Advanced Hotspots and Micro-architectural Analysis Using Vtune

SALIL BATRA, Arizona State University

1 INTRODUCTION TO VTUNE

Intel Vtune Amplifier is a performance profiler tool to collect statistics on different performance parameters of the code. The statistics collected using Vtune helps in detailed analysis of the code's performance as the tool pin points functions which consume the maximum time. The analysis includes algorithm, microarchitecture, and performance analysis of the code.

2 HARDWARE AND SOFTWARE SPECIFICATION FOR VTUNE ANALYSIS

CPU: 4th generation Intel(R) Core(TM) Processor family with 2.6 GHz frequency.
Memory (RAM): 16.0 GB
System Type: 64-bit Operating System, x64-based processor
Operating System: Microsoft Windows 10
Application Used: A C program which takes an array of 10,000 elements from a file, sorts the data using insertion sort, re-runs the data through selection sort; calculates mean, variance, and standard deviation of the data.
Number of Application functions: 14

3 ADVANCED HOTSPOTS ANALYSIS

Summary:

Elapsed Time: 0.720s	
CPU Time:	1.898s
Instructions Retired:	3,417,888,933
Estimated Call Count:	1,152,888
Total Iteration Count:	522,492,897
Loop Entry Count:	1,177,044
Average Loop Trip Count:	444
CPI Rate:	0.675
Wait Rate:	0.125
CPU Frequency Ratio:	0.469
Context Switch Time:	0.716s
Estimated Call Count:	1,152,888
Total Thread Count:	2
Paused Time:	0s

The CPU time taken to execute the code is 1.898s. The maximum time spent is on the following functions:

Function / Call Stack	CPU Time	Instructions Retired	Estim... Call Count	CPI Rate	Wait Rate	CPU Freq... Ratio	Con.. Swi... Time	Estimated Call Count	Context ... Pree..	Context ... Syn...	Module	Function (Full)	Source File	Start Address
sizeOfArray	340.504ms	1,133,473,508	20,000	0.688	0.016	1.435	0.395ms	20,000	3	23	SER520_SalilBatra.exe	sizeOfArray	SER520_SalilBatra.c	0x4013e0
findMin	278.362ms	931,612,219	0	0.702	0.019	1.472	0.420ms	0	5	18	SER520_SalilBatra.exe	findMin	SER520_SalilBatra.c	0x40186e
insertionSort	266.262ms	593,138,027	0	0.938	0.016	1.310	0.451ms	0	4	23	SER520_SalilBatra.exe	insertionSort	SER520_SalilBatra.c	0x4017d2
sizeOf2DArray	166.030ms	573,275,151	490,903	0.683	0.016	1.477	0.180ms	490,903	2	10	SER520_SalilBatra.exe	sizeOf2DArray	SER520_SalilBatra.c	0x40141b

sizeofArray():

So. Li.	Source	CPU Time: Total — Effective Time by Utilization (Idle, Poor, Ok, Ideal, Over)	CPU Time: Total — Spin Time	CPU Time: Total — Ove... Time	CPU Time: Self — Effective Time by Utilization (Idle, Poor, Ok, Ideal, Over)	CPU Time: Self — Spin Time	CPU Time: Self — Ove... Time	Instr... Retir... Total	Instructio... Retired: Self	Esti... Call Cou... Total	Esti... Call Cou... Self

37	`int sizeOfArray(int array[]){`										
38	`int i;`										
39	`for(i = 0; i < ROW; i++){`	3.9%	0.0%	0.0%	42.557ms	0ms	0ms	4.9%	162,810,615	3.5%	20,000
40	`if(array[i] == NULL)`	27.4%	0.0%	0.0%	297.947ms	0ms	0ms	29.3%	968,262,262	0.0%	0

findMin():

161	`for(i=start;i<numberOfElements;i++){`	2.6%	0.0%	0.0%	28.268ms	0ms	0ms	2.4%	79,9...	0.0%	0
162	`if(numbers[i]<numbers[index]){`	23.0%	0.0%	0.0%	250.094ms	0ms	0ms	25.8%	851, ...	0.0%	0

insertionSort():

149	`while((j >= 0) && (numbers[j] > key)){`	20.8%	0.0%	0.0%	226.967ms	0ms	0ms	15.0%	494, ...	0.0%	0
150	`numbers[j+1] = numbers[j];`	3.3%	0.0%	0.0%	36.343ms	0ms	0ms	2.8%	92,4 ...	0.0%	0

sizeOf2DArray():

48	`for(i = 0;i < ROW; i++){`	2.2%	0.0%	0.0%	23.456ms	0ms	0ms	3.2%	107,100,913	86.0%	490,903
49	`if(array[i][0] == NULL)`	12.8%	0.0%	0.0%	139.521ms	0ms	0ms	13.4%	442,206,498	0.0%	0
50	`return i;`	0.2%	0.0%	0.0%	2.052ms	0ms	0ms	0.7%	22,113,564	0.0%	0

The poor usage above suggests that the simultaneous running CPUs were utilized less than or equal to 50% of the target CPU usage.

Caller/Callee Data: The Top-Down tree pane depicts the caller/callee data and this can be used to analyze the Total and Self time data for the caller/callee functions. The following screen shot shows the amount of time spent during the caller/callee:

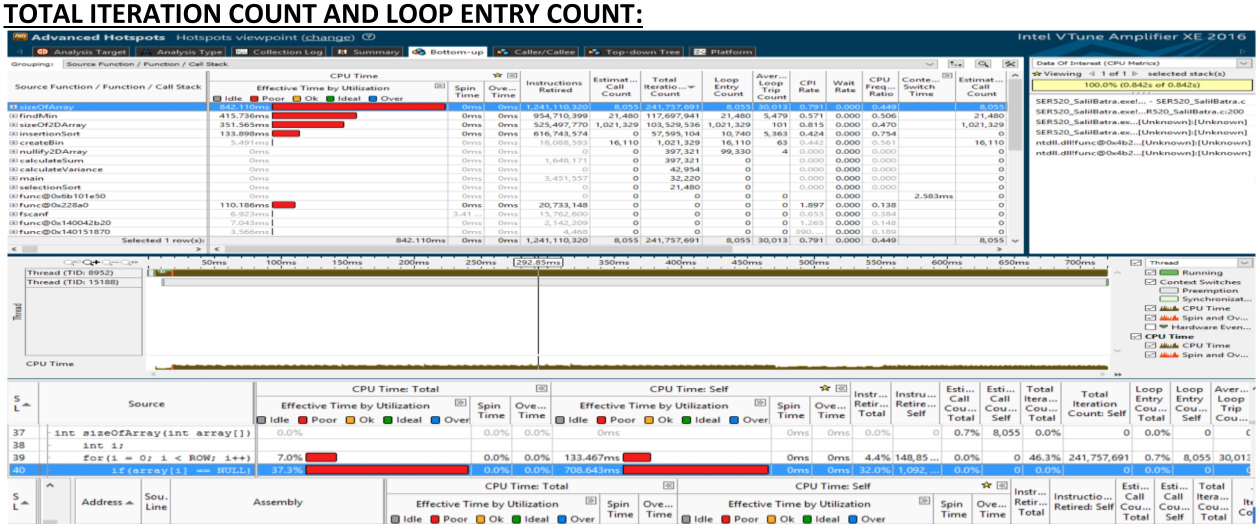
Analysis Target | Analysis Type | Collection Log | Summary | Bottom-up | Caller/Callee | Top-down Tree | Platform | SER520_Sal... | SER520_Sal... | SER520_Sa

Callers	CPU Time: Total – Effective Time by Utilization (Idle, Poor, Ok, Ideal, Over)	Spin Time	Overhead Time	CPU Time: Self – Effective Time by Utilization (Idle, Poor, Ok, Ideal, Over)	Spin Time	Overhead Time	Instr... Retir... Total	Instructio... Retired: Self	Estim... Call Coun... Total	Estim... Call Count: Self
sizeOf2DArray	100.0%	0.0%	0.0%	166.030ms	0ms	0ms	100.0%	573,275,151	100.0%	490,903
createBin	100.0%	0.0%	0.0%	166.030ms	0ms	0ms	100.0%	573,275,151	100.0%	490,903
main	100.0%	0.0%	0.0%	166.030ms	0ms	0ms	100.0%	573,275,151	100.0%	490,903
func@0x401180	100.0%	0.0%	0.0%	166.030ms	0ms	0ms	100.0%	573,275,151	100.0%	490,903
func@0x4012a0	100.0%	0.0%	0.0%	166.030ms	0ms	0ms	100.0%	573,275,151	100.0%	490,903
func@0x77f1b272	100.0%	0.0%	0.0%	166.030ms	0ms	0ms	100.0%	573,275,151	100.0%	490,903
func@0x77f1b251	100.0%	0.0%	0.0%	166.030ms	0ms	0ms	100.0%	573,275,151	100.0%	490,903
Selected 1 row(s):	100.0%	0.0%	0.0%	166.030ms	0ms	0ms	100.0%	573,275,151	100.0%	490,903

Callees	CPU Time: Total – Effective Time by Utilization (Idle, Poor, Ok, Ideal, Over)	Spin Time	Overhead Time	CPU Time: Self – Effective Time by Utilization (Idle, Poor, Ok, Ideal, Over)	Spin Time	Overhead Time	Instr... Retir... Total	Instructio... Retired: Self	Estim... Call Coun... Total	Estim... Call Count: Self
sizeOf2DArray	100.0%	0.0%	0.0%	166.030ms	0ms	0ms	100.0%	573,275,151	100.0%	490,903

As expected, the sizeOf2DArray function consumes the maximum time. This is so because the function works on the data received by the sizeOfArray function and then builds further data according to the program requirement.

TOTAL ITERATION COUNT AND LOOP ENTRY COUNT:



The sizeOfArray takes the maximum CPU time it has to first calculate the size of array and then it is called multiple times across different functions for the for-loop termination condition.

Wait Rate – 0.036: “The *Wait Rate* metric measures an average Wait time (in milliseconds) per synchronization context switch”. The wait time for insertion sort has been observed as maximum.

Function / Call Stack	CPU Time	Instructions Retired	Estim... Call Count	CPI Rate	W. Ra.	CPU Freq... Ratio	Con.. Swi... Time	Estimated Call Count	Context ... Pree..	Context ... Syn...	Module	Function (Full)	Source File	Start Address
insertionSort	238.135ms	608,863,743	0	0.928	0.017	1.487	0.323ms	0	0	19	SER520_SalilBatra.exe	insertionSort	SER520_SalilBatra.c	0x4017d2
sizeOf2DArray	165.475ms	559,326,596	501,243	0.699	0.015	1.481	0.303ms	501,243	2	19	SER520_SalilBatra.exe	sizeOf2DArray	SER520_SalilBatra.c	0x40141b
sizeOfArray	354.733ms	1,128,760,800	69,587	0.681	0.015	1.358	0.315ms	69,587	3	20	SER520_SalilBatra.exe	sizeOfArray	SER520_SalilBatra.c	0x4013e0
findMin	264.134ms	933,554,400	0	0.708	0.010	1.569	0.060ms	0	0	6	SER520_SalilBatra.exe	findMin	SER520_SalilBatra.c	0x40186e
createBin	5.007ms	22,138,468	0	0.525	0.007	1.456	0.007ms	0	0	1	SER520_SalilBatra.exe	createBin	SER520_SalilBatra.c	0x4014d0

The above data is also same for the CPU frequency ratio which is 0.469(i.e., <1) which indicates that the CPU is not running in the turbo boost mode.

TOP HOTSPOTS:

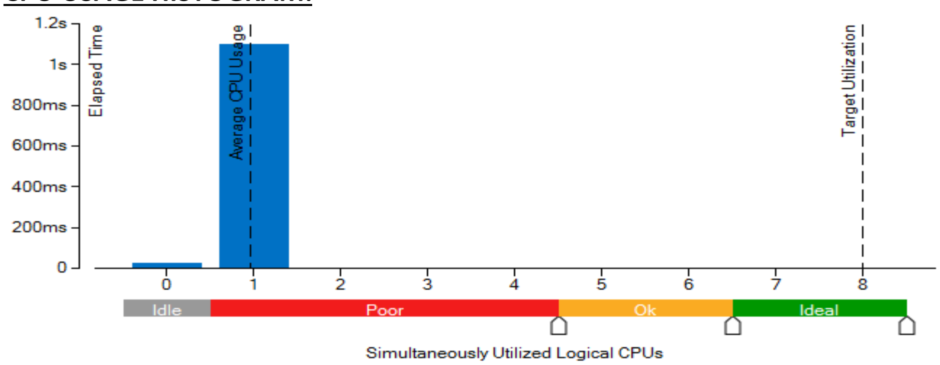
Function	Module	Estimated Call Count	CPU Time
sizeOfArray	SER520_SalilBatra.exe	69,587	0.355s
findMin	SER520_SalilBatra.exe	0	0.264s
insertionSort	SER520_SalilBatra.exe	0	0.238s
sizeOf2DArray	SER520_SalilBatra.exe	501,243	0.165s
func@0x48b204	ntkrnlpa.exe	0	0.011s
[Others]	N/A*	74,228	0.046s

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

As seen in the data above, the top hotspots are the functions which took maximum execution time, CPU performance went poor for their execution. These functions will have to be re-adjusted in the program order and their code has to be changed to make it more affective so as to improve the overall performance of the application. The following screenshot of the Top-Down tree shows that time consumed by the hotspots is maximum among other functions in the application:

sizeOfArray	32.8%	0.0%	0.0%	0.354s	0s	0s	33.9%	1,12...	10.8%	69,587	0.682	0.682	0.015	0.015	1
selectionSort	24.5%	0.0%	0.0%	0s	0s	0s	28.1%	0	0.0%	0	0.708	0.000	0.010	0.000	1
findMin	24.5%	0.0%	0.0%	0.264s	0s	0s	28.1%	933, ...	0.0%	0	0.708	0.708	0.010	0.010	1
insertionSort	22.1%	0.0%	0.0%	0.238s	0s	0s	18.3%	608, ...	0.0%	0	0.928	0.928	0.018	0.018	1
createBin	15.8%	0.0%	0.0%	0.005s	0s	0s	17.5%	22,1 ...	77.7%	0	0.696	0.525	0.015	0.007	1
sizeOf2DArray	15.3%	0.0%	0.0%	0.165s	0s	0s	16.8%	559, ...	77.7%	501, ...	0.702	0.699	0.016	0.016	1

CPU USAGE HISTOGRAM:



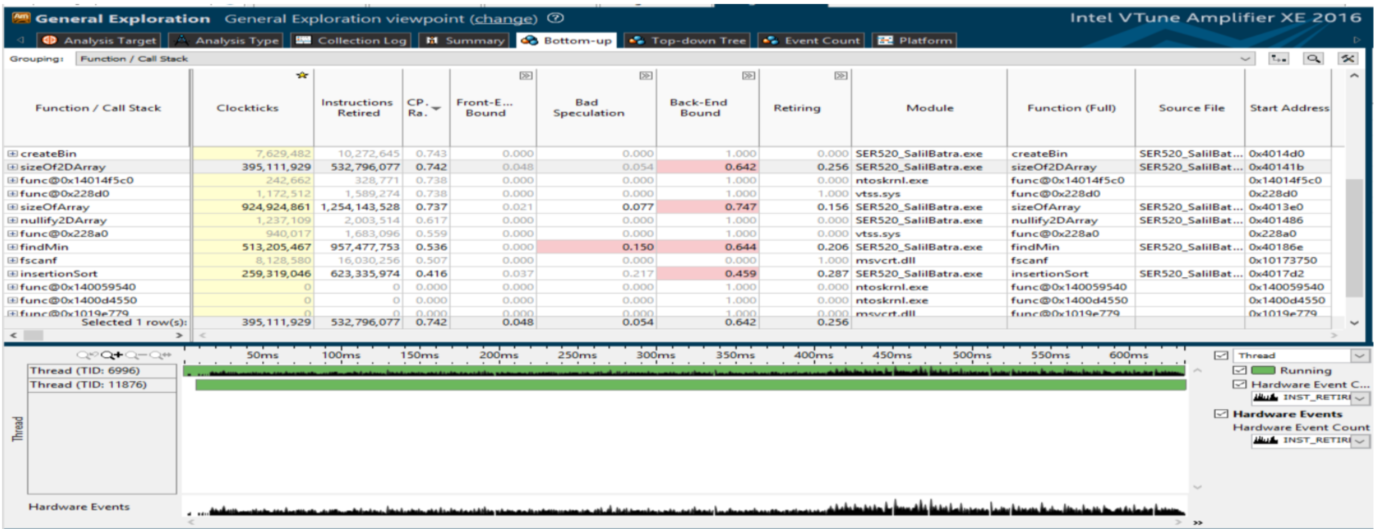
The above CPU usage shows that during the time of execution of the code, the CPU has performed poorly for most of the part.

4 MICRO ARCHITECTURE ANALYSIS

1. GENERAL EXPLORATION:

1.1 Collect Stacks:

Clockticks per Instructions Retired (CPI):



Function / Call Stack	Clockticks	Instructions Retired	CP. Ra.	Front-E... Bound	Bad Speculation	Back-End Bound	Retiring	Module	Function (Full)	Source File	Start Address
createBin	7,629,482	10,272,645	0.743	0.000	0.000	1.000	0.000	SER520_SalilBatra.exe	createBin	SER520_SalilBat...	0x4014d0
sizeOf2DArray	395,111,929	532,796,077	0.742	0.048	0.054	0.642	0.256	SER520_SalilBatra.exe	sizeOf2DArray	SER520_SalilBat...	0x40141b
func@0x14014f5c0	242,662	328,771	0.738	0.000	0.000	1.000	0.000	ntoskrnl.exe	func@0x14014f5c0		0x14014f5c0
func@0x228d0	1,172,512	1,589,274	0.738	0.000	0.000	1.000	1.000	vtss.sys	func@0x228d0		0x228d0
sizeOfArray	924,924,861	1,254,143,528	0.737	0.021	0.077	0.747	0.156	SER520_SalilBatra.exe	sizeOfArray	SER520_SalilBat...	0x4013e0
nullify2DArray	1,237,109	2,003,514	0.617	0.000	0.000	1.000	0.000	SER520_SalilBatra.exe	nullify2DArray	SER520_SalilBat...	0x401486
func@0x228a0	940,017	1,683,096	0.559	0.000	0.000	1.000	0.000	vtss.sys	func@0x228a0		0x228a0
findMin	513,205,467	957,477,753	0.536	0.000	0.150	0.644	0.206	SER520_SalilBatra.exe	findMin	SER520_SalilBat...	0x40186e
fscanf	8,128,580	16,030,256	0.507	0.000	0.000	0.000	1.000	msvcrt.dll	fscanf		0x10173750
insertionSort	259,319,046	623,335,974	0.416	0.037	0.217	0.459	0.287	SER520_SalilBatra.exe	insertionSort	SER520_SalilBat...	0x4017d2
func@0x140059540	0	0	0.000	0.000	0.000	1.000	0.000	ntoskrnl.exe	func@0x140059540		0x140059540
func@0x1400d4550	0	0	0.000	0.000	0.000	1.000	0.000	ntoskrnl.exe	func@0x1400d4550		0x1400d4550
func@0x1019e779	0	0	0.000	0.000	0.000	1.000	0.000	msvcrt.dll	func@0x1019e779		0x1019e779
Selected 1 row(s):	395,111,929	532,796,077	0.742	0.048	0.054	0.642	0.256				

The figure above shows the Bottom-up analysis in the general exploration of collect stack. The CPI rate achieved was 0.625, which according to superscalar architecture should have been (theoretically) 0.25. However, 0.625 is not bad, as a CPI of 1 is still acceptable. The back-end bouds for the functions which took most of the time were sizeOfArray, findMin, sizeOf2dArray, and insertion sort. The back-end bound tells that maximum stalls were observed during the execution of the above functions. The findMin function also had maximum bad speculation rate among other functions.

So. Li.	Source	Cloc... Total	Clockticks: Self	Instr... Retir... Total	Instr... Retir... Self	CPI Rate: Total	CPI Rate: Self	Fr. Bo. To.	Fr. Bo. Sel.	Bad Spec... Total	Bad Spec... Self	Back-E... Bound: Total	Back... Boun... Self	Retiring: Total	Retiri... Self
162	if(numbers[i]<numbers[index]){	20.9%	446,...	24.4%	832,...	0.536	0.536	0.000	0.000	0.194	0.194	0.591	0.591	0.215	0.215

Address	Source Line	Assembly	Cloc... Total	Clockticks: Self	Instructions Retired: Total	Instructio... Retired: Self	CPI Rate: Total	CPI Rate: Self	Front... Bound: Total	Front-E... Bound: Self	Ba. Sp. To.	Bad Specula... Self	Bac.. Bou. Tot..	Ba. Bo. Sel.	Re. To.	Re. Sel.
0x401882	162	mov eax, dword ptr [ebp-0x8]	3.8%	80,204,834	4.2%	144,578,922	0.555	0.555	0.000	0.000	0.000	0.000	1.000	1.000	0.000	0.000
0x401885	162	lea edx, ptr [eax*4]														
0x40188c	162	mov eax, dword ptr [ebp+0x10]	0.0%	126,066	0.0%	216,706	0.582	0.582	0.000	0.000	0.000	0.000	1.000	1.000	0.000	0.000
0x40188f	162	add eax, edx														
0x401891	162	mov edx, dword ptr [eax]	3.6%	77,394,134	4.3%	146,498,092	0.528	0.528	0.000	0.000	0.000	0.000	0.571	0.571	0.864	0.864
0x401893	162	mov eax, dword ptr [ebp-0x4]	9.5%	203,464,786	11.0%	376,517,636	0.540	0.540	0.000	0.000	0.651	0.651	0.347	0.347	0.002	0.002
0x401896	162	lea ecx, ptr [eax*4]														
0x40189d	162	mov eax, dword ptr [ebp+0x10]														
0x4018a0	162	add eax, ecx														
0x4018a2	162	mov eax, dword ptr [eax]	4.0%	85,268,999	4.8%	164,911,056	0.517	0.517	0.000	0.000	0.194	0.194	0.806	0.806	0.000	0.000
0x4018a4	162	cmp edx, eax														
0x4018a6	162	jnl 0x4018ae <Block 4>														

The above screen shot is the most important line of code for findMin function of selection sort, where all the elements of the array are being compared to be put in the correct position. The memory bound statistics show that the significant portion of useful work in the pipleline slot was cancelled during the execution of findMin function.

Function / Call Stack	Clockti...	Instructions Retired	CPI Rate	Front-End Bound: Front-End Latency: ICache Misses	ITLB Ove...	Bran... Rest...	DSB Swit...	Len... Cha.. Prefi...	MS Swit...	Front-End Bandwidth	Bad Speculat...	Back-End Bound: Memory Bound	Core Bound	Retiring	Module	Function (Full)	Source File	Start A
insertionSort	259,319,046	623,335,974	0.416	0.000	0.000	0.000	0.000	0.000	0.000	0.037	0.217	0.457	0.002	0.287	SER520_SalilBatra...	insertionSort	SER520_SalilBat...	0x401
findMin	513,205,467	957,477,753	0.536	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.150	0.405	0.239	0.206	SER520_SalilBatra...	findMin	SER520_SalilBat...	0x401
sizeOfArray	924,924,861	1,254,143,5...	0.737	0.000	0.000	0.000	0.000	0.000	0.000	0.021	0.077	0.706	0.040	0.156	SER520_SalilBatra...	sizeOfArray	SER520_SalilBat...	0x401

Memory bound statistics also shows that the maximum fraction of cycles where the pipeline was stalled was in sizeOfArray function .

The least amount of time is taken by the createBin function as the main part of the function only checks if element in the data exists and if yes, then it increases its count by 1. The following are the screen shots for the createBin function.

So. Li.	Source	Cloc... Total	Clo. Self	Instr... Retir... Total	Instr... Retir... Self	CPI Rate: Total	CPI Rate: Self	Front-E... Bound: Total	Fron... Bou... Self	Bad Spe.. Total	Bad Specu... Self	Back-E... Bound: Total	Back... Boun... Self	Re. To.	Retiring: Self
72	if(array[i][0] == x){	0.3%	7,24...	0.3%	9,79...	0.740	0.740	0.000	0.000	0.000	0.000	1.000	1.000	0.000	0.0

0.101ms was spent on bad speculation during execution. Out of 0.101ms, no time was spent due to branch misprediction and all the time was spent on machine clears.

Sour... Line	Source	Clockti... Total	Clockti... Self	Instr... Retir... Total	Instr... Retir... Self	CPI Rate: Total	CPI Rate: Self	Fr. Bo. To.	Fr. Bo. Sel.	Ba. Sp. To.	Ba. Sp. Sel.	Ba. Bo. To.	Ba. Bo. Sel.	Ret... Total	Reti... Self
40	if(array[i] == NULL)	33.7%	718,078,665	28.6%	974, ...	0.737	0.737	0.027	0.027	0.000	0.000	0.801	0.801	0.200	0.200

The metric is measured for three events- memory ordering violations, self-modifying code, and certain loads to illegal address ranges. The above line of code is the part of sizeOfArray function, a critical function for the entire code, returns the length of the array and is further used by functions as a terminating condition for for-loop.

<u>Back-end Bound: 0.668</u> – *"Identify slots where no uOps are delivered due to a lack of required resources for accepting more uOps in the back-end of the pipeline."*

a. <u>Memory Bound- 0.484:</u> As discussed above, the findMin functions has the highest memory bound metric.
b. <u>L1 cache – 0.584:</u> This metric shows how often the machine was stalled without missing the L1 cache. For this metric the observed code is line 40 as shown in the last screen shot.

Function / Call Stack	Clockti...	Instructions Retired	CPI Rate	Front-End Bound: Front-End Latency: ICache Misses	ITLB Ove...	Bran... Rest...	DSB Swit...	Len... Cha.. Prefi...	MS Swit...	Front-End Bandwidth	Bad Speculation: Branch Mispredict	Machine Clears	Back-End Bound: Memory Bound: L1 Bo.	L3 Bound: Con... Acc...	Data Shar...	LLC Hit	SQ Full	DRA. Mem. LLC ...	St. Bo.	Core Bound	Retiri...	Module	Func
insertionSort	259,319,046	623,335,974	0.416	0.000	0.000	0.000	0.000	0.000	0.000	0.037	0.000	0.217	0.147	0.000	0.000	0.000	0.000	0.000	0.000	0.002	0.287	SER520_Sal...	insertio
findMin	513,205,467	957,477,753	0.536	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.150	0.426	0.000	0.000	0.000	0.000	0.000	0.000	0.239	0.206	SER520_Sal...	findMin
sizeOfArray	924,924,861	1,254,143,5...	0.737	0.000	0.000	0.000	0.000	0.000	0.000	0.021	0.000	0.077	0.699	0.000	0.000	0.000	0.000	0.000	0.000	0.040	0.156	SER520_Sal...	sizeOfA

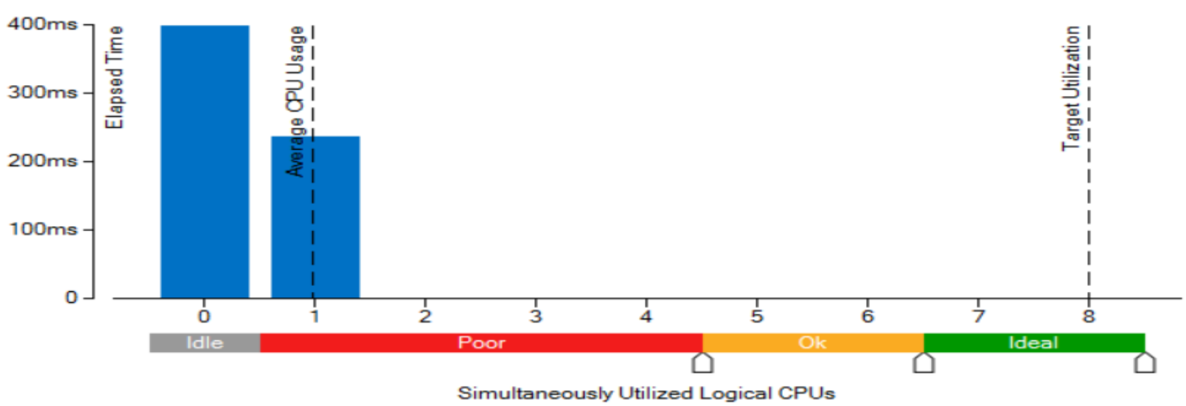
<u>Core Bound – 0.184:</u> The time spent on core non-memory bottleneck issues and these may arise due to hardware compute resources or dependencies on software's instructions. The core bound metric points to the <u>ntdll.dll file</u> which is not a part of the code used for analysis.

Function Range / Basic Block / Address	Source Line	Assembly	Clockticks	Instructions Retired	CPI Rate	Front-End Bound	Bad Speculati...	Back-E... Bound	Retiring
0x180025250	0	⊟Function range 0x180025250-0x1800253c9	2,426,662	1,178,859	2.058	0.000	0.000	1.000	0.000
0x180025250	0	⊟Block 1	2,426,662	1,178,859	2.058	0.000	0.000	1.000	0.000
0x180025272		mov r15, r9	2,426,662	1,178,859	2.058	0.000	0.000	1.000	0.000

Port Utilization – 0.221: Represents the fraction of cycles during which an application was stalled due to core non-divider-related issues. The analysis points to the ntoskrnl.exe file which is not the part of code used to perform the analysis.

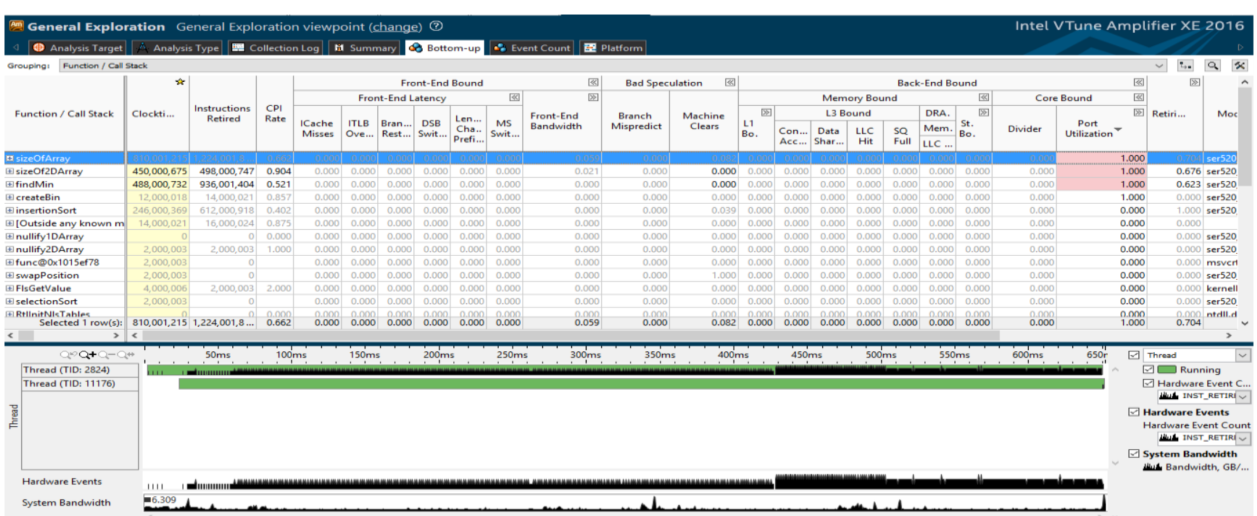
Retiring – 0.208: The total number of threads are 2 and there was no paused time reported.

CPU Usage Histogram:

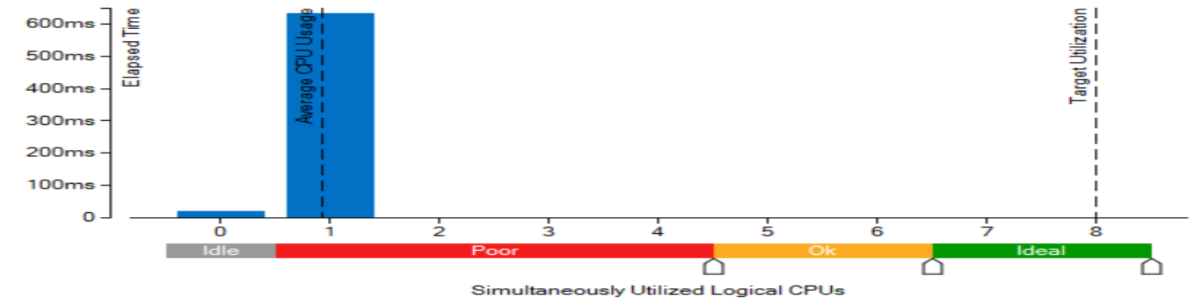


1.2 Analyze Memory Bandwidth: The memory bandwidth data shows that the CPI rate of 0.616 and MUX reliability of 0.399. The front-end and back-end bounds have 0.028 and 0.019 respectively.

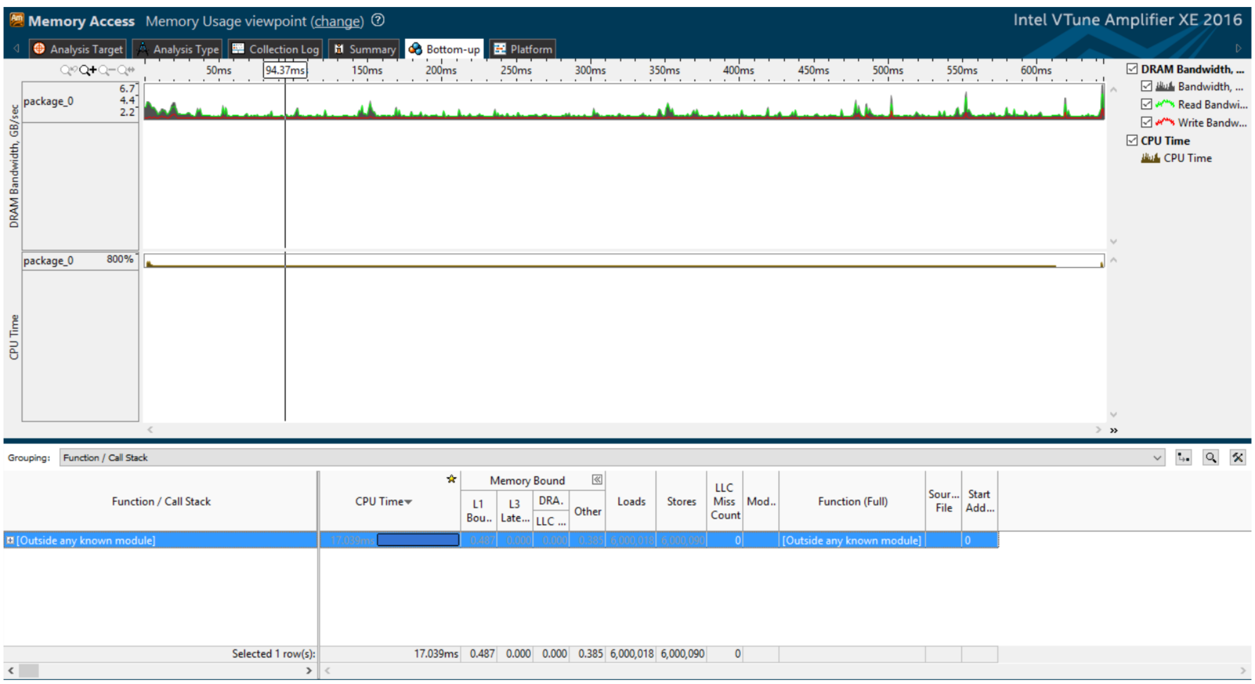
The port utilization of 1.000 is observed due to functions sizeOfArray, sizeOf2DArray, and findMin functions.



CPU Usage Histogram:

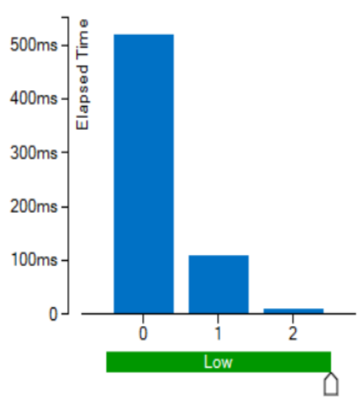


Memory Access Data:

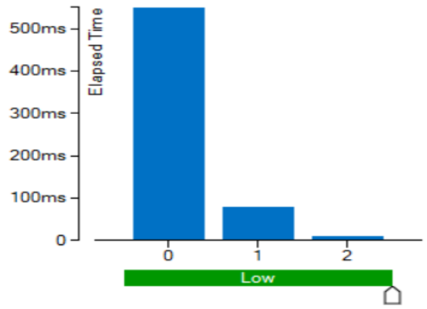


Bandwidth Utilization Histogram:

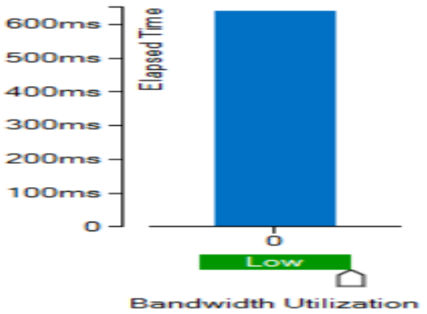
Bandwidth Domain – DRAM, GB/sec



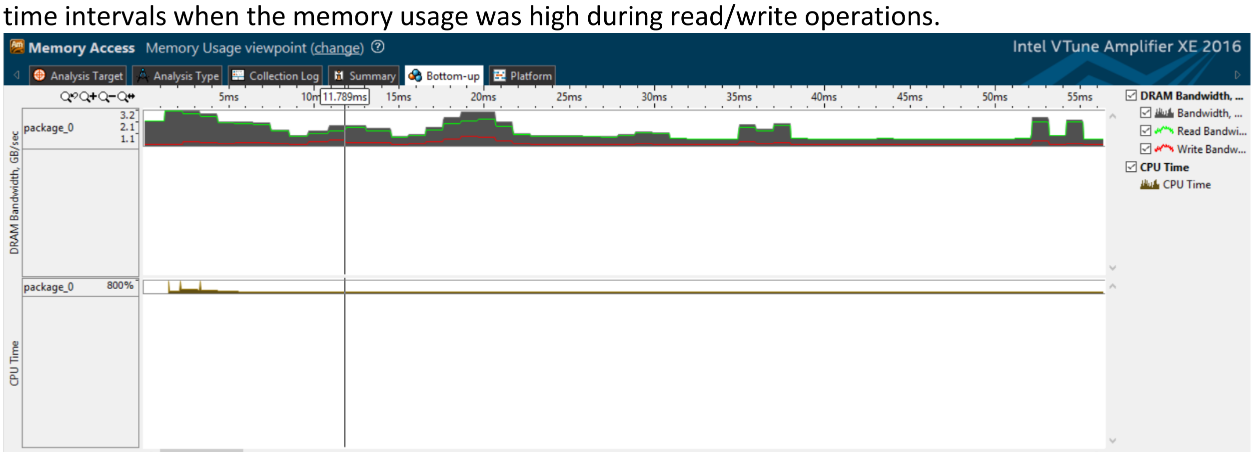
Bandwidth Domain – DRAM Read, GB/sec



Bandwidth Domain – DRAM Write, GB/sec



The screenshot below shows the Read (green line) and Write (blue line) for the functions which have highest memory usage. The interval with highest spikes was selected to obtain the data and it shows the time intervals when the memory usage was high during read/write operations.



→TSX (Intel Transactional Synchronization Extensions) Exploration and TSX Hotspots were not available.

5 SUMMARY:

The code used to get the statistics did not have any print statement to provide the output of the data. However, once the print statement is included to print the data for mean, standard deviation, mode, variance, insertion sort, and selection sort, the execution time is increased significantly. Also, the top hotspots functions includes the print function as the maximum time is spent printing the huge data.

The hotspot analysis has clearly indicate the performance of the application and has also shown the hardware performance. This analysis has clearly indicated the inter-dependencies of hardware and software and how a process can halt the performance of the overall system. Vtune analysis also helped to pin point the functions which consume the maximum resources, for example, printfunction which was earlier included in the code, and helps developers to optimize their code.