Learning Based Performance Prediction For Embedded Hardware

Master Thesis

Submitted in Fulfillment of the
Requirements for the Academic Degree
M.Sc. Automotive Software Engineering

Dept. of Computer Science
Chair of Automotive Software Engineering
Technische Universität Chemnitz, Germany

Thesis work carried out at Software Systems Engineering
Dept., Robert Bosch GmbH,
Renningen, Germany

Author:	Rahul Gajanan Mahajan
Student ID:	424592
Supervisors:	Dr. Arne Hamann
	Michael Pressler
	Falk Wurst
Examiner:	Dr.-Ing. Alejandro Javier Masrur

November 10, 2018

Acknowledgement

TBD

Abstract

TBD

Contents

1. Introduction

This thesis evaluates approach of estimating the performance of software workload on embedded hardware using machine learning. This is achieved by researching different available techniques, approaches and tools to evaluate performance of software on embedded hardware processor. This work is an attempt to represent the relevant features and highlighting various factors in existing methodology.

Now a days electronic systems are widely used in automotive industry. These electronic systems provide better control than mechanical control with use of the software. These systems are used in commercial vehicles, trucks, motorcycles, forklifts, tractors, excavators and many more automobiles. And these electronic systems provide better control to and collect information from automobiles. These systems consist of electronic components such as processors, sensors, actuators, electrical components, electrical circuits, communication devices, power supply units and others.

Electronic systems provide control operations in different domains of vehicles such as powertrain, chassis, passive safety, body/comfort, multimedia/telematics and man-machine interface which consist functionality like engine management, gearbox control, anti-locking system, electronic stability program, airbag control, navigation,instrument panel and many more. All these domains are connected and communicate to each other through on board electronic bus systems techniques such as Controller area network(CAN), Media oriented systems transport(MOST), etc.

In automotive electronics systems, Electronic control unit(ECU), is embedded system provides control to one or more electrical systems or subsystems in vehicles. ECU consist processors, interface for inputs and outputs, memory, communication links and many more. All sub domains discussed above consist one or more ECU as electronic system. Now a days upto 80 ECUs are used in modern cars. Development of ECU involves hardware and software to execute tasks for defined module. ECUs are programmable and can be reprogrammed according to functionality and requirements. Now a days, ECUs have more computing powers in single processor. Modern days processors provide higher computing powers typically upto 64 bit processing and operating frequency in gigahertz. Following figure **??** illustrate the ECUs used in various domains of a vehicle. Automotive electronic control unit market predicted to reach upto $55.97 billion by 2024. Around 40% of revenue in automotive industry is from automotive electronics. This is because most of vehicles are software driven[ref].

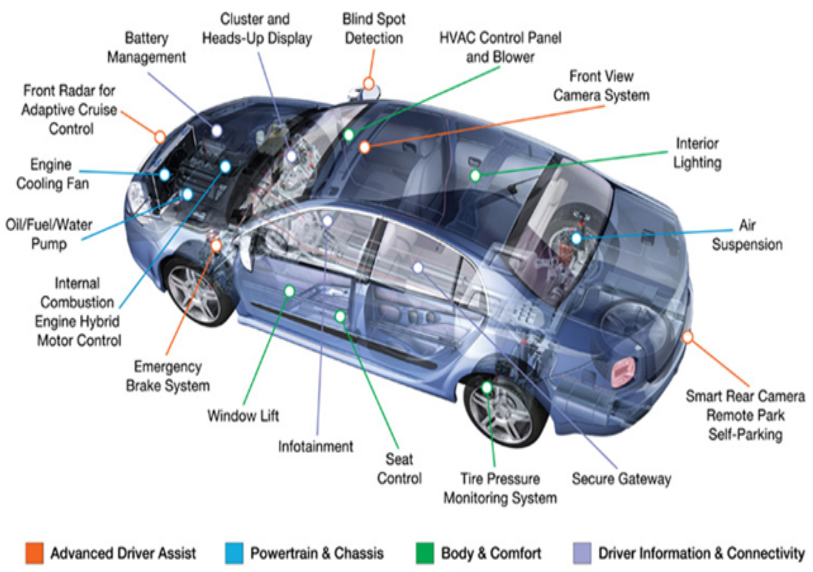

Figure 1.1.: Number of ECUs are used in different domains of single vehicle

Software is an integral part of embedded system in automotive domain. Software is running on microprocessors/microcontrollers provides the control mechanism. It is brain behind the every action taken in embedded systems. In current era, software is backbone for advanced features used in automobiles such as advance driving assistant systems have anti-breaking systems(ABS), electronic stability program(ESP), parking assistance system and many more. It also perform measure role in safety critical system used in automobile. As software plays important role in embedded systems then performance of software is also essential. Performance of software in application of safety critical systems is not only needed to be correct but also should have quick repose time. So performance of any software running on embedded hardware need to be evaluated before integrate with real hardware and used in application.

Software performance is not only about how fast it can handle large numbers but it is also about memory management, networks, cores, architectures. Some algorithms looks good on paper but they perform poorly on hardware. Being able to manage the performance of software requires well understanding of hardware architecture, operating systems and runtime libraries[ref- lemire.me]. As mentioned above, modern cars have more than 70 ECUs per vehicle. Following figure **??** shows increase in the number of ECUs year by year in automotive industry. Each ECU has number of functionalities to perform. Year by year number of ECUs per vehicle are increasing and ability to perform functionalities per ECU is also increasing. According to Moore's law, number of transistors on integrated circuits doubles approximately every two years. This advancement in capabilities of a ECU is due to growth in technology of semiconductor manufacturing as well as demand of software driven vehicles.

Each ECU in vehicle has it's own software stack to control the operations, diagnostics, monitoring the network, reading the sensor values, to operate the actuators and many operations. In demand of software driven vehicles software performs the crucial part in it. Whether it is safety critical systems, powertrain system, advance driving assistance systems, multimedia systems in vehicle. All these systems have their own ECUs which controlled by software. In world of competition, each original equipment manufacturers and suppliers wants to stay ahead of each other and they are coming with new features and technology in the automotive industry. In current trade of automotive industry and future of automotive industry, interest in software driven vehicle is increasing and it is expecting improvement in hardware which can deliver the adequate performance of software. Software is playing the vital role of performance of vehicle as well as in automotive industry.

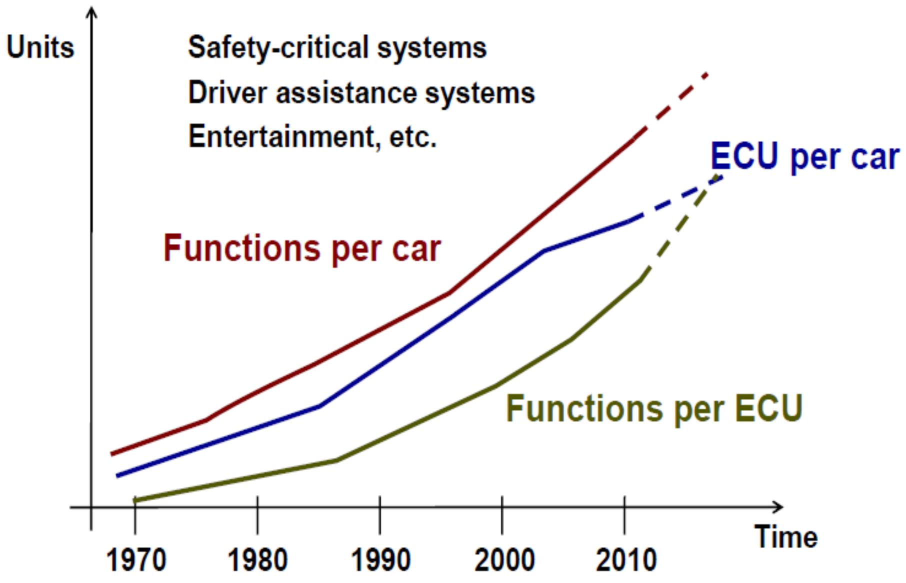

Figure 1.2.: Trends in automotive systems

1.1. Purpose of the Thesis Work

In initial phase of the thesis work, we discussed the problems behind the product development with sequential method and effect of the market pressure to release the product. Also tools available in market to evaluate the performance of software.

1.1.1. Sequential development

In automotive industry, sequential product development model is used widely to develop a product. V-model is one the broadly used process to develop a product. V model is also known as verification and validation model and processing is performed in V-shape. You can see in following figure ?? the V-model design.

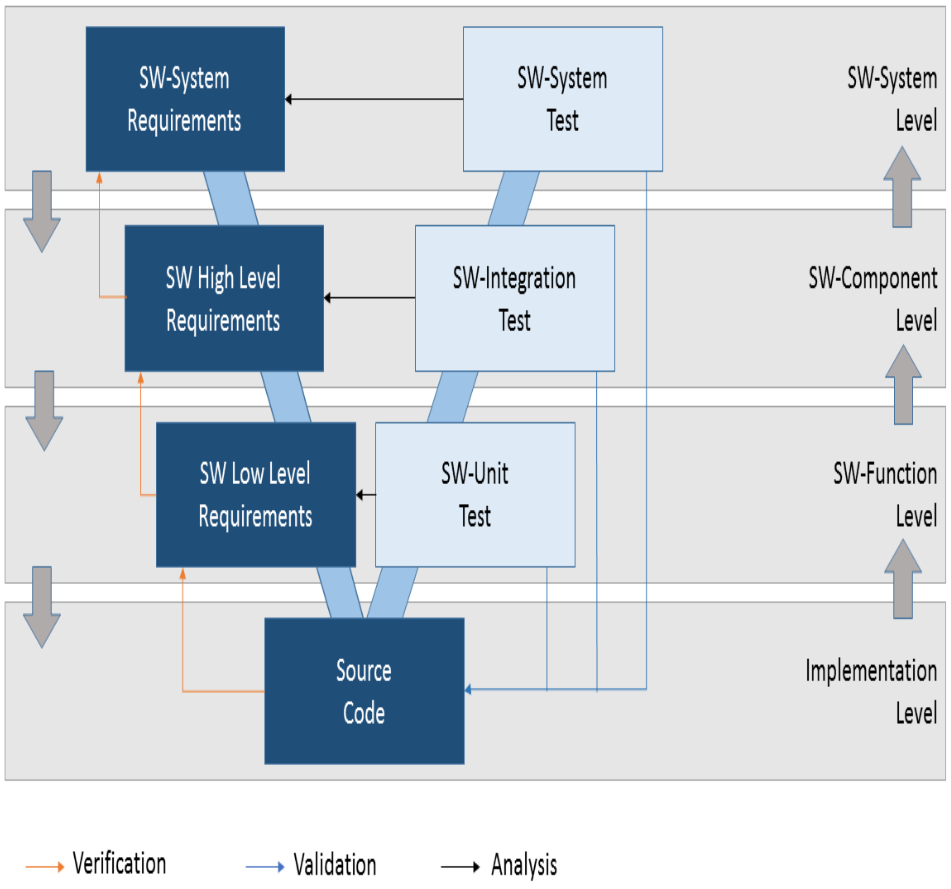

Figure 1.3.: V-Model

To give small overview of V-model, it starts with requirement phase. Where customers and OEMs provide their product requirements. After that hardware and software architecture requirements are carried out and then process go to much more detail. In next phase component level requirements are collected and analyzed. After all high level and low level requirements, implementation process starts. After implementation, different testing techniques are performed at each corresponding phase, to identify bugs in low level, high level and system level design and implementation of product. And after all this final product is delivered to customer or OEMs.

Sequential development where software development waits for available hardware, is still prevailing industry norm. And this flaw of sequential development method is that to implementing software for product, hardware requirement must be known to software developers. Software developers need hardware architecture, memory, input/output(I/O) and many more parameter to design the architecture of software. After building the software for given hardware architecture and product, software tester need to test it on hardware and which should be available to testers. In some cases this hardware dependencies are observed and it delays the product release to customer, OEMs or to market.

1.1.2. Parallel development

For every product development, time, cost and quality, these are key concerns. In device supply chain of semiconductor companies and OEMs are turning to parallel product development to shorten the development cycles and buy time to integrate and test products. Virtual prototypes are used in parallel development, where virtual prototypes are functional software model of systems under development. These virtual prototypes are used in unavailability of harware, which helps faster software development.

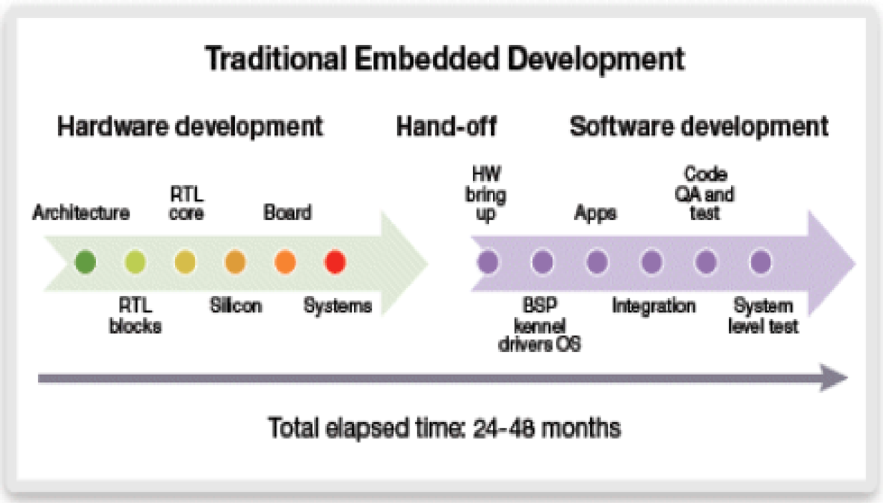

Figure 1.4.: Time estimation for sequential product development

Embedded devices mostly depends upon custom hardware. These devices have broad array of use cases and hence they give rise to broad array of hardware architectures. There is no standardize platform is available. Absence of standardization increases the time in product development. As mentioned in this [ref], realistic time estimation to develop complex embedded project requires 36 to 48 months of time with sequential development method. 12-18 months to hardware development, and 12-18 months for software development and 6-12 months for integration, verification and validation. Above figure?? illustrates the estimate time to build product using sequential method.

As we seen in above figure ??, software development lies on right hand side. And shifting left the software development process with parallel to hardware development results in parallel development. It can be seen in following figure ??. In parallel development, hardware and software team work in lock-step, communicating regularly, and continuously integrating hardware and software results into save time for modifications. With use virtual prototype tools parallel development of product can be possible.

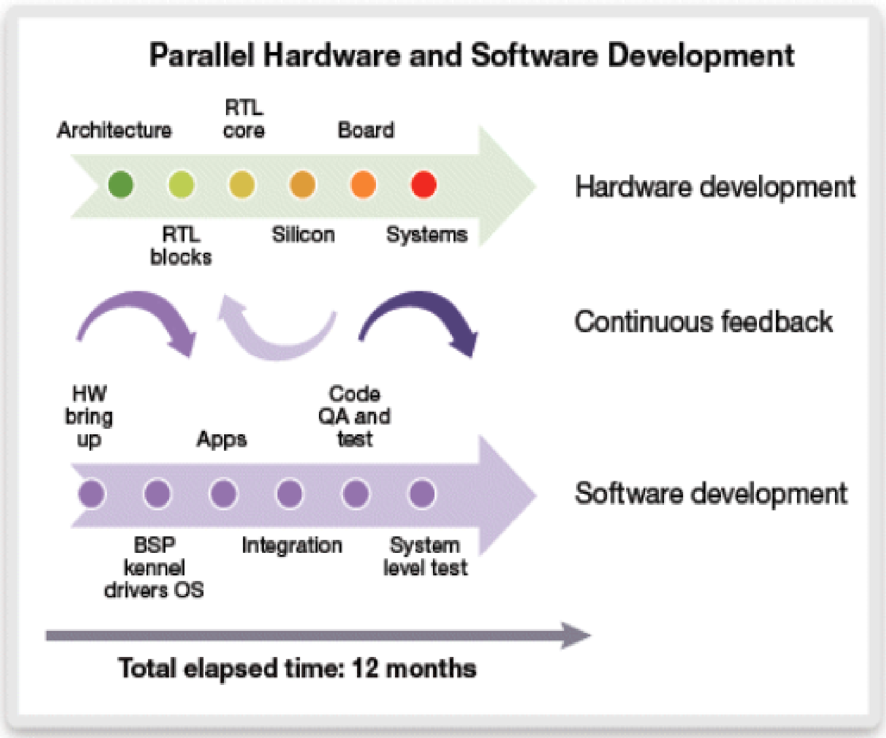

Figure 1.5.: Hardware and software development in parallel

In some scenarios, the system(s) on chip(SoC) takes time to come back from fabrication. In such scenarios, using virtual prototype of same SoC, software development can moves forward. It gives time to software developers for debugging and testing. With use of virtual prototypes in parallel development, avoid delays in product development as well as in product release in market.

1.2. Motivation

Advanced risc machine(ARM), is family of reduced instruction set computing(RISC) architectures for computer processors. Arm holdings develops architecture and licenses it to other companies, who design their SoC product that includes architecture provided by ARM incorporate with memory, interfaces and many more. Processor architectures made by ARM are widely used automotive industry to develop ECUs as well as in other consumer industry for electronic products. ARM also provides the solutions for virtual prototyping, which provides motivation to my thesis work. Also it provide development solutions to embedded systems and servers. These tools are used in development od billions of ARm-based devices on the market.

As previously discussed how virtual prototypes help in early software development independent to physical availability of hardware. Goal of the ARM is to provide pre-silicon software development and analysis with better analysis, accuracy, availability, performance and ease of use. Following diagram **??** illustrates available virtual prototypes/simulators from ARM with respect to abstraction level and simulation speed.

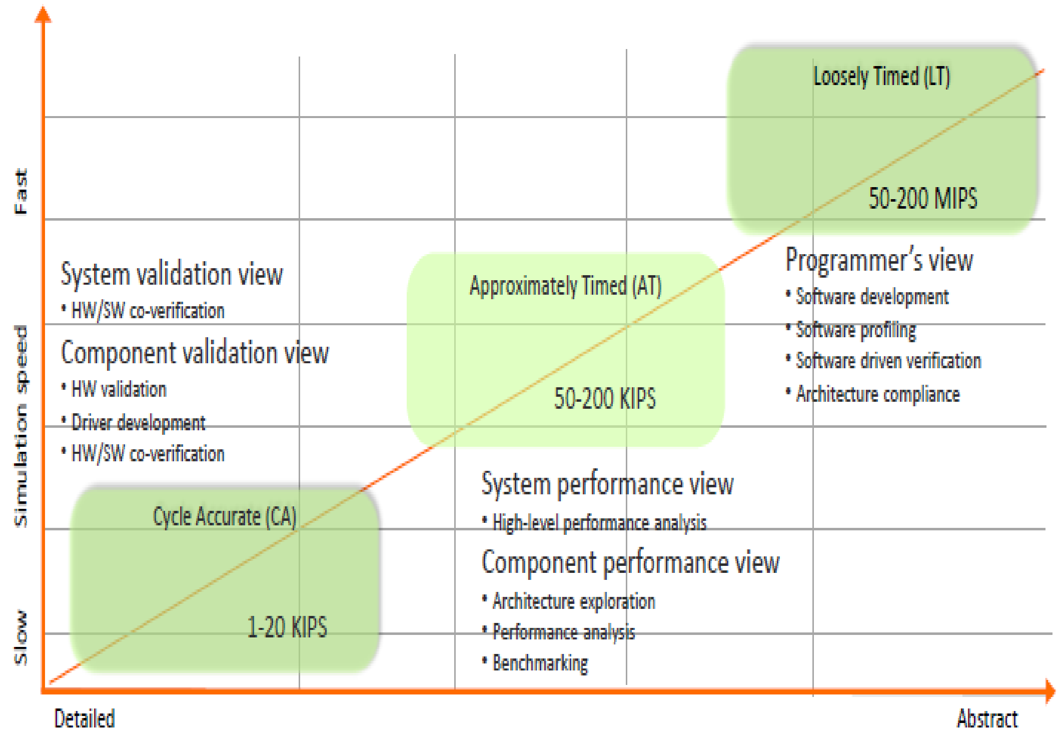

Figure 1.6.: ARM virtual prototype solutions

In above diagram **??** ARM has cycle accurate tool which provides detailed abstraction level but slow simulation speed in between 1-20 kilo instructions per second(KIPS). This tool can be used for system validation and component validation applications such as hardware/software co-verification, hardware validation, driver development. This cycle accurate tool is known as ARM cycle models. ARM also provide loosely timed tool for simulation of SoCs. Which does not need detailed abstraction level and provides high simulation speed witn 50-200 million instructions per cycle(MIPS). This loosly coupled tool is known as ARM Fast Models. It can be used for application like software development, software profiling, and software driven verification.

Above diagram **??** shows that for detailed abstraction level ARM cycle models can be applied but it provide slow simulation speed. Whereas ARM fast models can be applied where high simulation speed and not detailed abstraction level required. But there is no tool available where intermediate simulation speed and abstraction level required. For targeted application such as software profiling, benchmarking, performance analysis and software development. All these tools are expensive and also sometimes project dependent. And this problem motivates me to perform my research in my master thesis and find the optimal solution.

Performance of software, running on hardware can be measured. Using machine learning it is possible to learn the performance relevant aspects of hardware platforms (SoC) to be able to predict the timing behavior of different applications. Performance of software running hardware can be measured in terms of micro-architectural events. Now a days every processor comes with hardware performance counter to monitor micro-architectural events. By running different software applications on two embedded cross platform hardware, statistical relationship can be formed and using this relationship can be use to predict the performance of software using machine learning.

1.2.1. Outline

2. State of The Art

2.1. What is Performance?

Performance of any embedded system is depends upon the amount of the work accomplished by it and it depends upon many context. Performance depends upon many aspects. One of the aspect is response time, it is total time taken by system to respond the service being requested. Another aspect maybe processing speed, how many instructions are processed at a time provides the processing speed of the system. High throughput for input and output data, bandwidth, power consumption, latency, availability and many aspect are there for performance. In this thesis our main performance aspect is total number of cycles to execute the software at different phases.

There are many techniques and tools are available to observe the performance of software on given embedded hardware. Some of these techniques are provided by tool vendors and some of them are part of industrial and educational research. Using tools provided by vendors, which helps to analyze the performance data graphically and provide more functionality and features to software engineers. Such tools are Silexica[link], ARM Streamline Performance Analyzer[link], or hardware simulation tools such as gem5 simulator[link to gem5]. Also there are ongoing research to evaluate the performance of software using machine learning techniques with state of the art. Using machine learning technique, it is possible to predict the performance of software by creating dataset of performance. Also using Worst Case Execution Techniques[WCET] use to analyze the performance.

Traditional simulation approaches, simulator estimates the performance of software or application running on slow cycle accurate or cycle approximate models ISSs [1.3,2.7] of target embedded hardware processor. To improve the speed of ISSs and maintain the accuracy, hardware-assisted acceleration and transaction level techniques are recently proposed [2.5,2.8]. Chiang et al. [ieee] discusses the fast cycle accurate ISS for SoC based on QEMU and system C. Li et. al[IEEE] discusses the inspection of problem to determine the WCET of processor. This paper further discuss issue of micro-architectural modeling to determine the WCET of known instruction sequence. Without need of source code or reverse engineering on binary files,it is possible to estimate WCET for software or application using integer linear programming(ILP) and constraint logic programming(CLP)[ieee]. Analytical model to estimate the performance of software on cross platform with using learning based techniques is discussed by Gerstlauer et. al [IEEE]. It evaluates the performance prediction of whole program using regression learning technique. Further they added new methodology[ieee] using learning model, to predict the power and performance of software at different phases of software or applications. Using analytical techniques, Joseph et.al [] obtains good approximate model Akaike's information criteria is used in iterative process to extract a linear model. Lee et al. [1.18] and Khan et al [1.14] focused on predictive modeling approach from uni-processor to multi-processor and multicore proccessor.

2.2. Analytical Models

Analytical models are models which contains the information about behavior of system. It has data regarding input and output the system. Using this data which tell the behavior of system is used to form mathematical model or statistical model to understand the system. This analytical models are learned using learning algorithm techniques. There are different techniques to evaluate the performance of software on hardware by creating analytical models. Following are the state of the art performance analytical models used to predict the performance of software on processor.

2.2.1. Learning-based Analytical Cross-Platform Performance Prediction

This IEEE paper published by Gerstlauer and his colleagues from University of Texas at Austin, USA. This paper focuses about challenges with cycle accurate simulators and how analytical models is used to get performance evaluation of software at higher speed.

Simulation based models are widely used to model the hardware but these models are costly as well as slow to evaluate the performance of software using micro-architectural events. As collection of performance data on ISS is slow but collection of performance data on existing platform is faster because execution is done at native speed. Latent relationship can obtain by executing a program of processor and executing the same program of different processor. In this paper 'target' is preferred as platform for which the prediction is performed and 'host' is preferred as platform for which various statistics are collected. Host platforms are AMD Phenom II X6 1055T and Intel Core i7 Core. Target is platform is ARM processor model simulated on gem5 simulator.

Performance of software on target and host platform is measured using performance evaluation tool PAPI. PAPI measures all micro-architectural events from PMU with respect to software executing on hardware platforms. In this scenario, performance aspect is number of cycles consider. Total 157 programs with and 100 instances for each program are used to collect the training data. Input vector for Intel and AMD are different in term of number. Because number of performance counter supported by both processors are different. Intel i7 supports 14 where as AMD Phenom supports 8 hardware performance counter using PAPI. Training data collected from both host platform excepting the total cycles. total cycles event is collected on gem5 simulator for ARM processor for same 157 programs and 100 instances. Training data help to understand this relationship between input feature vector and output feature vector. Following figure **??** shows the framework for this paper.

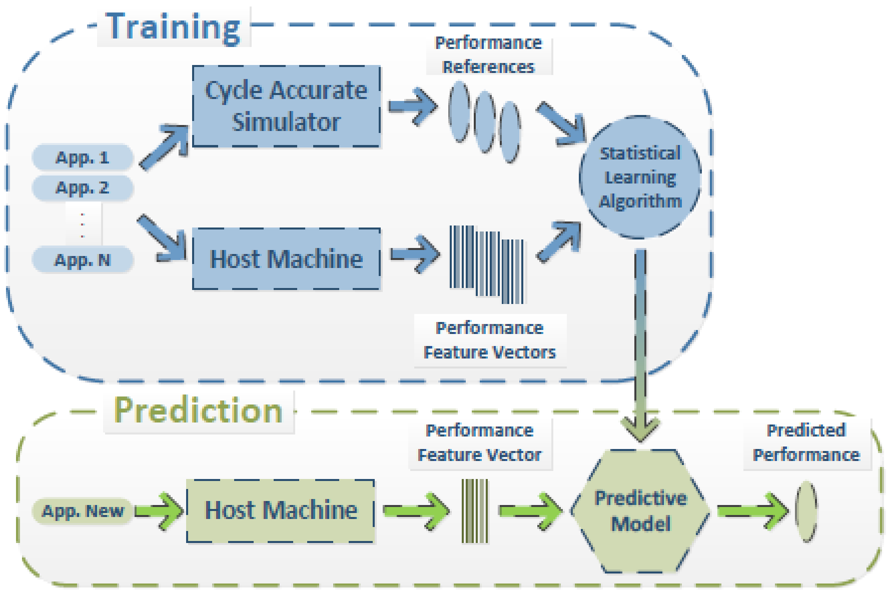

Figure 2.1.: Cross-platform performance prediction framework[]

To understand the collected training data, data is visualized using data analysis and data visualization techniques such as Principal Component Analysis(PCA) for reduce the dimensions and Co-relation coefficient for one to correlation of hardware performance counters. Using learning techniques this statistical model is learned and prediction is performed. Regression technique of machine learning is used. In which Lasso linear regression and Constrained Locally Sparse Linear Regression(CLSLR) are used. To select the best learning model cross-validation technique is used and as best result CLSLR is preferred.

To test the accuracy of CLSLR algorithm,accuracy test is conducted on 15 different programs which are not used during training data collection. Input feature vectors for all these test programs are collected on both platforms. Result for predicting the whole program performance, errors of more than 40% are observed on embedded benchmarks.

2.2.2. Accurate Phase-Level Cross-Platform Power and Performance Estimation

This IEEE paper is also published by Gerstlauer and his colleagues from University of Texas at Austin, USA with improved and additional features with respect to previous paper. In this research, authors provided LACross method learning based method, for cross-platform time varying software performance as well power consumption predictions. Where all data is collected with fine-grained phase based approach. Also instead of simulation model, native hardware platforms are used for target and host.

In this approach host platform is Intel i7 920 processor used. Whereas as target platform, ODROID-U3[ref] development board is used which contains ARM Cortex-A9 processor and to collect the power consumption ODROID-XU3 development board is used with on-chip TI IN A231 current sensor. To collect training data, similar programs are used as it mentioned in previous paper except 100 instances, here only single instance is used. Training data collected on these hardware platform using performance evaluation tool PAPI. Feature extraction for host platform contains all hardware performance counter except cycles where as feature extraction for target hardware are total cycles and power consumed. Following diagram illustrate the framework for LACross. It has similar goal is to accumulated latent relationship between to processor.

All these features are extracted using LLVM tool chain. Where LLVM uses Clang as compiler and it interfaced with PAPI libraries to collect the data at different phases of software. Time varying performance of software on hardware platform is collected in database. Values of performance counter is measured at each phase of software by inserting PAPI API calls to provide time varying data.

All training data is gathered from host and target platform applied to learning algorithms. In this paper, variant of Lasso regression, that is Simplex Constrained Quadratic Programming(SCQP) applied to predict the performance and power consumption of software on hardware platforms at each phase level. All results are calculated on 35 selected benchmark programs from three different standard benchmark suites. All these benchmark programs are not included during the training data collection phase. Results shows over average 97% accuracy for both performance and power consumption at higher speed of 500 MIPS.

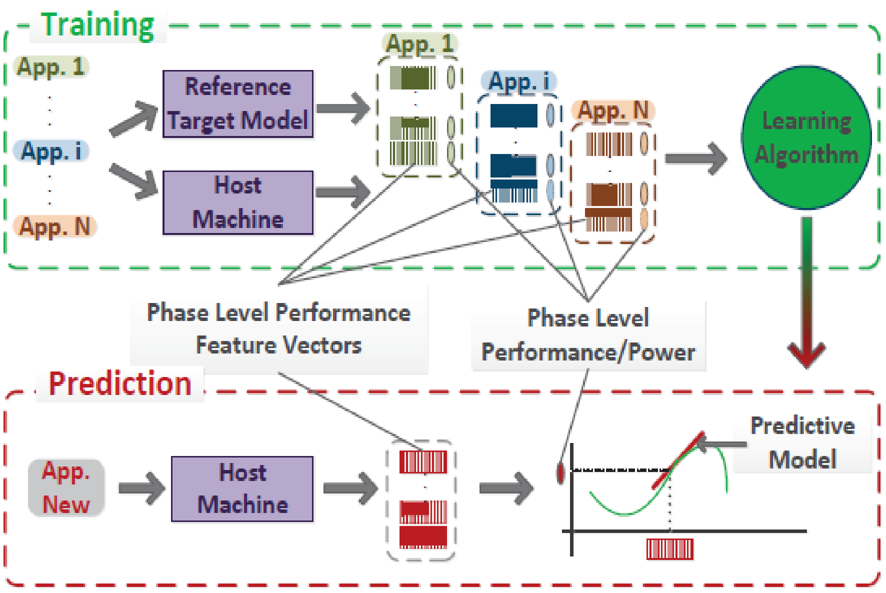

Figure 2.2.: LACross overview[]

2.2.3. Composable Performance Regression for Scalable Multiprocessor Models

2.3. WCET analysis

2.3.1. Combining instruction set simulation and WCET analysis for embedded software performance estimation

2.4. Silexica

Silexica GmbH provides tools for software design automation which enables software developers and system architecture to design and program multicore systems via SLX tool suite. System architects work with software models which maybe open source, in house, commercial source, the traditional methods are not scalable for software optimization in multicore system. To understand the software or system behavior on multiocre system, a full and precise analysis of software interdependencies are required. So a tool is need to give deep insights on software interdependencies that occur during execution on multicore system. SLX is multicore development tool that provides software execution insights into hardware and software interdependencies.It allows for code architecting and refactoring to achieve the most efficient utilization of CPU, DSP, FPGA and other acceleration engines on multicore systems[].

SLX also can be use for software/hardware exploration. Current generation tools are not able to provide exploration to today's complex heterogeneous SoCs. These SoCs are big unimaginable challenges with present available tools. In current market of SoCs, SoCs with increasing number of cores, acceleration engine, decreasing the chip size are available in lower cost and high performance with less power consumption. In such wide varieties of SoCs, it is more challenging to fing optimal target platform for developing application. To select correct SoC for application requires clarity of software and hardware interaction. This interation can be useful to minimize disagreement and maz=ximize the performance.

SLX features behavioral model for modeling multicore SoCs which made of co-processors, graphical processing units(GPUs), field programming gate arrays (FPGAs), communication buses, inter-connectivity, memory hierarchy and clusters. SLX system models provide rapid profile change with respect to hardware variance to facilitate hardware investigation for application software. SLX is able to predict the system behavior and memory performance based on simple and high level application model.

SLX analyze behvior of multiple application on target platform and calculate the intercommunication delays. It also provides static and dynamic source code analysis, data and control dependencies. SLX also provide optimization to optimize the distribution of application to hardware and identification of blocking dependencies and control. Optimizations are driven by performance, power, and memory requirements for given combinations of FPGAs, GPUs and cores. SLX support multiple programming languages to create high level application from scratch directly from your source code. It also support command line interface on windows and linux platforms.

Silexica provides wide range of hardware platforms to analyze the behavioral of application software on different hardware platforms by measuring computing power, inter-connectivity, number of cores, delays and many more factors. It support software developers to understand the behavior of application on hardware platforms which may not be available in market or in still in prototype phase.

3. Basics

In this section we will go through basics like performance measurement concept, tools to measure performance, LLVM, instruction set simulator(ISS) and basic learning techniques to understand the thesis. Performance of software on embedded hardware is essential in embedded product development. The measure question is how to measure performance of software on hardware. Here software maybe an application or operating system. Performance of application or operating system can be analyzed and improved using performance monitoring features provided by embedded hardware. Now every modern hardware processor has Performance Monitoring Unit(PMU) on chip. PMU is on-chip hardware device that monitors the micro-architectural events. Events like cache hits, cache miss, cycles etc. Data collected from PMU can be use to observe the behavior of application or operating system on hardware processor. This collected data can be use to improve the performance.

Performance monitoring events can be categorized in two types, one is hardware events and second is software events. Hardware events contain the events related to hardware like cycles, instructions, cache miss, cache hit etc. Where software events are related to software like page fault, segmentation fault, context switch etc. Performance monitoring hardware consists two components, performance event select registers and event counters. Performance event select registers are configuration registers to control what events to be monitored and how to monitor. Event counters registers, those count the number of events based on event select registers configuration. Also there are two approaches for performance monitoring, one is counting and other is event based sampling. Following figure **??** shows the block diagram of PMU in ARM Cortex A53 processor.

3.1. Tools for Performance Measurement

To evaluate the performance of software on hardware, variety of tools are available. These tools provide high level interface and analysis capabilities for performance monitoring hardware. The Performance Application Programming Interface (PAPI) tool suite [papi link] provides a common interface to performance monitoring hardware for many different processors like Alpha, ARM, MIPS, Pentium, PowerPC and etc. VTune Performance Analyzer [link for it] from Intel support performance mon-

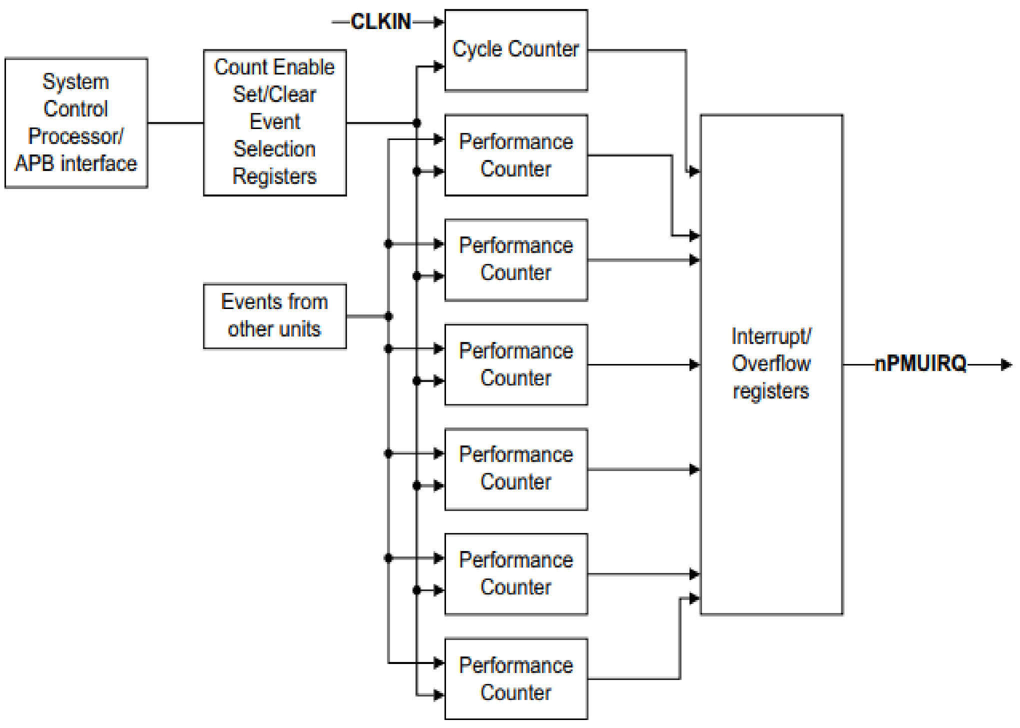
CLKIN
Cycle Counter
System Control Processor/ APB interface
Count Enable Set/Clear Event Selection Registers
Performance Counter
Events from other units
Interrupt/ Overflow registers
nPMUIRQ

Figure 3.1.: Block diagram of PMU in ARM Cortex A53 processor

itoring on Intel's processor. PERF is performance analyzing tool in Linux, available from Linux kernel version 2.6.31 [link for perf]. OProfile[link for OProfile] is system-wide statistical profiling tools for Linux to measure the performance of underlying hardware processor. ARM's Streamline Performance Analyzer provide in depth look of software performance on ARM processors. In this thesis, we are going to see tools like PAPI, PERF and Streamline Performance analyzer to evaluate the performance of software. In this thesis, we are focusing on hardware events and the performance data provided by these performance measurement tools measured on Raspberry Pi 3B board, which includes ARM Cortex A53 processor.

3.1.1. PERF

PERF is performance analyzing tool in Linux operating system and it is available from version 2.6+ of Linux kernel. PERF can accessed by userspace controlling utility, named 'perf'. PERF command line interface on Linux provides number of subcommands. PERF provides statictical data performance data of Linux kernel and user application. PERF supports hardware performance counters, software performance counters and tracepoints. PERF is based on 'perf_events' interface.

PERF Linux tool supports wide majority of events to provide the performance data. This tool can measure events from different sources using kernel interface. One source of events are kernel counters and these events are called software events like context switches, page-fault, alignment fault, emulation fault and etc. Another source of events is processor itself. PERF reads the events from hardware performance counter of PMU of processor. PERF provides the list micro-architectural events supported by it such as number of cycles, L1 cache miss, total instructions, branch mispredicted and many more. Finally, the tracepoint events which are provided by kernel through 'ftrace' infrastructure.

One of the main feature of the PERF is that it allowed to select the specific core/s of processor using processor-wide mode. The performance data of specific core/s can be collected for given application or operating system. Which had very significance in this thesis. It also creates the report of the collected sample data in file called 'perf.data'.It also possible with PERF to go more in granularity of instruction level with 'perf annotate'. PERF also provide the live analysis like Linux top command with processor wide feature. it is possible to measure the performance of application in userspace as well as kernel space.

PERF provides subcommands on command line interface such as 'stat, record, report', etc. It is possible to get the list of software and hardware events supported by PERF using 'list' command for running embedded hardware processor. Following is the PERF usage on Raspberry Pi 3B board with Raspbian, Linux based operating system and we are measuring the performance of basicmath application from Mibench suite[] for single core i.e. second core of Cortex A53..

```
$ sudo perf_3.16 state -e cycles,instructions,cache-misses,
 branches,branch-misses -C 2 ./runme_large.sh

# started on Wed Apr 4 14:02:59 2018

Performance counter stats for 'CPU(s) 2':
1,654,827,425    cycles
1,397,787,969    instructions # 0.84 insns per cycle
3,528      cache-misses
94,047,959     branches
2,669,120     branch-misses # 2.84% of all branches
2.445278109 seconds time elapsed
```

3.1.2. PAPI

PAPI is abbreviation for Performance Application Programming Interface. PAPI project is developed at University of Tennessee's Innnovative Computing Laboratory in Computer Science Department. This project was created to design, standardize, and implement a portable and efficient Application Programming Interface (API) to access the hardware performance counters found on most modern microprocessors[papi user guide]. PAPI provides an API to read hardware performance counter values for application running on processor. This data can be use to improve the performance of application or code. Similar kinds of APIs are available but PAPI provides stable API, documentation and upgrade it with features and different platforms.

PAPI provides solid base for cross-platform tools. It provides standardize API. It is easy to use and freely available. It provides consistent interface and methodology to tool designers as well as application engineers to collect data from performance counters of PMU. It provides near real time performance data between application and processor events. PAPI provides operating system independent access to hardware performance counters.

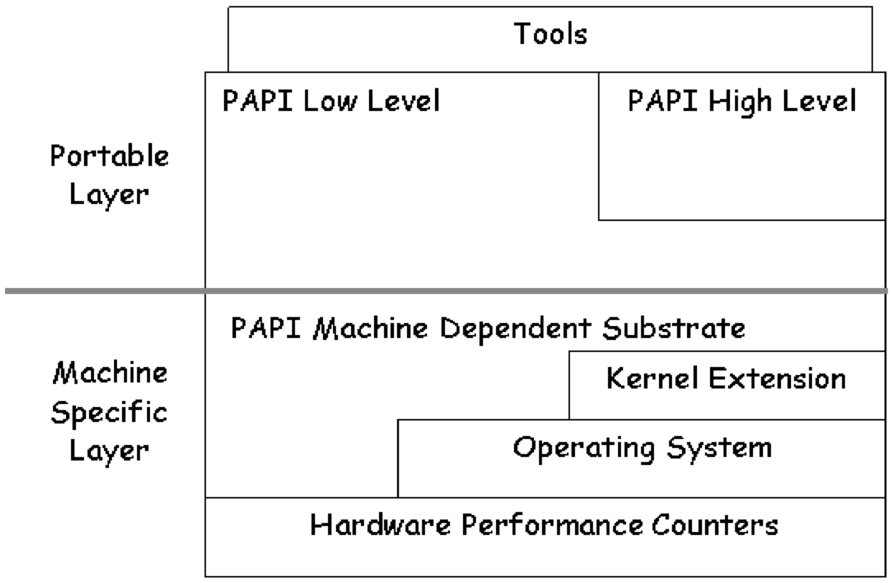

Figure 3.2.: Internal architecture of PAPI

Above figure **??** shows the internal architecture of PAPI. The architecture is divided into two parts, portable layer and machine specific layer. Portable layer has low level, high level and machine independent support functions. Whereas machine dependent layer consists of machine dependent functions and data structures. Layer also consists kernel extensions and operating system calls or assembly language to retrieve the data from counters of PMU. PAPI provides two interfaces, High level API and Low level API. High level API is easy to use and require less set-up but it has higher overhead and less flexibility. Considering that Low level API provides fine grained measurements, more control to interface and increase the functionality and efficiency.

PAPI also provides command line interface event queries for existence of native or preset event to give more light about hardware supports specific events. Following code shows the PAPI API calls in code.

```
#include <papi.h>
#define NUM_EVENTS 2

int main()
{
int Events[NUM_EVENTS] = {PAPI_TOT_INS, PAPI_TOT_CYC};
long_long values[NUM_EVENTS];
/* Start counting events */
if (PAPI_start_counters(Events, NUM_EVENTS) != PAPI_OK)
  handle_error(1);
for (int i=0;i<1000:i++);
if (PAPI_read_counters(values, NUM_EVENTS) != PAPI_OK)
  handle_error(1);
if (PAPI_stop_counters(values, NUM_EVENTS) != PAPI_OK)
  handle_error(1);
}
```

Following stats are showing the performance of Basicmath application from MiBench suite[]. Performance data is collected using PAPI API on Cortex A53 on single core that is core 2.

```
$sudo ./runme_large.sh
cycles:  1653593201,
instructions: 1397640501,
L2 cache miss: 2178,
branches: 137950307,
branch misses: 2668136
```

3.1.3. Streamline Performance Analyzer

Streamline Performance Analyzer is system-wide visualizer and profiler tool from ARM for ARMv7 and ARMv8 hardware targets. It generates the processor profile by reading the program counter at regular sample and collect the information about where processor spends its time. It collects the information of micro-architectural events from PMU of target hardware. Streamline gathers the performance data by communicating with software agent on target hardware. Streamline uses two communications agents, gator and barman. Gator is use for Linux targets whereas barman used for bare metal targets.

In Streamline, best requisite set of hardware performance counters are used by default. But it is possible to change the configuration of hardware counters using Counter configurations dialog, which contains the list of available hardware performance counter for given specific target hardware. Streamline collects data in terms of samples and there are two modes of collecting samples in Streamline. One is fixed rate, where samples are collected in fixed intervals of time and this option is default in sampling method. Other method is Event-based sampling, where sampling is done on context switching or when an event occurs number of times.

Streamline provides Graphical User Interface (GUI). It shows captured data graphically. Streamline provides the functionality to capture data with counter configuration, start and stop of new captures of performance data. It provides live capture where in real time performance data is shown graphically in Streamline window for application being measured or profiled. It plots the counters in real time. After completion of the capture,capture information is shown in charts and details panel. Call paths in Streamline displays a hierarchy of functions, thread, processes. It also lists all functions in application called while capture. It also shows the core wise execution of thread and processes. Streamline also provide statistics line by line for source code. It also provides the string annotations and markers features which allows to add context to the information. Following image **??** show the performance data collected for basicmath application from MiBench suite[] on Cortex A53 processor using Streamline performance analyzer.

3.2. Tools for Framework Development

Apart from performance measurement tools, in this thesis we are also using tools which are freely available to use and create the framework for performance measurement. We are going to see LLVM compiler infrastructure project and Instruction Set Simulator(ISS) for create virtual simulation of hardware.

3.2.1. LLVM

LLVM is collection of compiler and toolchain technologies used for development of front end and back end of compiler. Low level toolscahins like assemblers, debuggers, etc. LLVM started as a research project at University of Illinois. Initial motivation of LLVM was investigate dynamic compilation technique for static and dynamic programming languages[llvm.org]. LLVM is written in C++ language and it is designed for programs written in arbitrary programming language. Clang is LLVM native C/C++ compiler which provides more features than GCC and faster than GCC. Over last decade, LLVM is used as common infrastructure to implement

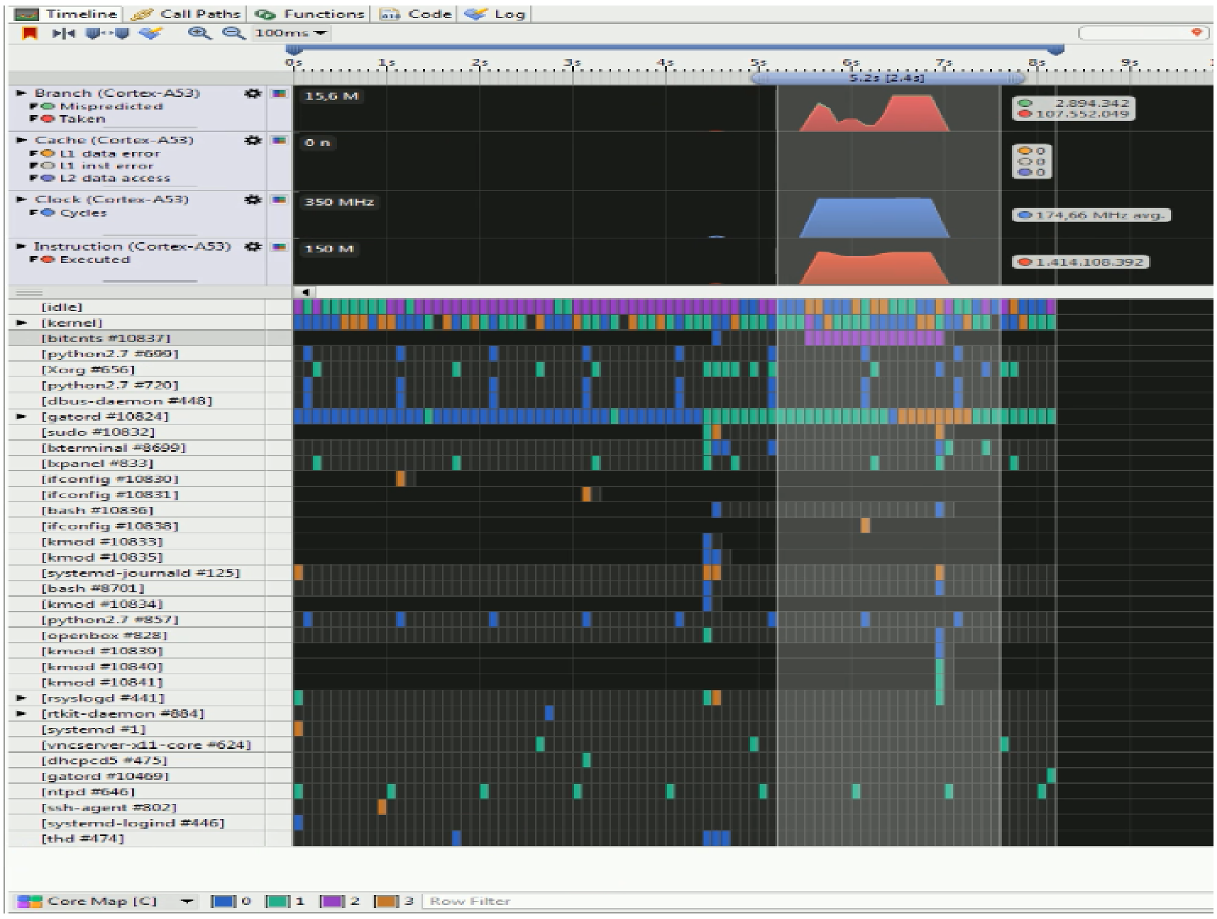

Figure 3.3.: Timeline of basicmath application using Streamline Performance Analyzer

runtime compiled languages, family of languages supported by GCC, Java, .NET, Python, Ruby, Scheme, etc.

Traditional and popular design for compiler is divided into three major components, those components are the front end, the optimizer and the backend. The frontend component perform parsing of source code, checking the errors and represent the input code. The optimizer component try to improve the code by eliminating redundant computations and it is independent of language and target hardware architectures. The backend component converts the code into machine code compatible with target hardware architecture. One of the major implication of this design is that retargetability. If compiler uses common code representation for it's optimizer then backend can written for any target hardware architecture and frontend can be written for any source language. Which eliminate the wide number of compilers for each source language and for each hardware architecture. One of the successful implementation of this mode is GCC. It supports many frontends and backends and have wide community of contributors.

But there are limitations to this traditional approach. They are designed as monolithic applications for example GCC. Some pieces of libraries can not be reused. It has uncontrolling use of global variables, weakly enforced invariants, poorly-designed data structures, sprawling code base, and the use of macros that prevent the codebase from being compiled to support more than one front-end/target pair at a time. GCC suffers from layering problems and leaky abstractions: the back end walks front-end ASTs to generate debug info, the front ends generate back-end data structures, and the entire compiler depends on global data structures set up by the command line interface [chris lattner].

LLVM uses Intermediate Representation(IR) for code representation in compiler infrastructure. For any source code at frontend is converted into IR code and given to optimizer and output of the optimizer is also IR code which is faded to backend to create target machine code. You can see this behavior in following figure **??** of three phase structure of LLVM compiler.

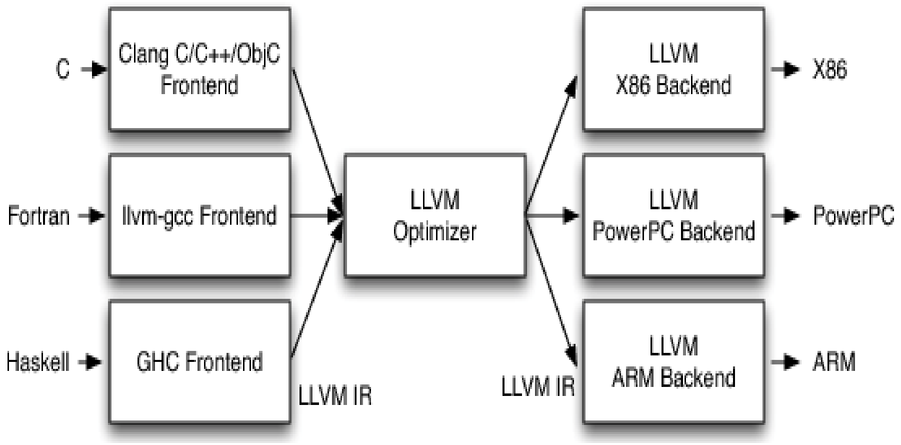

Figure 3.4.: LLVM's three phase design

LLVM IR is low-level Reduced Instruction Set Architecture(RISC) like virtual instruction set. We can see in this following example.

```
define i32 @add1(i32 %a, i32 %b) {
entry:
  %tmp1 = add i32 %a, %b
  ret i32 %tmp1
}

define i32 @add2(i32 %a, i32 %b) {
entry:
  %tmp1 = icmp eq i32 %a, 0
  br i1 %tmp1, label %done, label %recurse

recurse:
  %tmp2 = sub i32 %a, 1
  %tmp3 = add i32 %b, 1
  %tmp4 = call i32 @add2(i32 %tmp2, i32 %tmp3)
  ret i32 %tmp4

done:
  ret i32 %b
}
```

This is the LLVM IR for following C code.

```
unsigned add1(unsigned a, unsigned b) {
  return a+b;
}

unsigned add2(unsigned a, unsigned b) {
  if (a == 0) return b;
  return add2(a-1, b+1);
}
```

LLVM IR representation help to understand the code at machine level. LLVM also provide optimization at IR representation. LLVM is collection of libraries. LLVM also offers optimizations in terms of passes. LLVM passes is written in C++ class. These passes traverse portion of program to either collect the information or transform the program. In this thesis we are going make use LLVM basic block pass to collect the performance data at basic block level. We will see it in detail in further chapters. LLVM also provide retargetablity for various source code languages and target architectures. It also provides capabiites by modular design such as when and where each phase of compiler runs, unit testing the optimizer and automatic test case reduction.

3.2.2. Instruction Set Simulator

Instruction Ste Simulators(ISS) is simulation model and mostly coded into high level programming languages. This simulation models reflects the behavior of actual microprocessor/microcontroller. This simulation models can mimic various hardware architectures. ISS mimics the behavior by reading the assembly instructions and maintaining the processor registers. ISS are widely used for simulating the machine code of another hardware to check compatibility, monitoring and testing of machine code, testing and debugging of machine code, and to improve the performance. ISS are important in early embedded product release in market. It can help to simulate target hardware if it is not available in market or in prototype phase of development. Disadvantage of ISS is that they costly and oftenely they are slow.

3.3. Basic Learning Techniques

It is possible to get mathematical relationship between input and respective output by observing the behavior. Mathematical statement or equation is formed using this one to one, one to many, many to one or many to many. Mathematical relationship can be formed by creating with support the dataset of this relationship. The dataset contains the history of wide range data of input and output. This analysis of of input output model helps to create statistical model that can be useful predict output for new every input using learned mathematical model and accurate dataset.

This relationship between input and output is learned using Machine Learning. Machine learning techniques allows to form this relationship and create dataset to predict the result for every new input. This is done by without any explicit programming. Only important thing is create dataset of input and output that helps to understand the behavior of relationship between them. And mathematical model can be created to learned this relationship and used to predict the output. There basically three main machine learning techniques are available,they are, supervised learning, unsupervised learning and regression learning. Again in these three main learning techniques many sub-techniques are available. Obtained dataset in machine learning filed is called as training data. Which helps to understand the behavior. Machine learning technique is any application is depending upon relationship between input and output and use case. In some cases relationship is linear or in some cases data is in number of chunks.

3.3.1. Regression

Regression is most popular technique in the field of machine learning, in which linear regression and logistic regression are most widely used and known to data analytics. There are also regression techniques are available. Regression is popular statistical techniques use for application of predictive modeling. In regression technique, relationship model is formed in between dependent variable and independent variable. These variable can be one or many. For example in annual sale prediction, sales is dependent variable whichis going to predicted whereas factors affecting the sales are independent variables, these can be one or many.

In linear regression techniques, dependent variable is continuous in nature with respect to independent variable. And relationship between these variable is assumed to be linear in nature. Logistic regression method is used for classification. Classification can binary or multi-class classification. There are also different techniques such as lasso regression, ridge regression, and many more techniques are available. These techniques are depend upon application and analysis of the data. We will see in more detail in upcoming chapters.

4. Design and Implementation Storyboards

TBD

5. Safety Competency

TBD

6. Performance Evaluation

TBD

7. Future Outlook

TBD

8. Conclusion

TBD

Bibliography

List of Abbreviations

API	Applicaton Program Interface
ASIL	Automotive Safety Integrity Levels
BPF	Berkeley Packet Filter
Bash	Bourne-again shell
BIOS	Basic Input/Output System
BLOB	Binary Large Object
BMP	Bitmap
CISC	Complex Instruction Set Computing
CPU	Central Processing Unit
DLL	Dynamic Link Library
DMA	Direct Memory Access
DOS	Disk Operating System
DRAM	Dynamic Random Access Memory
eBPF	Linux Enhanced BPF
FIDL	Fuchsia Interface Description Language
FTP	File Transfer Protocol
GPU	Graphics Processing Unit
GUI	Graphical User Interface
GUID	Globally Unique Identifier
IDE	Integrated Development Environment
IP	Internet Protocol
IPC	Inter Process Communication
ISO	International Organization For Standardization
KDE	K Desktop Environment
KOID	Kernel Object ID
LK	Little Kernel
MBR	Master Boot Record
OS	Operating System
QEMU	Quick Emulator
RAM	Random Access Memory
RTE	Runtime Environment
SDK	Software Developement Kit
SSH	Secure Shell
SSL	Secure Sockets Layer
TOML	Tom's Own Minimal Language
VM	Virtual Machine
UDP	User Datagram Protocol
URL	Uniform Resource Locator
VGA	Video Graphics Array

VPI	Virtual Path Identifier
vDSO	Virtual Dynamic Shared Object
WIP	Work in Progress

List of Figures

List of Tables

A. AAAA

TBD

B. BBBB

TBD

C. Performance Evaluation

TBD