

USING INTEL® VTUNE™ AMPLIFIER FOR PROFILING AND OPTIMIZATIONS

Wang Yang
Intel IAGS CPDP
August, 2019



Intel® VTune™ Amplifier – Tool Suite Options



HPC, Enterprise, & Cloud

INTEL® PARALLEL STUDIO

PROFESSIONAL & CLUSTER EDITIONS

Improve performance, scalability, & reliability for parallel applications



intel.ly/parallel-studio-xe



Manuf., Retail, Drones, Robots...

INTEL® SYSTEM STUDIO

PROFESSIONAL & ULTIMATE EDITIONS

Develop IOT/embedded & system solutions & apps faster



intel.ly/system-studio



Fast, Dense, High Quality Transcoding

INTEL® MEDIA SERVER STUDIO

PROFESSIONAL EDITION

Deliver fast, high density & quality media/video processing



intel.ly/intel-media-server-studio

INTEL® VTUNE™ AMPLIFIER

AVAILABLE INDIVIDUALLY

Analyze & Tune Application Performance & Scalability

Provides deep insight that saves time optimizing code



intel.ly/vtune-amplifier-xe

[Free/discounted versions are available for Students & Academia](#)

Optimization Notice

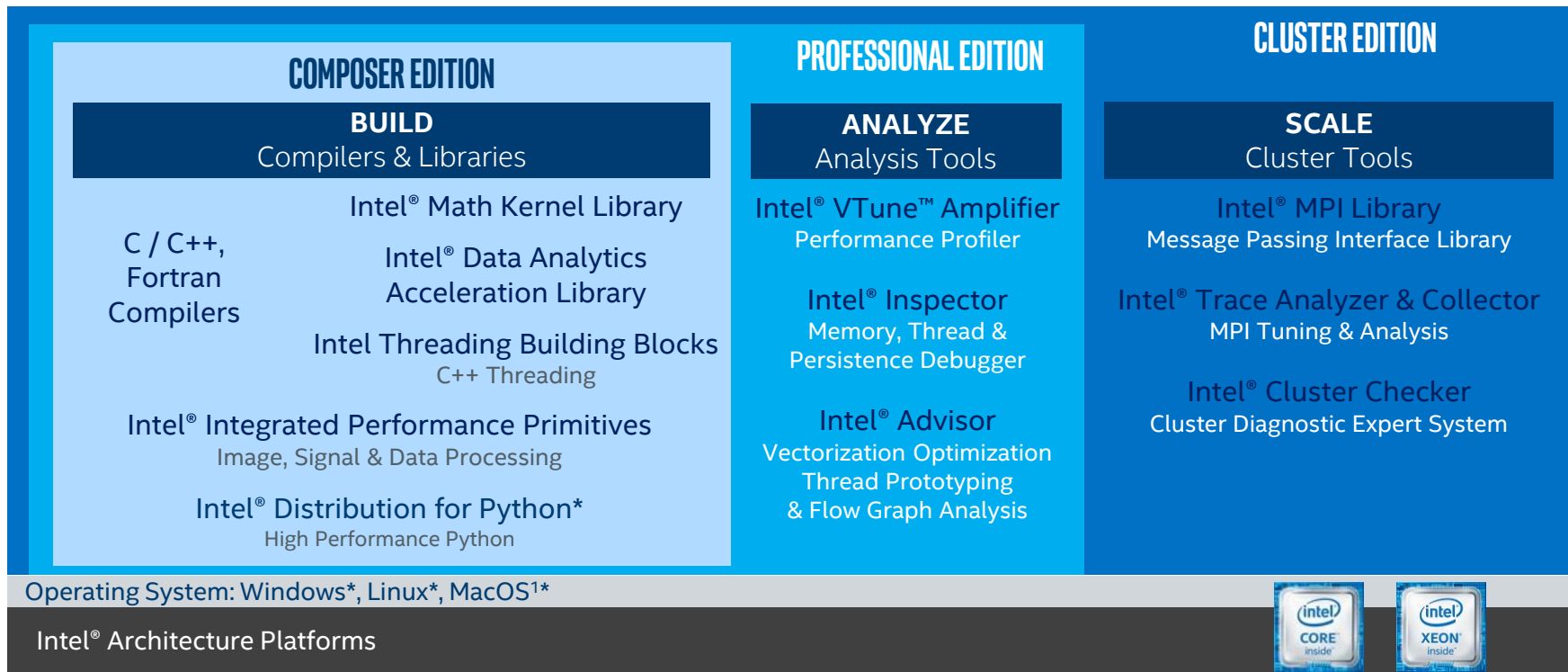
Copyright © 2018, Intel Corporation. All rights reserved.

*Other names and brands may be claimed as the property of others.



What's Inside Intel® Parallel Studio XE

Comprehensive Software Development Tool Suite



Optimization Notice

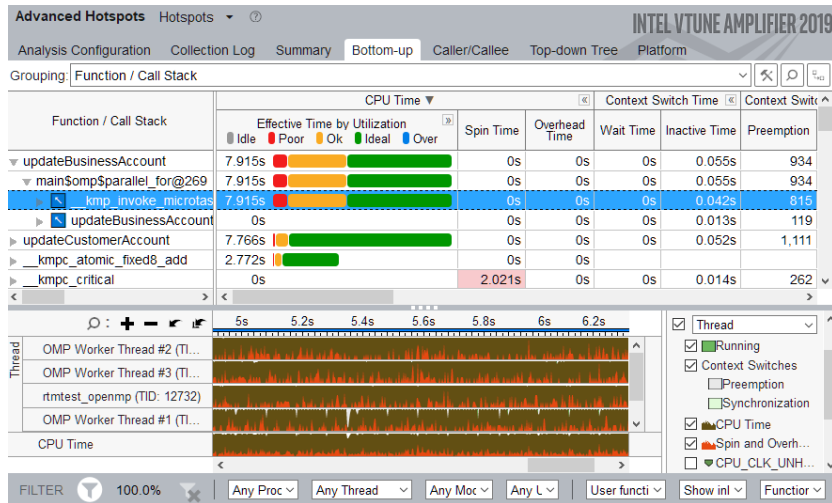
Copyright © 2018, Intel Corporation. All rights reserved.

*Other names and brands may be claimed as the property of others.



Analyze & Tune Application Performance

Intel® VTune™ Amplifier—Performance Profiler



Save Time Optimizing Code

- Accurately profile C, C++, Fortran*, Python*, Go*, Java*, or any mix
- Optimize CPU, threading, memory, cache, storage & more
- Save time: rich analysis leads to insight
- Take advantage of [Priority Support](#)
 - Connects customers to Intel engineers for confidential inquiries (paid versions)

What's New in 2019 Release (partial list)

- New Platform Profiler! - Longer Data Collection
- A more accessible user interface provides a simplified profiling workflow
- Smarter, faster Application Performance Snapshot: Analyze CPU utilization of physical cores, pause/resume, more... (Linux*)
- Improved JIT profiling for server-side/cloud applications

Learn More: software.intel.com/intel-vtune-amplifier-xe

Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.

*Other names and brands may be claimed as the property of others.



Rich Set of Profiling Capabilities for Multiple Markets

Intel® VTune Amplifier



Single Thread

Optimize single-threaded performance.



Multithreaded

Effectively use all available cores.



System

See a system-level view of application performance.



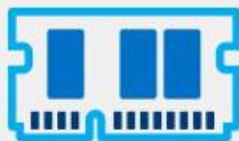
Media & OpenCL™ Applications

Deliver high-performance image and video processing pipelines.



HPC & CCloud

Access specialized, in-depth analyses for HPC and cloud computing.



Memory & Storage Management

Diagnose memory, storage, and data plane bottlenecks.



Analyze & Filter Data

Mine data for answers.



Environment


Fits your environment and workflow.

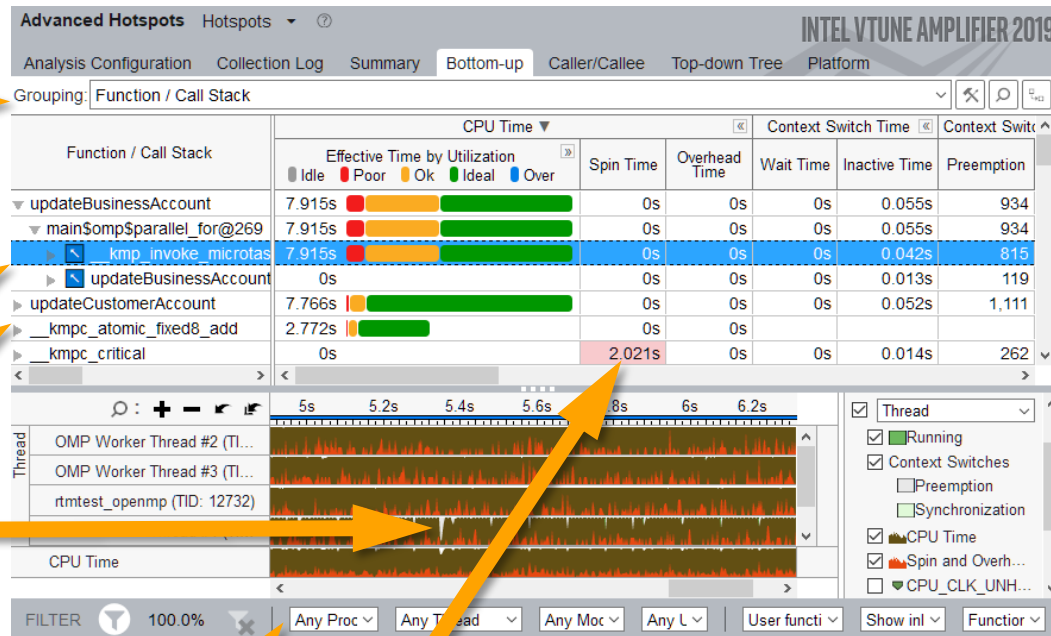
Intel® VTune™ Amplifier

- Function - Call Stack
- Module - Function - Call Stack
- Source File - Function - Call Stack
- Thread - Function - Call Stack
- ... (Partial list shown)

Click [▶] for Call Stack

Filter by Timeline Selection (or by Grid Selection)

Filter In by Selection 



Filter by Process & Other Controls

Tuning Opportunities Shown in Pink.
Hover for Tips

See Profile Data On Source / Asm

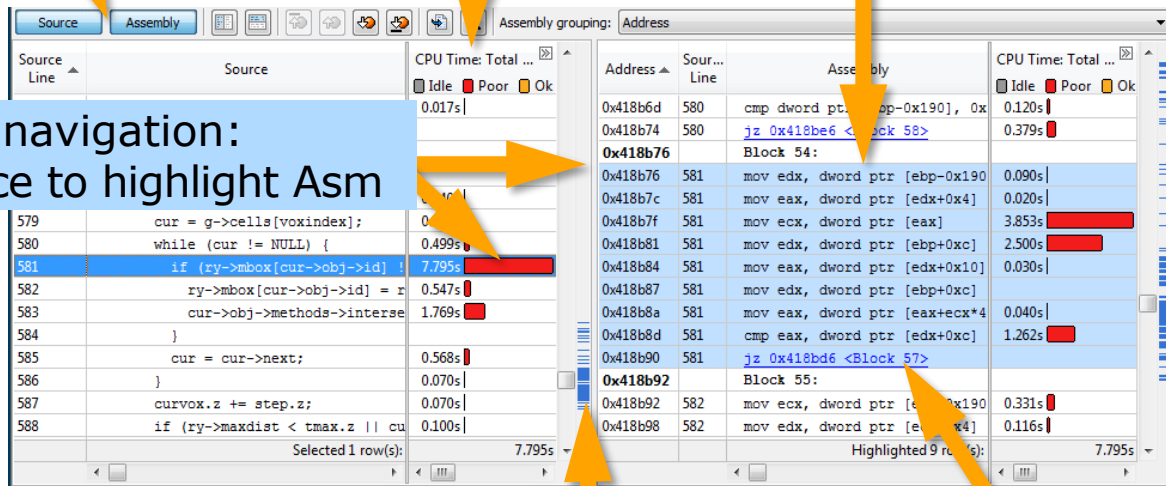
Double Click from Grid or Timeline

View Source / Asm or both

CPU Time

Right click for instruction reference manual

Quick Asm navigation:
Select source to highlight Asm



Scroll Bar "Heat Map" is an overview of hot spots

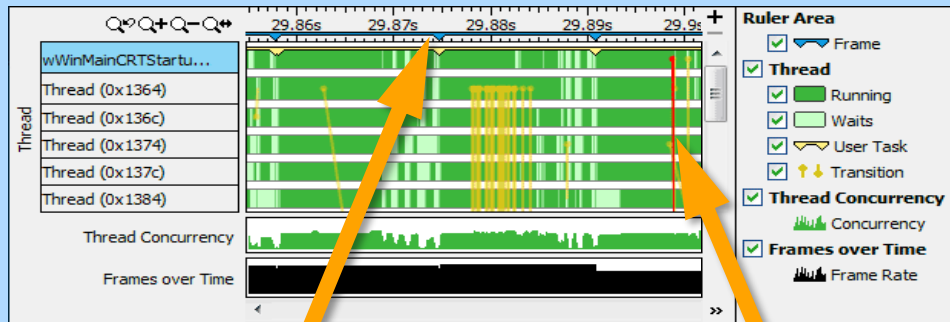
Click jump to scroll Asm

Timeline Visualizes Thread Behavior

Intel® VTune™ Amplifier

🔑 Transitions

Locks & Waits



Hovers:

Frame

Frame
Start: 29.858s Duration: 0.017s
Frame: 72
Frame Domain: Smoke::Framework::execute()
Frame Type: Good
Frame Rate: 59.8242179

Transition

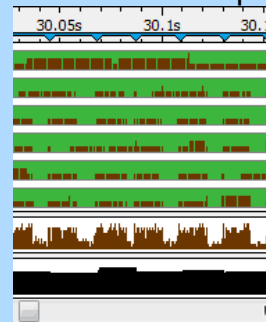
Transition
wWinMainCRTStartup (0x12d4) to Thread (0x138c) (29.899s to 29.899s)
Sync Object: TBB Scheduler
Object Creation File: taskmanagetbb.cpp
Object Creation Line: 318

CPU Time

Basic Hotspots



Advanced Hotspots



User Task

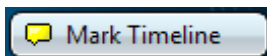
User Task
Start: 29.958s Duration: 0.018s
Task Type: Smoke::Framework::execute()::Other
Task End Call Stack: Framework::Execute

CPU Time
94.233472%

Optional: Use API to mark frames and user tasks



Optional: Add a mark during collection



Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.



Intel® VTune™ Amplifier 2019

Easier Setup, More Intelligible Results

Fresh, Accessible Analysis Setup

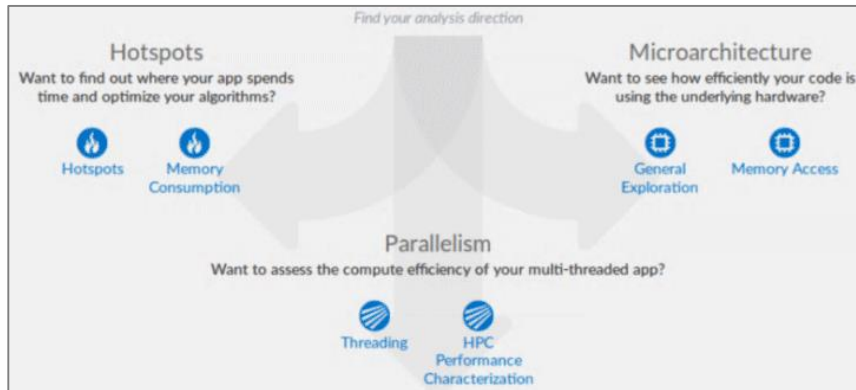
- Simplified workflow
- More familiar terminology
- More logical groupings

Performance Insights

- Suggestions for further analysis

Improved Displays

- New hardware pipeline display



Hotspots Insights

If you see significant hotspots in the Top Hotspots list, switch to the [Bottom-up](#) view for in-depth analysis per function. Otherwise, use the [Caller/Callee](#) view to track critical paths for these hotspots.

Explore Additional Insights

Parallelism ⓘ : 17.8% (15.622 out of 88 logical CPUs) 📈

Use [Concurrency](#) to explore more opportunities to increase parallelism in your application.

Intel® VTune™ Amplifier Performance Profiler 2019

New workflow provides easier to learn tuning workflow and a simplified setup

The screenshot shows the Intel VTune Amplifier 2019 'Configure Analysis' window. The interface is divided into several sections:

- WHERE:** Options for 'Local Host', 'Android Device (ADB)', and 'Remote Linux (SSH)'. Below these are fields for 'VTune Amplifier installation directory on the remote system' and 'Temporary directory on the remote system'.
- WHAT:** Options for 'Attach to Process', 'Profile System', and 'Launch Application'. Below these are fields for 'Specify Application' and 'Application parameters'.
- Find your analysis direction:** A central area with icons for 'Hotspots', 'Microarchitecture', 'Parallelism', 'Platform Analysis', and 'Custom Analysis'. The 'Hotspots' icon is circled in red.
- Start/Run Controls:** A row of buttons at the bottom right: 'Start' (play icon), 'Start Paused' (play icon with a pause symbol), 'Search Binaries Search Source' (magnifying glass icon), and 'Command Line' (terminal icon).

Annotations with orange arrows point to specific elements:

- 'where' points to the 'WHERE' section.
- 'what' points to the 'WHAT' section.
- 'how' points to the 'Find your analysis direction' section.
- 'run' points to the 'Start' button.

Hotspots Analysis – Your first step for optimization

HOW

Use this mode for:

- Profiles longer than a few seconds
- Profiling a single process or a process-tree
- Profiling Python and Intel runtimes

Use this mode for:

- Profiles shorter than a few seconds
- Profiling all processes on a system, including kernel



Hotspots



Identify the most time consuming functions and drill down to see time spent on each line of source code. Focus optimization efforts on hot code for the greatest performance impact. [Learn more](#)



User-Mode Sampling ?



Hardware Event-Based Sampling ?

CPU sampling interval, ms

1

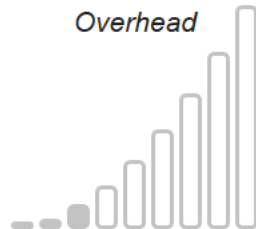


Collect stacks



Show additional performance insights

Overhead



► Details

Case Study: Hotspot Analysis

- Case about Adler32 checksum calculation
- A developer who works on storage applications needs to calculate the Adler32 checksum for big files
- Opensource Adler32 implementation was used: [Source Code Here](https://github.com/madler/zlib/blob/master/adler32.c)
- Run the VTune “Advance Hotspots” analysis for the sample application and investigate the result

GitHub, Inc. [US] | <https://github.com/madler/zlib/blob/master/adler32.c>

```
88      /* initial Adler-32 value (deferred check for len == 1 speed) */
89      if (buf == Z_NULL)
90          return 1L;
91
92      /* in case short lengths are provided, keep it somewhat fast */
93      if (len < 16) {
94          while (len-- > 0) {
95              adler += *buf++;
96              sum2 += adler;
97          }
98          if (adler >= BASE)
99              adler -= BASE;
100          MOD28(sum2);          /* only added so many BASE's */
101          return adler | (sum2 << 16);
102      }
103
104      /* do length NMAX blocks -- requires just one modulo operation */
105      while (len >= NMAX) {
106          len -= NMAX;
107          n = NMAX / 16;        /* NMAX is divisible by 16 */
108          do {
109              DO16(buf);        /* 16 sums unrolled */
110              buf += 16;
111          } while (--n > 0);
112          MOD(adler);
113          MOD(sum2);
114      }
115
116      /* do remaining bytes (less than NMAX, still just one modulo) */
117      if (len) {                /* avoid modulus if none remaining */
```

Hotspots for Adler32 opensource implementation

Advanced Hotspots Hotspots viewpoint (change) ? INTEL VTUNE AMPLIFIER 2017 FOR

Collection Log Analysis Target Analysis Type Summary Bottom-up Caller/Callee Top-down Tree Platform adler32.c

Grouping: Process / Module / Function / Thread / Call Stack

Process / Module / Function / Thread / Call ...	CPU Time ▾	Instructions Retired	CPI Rate	CPU Frequency Ratio	Module
▼ test_adler32_os	46.567s	353,579,000,000	0.468	1.550	
▶ libc-2.17.so	32.954s	241,412,600,000	0.486	1.551	
▼ test_adler32_os	12.207s	111,890,400,000	0.386	1.542	
▶ adler32	8.695s	85,502,500,000	0.352	1.509	test_adl.
▶ test_5times	2.170s	17,866,400,000	0.44	1.581	test_adl.
▶ [Import thunk rand]	1.317s	8,312,200,000	0.614	1.689	test_adl.
▶ test_5times	0.019s	158,700,000	0.594	2.158	test_adl.
▶ test_5times	0.004s	18,400,000	0.500	1.000	test_adl.
▶ test_5times	0.001s	29,900,000	0.231	3.000	test_adl.
▶ test_5times	0.001s	2,300,000	1.000	1.000	test_adl.

Function to be
optimized

CPU Time and
Instructions

CPI is relatively
good

Source/Assembly correlated view in VTune

Advanced Hotspots Hotspots viewpoint (change) ? INTEL VTUNE AMPLIFIER 2017 FOR SYSTEM

Collection Log Analysis Target Analysis Type Summary Bottom-up Caller/Callee Top-down Tree Platform adler32.c adl

Source Assembly Assembly grouping: Address

So. Li.	Source	Effectiv	Address	Sour... Line	Assembly	Effectiv
		Idle Poor				Idle Poor
103	if (adler >= BASE)		0x400e20		Block 12:	
104	adler -= BASE;		0x400e20	114	movzxb (%r11), %r13d	5.012ms
105	MOD28(sum2); /* only added so mar		0x400e24	115	add \$0x10, %r11	115.265ms
106	return adler (sum2 << 16);		0x400e28	114	movzxb -0xf(%r11), %r14d	
107	}		0x400e2d	114	movzxb -0xe(%r11), %r12d	314.725ms
108			0x400e32	114	movzxb -0xd(%r11), %ebp	2.005ms
109	/* do length NMAX blocks -- requires just one mc		0x400e37	114	movzxb -0xc(%r11), %ebx	
110	while (len >= NMAX) {	0.002s	0x400e3c	114	movzxb -0xb(%r11), %r10d	2.005ms
111	len -= NMAX;	0.002s	0x400e41	114	add %r15, %r13	282.651ms
112	n = NMAX / 16; /* NMAX is divisible		0x400e44	114	movzxb -0xa(%r11), %r9d	3.007ms
113	do {		0x400e49	114	movzxb -0x9(%r11), %r8d	
114	DO16(buf); /* 16 sums unrolled	5.328s	0x400e4e	114	add %r13, %r14	
115	buf += 16;	0.115s	0x400e51	114	movzxb -0x8(%r11), %eax	309.713ms
116	} while (--n);	0.608s	0x400e56	114	movzxb -0x7(%r11), %edi	0ms
117	MOD(adler);	0.004s	0x400e5b	114	add %r14, %r12	0ms
118	MOD(sum2);	0.013s	0x400e5e	114	add %r14, %r13	
119	}		0x400e61	114	movzxb -0x5(%r11), %esi	280.646ms
120			0x400e66	114	add %r12, %rbp	1.002ms
121	/* do remaining bytes (less than NMAX, still jus		0x400e69	114	add %r12, %r13	
122	if (len) { /* avoid modulus if		0x400e6c	114	movzxb -0x4(%r11), %ecx	
123	while (len >= 16) {	0.010s	0x400e71	114	add %rbp, %rbx	327.755ms
124	len -= 16;		0x400e74	114	add %rbp, %r13	1.002ms
125	DO16(buf);	2.272s	0x400e77	114	movzxb -0x3(%r11), %edx	0ms

Check the assembly code and we find the loop is not vectorized

Use the optimized Adler32 function from Intel IPP library should help!

Switch to the IPP implementation

```
/* =====  
uLong ZEXPORT Adler32(adler, buf, len)  
    uLong adler;  
    const Bytef *buf;  
    uInt len;  
{  
    unsigned long sum2;  
    unsigned n;  
  
    /* split Adler-32 into component sums */  
    sum2 = (adler >> 16) & 0xffff;  
    adler &= 0xffff;  
  
    ...  
  
    /* initial Adler-32 value (deferred check for len == 1 speed) */  
    if (buf == Z_NULL)  
        return 1L;  
  
    ...  
  
    /* do length NMAX blocks -- requires just one modulo operation */  
    while (len >= NMAX) {  
        len -= NMAX;  
        n = NMAX / 16;          /* NMAX is divisible by 16 */  
        do {  
            D016(buf);          /* 16 sums unrolled */  
            buf += 16;  
        } while (--n);  
        MOD(adler);  
        MOD(sum2);  
    }  
  
    ...  
    /* return recombined sums */  
    return adler | (sum2 << 16);  
}
```



```
#include "ippdc.h"  
  
uLong ZEXPORT Adler32_ipp(adler, buf, len)  
    uLong adler;  
    const Bytef *buf;  
    uInt len;  
{  
    Ipp32u resAdler32 = (Ipp32u)adler;  
    if( Z_NULL == buf ) return 1L;  
    ippAdler32_8u(buf, len, &resAdler32);  
    return ((uLong)resAdler32 & 0xffffffff);  
}
```

Use the optimized IPP
function to take advantage
of HW features!

Vectorized code can get significant performance

Grouping: Process / Module / Function / Thread / Call Stack			
Process / Module / Function / Thread / Call ...	CPU Time ▾	Instructions Retired	CPI Rate
▼ test_adler32_ipp	38.907s	290,876,400,000	0.476
▶ libc-2.17.so	31.206s	241,159,600,000	0.462
▼ test_adler32_ipp	6.348s	10,450,000,000	0.449
▶ test_5times	2.475s	14,837,300,000	0.577
▶ 19_ownsAdler32_8u	2.384s	22,703,300,000	0.364
▶ [Import thunk rand]	1.431s	11,582,800,000	0.449
▶ test_5times	0.026s	149,500,000	0.677
▶ adler32_ipp	0.012s	78,200,000	0.382

- CPU Time reduced from 8.6s to 2.3s, +3.7times improvement
- Instructions reduced, 85G→22G

Optimized with AVX
SIMD instructions

Address ▲		Sour... Line	Assembly	CPU Time	
				Effective Time by Utilization	
				Idle Poor Ok Ideal Over	
				1.002ms	
0x402014			nopl %eax, (%rax,%rax,1)		
0x402019			nopl %eax, (%rax)		
0x402020			Block 14:		
0x402020			vmovdqux (%rdi), %xmm1	36.083ms	
0x402024			vpshld \$0x4, %xmm0, %xmm7	402.928ms	
0x402029			vpunpcklbw %xmm2, %xmm11, %xmm5	382.882ms	
0x40202d			vpadd %xmm7, %xmm1, %xmm1	57.132ms	
0x402031			vpaddw %xmm4, %xmm5, %xmm8	170.392ms	
0x402035			add \$0x10, %rdi	63.145ms	
0x402039			vpasdbw %xmm2, %xmm11, %xmm12	46.106ms	
0x40203d			vpunpckhbw %xmm2, %xmm11, %xmm6	43.099ms	
0x402041			vpadd %xmm8, %xmm1, %xmm9	207.478ms	
0x402046			vpaddw %xmm3, %xmm6, %xmm10	39.090ms	
0x40204a			vpadd %xmm12, %xmm0, %xmm0	44.102ms	
0x40204f			vpadd %xmm10, %xmm9, %xmm1	28.065ms	
0x402054			dec %eax	207.478ms	
0x402056			jnz 0x402020 <Block 14>		
0x402058			Block 15:		

Microarchitecture Exploration – The way to check CPU execution efficiency

<https://software.intel.com/en-us/vtune-amplifier-help-microarchitecture-exploration-analysis>

<https://software.intel.com/en-us/vtune-amplifier-help-tuning-applications-using-a-top-down-microarchitecture-analysis-method>

<https://software.intel.com/en-us/articles/processor-specific-performance-analysis-papers>



Microarchitecture Exploration



Analyze CPU microarchitecture bottlenecks affecting the performance of your application. This analysis type is based on the hardware event-based sampling collection. [Learn more](#)

⚠ CPU frequency data collection is not supported on this platform.

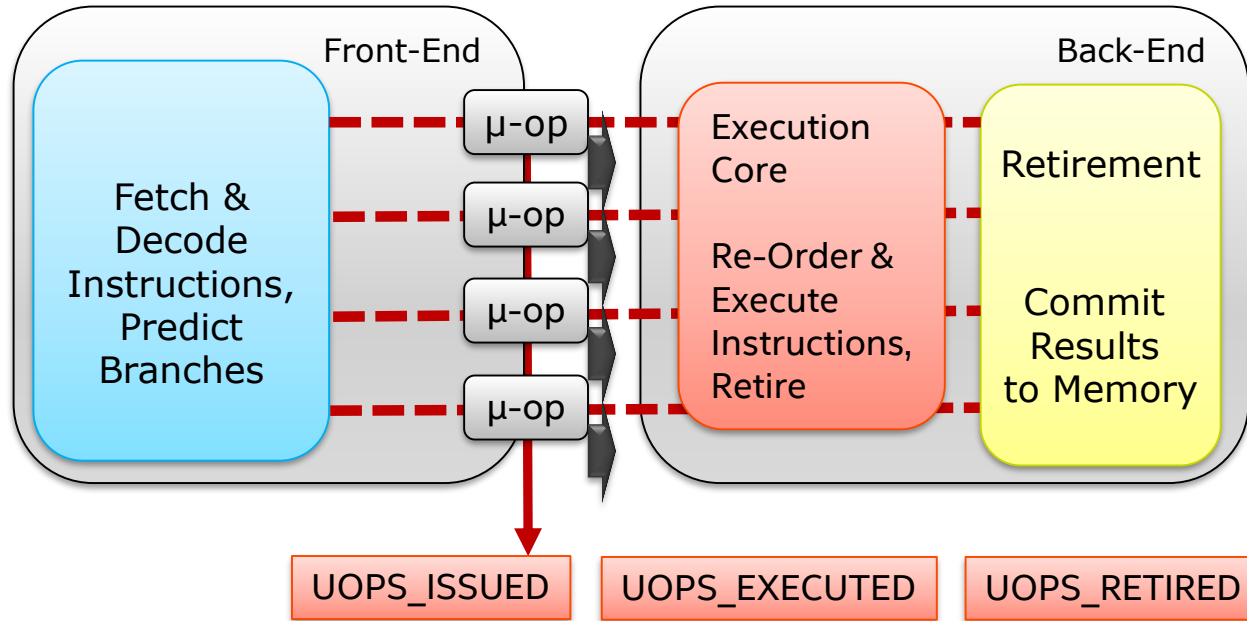
CPU sampling interval, ms

10

Extend granularity for the top-level metrics:

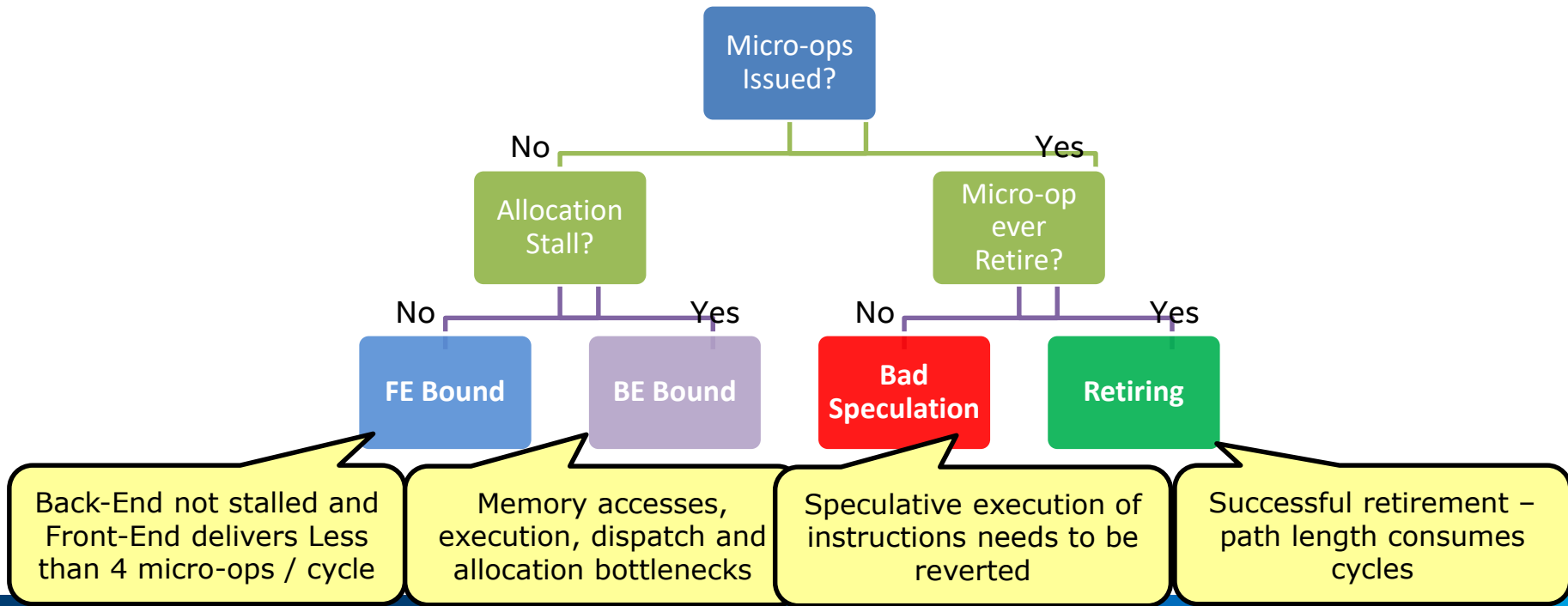
- ☒ Front-End Bound
- ☒ Bad Speculation
- ☒ Memory Bound
- ☒ Core Bound
- ☒ Retiring

A simplified CPU execution pipeline flow



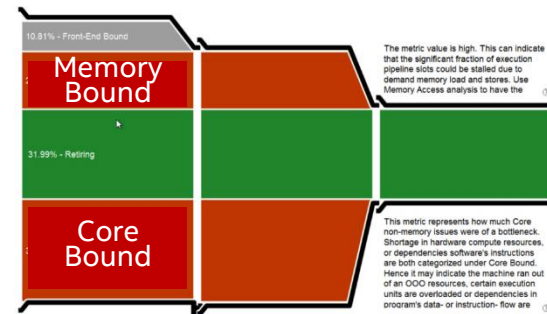
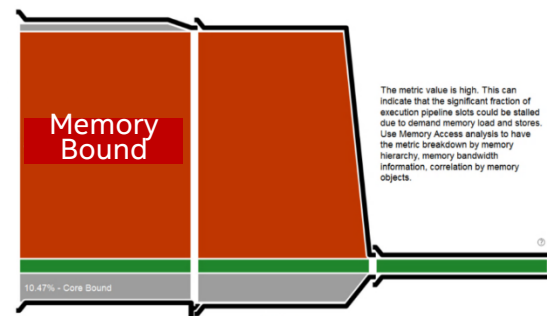
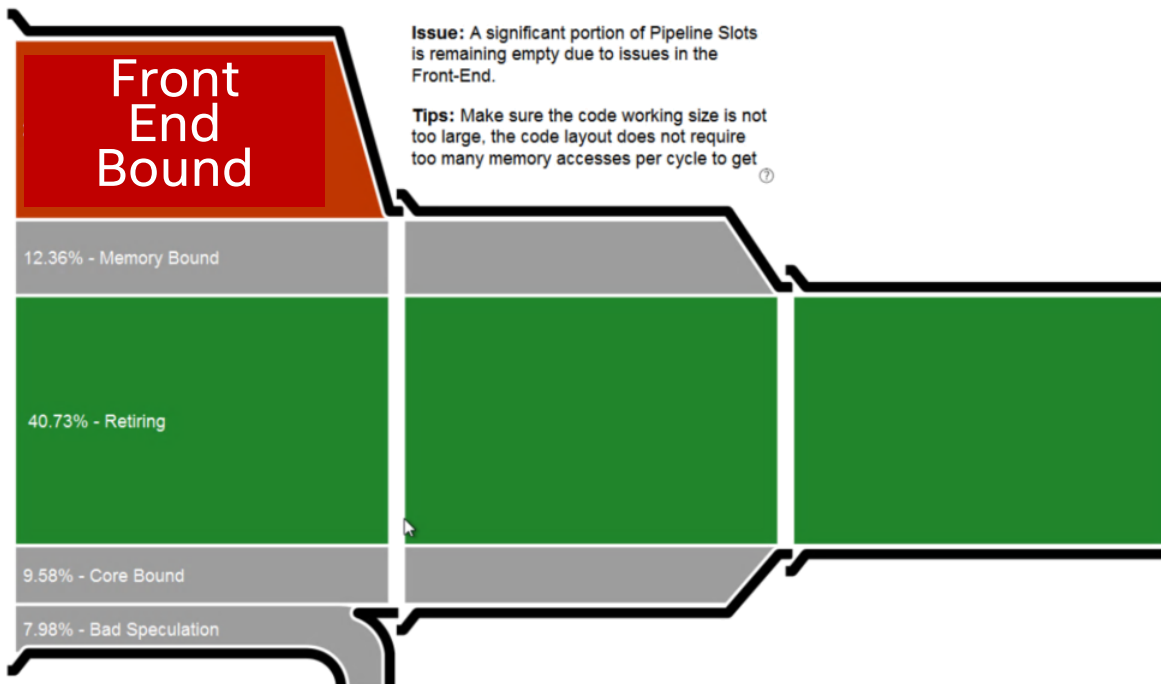
Bottleneck Domain – A Top-Down hierarchy

Performance is classified according to what happened for each slot available to the application or hotspot:



Visualize the Micro-Architectural Bottleneck

Intel® VTune™ Amplifier – Performance Profiler



Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.

*Other names and brands may be claimed as the property of others.

Case Study: Microarchitecture exploration analysis

- It is real case from stackoverflow:

<http://stackoverflow.com/questions/11227809/why-is-processing-a-sorted-array-faster-than-an-unsorted-array>

- Run the 'sumtest' with "General Exploration" analysis and investigate the result

Why a 'sort' on 'data' make huge performance difference?

Here is a piece of **C++** code that seems very peculiar. For some strange reason, sorting the data miraculously makes the code almost six times faster.

```
#include <algorithm>
#include <ctime>
#include <iostream>

int main()
{
    // Generate data
    const unsigned arraySize = 32768;
    int data[arraySize];

    for (unsigned c = 0; c < arraySize; ++c)
        data[c] = std::rand() % 256;

    // !!! With this, the next loop runs faster
    std::sort(data, data + arraySize);

    // Test
    clock_t start = clock();
    long long sum = 0;

    for (unsigned i = 0; i < 100000; ++i)
    {
        // Primary loop
        for (unsigned c = 0; c < arraySize; ++c)
        {
            if (data[c] >= 128)
                sum += data[c];
        }
    }

    double elapsedTime = static_cast<double>(clock() - start) / CLOCKS_PER_SEC;

    std::cout << elapsedTime << std::endl;
    std::cout << "sum = " << sum << std::endl;
}
```

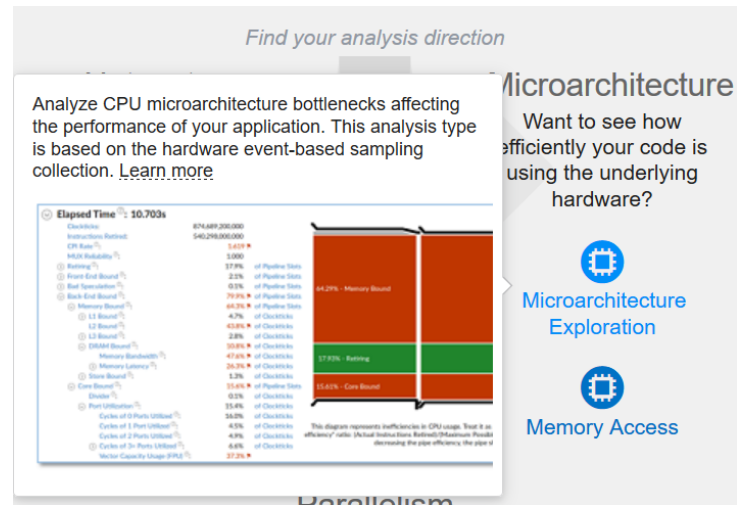
- Without `std::sort(data, data + arraySize);`, the code runs in 11.54 seconds.
- With the sorted data, the code runs in 1.93 seconds.

How can VTune Amplifier help to identify the root cause?

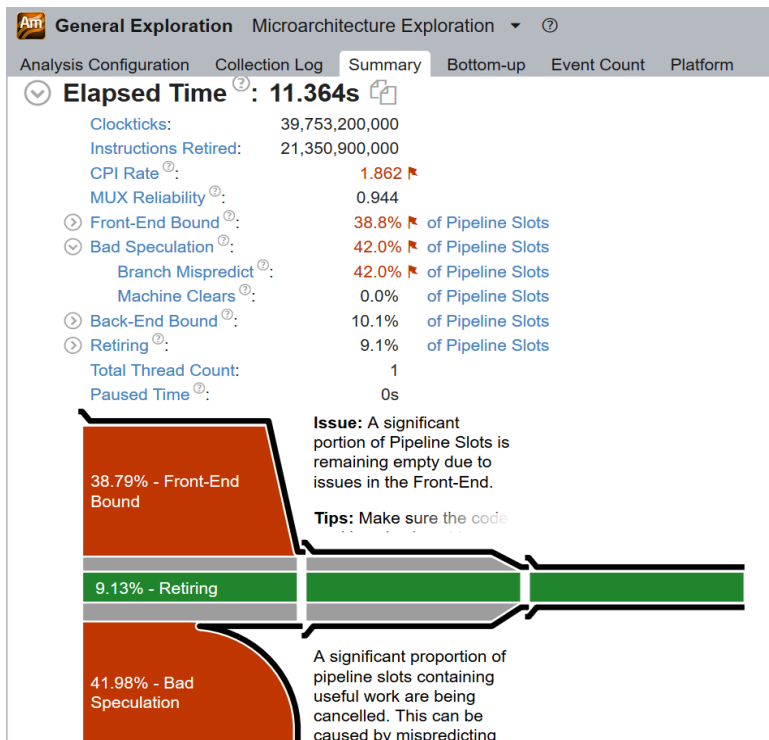
- Build the code with and without the 'sort'
- Use VTune Amplifier 'Microarchitecture Exploration' analysis for profiling

Microarchitecture Exploration

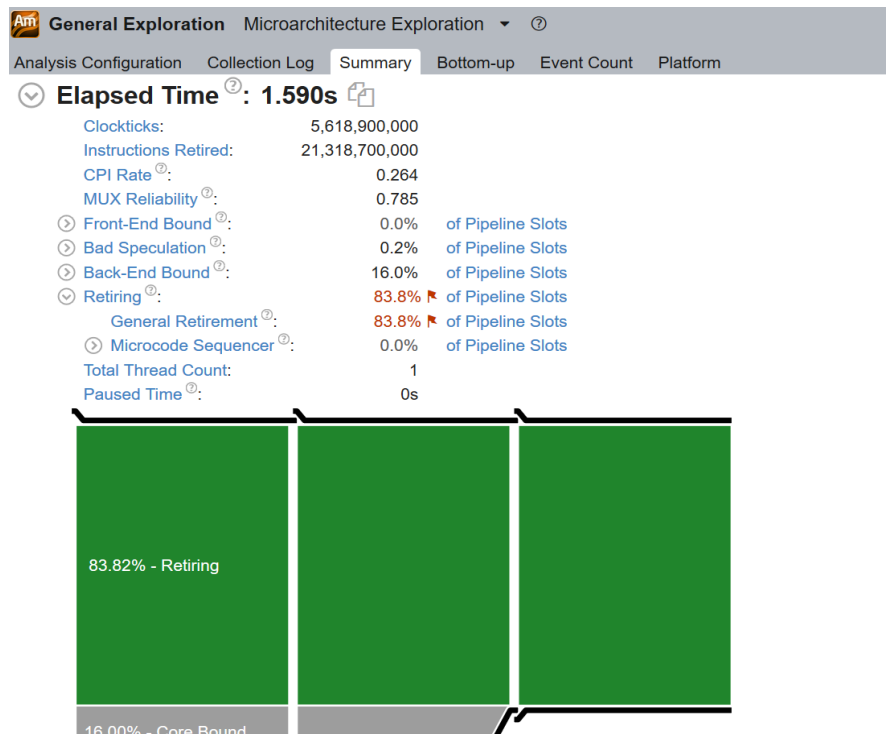
- ✓ Hardware event-based sampling
- ✓ Good starting point to triage hardware issues
- ✓ A complete list of events is collected for analyzing
- ✓ It calculates a set of predefined ratios used for the metrics and facilitates identifying hardware-level performance problems.



Microarchitecture Exploration - Summary



VS



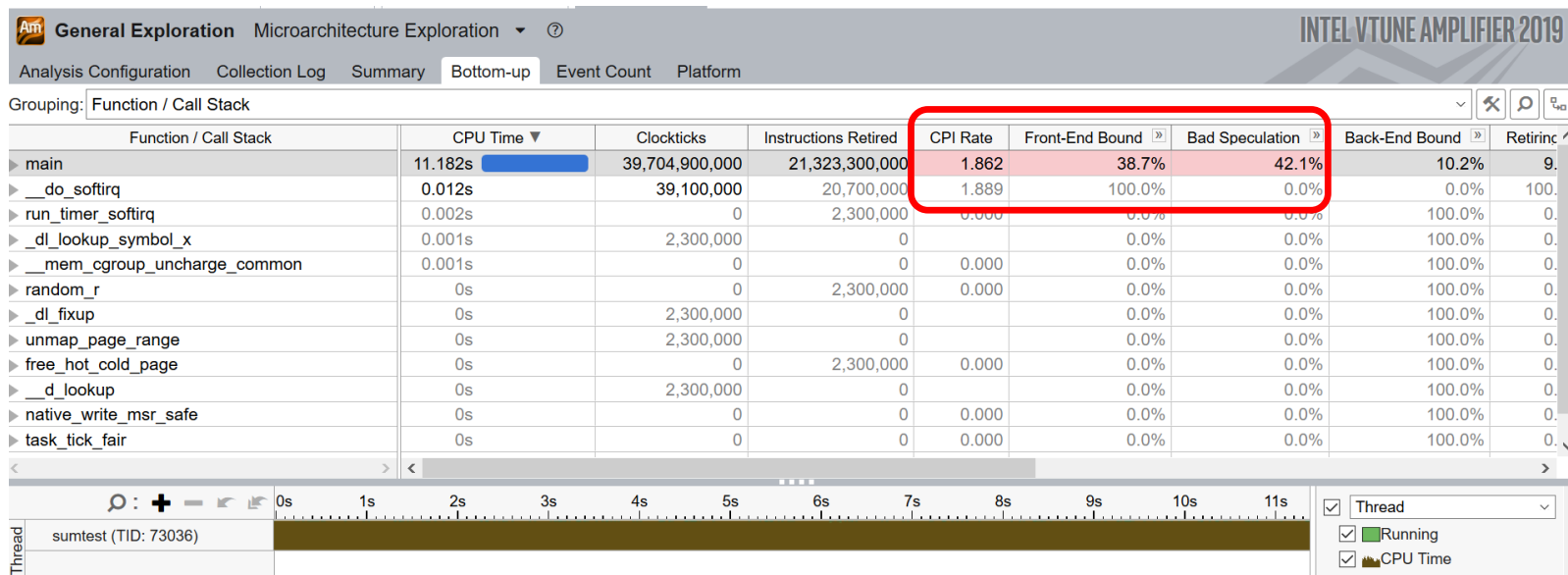
Optimization Notice

Copyright © 2018, Intel Corporation. All rights reserved.
*Other names and brands may be claimed as the property of others.

Intel Confidential



Microarchitecture Exploration for the case without 'sort'



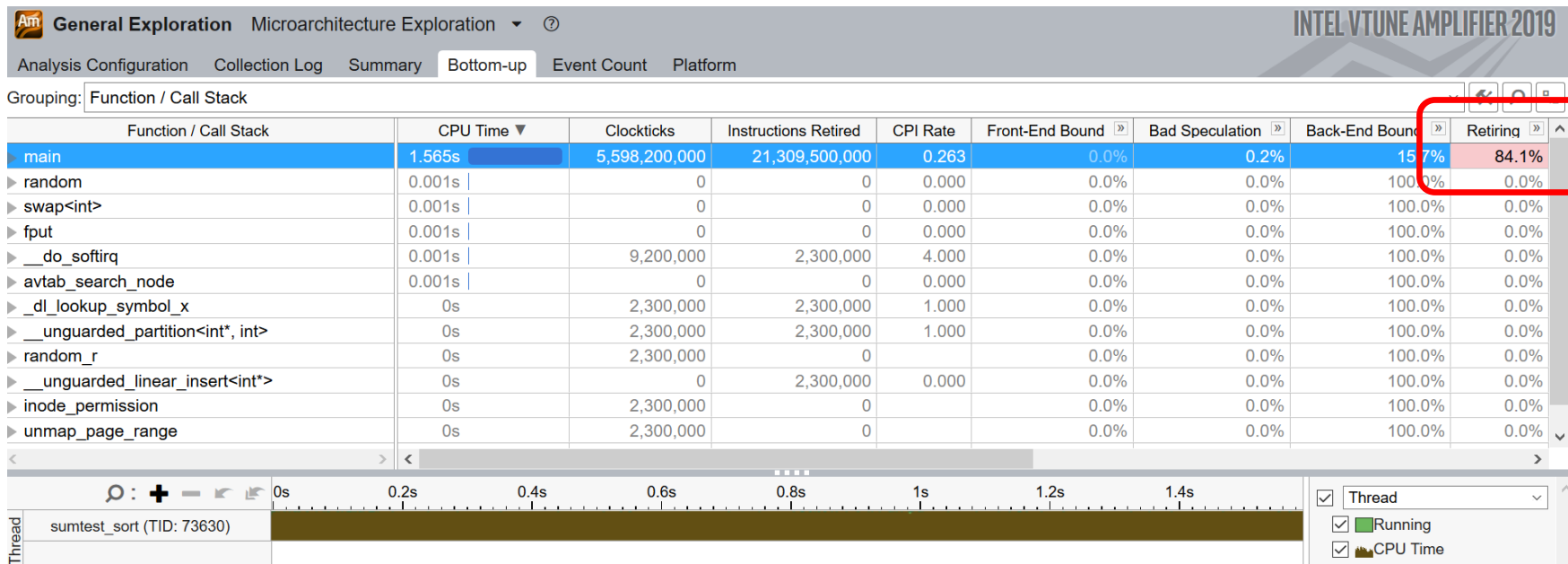
- Performance issue marked with pink
- VTune reported that this program has big 'Branch Mispredict' penalty

Which branch code cause the problem?

Go to source view to check which branch source code cause the problem

So. Li. ▲	Source	Clockticks	Instructions Retired	CPI Rate	Re.	Bad Speculation	
						Branch Mispredict	Machine Clears
6	{						
7	// Generate data						
8	const unsigned arraySize = 32768;						
9	int data[arraySize];						
10							
11	for (unsigned c = 0; c < arraySize; ++c)						
12	data[c] = std::rand() % 256;						
13							
14	// !!! With this, the next loop runs faster						
15	//std::sort(data, data + arraySize);						
16							
17	// Test						
18	clock_t start = clock();						
19	long long sum = 0;						
20							
21	for (unsigned i = 0; i < 100000; ++i)						
22	{						
23	// Primary loop						
24	for (unsigned c = 0; c < arraySize; ++c)	4,872,007,308	1,444,002,166	3.374	14.0%	42.1%	0.0%
25	{						
26	if (data[c] >= 128)	12,200,018,300	4,198,006,297	2.906	5.3%	40.2%	0.0%
27	sum += data[c];	21,602,032,403	15,664,023,4...	1.379	10.6%	0.0%	30.8%
28	}						
29	}						

Microarchitecture Exploration for the case with 'sort'



- 'sort' does help CPU to make the right decision on the branch prediction
- The 'Branch Mispredict' disappear, the performance (clockticks) improved significantly

Case Study: Microarchitecture exploration analysis

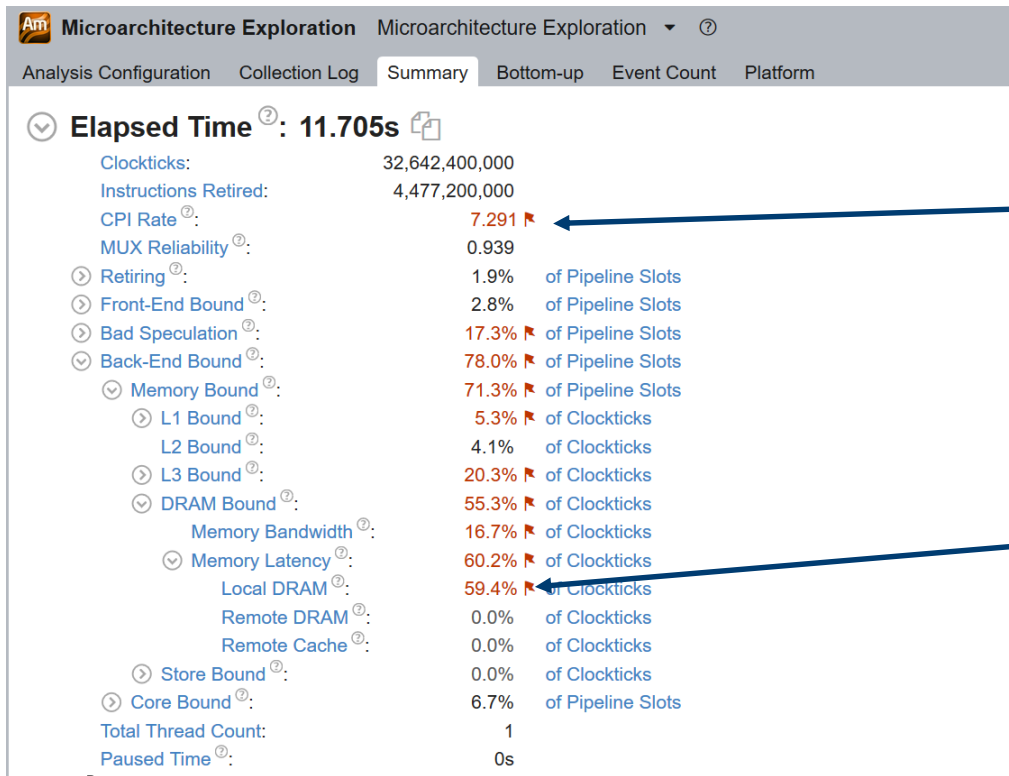
- A case from stackoverflow:
<http://stackoverflow.com/questions/7327994/prefetching-examples>
- 'array' is in memory
- The index is not a predictable value, hard for HW to prefetch array[mid]
- We may see high LLC cache miss for this case
- Run the customized analysis with Microarchitecture exploration for application "binary_search" and investigate

```
int binarySearch(int *array, int number_of_elements, int key) {  
    int low = 0, high = number_of_elements-1, mid;  
    while(low <= high) {  
        mid = (low + high)/2;  
  
        if(array[mid] < key)  
            low = mid + 1;  
        else if(array[mid] == key)  
            return mid;  
        else if(array[mid] > key)  
            high = mid-1;  
    }  
    return -1;  
}  
  
int main() {  
    int SIZE = 1024*1024*512;  
    int *array = malloc(SIZE*sizeof(int));  
    for (int i=0;i<SIZE;i++){  
        array[i] = i;  
    }  
    int NUM_LOOKUPS = 1024*1024*8;  
    srand(time(NULL));  
    int *lookups = malloc(NUM_LOOKUPS * sizeof(int));  
    for (int i=0;i<NUM_LOOKUPS;i++){  
        lookups[i] = rand() % SIZE;  
    }  
    for (int i=0;i<NUM_LOOKUPS;i++){  
        int result = binarySearch(array, SIZE, lookups[i]);  
    }  
    free(array);  
    free(lookups);  
}
```

What performance issue here?

Profiling with VTune and check the results

Microarchitecture exploration analysis Result

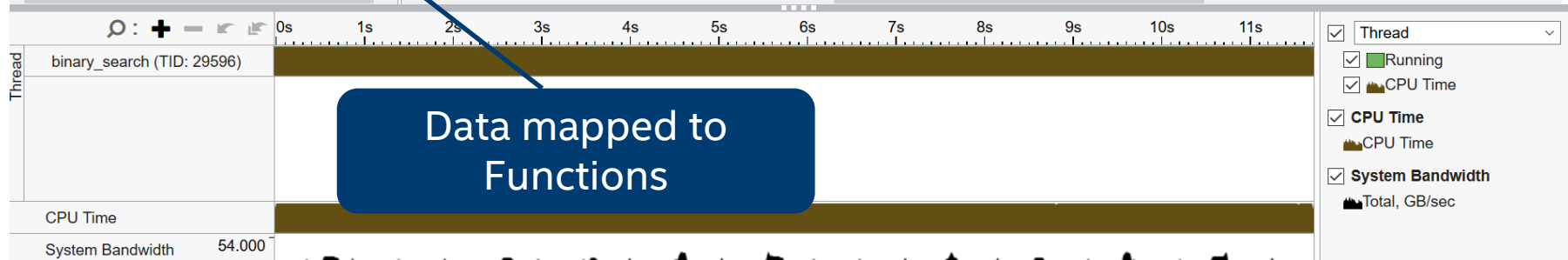


High CPI

Memory Latency issue
from Local DRAM

Grouping: Function / Call Stack

Function / Call Stack	Back-End Bound									
	Memory Bound								Store Bound	Core Bound
	L1 Bound	L2 Bound	L3 Bound	Memory Bandwidth	DRAM Bound					
					Memory Latency					
					Local DRAM	Remote DRAM	Remote Cache			
► binarySearch	5.5%	4.1%	21.4%	17.4%	61.2%	0.0%	0.0%	0.0%	6.7%	
► main	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	40.4%	
► change_protection_range	57.1%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%		
► _random_r	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	100.0%	
► _raw_spin_lock	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	100.0%	
► page_fault	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	
► clear_page_erns	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	100.0%	
► try_charge	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	100.0%	



Source	Locators	Back-End Bound		
		Memory Bound		
		DRAM Bound		
		Memory Bandwidth	Memory Latency	
			Local DRAM	Remote DRAM
14	endif			
15				
16	if(array[mid] < key)	15.9%	61.2%	
17	low = mid + 1;	0.2%	0.0%	
18	else if(array[mid] == key)	0.5%	0.0%	
19	return mid;			
20	else if(array[mid] > key)			
21	high = mid-1;	0.7%	0.0%	
22				
23	return -1;			
24				
25				
26				
27				
28				
29				
30				
31	LOOKUP			
32	return (0);			

Address	Source	Assembly	Clockticks	Instructions
0x4005fd		Block 2:		
0x4005fd	6	nopl %eax, (%rax)		
0x400600		Block 3:		
0x400600	17	leal 0x1(%rdx), %esi	2.4%	130,
0x400603	7	cmp %ecx, %esi	1.8%	252,
0x400605	7	jnle 0x400628 <Block 7>		
0x400607		Block 4:		
0x400607	8	leal (%rsi,%rcx,1), %edx	0.8%	33,
0x40060a	8	sar \$0x1, %edx	0.1%	6,
0x40060c	16	movsxd %edx, %r8	0.4%	12,
0x40060f	16	movl (%rbp,%r8,4), %r8d	0.6%	14,
0x400614	16	cmp %r8d, %r9d	84.7%	1,864,
0x400617	16	jnle 0x400600 <Block 3>	0.0%	
0x400619		Block 5:		
0x400619	18	jz 0x400628 <Block 7>	2.3%	161,
0x40061b		Block 6:		
0x40061b	21	sub \$0x1, %edx	1.9%	258,
0x40061e	21	cmp %r8d, %r9d	0.3%	33,
0x400621	21	cmovl %edx, %ecx	0.0%	4,
0x400624	7	cmp %ecx, %esi	0.1%	
0x400626	7	jle 0x400607 <Block 4>	0.0%	
0x400628		Block 7:		
0x400628	7	add \$0x4, %rax	0.4%	81,

Correlated Source and Assembly highlighted

Change the code with data prefetch

Add prefetch to
reduce the cache miss

Good performance
gain

```
int binarySearch(int *array, int number_of_elements, int key) {
    int low = 0, high = number_of_elements-1, mid;
    while(low <= high) {
        mid = (low + high)/2;
        #ifdef DO_PREFETCH
        // low path
        __builtin_prefetch (&array[(mid + 1 + high)/2], 0, 1);
        // high path
        __builtin_prefetch (&array[(low + mid - 1)/2], 0, 1);
        #endif

        if(array[mid] < key)
            low = mid + 1;
        else if(array[mid] == key)
            return mid;
        else if(array[mid] > key)
            high = mid-1;
    }
    return -1;
}

int main() {
    int SIZE = 1024*1024*512;
    int *array = malloc(SIZE*sizeof(int));
    for (int i=0; i<SIZE; i++){
        array[i] = i;
    }
    int NUM_LOOKUPS = 1024*1024*8;
    srand(time(NULL));
    int *lookups = malloc(NUM_LOOKUPS * sizeof(int));
    for (int i=0; i<NUM_LOOKUPS; i++){
        lookups[i] = rand() % SIZE;
    }
    for (int i=0; i<NUM_LOOKUPS; i++){
        int result = binarySearch(array, SIZE, lookups[i]);
    }
    free(array);
    free(lookups);
}
```

When I compile and run this example with DO_PREFETCH enabled, I see a 20% reduction in runtime:

```
$ gcc c-binarysearch.c -DDO_PREFETCH -o with-prefetch -std=c11 -O3
$ gcc c-binarysearch.c -o no-prefetch -std=c11 -O3
```

Case Study – Use vTune with DPDK IO API

- Using DPDK l3fwd to test the maximum forwarding performance. When CPU number is increased, the throughput doesn't increase.
- Testing with 1 CPU Core - 32Mpps for 64 bytes packet.
- Testing with 2 CPU cores - Still about 32M pps
- What is the bottleneck? CPU or NIC?

HOW



Input and Output



Analyze utilization of IO subsystems, CPU, and processor buses. [Learn more](#)

Select IO API type to profile:

☐ System Disk IO API

☐ SPDK IO API

☒ DPDK IO API

☐ Analyze PCIe bandwidth

☐ Analyze memory bandwidth

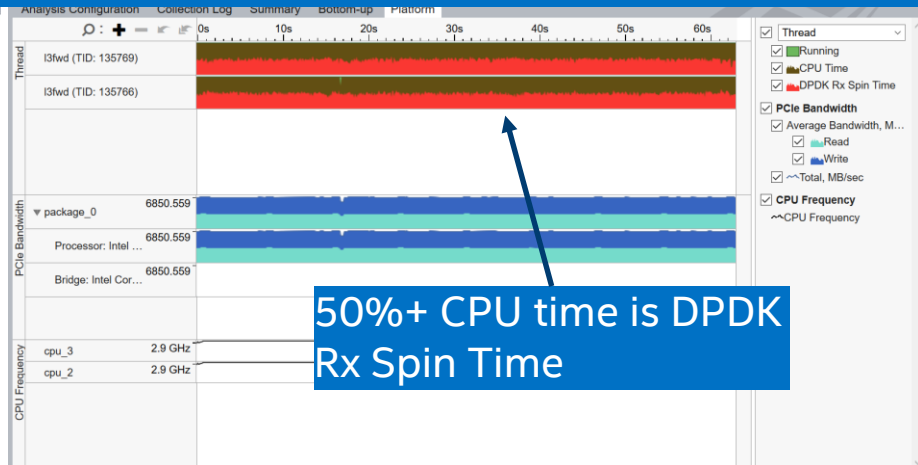
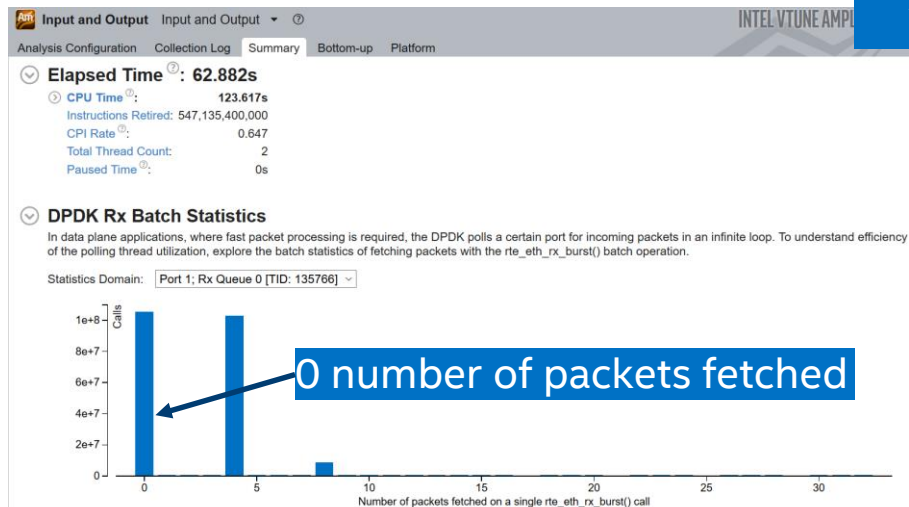
☒ Evaluate max DRAM bandwidth

► Details

Intel® VTune™ Amplifier 2019 – “Input and Output” analysis

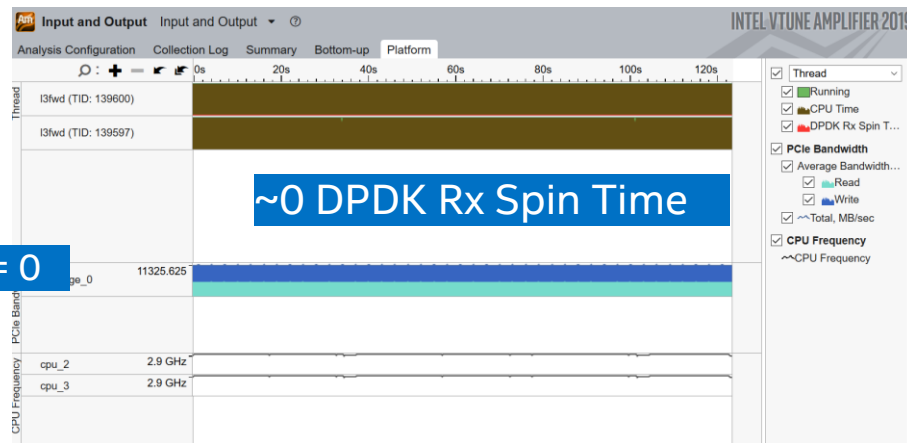
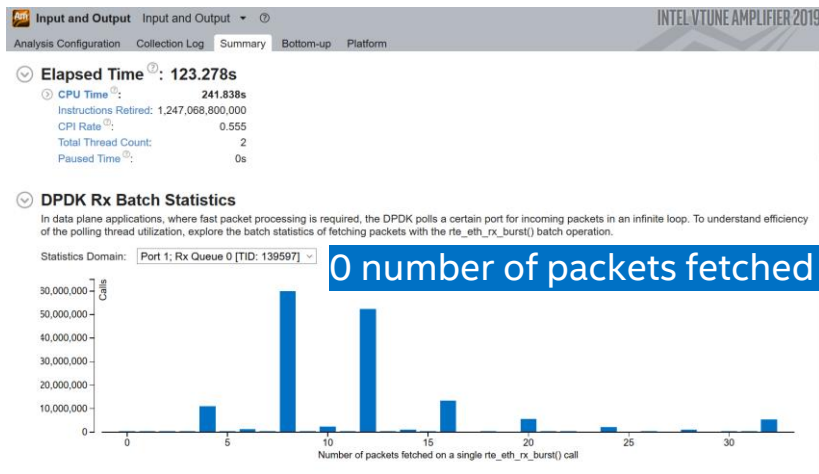
The “Input and Output” analysis from Intel® VTune™ Amplifier 2019 pinpoints a clear answer.

$$Rx\ Spin\ Time = \frac{num_rx_burst_calls_ret_0_pkts}{total_num_rx_burst_calls} \cdot 100\%$$



- CPU is not fully utilized for packets receiving. 50% CPU time is DPDK Rx Spin Time with 0 number of packets fetched → CPU is not the bottleneck!

Intel® VTune™ Amplifier 2019 – “Input and Output” analysis



- Add a new NIC and enable the packets receiving with two NICs.
- DPDK Rx Spin Time is almost 0. CPU always get x number of packets.
- The throughput performance increased as expected.

More Resources

Intel® VTune™ Amplifier – Performance Profiler

- [Product page](#) – overview, features, FAQs...
- [Training materials](#) – tech briefs, documentation, eval guides..
- [Reviews](#)
- [Support](#) – forums, secure support...

Webinars

Free in-depth presentations

- [Register](#)
- [View Archives](#)

Additional Analysis Tools

- [Intel® Inspector](#) – memory and thread checker/ debugger
- [Intel® Advisor](#) – vectorization optimization and thread protot
- [Intel® Trace Analyzer and Collector](#) - MPI Analyzer and Profil

[What's New?](#)

Purchase includes a year of updates. Check out the latest improvements.

Additional Development Products

- [Intel® Software Development Products](#)

Legal Disclaimer & Optimization Notice

Performance results are based on testing as of August 2017 to September 2018 and may not reflect all publicly available security updates. See configuration disclosure for details. No product can be absolutely secure.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit www.intel.com/benchmarks.

INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

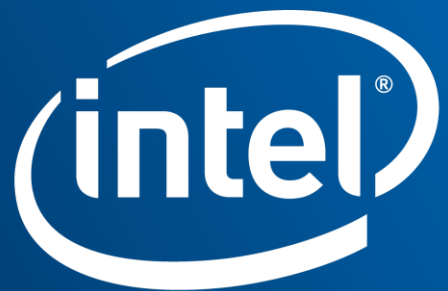
Copyright © 2018, Intel Corporation. All rights reserved. Intel, the Intel logo, Pentium, Xeon, Core, VTune, OpenVINO, Cilk, are trademarks of Intel Corporation or its subsidiaries in the U.S. and other countries.

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

BACKUP



Software