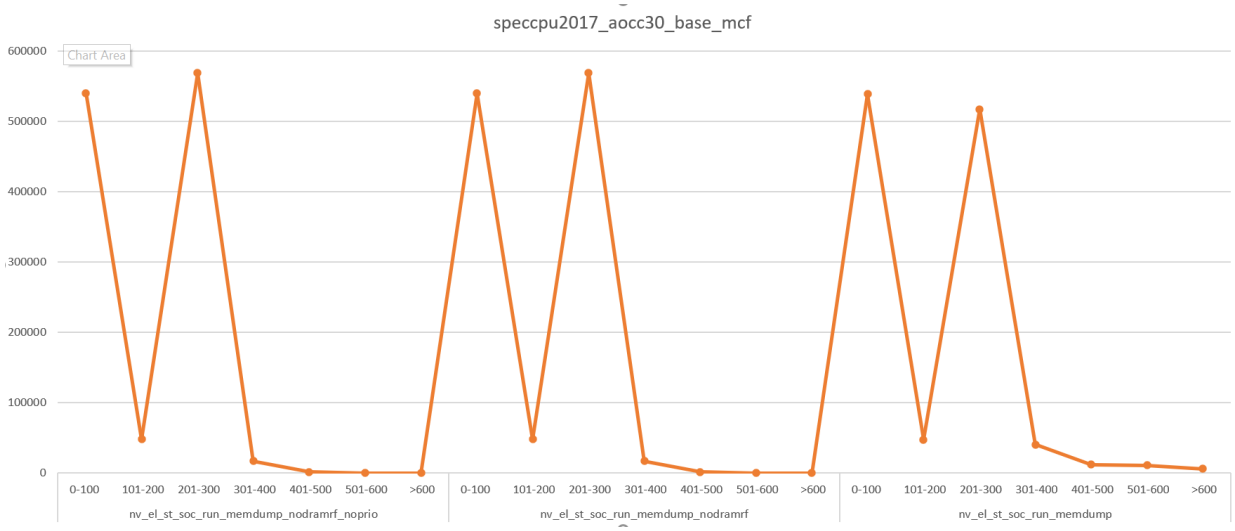


Fig. 5.18 Latency Buckets value comparision for Base MCF trace

5.4.2 Bucketize Python Script

```
import pandas as pd
import matplotlib.pyplot as plt
import os
with open("traces2.txt") as fp:
Lines = fp.readlines()
for line in Lines:
file = line.strip()
if file.endswith(".xz"):
try:
p1 = os.path.join("memdump_nodramrf_noprio/", file)
p2 = os.path.join("memdump_nodramrf/", file)
p3 = os.path.join("soc_run_memdump/", file)
f1 = pd.read_csv(p1, delimiter=r's+')
f2 = pd.read_csv(p2, delimiter=r's+')
f3 = pd.read_csv(p3, delimiter=r's+')
print("Blue: nodramrf_noprio")
print("Orange: nodramrf")
print("Green: memdump")


    except:pass
f_out = open("z2.txt","a")
f = [f1, f2, f3]
f_out.write(file)
for df in f:
c1 = len(df[(df['latency']≥ 0) & (df['latency']≤ 100)])
c2 = len(df[(df['latency']≥ 101) & (ds (df['latency']≤ 300)])
```

c4 = len(df[(df['latency']≥ 301) & (df['latency']≤ 400)])

c5 = len(df[(df['latency']≥ 401) & (df['latency']≤ 500)])

c6 = len(df[(df['latency']≥ 501) & (df['latency']≤ 600)])

c7 = len(df[(df['latency']> 600)])

temp = os.path.join(file + "." + "txt")

f_out.write(","+str(c1)+","+str(c2)+","+str(c3)+","+str(c4)+","+str(c5)+","+str(c6)+","+str(c7)

f_out.close()

5.5 Improving Accuracy by Splitting PA

PA - Physcial Address and NA - Normalized Address. Assumption is that PA, which is more than 30 bits, confuses the ML model, so an experiment is done by splitting the physical address parameter into many parameters like row, bank, column, rank and normalized address. The though process is that splitting the PA reduces the total sample space by the ML model, as all of the address bits might not be useful for the latency prediction.

For checking the ML accuracy percentage, two studies have been made on the both speccpu benchmark suite and also in largefrontend server benchmark suite comparing against soc run with no dram refresh model, soc run with no dram refresh and ten parameter model. One is no_na_pa i.e with no na, no pa, but with other extra address attributes as mentioned above and other one is no_pa i.e with no pa, but with na, row, bank, column, rank. Its been observed that these two studies have achieved almost on average -40% absolute error and -60% rms error(negative is better), where as the other mentioned models was around -2% of both errors on average. But this study is still in initial stage and the results needed to be re-verified as the comparison data between the measured and predicted latency values were not in the expected range and debugging the training script is required.

5.6 Clustering N_Models Optimisations

In the current model for every trace a single model is trained to predict the memory latency values per each transaction, currently a study was made by using clustering mechanism. The main idea behind the clustering study is to use a single representative model for training all traces in cluster and that single model is chosen by picking the largest trace in each cluster and treated as representative trace in that cluster.

All SpecCpu traces are grouped into 10 cluster groups using DBScan and K-means approach by considering following parameters

UIPC, TotalCompletedReads, TotalCompletedWrites, AvgRdLatencyCycles, AvgWrLatencyCycles, AvgCleanWrLatencyCycles, AvgDirtyWrLatencyCycles, TotalCompletedReads+TotalComplete

5.6.1 Clustering Code

DbScan Clustering Code

import seaborn as sns

import sklearn.cluster as cluster

from sklearn.cluster import DBSCAN

matplotlib inline

df = pd.read_csv('zclusters.csv')

df.rename(columns= 'AvgRdLatencyCycles' : 'Rd','AvgWrLatencyCycles' : 'Wr', 'AvgCleanWrLatencyCycles' : 'cln', 'AvgDirtyWrLatencyCycles' : 'Dty',inplace = True)

df.head()

df.describe()

x = df.iloc[:,1:9].values

print(x)

print (x.shape)

db=DBSCAN(eps=3,min_samples=4,metric='euclidean')

```
db=DBSCAN(eps=5,min_samples=5,metric='euclidean')
model=db.fit(x)
label=model.labels_
print(model, label)
print(len(label))
df['Clst'] = label
df.head()
df.describe()
n_clusters=len(set(label))- (1 if -1 in label else 0)
print('No of clusters:',n_clusters)
df['Clst'].value_counts()
df.to_csv('z_temp.csv', index = False)
```

K-Means Clustering Code

```
import pandas as pd
import matplotlib as plt
import numpy as np
import seaborn as sns
import sklearn.cluster as cluster
matplotlib inline
df = pd.readcsv('temp.csv')
df.rename(columns= 'AvgRdLatencyCycles' : 'Rd','AvgWrLatencyCycles' : 'Wr', 'Avg-
CleanWrLatencyCycles' : 'cln', 'AvgDirtyWrLatencyCycles' : 'Dty',inplace = True)
df.head
df.describe()
sns.pairplot(df[[ 'Rd', 'Wr']])
kmeans = cluster.KMeans(nclusters=10, init="k-means++")
```

kmeans = kmeans.fit(df[['Rd', 'Wr']])

kmeans.cluster_centers_

df['Clst'] = kmeans.labels_

df.head()

df['Clst'].value_counts()

df.to_csv('z_temp.csv', index = False)

sns.scatterplot(x='Rd', y='Wr', hue = 'Clst', data = df)

Below is the table that consists the details of traces distribution count in each of the clusters and based on the highest TotalCompletedReads + TotalCompletedWrites count value in each cluster, the representative suite is selected.

A	B	C
largest trace in each cluster	**Cluster_Number**	**No of Traces in the Cluster**
speccpu2017_aocc30_base_cam4	0	102
speccpu2017_aocc30_base_nab_r	1	74
speccpu2017_aocc30_base_foton	2	8
speccpu2017_aocc30_base_cactul	3	6
speccpu2017_aocc30_base_foton	4	13
speccpu2017_aocc30_base_mcf_r	5	174
speccpu2017_aocc30_base_blend	6	12
speccpu2017_aocc30_base_blend	7	43
speccpu2017_aocc30_base_bwav	8	36
speccpu2017_aocc30_base_roms	9	51

Table 5.2 Representative trace in each Cluster

5.6.2 Clustering Results

After experimenting with different K values for K Means Clustering Algorithm, its found out that the k $\bar{1}0$ cluster value performs better correlation than the k values 20 and 50

when compared with the Golden SoC Model. The K cluster value 20's error percentage was worstly correlated than all of the other values and K cluster value 50's error percentage was almost similar to that of cluster K value 10. But one model per one trace is still better than the cluster K value10.

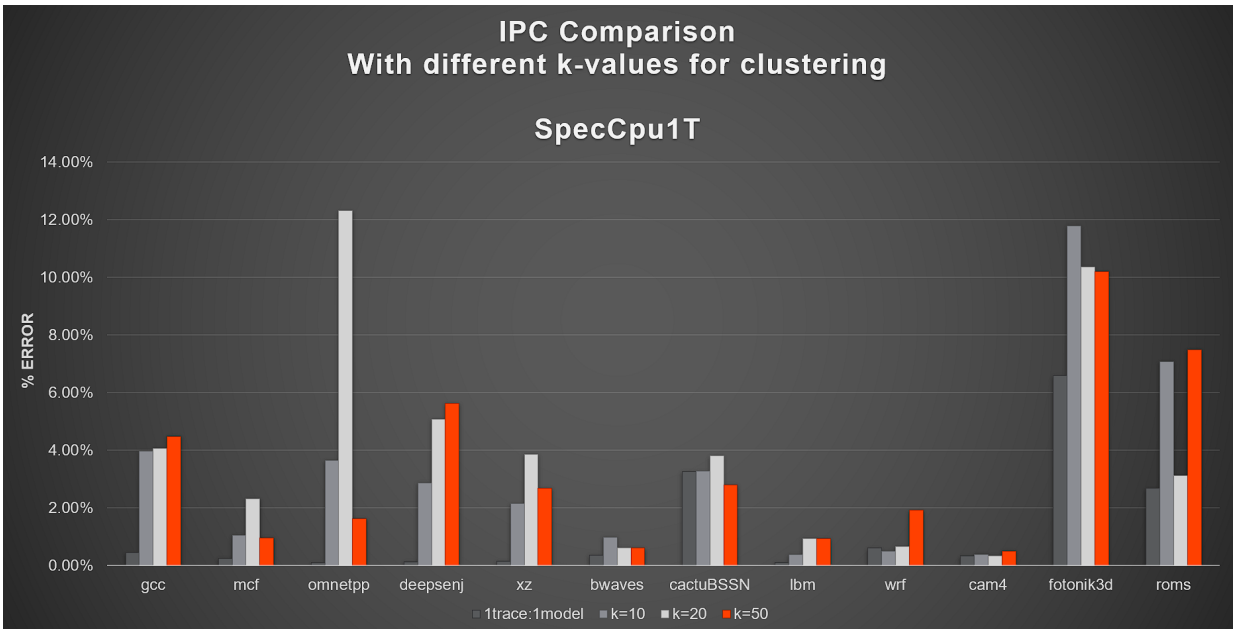


Fig. 5.19 Representative trace in each Cluster

Chapter 6

Conclusion and Future Work

There is an improvement in reducing the error rate by removing the unwanted four previously mentioned parameters.

Bucketizing the memory latency values reveals the regions where more memory transactions are happening.

Comparing of predicted and measured latency reveals important observations.

Splitting of physical address parameter reduces the error percentage.

DBScan clustering algorithm provides optimal number of clusters in speccpu suite.

Applying the K-means clustering on the speccpu benchmark suite effectively reduces the training effort in terms of compute and wall clock time.

The Machine learning part there are few challenges associated with it.

Some of them are first we need to handle cases like simulating with different SoC configurations. in which for each configuration we need to generate one model.

We will also be running the simulation on various number of cores and with the different number of threads per core and again this will generate a huge number of models.

Finally Using the ML infrastructure in other potential areas. This project is Proof of Concept for using ML models integrated and compatible with the simulator, that can be used for future applications.

References

[1] [Online]. Available: https://www.nap.edu/read/12980/chapter/5#64

[2] [Online]. Available: https://en.wikipedia.org/wiki/Analytical_Performance_Modeling

[3] [Online]. Available: https://www.tutorialspoint.com/modelling_and_simulation/modelling_and_simulation_discussion.html

[4] D. H. Bailey1 and A. S. asnavely@cs.ucsd.edu, "Performance modeling: Understanding the present and predicting the future." [Online]. Available: http://www.sdsc.edu/~allans

[5] [Online]. Available: http://www.foremost.co.uk/glossary/system_tuning.html

[6] [Online]. Available: https://en.wikipedia.org/wiki/Benchmark_(computing).html

[7] [Online]. Available: https://en.wikipedia.org/wiki/Standard_Performance_Evaluation_Corporation

[8] [Online]. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[9] [Online]. Available: http://towardsdatascience.com/lstm-and-bidirectional

[10] [Online]. Available: https://www.geeksforgeeks.org/dbscan-clustering-in-ml-density-based-clustering/

[11] [Online]. Available: https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1