Department of Computer Science and Engineering
Indian Institute of Technology,
Kharagpur

Report
For

Reverse Engineering Sorting Algorithm Using Performance Counters

Submitted by

Nikhil Kumar Singh (17CS60R63)

SUPERVISED BY

Debdeep Mukhopadhyay

November 14, 2018

Certificate

This is to certify that the thesis titled **Reverse Engineering Sorting Algorithm Using Performance Counters** submitted by **Nikhil kumar singh (17CS60R63)** to the Department of Computer Science and Engineering is a bonafide record of work carried out by him under my supervision and guidance. The thesis has fulfilled all the requirements as per the regulations of the Institute and, in my opinion, has reached the standard needed for submission.

Dr. Debdeep Mukhopadhyay
Professor,
Computer Science and Engineering
Indian Institute of Technology, Kharagpur
November 14, 2017

Declaration

This is to certify that

1. The work contained in this thesis is original and has been done by myself under the general supervision of my supervisor.
2. The work has not been submitted to any other Institute for any degree or diploma.
3. I have followed the guidelines provided by the Institute in writing the thesis.
4. I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.
5. Whenever I have used materials (data, theoretical analysis, and text) from other sources, I have given due credit to them by citing them in the text of the thesis and giving their details in the references.
6. Whenever I have quoted written materials from other sources, I have put them under quotation marks and given due credit to the sources by citing them in the text of the thesis and giving their details in the references.

Nikhil Kumar Singh
Department of CSE
IIT Kharagpur
Date : November 2018

Abstract

Name of the student: **Nikhil kumar singh** Roll No: **16CS60R8**3

Degree for which submitted: **Master of Technology**

Department: **Department of Computer Science and Engineerin**g

Thesis title: **Reverse Engineering Sorting Algorithm Using Performance Counters.**

Thesis supervisor: **Prof. Debdeep Mukhopadhyay**

Month and year of thesis submission: **November 12, 2017**

Sorting operation are fundamental and are often repeatedly used in many large-scale scientific and commercial application. Because of this prominence, any effort to maximize the efficiency in these programs requires ensuring that the sorting algorithm used have been correctly selected and are precisely implemented. Restructuring standard efficient sorting algorithm (such as mergesort and quicksort) to exploit cache locality has proven to be an effective approach for improving performance on high-end system. In this thesis we are comparing sorting algorithm on the basis of their hardware performance value like cache misses, branches misses and on the basis of this we will we will try Restructure sorting for more efficiency. Also we are reverse engineering sorting algorithm, given an unknown sorting algorithm running an a end-system we will find which sorting algorithm it is on the basis of it’s HPCs values.

Acknowledgment

I would like to thank my supervisor Debdeep Mukhopadhyay, professor, dept. of CSE, IIT Kharagpur for his valuable guidance and consistent support. I would like to thank Mr. Manar Alam, Reserch Scholar, dept. of CSE, IIT Kharagpur, who has been very much involved in this project. Without his tremendous support, this work might not be finished so early. I am grateful to my parents for their unconditional love and supports. I have learned a lot from the professors and my friends in IIT Kharagpur.

Thanks to all of them.

Contents

Introduction

1.1 Background

1.1.1 Hardware performance counter

Hardware Performance Counters (HPCs) are a set of special purpose registers, which are present in most of the modern microprocessor’s Performance Monitoring Unit (PMU). These registers can be programmed to store the number of occurrences of different types of hardware and software events related to the execution of a program, such as cache misses, retired instructions, retired branch instructions, and so on. HPCs were primarily designed to debug the performance of complex software systems, but currently, they are widely used for collecting the run-time behavioral information of software execution. HPCs work along with the event selectors, which specify the hardware events to be monitored and a digital logic which increments a counter based on the occurrence of the specified hardware events. These performance counters can be accessed very fast without affecting or slowing down any software execution. Moreover, to get access to these registers no source code modification is required. Hardware Performance counters has been used in many recent literatures [18–21] for dynamic profiling of a system. The most useful mode of operation of PMUs is the interrupt-based mode. The central working principle behind this mode of operation is, a system interrupt is generated when a specified event occurs more than or equal to a predefined threshold value or a preset amount of time has elapsed. This mode of operation makes both event-based and time-based sampling possible. High-level libraries like PAPI [22], OProfile [23] provide interfaces to HPCs. Linux perf [24] among them is a widely used new implementation of performance counters support for all Linux 2.6+ based systems, which we can access from user-space. This tool is capable of providing per-process, per-CPU, and systemwide statistical profile. We used this tool for our experimentation purpose. Perf tool is based on Linux perf event open() system call, which can be used to profile system in very low granularity. Almost every popular operating systems have HPC-based profilers, though the type and number of hardware events may vary across different Instruction Set Architectures (ISA) [25]. Most of the modern processors may offer thousands of hardware and software events to monitor, however, only a selected few of them can be monitored in parallel due to the limitation in the number of built-in HPC registers. Intel 64 and IA-32 architectures [25] provide facilities for monitoring performance via a Performance Monitoring Unit (PMU). There are more than 100 performance events which can be monitored to measure the performance of a program. Since we target micro-architectural attack in our study, we considered the hardware events which are more likely to be affected by these attacks. Micro-architectural attacks like branch prediction based attack work in such a way that we can observe its influence in hardware events such as Branch Instruction Retired and Branch Misses Retired. The cache-based attacks will affect the hardware events such as LLC References, and LLC Misses more than the other events. Some other attacks may influence some other hardware counters also.

In our experimentation system, we were able to monitor six events , which we discuss in brief -

1. Branch Instruction Retired : This event counts the number of branch instructions at retirement.

2. **Branch Misses Retired** : This event counts the number of mispredicted branch instructions at retirement.

3. Cache References : This event counts the number of requests originating from the core that references a cache line in cache.

4. Cache Misses : This event counts the number of cache miss conditions for references to the cache.

5. Instruction Retired : This event counts the total number of instructions at retirement.

8. Bus Cycles : This event counts the number of bus cycles required with the system clock signal on the core running.

1.1.2 Perf Tool Commands

Useful perf commands include the following:

perf stat

This perf command provides overall statistics for common performance events, including instructions executed and clock cycles consumed. Options allow selection of events other than the default measurement events.

perf record

This perf command records performance data into a file which can be later analyzed using perf report.

perf report

This perf command reads the performance data from a file and analyzes the recorded data.

perf list

This perf command lists the events available on a particular machine. These events will vary based on the performance monitoring hardware and the software configuration of the system.

We can use perf help to obtain a complete list of perf commands. To retrieve man page information on each perf command, use perf help *command*.

1.3 Problem Definition

To Compare and analyze performance of Different Sorting algorithm Based On their HPCs values.

2 Literature survey

2.1 Reverse Engineering Hardware Page Table Caches Using Side-Channel Attacks on the MMU.[4]

As software is becoming harder to compromise due to the plethora of advanced defenses attacks on hardware are instead becoming an attractive alternative.These attacks range from compromising the system using the Rowhammer vulnerability [24, 26, 4, 25] and using side channels for breaking address space layout randomization as page table caches. Since the information on the size of TLB and the LLC is available, we can use AnC to reverse engineer the properties of the page table caches that are of interest to attackers, like their internal architecture and size. Recent hardware-based attacks that compromise systems with Rowhammer or bypass address-space layout randomization rely on how the processor's memory management unit (MMU) interacts with page tables. These attacks often need to reload page tables repeatedly in order to observe changes in the target system's behavior. To speed up the MMU page table lookups, modern processors make use of multiple levels of caches such as translation lookaside buffers (TLBs), special-purpose page table caches and even general data caches. A successful attack needs to flush these caches reliably before accessing page tables. To flush these caches from an unprivileged process, the attacker needs to create specialized memory access patterns based on the internal architecture and size of these caches, as well as on how the caches interact with each other. While information about TLBs and data caches are often reported in processor manuals released by the vendors, there is typically little or no information about the properties of page table caches on different processors. In this paper, we retrofit a recently proposed EVICT+TIME attack on the MMU to reverse engineer the internal architecture, size and the interaction of these page table caches with other caches in 20 different microarchitectures from Intel, ARM and AMD. We release our findings in the form of a library that provides a convenient interface for flushing these caches as well as automatically reverse engineering page table caches on new architectures.

2.2 Reverse Engineering Intel Last-Level Cache Complex Addressing Using Performance Counters[3]

Cache attacks, which exploit differences in timing to perform covert or side channels, are now well understood. Recent works leverage the last level cache to perform cache attacks across cores. This cache is split in slices, with one slice per core. While predicting the slices used by an address is simple in older processors, recent processors are using an undocumented technique called complex addressing. This renders some attacks more difficult and makes other attacks impossible, because of the loss of precision in the prediction of cache collisions. In this paper, we build an automatic and generic method for reverse engineering Intel's last-level cache complex addressing, consequently rendering the class of cache attacks highly practical. Our method relies on CPU hardware performance counters to determine the cache slice an address is mapped to. We show that our method gives a more precise description of the complex addressing function than previous work. We validated our method by reversing the complex addressing functions on a diverse set of Intel processors. This set encompasses Sandy Bridge, Ivy Bridge and Haswell micro-architectures, with different number of cores, for mobile and server ranges of processors. We show the correctness of our function by building a covert channel. Finally, we discuss how other attacks benefit from knowing the complex addressing of a cache, such as sandboxed rowhammer.

3.1 System Description

3.1.1 Hardware

Configuration of System is 5th generation Intel core-i7 Processor. 8 GB DDR4 RAM.

3.1.2 Software

Operating system

We are using Ubuntu 18.04 on PCs. Ubuntu 18.04 is open-source and supports a large variety of HPCs tool

perf tool

perf (sometimes called perf_events or perf tools, originally Performance Counters for Linux, PCL) is a performance analyzing tool in Linux, available from Linux kernel version 2.6.31. Userspace controlling utility, named perf, is accessed from the command line and provides a number of subcommands; it is capable of statistical profiling of the entire system (both kernel and userland code). It supports hardware performance counters, tracepoints, software performance counters (e.g. hrtimer), and dynamic probes (for example, kprobes or uprobes) In 2012, two IBM engineers recognized perf (along with OProfile) as one of the two most commonly used performance counter profiling tools on Linux.

4 Methodology

4.1 Sorting

Sorting algorithms are used extensively in a variety of applications, making it one of the most representative algorithms in general purpose computing. Sorting data is required in high-performance computing (HPC) systems or large servers to rank Internet pages or to mine data. Sorting is also used in physical simulations algorithms and artificial intelligence . Safety-critical applications, such as aerospace or automotive ones, use sort algorithms to encode video, track and classify star luminosity or detect pedestrians or other objects.

4.2 Implementation

We took five sorting algorithm - *merge sort, quick sort, heap sort, selection sort, insertion sort*. We selected following six Hardware performance counters- *Branches, branch_misses, bus-cycle, cache misses, cache references, cycles.*

Fore each hardware Counters we collected 500 values for each sorting algorithm. For example for selection sort we run the following command.

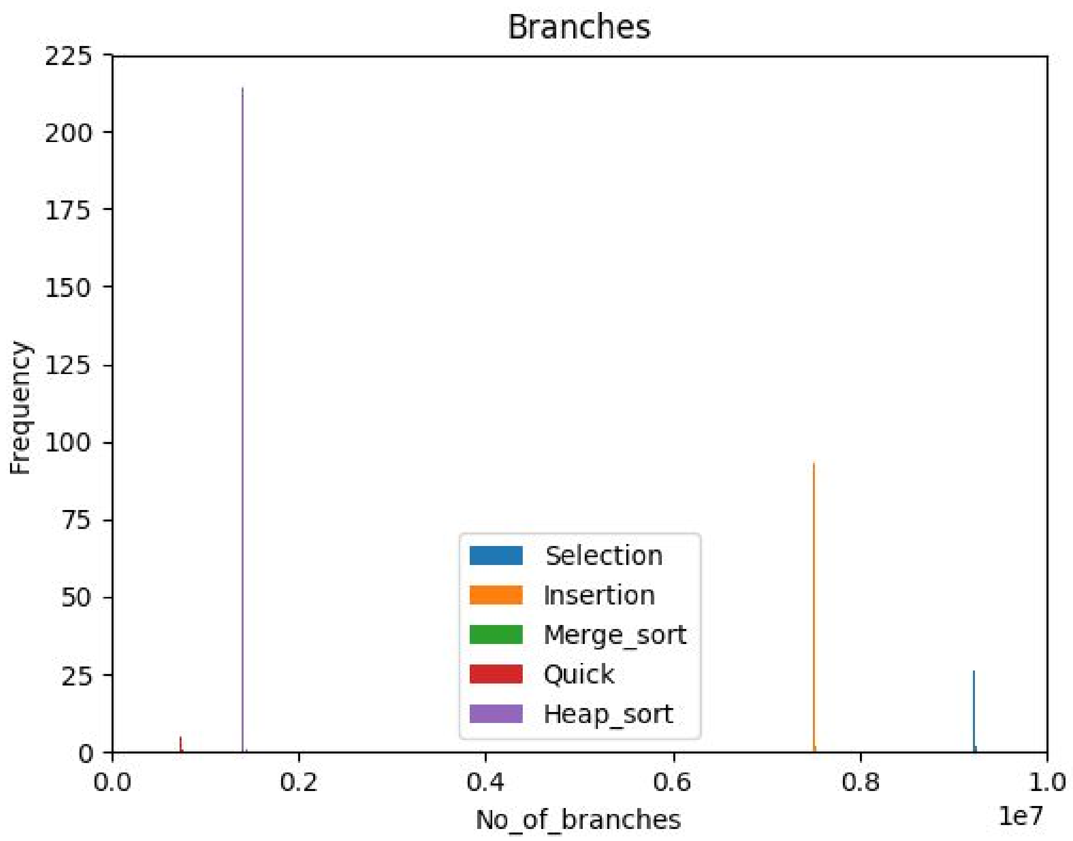
```
g++ selection.cpp;

for i in {1..500} ;

do sudo perf stat -o s.txt --append -e branches ./a.out

Done
```

It runs selection sort for five hundred time and collects the Branches value in a text file named s.txt. Similarly for other four sorting algorithm we collects the branches value.We repeat the above steps for other HPCs. then we plotted the histogram for analysing the result.

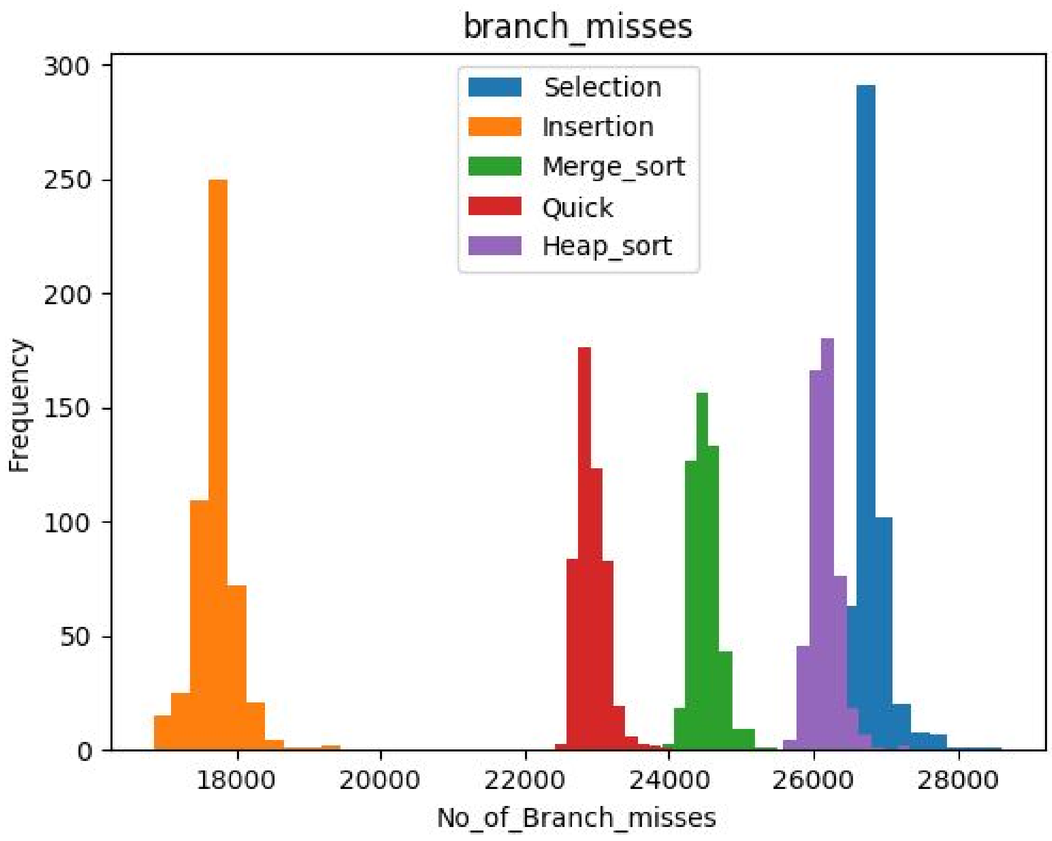
5 Experiments and Result

We get the following Result:

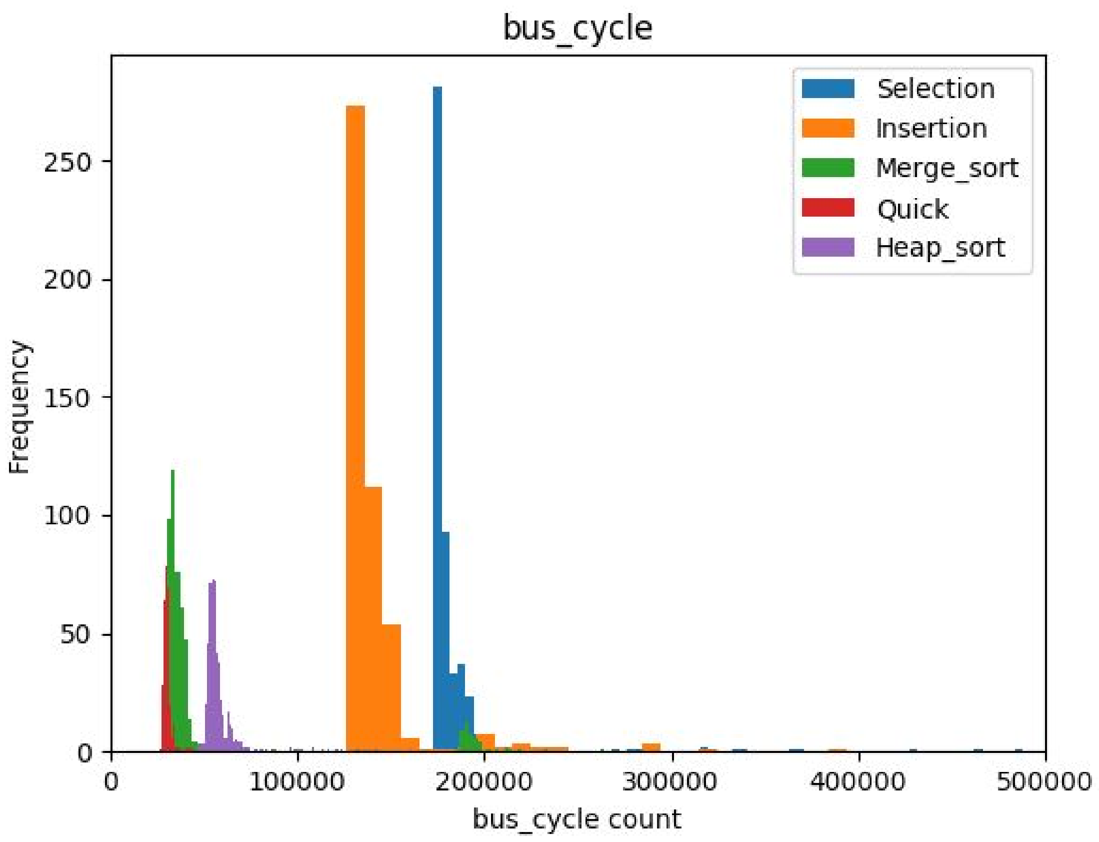
5.1 Branches



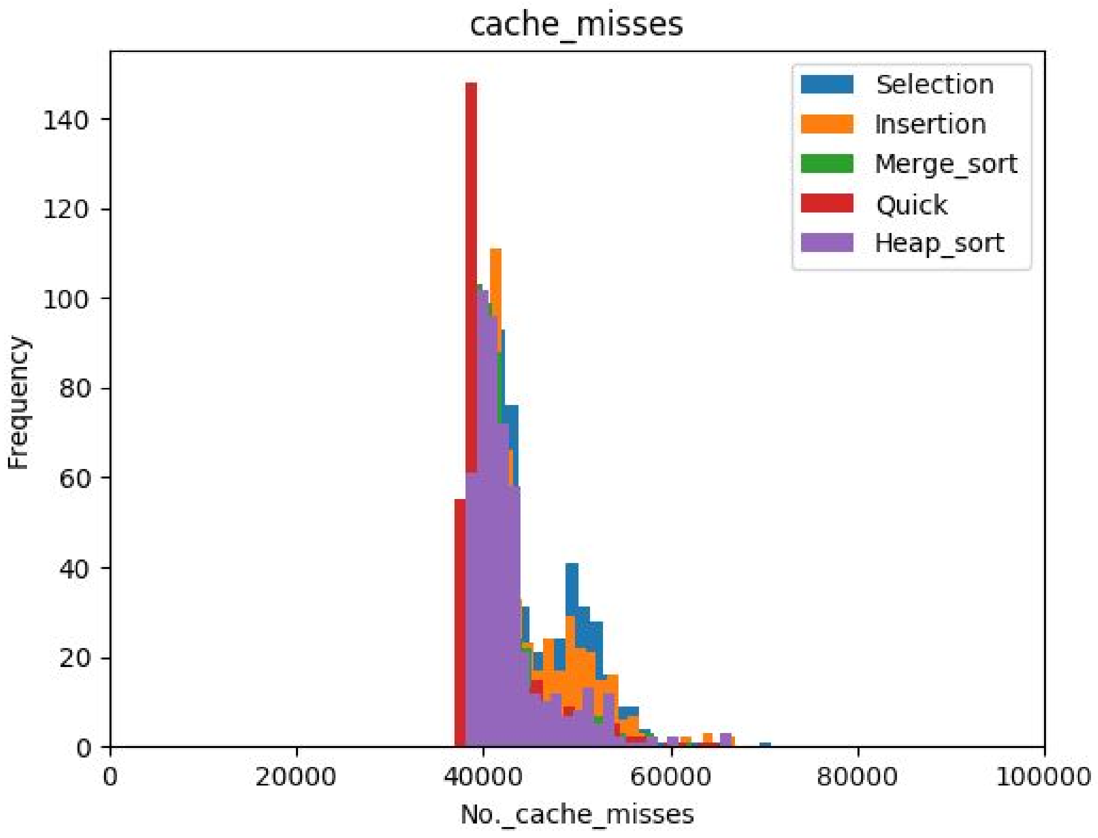
5.2 Branch-misses



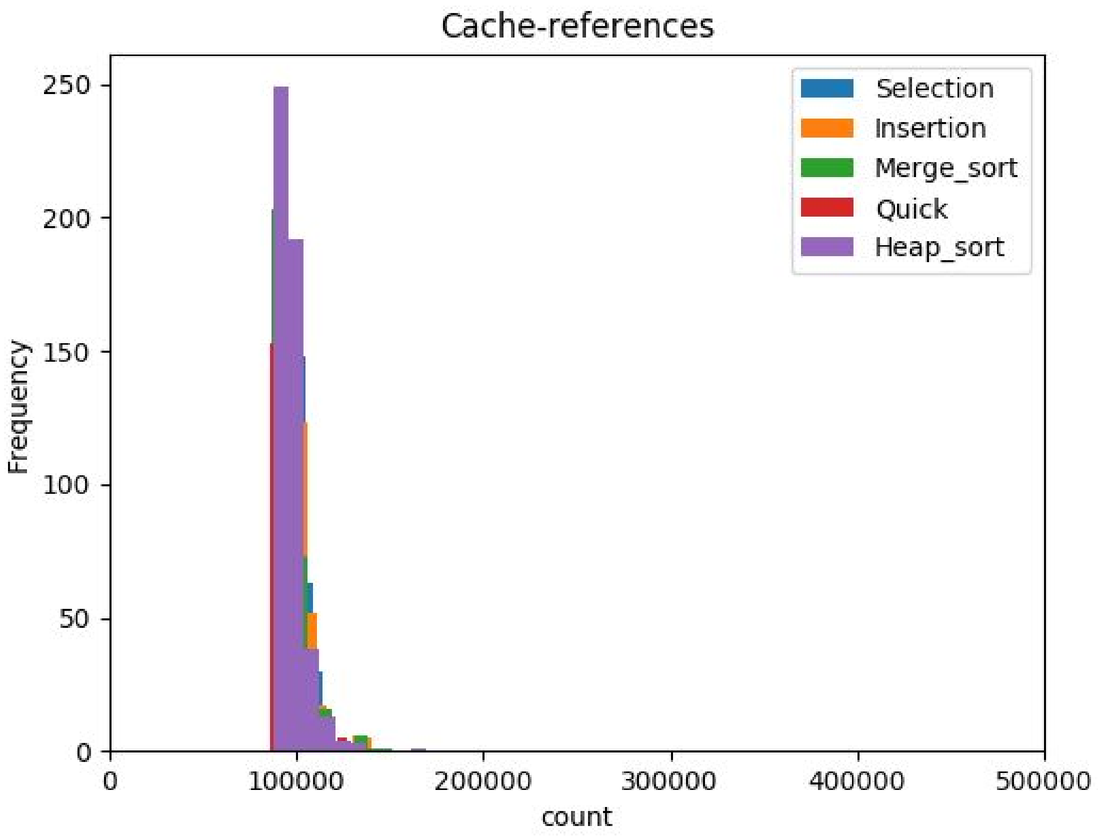
5.3 Bus cycles



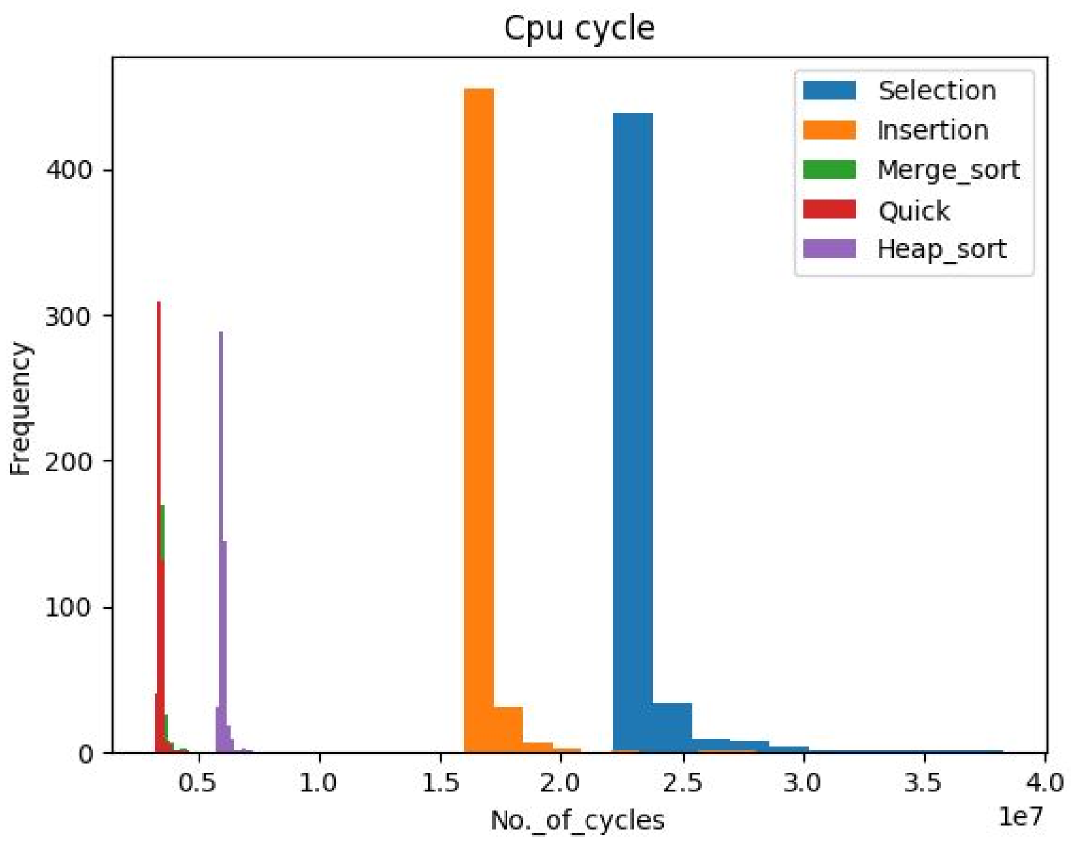
5.4 Cache misses



5.5 Cache references



5.6 Cycles



Cpu cycle
Selection
Insertion
Merge_sort
Quick
Heap_sort
Frequency
0
100
200
300
400
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
1e7
No._of_cycles

6 Conclusion

In this work we captured the hardware performance counter of different standard sorting algorithm and compared them based on their hardware performance counter value. We found that except for cache references and cache misses all other HPCs are distinguable. Also we observe that quicksort is most optimal sorting algorithm

7 Future work

we will introduced a method to reverse engineer Sorting Algorithms, using hardware performance counters.

We will try to Implement sorting in different way, so that it is optimal and reverse engineering cannot be applied.

References

1https://pdfs.semanticscholar.org/c419/3d67010a54c386bc9a2c75417cd6b0ad67c9.pdf

2 http://web.cse.ohio-state.edu/hpcs/WWW/HTML/publications/papers/TR-00-6.pdf

3https://pdfs.semanticscholar.org/0204/b810b62ced1b96d337d215572f5a56e05440.pdf?_ga=2.226506200.419825127.1542051089-45807953.1542051089

4 https://www.cs.vu.nl/~herbert b/download/papers/revanc_ir-cs-77.pdf

5 http://cse.iitkgp.ac.in/~debdeep/courses_iitkgp/HardSec/resources/ppt.pdf