

Performance Analysis of AI workload on Intel
hardware platform

Major Project Report

*Submitted in fulfillment of the requirements
for the degree of*

**Master of Technology
in
Electronics & Communication Engineering
(Embedded Systems)**

By

Malhar Bhatt

(17MECE02)



**Electronics & Communication
Engineering Department
Institute of Technology
Nirma University
Ahmedabad-382 481
May-2019**

Performance Analysis of AI workload on Intel hardware platform

Major Project Report

*Submitted in partial fulfillment of the requirements
for the degree of*

Master of Technology

in

Electronics & Communication Engineering

By

**Malhar Bhatt
(17MECE02)**

Under the guidance of

External Project Guide:

Mrs. Juby Jose
Engineering Manager
Intel Technologies Pvt. Ltd.,
Bengaluru

Internal Project Guide:

Dr. Nagendra Gajjar
Professor & PG Coordinator , Embedded Systems,
Institute of Technology,
Nirma University, Ahmedabad.



Electronics & Communication Engineering Department
Institute of Technology, Nirma University
Ahmedabad-382 481
May-2019

Declaration

This is to certify that

- a. The thesis comprises my original work towards the degree of Master of Technology in Embedded Systems at Nirma University and has not been submitted elsewhere for a degree.
- b. Due acknowledgment has been made in the text to all other material used.
- c. Complete document with its contents and its results has been verified and reviewed from the Intel.

- **Malhar Bhatt**
17MECE02

Disclaimer

“The content of this thesis does not represent the technology, opinions, beliefs, or positions of Intel Technology India Pvt. Ltd., its employees, vendors, customers, or associates.”



Certificate

This is to certify that the Major Project entitled “**Performance Analysis of AI workload on Intel hardware platform**” submitted by **Malhar Bhatt (17MECE02)**, towards the partial fulfillment of the requirements for the degree of Master of Technology in Embedded Systems, Nirma University, Ahmedabad is the record of work carried out by him under our supervision and guidance. In our opinion, the submitted work has reached a level required for being accepted for examination. The results embodied in this major project, to the best of our knowledge, haven’t been submitted to any other university or institution for award of any degree or diploma.

Date:

Place: Ahmedabad

Dr. N. P. Gajjar
Internal Guide

Dr. N. P. Gajjar
Program Coordinator

Dr. D. K. Kothari
Head, EC Department

Dr. Alka Mahajan
Director, ITNU

Certificate

This is to certify that the Major Project entitled “**Performance Analysis of AI Workload on Intel hardware platform**” submitted by **Malhar Bhatt (17MECE02)**, towards the submission of the project for the requirements for the degree of Master of Technology in Embedded Systems, Nirma University, Ahmedabad is the record of work carried out by him under my supervision and guidance at **Intel Technology India Pvt. Ltd.** In my opinion, the submitted work has reached a level required for being accepted for examination.

External Guide

Mrs. Juby Jose

Engineering Manager

Intel Technology India Pvt. Ltd.

Bengaluru

Company Seal

Intel Technology India Pvt. Ltd. (Bangalore)

Date:

Place: Bengaluru

Acknowledgements

I take this opportunity to express my profound gratitude and regards to my internship project guide **Dr. Nagendra Gajjar** for his exemplary guidance, monitoring and constant encouragement.

I express my gratitude to the staff of **Intel Technologies Pvt. Ltd., Bengaluru** for providing most encouraging working environment along with providing their valuable time and effort during implementation of this project.

I would like to express sincere gratitude to manager **Mrs. Juby Jose**, Engineering Manager, Intel India Pvt. Ltd., for giving me an opportunity to work for Intel. I would also like to thank mentor **Mr. Pravin Chandra**, Software Engineer, Intel India Pvt. Ltd. for his encouragement and guidance. I would also like to cordially thank **Mr. Raghavendra Bhat**, Principal Engineer, Intel India Pvt. Ltd. for his valuable advices and support throughout the project. I thank **Mr. Ajith Prakash** and **Mr. Avinash Chakravarty**, Intel India Pvt. Ltd. for his great support in setting up the setup and helping me whenever it required.

I thank my Parents, faculty members and colleagues for their constant support and encouragement during this project work.

- Malhar Bhatt
17MECE02

Abstract

The increased popularity of Deep Neural Network (DNN) has led to a good amount of research in getting the best hardware configuration to achieve the optimum throughput from server processors running an end to end DNN topology. The primary focus here is to do the inference which is to know how efficiently the processor can execute the already trained models for the Text To Speech (TTS) topology. This thesis explores the benchmarking performed on Intel's server platform with DeepMind's Tacotron-2 topology as a workload. Benchmarking consists of running the topology using baseline and Intel optimized Tensorflow framework. The timeline analysis and system performance analysis helps in understanding the behavior of the topology on the server platform. The processor's system performance is measured using Intel's VTune amplifier tool. In order to understand the impact of workload on the server processor, this thesis focuses on the key parameters like Processing time, Memory bandwidth utilization and Memory latency, Hotspot analysis, and effective Central Processing Unit (CPU) utilization. Along with the above parameters, timeline and baseline profiling on the workload is also done. As an outcome of this benchmarking, several observations and recommendations are proposed for further research and improvements that can be done in the area of Silicon architecture, Software frameworks, and Kernels. The results highlight the key observations and the performance graphs and comparison using baseline and Intel Optimized framework for the tacotron module.

Keywords: Tensorflow, DNN, TTS, Benchmarking, CPU utilization

Abbreviation Notation and Nomenclature

AI Artificial Intelligence

API Application Programming Interface

BIN Binary

CNN Convolutional Neural Network

CPU Central Processing Unit

CSV Comma Separated Values

DLDT Deep Learning Deployment Toolkit

DNN Deep Neural Network

FPGA Field-Programmable Gate Array

FPU Floating Point Units

GUI Graphical User Interface

GPU Graphics Processing Unit

HPC High Performance Computing

I/O Input/Output

JSON JavaScript Object Notation

MOS Mean Opinion Score

MPI Message Passing Interface

NUMA Non-uniform memory access

OpenVino Open Visual Inferencing and Neural Network Optimization

PCIe Peripheral Component Interconnect Express

PCM Performance Counter Monitor

PMU Performance Monitoring Units

QPI QuickPath Interconnect

SPEC Standard Performance Evaluation Corporation

SMT Simultaneous Multithreading

Performance Analysis of AI workload on Intel hardware platform

SOC System On Chip

TF Tensorflow

TTS Text To Speech

XML Extensible Markup Language

List of Figures

1.1	Project Work-Flow Phase-1	3
3.1	A detailed look at Tacotron 2’s model architecture	7
4.1	Chrome Tracing Example code	13
4.2	Chrome Tracing time profiling for above example	14
4.3	The Hotspots analysis with Intel VTune Amplifier provides two sampling-based collection modes [10]	16
4.4	The high-level performance metrics in the Micro-architecture Exploration viewpoint	17
4.5	Summary Tab in General Exploration	18
4.6	Bottom-up Tab in General Exploration	19
4.7	Image showing back-end issues using function / call stack	19
4.8	An overview of a modern multi-processor, multi-core system [14]	20
4.9	Example for PCM.exe command	22
4.10	Example for pcm-core.exe command	23
4.11	Example for pcm-memory.exe command	24
4.12	Example for pcm-numa.exe command	25
4.13	Example for pcm-pcie.exe command	26
4.14	Block Diagram for OpenVINO integration process	28
5.1	Tensorboard Topology Diagram for TTS Tacotron2	31
5.2	Hotspot in tacotron	32
5.3	Wavenet visualization in tacotron	33
5.4	Comparison between lower case and upper case text	34
5.5	Random data vs Corpus data for Baseline Tensorflow (TF)	35
5.6	Random data vs Corpus data for Intel optimized TF	35
5.7	Time comparison between Intel optimized and Baseline tensorflow for batch	36
5.8	Baseline Bandwidth Utilization Histogram	36
5.9	Optimized Bandwidth Utilization Histogram	37
5.10	Optimized Effective Physical Core Utilization	37
5.11	Baseline Effective Physical Core Utilization	37
5.12	Hotspot analysis for Baseline tensorflow	37

Performance Analysis of AI workload on Intel hardware platform

5.13 Hotspot analysis for Optimized tensorflow	38
5.14 Micro Architecture analysis comparison	38

Contents

Declaration	iii
Disclaimer	iv
Certificate	v
Acknowledgements	vii
Abstract	viii
Abbreviation Notation and Nomenclature	ix
List of Figures	xi
1 Introduction	1
1.1 Purpose	1
1.2 Motivation	1
1.3 Objective	2
1.4 Requirements	2
1.5 Scope of Work	2
1.6 Gantt Charts	3
1.6.1 Project Work-Flow Phase-1	3
1.7 Thesis Outline	3
2 Literature Survey	4
3 Speech Workload Performance Characterization	6
3.1 Tacotron 2	6
3.1.1 About the Topology	6
3.2 System Configuration	8
3.2.1 Hardware Configuration	8
3.2.2 Software Configuration	9
3.2.3 Dataset / Corpus	9
3.2.4 Implementation used	9

3.3	Benchmarking methodology for AI topology	10
4	Tools used	12
4.1	Chrome Tracing	12
4.1.1	Introduction	12
4.1.2	How to get it ?	12
4.1.3	Usage	12
4.1.4	Demo	13
4.2	Intel VTune Amplifier	14
4.2.1	Introduction	14
4.2.2	How to get it ?	15
4.2.3	Usage	15
4.2.4	Demo	15
4.3	Intel Performance Counter Monitor	20
4.3.1	Introduction	20
4.3.2	Usage	20
4.3.3	Demo	21
4.4	OPENVINO Toolkit	27
4.4.1	Introduction	27
4.4.2	OpenVINO as Inference	28
4.4.3	Steps to Create a custom plugin	29
5	Results, Conclusion and Future Work	30
5.1	Results	30
5.2	Conclusion	39
5.3	Future Work	41
	References	42

Chapter 1

Introduction

1.1 Purpose

One of the methods used to analyze performance of the processor is benchmarking. In the data center scenario, the workloads are not similar to the workloads seen in the desktop processors. The benchmarking done on the existing systems helps us in pointing out the limitations and allow us to make suggestions for future accelerated or general purpose inference hardware. The workloads should provide a sufficient and accurate description of the programs of interest, for the resulting instruction streams. It is challenging in nature to prove that the selected workload is representative as the amount of information which is known to us is limited. For such scenarios where the information or the resources are limited we can start with a narrowing down the range of application of interest and start with small benchmarking program.

In this thesis, two stages of benchmarking is carried out : Time-line profiling of the workload and System level performance analysis for the workload. Timeline profiling provides the information on the time taken by the process to run as an inference on the server and the system level performance provides the information of the core and memory computational analysis.

1.2 Motivation

Artificial Intelligence (AI) Workload analysis plays critical role in understanding the problems in design and development of server systems. This helps the architects to predict the system behaviour of the upcoming systems or System On Chip (SOC)s.

It is important to understand the target market, predict the future devices (SOCs , Field-Programmable Gate Array (FPGA) accelerators) and the associated workload, which the market is expecting.

1.3 Objective

The main objectives of the project are as follows:

- To benchmark an implementation of AI workload on end to end speech synthesis directly from text (through its inference) on Intel hardware and compare it against other competitive hardware platforms such as those based on Graphics Processing Unit (GPU)s.
- After performance analysis and finding hot-spots, provide recommendations in Intel software and hardware domains.

1.4 Requirements

The development and implementation of this project requires following:

- Knowledge of workload/ topology
- Understanding of processor performance metrics
- Knowledge of profiling tools
- Shell scripting
- Python scripting

1.5 Scope of Work

The main objective is to characterize and analyze the speech AI workload proxies, benchmark it and come up with the recommendations for core architects and help them in defining new SOCs.

1.6 Gantt Charts

1.6.1 Project Work-Flow Phase-1

Project Timeline		2018	2018	2018	2018	2018	2018	2018	2019	2019
Project Stages	Tasks	June to August		August to September		October to December			January to March	April - May
Intro	Basic Training and Webbased Sessions	Trainings								
0	Literature Study, Choice of Algorithm and Implementation, Tools Familiarization, Bench Setup			Stage 0						
1	Performance Benchmarking				Stage 1					
2	Topology Benchmarking					Stage 2				
3	System Benchmarking						Stage 3			
4	Recommendations							Stage 4		
5	OpenVINO Exploration and Plugin Creation								Stage 5	
6	Documentation and Report Generation									Stage 6
		Completed		In Progress		Yet to Start				

Figure 1.1: Project Work-Flow Phase-1

1.7 Thesis Outline

- **Chapter-1** contains the brief information about motivation and objective of the project along with the Gantt charts indicating project development work flow.
- **Chapter-2** describes the literature survey by providing the overview of workload characterization.
- **Chapter-3** discusses about the speech workload topology used for benchmarking and the methodology used for benchmarking
- **Chapter-4** discusses the tools used for TTS Tacotron2 topology benchmarking
- **Chapter-5** project results and concludes the project report and discusses the possible future work.

Chapter 2

Literature Survey

The deep learning techniques for machine learning is related to three main complementary trends namely the progress in new algorithms, the availability of big amounts of labeled data and the increasing computational power. As now the data for machine learning is increasing, improving one of these areas usually demands improvements in the others. There is a huge computational demand for existing DNN platforms. Efforts have been increased significantly in the fields of computing and storage according to the needs of DNN models with improvements in performance compared to conventional computing systems.

In 2005, Standard Performance Evaluation Corporation (SPEC) issued set of benchmarks evaluating CPU, main memory, disk subsystems, network cards including web servers and operating systems[29]. But this could not fulfill the requirement of benchmarking AI workload at inference. In the year 2017, people from google and stanford university came up with the first DNN benchmarks which focussed on end-to-end performance called DAWNbench [30]. It provided an objective of normalizing across differences in the areas of computer hardware, computation frameworks, algorithms and its optimizations, different hyperparameter settings, and other real-world performance affecting factors. In 2018, UC Berkeley together with Baidu, Google, Harvard University, and Stanford University came up with new benchmarking suite for machine learning called MLPerf. MLPerf aims to provide complete system level measures for both training and inference. MLPerf provides the benchmarking suite[31] for the areas such as Image classification, Object detection, Speech to text, Machine Translation, Sentiment Analysis and Reinforcement Learning. Along with MLPerf facebook too came up with its own deep learning inference benchmarks [32] for its own data centers.

The TTS Tacotron2 workload is characterized in terms of Processing time for single and batch streams of data, Memory bandwidth utilization and Memory latency, Hotspot analysis, and an effective CPU Utilization, Core Scaling, Memory Bandwidth. It is necessary to understand CPU states in order to understand the behavior of any workload on any platform. CPU might get fully utilized even at maximum frequency due to the high intensive workload. Similarly, a low profile

Performance Analysis of AI workload on Intel hardware platform

workload utilizes lower CPU frequency for a less fraction of time. This will also vary energy efficiency of the system and performance of the workload.

Here the workload is repeatedly tested multiple alternatives under same conditions.

Chapter 3

Speech Workload Performance Characterization

3.1 Tacotron 2

3.1.1 About the Topology

In this section, we will cover the detailed architecture of the Google's tacotron2 TTS workload and explicitly show how the model works for speech synthesis [1] (as inference).

The TTS Tacotron 2 network is a combination of wavenet [2] and Google's previous speech project Tacotron.

It consists of sequence-to-sequence [3] model with attention layer. Encoder, Decoder and Attention mechanism are customized compared to a basic sequence-to-sequence architecture composed using only recurrent layers.

WaveNet provides a generative model of time domain waveform. It produces natural sounding audio fidelity. However, the inputs to WaveNet need a significant domain expertise to produce a human like voice as they require elaborate text-analysis systems and a detailed pronunciation guide.

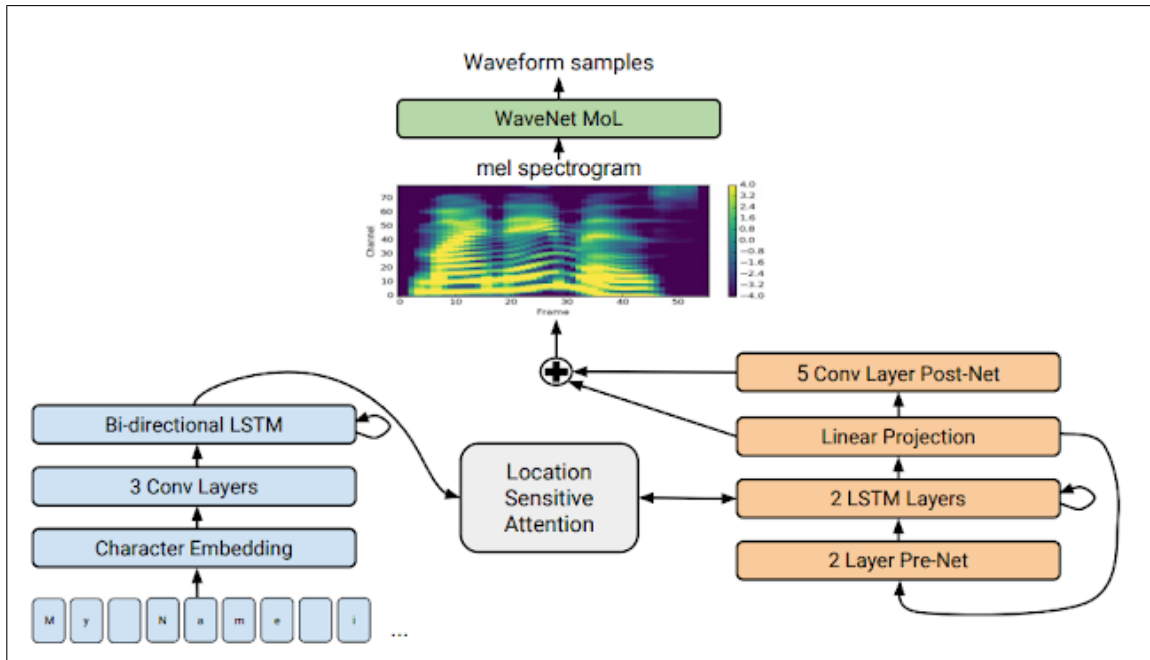


Figure 3.1: A detailed look at Tacotron 2's model architecture

A recurrent sequence-to-sequence [3] architecture is used for producing magnitude spectrograms from a sequence of characters i.e. it synthesizes speech directly from words. A single neural network trained from data alone for production of the linguistic and acoustic features. Tacotron uses the Griffin-Lim algorithm for phase estimation.

In comparison to Tacotron, Tacotron 2 uses simpler building blocks i.e. vanilla LSTM and convolutional layers in the encoder and decoder. Each decoder step uses a single spectrogram frame.

Summarizing the architecture, Tacotron 2 uses a sequence-to-sequence model optimized for TTS in order to map a sequence of letters to a sequence of features that encode the audio.

These sequence of features include an 80-dimensional audio spectrogram with frames computed every 12.5 milliseconds. They are used for capturing word pronunciations, and various other qualities of human speech such as volume, speed and pitch. Finally, these features are converted to a waveform of 24 kHz using a WaveNet-like architecture. Tacotron 2 system can be trained directly from data without relying on complex feature engineering. It achieves state-of-the-art sound quality close to that of natural human speech. Their model achieves a Mean Opinion Score (MOS) of 4.53 [4] comparable to a MOS of 4.58 [4] for professionally recorded speech. Google has also provided Tacotron 2 audio samples that demonstrate the results of their TTS system.

3.2 System Configuration

3.2.1 Hardware Configuration

Table 3.1: Hardware Configuration for Test

Sr. No	Title	Configuration
1	Architecture	x86 ₆₄
2	CPU op-mode(s)	32-bit
3	Byte Order	Little Endian
4	CPU(s)	96
5	On-line CPU	0-95
6	Thread(s) per core	2
7	Core(s) per socket	2
8	Socket(s)	2
9	NUMA node(s)	2
10	Vendor ID	GenuineIntel
11	CPU family	Skylake 6
12	Model name	Intel(R) Xeon(R) Platinum 8160 CPU @ 2.10GHz
13	CPU max MHz	3700
14	CPU min MHz	1000
15	BogoMIPS	4191.54
16	Virtualization	VT-x
17	L1d cache	32K
18	L1i cache	32K
19	L2 cache	1024K
20	L3 cache	33792K

3.2.2 Software Configuration

- Ubuntu 16.04.4 LTS
- Kernel: 4.4.0-97-generic
- Python 3.5
- Intel Optimized Tensorflow v1.9
- Source: https://storage.googleapis.com/intel-optimized-tensorflow/tensorflow-1.9.0-cp35-cp35m-linux_x86_64.whl
- Numpy v1.15.0

3.2.3 Dataset / Corpus

- **LJ Speech** : It is a publicly available (open-source) speech dataset consisting of 13100 [5] short audio clips of single speaker. These audio clips are the recordings for the speaker reading non-fiction books. Each clip varies in the length of 1 to 10 seconds. A total length of 24 hours audio is recorded in this dataset.

3.2.4 Implementation used

To do the benchmarking on TTS Tacotron 2, we have used DeepMind's Tacotron-2 Tensorflow implementation done by Rayhane-mamah available on the GitHub. More details about the implementation -

- **URL:** <https://github.com/Rayhane-mamah/Tacotron-2>
- **Author:** Rayhane-mamah
- **Number of Commits:** 149
- **Contributors:** 14

3.3 Benchmarking methodology for AI topology

For benchmarking TTS topology we will be following the below mentioned steps:

- a. **Performance** : The first benchmark we carry out is to do the time-line profiling. The time-line profiling gives us the amount of time the application is taking to run on the targeted hardware. Different single and batch data is provided and a matrix is created to understand the time consumption of the application.

Here we use system time and the python's timing Application Programming Interface (API).

- b. **Topology** : After the timeline profiling, the next part is understanding the portion in the topology consuming more time. This analysis helps us in moving forward into benchmarking the most core and memory compute area's of the topology.

With the help of Google's chrome tracing and tensor flow's tensorboard visualizer we do topology benchmarking.

- c. **System** : After analyzing time taken and the topology of the workload, system benchmarking is done.

In this section, various analysis on topology is done.

System benchmarking can again be subdivided into 3 parts namely -

- (1) **Finding Hotspots** : This is done for analyzing the implemented algorithm's call paths. Analyzing call paths to find where the code is spending the most time and discover opportunities for tuning the algorithms. Advanced Hotspots helps us for analyzing an application or the entire system and getting the kernel information with higher resolution and shorter sampling intervals.
- (2) **Parallelism** : Parallelism helps us to understand whether algorithm wants to access the compute efficiency of the multi-threaded application. Different checks like Concurrency, Locks and waits and High Performance Computing (HPC) Characterization.

Concurrency is finding areas with high or low concurrency, and identifying serial bottlenecks in the code.

Locks and waits is best for locating causes of low concurrency, such as heavily used locks and large critical sections.

HPC Performance characterization is best for understanding how the compute-intensive OPENMP or Message Passing Interface (MPI) app is using the CPU, memory and Floating Point Units (FPU) resources.

- (3) **Micro-architecture:** Here we look into the micro-architecture of the hardware for identifying the CPU pipeline stage (front-end and back-end) and hardware units responsible for the hardware bottlenecks.

Along with that we check for the memory bounds to determine which level of memory hierarchy is impacting the performance by reviewing the CPU cache and main memory usage, including possible Non-uniform memory access (NUMA) issues.

d. **Platform Analysis:** In this analysis we check -

- **Disk Input and Output:** Best for monitoring usage of the disk subsystem, CPU, and processor buses.
- **System Overview :** Best for understanding how interrupts (IRQ), frequency, threads, and sleep states impact system performance.

Chapter 4

Tools used

This chapter describes various tools used during this project and some information on its usage.

4.1 Chrome Tracing

4.1.1 Introduction

When diagnosing performance problems it can be valuable to see which processes or functions are consuming more time to compute. Google chrome timeline feature API helps us in getting the complete time-line for the given application.

The activities are recorded in chrome's processes, it records C++, JavaScript and Python [5] method signatures. Chrome tracing tool helps user to identify performance bottlenecks, events and slow operations.

4.1.2 How to get it ?

No special tool is required to download, user just have to include few lines of code in the existing python script file. Details are shown in the usage topic [topic 4.1.4]

4.1.3 Usage

Its very simple to use, user needs to use chrome tracing view is -

- Produce a JSON [5] file with the format expected by Chrome.
- Navigate to chrome://tracing in Chrome browser
- Click Load button and open your file explorer, or alternatively drag the file into Chrome.

4.1.4 Demo

The below snapshot shows a sample example of how to insert the required libraries in the tensorflow code.

NOTE: This sample code is taken as a reference to explain the chrome tracing usage from Illarion [7], Git Repository.

```
import tensorflow as tf
from tensorflow.python.client import timeline

a = tf.random_normal([2000, 5000])
b = tf.random_normal([5000, 1000])
res = tf.matmul(a, b)

with tf.Session() as sess:
    # add additional options to trace the session execution
    options = tf.RunOptions(trace_level=tf.RunOptions.FULL_TRACE)
    run_metadata = tf.RunMetadata()
    sess.run(res, options=options, run_metadata=run_metadata)

    # Create the Timeline object, and write it to a json file
    fetched_timeline = timeline.Timeline(run_metadata.step_stats)
    chrome_trace = fetched_timeline.generate_chrome_trace_format()
    with open('timeline_01.json', 'w') as f:
        f.write(chrome_trace)
```

Figure 4.1: Chrome Tracing Example code

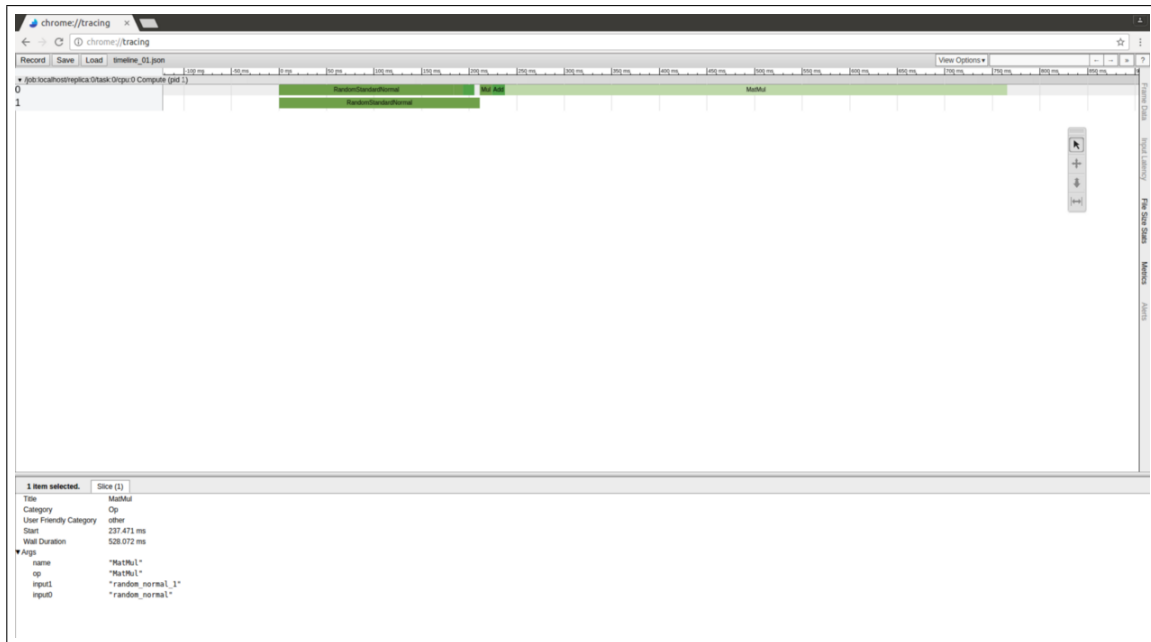


Figure 4.2: Chrome Tracing time profiling for above example

4.2 Intel VTune Amplifier

4.2.1 Introduction

Intel's VTune Amplifier [10] is source-code profiling software. It is popular in the HPC community for its highly versatile accurate sampling capacity along with its low collection overhead. It includes features such as Software stack sampling, thread profiling, and low-level hardware event sampling. Intel VTune comes with command line interface, and a mature and convenient graphical user interface. It is easy to use for a user, who can just mouse-around and effectively get into the code and map bottlenecks to specific lines in the user's source code.

Intel VTune Amplifier is used to locate or determine the following:

- The most time-consuming (hot) functions in your application.
- Identify sections of code that do not effectively utilize available processor.
- Sections of code to optimize for sequential performance and for threaded performance
- Time consumption on input/output operations
- The performance impact of different synchronization methods, different numbers of threads, or different algorithms

- Thread activities and transitions
- Hardware-related issues in your code such as data sharing, cache misses, branch miss-prediction, and others

4.2.2 How to get it ?

Intel's VTune Amplifier is available in different software suits through `software.intel.com`. It is now integrated with the following tools from Intel, Intel Parallel Studio XE [8], Intel System Studio[9], and Intel Media Server Studio[10].

4.2.3 Usage

User can use this tool to analyze performance using following configurations -

- Hotspots analysis
- Micro-architecture analysis
- Platform analysis

User may choose to run any of the available analysis types either from the Graphical User Interface (GUI) (`amplxe-gui`) or using command line interface (`amplxe-cl`).

4.2.4 Demo

In this project course, we have explored the two configurations namely Hotspots and Micro-architecture analysis.

Hotspots Analysis

The Hotspots analysis configuration targets software tuning and helps user to understand where the application spends time and analyze the efficiency of the algorithms.

Hotspots analysis is used to understand an application flow and identify sections of code that get a lot of execution time (Hotspots). This is a starting point for your algorithm analysis.

The Hardware event based sampling mode is based on the hardware event-based sampling collection and analyzes all the processes running on your system at that instance, providing CPU time data on whole system performance.

Below image shows the screen-shot for the Hotspots analysis module.

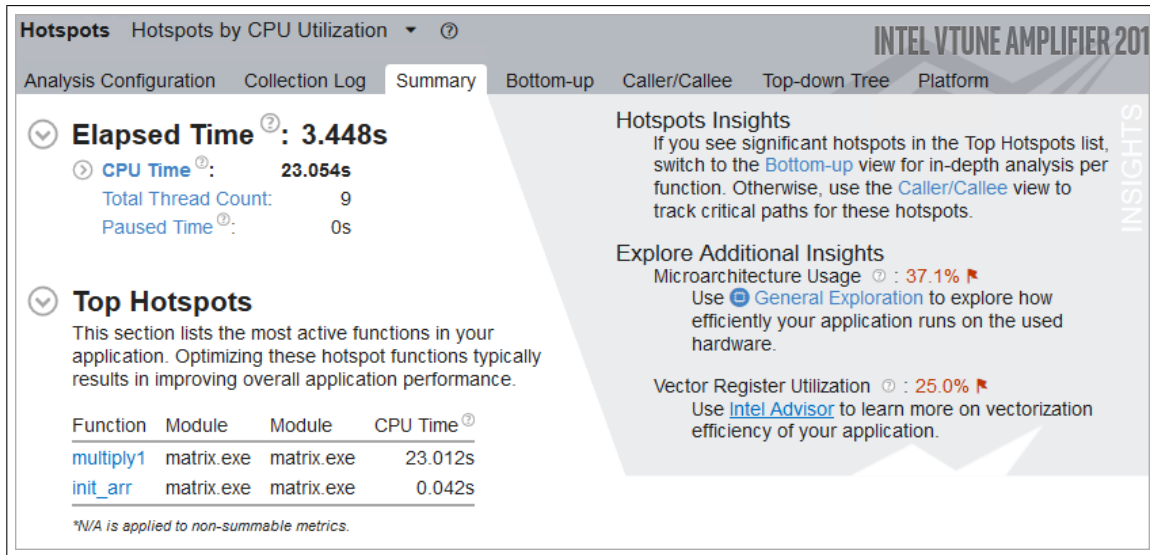


Figure 4.3: The Hotspots analysis with Intel VTune Amplifier provides two sampling-based collection modes [10]

Micro-architecture Exploration View

When Micro-architecture exploration analysis (commonly known as General Exploration) is completed, the VTune amplifier opens the event-based metrics in a hierarchical viewpoint depending on the hardware architecture. Below figure shows the top-down characterization of the application. Each pipeline slot is classified into exactly one of these four categories.

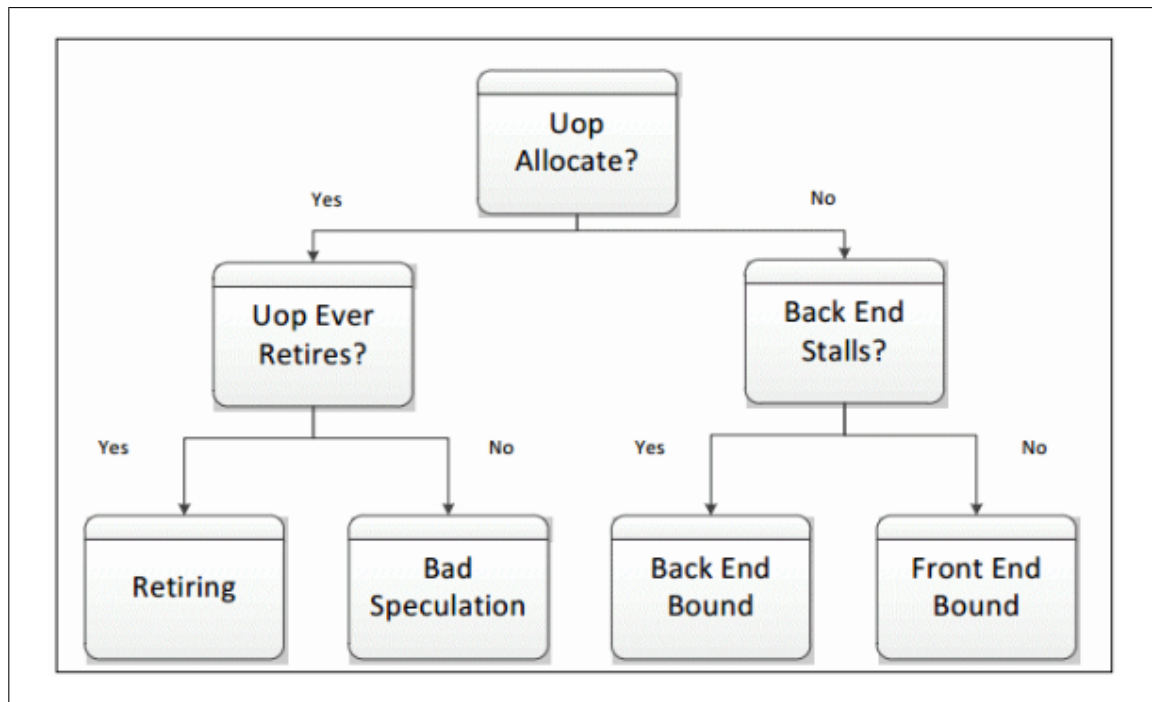


Figure 4.4: The high-level performance metrics in the Micro-architecture Exploration viewpoint

The summary window of micro-architecture exploration tab provides the following application level statistics -

- Micro-architecture metric diagram
- Analysis metrics
- CPU Utilization Histogram
- Collection and Platform Info

Below snapshots provides information on the performance data captured during the hardware event-based sampling analysis [?, r10]y following the steps below: **NOTE:** The data shown in the images are just for reference purpose, doesn't indicate the actual data of the workload.

- a. **Learn metrics and define a performance baseline.** Summary tab displays the summary statistics on the overall application execution per hardware-related metrics.

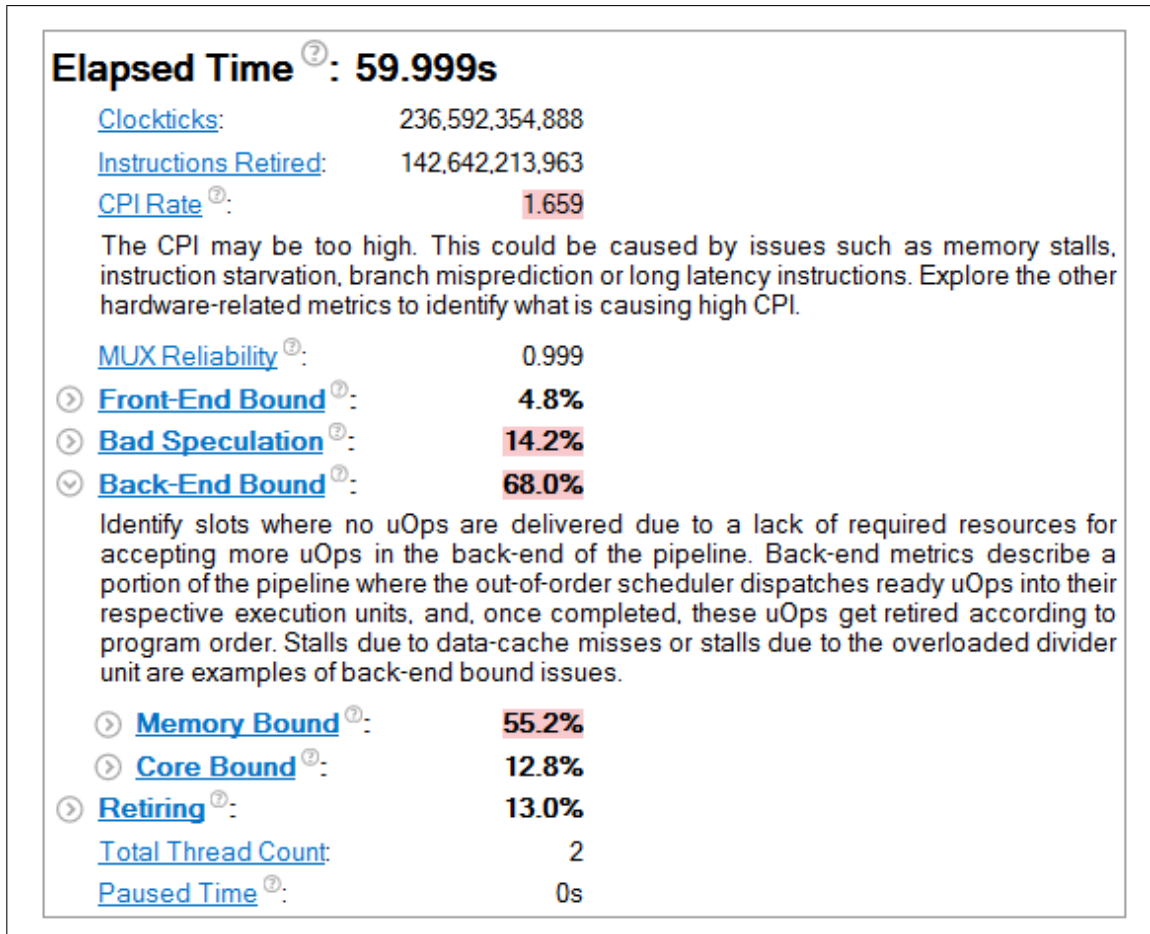


Figure 4.5: Summary Tab in General Exploration

- b. **Identify hardware issues.** Here in this example each row represents a program unit and percentage of time used by this unit. Program units that take more than 5% of the CPU time are considered as hotspots. The VTune Amplifier sorts the data in the descending order by Clock-ticks by default and provides the hotspots at the top of the list.

Performance Analysis of AI workload on Intel hardware platform

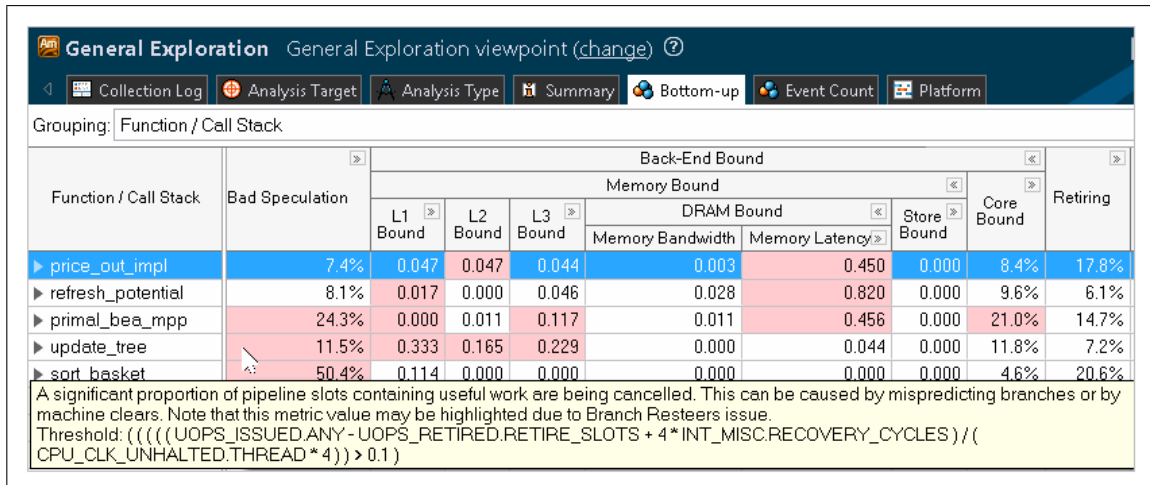


Figure 4.6: Bottom-up Tab in General Exploration

- c. **Analyze source.** To check the a critical function (hotspot), by double-clicking to open the Source/Assembly window, user can analyze the source code. It basically redirects user to the source code and Source/Assembly window displays event data.

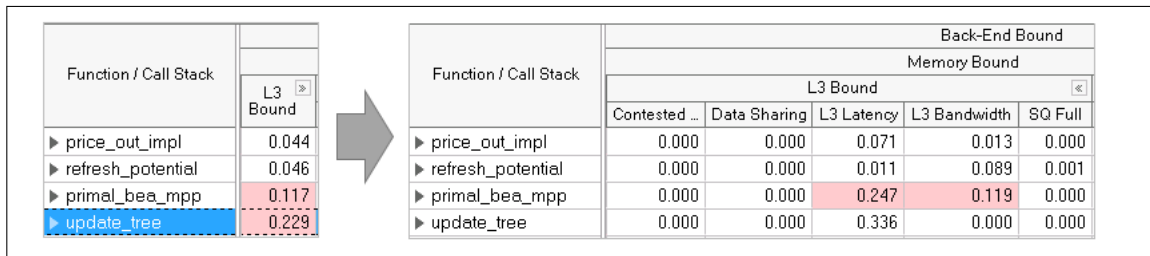


Figure 4.7: Image showing back-end issues using function / call stack

4.3 Intel Performance Counter Monitor

4.3.1 Introduction

Microsoft Windows OS provides a utilization meter which shows the percentage of CPU being utilized, but it fails to provide more details on memory access stalls, synchronization, CPU I/Os, Simultaneous Multithreading (SMT) etc. Even UNIX's top utility [11] lists the portion of time slots that the scheduler in OS could assign to execution of the running programs and the rest of time it sits idle.

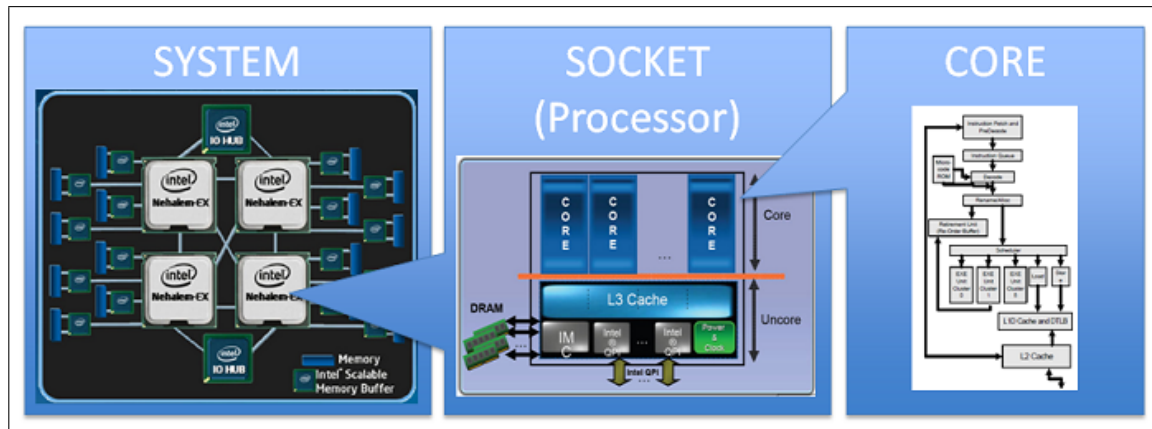


Figure 4.8: An overview of a modern multi-processor, multi-core system [14]

To get that data, Intel processors already provide the capability to monitor performance events inside processors named Performance Monitoring Units (PMU) implemented in Intel's processors. This feature set available in the following Intel processors: Intel Xeon 5500, 5600, 7500, E5, E7 and Core i7 processor series [2-4][14].

The following metrics are supported by Performance Counter Monitor (PCM)

- **Core:** features like instructions retired, elapsed core clock ticks, core frequency including Intel Turbo boost technology, L2 cache hits and misses, L3 cache misses and hits (including or excluding snoops) are included.
- **Uncore:** features like read bytes from memory controller(s), bytes written to memory controller(s), data traffic transferred by the Intel's QuickPath Interconnect (QPI) links are included.

4.3.2 Usage

Two ways it can be used i.e. by calling its C++ API or by calling it from the command line / UNIX shell. Different commands provides different functionality. Below table shows the different commands with its functionality.

Table 4.1: PCM Commands and its Functionality

Sr. No	Command	Functionality
1	pcm.exe	Provides CPU statistics for both sockets and cores
2	pcm-core.exe	Provides detailed core level information
3	pcm-memory.exe	Provides socket and channel level read/write throughput information
4	pcm-numa.exe	Provides memory NUMA memory access information information
5	pcm-pcie.exe	Provides PCIe usage information

The example or the demo[14] screen-shots and the screen-shots shall be discussed in the topic 4.3.3.

4.3.3 Demo

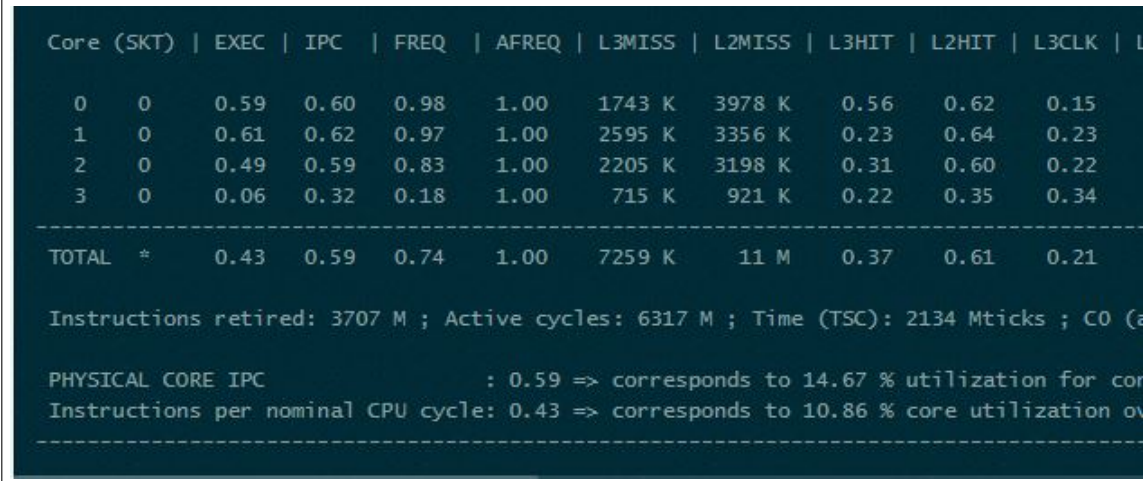
Here is a demo of each of the executable in the directory and a sample output of each. Note: user can export data to a Comma Separated Values (CSV) file for easier analysis. The following are the abbreviations which are used in representing the PCM data.

- **EXEC** : instructions per nominal CPU cycle
- **IPC** : instructions per CPU cycle
- **FREQ** :relation to nominal CPU frequency='unhalted clock ticks'/'invariant timer ticks' (includes Intel Turbo Boost)
- **AFREQ** : relation to nominal CPU frequency while in active state (not in power-saving C state)='unhalted clock ticks'/'invariant timer ticks while in C0-state' (includes Intel Turbo Boost)
- **L3MISS** : L3 cache misses
- **L2MISS** :L2 cache misses (including other core's L2 cache *hits*)
- **L3HIT** :L3 cache hit ratio (0.00-1.00)
- **L2HIT** :L2 cache hit ratio (0.00-1.00)
- **L3MPI** :number of L3 cache misses per instruction
- **L2MPI** :number of L2 cache misses per instruction
- **READ** : bytes read from memory controller (in GBytes)
- **WRITE** :bytes written to memory controller (in GBytes)

Performance Analysis of AI workload on Intel hardware platform

- **TEMP** : Temperature reading in 1 degree Celsius relative to the TjMax temperature (thermal headroom): 0 corresponds to the max temperature energy: Energy in Joules

Below are the screen-shots for PCM executable mentioned in table 4.1.



```
Core (SKT) | EXEC | IPC | FREQ | AFREQ | L3MISS | L2MISS | L3HIT | L2HIT | L3CLK | L
0 0 0.59 0.60 0.98 1.00 1743 K 3978 K 0.56 0.62 0.15
1 0 0.61 0.62 0.97 1.00 2595 K 3356 K 0.23 0.64 0.23
2 0 0.49 0.59 0.83 1.00 2205 K 3198 K 0.31 0.60 0.22
3 0 0.06 0.32 0.18 1.00 715 K 921 K 0.22 0.35 0.34
-----
TOTAL * 0.43 0.59 0.74 1.00 7259 K 11 M 0.37 0.61 0.21

Instructions retired: 3707 M ; Active cycles: 6317 M ; Time (TSC): 2134 Mticks ; C0 (e
PHYSICAL CORE IPC : 0.59 => corresponds to 14.67 % utilization for cor
Instructions per nominal CPU cycle: 0.43 => corresponds to 10.86 % core utilization ov
-----
```

Figure 4.9: Example for PCM.exe command

Performance Analysis of AI workload on Intel hardware platform

```

Time elapsed: 1004 ms
txn_rate: 1

```

Core	IPC	Instructions	Cycles	Event0	Event1	Event2	Event3
0	0.44	102 M	232 M	301 K	768 K	91 K	830 K
1	1.04	137 M	131 M	140 K	336 K	12 K	918 K
2	0.85	194 M	228 M	247 K	569 K	82 K	613 K
3	0.25	7377 K	29 M	17 K	31 K	4364	93 K
4	0.66	99 M	149 M	148 K	373 K	49 K	407 K
5	0.61	169 M	275 M	163 K	770 K	94 K	1105 K
6	0.89	186 M	209 M	258 K	399 K	55 K	635 K
7	0.48	101 M	211 M	200 K	641 K	64 K	670 K
8	0.50	88 M	176 M	177 K	547 K	73 K	510 K
9	0.19	4422 K	22 M	4572	20 K	3379	83 K
10	0.71	124 M	175 M	167 K	389 K	49 K	388 K
11	0.24	5738 K	24 M	6407	24 K	4258	90 K
12	0.67	58 M	87 M	73 K	184 K	23 K	249 K
13	0.90	161 M	180 M	160 K	308 K	80 K	603 K
14	0.71	49 M	69 M	70 K	100 K	16 K	193 K
15	0.29	16 M	56 M	37 K	51 K	37 K	241 K
16	0.73	46 M	63 M	40 K	80 K	25 K	300 K
17	0.28	6441 K	23 M	6106	22 K	4619	104 K
18	0.27	9346 K	34 M	28 K	52 K	8449	120 K
19	0.46	130 M	285 M	358 K	914 K	95 K	874 K
20	0.65	807 M	1240 M	502 K	4783 K	785 K	5832 K
21	0.16	4350 K	26 M	4635	74 K	3481	84 K
22	0.53	123 M	232 M	207 K	710 K	131 K	738 K
23	0.17	4402 K	25 M	5703	32 K	4500	93 K
24	0.50	87 M	175 M	188 K	617 K	37 K	524 K
25	0.18	4483 K	24 M	5430	24 K	4040	90 K
26	0.56	200 M	360 M	250 K	1192 K	84 K	3315 K
27	0.17	272 K	222 M	121 K	202 K	58 K	127 K

Figure 4.10: Example for pcm-core.exe command

```

Time elapsed: 1000 ms
Called sleep function for 1000 ms
-----
--          Socket 0          --  --          Socket 1          --
-----
--      Memory Channel Monitoring      --  --      Memory Channel Monitoring      --
-----
-- Mem Ch 0: Reads (MB/s):    49.91 --  -- Mem Ch 0: Reads (MB/s):    3.42 --
--           Writes(MB/s):    43.65 --  --           Writes(MB/s):    1.13 --
-- Mem Ch 1: Reads (MB/s):    13.95 --  -- Mem Ch 1: Reads (MB/s):    3.37 --
--           Writes(MB/s):     5.32 --  --           Writes(MB/s):    1.15 --
-- Mem Ch 2: Reads (MB/s):    10.08 --  -- Mem Ch 2: Reads (MB/s):   46.07 --
--           Writes(MB/s):     3.59 --  --           Writes(MB/s):   42.18 --
-- Mem Ch 3: Reads (MB/s):    13.52 --  -- Mem Ch 3: Reads (MB/s):    3.31 --
--           Writes(MB/s):     4.43 --  --           Writes(MB/s):    1.10 --
-- NODE 0 Mem Read (MB/s) :    87.47 --  -- NODE 1 Mem Read (MB/s) :    56.17 --
-- NODE 0 Mem Write(MB/s) :    56.98 --  -- NODE 1 Mem Write(MB/s) :    45.56 --
-- NODE 0 P. Write (T/s):   624374 --  -- NODE 1 P. Write (T/s):   622531 --
-- NODE 0 Memory (MB/s):   144.45 --  -- NODE 1 Memory (MB/s):   101.74 --
-----

|-----| |-----|
|--          System Read Throughput(MB/s):    143.64          --|
|--          System Write Throughput(MB/s):    102.54          --|
|--          System Memory Throughput(MB/s):    246.19          --|
|-----| |-----|

```

Figure 4.11: Example for pcm-memory.exe command

Performance Analysis of AI workload on Intel hardware platform

```

Time elapsed: 1014 ms
Core | IPC | Instructions | Cycles | Local DRAM accesses | Remote DRAM Accesses
0 | 0.33 | 15 M | 47 M | 22 K | 3620
1 | 0.23 | 4114 K | 17 M | 4843 | 1060
2 | 0.20 | 5205 K | 25 M | 6682 | 4486
3 | 0.23 | 6016 K | 26 M | 1369 | 1070
4 | 0.80 | 22 M | 28 M | 4045 | 1435
5 | 0.23 | 9756 K | 42 M | 11 K | 6362
6 | 0.22 | 5305 K | 24 M | 4357 | 1152
7 | 0.56 | 25 M | 44 M | 57 K | 10 K
8 | 0.24 | 5380 K | 22 M | 3655 | 1807
9 | 0.21 | 4525 K | 21 M | 2075 | 1219
10 | 0.53 | 20 M | 38 M | 6579 | 2557
11 | 0.22 | 4857 K | 22 M | 4607 | 2460
12 | 0.38 | 16 M | 44 M | 25 K | 2940
13 | 1.42 | 70 M | 49 M | 5793 | 2280
14 | 0.24 | 5952 K | 24 M | 2233 | 1007
15 | 0.25 | 5551 K | 22 M | 2150 | 835
16 | 0.31 | 8273 K | 26 M | 22 K | 1730
17 | 0.23 | 3939 K | 17 M | 1309 | 592
18 | 0.20 | 4401 K | 21 M | 3583 | 1833
19 | 0.27 | 5272 K | 19 M | 10 K | 1558
20 | 0.55 | 102 M | 188 M | 76 K | 69 K
21 | 0.20 | 4772 K | 24 M | 1801 | 1430
22 | 0.50 | 68 M | 137 M | 89 K | 46 K
23 | 0.25 | 7923 K | 31 M | 8629 | 17 K
24 | 0.35 | 17 M | 51 M | 38 K | 7632
25 | 0.19 | 5416 K | 27 M | 3670 | 1265
26 | 0.34 | 16 M | 48 M | 24 K | 9108
27 | 0.31 | 12 M | 40 M | 21 K | 34 K
28 | 0.34 | 14 M | 43 M | 7770 | 3473
29 | 0.24 | 7116 K | 30 M | 6161 | 1686
30 | 0.33 | 13 M | 41 M | 9403 | 3111
31 | 0.32 | 12 M | 40 M | 13 K | 2672
32 | 0.30 | 11 M | 37 M | 12 K | 1773
33 | 0.32 | 10 M | 31 M | 77 K | 2129
34 | 0.32 | 11 M | 36 M | 5342 | 2449
35 | 0.24 | 6862 K | 28 M | 4013 | 5977
-- | -- | -- | -- | -- | --

```

Figure 4.12: Example for pcm-numa.exe command

```
-----  
Time elapsed: 1000 ms  
Called sleep function for 1000 ms  
S0CH0; DRAMClocks: 933924607; Rank0 CKE Off Residency: 0.02%; Rank0 CKE Off Average Cycles:  
S0CH0; DRAMClocks: 933924607; Rank1 CKE Off Residency: 0.02%; Rank1 CKE Off Average Cycles:  
S0CH1; DRAMClocks: 933925096; Rank0 CKE Off Residency: 0.02%; Rank0 CKE Off Average Cycles:  
S0CH1; DRAMClocks: 933925096; Rank1 CKE Off Residency: 0.02%; Rank1 CKE Off Average Cycles:  
S0CH2; DRAMClocks: 933925354; Rank0 CKE Off Residency: 0.02%; Rank0 CKE Off Average Cycles:  
S0CH2; DRAMClocks: 933925354; Rank1 CKE Off Residency: 0.02%; Rank1 CKE Off Average Cycles:  
S0CH3; DRAMClocks: 933924943; Rank0 CKE Off Residency: 0.02%; Rank0 CKE Off Average Cycles:  
S0CH3; DRAMClocks: 933924943; Rank1 CKE Off Residency: 0.02%; Rank1 CKE Off Average Cycles:  
S0; PCUClocks: 800627536; Freq band 0/1/2 cycles: 99.84%; 99.84%; 0.00%  
S0; Consumed energy units: 2457737; Consumed Joules: 37.50; Watts: 37.50; Thermal headroom b  
S0; Consumed DRAM energy units: 1061128; Consumed DRAM Joules: 16.19; DRAM Watts: 16.19  
S1CH0; DRAMClocks: 933902607; Rank0 CKE Off Residency: 0.02%; Rank0 CKE Off Average Cycles:  
S1CH0; DRAMClocks: 933902607; Rank1 CKE Off Residency: 0.02%; Rank1 CKE Off Average Cycles:  
S1CH1; DRAMClocks: 933901094; Rank0 CKE Off Residency: 0.02%; Rank0 CKE Off Average Cycles:  
S1CH1; DRAMClocks: 933901094; Rank1 CKE Off Residency: 0.02%; Rank1 CKE Off Average Cycles:  
S1CH2; DRAMClocks: 933900756; Rank0 CKE Off Residency: 0.02%; Rank0 CKE Off Average Cycles:  
S1CH2; DRAMClocks: 933900756; Rank1 CKE Off Residency: 0.02%; Rank1 CKE Off Average Cycles:  
S1CH3; DRAMClocks: 933900898; Rank0 CKE Off Residency: 0.02%; Rank0 CKE Off Average Cycles:  
S1CH3; DRAMClocks: 933900898; Rank1 CKE Off Residency: 0.02%; Rank1 CKE Off Average Cycles:  
S1; PCUClocks: 800628916; Freq band 0/1/2 cycles: 100.00%; 100.00%; 100.00%  
S1; Consumed energy units: 2521661; Consumed Joules: 38.48; Watts: 38.48; Thermal headroom b  
S1; Consumed DRAM energy units: 854553; Consumed DRAM Joules: 13.04; DRAM Watts: 13.04
```

Figure 4.13: Example for pcm-pcie.exe command

4.4 OPENVINO Toolkit

4.4.1 Introduction

Intel launched its new toolkit named OpenVINO i.e. Open Visual Inference Neural Network Optimization in 2018 as free to use and open source. This toolkit provides key functionality to develop applications which emulates human vision, making it fast and optimized across all Intel platforms (GPGPU [16], CPU, Intel Movidius NCS [17], and ARRIA FPGA cards [18]) and making them easier to work on the heterogeneous computing by offering a common API.

This toolkit helps in quickly deploying applications. Through its optimization for the computer vision workloads across Intel hardware, providing an amplified performance. OpenVINO includes Deep Learning Deployment Toolkit (DLDT) [19] as its base.

Following are the features of the new released OpenVINO toolkit - Following are the features of the new released OpenVINO toolkit -

- It Enables Convolutional Neural Network (CNN) - based deep learning inference on the edge.
- Supports heterogeneous execution across CPU, Integrated Graphics (GPGPU), Movidius Neural Compute Sticks, and Movidius VPUs.
- It helps in speeding up time-to-market via an easy-to-use library of computer vision functions and pre-optimized kernels.
- It also includes optimized calls for computer vision standards including OpenCV [20], OpenCL [21], and OpenVX [22]

There are major two components for using OpenVINO,

- **Model Optimizer** - Model Optimizer tool is a cross-platform command line tool facilitating the transition between the training and deployment environment. It performs model analysis and optimizes the model execution for end-point devices.

Model Optimizer supports multiple deep learning frameworks and formats.

As the outcome of the model optimizer, we get intermediate representation file i.e. an Extensible Markup Language (XML) and a Binary (BIN) file.

- **Inference Engine** - Once the intermediate representative - IR is generated, we need to use the Inference Engine to infer input data. The Inference Engine library comes with a set of C++ classes to infer input data (images) and get a appropriate results. This library provides an API to read the output of model optimizer, set the input and output formats, and execute the model on the required device. The main focus in this thesis is on Inference Engine.

4.4.2 OpenVINO as Inference

Workflow and API overview

The common workflow contains the following steps:

- a. First step is to read an IR file into a `InferenceEngine::CNNNetwork` class of inference engine API. This class represents the network in host memory.
- b. Now Prepare inputs and outputs format, specific input and output precision, and the layout on the network is required. `CNNNetwork::getInputInfo()` and `CNNNetwork::getOutputInfo()` API are used here.
- c. Select the plugin (CPU, GPU, MYRAID, DLA etc.) on which to load your network. The `InferenceEngine::PluginDispatcher` load helper class is used here.
- d. Time to Compile and Load - to do so use the plugin interface wrapper class `InferenceEngine::InferencePlugin` and call the `LoadNetwork()` API to compile and load.
- e. Time to Set input data - With the network loaded using `LoadNetwork()`, an `ExecutableNetwork` object is used. Using this object to create an `InferRequest`, further it can be used for input and output.
- f. Execute - With the input and output memory now defined, choose your execution mode: **Synchronously** - `Infer()` is a method which blocks until inference is finished. **Asynchronously** - `StartAsync()` is a method to check status with the `wait()` method (0 timeout).
- g. Finally we need the Get the output after the inference gets completed. Assign the output memory or read the memory which is provided earlier. `InferRequest::GetBlob()` API is used to do this.

The following block diagram shows the integration process [23].

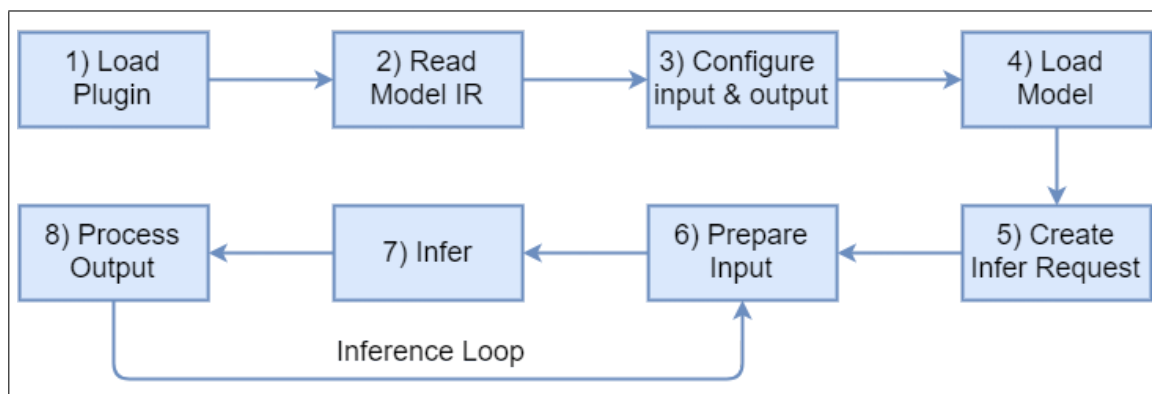


Figure 4.14: Block Diagram for OpenVINO integration process

4.4.3 Steps to Create a custom plugin

To create the custom plugin for the device running for inference using openvino, the following things should be taken care -

- a. **Plugin Registration** - device should be registered in IE i.e. Inference Engine, which is main inference API for openvino. Custom plugin can do a public inheritance from its inference engine plugin API wrapper.
- b. **Graph Creation** - can be done by using the minimum API which is used in ExecutableNetwork base.
- c. **Infer** - Infer is main call in the architecture, the executable network or graph network which is created in the above, is executed using infer. To implement this, API is to be implemented by plugin, which is used in InferRequest base of the openvino framework.

Chapter 5

Results, Conclusion and Future Work

5.1 Results

This section will highlight some graphs and snapshots observed during the benchmarking or performance analysis for TTS Tacotron 2 algorithm.

- a. **Topology Analysis** - Below are the screenshots of the topology used for benchmarking. The visualization of the topology is done using tensorboard provided by tensorflow.

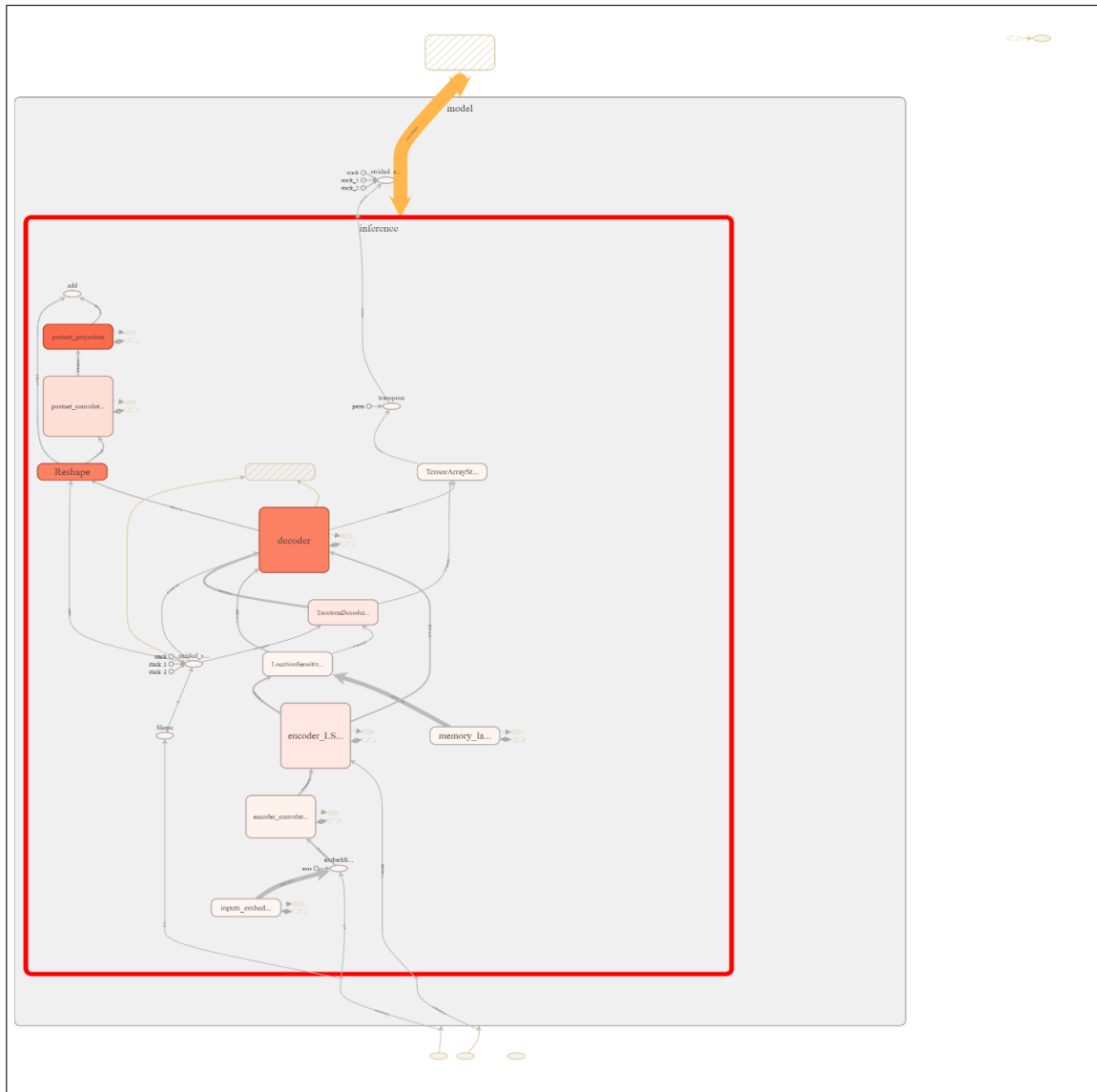


Figure 5.1: Tensorboard Topology Diagram for TTS Tacotron2

5.1 shows the pictorial view of the tacotron topology. It is generated using tensorboard. The portions in Red color suggests the amount of time taken in that section is more. In the above figure, decoder, reshape and postnet_projection are the sections consuming most time.

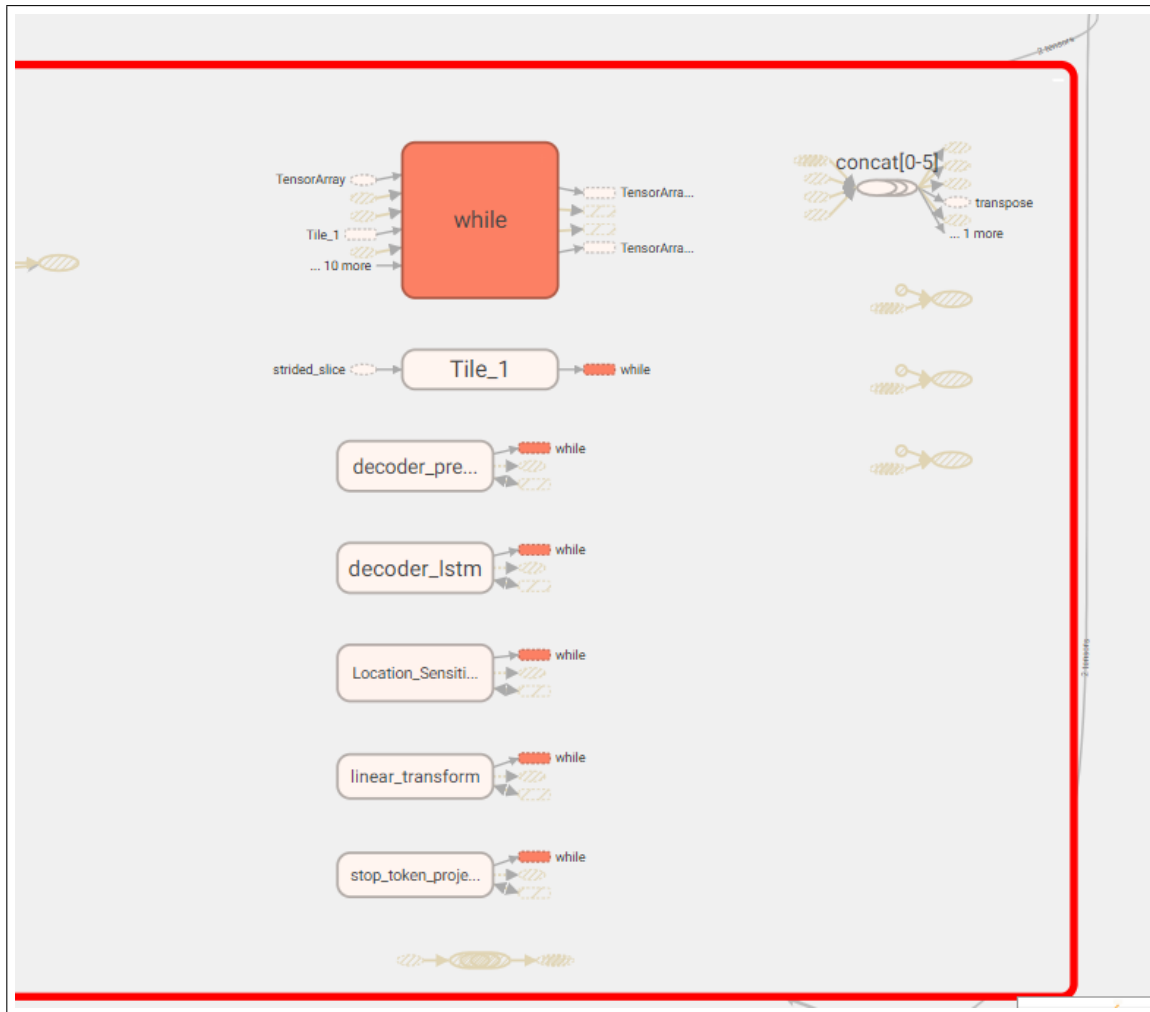


Figure 5.2: Hotspot in tacotron

On further investigations inside decoder section, the most time consuming element found are lstm's, prenet, attention mechanism, linear transformations and stop projection as seen in 5.2. These elements internally uses RNN's and LSTM's.

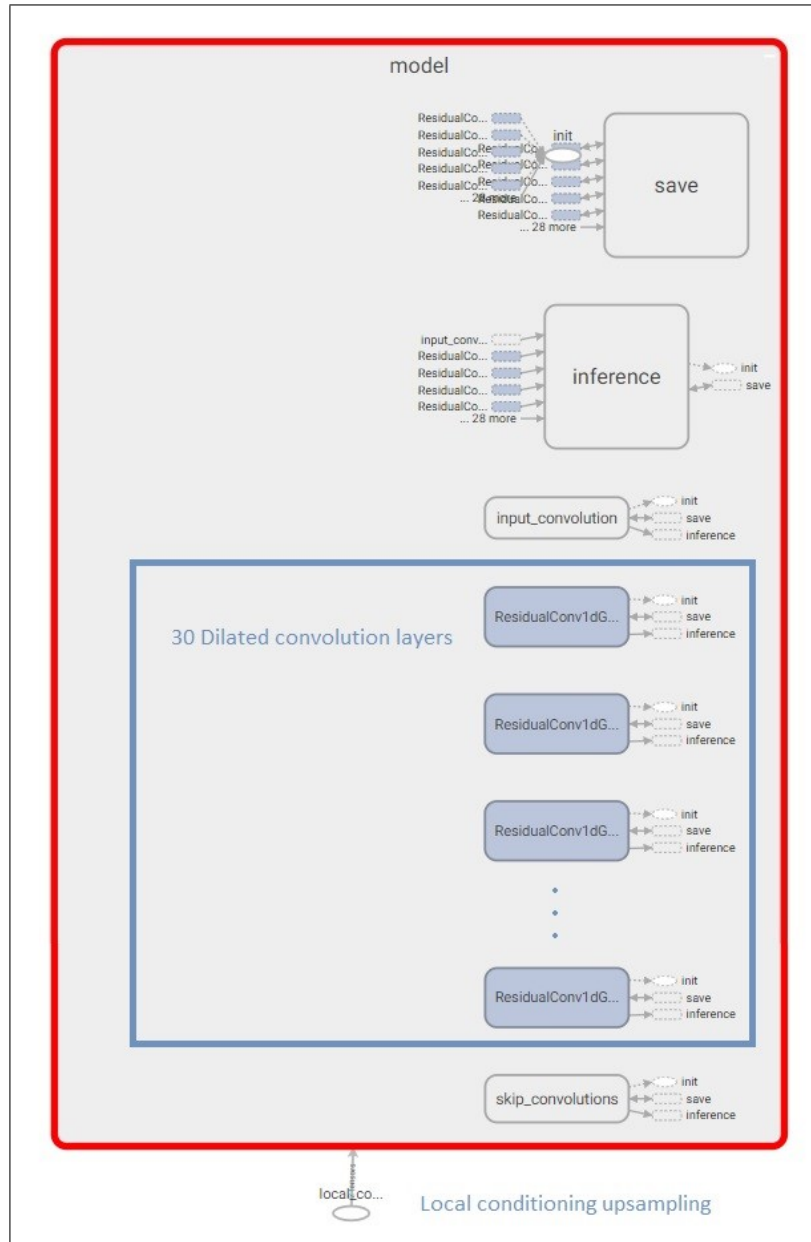


Figure 5.3: Wavenet visualization in tacotron

As seen in the above fig 5.3 the hotspot seen in the wavenet implementation is dilated convolutions [27] which takes 90% of the overall end-to-end speech synthesis time. Wavenet is responsible to generate audio from the given mel-spectrum files.

Initial observations for Tacotron2 suggests that the wavenet [2] is the major hotspot in the topology. Also the repository what we are using for benchmarking has implemented the Wavnet in pytorch [24] which is entirely different

framework implementation.

The following benchmarks are done on the Tacotron part from Tacotron2 topology.

b. **Baseline analysis** - List of libraries and versions of libraries used in this analysis are -

- falcon = 1.2.0
- inflect = 0.2.5
- librosa = 0.5.1
- matplotlib = 2.0.2
- numpy = 1.16.3
- scipy = 1.2.1
- tqdm = 4.31.1
- Unidecode = 0.4.20
- tensorflow = 1.9 (baseline and Intel optimized)

c. **Timeline analysis** - Timeline analysis is taken by running the tacotron module of TTS Tacotron2 in single and batch modes. Different time profiling graphs are shown below with the suitable test scenario.

(1) Time analysis for single sentence with lower and upper case.

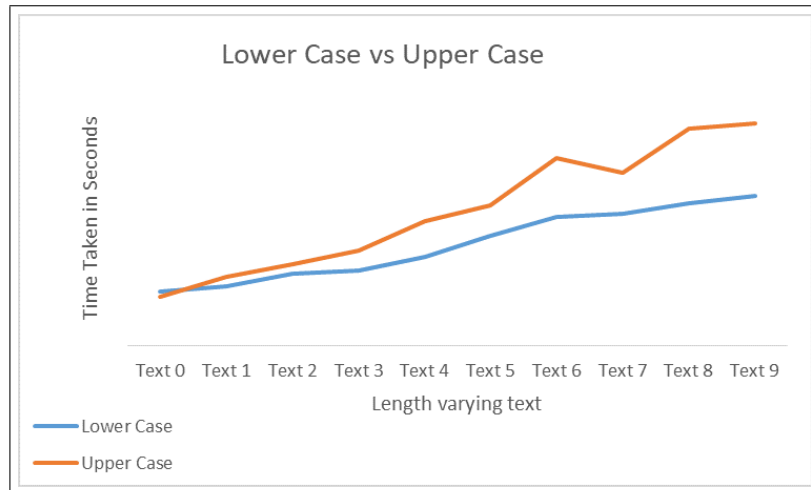


Figure 5.4: Comparison between lower case and upper case text

(2) Time taken by the topology to infer with the random data vs English language corpus.

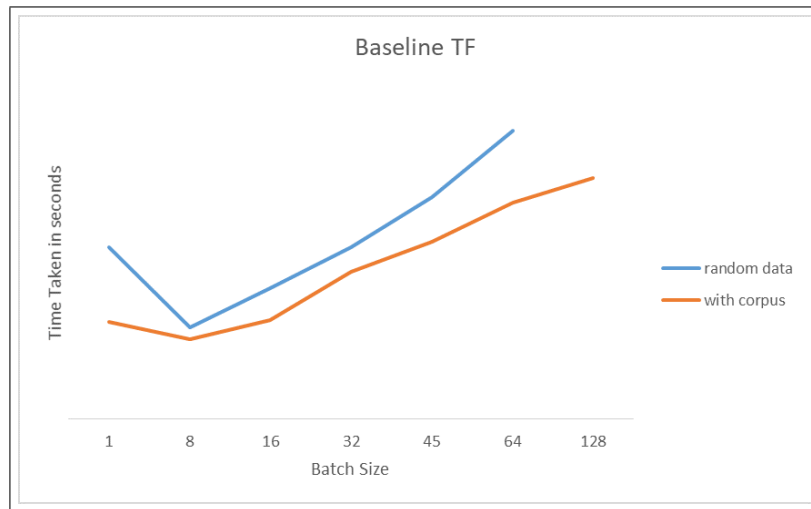


Figure 5.5: Random data vs Corpus data for Baseline TF

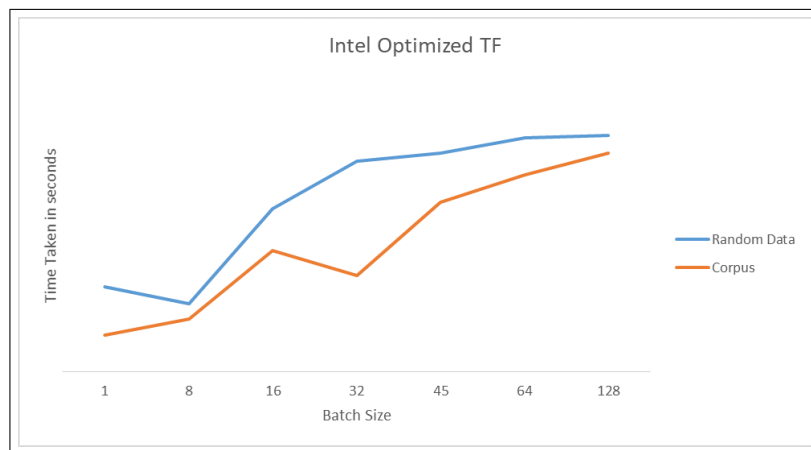


Figure 5.6: Random data vs Corpus data for Intel optimized TF

- (3) Time analysis graph for Batch streams showing comparison for baseline and Intel optimized tensorflow.

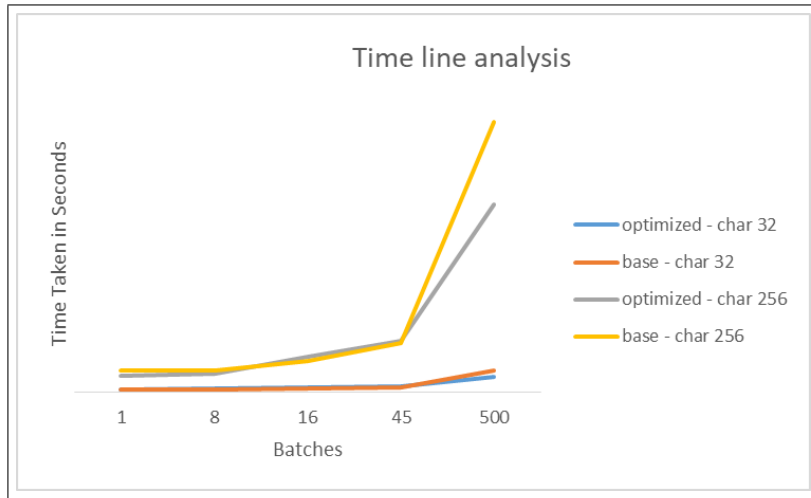


Figure 5.7: Time comparison between Intel optimized and Baseline tensorflow for batch

Figure 5.4 state that if the text used in the inference corpus is written in the upper case, the time taken to process is more. Adviced corpus representation should be in camelcase. Figure 5.5 and 5.6 shows the improvement using Intel’s optimized TF. These analysis helps in further evaluation and start system level performance.

d. System analysis - VTune Amplifier

System analysis gives the signature of the algorithm or the behavior of the algorithm on the server system. There are three major parameters checked inside the system analysis namely

- High Performance Characterization analysis -

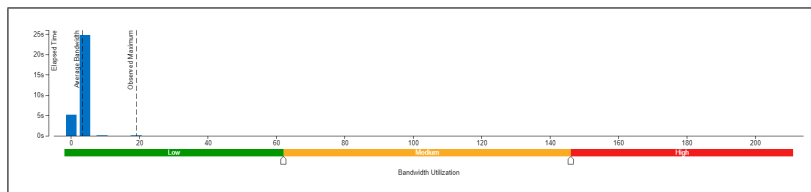


Figure 5.8: Baseline Bandwidth Utilization Histogram

Performance Analysis of AI workload on Intel hardware platform

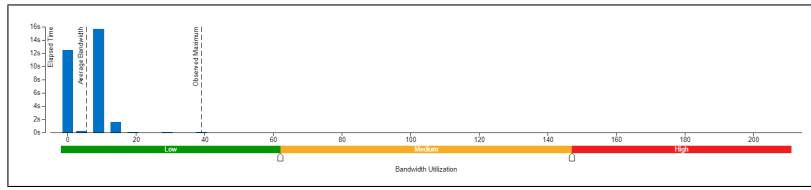


Figure 5.9: Optimized Bandwidth Utilization Histogram

The graphs plot indicates the bandwidth utilization or the memory objects with maximum contribution to the high bandwidth utilization. It is observed that Baseline tensorflow framework utilizes very less bandwidth.

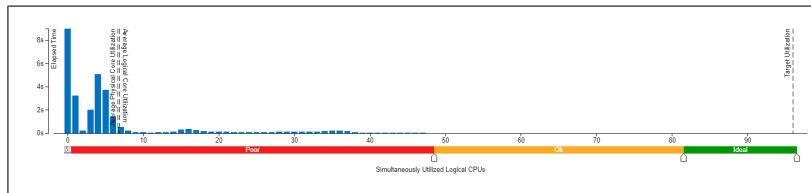


Figure 5.10: Optimized Effective Physical Core Utilization

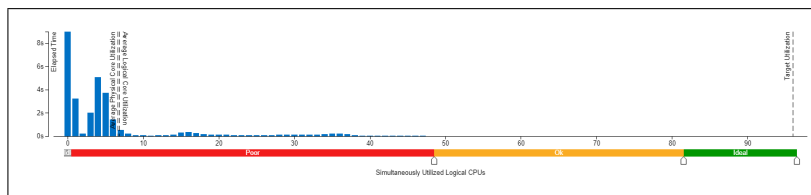


Figure 5.11: Baseline Effective Physical Core Utilization

The above graphs suggests the CPU is utilized more when Intel optimized framework is used compared to baseline tensorflow framework.

- **Hotspots by CPU Utilization-**

Function	Module
Eigen::internal::gemm_pack_lhs<float, long, Eigen::internal::TensorContractionSubMapper<float, long, (int)1, Eigen::TensorEvaluator<Eigen::TensorMap<Eigen::Tensor<float const, (int)2, (int)1, long>, (int)16, Eigen::MakePointer<const, Eigen::ThreadPoolDevice>, Eigen::array<long, (unsigned long)1>, Eigen::array<long, (unsigned long)1>, (int)8, (bool)1, (bool)0, (int)0, Eigen::MakePointer<, (int)16, (int)8, (bool)0, (bool)0>->operator()	_pywrap_tensorflow_internal.so
Eigen::internal::gebp_kernel<float, float, long, Eigen::internal::bias_data_mapper<float, long, (int)0, (int)0>, (int)16, (int)4, (bool)0, (bool)0>->operator()	_pywrap_tensorflow_internal.so
[vmlinux]	vmlinux
Eigen::NonBlockingThreadPoolTempl-tensorflow::thread::EigenEnvironment>::WorkerLoop	libtensorflow_framework.so
Eigen::TensorEvaluator<Eigen::TensorBroadcastingOp<Eigen::array<long, (unsigned long)3> const, Eigen::TensorMap<Eigen::Tensor<float const, (int)3, (int)1, long>, (int)16, Eigen::MakePointer<const> const, Eigen::ThreadPoolDevice>->packetRowMajor<(int)16>	_pywrap_tensorflow_internal.so
[Others]	

*NA is applied to non-summable metrics.

Figure 5.12: Hotspot analysis for Baseline tensorflow

Function	Module
<code>__INTERNAL_25____src_kmp_barrier_cpp_34128d84: __kmp_hyper_barrier_release</code>	libiomp5.so
<code>__kmp_yield</code>	libiomp5.so
<code>[MKL BLAS]@avx512_sgemv_scopy_down48_ea</code>	libmkiml_intel.so
<code>__schedule</code>	vmlinux
<code>update_curr</code>	vmlinux
[Others]	

*N/A is applied to non-summable metrics.

Figure 5.13: Hotspot analysis for Optimized tensorflow

Figures 5.10 and 5.11 indicates the major hotspot for the topology. It is observed that the optimized tensorflow framework uses MKL i.e. Math Kernel Library optimized for Intel hardware. Baseline framework uses the eigen operations.

- **Micro Architecture Exploration -**

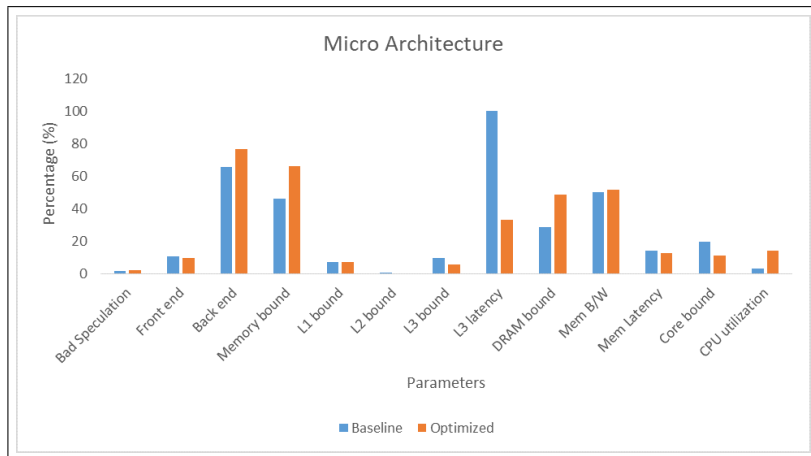


Figure 5.14: Micro Architecture analysis comparison

Figure 5.14 shows the comparison of base line vs intel optimized tensorflow effect on the hardware. Observations from above graph is stated below.

- **Bad Speculation** - represents a pipeline slot fraction wasted due to incorrect speculation. Here in our topology it is observed quite less. So less branch miss-prediction is observed.
- **Front End Bound** - responsible for fetching operations that are executed later on by Back end part. Stalls due to instruction cache misses would be categorized as Front end bound. Intel optimized TF shows improvement compared to baseline.
- **Back End Bound** - is divided into two main categories.

- * **Memory Bound** - measures a fraction of slots where pipeline could be stalled due to demand load or store instructions. It further consists of L1, L2 and L3 cache bounds. Ideal response of cache and cache latency indicates the fast availability of data to the processing core. Less the value is beneficial. Intel optimized TF shows better result on this.
- * **Core Bound** - suggests Core Non-memory issues of a bottleneck. It also represents the shortage of hardware compute resources or indicates the machine ranout of an Out of order resources. Intel Optimized TF shows better performance.
- **DRAM Bound** - indicates how often CPU was stalled on the main memory i.e. DRAM. Caching improves the latency and increases the performance.
- **Memory Bandwidth and Memory Latency** - Memory bandwidth represents the bandwidth limits of the main memory. Memory latency can be improved by optimizing data layout or using software prefetches.
- **CPU Utilization** - indicates how efficiently the application utilizes the physical CPU cores. An utilization of 100% shows that all the physical cores are utilized.
In this usecase, Intel optimized TF shows better results.

5.2 Conclusion

In this part of work, the performance analysis of TTS Tacotron2 AI topology on Intel platforms by characterizing workload proxies related to speech was carried out. Using Intel's tool VTune Amplifier tools we can get a performance statistics of how the topology is behaving on Intel's hardware. Using the results generated by the tools on the compute and memory usage of the topology, further recommendation's are to be proposed in the hardware(in terms of architecture i.e. instruction set architecture) and software(python, tensorflow etc.) optimization's teams internally in Intel. The following are the major observations and possible recommendations for the analysis done.

- Observed a 4.6x times better performance on CPU utilization for tacotron module in TTS Tacotron2 topology by simply using Intel's optimized TF framework.
- As we go on increasing the batch size and number of characters, the workload gets backend bound with memory as a bottleneck. This may be taken to further deeper analysis with focus on memory transactions or data partitioning techniques.

Performance Analysis of AI workload on Intel hardware platform

- As seen from the topology analysis, major hotspot for tacotron is its decoder. Decoder is mainly made up using LSTM's. Possible recommendations could be running these LSTM's on the other devices using openvino toolkit or optimization's in tensorflow to avoid the memory bounds. Even optimization's can be done in GEMM -General Matrix Multiplication.

5.3 Future Work

- Setting up tacotron on FPGA for benchmarking.
- Implementation of AI topology on FPGA using Open Visual Inferencing and Neural Network Optimization (OpenVino) toolkit.

References

- [1] speech synthesis defination, <https://whatis.techtarget.com/definition/speech-synthesis>
- [2] WaveNet: A Generative Model for Raw Audio <https://deepmind.com/blog/wavenet-generative-model-raw-audio/>
- [3] Ilya Sutskever, Oriol Vinyals, Quoc V. Le, “Sequence to Sequence Learning with Neural Networks, ”, Proc. NIPS, Montreal, CA, 2014 <https://ai.google/research/pubs/pub43155>
- [4] Jonathan Shen¹, Ruoming Pang¹. “NATURAL TTS SYNTHESIS BY CONDITIONING WAVENET ON MEL SPECTROGRAM PREDICTIONS, ”, Google, Inc. Feb 2018 <https://arxiv.org/pdf/1712.05884.pdf>
- [5] The LJ Speech Dataset, recorded by Linda Johnson and annotation by Keith Ito. <https://keithito.com/LJ-Speech-Dataset/>
- [6] The Trace Event Profiling Tool (about:tracing) <https://www.chromium.org/developers/how-tos/trace-event-profiling-tool>
- [7] Illarion Khlestov, Machine Learning Researcher <https://towardsdatascience.com/howto-profile-tensorflow-1a49fb18073d>
- [8] Intel System Studio from Intel’s Developer Zone, retrieved from <https://software.intel.com/en-us/system-studio>
- [9] Intel Parallel Studio XE from Intel’s Developer Zone, retrieved from <https://software.intel.com/en-us/system-studio>
- [10] Intel Media Server Studio from Intel’s Developer Zone, retrieved from <https://software.intel.com/en-us/intel-media-server-studio>
- [11] Intel VTune Amplifier, Intel Software Developer Zone <https://software.intel.com/en-us/vtune>
- [12] Top - Table of process , Unix Task manager [https://en.wikipedia.org/wiki/Top_\(software\)](https://en.wikipedia.org/wiki/Top_(software))

Performance Analysis of AI workload on Intel hardware platform

- [13] Interpreting General Exploration Data, Retrieved from https://scc.ustc.edu.cn/zlsc/tc4600/intel/2017.0.098/vtune_amplifier_xe/help/GUID-8FCE6EF8-301B-4D62-B09E-EF79FE7CC33D.html
- [14] Intel Performance Counter Monitor , Retrieved from <https://unhandled.wordpress.com/2017/05/17/hardware-performancemonitoring-deep-dive-using-intel-performance-counter-monitor/>
- [15] Intel Performance Counter Monitor , Retrieved from <https://software.intel.com/en-us/articles/intel-performance-countermonitorintro>
- [16] Intel Graphics Technology , Retrieved from <https://www.intel.in/content/www/in/en/architecture-and-technology/visual-technology/graphics-overview.html>
- [17] Intel Movidius , Retrieved from <https://www.movidius.com/>
- [18] Intel ARRIA 10 , Retrieved from <https://www.intel.in/content/www/in/en/products/programmable/10.html>
- [19] Intel Deep Learning Deployment Toolkit, Retrieved from <https://software.intel.com/en-us/opencv-toolkit/deep-learning-cv>
- [20] OpenCV, Retrieved from <https://opencv.org/>
- [21] OpenCL, Retrieved from <https://www.khronos.org/opencl/>
- [22] OpenVX, Retrieved from <https://www.khronos.org/openvx/>
- [23] OpenVINO integrate with custom application, Retrieved from https://docs.openvino-toolkit.org/latest/_docs_IE_DG_Integrate_with_customer_application_new_A
- [24] Pytorch, Retrieved from <https://pytorch.org/>
- [25] MKL-DNN, Retrieved from <https://github.com/intel/mkl-dnn>
- [26] Fused RNN Operators for CPU , Retrieved from <https://cwiki.apache.org/confluence/display/MXNET/Fused+RNN+Operators+for+CPU>
- [27] Long, J., Shelhamer, E., Darrell, T. (2014). Fully Convolutional Networks for Semantic Segmentation. Retrieved from <http://arxiv.org/abs/1411.4038v1>
- [28] GEMM - General Matrix Multiply , Retrieved from <https://software.intel.com/en-us/articles/gemm>
- [29] SPEC - Standard Performance Evaluation Corporation, Retrieved from <https://www.spec.org/web2005/>

- [30] DAWNBench Retrieved from <https://dawn.cs.stanford.edu/2017/11/29/dawnbench-intro/>
- [31] MLPerf Retrieved from <https://mlperf.org/>
- [32] Facebook research author Park, Jongsoo and Naumov, Maxim and Basu, Protonu and Deng, Summer and Kalaiyah, Aravind and Khudia, Daya and Law, James and Malani, Parth and Malevich, Andrey and Nadathur, Satish and Miguel Pino, Juan and Schatz, Martin and Sidorov, Alexander and Sivakumar, Viswanath and Tulloch, Andrew and Wang, Xiaodong and Wu, Yiming and Yuen, Hector and Diril, Utku and Smelyanskiy, Mikhail, year = 2018, month = 11, pages = , title = Deep Learning Inference in Facebook Data Centers: Characterization, Performance Optimizations and Hardware Implications