Benchmark Analysis 508.namd_r SPEC CPU2017

line 1: 1st Valentinos Pariza
line 2: *Faculty of Computer Science*
line 3: *University of Cyprus*
line 4: Nicosia, Cyprus
line 5: vapriz01@cs.ucy.ac.cy
line 5: 909759

line 1: 2nd Marios Pafitis
line 2: *Faculty of Computer Science*
line 3: *University of Cyprus*
line 4: Nicosia, Cyprus
line 5: mpafit02@cs.ucy.ac.cy
line 6: 911719

Περίληψη:

In this benchmark analysis, firstly we have checked as most as we could the capabilities and characteristics of the processors of the computers in the department of computer science in University of Cyprus Labs. After these we have taken the benchmark 508.namd_r from SPEC CPU2017 ,we have studied the use, its inputs and outputs of this benchmark program, we have tested the performance of the specific type of computers in the labs on this benchmark program (we checked the performance on a program that is strongly associated with parallelism) with command perf stat, we estimated the time that the program need to run and we have analyzed all these data by creating tables and graphs that show how the performance and the time of a program like this can be affected by the hardware and more important ,what are the reasons and factors that affect it and why these factors appear.

Keywords—component, formatting, style, styling, insert (*key words****)***

Analysis of program uses parallelism, SPEC CPU2017, 508.namd_r, Intel Core i5-4590,

I. ΕΙΣΑΓΩΓΗ

Many people today think that the code is the power of the programming. But at some point they ignore that all the power of the programming is derived from the hardware. All the capabilities and power that each programming language has is an association of some programming techniques and the operations that a hardware offers. This doesn't mean that the programming isn't important. We all know that with programming techniques (OOP…) we can illustrate great and efficient algorithms, build efficient data structures that can be used in many aspects of our lives and many more appliances. But all these have a conceivable upper bound of optimization. The hardware can limit you even if you have the fastest algorithm in the world. If you don't know what is happening below the abstraction of the programming language then you haven't accomplished nothing.

So in this benchmark analysis we come to show how the hardware behaves in executing a benchmark program. This benchmark program is focused on the parallelism and on the big number of manipulations and calculations made on floating point numbers of big precision (because this benchmark is associated with protein atoms which these elements of the world exist in a system which many of its sizes are smaller than 1 and the precision is important in this type of systems) and integer numbers. Furthermore in this benchmark program analysis we distinguish the Hardware factors that affect the execution time and performance of this program. The analysis is made on a small set of a data, but it is at some point clear to see all the factors and unexpected situations that occur in the execution of the benchmark program.

II. ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ ΤΟΥ ΕΠΕΞΕΡΓΑΣΤΗ ΚΑΙ ΤΗΣ ΜΝΗΜΗΣ ΤΟΥ ΥΠΟΛΟΓΙΣΤΗ

CPU Processor Characteristics

Model name:	Intel(R) Core (TM) i5-4590 CPU @ 3.30GHz
Announced on:	1/03/2014
Price:	$285
Lithography Process:	22nm
Architecture:	x86_64
CPU op-mode(s):	32-bit, 64-bit
Byte Order:	Little Endian
CPU(s):	4
On-line CPU(s) list:	0 - 3
Thread(s) per core:	1
Core(s) per socket:	4
Socket(s):	1
Vendor ID:	GenuineIntel
CPU family:	6
Model:	60
Stepping:	3
CPU MHz:	3294.36
CPU max MHz:	3700
CPU min MHz:	800
Virtualization:	VT-x
L1d cache:	32K
L1i cache:	32K
L2 cache:	256K
L3 cache:	6144K
Memory Channels:	2
Memory Bandwidth:	25.6GB/s
Scalarity:	4-way Superscalar
TDP (Watts):	84W
Pipeline Stages	12
Out-Of-Order execution	Yes

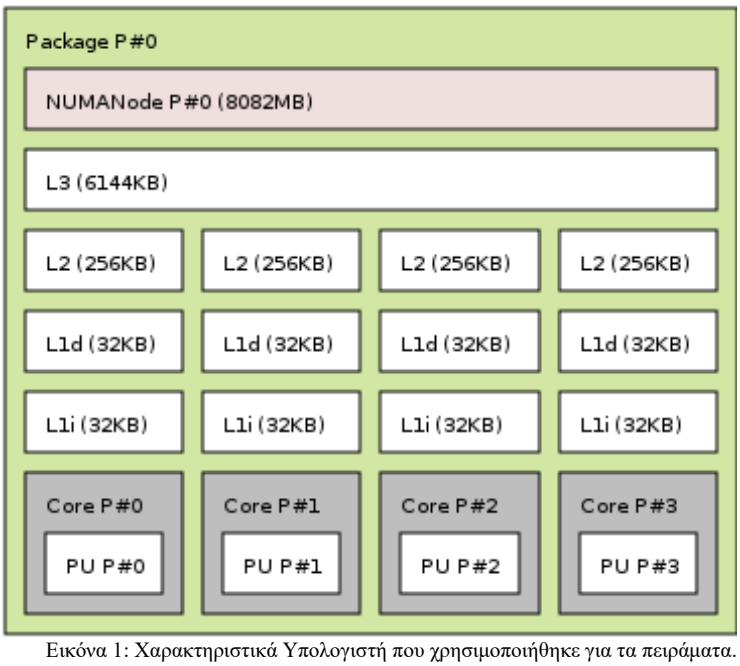
Main Memory Characteristics

MemTotal:	7897984 kB
RAM Frequency	1600MHz
Technology	DDR3
DIMM slots	4
Enable up to	32GB

We have tested the SPEC in an HP EliteDesk 800 G1 TWR. The machine had a code core i5 4th generation CPU from Intel. The frequency of the CPU is 3.3GHz and 3.7GHz in turbo boost. The processor has 3 level cache. The L1 instruction and data cache and the L2 cache is unique for each core. The L3 is common for all four cores. The system is supported by an 8GB DDR3 RAM. Because the model of the CPU is i5 we have available 4 threads, one for each core. The Operating System is Linux Centos with x86_64 architecture. The machine can support either 32bit operating system or 64bit.

CPU Cores Characteristics

Processor	0	1	2	3
Stepping	3			
Microcode	0x24			
CPU MHz	2129.77	3300.00	1682.42	3041.78
Cache size	6144 KB			
Physical id	0			
Siblings	4			
Core id	0	1	2	3
CPU cores	4			
Apicid	0	2	4	6
Initial Apicid	0	2	4	6
FPU	Yes			
FPU_exception	Yes			
cpuid level	13			
Ways-of-Associativity	8			
Clflush size	64			
Cache Alignment	64			
Address Sizes	39 bits physical, 48 bits virtual			
Extensions and Technologies	MMX, SSE, SSE2, SSE3, SSSE3, SSE4 / SSE4.1 + SSE4.2, AES, AVX, AVX2			
TLB/Cache details:	64-byte Prefetching			
	Data TLB: 1-GB pages, 4-way set associative, 4 entries			
	Data TLB: 4-KB Pages, 4-way set associative, 64 entries			
	Instruction TLB: 4-KByte pages, 8-way set associative, 128 entries			
	L2 TLB: 1-MB, 4-way set associative, 64-byte line size			
	Shared 2nd-Level TLB: 4-KByte / 2-MB pages, 8-way associative, 1024 entries			



Εικόνα 1: Χαρακτηριστικά Υπολογιστή που χρησιμοποιήθηκε για τα πειράματα.

III. ΛΕΠΤΟΜΕΡΗΣ ΠΕΡΙΓΡΑΦΗ ΤΟΥ ΕΠΕΞΕΡΓΑΣΤΗ

Model name:	**Intel(R) Core(TM) i5-4590 CPU @ 3.30GHz**	**Intel(R) Core™: Brand i5: Brand Modifier 4: Generation Indicator 590: SKU Numeric Digits**
Architecture:	**x86_64**	**x86 Architecture 64 bits**
CPU op-mode(s):	**32-bit, 64-bit**	**Compatible op-modes 32-bit, 64-bit**
Byte Order:	**Little Endian**	**Endianness**
CPU(s):	**4**	**4 CPU Cores**
Thread(s) per core:	**1**	**(1S only) Scalar**
Core(s) per socket:	**4**	**4 Cores, we have only one socket**
Socket(s):	**1**	**Number of Sockets**
CPU MHz:	**3294.36**	**Current CPU MHz**
CPU max MHz:	**3700**	**Maximum MHz**
CPU min MHz:	**800**	**Minimum MHz**
L1d cache:	**32K**	**Data L1 Cache, Dedicated for each Core**
L1i cache:	**32K**	**Instruction L1 Cache, Dedicated for each Core**
L2 cache:	**256K**	**Dedicated for each Core**
L3 cache:	**6144K**	**Shared cache for all Cores**
Scalarity:	**1S Only**	**Scalar**
Out-Of-Order Execution:	**Yes**	**Since 1998 all CPU Processors of Intel are Out-Of-Order Execution ***
Extensions and Technologies:	**MMX**	**Single Instruction Multiple Data Set**
	SSE, SSE2, SSE3, SSE4 / SSE4.1 / SSE4.2	**Streaming SIMD Extensions**
	AVX	**Advanced Vector Extensions**
	AVX2	**Advanced Vector Extensions 2.0**
	AES	**Advanced Encryption Standard instructions**
Ways-of-Associativity:	**8**	**8 cache lines are included in each cache set in the cache memory**
FPU:	**Yes**	**Floating Point Unit**
FPU_exception:	**Yes**	**Handling Exceptions in Floating Point Unit**
TLB/Cache details:	**64-byte Prefetching**	**translation lookaside buffer between the CPU and the CPU Cache**

IV. SPEC CPU 2017: 508.NAMD_R

Benchmark Name:

508.namd_r

Benchmark Author:

Jim Phillips, Theoretical and Computational Biophysics Group, University of Illinois

Benchmark Description:

This Benchmark ,which is known as 508.namd_r is derived-taken from the program NAMD (Basically from the data layout and inner loop of NAMD which are some structural elements that are included in software NAMD) and is a parallel program for simulating large biomolecular systems(more specific, simulating millions of atoms).NAMD is the abbreviation for Nanoscale Molecular Dynamics ,and it is a computer software for molecular dynamics simulation, build with Charm++(parallel programming model).NAMD scales to over 200,000 cores for very large systems in order to simulate efficiently its biomolecular systems(using all the parallel capabilities that offers),but serial performance is also important to the over 50,000 users who have tried the program over the past decade. Because of the parallel efficiency of this program, its maturing (The continuous updates of NAMD) and of the importance of the operations that this program does (calculating inter-atomic interactions in a small set of functions, most of its runtime), this program was used as a compact benchmark for CPU2017.

Input Description:

The file is named apoa1.input (APOA1 is the gene which encodes apolipoprotein A-I, which is the most important protein component of high density lipoprotein (HDL) in plasma).This file format is created by NAMD 2.9 using a special command ("dumpbench") and it doesn't need file readers. Or any other setup code for the benchmark. At the beginning some parameters that are associated with atoms-proteins appear (PMETolerance, PMEEwaldCoefficient…) which indicate some attributes of the system where the atoms will be placed and be simulated in. After these parameters ,some values-numbers follow that indicate coordination and attributes(like velocity…) for the atoms-Proteins .If we look more carefully at the format of the file ,the file is

broken into segments that indicate some sets of data. For example there is a data set that starts from TABLE_BEGIN and ends at word TABLE_END that is a set with some floating point numbers. More general the file consists of a lot of numbers, floating point numbers and integers that need to be analyzed. This is a great example of how a computer can be checked in efficiency in parallel processing of floating point and integer numbers.

Output Description:

The CPU2017 uses the "—output namd.out" command –line option in order to produce a brief output file, which has name namd.out and contains various checksums of the calculations made on the forces of the proteins' atoms, in each iteration which the number of iterations is indicated by "-–iterations ".These should be consistent across platforms and work normally, within round off error, and it is used for validation.

V. ΑΝΑΛΥΣΗ ΣΤΑΤΙΣΤΙΚΩΝ: ΜΕΣΟΣ ΟΡΟΣ/ ΤΥΠΙΚΗ ΑΠΟΚΛΙΣΗ ΚΤΛ.

Run:	*Time:*	*Instructions:*	*Cycles:*	*Cache-References:*	*Cache Misses*
1	278.6167769	2,598,572,799,099	1,011,272,820,498	426,794,502	83,634,338
2	282.0349881	2,598,020,796,844	1,019,347,821,638	426,601,910	97,953,905
3	282.1891051	2,599,140,256,996	1,017,613,217,687	509,954,876	106,823,566
4	277.6292747	2,598,679,185,496	1,012,297,762,928	425,635,355	82,064,939
5	278.6058041	2,598,781,539,355	1,014,213,617,452	651,499,800	92,872,258
6	278.0991725	2,598,302,727,257	1,012,327,250,147	437,649,228	87,355,244
7	278.3836300	2,597,897,308,778	1,014,740,218,106	516,742,022	97,035,237
8	282.2383814	2,599,667,522,597	1,018,158,641,332	434,544,226	105,896,342
9	278.7860509	2,598,391,723,431	1,012,403,566,426	441,164,241	96,131,381
10	280.4582602	2,598,673,965,855	1,014,161,625,559	430,659,510	93,354,873
Average	279.7041444	2,598,612,782,571	1,014,653,654,177	449,971,763	94,312,208
Standard Deviation	1.8395441	520,287,031	2,808,275,224	36,352,832	8,367,359
Diversion Percentage	1	0	0	8	9
Description	The Deviation for the Execution time is <1%.	The amount of instructions per run is stable with Deviation <0%	The total number of Cycles per run is stable with Deviation <0%	The Deviation for the Cache-References is 8%.	The Deviation for the Cache-Misses is 9%

Branch-Instructions:	*Branch-Misses:*	*L1-dcache-loads*	*L1-dcache-load-misses*	*L1-dcache-stores:*
41,976,372,697	1,867,180,597	707,153,207,226	31,900,909,167	242,134,015,330
42,083,712,071	1,865,640,444	707,260,540,760	31,902,922,930	242,197,864,033
42,048,339,562	1,864,594,077	707,232,451,284	31,921,629,954	242,102,493,816
42,002,227,727	1,869,047,725	707,257,345,718	31,910,264,018	242,166,135,231
41,956,801,070	1,868,024,747	707,375,532,351	31,841,840,742	242,109,014,357
42,002,740,768	1,867,717,220	707,435,600,432	31,916,446,895	242,136,540,370
42,002,767,900	1,868,118,973	707,340,662,244	31,952,822,542	242,186,263,143
41,970,016,340	1,867,820,082	707,435,599,945	32,008,305,828	242,073,886,974
42,022,107,797	1,867,749,612	707,359,631,724	31,843,562,325	242,161,830,724
42,066,837,703	1,867,672,458	707,284,361,891	31,872,187,889	242,174,407,279
42,013,192,364	1,867,356,594	707,313,493,358	31,907,089,229	242,144,245,126
42,036,526	1,295,047	91,651,835	49,679,082	40,051,837
0	0	0	0	0
The Branch-Instructions are stable with Deviation <0%	The Branch-Misses are stable with Deviation <0%	The L1-dcache-loads are stable with Deviation <0%	The L1-dcache-load-misses are stable with Deviation <0%	The L1-dcache-stores are stable with Deviation <0%

L1-icache-load-misses:	*LLC-loads:*	*LLC-load-misses:*	*LLC-stores:*	*LLC-store-misses:*	*dTLB-loads:*
18,147,342	357,895,359	70,747,107	38,393,385	15,515,000	706,337,329,273
18,735,454	354,471,427	81,844,031	36,440,531	14,519,331	706,327,424,116
18,277,934	444,812,735	90,272,455	38,190,637	18,263,763	706,682,217,942
18,792,476	359,918,715	69,764,726	33,680,606	13,531,606	706,235,530,360
18,028,760	582,117,308	78,388,618	45,448,296	18,860,398	706,094,119,868
18,015,442	370,309,808	73,951,456	35,141,148	12,398,861	706,098,472,527
19,008,144	445,774,620	79,967,264	45,192,861	20,868,651	705,934,437,773
18,305,914	368,507,405	88,019,910	38,585,346	19,835,346	706,368,217,147
18,379,864	367,671,127	78,134,925	43,681,156	14,878,104	706,361,825,864
18,306,232	361,581,690	80,242,855	37,272,275	14,812,520	706,329,009,708
18,399,756	381,215,876	76,630,123	39,202,624	16,277,009	706,276,858,458
336,468	36,698,914	4,205,048	4,154,680	2,348,003	203,576,469
2	10	5	11	14	0
The L1-icache-load-misses are stable with Deviation <2%	The Deviation for the LLC-Loads is 10%.	The Deviation for the LLC-load-misses is 5%	The Deviation for the LLC-Stores is 11%.	The Deviation for the LLC-store-misses is 14%	The L1-dcache-stores are stable with Deviation <0%

dTLB-load-misses:	*dTLB-stores:*	*dTLB-store-misses:*	*iTLB-loads:*	*iTLB-load-misses:*
6,094,892	241,624,827,924	2,237,867	27,585	372,083
6,208,282	241,531,608,018	1,986,190	42,821	439,329
7,389,569	241,655,167,307	2,302,869	35,357	334,430
6,065,202	241,560,997,578	1,953,321	38,766	350,802
6,445,339	241,613,604,809	2,244,629	49,486	911,590
5,873,537	241,614,833,057	1,926,576	31,412	476,631
7,865,371	241,564,762,627	2,216,828	49,637	367,826
8,239,327	241,685,056,041	2,163,695	42,669	401,985
6,047,156	241,625,004,884	2,078,087	35,145	332,864
6,219,899	241,564,197,009	2,070,753	31,677	564,123
6,467,694	241,604,005,925	2,118,082	38,456	408,499
717,839	47,710,291	133,914	7,586	76,830
11	0	6	20	19
The Deviation for the dTLB-load-misses is 11%.	The dTLB-stores are stable with Deviation <0%	The Deviation for the dTLB-store-misses is 6%.	The Deviation for the iTLB-loads is 20%.	The Deviation for the iTLB-load-misses is 19%.

Observations based on Mean and Standard Deviation:

1. All the events that were calculated and diverge from the standard deviation, approximately zero percent of their diverge, we suppose that their value is stable during all the executions. All the diversions of these events are because of the time multiplexing.
2. Because the cycles are behaving according to the instructions, and the instructions are the same always. The standard deviation of the instruction and cycles is near to zero.
3. Because of the other processes that might exist simultaneously on a pc, there is a diversion from a stable value.
4. Loads, stores, load-misses in data cache L1 are not affected by the other cores because all the L1 caches are dedicated to the cores. So the counting's of these events associated with L1 cache are quite similar in different executions.
5. But at the same time, the LLC (Last Level Cache) is shared and each core uses the same LLC. This has an effect that in different executions of the program ,the LLC can have different number of loads from different cores ,misses and stores because each time the number of references to LLC which is shared, can vary and the references to LLC may happen in different sequence by different cores.
6. Also the result that we have different number of references in different executions to LLC, this leads that the number of misses to LLC changes each time the references to LLC changes. This at the end affects the number of references to Main Memory which are corresponding to the number of misses in LLC.
7. The DTLB loads and stores do not change because the number of references to different segments in memory doesn't change in each execution of the program.
8. The data TLB hasn't has stable dTLB load-misses and dTLB store-misses because in each execution the TLB may find different available addresses to correspond to main memory addresses with the virtual memory addresses.
9. Cache misses behave with the same way as a sum with the LLC-load misses and LLC-store misses.

Execution Time:

From the execution of the command time we get the output is 276.974s

The time of the program which we calculated by the number of cycles multiplied by the time in seconds that needed for a cycle to complete is 307.471s for frequency value 3.3GHz and 274.231 for 3.7GHz.

We can see that the time that we have calculated is greater than the time that the processor needed to complete the execution of the program for the standard which is 3.3GHz. This is because the processor is working with the maximum frequency (turbo boost) 3.7GHz.

Time Multiplexing:

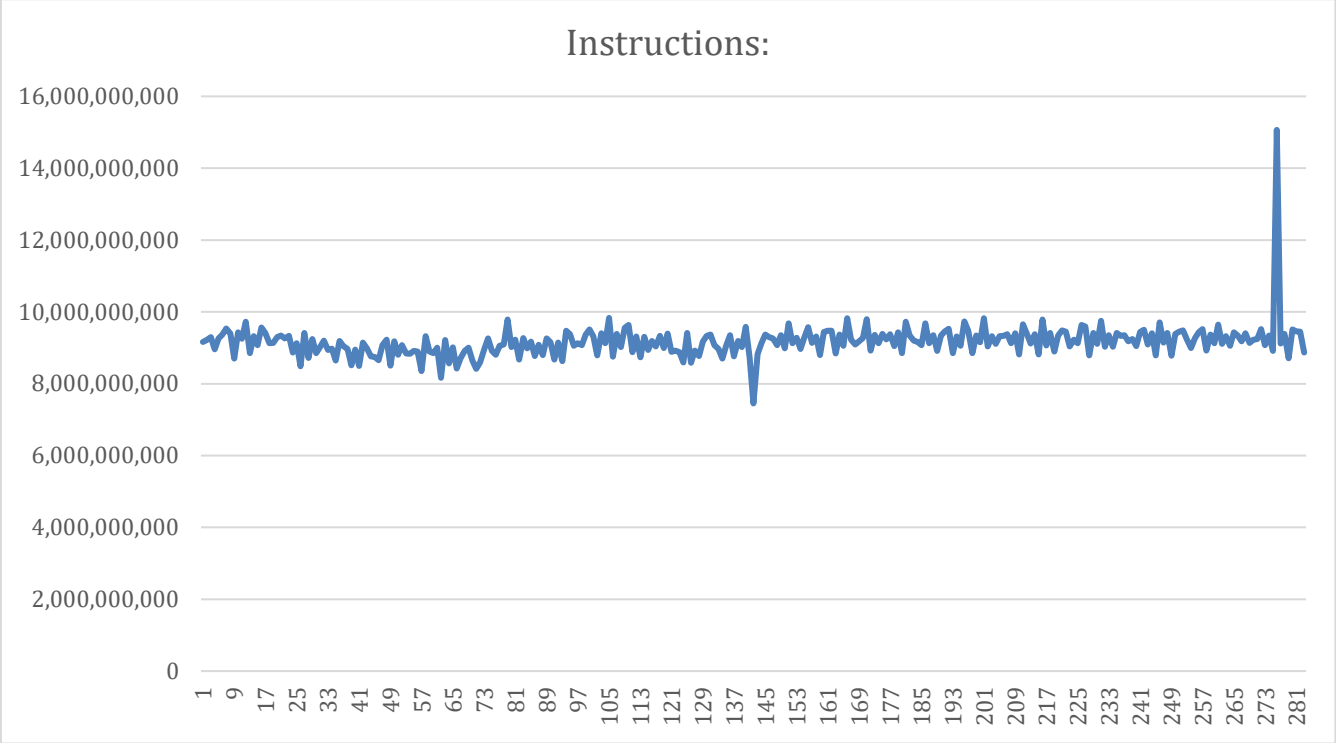
The kernel uses hardware counters in order to count an event. If there are more events than counters, then the system is using the Time Multiplexing method in order to estimate the final count. The time Multiplexing applies to PMU (Performance Monitoring Unit) events. Example of PMU events are the number of cycles, instructions retired, L1 cache misses, cache-references and so on. It uses a formula to scale the count based on total time enabled vs time running.

final_count = raw_count * time_enabled/time_running

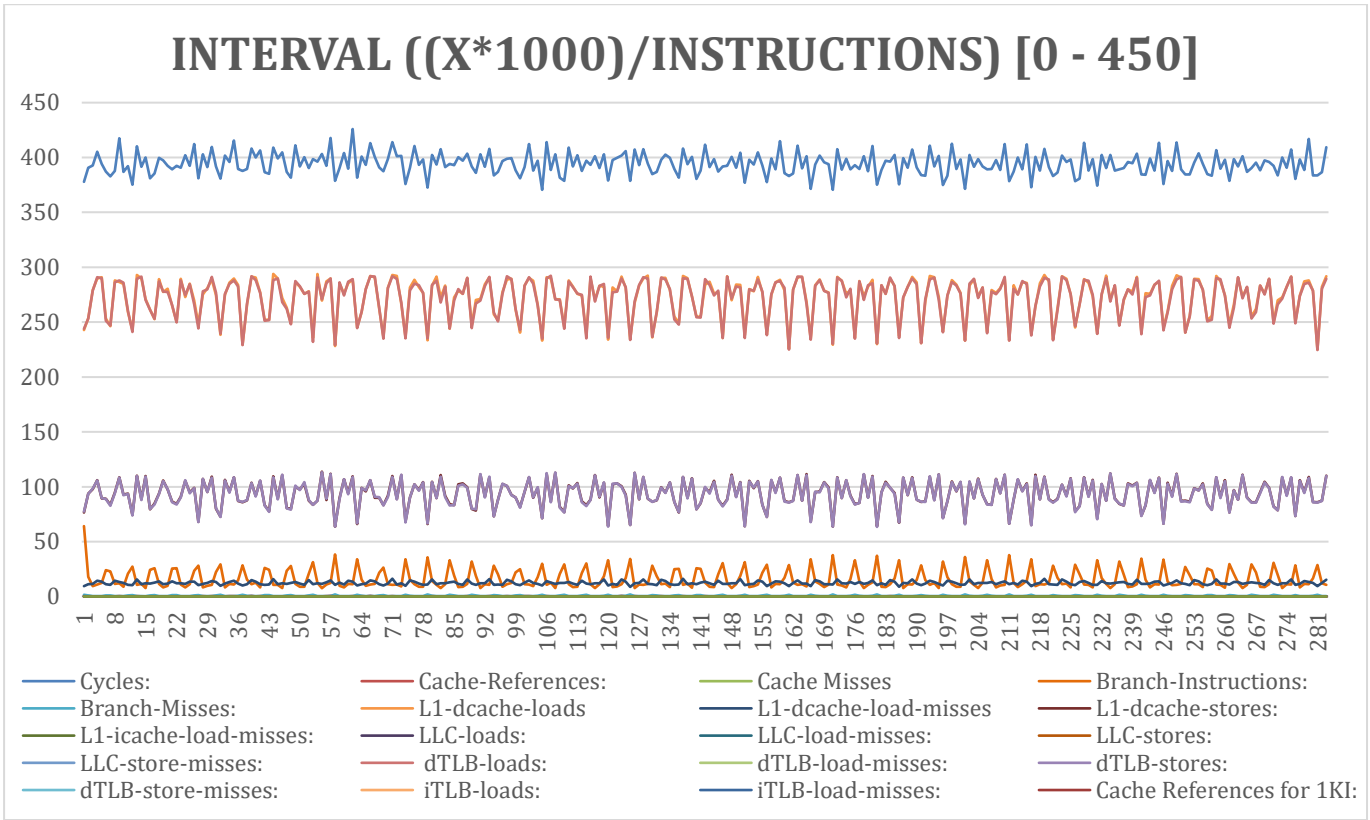
The final count is an estimation of the time enabled and not the actual time running. Simpler because the hardware has limited number of counters and because the counters must be used for all the events, this method takes the first N events (where N is the number of the available counters in the hardware) and let them to use the counters for a period of time. After the time has expired for the first N events to use the counters, the next N events are taken for using the counters, and the same process continues with the method of round-robin list (List which all the elements are taken with a rational sequence and when they reach the end, the picking starts again from the beginning).This method can be applied a little more different if some events must be used together for counting. After the end of the execution of the program, the time that was used for each event (sum) is scaled in order to correspond to the time that the program was running (With the equation above). The problem that occurs with this is that the events that we ask for, are not complete counting results but are some patterned-scaled results that correspond only to segments of the execution of the program and not at the whole program. This is a big problem because we lose some calculations for some events and for some others not, which these calculations might be important (like a cache-miss might occur, a branch-miss and so on) and this can lead that our data might be a little bit inconsistent.

\

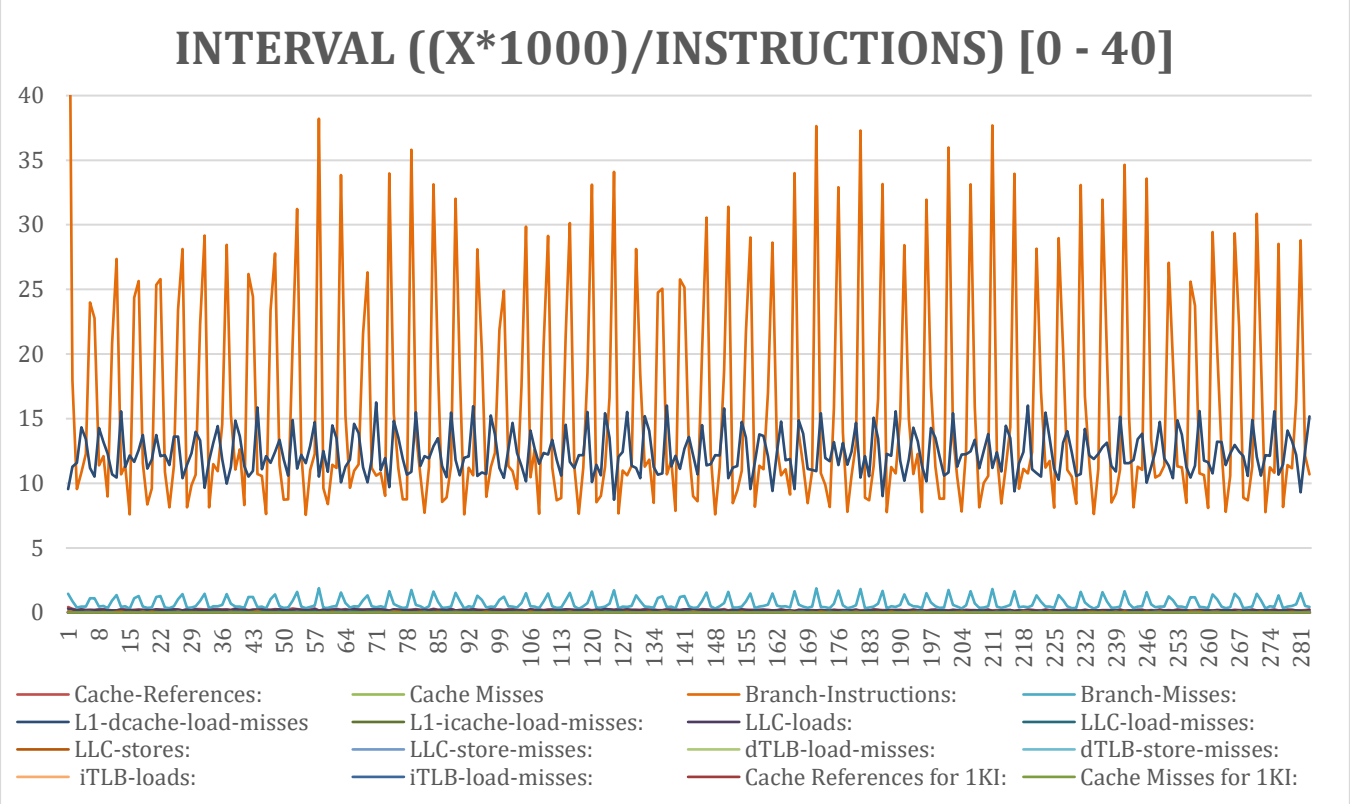
VI. CORPELATION ANALYSIS/ΓΡΑΦΙΚΕΣ ΠΑΡΑΣΤΑΣΕΙΣ 1SEC



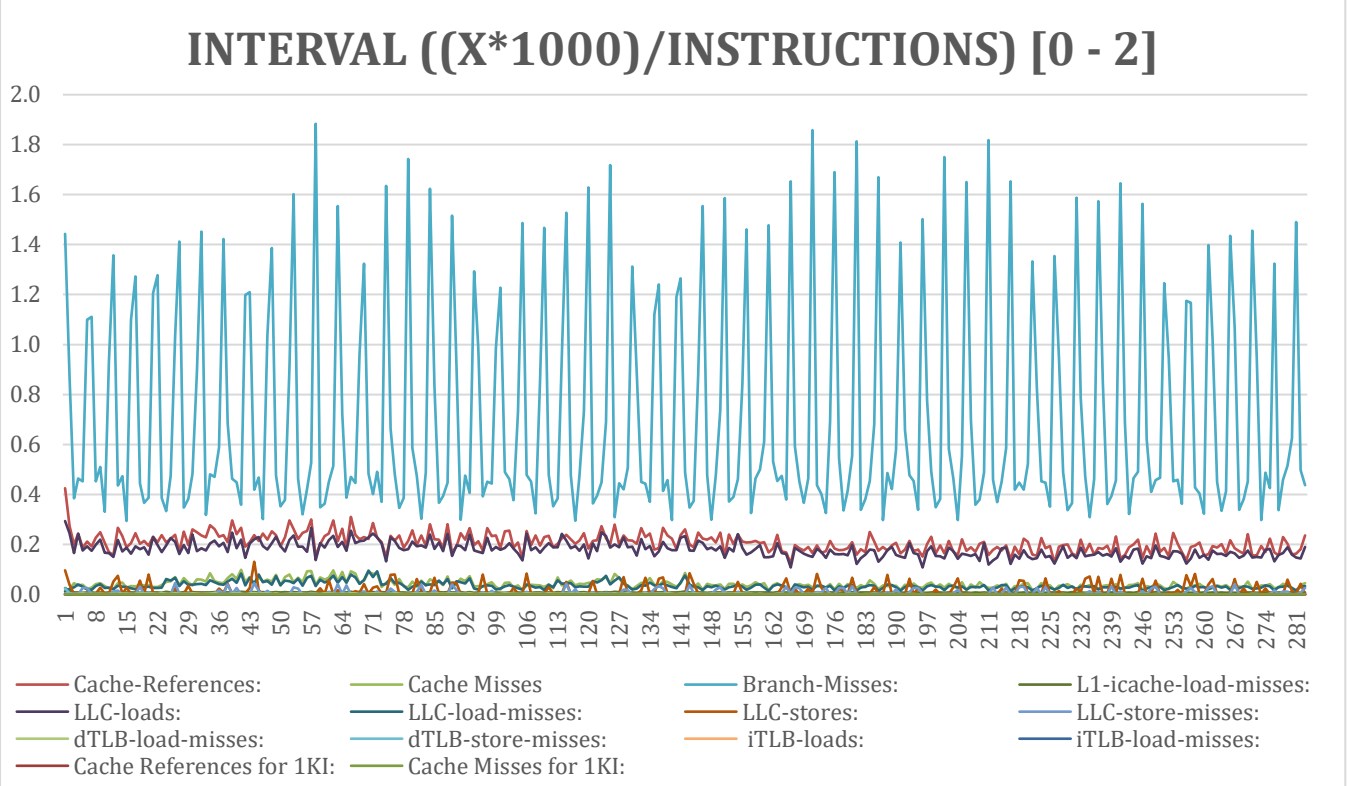
The amount of Instructions is not stable. It is changing between 8.5G to 9.6G Instructions. That means that each time the number of Instructions executed is different because different situations may occur (like branches, misses...) So, every 1 second the amount of instructions executed will not be the same.



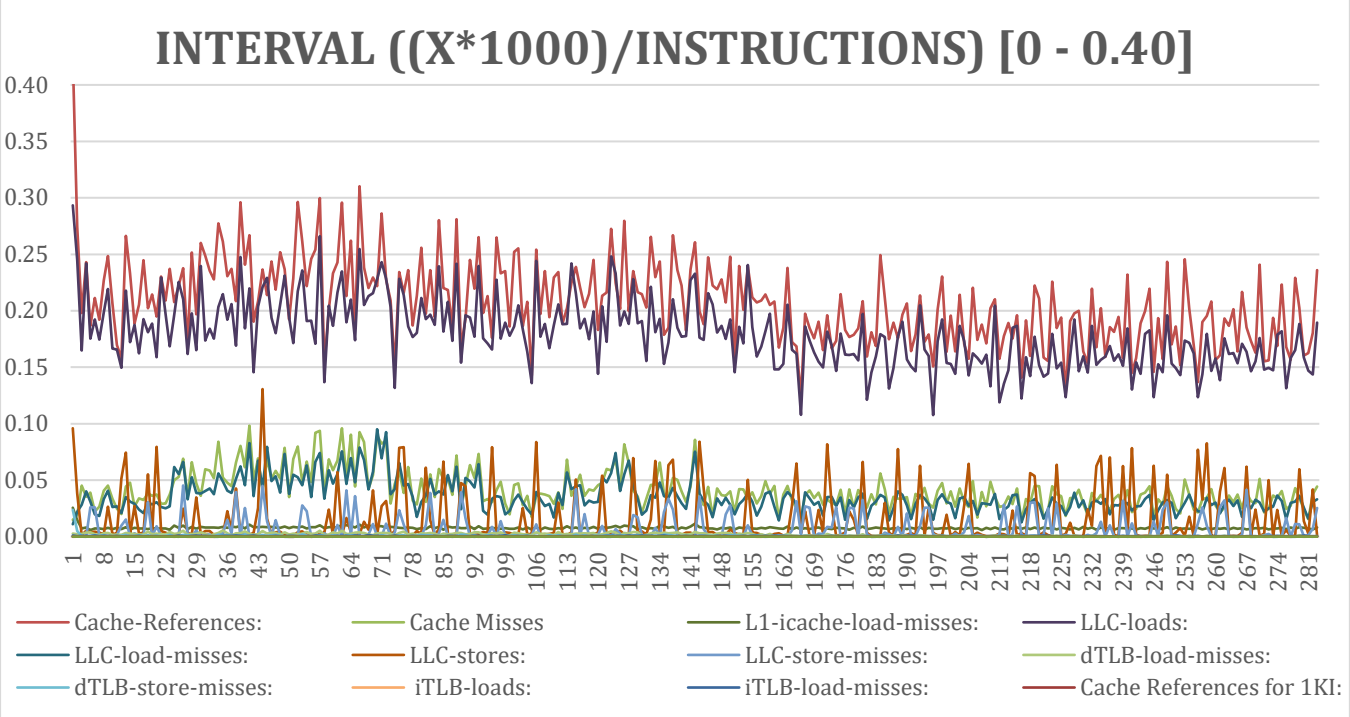
The cycles are stable because it depends on the hardware and not on our software. Also we can see that dTLB-loads and L1-cache-loads take the same values. This shows that when we want a data from the cache L1, we search for its physical memory address in the TLB(we load from TLB cache) .This shows that the number of references for load from cache L1 corresponds to the number of dTLB-loads .Also the IPC is approximately stable.



The L1-dcache-load-misses are more frequent than other metrics. But the most frequent is the appearance of branch instructions. We can see that the branches follow a stable continuous pattern. We can see this from the stable changes of the branch-instructions line, that appears intensively approximately at every 3 seconds and lasts for approximately 3.5 seconds. This can show that the program executes repeatedly a segment of code that contains similar branch instructions many times.



The Cache-References and the LLC-loads are more frequent at the beginning of the execution than later. This is happening because we do not have the data in the caches and we have to load them from the main memory.



The L1-cache-load-misses and the dTLB-store-misses are more frequent at the beginning of the execution than later. This is happening because we do not have the data in the caches and when the dTLB checks for an address that did not load yet.

s

	IPC:
IPC:	1
Time:	0.144287917
Instructions:	0.996026969
Cycles:	0.907459319
Cache-References:	-0.018535752
Cache Misses	-0.285972842
Branch-Instructions:	0.444123917
Branch-Misses:	0.431768901
L1-dcache-loads	0.517525418
L1-dcache-load-misses	0.177134149
L1-dcache-stores:	0.008689779
L1-icache-load-misses:	0.09131317
LLC-loads:	-0.077068634
LLC-load-misses:	-0.359337931
LLC-stores:	-0.022800987
LLC-store-misses:	0.038729512
dTLB-loads:	0.517974386
dTLB-load-misses:	-0.292240687
dTLB-stores:	0.009787868
dTLB-store-misses:	0.023032284
iTLB-loads:	0.05190647
iTLB-load-misses:	0.088748539
Cache References for 1KI:	-0.342414011
Cache Misses for 1KI:	-0.397216064

- Instructions follow the same pattern as IPC because the instructions are associated straightly with IPC.
- Cycles are inversely associated with IPC because IPC is the number of instructions divided by cycles
- Many references to memory (cache-references) might affect the IPC because the references might be misses (stores or loads)
- Branch instructions are approximately 50% of the instructions in the program because we can see as the instructions increase the number of branch instructions is increasing respectively. Generally, we can see that in a segment of instructions the number of branches appears at 40% -50% of the instructions.
- L1-dcache-loads are inverse associated with IPC because cache-loads at L1 may be happened after a miss and so increasing the number of loads, respectively the number of misses might increase.
- L1-dcache-stores are also inverse associated with IPC because cache-stores at L1increase the chance of having to store something after a store-miss.
- LLC-load-misses is inverse associated with IPC because when the number of the misses that might occur at last level cache are decreasing then the IPC increases. This indicates that the rate of looking for data at Main Memory when IPC is high, is low.
- LLC-loads increases the possibility for miss in that cache, because we have more load-references to that last level cache and this will lead to pay a hardware penalty if miss occur. But it doesn't happen to often, but it is still inverse associated.
- Time is inverse associated with IPC because as the IPC increases the time is decreasing because with higher IPC, we achieve the execution of more instructions in a specific amount of cycles.
- dTLB-loads is inverse associated with IPC because the IPC decreases when the dTLB-loads are low. Which means that if the references to TLB are decreasing, then the situation of having a miss in TLB cache is decreased respectively, decreasing also the possibility of having a miss and so, to pay a hardware penalty.
- **dTLB-load-misses is** also inverse associated with IPC because if the number of load- misses in TLB decreases then the IPC increases due to the fact that the penalties of load-misses are reduced.

VII. CONCLUSION

As a conclusion we observe that the program depends not only from the software but also from the hardware. We can observe the differences of some values in different executions and that this depends on the hardware, its initial and general state, and also we can see how the performance of the software changes in some hardware dependencies .Also the analusis that hase been made above has clarified the fact that the programs take advantage of some capabilities of the hardware, like parallelism in order to achieve their purpose. This is significantly important if

We would like to thank our Professors Dr. Petros Panagi and Dr. Giannos Sazzeidis for their support in analyzing this benchmark and for providing us with the knowledge which we needed in order to start researching.

REFERENCES

[1] G. Eason, B. Noble, and I. N. Sneddon, “On certain integrals of Lipschitz-Hankel type involving products of Bessel functions,” Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955. (references)

[2] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[3] I. S. Jacobs and C. P. Bean, “Fine particles, thin films and exchange anisotropy,” in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[4] K. Elissa, “Title of paper if known,” unpublished.

[5] R. Nicole, “Title of paper with only first word capitalized,” J. Name Stand. Abbrev., in press.

[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, “Electron spectroscopy studies on magneto-optical media and plastic substrate interface,” IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[7] M. Young, The Technical Writer’s Handbook. Mill Valley, CA: University Science, 1989.

[8] https://www.ks.uiuc.edu/Research/namd/

[9] https://en.wikipedia.org/wiki/NAMD

[10] https://ark.intel.com/products/80815/Intel-Core-i5-4590-Processor-6M-Cache-up-to-3-70-GHz-

[11] https://www.intel.com/content/www/us/en/processors/processor-numbers.htm

[12] https://www.spec.org/cpu2017/

[13] https://perf.wiki.kernel.org/index.php/Tutorial#multiplexing_and_scaling_events