# Intel<sup>®</sup> VTune<sup>™</sup> Profiler

Vladimir Tsymbal, Technical Consulting Engineer, Intel



### Agenda

A short intro to the Intel® VTune Profiler

Collecting Hotspots with and w/o EBS

Analysis types overview

Microarchitecture Exploration

HPC Performance Characterization

Memory Analysis

### What is VTune Profiler, and what it is not?

- The Intel<sup>®</sup> VTune<sup>™</sup> Profiler (aka VTune) if one of the many dynamic performance profiling tools (WPA on Windows, Linux Perf, gprof) for native (C,C++, C#,Fortran, and to some extent jit or interpreted languages, Java, Python, OpenCL, Sycl/DPC++, Go)
  - o Highest awareness of Intel CPU, GPU, and FPGA microarchitecture
  - o Hight awareness of parallel runtimes: native/Posix threads, OpenMP, TBB, MPI, Intel's C/C++ Parallel Language Extensions

o Fully integrated and self-contained tool

- Nots (Buts):
  - o Not a static code analyzer, but...
  - o Not a code/system debugger, but...
  - o Not a system tracing tool like strace, but...
  - o Not a comprehensive advisor for code change, but...

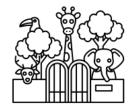
# VTune is all the following

- System wide profiling
- Application-level profiling
- Profiling hardware events
- Detecting CPU/GPU microarchitecture issues
- Heap/stack memory analysis
- System I/O analysis
- Statistical and Instrumented measurement
- Averaged measurements and precise tracing

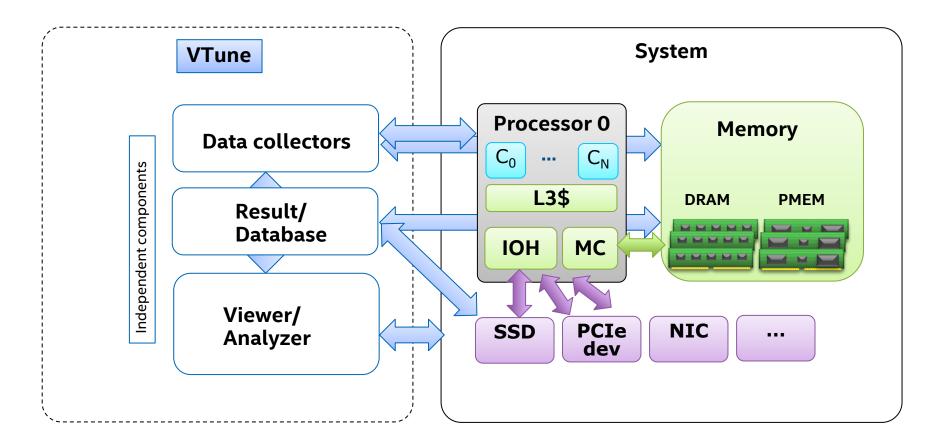


## Tools diversity in the VTune Zoo

- VTune Profiler
  - vtune, vtune-gui, vtune-backend
- APS Application Performance Snapshot tool aps, aps-report
- VPP VTune Platform Profiler
  - vpp-server, vpp-collect
- Utilities
  - vtune-self-checker, prepare-debugfs
- Standalone data collectors
  - sep, pin, emon

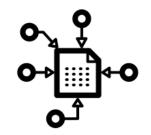


# Visible to a user VTune (standalone) structure



More complex structure with remote and server usage

### Data collectors



- Software data collector (Hotspots, Threading, etc.)
  - o No restrictions on virtual environments
  - o No driver
  - o Instrumentation based (overhead)
  - o No administrative rights required
- Hardware collector (Hotspots, Microarchitecture, Memory Analysis, I/O, Accelerators, etc.)
  - o VTune driver (sep) or driverless mode (Linux Perf), some admin's <u>help</u> is required
  - o For stack collection yet another driver (vtss) or limited Perf stacks
  - o All Hardware PMU events and Uncore events
  - o Hight resolution (down to 0.1 ms sampling interval)
  - o VMs need to virtualize PMU MSRs
- System collectors
  - o Windows (ETW), Linux (strace)

### Profiling result directory

- A self-contained config, logs, raw traces, and resolved results data base
- Created automatically in a project or current dir, or on path defined by a user
- Auto-numeration and pre-post-fixing for the analysis type
- CL option: -result-dir <path-to-my-result>
- Can be shared and viewed by other users
- Usually asked by Intel support engineers or developers (most system and config info is there)
- Can be accompanied with profiled app binaries. CL option: -archive -result-dir <path-to-my-result>
- Does not contain user source code by default (but can be added in a scope of opened results)
- Raw data traces obtained by collectors (sep, perf) can be imported/added to a result dir

# Viewer/Analyzer interface

# VT

#### GUI

The most comfortable and useful method of analysis

Good visual experience (complex data, scrolling, filtering, zooming in/out, etc.)

Can be launched locally (on an analyzed machine) or remotely

#### Command Line

- Good for collection or analysis procedures automation
- Can be launched locally on a machine without GUI or remotely

No memory overhead for weak machines (try GNOME 3 Desktop on a machine with 4GB RAM)

#### HTML GUI via Web Server

Super powerful method for machines management, remote computing and work collaboration

Requires VTune web server deployment (not a big deal, though)

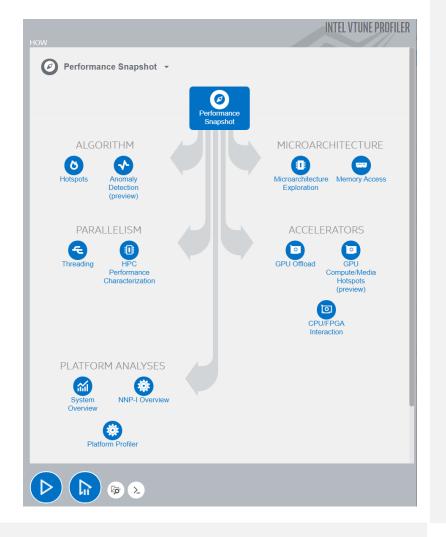
# Performance Profiling with Intel® VTune™ Profiler

# Intel® VTune™ Profiler

### A full set of performance analysis types

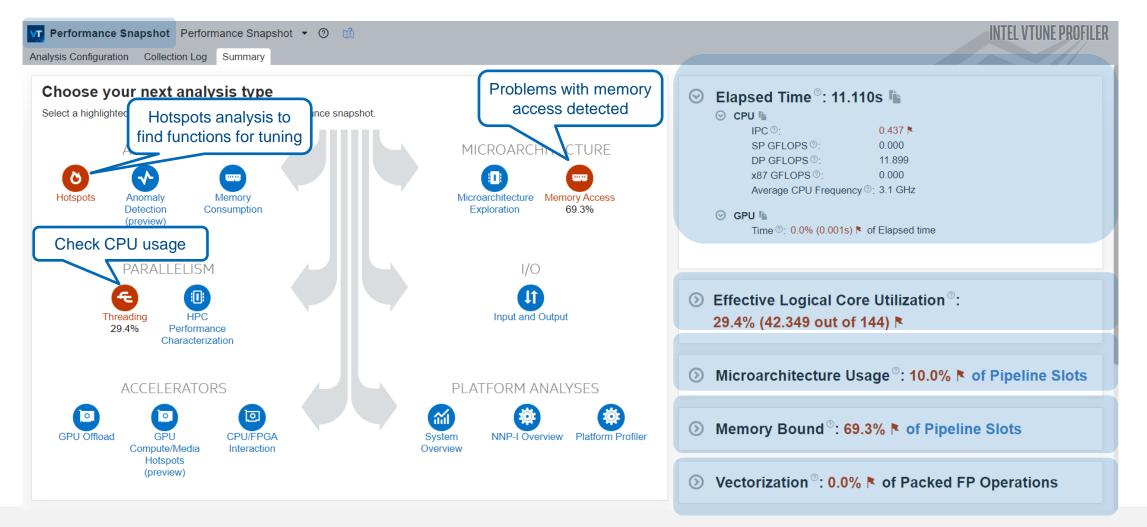
Which one to start with?

- Start with Performance Snapshot if you want to explore performance weakness of your app and get a recommendation how to continue
- Run Platform Profiler for analyzing long running apps on the whole platform (CPU, Memory, Disks, Ethernet, etc.)
- With Input-Output analysis make a snapshot of communication interfaces performance (PCIe, QPI, DRAM, SATA)
- The HPC Performance Characterization helps when your app is parallelized with OpenMP or MPI runtimes
- Dive deeper into a microarchitecture level inefficiencies of application execution with the Microarchitecture Exploration
- Investigate application memory bandwidth and latency problems with the Memory Access



# Performance Snapshot

### All application weaknesses in one snapshot



### **HPC Performance Characterization**

VT HPC Performance Characterization HPC Performance Characterization 🝷 🕐 👘

Analysis Configuration Collection Log Summary Bottom-up

#### Seffective Physical Core Utilization<sup>®</sup>: 54.7% (39.378 out of 72) ►

Effective Logical Core Utilization 2: 34.8% (50.169 out of 144) N

- Serial Time (outside parallel regions) 2: 0.191s (1.9%)
- Section Se

Estimated Ideal Time <sup>(2)</sup>: 7.810s (78.6%)

OpenMP Potential Gain <sup>®</sup>: 1.933s (19.5%) ▶

#### State of the second second

This section lists OpenMP regions with the highest potential for performance improvement. The Potential Gain metric shows the elapsed time that could be saved if the region was optimized to have no load imbalance assuming no runtime overhead.

| OpenMP Region                               | OpenMP Potential Gain ② | <b>(%)</b> ⑦ | OpenMP Region Time <a>&gt;</a> |
|---|-------------------------|--------------|--------------------------------|
| multiply1\$omp\$parallel:64@unknown:179:180 | 1.933s 🖡                | 19.5% 🏲      | 9.744s                         |

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

#### S Effective CPU Utilization Histogram

#### Vectorization<sup>®</sup>: 0.0% k of Packed FP Operations

- Instruction Mix:
  - ③ SP FLOPs <sup>③</sup>:
     DP FLOPs <sup>③</sup>:
     Packed <sup>③</sup>:
     Packed <sup>③</sup>:
     Constant <sup>△</sup>:
     Cons
    - Non-FP®: 77.7% of uOps

FP Arith/Mem Rd Instr. Ratio 2: 0.887

FP Arith/Mem Wr Instr. Ratio 2: 1.890

#### ○ Top Loops/Functions with FPU Usage by CPU Time

This section provides information for the most time consuming loops/functions with floating point operations.

| Function   | CPU Time 💿 | $\%$ of FP Ops $\oslash$ | FP Ops: Packed <a>?</a> | FP Ops: Scalar <a>?</a> | Vector Instruction Set <a>&gt;</a> | Loop Type 🔊 |
|--|------------|--------------------------|-------------------------|-------------------------|------------------------------------|-------------|
| [Loop at line 182 in multiply1\$omp\$parallel@179] | 493.126s   | 24.6%                    | 0.0%                    | 100.0% 🏲                |                                    | Body        |

#### Characterize OpenMP or MPI application

INTEL VTUNE PROFILER

### Parallelization: more details Now many cores were really used in a system

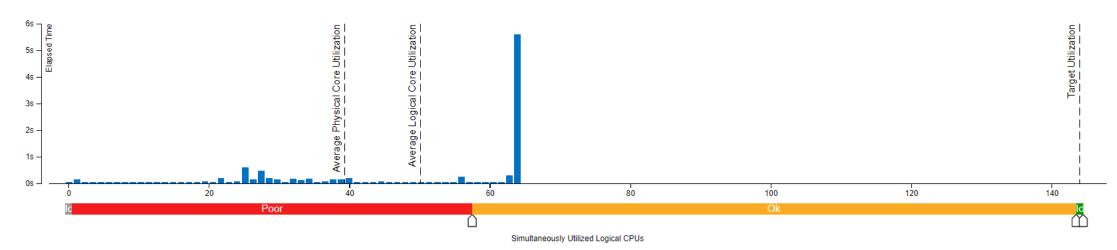
#### Effective Physical Core Utilization<sup>®</sup>: 54.7% (39.378 out of 72)

Effective Logical Core Utilization 144) 8

- Serial Time (outside parallel regions) <sup>(2)</sup>: 0.191s (1.9%)
- > Parallel Region Time : 9.744s (98.1%)
- Effective CPU Utilization Histogram

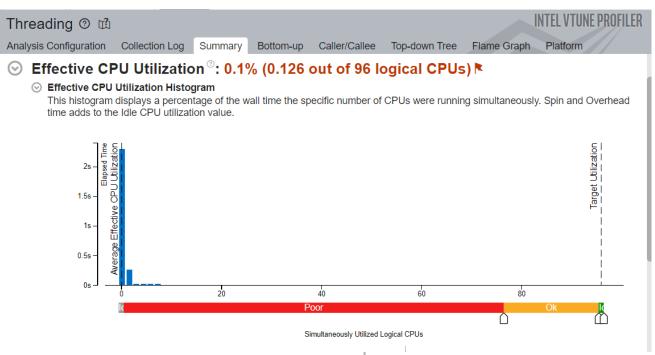
This histogram displays a percentage of the wall time the specific number of CPUs were running simultaneously. Spin and Overhead time adds to the Idle CPU utilization value.

#### Addr of buf1 = 0x7f51f38b2010 Offs of buf1 = 0x7f51f38b2180 Addr of buf2 = 0x7f51eb8b1010 Offs of buf2 = 0x7f51eb8b11c0 Addr of buf3 = 0x7f51e38b0010 Offs of buf3 = 0x7f51e38b0100 Addr of buf4 = 0x7f51db8af010 Offs of buf4 = 0x7f51db8af140 Threads #: 64 OpenMP threads Matrix size: 4096 Using multiply kernel: multiply1 Execution time = 7.114 seconds



intel.<sup>14</sup>

### Threading analysis, TBB based apps



#### ○ Wait Time with poor CPU Utilization <sup>②</sup>: 6.325s (100.0% of Wait Time) <sup>№</sup>

#### 📀 Top Waiting Objects 临

This section lists the objects that spent the most time waiting in your application. Objects can wait on specific calls, such as sleep() or I/O, or on contended synchronizations. A significant amount of Wait time associated with a synchronization object reflects high contention for that object and, thus, reduced parallelism.

| Sync Object                     | Wait Time with poor CPU Utilization ③ | (% from Object Wait Time) <a>?</a> | Wait Count ? |
|---------------------------------|---------------------------------------|------------------------------------|--------------|
| Futex 0xb2cbe538                | 6.294s                                | 100.0%                             | 913          |
| Futex 0xbcf65048                | 0.031s                                | 100.0%                             | 10           |
| Stream 0x59b268b6               | 0.000s                                | 100.0%                             | 1            |
| Stream /proc/cpuinfo 0x1ca6ac0a | 0.000s                                | 100.0%                             | 1            |
| Stream /proc/meminfo 0xc313ae18 | 0.000s                                | 100.0%                             | 1            |
| [Others]                        | 0.000s                                | 100.0%                             | 3            |

#### Spin and Overhead Time <sup>②</sup>: 188.376s (99.8% of CPU Time) ▶

#### O Top Functions with Spin or Overhead Time h

The section lists top functions in your application with the most spin and overhead time.

| Function   | Module                           | Spin and ③<br>Overhead<br>Time | (% from |
|--|----------------------------------|--------------------------------|---------|
| tbb::detail::d1::start_for <tbb::detail::d1::blocked_range<unsigned long="">, gemm<float><br/>(void, tbb::detail::d1::blocked_range<unsigned long="">*, tbb::detail::d1::blocked_range&lt;<br/>unsigned long&gt; const*, tbb::detail::d1::blocked_range<unsigned long=""> const*, unsigne<br/>d long, unsigned long, unsigned long)::{lambda(tbb::detail::d1::blocked_range<unsign<br>ed long&gt; const&amp;)#1}, tbb::detail::d1::alt::alt::alt::d1::blocked_range<unsign<br>ed long&gt; const&amp;)#1}, tbb::detail::d1::alt::alt::alt::alt::alt::alt::alt::al</unsign<br></unsign<br></unsigned></unsigned></float></tbb::detail::d1::blocked_range<unsigned> | 01_paralle<br>I_for_solut<br>ion | 183.663s 🏽                     | 97.3% 🖻 |
| tbb::detail::r1::market::create_one_job  | libtbb.so.1<br>2                 | 4.678s                         | 2.5%    |
| sched_yield  | libc.so.6                        | 0.010s                         | 0.0%    |

### What were the TBB internals that keep waits

| Threading ⑦ 础   |  |            |             |   |
|---|--|------------|-------------|---|
| Analysis Configuration Collection Log Summ                  | nary Bottom-up Caller/Calle                            | e Top-down | Tree Flame  | e Graph Platform  |
| Grouping: Sync Object / Function / Call Stack               |  |            |             | <ul> <li>✓ </li> <li></li></ul> |
| Sync Object / Function / Call Stack                         | Wait Time by Utilization ▼ <sup>≫</sup><br>I I P O I O | Wait Count | Object Type | Object Creation Module and Function   |
| ▶ Futex 0xb2cbe538  | 6.294s   | 913        | Futex       | libtbb.so.12!_INTERNAL795efa8b::tbb::detail::r1::futex_wait   |
| Futex 0xbcf65048  | 0.031s   | 10         | Futex       | libtbb.so.12!_INTERNAL795efa8b::tbb::detail::r1::futex_wait   |
| ▶ Stream 0x59b268b6   | 0.000s   | 1          | Stream      | libtbbmalloc.so.2!fgets.3   |
| Stream /proc/cpuinfo 0x1ca6ac0a                             | 0.000s   | 1          | Stream      | libtbb.so.12!tbb::detail::r1::ITT_DoUnsafeOneTimeInitialization   |
| Stream /proc/meminfo 0xc313ae18                             | 0.000s   | 1          | Stream      | libtbbmalloc.so.2!_Z9parseFileILi100ELi2EEvPKcRAT0K1  |
| Stream /proc/sys/vm/nr_hugepages 0x7fa52fc0                 | 0.000s   | 1          | Stream      | libtbbmalloc.so.2!_Z9parseFileILi100ELi1EEvPKcRAT0K1  |
| Stream /sys/kernel/mm/transparent_hugepage/e                | 0.000s   | 1          | Stream      | libtbbmalloc.so.2!_Z9parseFileILi100ELi1EEvPKcRAT0K1  |
| Stream /proc/self/maps 0x69913e68                           | 0.000s   | 1          | Stream      | libpthread.so.0!pthread_getattr_np  |
|   |  |            |             |   |
|   |  |            |             |   |
| 🔎 : 🕇 🗕 🖝 🖝 0s 0.2s   | 0.4s 0.6s 0.8s   |            | 1.2s 1.4s   | 1.6s 1.8s 2s 2.2s 2.4s  |
|   |  |            |             | · · · · · · · · · · · · · · · · · · ·   |
| 명 01_parallel_for (TID: 96921)<br>TBB Worker Thread (TID: 9 |  |            |             |   |
| TBB Worker Thread (TID: 9                                   |  |            |             | 1   |
| TBB Worker Thread (TID: 9                                   |  |            |             |   |
| TBB Worker Thread (TID: 9                                   |  | 1          | l.          | * <u>1</u>  |
| TBB Worker Thread (TID: 9                                   |  |            | Jr          |   |
| TBB Worker Thread (TID: 9                                   |  |            | , P         |   |
| TBB Worker Thread (TID: 9                                   |  |            | 1           | 4 4   |



| Hotspots Hots  | pots by CPU U  | Itilization 🔹 🕐   | ۲ů  |   |   |                               |                   |                                       |         |            |   |
|--|--|---|---|---|---|-------------------------------|-------------------|---------------------------------------|---------|------------|---|
| Analysis Configura   | ation Collection   | on Log Summary  | Bottom-up                                   | Caller/Callee                             | Top-down Tree   | Flame Graph                   | Platform          |                                       |         |            |   |
| User   | System   | Synchronizat  | 🗧 Overhead                                  | 🔲 Other                                   | 4   |                               | 5 🛎               | ₩ 10                                  | Search. | 6          | ٩   |
| miniFE::driver <doub<br>main<br/>libc_start_main<br/>start</doub<br> | k [<br>[Op [<br>[Stit [<br>_IN ]<br>k ]<br>. [Op [<br>. mini r<br>solve <minife::csi< th=""><th>miniFE::matvec_std<mi<br>kmp_invoke_microta<br/>OpenMP dispatcher]<br/>Stitch point frame]<br/>_INTERNAL_25<br/>_kmp_fork_call<br/>OpenMP fork]<br/>miniFE::matvec_std<mi<br>RMatrix<double, int,="" int<="" th=""><th>sk<br/>_src_kmp_runtime<br/>niFE::CSRMatrix&lt;(</th><th>_cpp_6e21bbbe::_<br/>double, int, int&gt;, mi</th><th>_kmp_itt_stack_caller<br/>niFE::Vector<double,< th=""><th>_create<br/>int, int&gt;&gt;::operat</th><th>k k<br/>[Op<br/>min</th><th>_km Functi<br/>_km Modu<br/>INTE Source</th><th></th><th>-3<br/>vait</th><th>r_cpp_b9f<br/>r_cpp_b9f8<br/>L_cpp_cfe967e4</th></double,<></th></double,></mi<br></mi<br></th></minife::csi<> | miniFE::matvec_std <mi<br>kmp_invoke_microta<br/>OpenMP dispatcher]<br/>Stitch point frame]<br/>_INTERNAL_25<br/>_kmp_fork_call<br/>OpenMP fork]<br/>miniFE::matvec_std<mi<br>RMatrix<double, int,="" int<="" th=""><th>sk<br/>_src_kmp_runtime<br/>niFE::CSRMatrix&lt;(</th><th>_cpp_6e21bbbe::_<br/>double, int, int&gt;, mi</th><th>_kmp_itt_stack_caller<br/>niFE::Vector<double,< th=""><th>_create<br/>int, int&gt;&gt;::operat</th><th>k k<br/>[Op<br/>min</th><th>_km Functi<br/>_km Modu<br/>INTE Source</th><th></th><th>-3<br/>vait</th><th>r_cpp_b9f<br/>r_cpp_b9f8<br/>L_cpp_cfe967e4</th></double,<></th></double,></mi<br></mi<br> | sk<br>_src_kmp_runtime<br>niFE::CSRMatrix<( | _cpp_6e21bbbe::_<br>double, int, int>, mi | _kmp_itt_stack_caller<br>niFE::Vector <double,< th=""><th>_create<br/>int, int&gt;&gt;::operat</th><th>k k<br/>[Op<br/>min</th><th>_km Functi<br/>_km Modu<br/>INTE Source</th><th></th><th>-3<br/>vait</th><th>r_cpp_b9f<br/>r_cpp_b9f8<br/>L_cpp_cfe967e4</th></double,<> | _create<br>int, int>>::operat | k k<br>[Op<br>min | _km Functi<br>_km Modu<br>INTE Source |         | -3<br>vait | r_cpp_b9f<br>r_cpp_b9f8<br>L_cpp_cfe967e4 |
| Total  |  | 2   |   |   |   |                               |                   |                                       |         |            |   |

CPU Time: 439.066s of 1703.590s (25.8%)



### Hotspots and Optimization Insights Get additional insights on execution efficiency

| Microarchitecture Exploration Hotspots by CPU Utilization - ⑦ 🕅  |                     |                             | INTEL VTUNE PROFILE  |
|--|---------------------|-----------------------------|--|
| alysis Configuration Collection Log Summary Bottom-up Caller/Callee Top-down Tree Pl   | latform             |                             |  |
| <ul> <li>Elapsed Time<sup>®</sup>: 11.238s <sup>1</sup></li> <li>CPU Time<sup>®</sup>: 467.981s<br/>Instructions Retired: 598,790,400,000<br/>CPI Rate<sup>®</sup>: 2.516 <sup>1</sup><br/>Total Thread Count: 576<br/>Paused Time<sup>®</sup>: 0s</li> <li>Top Hotspots<br/>This section lists the most active functions in your application. Optimizing these hotspot functions typic</li> </ul> | cally results in im | proving overall application | <ul> <li>Hotspots Insights         <ul> <li>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.</li> </ul> </li> <li>Explore Additional Insights         <ul> <li>Parallelism ③ : 28.1% (40.451 out of 144 logical CPUs) ►</li> <li>Use <ul> <li>Threading to explore more opportunities to increase parallelism in your application.</li> </ul> </li> <li>Microarchitecture Usage ③ : 9.2% ►         <ul> <li>Use <ul> <li>Microarchitecture Exploration to explore how efficiently your application runs on the used hardware.</li> </ul> </li> </ul></li></ul></li></ul> |
| performance.<br>Function   | Module              | CPU Time 💿                  | Vector Register Utilization ⊚ : 12.5% ►<br>Use Intel Advisor to learn more on vectorization efficiency of your   |
| multiply1\$omp\$parallel@179   | matrix.icc          | 438.366s                    | application.   |
|  |                     |                             |  |
| init_scratch_end   | vmlinux             | 13.733s                     |  |
|  |                     | 13.733s<br>10.906s          |  |
| init_scratch_end   |                     |                             |  |
| init_scratch_end<br>_INTERNALc1e8d79f::kmp_wait_template <kmp_flag_64<(bool)0, (bool)1="">, (bool)1, (bool)0, (bool)</kmp_flag_64<(bool)0,>  | 1> libiomp5.so      | 10.906s                     |  |

## Compiler Optimization Options

### Use vectorization switches

Linux\*, OS X\*: -x

- Might enable Intel processor specific optimizations
- Processor-check added to "main" routine: Application errors in case SIMD feature missing or non-Intel processor with appropriate/informative message

Example: -xCORE-AVX512

Special switch for Linux\*, OS X\*: -xHost

- Compiler checks SIMD features of current host processor (where built on) and makes use of latest SIMD feature available
- Code only executes on processors with same SIMD feature or later as on build host

gcc options:

-march=*cpu-type* 

'icelake-server'

### Help Compiler for Better Parallelization

### Remove dependencies between loop iterations for vectorization

- Improving unit stride (at fastest index walk)
- Loop interchange a known trick (leading to an excessive data transfers)
- Give compiler a hint like ivdev pragma when arrays are passed via pointers
- If possible, align data allocation to 64 Bytes (for AVX-512)

### Increase level of parallelization and work balance

- Utilize all CPU physical or logical Cores
- Adjust OMP compactness for OpenMP workloads
- Provide enough data size for workload balancing

### First Optimization Results

### Implemented loop interchange, data alignment and vectorization options

- Significantly improved Elapsed Time
- CPI Rate is only slightly better
- Still underutilizing Cores
- Cache bound memory accesses
- Vectorization with 256-bit registers, which is AVX only

| VT.     | Microarchitecture Explore   | tion HPC Performance Characterization 🝷 🕐 👔              |
|---------|---|--|
|         |   |  |
| Anal    | ysis Configuration Collectio  | n Log Summary Bottom-up                                  |
| $\odot$ | Elapsed Time <sup>®</sup> : 2                                       | .329s ኬ  |
|         | SP GFLOPS :   | 0.000  |
|         | DP GFLOPS :   | 59.645   |
|         | x87 GFLOPS ():  | 0.000  |
|         | CPI Rate <sup>(2)</sup> :   | 1.984 💌  |
|         | Average CPU Frequency   | / ℗: 3.0 GHz   |
|         | Total Thread Count:   | 579  |
|         |   |  |
| $\odot$ | Effective Physical  | Core Utilization <sup>©</sup> : 24.3% (17.472 out of 72) |
| 0       | -   | tilization ©: 21.3% (30.741 out of 144) N                |
|         | -   |  |
|         | Effective CPU Utilization   | n Histogram  |
|         |   |  |
| $\odot$ | Memory Bound <sup>®</sup> :   | 54.2% Nof Pipeline Slots                                 |
|         | Cache Bound :   | 54.6% 💌 of Clockticks                                    |
|         | DRAM Bound :  | 7.6% of Clockticks                                       |
|         | NUMA: % of Remote Ac  | cesses : 52.8%   |
|         |   |  |
| 0       | Vectorization®. 10  | 00.0% of Packed FP Operations                            |
| 0       |   | 0.0% of racked in operations                             |
|         | <ul> <li>Instruction Mix:</li> <li>SP FLOPs<sup>®</sup>:</li> </ul> | 0.0% of uOps   |
|         | O DP FLOPS <sup>(2)</sup> :   | 30.9% of uOps  |
|         | OP FLOPS OP Packed <sup>(2)</sup> :                                 | 100.0% from DP FP  |
|         | Packed ♥.<br>128-bit ℗:   | 0.0% from DP FP  |
|         | 256-bit <sup>(</sup> ):   | 100.0% F from DP FP                                      |
|         | 250-bit ©:  | 0.0% from DP FP  |
|         | Scalar :<br>Scalar :  | 0.0% from DP FP  |
|         | x87 ELOPs <sup></sup>   | 0.0% of uOps   |
|         | Non-FP <sup>®</sup>   | 69.1% of uOps  |
|         | FP Arith/Mem Wr Instr   |  |
|         | i r Anun/meni wi Ilisu. r   | allo . 1.303   |

### **Compiler Vectorization**

### Why not AVX-512?

| Address 🔺 | Source Line | Assembly                                  | 성 CPU Time 🖉 |
|-----------|-------------|---|--------------|
| 0x402770  |             | Block 16:                                 |              |
| 0x402770  | 66          | vmovupdy (%r12,%r10,8), %ymm2             | 40.037s      |
| 0x402776  | 66          | vfmadd213pdy (%r15,%r10,8), %ymm1, %ymm2  | 8.752s 📒     |
| 0x40277c  | 66          | vmovupdy %ymm2, (%r15,%r10,8)             | 6.690s 📒     |
| 0x402782  | 65          | add \$0x4, %r10                           | 0.025s       |
| 0x402786  | 65          | cmp %rax, %r10                            | 0.003s       |
| 0x402789  | 65          | j <u>b 0x402770 <block 16=""></block></u> | 7.972s 📕     |

### Check Compiler's Opt report

-qopt-report=2-qopt-report-phase=vec

LOOP BEGIN at ../src/multiply.c(65,4) remark #15300: LOOP WAS VECTORIZED remark #26013: Compiler has chosen to target XMM/YMM vector. Try using -qopt-zmm-usage=high to override LOOP END

### Force Compiler for ZMM vector?

-qopt-zmm-usage=high

### Check Opt report again

LOOP BEGIN at ../src/multiply.c(65,4) remark #15300: LOOP WAS VECTORIZED LOOP END

### Collect VTune profile and compare

| Address 🔺 | Source Line | Assembly                                  | 👍 CPU Time 🖹 |
|-----------|-------------|---|--------------|
| 0x402871  |             | Block 16:                                 |              |
| 0x402871  | 66          | vmovupsz (%r12,%rbx,8), %zmm4             | 63.988s      |
| 0x402878  | 66          | vfmadd213pdz (%r9,%rbx,8), %zmm3, %zmm4   | 23.717s 💼    |
| 0x40287f  | 66          | vmovupdz %zmm4, (%r9,%rbx,8)              | 3.327s       |
| 0x402886  | 65          | add \$0x8, %rbx                           | 0.026s       |
| 0x40288a  | 65          | cmp %rax, %rbx                            | 0.004s       |
| 0x40288d  | 65          | j <u>b 0x402871 <block 16=""></block></u> | 3.583s       |

### Why Performance Results are not much better?

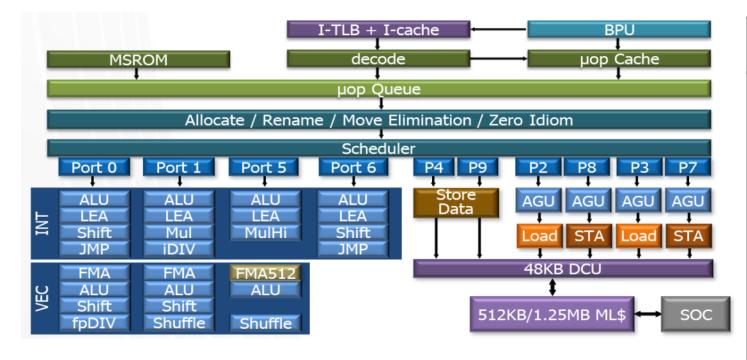
zmm registers

#### xmm/ymm registers

#### VT Microarchitecture Exploration HPC Performance Characterization • ③ 🗤 Microarchitecture Exploration HPC Performance Characterization 🝷 💿 📆 **111** Analysis Configuration Collection Log Summary Bottom-up Analysis Configuration Collection Log Summary Bottom-up Elapsed Time <sup>©</sup>: 2.329s 🐚 😔 Elapsed Time<sup>®</sup>: 1.976s 0.000 SP GELOPS 2 SP GFLOPS 2: 0 0 000 68 910 DP GFLOPS 2: 59.645 DP GFLOPS 2: x87 GFLOPS : 0.000 x87 GELOPS <sup>②·</sup> 0.000 CPI Rate 2: 1.984 🎙 CPI Rate 2 4.116 🖻 Average CPU Frequency 2: 3.0 GHz Average CPU Frequency 2: 2.7 GHz 579 Total Thread Count: 645 Total Thread Count: Substitution Section Secti Effective Physical Core Utilization 36.3% (26.131 out of 72) $\odot$ Effective Logical Core Utilization 2: 31.7% (45.618 out of 144) Effective Logical Core Utilization 2: 21.3% (30.741 out of 144) N Effective CPU Utilization Histogram Effective CPU Utilization Histogram Memory Bound<sup>≫</sup>: 66.6% ► of Pipeline Slots Memory Bound<sup>®</sup>: 54.2% of Pipeline Slots $(\checkmark)$ 57.9% Nof Clockticks Cache Bound 2: 54.6% 64 of Clockticks Cache Bound 2: DRAM Bound 2: 14.3% Not Clockticks DRAM Bound <sup>(2)</sup> 7.6% of Clockticks NUMA: % of Remote Accesses 2: 52.8% NUMA: % of Remote Accesses 2: 70.9% NUMA: % Vectorization<sup>®</sup>: 100.0% of Packed FP Operations ✓ Vectorization<sup>®</sup>: 100.0% of Packed FP Operations Instruction Mix: Instruction Mix: SP FLOPs : SP FLOPs 2: 0.0% of uOps 0.0% of uOps OP FLOPs <sup>②</sup>: 25.9% of uOps OP FLOPs <sup></sup>𝔅: 30.9% of uOps ⊘ Packed <sup>②</sup>: 100.0% from DP FP Packed <sup>(2)</sup>: 100.0% from DP FP from DP FP 128-bit 2 0.0% from DP FP 128-bit 2: 0.0% 0.0% from DP FP 256-bit 🕐 100.0% **\*** from DP FP 256-bit 2: 512-bit 2: 100.0% from DP FP 512-bit 2: 0.0% from DP FP 0.0% from DP FP Scalar : 0.0% from DP FP Scalar : x87 FLOPs 2: 0.0% of uOps x87 FLOPs 🛛 : 0.0% of uOps Non-FP <sup>⑦·</sup> 74.1% of uOps Non-FP 2: 69.1% of uOps FP Arith/Mem Wr Instr. Ratio 2: 2.054 FP Arith/Mem Wr Instr. Ratio 2 1,989

# Top-Down Method for Performance Analysis

## Ice Lake Core Microarchitecture



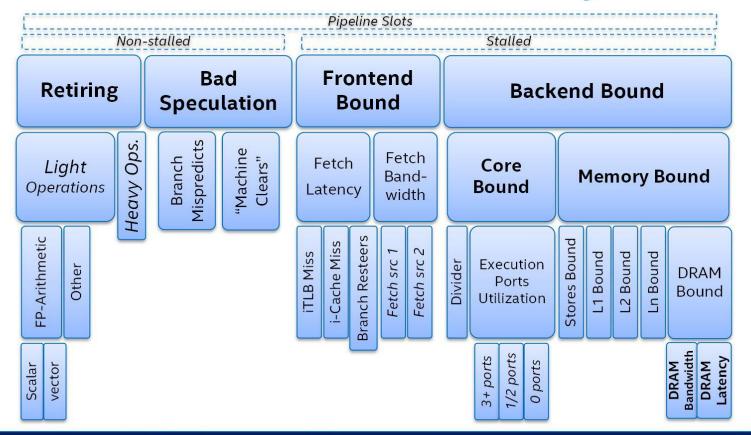
|                                  | Cascade Lake | Ice Lake<br>(per core)     |
|----------------------------------|--------------|----------------------------|
| Out-of-order Window              | 224          | 384                        |
| In-flight Loads + Stores         | 72 + 56      | 128 + 72                   |
| Scheduler Entries                | 97           | 160                        |
| Register Files –<br>Integer + FP | 180 + 168    | 280 +224                   |
| Allocation Queue                 | 64/thread    | 70/thread;<br>140/1 thread |
| L1D Cache (KB)                   | 32           | 48                         |
| L1D BW (B/Cyc) –<br>Load + Store | 128 + 64     | 128 + 64                   |
| L2 Unified TLB                   | 1.5K         | 2K                         |
| Mid-level Cache (MB)             | 1            | 1.25                       |

- Wider and deeper machine: wider allocation and execution resources + larger structures Improved Front-end: higher capacity and improved branch predictor
- Enhancements in TLBs, single thread execution, prefetching
- Server enhancements larger Mid-level Cache (L2) + second FMA

#### Increased instruction level parallelism

### Top-Down Method for Performance Analysis

### **One Bottlenecks Hierarchy\***



\*Reference paper: A. Yasin, "A Top-Down Method for Performance Analysis and Counters Architecture", ISPASS 2014

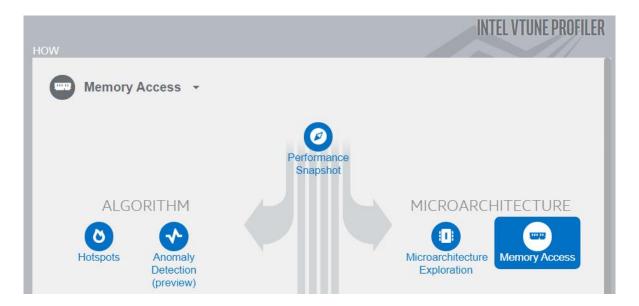
### Microarchitecture Exploration

| Microarchitecture Exploration Micro  | architecture Exploration 🝷 🧿 | INTEL VTUNE PROFILER   |
|--|------------------------------|--|
| Analysis Configuration Collection Log Sum  | mary Bottom-up Event Count   | Platform   |
| Solution State |                              | How good your code for the CPU   |
| Clockticks: 2  | 46,564,000,000               |  |
| Instructions Retired:  | 59,906,400,000               | The metric value is high.  |
| CPI Rate <sup>(2)</sup> :  | 4.116 🖻                      | This can indicate that the   |
| MUX Reliability <sup>(2)</sup> :   | 0.998                        | significant fraction of execution pipeline slots   |
| Retiring <sup>®</sup> :  | 9.2% of Pipeline Slots       | could be stalled due to  |
| Front-End Bound <sup>®</sup> :   | 5.5% of Pipeline Slots       |  |
| Bad Speculation <sup>®</sup> :   | 0.3% of Pipeline Slots       | 66.6% - Memory Bound stores. Use Memory Access   |
| Sack-End Bound <sup>®</sup> :  | 85.0% 🖻 of Pipeline Slots    | analysis to have the metric<br>breakdown by memory   |
| Memory Bound <sup>(2)</sup> :  | 66.6% 🖻 of Pipeline Slots    | hierarchy, memory  |
| O L1 Bound <sup>®</sup> :  | 8.2% of Clockticks           | bandwidth information,   |
| L2 Bound <sup>®</sup> :  | 2.4% of Clockticks           | Correlation by memory  |
| O L3 Bound <sup>®</sup> :  | 47.4% 🖻 of Clockticks        | 9.2% - Retiring  |
| Contested Accesses (?):  | 0.0% 🕅 of Clockticks         |  |
| Data Sharing <sup>®</sup> :  | 15.7% 🖻 of Clockticks        | 18.4% - Core Bound This metric represents how  |
| L3 Latency <sup>(2)</sup> :  | 58.9% 🕅 of Clockticks        | much Core non-memory   |
| SQ Full <sup>(2)</sup> :   | 17.1% 🖻 of Clockticks        | , uDino  |
| DRAM Bound <sup>(2)</sup> :  | 14.3% 💌 of Clockticks        | μPipe  |
| Store Bound <sup>(2)</sup> :   | 0.0% of Clockticks           | This diagram represents inefficiencies in CPU usage. Treat it as a pipe with an output flow<br>equal to the "pipe efficiency" ratio: (Actual Instructions Retired)/(Maximum Possible |
| Ore Bound <sup>(2)</sup> :   | 18.4% 🖻 of Pipeline Slots    | Instruction Retired). If there are pipeline stalls decreasing the pipe efficiency, the pipe  |
| Average CPU Frequency <sup>(2)</sup> :   | 2.7 GHz                      | shape gets more narrow.  |
| Total Thread Count:  | 645                          |  |
| Paused Time :  | 0s                           |  |

# Intel® VTune™ Profiler Memory Access Analysis

# VTune<sup>™</sup> Profiler Memory Access

Both problems: Memory Latency and Memory Bandwidth are estimated



- Latency problem estimation against Code and Memory Objects, and the Memory Level involved
- Bandwidth measurements are system wide (no code attribution, but Time stamps attribution)
- Like with any other type of analysis, investigate the Summary for your application and then focus on smaller range: functions or loops

### Platform Diagram

### A high-level view on data flow in a system

#### Platform Diagram



### Memory Issues Hierarchy

| Memory Access Memory Usage -              | 0 🕅                       |  |  |  |
|---|---------------------------|--|--|--|
| Analysis Configuration Collection Log Sum | mary Bottom-up Platform   |  |  |  |
| Selapsed Time <sup>®</sup> : 2.140s       |                           |  |  |  |
| CPU Time <sup>(2)</sup> :                 | 108.691s                  |  |  |  |
| Memory Bound <sup>③</sup> :               | 72.3% 🏲 of Pipeline Slots |  |  |  |
| L1 Bound <sup>®</sup> :                   | 6.6% of Clockticks        |  |  |  |
| L2 Bound <sup>(2)</sup> :                 | 2.7% of Clockticks        |  |  |  |
| L3 Bound <sup>®</sup> :                   | 50.0% 🕅 of Clockticks     |  |  |  |
| ⊘ DRAM Bound <sup>®</sup> :               | 17.8% 🕅 of Clockticks     |  |  |  |
| DRAM Bandwidth Bound <sup>(2)</sup> :     | 0.0% of Elapsed Time      |  |  |  |
| Store Bound <sup>(2)</sup> :              | 0.1% of Clockticks        |  |  |  |
| NUMA: % of Remote Accesses ?:             | 72.2% 🛌                   |  |  |  |
| UPI Utilization Bound <sup>®</sup> :      | 36.5% 🖻 of Elapsed Time   |  |  |  |
| Loads:                                    | 19,411,782,336            |  |  |  |
| Stores:                                   | 9,571,487,136             |  |  |  |
| LLC Miss Count <sup>®</sup> :             | 689,089,296               |  |  |  |
| Average Latency (cycles) .                | 280                       |  |  |  |
| Total Thread Count:                       | 641                       |  |  |  |
| Paused Time ?:                            | 0s                        |  |  |  |

#### Discover inefficient data access

- Finding which memory level is providing data with highest latency
- How much in % the problem is affecting performance
- If it's DRAM, then quickly identify if there is a NUMA problem (unbalanced access to remote DRAM)
- Go to the functions level (Bottom-Up Tab) for more details of responsible code
- Collect Memory Objects for more details on responsible data

# Memory Objects

### Instrument memory allocation with a single option

| HOW  |                   |           |  |   |
|--|-------------------|-----------|--|---|
| ت  | Memory /          | Access    |  | • |
| Measure a set of metrics to identify memory access related issues (for example, specific for NUMA architectures). This analysis type is based on the hardware event-based sampling collection. <u>Learn more</u> |                   |           |  |   |
| CPU samp   | ling interval, ms |           |  |   |
| ✓ Analyz   | e dynamic memory  | / objects |  |   |
| Minimal dynamic memory object size to track, in bytes  |                   |           |  |   |
| 1024   |                   |           |  |   |
| ✓ Evalua   | ate max DRAM ban  | dwidth    |  |   |
| Analyz   | e OpenMP regions  | 5         |  |   |

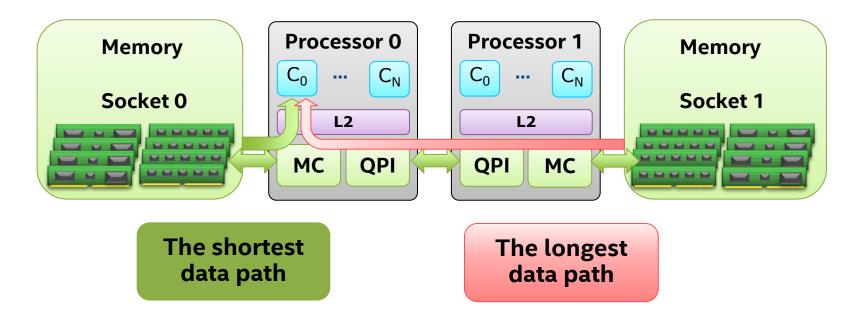
- Finding allocation place for memory objects that are "responsible" for CPU stalls
- Memory objects are identified by allocation source line and call stack
- Enables allocations instrumentation (off by default) and threshold for objects size (1024 by default)
- Don't use it when not needed (due to overhead)

### Memory Objects Decomposition

| Grouping: Memory Object / Function / Allocation Stack |                |                  |                  |                          |                   |
|---|----------------|------------------|------------------|--------------------------|-------------------|
| Memory Object / Function                              | Loads          | Stores           | LLC Miss Count » | Average Latency (cycles) |                   |
| [Unknown]   | 42,001,260     | 12,600,378       | 0                | 2                        |                   |
| matrix.c:126 ( 128 MB )                               | 11,200,336     | 18,565,956,962   | 2 0              | 0                        |                   |
| matrix.c:121 ( 128 MB )                               | 19,659,389,764 |                  | 8,998,429,846    | 805                      |                   |
| matrix.c:116 ( 128 MB )                               | 17,520,125,588 | c                | 70,004,900       | 46                       |                   |
| Load ope  | erations       | Store operations |                  | ost of LLC Miss          |                   |
|   |                |                  |                  |                          | Biggest latencies |

- Helps when you do not know data layout
- Helps when the same code line operates at many arrays
- "Memory Object" Grouping helps to find significant data objects first

# Dual Socket (-EP) System and NUMA Effects



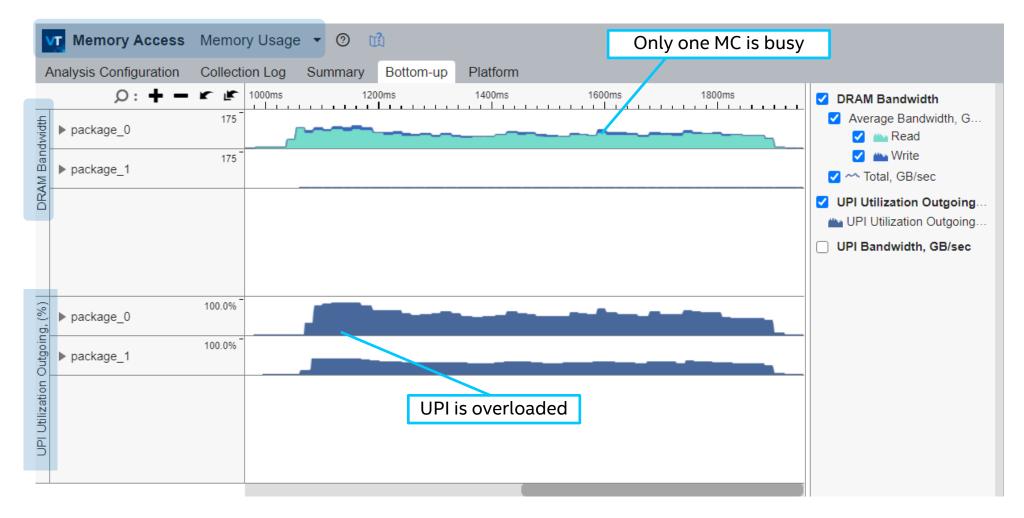
### Measuring Remote Memory Access

| Memory Access Memory Usage -             | · • •                        |  |  |
|--|------------------------------|--|--|
| Analysis Configuration Collection Log Su | ummary Bottom-up Platform    |  |  |
| Selapsed Time <sup>®</sup> : 2.140s      |                              |  |  |
| CPU Time <sup>®</sup> :                  | 108.691s                     |  |  |
| Memory Bound <sup>®</sup> :              | 72.3% 🖻 of Pipeline Slots    |  |  |
| L1 Bound <sup>(*)</sup> :                | 6.6% of Clockticks           |  |  |
| L2 Bound <sup>(*)</sup> :                | 2.7% of Clockticks           |  |  |
| L3 Bound <sup>(*)</sup> :                | 50.0% 🖻 of Clockticks        |  |  |
| ⊘ DRAM Bound <sup>®</sup> :              | 17.8% 🖻 of Clockticks        |  |  |
| DRAM Bandwidth Bound ②                   | 0.0% of Elapsed Time         |  |  |
| Store Bound <sup>®</sup> :               | 0.1% of Clockticks           |  |  |
| NUMA: % of Remote Accesses               | ⑦: 72.2% ►                   |  |  |
| UPI Utilization Bound <sup>®</sup> :     | 36.5% Sof Elapsed Time       |  |  |
| Loads:                                   | 19,411,782,336 Remote access |  |  |
| Stores:                                  | 9,571,487,136                |  |  |
| LLC Miss Count <sup>®</sup> :            | 689,089,296                  |  |  |
| Average Latency (cycles) ?:              | 280                          |  |  |
| Total Thread Count:                      | 641                          |  |  |
| Paused Time <sup>②</sup> :               | 0s                           |  |  |

- Remote DRAM access has biggest effect
- Remote memory access latency ~1.7x greater than local memory
- Local memory **bandwidth** can be up to ~2x greater than remote
- Remote Cache access bares its penalty as well

Refer to: Local and Remote Memory: Memory in a Linux/NUMA System

### Visualizing NUMA Problems



Refer to: Optimizing Applications for NUMA



- VTune Profiler is a comprehensive set that contains tools covering all aspects of platforms performance, from system wide and long app runs down to small execution kernels and microarchitecture specific on all intel platforms
- VTune Profiler helps to address algorithmic, multithreading, microarchitecture and memory issues
- VTune Profiler infrastructure provides flexible means for data collection, analysis, storage and team collaboration

# Quick References

#### Intel<sup>®</sup> VTune<sup>™</sup> Profiler – Performance Profiler

- Product page overview, features, FAQs...
- Training materials <u>Cookbooks</u>, <u>User Guide</u>, <u>Processor Tuning Guides</u>
- Support Forum
- Online Service Center Secure Priority Support
- What's New?

#### Additional Analysis Tools

- Intel<sup>®</sup> Advisor Design and optimize for efficient vectorization, threading, memory usage, and accelerator offload. Roofline and flow graph analysis.
- Intel<sup>®</sup> Inspector memory and thread checker/ debugger
- Intel<sup>®</sup> Trace Analyzer and Collector MPI Analyzer and Profiler

#### Additional Development Products

Intel<sup>®</sup> Software Development Products





### Notices & Disclaimers

Intel technologies may require enabled hardware, software or service activation. Learn more at intel.com or from the OEM or retailer.

Your costs and results may vary.

Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.

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. Microprocessors-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. <a href="https://software.intel.com/en-us/articles/optimization-notice">https://software.intel.com/en-us/articles/optimization-notice</a>

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. See backup for configuration details. For more complete information about performance and benchmark results, visit <u>www.intel.com/benchmarks</u>.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See configuration disclosure for details. No product or component can be absolutely secure.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

© Intel Corporation. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.

#