International Journal of Reconfigurable and Embedded Systems (IJRES)
Vol. 10, No. 2, July 2021, pp. 137~1498
ISSN: 2089-4864, DOI: 10.11591/ijres.v10.i2.pp137-148

Processor performance metrics analysis and implementation for MIPS using an open source OS

Varuna Eswer and Sanket Dessai
Department of Computer Science and Engineering, M. S. Ramaiah University of Applied Sciences, Bengaluru, India

Article Info

Article history:

Received Jan 29, 2021
Revised May 25, 2021
Accepted Jun 10, 2021

Keywords:

MIPS32
Operating system
Processor performance
TLB
Metrics
Open source

ABSTRACT

Processor efficiency is a important in embedded system. The efficiency of the processor depends on the L1 cache and translation lookaside buffer (TLB). It is required to understand the L1 cache and TLB performances during varied load for the execution on the processor and hence studies the performance of the varying load and its performance with caches with MIPS and operating system (OS) are studied in this paper. The proposed methods of implementation in the paper considers the counting of the instruction execution for respective cache and TLB management and the events are measured using a dedicated counters in software. The software counters are used as there are limitation to hardware counters in the MIPS32. Twenty-seven metrics are considered for analysis and proper identification and implemented for the performance measurement of L1 cache and TLB on the MIPS32 processor. The generated data helps in future research in compiler tuning, memory management design for OS, analyzing architectural issues, system benchmarking, scalability, address space analysis, studies of bus communication among processor and its workload sharing characterization and kernel profiling.

Corresponding Author:

Sanket Dessai
Department of Computer Science and Engineering
M.S. Ramaiah University of Applied Sciences
#470-P, Peenya Industrial Area, Peenya Second Stage, Peenya Bengaluru (Bangalore)-560058
Email: sanketdessai0808@gmail.com

1. INTRODUCTION

The performance measurement is measured based on the desired events occurring in the system at the time of program execution. The measure of the events and states of the quantum of work done over the processor is called the processor performance. Workload of the processor executions depends on the loading and storing of the data computations and the movement of data in a sequence of operations of the processor components involves pipeline, memory cache and peripherals. When OS is loaded over the processor, the performance of the processor is achieved either with the counters in hardware or/and software or both. The hardware counters which make of physical counters as the peripherals of the processor and is loaded with the measurement counter values for the execution process made of for any application or for an operating system [1]-[3].

The event occurrence associated with the performance measurement and modified code execution are performed through software counters. In software counters the event takes an additional instruction cycle to count the event. Whereas the counters in hardware are non-intrusive on the instruction execution cycles providing advantage for the performance of the system. The biggest disadvantage is the number of hardware counters are limited as compared to software counters. Hardware counters are utilized and focused to fine-

tuning either the operating system or the executing process in the processor performance identified bottlenecks. Whereas the software counters are utilized and focused either for MIPS32 general-purpose performance measurement or for measure a bottleneck specifically of the MIPS32 based on OS.

This measurement is essential for mapping the behavior and efficiency not only of the underlying processor architecture and its associated subsystems, but it is also to measure the performance of the various processes executing on the architecture as well. The evolved understanding aids to monitor the performances, optimize the code, tune the code parameters, and code constructs, model and benchmark the system that comprise of the architecture, its subsystems, and the executing processes; be it the OS or the applications using the OS. Of interest is the software counters as it provides the flexibility to define the measurement framework.

The cache and TLB form an important measurement as it determines the efficiency of the processor in the context of the performance framework. The measurement framework addressed in this paper is to measure the L1 cache and TLB activities on a MIPS32 architecture implementation. The MIPS32 architecture implementation does not have a hardware-based counters; hence, software-based counters have been implemented. The OS manages the L1 cache and TLB on the MIPS32 implementation; hence, the software counters play an important role in measuring the performance of the processor. The OS has been instrumented with software counters to get data on the events associated with the L1 cache and TLB. The data generated by the software counters for L1 cache and TLB events is provided in the ASCII format to enable ease of analysis. A set of thus generated event data over a period has been utilized for generating the histogram for validation purposes.

Software counters are additional code added into the OS; hence, a certain amount of performance drop is expected when compared to an implementation of the OS without software counters. The focus of this performance measurement will be to generate as much as data possible at the first instance, then through analysis of the generated data the instrumentation can be reduced to move the OS closer to the desired performance level. Iterative analysis of the generated data and further tuning of the OS will then be essential.

The challenge offered by the MIPS32 architecture implementation has been the availability of free space on the flash memory. The code thus instrumented necessitates having a small footprint; hence, flash memory space saving techniques was formulated. The OS source code provided by the implementer of the MIPS32 architecture has stripped a general OS suitable for implementation on an embedded system; hence, compatibility with user loadable modules, file system writing, documentation, data extraction through FTP, and command line parameters processing are restricted. These challenges have been addressed to design, develop, and validate an online L1 cache and TLB performance measuring system for the MIPS32 architecture implementation.

2. PROCESSOR PERFORMANCE MEASUREMENT

The study of the project which is developing the concept would be developing the context associated with the MIPS32 processor architecture along its organization includes processor pipeline, memories and different level of caches, performance measurement matrices, data input output methods from the processor implementation board and how the setup of the development system.

2.1. MIPS32 architecture pipeline and cache

The architecture of MIPS32 processor is RISC processor based and they are either types in 32- or 64-bit addressing mode [4, 5]. The speed of the processor depends on the pipeline and caches. Due to pipeline and caches advancement, there are wide applications in workstations to complex embedded systems. The mechanism of the pipeline deals with the division of the workload (instructions and data) into an ordered sequence providing to enable faster turnaround time for execution of the workload without and interlocking of the pipeline and stall within the processor pipeline. Pipeline of MIPS32 processor is shown in Figure 1 and it consists of five stages of pipeline.

A cache helps to access the instructions and data transfer faster to the between the memory and processor. The instructions are residing in the instruction memory and the data is residing in the data memory [4]-[6]. The CPU request the data and the cache provide it if it is available in cache block. If the data is not available in the cache, then the cache is refreshed and invalidated the set of data. This invalidating will force the cache to update the data required for the processor from the main memory through the TLB walk making it write back. As shown in the Figure 1 the representation of instruction cache is Icache and the data cache is Dcache. Both the caches are separated as per Harward architecture of the computer design principle to provide a better performance for instructions and data executions on the CPU avoiding the starvation of the CPU [4].

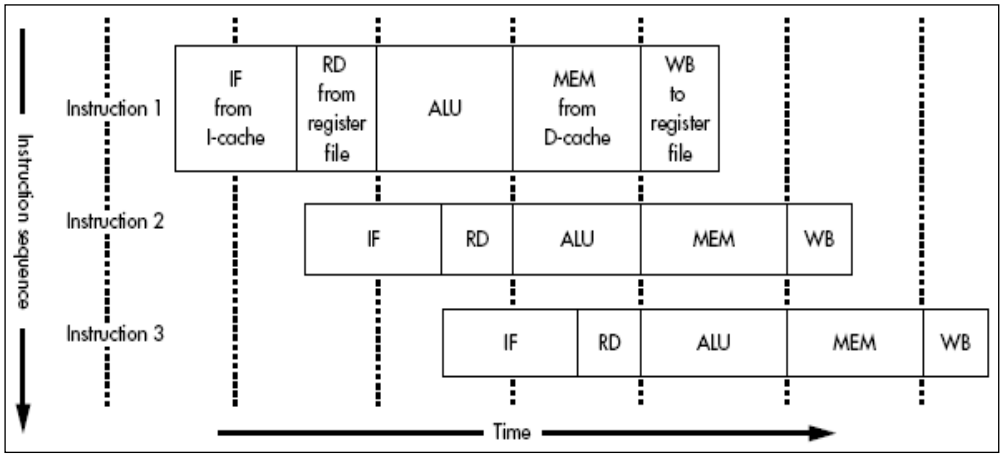

Figure 1. MIPS 5 stage pipeline [4]

2.2. Performance measurement

Performance of the processor depends on gathering events data, occurring data movement in the processor and state transitions. The execution of the instructions depends on the event of data movement. Hence, when the CPU put the request on the cache, the responses of the cache and TLB on the data takes a higher priority as compared with the other operations requested by the CPU. The performance measurement is achieved using hardware counters or software-based counters. As shown in the Figure 2 is a hit-miss of the software counter [6]. The metrics classification is divided into two-categories of base and derived. In base metrics consists of the direct or raw counting of the monitored events, in base of the derived metrics are those arrived at with the combination of two or more metrics of either category such as base category or derived category or a both combination of base and derived category.

The generation of data by the software for the measurement of by the counters is either be archived for long or short-term duration. The data analysis of the quantum data is obtained from the extent of data archival. The begin of the experiment consists of modified source code analysis for metric collection for performance measurement of counter made in hardware or software. The performance measurement framework is designed to perform metric data of high amount varied load on the processor. The metric data analysis of data provides analysis with direction of further code instrumentation or reduction in the code instrumentation for arriving a measurement framework. The instrumentation of code, metric data collection and the analysis process is iterative as shown in Figure 3, until the frozening of the framework for performance measurement.

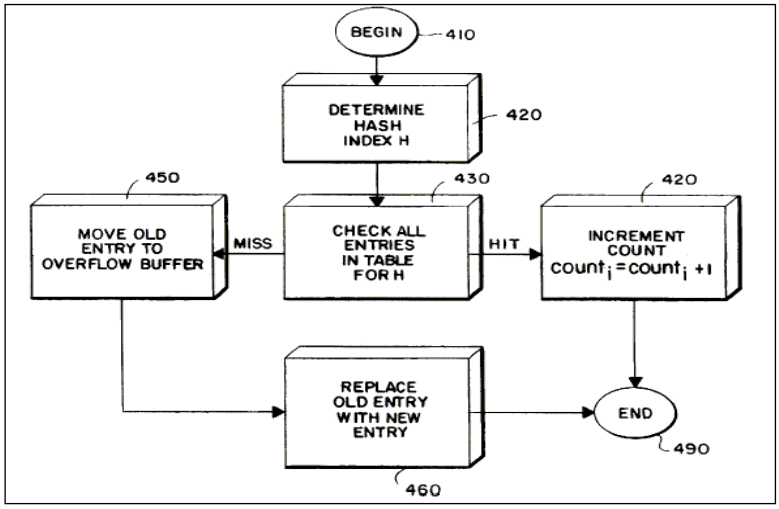

Figure 2. Implementation of software counter [6]

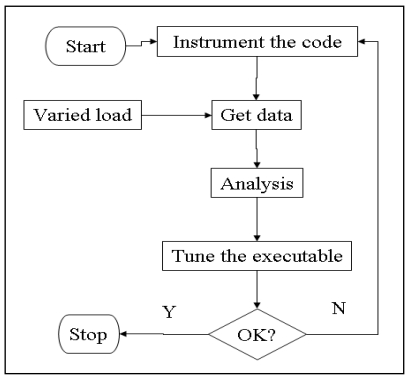

Figure 3. Performance measurement flow chart [6]

2.3. Development and system setup

An implementation of the 32-bit MIPS processor; BROADCOM BCM5354; is available in the NETGEAR OSP router WGR614 series [7]. The available OOS supporting the BCM5354 MIPS processors are the flavours of BSD [6-10] and Linux that permit the modification of the source code for incorporating the defined software counters for the MIPS processor. The source code compilation process is generally categorized as architecture independent and dependent. The architecture independent is common across all platforms supported by the OOS, while the vendor of the processor provides the dependent section of the source code. The development system being x86 based, requires the usage of OOS cross-compiler for generating the modified kernel for the MIPS processor, and these tools are available on the internet that are specific to the chosen OOS. The cross-compiler; commonly referred to as the toolchain [11], [12] is provided by the vendor that is specific to the processor implementation. The environment for source code and tool chain build would necessitate a disk space of about three-hundred and fifty megabytes; hence, the build environment should be setup in a location with adequate disk space. The kernel version of the operating system will not have any direct bearing on the tool chain, and the resulting source code build. Nevertheless, it is preferable to have the development system OS patched to the highest possible level.

The development system configuration will be as per the specification mandated by the chosen OS, and it is essential to ensure network connectivity is available for transferring the firmware image from the development system to that of the chosen hardware of MIPS32 implementation. The protocols that will be utilized during data transfer can be with either TCP or TFTP.

3. REQUIREMENT ANALYSIS FOR PERFORMANCE MEASUREMENT

Hardware of MIPS32 implementation. The protocols that will be utilized during data transfer can be with either TCP or TFTP.

3.1. Requirement specification

The following functional requirements are:

- ASCII format to be used for data collection and accessible for data correlation and its applications.
- The/proc file system in the file CPU info will show the values of the counter
- Types of performance measurement is preferred in the hit-miss-refresh cycle related to the: *Dcache; Icache; Scache; TLB*
- All operations defined for the cache, and the TLB are to be covered
- The use of a separate variable counters to be used to count the overflows for each metric will be essential
- Data structure of the performance measurement to be placed in the architecture specific *.../include/asm-mips* directory
- Metric update routines of the performance measurement to be placed in the architecture *.../arch/mips/kernel* directory
- The source code modification should ensure minimal change in the firmware footprint size
- It is necessary to provide the ability to compile the kernel without the metric collection
- Efficiency of the introduced code is not the goal as the focus is to get as much as data from the cache and TLB management routines.

3.2. System analysis

In cache either write-back or/and invalidate are the major operations. When the CPU has updated cache write-back operation are performed. Hence, a required and the corresponding memory update will take place. To access a fresh set of data from the memory an invalidate operation is performed. The write-back and invalidate operations are applied for the Icache, Dcache and Scache lines. It is necessary to cache initialization is first done on the Icache, followed by the Dcache. The BCM5354 processor source code analyzing of the cache operations are as defined by [3] are:

#define Index_I_Invalidate_Ins 0x00
#define Index_I_Writeback_Inv_Data 0x01
#define Index_I_Writeback_Inv_SData 0x03
#define Hit_I_Invalidate_Ins 0x10
#define Hit_I_Invalidate_Data 0x11
#define Hit_I_Invalidate_SData 0x13
#define Hit_W_Writeback_Inv_Data 0x15
#define Hit_W_Writeback_Inv_SData 0x17
#define Hit_W_Writeback_Ins 0x18
#define Hit_W_Writeback_Data 0x19
#define Hit_W_Writeback_SData 0x1b

3.2. System design

The cache operations listed [3] are utilized by multiple routines on the caches to either flush the lines or the KSeg0; So, it is necessary to trace and find the function used to defined cache operations using unique metrics. The list of traced metrics is a perspective of generating the complete view of the cache operations of the MIPS32 processor. The source code analyzing; [3], [13]-[19]; for the analyzed and the defined function calls for the performance measurement of the caches; consists of I, D and S; along with the TLB operations of the MIPS32 design and implementation is categorized in the Table 1. The visualized of the cache operations from Table 1, the operations can be repeated on the lines or the ways or on KSeg0.

The base metric with the help of a roll over counter helps to analyses and track the metric data over a suitable extended period as per the design and analyzed requirement. The processor will cause a rollover of the metric for a high-rate activity. The metrics choice of the data type is base and the roll over which are of the type of unsigned int. The base and the rollover metrics data type will necessitate a change to an unsigned long type of data if it is necessary by the decision based on the processor activity. The data structure of the designed metrics will thus consist of the based metrics and the roll over counters.

Table 1. Mapping of function call, operation, and metrics

Metric to be Updated	Listed Function Name and Cache Operation
i_i_way	flush_i_icache_i_line_i_indexed, Index_I_Invalidate_I_I on ways
i_i_unroll_i_kseg0	blast_i_icache, Index_I_Invalidate_I and cache unroll of kseg0
i_i_unroll_i_way	blast_i_icache_i_page_i_indexed, Index_I_Invalidate_I_I and cache unroll on ways
i_i_line_i_flush	flush_i_icache_i_line, Hit_I_Invalidate_I of line
i_i_pline_i_flush	protected_i_flush_i_icache_i_line, Hit_I_Invalidate_I_I of line
i_i_unroll_i_page	blast_i_icache_i_page, Hit_I_Invalidate_I_I and cache unroll on page
i_i_d_way	flush_i_dcache_i_line_i_indexed, Index_I_Writeback_I_Inv_I_D on ways
i_i_d_unroll_way	blast_i_dcache_i_page_i_indexed, Index_I_Writeback_I_Inv_I_D and cache unroll of ways
i_d_unroll_kseg0	blast_i_dcache, Index_I_Writeback_I_Inv_I_D and cache unroll of kseg0
i_d_line_flush	flush_i_dcache_i_line, Hit_I_Writeback_I_Inv_I_D on line
i_d_unroll_page	blast_i_dcache_i_page, Hit_I_Writeback_I_Inv_I_D and cache unroll on page
i_d_invld_ln	Invalidate_I_dcache_I_line, Hit_I_Invalidate_I_D on line
i_d_writeback	protected_i_writeback_i_dcache_i_line, Hit_i_Writeback_i_D on line
i_s_way	flush_i_scache_i_line_i_indexed, Index_I_Writeback_I_Inv_I_SD on ways
i_s_unroll_i_pg_i_ways	blast_i_scache_i_page_i_indexed, Index_I_Writeback_I_Inv_I_SD and cache unroll on page and ways
i_s_unroll_i_kseg0	blast_i_scache, Index_I_Writeback_I_Inv_I_SD and cache unroll on kseg0
i_s_invld_i_ln	Invalidate_I_scache_I_line, Hit_I_Invalidate_I_SD
i_s_line_i_flush	flush_i_scache_i_line, Hit_I_Writeback_I_Inv_I_SD on line
i_s_unroll_i_page	blast_i_scache_i_page, Hit_I_Writeback_I_Inv_I_SD and cache unroll on page
i_i_fill	Fill_t_icache_t_line, Fill_t_Icache_t_line
t_tlb_t_lflush_t_all	Local_t_flush_t_tlb_t_all
t_tlb_t_lflush_t_mm	Local_t_flush_t_tlb_t_mm
t_tlb_t_lflush_t_rng	Local_t_flush_t_tlb_t_range
t_tlb_t_updt_t_mmu	Update_t_mmu_t_cache
t_tlb_t_lflush_t_pg	Local_t_flush_t_tlb_t_page

Based on the defined cache operations the metric update is organized [3], [20]-[30] hence, the metric computation based on the statement of switch operations. The routines to be updated the metrics are defined as:
void update_i_cache_i_metric (int type_i, int cm_i_ops);
void update_i_tlb_i_metric (int tlb_i_fn);

The operation for the cache *cm_i_ops* parameters, while *type* indicates the cache operation on either the line, or the way, or the Kseg0 in the *update_i_cache_i_metrics* functions. The operation on the TLB with the parameter *tlb_i_fn*. The parameters *type* and *tlb_i_fn* are arrived and defined with a unique value in the file .../*inc/asm/cache_perf_mips32.h*, and is indicated as follows:
#define unroll_i_c 0xe0 / cache_unroll */*
#define line_i_c 0xe1 / cache_line */*
#define kseg0_i_c 0xe2 / kseg0_address */*
#define ways_i_c 0xe3 / mip_cache_ways */*
#define page_i_c 0xe4 / cache_ page */*
#define pline_i_c 0xe5 / protected_cache_line */*
#define Fill_I_Icache_I_line 0xe6 / fill_icache_line */*
#define lf_t_tlb_t_all 0xf0 / local_tlb_ flush_all */*
#define lf_t_tlb_t_mm 0xf1 / local_flush_tlb_mm_struct */*
#define lf_t_tlb_t_rng 0xf2 / local_ flush_tlb_range */*
#define lf_t_tlb_t_pg 0xf3 / local_flush_tlb_page */*
#define up_t_tlb_t_mmu 0xf4 / update_ tlb_mmu */*

As shown in Table 1 the metric is updated with, he method of the function call
Update_i_cache_i_metric (line_i_c, Hit_I_Invalidate_I_I);
Update_t_tlb_t_metric (lf_t_tlb_t_mm);

For the further analysis, the matric data will be available in the OS provided in the file */proc/cpuinfo*, individually categorized for data and instruction caches as Dcache, Icache, Scache and TLB. Based on the requirement of the processor performance analysis the metric data can be read from the */proc/cpuinfo* file, with each read providing the current data of the metrics. The data format is indicated in the path */proc/cpuinfo*, with metric display sequence as indicated in Table 2:
dcache_i_ metrics:(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
icache_i_ metrics: (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
scache_i_metrics: (0,0,0,0,0,0,0,0,0,0,0,0)
tlb_t_metrics: (0,0,0,0,0,0,0,0,0,0)

Table 2. In/proc/cpuinfo metric display sequence

Cache type	Metric Sequence
dcache_i_metrics	i_d_way, i_d_way_roll, I_d_invld_I_ln, I_d_I_invln_I_roll_i_d_writeback, i_d_wb_i_roll, i_d_blast, i_d_blast_i_roll, i_d_unroll_i_kseg0, i_d_ukseg_i_roll, i_d_i_unroll_i_way, i_d_i_uw_i_roll, i_d_i_line_i_flush, i_d_i_lf_i_roll, i_d_i_unroll_i_page, i_d_i_up_i_roll
icache_i_metrics	i_i_way, i_i_way_i_roll, i_i_blast, i_i_blast_i_roll, i_i_unroll_i_way, i_i_uw_i_roll, i_i_unroll_i_kseg0, i_i_ukseg_i_roll, i_i_line_i_flush, i_i_lf_i_roll, i_i_pline_i_flush, i_i_plf_i_roll, i_i_unroll_i_page, i_i_up_i_roll, i_i_fill, i_i_fill_i_roll
scache_i_metrics	s_I_invld_I_ln, s_I_invln_I_roll, s_I_way, s_I_way_I_roll, s_I_unroll_I_kseg0, s_I_ukseg_I_roll, s_I_unroll_I_pg_I_ways, s_I_upw_I_roll, s_I_line_I_flush, s_i_lf_I_roll, s_I_unroll_I_page, s_I_up_I_roll
tlb_t_metrics	tlb_t_lflush_t_all, tlb_t_lfa_t_roll, tlb_t_lflush_t_mm, tlb_t_lfmm_t_roll, tlb_t_lflush_t_rng, tlb_t_lflrng_t_roll, tlb_t_lflush_t_pg, tlbt_lfpg_t_roll, tlbtlb_t_updt_tmmu, tlb_t_upmmu_t_roll

3.3. Building environment and hardware setup

The major section of the Linux source code is splitted into major two section which are architecture dependent and architecture independent. The router used here are NETGEAR router which comes with the bundle and source code with a third section and specific to router specific board. The building process of the provided code involves compiling the architecture specific code, following the Linux specific code and then architecture independent, the code compiling the results in the kernel images following the compile of the

router board specific and the resulting firmware images for the NETGEAR WGR614v9 router. For building the firmware images for the NETGEAR WGR614v9 router is as follows:

a. Downloading and installing the libstdc++.so.5 OS development library.
b. Downloading the cross-compiler tool chain TOOLSOURCE_2004_03_31.tgz from the path *ftp://ftp.gpl-devices.org/pub/vendors/Belkin* [9] and installing the tool chain in the user computer home directory.
c. Creating a symbolic link */opt/brcm* to the tool chain directory path *..../org/tools/brcm* that is available in the user home directory.
d. Downloading from the internet and installing the utility *trx* in the directories of the path */opt/brcm/hndtools-mipsel-linux-3.2.3/bin*, and */opt/brcm/hndtools-mipsel-uclibc-3.2.3/bin*. Ensure the utility *trx* has the execute permission for the owner, group, and the user.
e. Ensuring the path to */opt/brcm/hndtools-mipsel-linux-3.2.3/bin* and */opt/brcm/hndtools-mipsel-uclibc-3.2.3/bin* directories are available in the shell environment variable *PATH*.
f. Downloading the NETGEAR WGR614v9 source code; WGR614v9-V1.2.6_18.0.17WW_src.tar.bz2.zip; available at the internet website *ftp://downloads.netgear.com/files/GP*L [13] and installing the code in a suitable directory under the home directory of the user computer.
g. Cleaning the existing object files, and the kernel image vmlinux under the Linux and the router sections of the source code tree using the followings indicated steps:
   cd …/src/router
   make clean
   make router-clean
   cd ../linux/linux
   make clean
h. Building the Linux kernel image from the source code directory path *.../src/linux/linux* using the following steps for generating the MIPS32 kernel image vmlinux:
   make dep
   make
i. Building the router code in the directory path *.../src/router* using the following steps:
   make
   make install
j. The router WGR614v9 firmware upgrading of the image file will be created in the directory path *.../src/router/mipsel-uclibc* in the file name beginning with the *WGR614v9* and ending with the extension *chk*. The firmware file name example is *WGR614v9-12051706.chk*.

The router board is the hardware step for implementation of MIPS32 core by BROADCOM processor BCM5354 and is indicated in Figure 4.

Figure 4. NETGEAR WGR614 router board

3.4. Pseudo-code implementation procedure

The implementation of the performance metric data collection under the architecture specific memory managementroutines are available in the locations path *…/src/linux/linux/arch/mips/mm*, and the *…/src/linux/linux/include/asm-mips* directories. and the listing of the functions are listed in Table 1 and is available in the indicated directories. The pseudo-code is developed for collecting the performance metrics is represented in the following steps a to e:

a. Calling the function in the code for the metric update from the function calls associated with the MIPS32 cache and TLB management, along with the designed parameters of cache / TLB operation and the type of the required operation; either on the ways, or the line, or the KSeg0.
b. Updating the associated metric as designed for the functions listed in Table 1.
c. If the metric counter is overflowing; then it made to wraps to the value zero and then then increment of the counter is performed corresponding rollover metric counter.
d. Capturing and printing the values of all the designed metrics in the */proc/cpuinfo* file.
e. Repeating the steps a through d for each of the function call as listed in Table 1.

The method of implementing the pseudo-code is indicated below:

a. Locating the section in the source code which are handling cache and TLB function calls from Linux kernel and memory management routines that are available in the directory *.../src/linux/linux//kernel* and *.../src/linux/linux/mm* directories.
b. Locating the section under the architecture specific source code which are handling the kernel and the memory management, and identify the section handling for the the cache and TLB management located under the directories *.../src/linux/linux//arch/mips/mm* and *.../src/linux/linux/arch/mips/kernel*. The associated architecture dependend specific header file is located under the directory *.../src/linux/linux/include/asm-mips*. Specific files that will be used are:
c. Defining the header file listing for the data structure in a file under the *.../src/linux/linux/include/asm-mips* directory. As given as example: *cache_perf_mips32.h*
d. Defining the routines to updating and displaying the metric counters; as listed in Section 3.3; in a file under the *.../src/linux/linux/arch/mips/kernel* directory, example: *cache_perf_proc.c*. Ensure that it is initialise the metrics data structure to zero.
e. Modifying the *.../src/linux/linux/arch/mips/kernel/Makefile* to include the resulting object file generated in step d.
f. Modifying the architecture specific listed cache operations; listed in Table 1; in the source code file *.../src/linux/linux//include/asm-mips/mips32_cache.h* and *.../src/linux/linux/arch/mips/mm/tlb-r4k.c* to call the metric updating functions; defined and analysed in step d; along with the necessary parameters. Based on the design requirements the parameters are listed in Table 1 and Section 2.3.
g. As per the design requiremts calling the metric display function; created in step d; from the file *.../src/linux/linux//arch/mips/kernel/proc.c* to displaying the data in the */proc/cpuinfo* file.

Validating the changes on the hardware by building the firmware image for the NETGEAR WGR614v9 router based MIPS32 processor implementation along with the changes in the source code.

4. RESULTS AND DISCUSSIONS

The processor performance measurement results involve in understanding the metrics as per the design considerations and making its interpretations to concludes the observations of the results associating with these and its data-based interpretations. With the help of these interpretations wherever required a suitable code modification is performed and performance measurement had analyse.

4.1. Metric interpretation

For considering the example of the cache operation *Index_I_Invalidate_I_I*, and as seen in Table 1 under Section 2.3, these operations are performed on the Icache line and in Icache ways and on the KSeg0. The operation Index_Invalidate_I was performed to get the exact number of times about the metrics, the method is:

$$Number\ of\ Index\_I\_Invalidate\_I\_I = i\_i\_way + i\_i\_unroll\_i\_kseg0 + i\_i\_unroll\_i\_way \quad (1)$$

Similarly, for each design requirements based defined operation of the cache and TLB, the corresponding metrics are indicated in Table 1 are to be added as shown in (1). The combination of metrics generates a derived metric, whereas the value of the individual metrics is providing the base metric. The Column 3 of the Table 1 indicates the base metrics. The data provided by the metrics which are of base and derived are helping to utilized and draw a histogram tracing for the processor cache activity over a period.

The histograms are plotted based on the data generated by the metric analysis tools met. Figure 5, Figure 6, Figure 7, and Figure 8 which are plotted based on the chosen set of metrics for the Dcache, Icache and the TLB. In the plot of the histogram the y-axis indicating the number of events and the x-axis is the time in seconds. The collected data for the histogram had every five seconds of sample, over a period of one hundred and ten seconds. The command used for executing the data are simple command of the standard OS

that are there for displaying the contents of the directory and files under the */proc* file system, etc. To get a better example perspective of the effect of the commands, consider an example of the command: *cat /proc/cpuinfo*. The execution of the command has the following indicative steps:

a. The shell spawning a new process by creating a process table entry in the program and copies the file descriptors. The process association with the cat application.
b. The created process is placed by the scheduler for execution.
c. By accessing the cat command, the code is brought into the memory from the file system.
d. The file name is parsed by executing the code. In this paper it is the case, the file is /proc/cpuinfo.
e. Checking if the file is a directory the respective file table entry is accessed. if it is true and yes, exit.
f. Opening the file for reading.
g. Reading the line until it reaches to the end of line mark and store it in the buffer.
h. Accessing the character device driver for console terminal to interact with the application of the performance measurement.
i. Opening the device for writing.
j. Getting back to the file for reading operation, and now calling the print routine to output data to the character device file.
k. Repeating the steps g to j until it is end of file.
l. Releasing resources of the system occupied for reading the file /proc/cpuinfo.
m. Releasing resources of the system occupied by the cat application.
n. Releasing the system resources associated with process table entry.

There are multiple instructions from the steps a to n with each is either can be in a memory or on the flash file system of the router. The instructions are available into the memory which are prefetched due to the earlier executions. When the data are not available lead to the flush of the data to bring the new data required for the CPU of MIPS32. So, the cache invokes the TLB to perform the data transfer from the main memory to the required cache. As shown in the histograms in Figure 5, Figure 6, Figure 7 and Figure 8, There is a high activity at the first five-seconds of data collection of the Dcache and Icache when compared to the TLB and Scache.

As shown in the Figure 3, when there is execution of the instruction happening causing a fetch of the data from the memory into the Dcache. Before loading the data into Icache or Dcache, the entire Kseg0, or the way or the lines are flushed simultaneously updating the TLB. Each activity on the caches and TLB are a measurable event and hence are updated. The flushing activities either can be writeback or invalidate of the respective caches. The four histograms as shown in the Figure 5, Figure 6, Figure 7 and Figure 8 should be seen visualized simultaneously to have a proper cache and TLB activity on the processor.

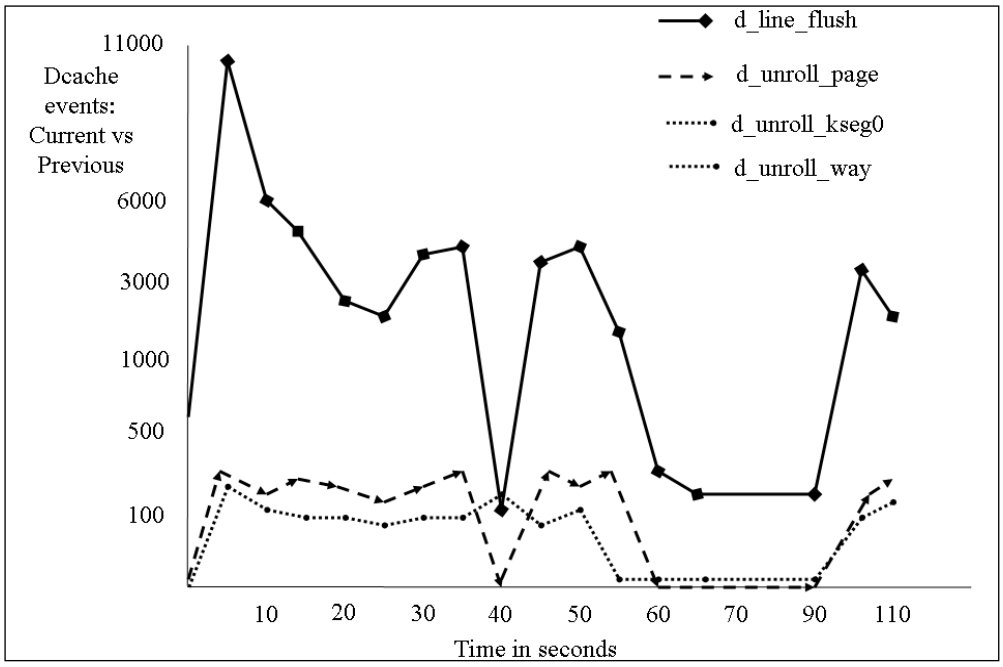

Figure 5. Histogram of Dcache events

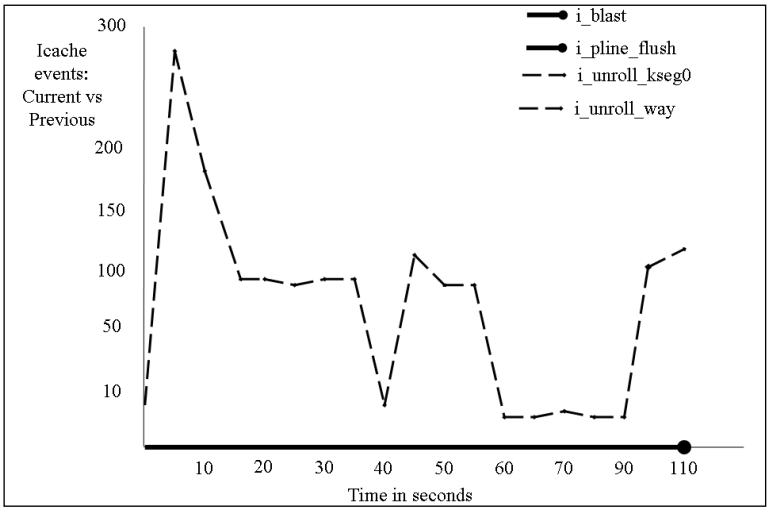

Figure 6. Histogram of Icache events

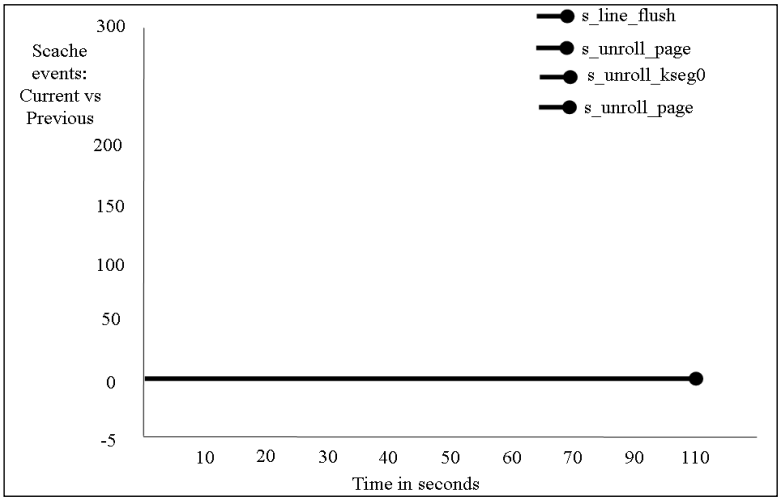

Figure 7. Histogram of Scache events

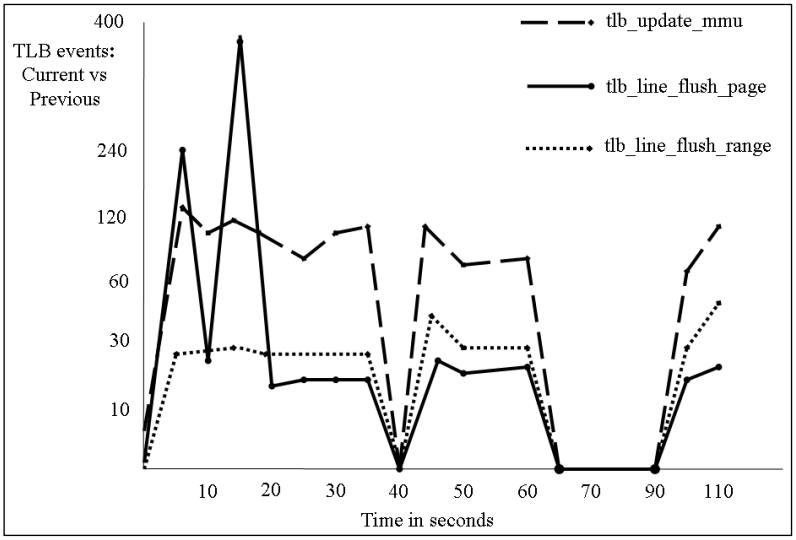

Figure 8. Histogram of TLB events

4.2. Analysis of the code modifications

Table 3 shows listing of the code lines that were added into the stock source code [31]-[34]. The files *mips32_cache.h*, *Makefile*, *proc.c*, *tlb-r4k.c*, *Config.h*, *applets.h* and the *usage.h* were part of the source code tree, while the files *cache_perf_proc.c*, *cache_perf_mips32.h* and *zmet.c* were added into the source code tree as part of the implementation of the cache and TLB performance measurement metrics for MIPS32 architecture. The total number of lines that were added into the source code tree has been 1009 (one-thousand and nine), counted without the comments or introduced blank lines for formatting of the code.

5. CONCLUSIONS

Most of the processor utilize the hardware counters, where are MIPS32 architecture does not have hardware counter. So, the performance system requires the usage of the software counters. Software counters are defined and available in the kernel and need to write the code to access them. While using the software counter there are rise of overflow of the counters. How to use the overflow of the counters are incorporated in the L1 Cache and TLB of the MIPS32 memory management. The experiments are conducted by designing the twenty-seven metrics based on the software engineering principle for measuring the performance of the software counters for the MIPS32 processor for cache operations. The studies help to understand the importance of the software counters into the OS based processor performance.

REFERENCES

[1] Eswer V. and Naik Dessai S. S., "Embedded software engineering approach to implement BCM5354 processor performance," *International Journal of Software Engineering and Technology*, vol. 1, no. 1, pp. 41-58, 2016.

[2] Naik Dessai S. S. and Eswer V., "Embedded software testing to determine BCM5354 processor performance," *International Journal of Software Engineering and Technology*, vol 1, no. 4, pp.121-151, 2016.

[3] Abran, A., Moore, J. W., Bourque, P., Dupuis, R., and Tripp, L., "Software engineering body of knowledge," *IEEE Computer Society, Angela Burgess*, pp 2-1 to 2-10, 2004. [Online]. Available: http://www.swebok.org.

[4] Broadcom Corporation, *BCM5354 Product Brief–Optimized 802.11G Router with Broadrange*, Broadcom Corporation, 5354-PB01-R, 11 Sep. 2007.

[5] BELKIN Inc., "How to connect a USB storage device and share files through your Belkin router," https://www.belkin.com/ph/support-article?articleNum=8066, (accessed May, 2021).

[6] Brinkley Sprunt, "The basics of performance monitoring hardware," *IEEE Micro*, vol. 22, no. 4, pp. 64-71, 2002.

[7] Don Anderson, *Universal serial bus system architecture*, 2nd ed., Addison-Wesley Developer's Press, 2001.

[8] Lance M. Berc, Sanjay Ghemawat, Moniika H. Henzinger, Richard L. Sites, Carl A. Waldspurger, and William E. Weihl, "High frequency sampling of processor performance counters," *USA Patent 5796939*, 1998. [Online]. Available: http://freepatentsonline.com/5796939.html.

[9] D. S. Miller and R. Baechle, "arch/mips/mm/tlb-r4k.c," *Linux Kernel 2.4.20 source code*, https://solembtech.com/browse/linux/v4.19.156/source/arch/mips/mm/sc-ip22.c, (accessed May, 2021).

[10] Dominic Sweetman, *See MIPS Run*, Second Edition, Morgan Kaufmann Publishers, 2007.

[11] Guy G. F. Lemieux, "Hardware performance monitoring in multiprocessors," Masters degree thesis - University of Toronto, 1996.

[12] Gregory S. Freeland, Joel L. Gross, and Jose A. Laboy, "Tuneable processor performance benchmarking," *USA Patent 20070136726 A1*, 2007 [Online]. Available: http://www.freepatentsonline.com/20070136726.html.

[13] Jack Dongarra, Kevin London, Shirley Moore, Phil Mucci, and Dan Terpstra, "Using PAPI for hardware performance monitoring on Linux systems," *Conference on Linux Clusters: The HPC Revolution*, vol. 5, 2001.

[14] Lance M. Berc, Sanjay Ghemawat, Moniika H. Henzinger, Richard L. Sites, Carl A. Waldspurger, and William E. Weihl, "High frequency sampling of processor performance counters," *USA Patent 5796939*, 1998. [Online]. Available: http://freepatentsonline.com/5796939.html

[15] Marco Zagha, Brond Larson, Steve Turner, and Marty Itzkowitz, "Performance analysis using the MIPS R10000 performance counters," in *Supercomputing'96: Proceedings of the 1996 ACM/IEEE Conference on Supercomputing*, 1996, pp. 16-16.

[16] Michael Huy Phan, "Performance measurement for embedded systems," *USA Patent 6643609 B2,* Nov 4, 2003. [Online]. Available: http://www.freepatentsonline.com/6643609.html.
[17] MIPS Technologies, "MIPS32 ® 4Kc ™ Processor Core Datasheet - Revision 01.03," *MIPS Technologies Inc., MD00247,* June 2000.
[18] MIPS Technologies, "…/include/asm-mips/mips32_cache.h," *Linux Kernel 2.4.20 source code,* http://cgit.openembedded.org/openembedded/plain/recipes/linux/linux-wrt-2.4.20/2.4.20_broadcom_3_37_2_1109_US.patch, (accessed May, 2021).
[19] M. Warner Losh, "An overview of FreeBSD/mips," *AsiaBSDCon,* 2009. [Online]. Available: http://2009.asiabsdcon.org/papers/abc2009-P4B-paper.pdf.
[20] NETGEAR Inc., "Wireless-G router WGR614v9 reference manual," *NETGEAR Inc., 202-10308-01,* May 2008.
[21] Paul J. Drongowski, "Basic performance measurements for AMD Athlon 64, AMD Opteron and AMD Phenom processors," *Advanced Micro Devices, Inc,* 2008.
[22] R. Baechle, "Cache operations for the cache instruction," gitlab.inria.fr, *2002.* https://gitlab.inria.fr/blare/kblare/-/blob/84c3d4aaec3338201b449034beac41635866bddf/include/asm-mips/cacheops.h, (accessed May, 2021).
[23] Roger S. Pressman., *Software engineering: a practitioner's approach*, 5th ed., McGraw-Hill, 2005
[24] S. Browne, J Dongarra, N. Garner, G. Ho, P. Mucci, "A portable programming interface for performance evaluation on modern processors," *The international journal of high-performance computing applications,* vol. 14, no. 3, pp. 189-204, 2000.
[25] Shirley Moore, Patricia Teller, and Michael Maxwelll, "Efficiency and accuracy issues for sampling vs. counting modes of performance monitoring hardware," *In Proceedings of the DoD High Performance Computing Modernization Program's User Group Conference*, 2002.
[26] Tsuyoshi Nagao and Hitoshi Suzuki, "Processor system and performance measurement method for processor system," *USA Patent 20070277178A1,* Nov 29, 2007, [Online]. Available: http://www.freepatentsonline.com/20070277178.html.
[27] Warner Losh, "A brief history of FreeBSD/MIPS," *BSDCan 2008 Canada,* 2008. [Online]. Available: http://www.freebsd.org/~imp/bsdcan2008.pdf.
[28] FreeBSD/MIPS Project, [Online]. Available: http://www.freebsd.org/platforms/mips.html, Aug 2011
[29] Toolchains, https://www.mips.com/develop/tools/compilers/linux-toolchain/, (accessed May, 2021).
[30] Vendor toolchains and third-party software, belkin.com. https://www.belkin.com/support/assets/wemo/license/FW_License_Wemo_Maker_v2.00.11423.pdf, (accessed May, 2021).
[31] M. Bolado, H. Posadas1, J. Castillo, P. Huerta1, P. Sánchez, C. Sánchez, H. Fouren, and F. Blasco, "Platform based on open-source cores for industrial applications," *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition,* vol. 2, 2004, pp. 1014-1019.
[32] Wiplove Mathur, and Jeanine Cook, "Improved estimation for software multiplexing of performance counters," *13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems. IEEE*, 2005.
[33] WGR614v9 router source code, https://www.downloads.netgear.com/files/GDC/2649_GPL.html, (accessed May, 2021).
[34] Xiao Zhang, Sandhya Dwarkadas, Girts Folkmanis, and Kai Shen, "Processor hardware counter statistics as a first-class system resource," *Department of Computer Science, University of Rochester,* 2007, [Online]. Available: http://www.cs.rochester.edu/u/sandhya/papers/hotos07.pdf