Intel® Memory Latency Checker v3.5

Documentation

Contents

1 Introduction

An important factor in determining application performance is the time required for the application to fetch data from the processor's cache hierarchy and from the memory subsystem. In a multi-socket system where Non-Uniform Memory Access (NUMA) is enabled, local memory latencies and cross-socket memory latencies will vary significantly. Besides latency, bandwidth (b/w) also plays a significant role in determining performance. So, measuring these latencies and b/w is important to establish a baseline for the system under test, and for performance analysis.

Intel® Memory Latency Checker (Intel® MLC) is a tool used to measure memory latencies and b/w, and how they change with increasing load on the system. It also provides several options for more fine-grained investigation where b/w and latencies from a specific set of cores to caches or memory can be measured.

2 Installation

Intel® MLC supports both Linux and Windows.

Linux

- Copy the mlc binary to any directory on your system
- Intel® MLC dynamically links to GNU C library (glibc/lpthread) and this library must be present on the system
- Root privileges are required to run this tool as the tool modifies the H/W prefetch control MSR to enable/disable prefetchers for latency and b/w measurements. See section 9.1.2 for instructions on running without root privileges
- MSR driver (not part of the install package) should be loaded. This can typically be done with 'modprobe msr' command if it is not already included.

Windows

- Copy mlc.exe and mlcdrv.sys driver to the same directory. The mlcdrv.sys driver is used to modify the h/w prefetcher settings

There are two sets of binaries (mlc and mlc_avx512). One is compiled with newer tool chain to support AVX512 instructions. The other binary supports SSE2 and AVX2 instructions. mlc_avx512 binary is a super set of mlc binary in that it supports SSE2/AVX2 as well. So, mlc_avx512 can be run on processors without support for AVX512 also. By default AVX512 instructions won't be used whether the processor supports it or not unless –Z argument is specified. We recommend you start with mlc_avx512 and if your system does not have the newer versions of glibc, then you can fall back to mlc binary.

3 H/W prefetcher control

It is challenging to accurately measure memory latencies on modern Intel processors as they have sophisticated h/w prefetchers. Intel® MLC automatically disables these prefetchers while measuring the latencies and restores them to their previous state on completion. The prefetcher control is exposed through a MSR (https://software.intel.com/en-us/articles/disclosure-of-hw-prefetcher-control-on-some-intel-processors) and MSR access requires root level permission. So, Intel® MLC needs to be run as 'root' on Linux. On Windows, we have provided a signed driver that is used for this MSR access. If Intel® MLC can't be run with root permissions, please see section 9.1.2 for more information.

4 What does the tool measure

When Intel® MLC is launched without any additional parameters, it automatically identifies the system topology and measures the following (a screen shot is shown for each):

1. A matrix of idle memory latencies for requests originating from each of the sockets and addressed to each of the available sockets

```
Measuring idle latencies (in ns)...
        Memory node
Socket       0        1
     0    67.5    125.2
     1   126.5     68.5
```

2. Peak injection memory b/w measured (with all accesses to local memory) for requests with varying amounts of reads and writes (each core generating requests as fast as possible)

```
Measuring Peak Memory Bandwidths for the system
Bandwidths are in MB/sec (1 MB/sec = 1,000,000 Bytes/sec)
Using traffic with the following read-write ratios
ALL Reads        :      76789.8
3:1 Reads-Writes :      70322.1
2:1 Reads-Writes :      69779.4
1:1 Reads-Writes :      69465.2
Stream-triad like:      66288.9
```

A

3. A matrix of memory b/w values for requests originating from each of the sockets and addressed to each of the available sockets

```
Measuring Memory Bandwidths between nodes within system
Bandwidths are in MB/sec (1 MB/sec = 1,000,000 Bytes/sec)
Using Read-only traffic type
        Memory node
 Socket      0        1
     0  38477.8 19373.1
     1  19510.3 38610.4
```

4. Latencies at different b/w points

```
Measuring Loaded Latencies for the system
Inject  Latency Bandwidth
Delay   (ns)    MB/sec
==========================
 00000  194.33    76881.6
 00002  193.96    76981.4
 00008  193.70    77464.8
 00015  193.23    77567.1
 00050  190.45    77383.8
 00100  154.80    76586.6
 00200  102.27    50427.6
 00300   93.00    35460.0
 00400   88.77    27386.1
 00500   85.00    22365.4
 00700   82.68    16459.8
 01000   78.56    11889.6
 01300   76.48     9462.3
 01700   75.17     7413.2
 02500   73.84     5392.5
```

5. Latencies between caches in the processor

```
Measuring cache-to-cache transfer latency (in ns)...
Local Socket L2->L2 HIT  latency        24.5
Local Socket L2->L2 HITM latency        28.9
Remote Socket LLC->LLC HITM latency (data address homed in writer socket)
                        Reader Numa Node
Writer Numa Node     0       1
            0        -   113.8
            1    113.9       -
Remote Socket LLC->LLC HITM latency (data address homed in reader socket)
                        Reader Numa Node
Writer Numa Node     0       1
            0        -    87.3
            1     87.9       -
```

Intel® MLC also provides command line arguments for fine grained control over latencies and b/w that are measured.

Here are some of the things that are possible with command line arguments:

- Measure latencies from a specific core for requests addressed to a specific node with memory.
- Measure cache latencies
- Measure b/w from a subset of the cores/sockets
- Measure b/w for different read/write ratios
- Measure latencies for random address patterns instead of sequential
- Change stride size for latency measurements

5 Impact of Transparent Huge Pages

Some Linux distributions support Transparent Huge Pages (THP) which provides advanced memory management support. This feature automatically combines small (4KB) pages into large pages (2MB) and may migrate memory into a single node. However, Intel® MLC relies on memory remaining local to ensure accurate local memory latency and b/w measurements. So, it is essential to disable this feature. THP can be disabled as shown below

*echo never > sys/kernel/mm/*transparent_hugepage/defrag*
*echo never > sys/kernel/mm/*transparent_hugepage/enabled*

6 Generating latency vs. b/w data

One of the main features of Intel® MLC is measuring how latency changes as b/w demand increases. To facilitate this, it creates several threads where the number of threads matches the number of logical CPUs minus 1. These threads are used to generate the load (henceforth, these threads will be referred to as either load-generation threads or bandwidth generation threads). The primary purpose of the load-generation threads is to generate as many memory references as possible. While the system is loaded like this, the remaining one CPU (that is not being used for load generation) runs a thread that is used to measure the latency. This thread, which typically runs on cpu#0, is known as the latency thread and issues dependent reads. Basically, this thread traverses an array of pointers where each pointer is pointing to the next one, thereby creating a dependency in reads. The average time taken for each of these reads provides the latency. Depending on the load generated by the load-generation threads, this latency will vary. Once every few seconds the load-generation threads automatically throttle the load generated by injecting delays, thus measuring the latency under various load conditions. By default, h/w prefetchers on the core running the latency thread are disabled as the latency thread does sequential accesses.

By default, we pin each load-generation thread to one logical cpu. For example, on a system with 10-cores with hyper-threading enabled, MLC will create 18 load-generation threads and reserving physical core 0 for running latency thread. Each load generation thread can be configured to generate varying degrees of reads and writes to the cache hierarchy. Each of these threads allocates a buffer for reads and a separate buffer for writes (there is no sharing of data between any of the threads). By appropriately sizing the buffers, one can ensure that the references are satisfied at any particular cache level or serviced by memory.

There are several options to control the number of load generation threads, the size of buffers allocated for each of them, where they allocate their memory, ratio of reads to writes and whether the accesses are sequential or random.

7 Cache-to-cache transfer latencies

MLC supports measuring cache-to-cache transfer latencies. The basic idea is to bring in lines into L1/L2/L3 and then transfer control to another thread (which is either running on another core on the same socket or a different socket). This thread will read the same data and this will force cache-to-cache transfers from the cache that already has these lines. We can measure both Hit (hitting clean lines) and HitM (hitting lines in modified state) latencies by manipulating the initial thread to either just read the data into clean state or modify the data and keep it in M state. MLC supports several parameters for finer control.

8 Command line parameters

Launching Intel® MLC without any parameters measures several things as stated earlier. However, with command line arguments, each of the following specific actions can be performed:

mlc --latency_matrix
    Prints a matrix of local and cross-socket memory latencies

mlc --bandwidth_matrix
    Prints a matrix of local and cross-socket memory b/w

mlc --peak_injection_bandwidth
    Prints peak memory b/w (core generates requests at fastest possible rate) for various read-write ratios with all local accesses

mlc --max_bandwidth
    Prints maximum memory b/w (by automatically varying load injection rates) for various read-write ratios with all local accesses

mlc --idle_latency
    Prints the idle memory latency of the platform

mlc --loaded_latency
    Prints the loaded memory latency of the platform

mlc --c2c_latency
    Prints the hit/hitm latency of the platform

With the addition of –e flag as shown below, Intel® MLC will not modify the h/w prefetchers for any of the measurements.

mlc --e
    Do not modify prefetcher settings

With the addition of –X flag, only 1 hyper-thread per core will be used for all bandwidth measurements

mlc -X

Use only 1 hyperthread per core for bandwidth measurements. Otherwise all threads in the core will be used for bandwidth measurements

More parameters can be passed to customize the actions performed as listed below:

mlc ***--idle_latency*** *[-bn] [-tn] [-xn] [-ln] [-r] [-L] [-cn] [-in] [-jn] [-Jn] [-pn] [-Dn] [-e]*
mlc ***--latency_matrix*** *[-bn] [-tn] [-xn] [-ln] [-r] [-L][-e] [-a][-X]*
mlc ***--bandwidth_matrix*** *[-bn] [-tn] [-ln] [-L] [-R] [-Wn] [-e] [-X] [-Y] [-Z]*
mlc ***--peak_injection_bandwidth*** *[-bn] [-tn] [-ln] [-L] [-mn] [-e] [-X] [-Y] [-Z]*
mlc ***--max_bandwidth*** *[-bn] [-tn] [-ln] [-L] [-mn] [-e] [-X] [-Y] [-Z]*
mlc ***--loaded_latency*** *[-bn] [-tn] [-ln] [-L] [-R] [-Wn] [-mn] [-r] [-g<filename>] [-T] [-dn] [-e] [-o<filename>] [-kn] [-jn] [-nn] [-U] [-X] [-Y] [-Z][-P] [-Q] [-Kn]*
mlc ***--c2c_latency*** *[-bn] [-cn] [-Cn] [-H] [-in] [-r] [-wn] [-xn]*

Detailed descriptions for each of the arguments are provided below:

-a Measure idle latencies from each of the available CPUs. This option is valid only in latency_matrix mode.

-b Select the size of the buffer (in KB) to be allocated by each CPU. Default is 200,000 KB for latency measurements and 100,000 KB for b/w measurements. **In case of b/w measurements with both read and write traffic, same size buffer will be allocated once for reads and once for writes thus requiring twice the amount of memory**. This option is valid in all modes. The buffer size can also be specified with a suffix of m or g (like –b2g or –b10m)

-c Pin the latency-measuring thread to a particular CPU. All memory accesses will be initiated from this CPU irrespective of where the memory that is being addressed is located. This option is valid in both --idle_latency and --c2c_latency modes.

-C Specify the window size in KB for cache-to-cache transfer latency (default is 2,000 KB)

-d Specify load injection delay. This option selects the number of cycles of delay that will be injected between bursts of memory accesses to throttle the b/w generated. When this option is not specified, once every few seconds (as specified by –t option) this value will be automatically changed and data gathered. However, you may want to determine latency for a particular b/w value. By trying different values for this parameter, b/w that is close to what is desired can be generated. When this option is specified, the program will measure latency for only one b/w. A value of 0 for –d may provide maximum throughput. The throughput generally drops as this value is increased. At very high delay values (like 30,000), the system can almost be considered as idle and the latency would be very close to idle latencies. This option is valid in loaded_latency mode only.

-D Specify a maximum value for the random numbers that will be used in generating accesses in latency thread. For example, if a value of 8192 is specified, random numbers in the range of 1-8192 will be used. The randomization is not done over the entire buffer (-b size) to minimize the TLB misses. Instead, the entire buffer is divided into multiple blocks (each block will have –D number of cache lines) and then accesses will be randomized within each block. Once all the lines in a block have been accessed, the same process will be repeated for the next block and so on. This option is valid in idle_latency mode only.

-e Do not modify the prefetcher settings. If this option is not specified, the tool will automatically change the prefetcher settings appropriately for various measurements. This option is valid in all modes.

-E read extra n KB lines in writer thread in c2c_latency (default=0). This is used to force the lines out of either L1 or L2 into lower level caches. This option is valid only for c2c_latency mode.

-g Specify the input file name with delays to inject for measuring latencies under different load levels. This file can specify many inject-delay values (one per line). If this option is not specified, a default list of values that are hard coded into the program is used. The default list has the following values {0, 2, 8, 15, 50, 100, 200, 300, 400, 500, 700, 1000, 1300, 1700, 2500, 3500, 5000, 9000, 20000}. Each value specifies the number of cycles the load generator thread will spin in a busy loop between a burst of transactions. This reduces the total b/w generated and enables gathering latencies under varied load conditions. This option is valid only in loaded_latency mode.

-G Specify the buffer size in kB for extra reads in reader thread in c2c_latency mode

-H Measure clean line hit latency in c2c_latency mode. The default setting measures Hit-modified latency

-i Select on which CPU the requested memory will reside. It identifies the logical CPU that will allocate the memory and initialize it. The thread that is allocating the memory will pin itself to this CPU, allocate the memory and do a first 'touch' of that memory. This option is valid in idle_latency and c2c_latency modes.

-j Allocate memory from the specified numa node. If this option is selected, -J or –i parameter should not be selected. This option is valid in idle_latency and loaded_latency modes.

-J Specify a directory in which files for mmap will be created. This option relies on using file system mounted on persistent memory. This option is valid only in idle_latency mode.

-k Specify a list of cpu numbers to be used for measurements. The cpu# can be a range separated by a hyphen, for example, "-k4-8" means logical cpu#s 4,5,6,7,8 are to be used. Comma separation is also supported, for example, "-k4-8,12-16,18" means cpus

4,5,6,7,8,12,13,14,15,16,18 are to be used. This option is valid only in loaded_latency mode.

-K   Specify finer control over whether all 64 bytes of a cache line are accessed during loads/stores. Typically, more instructions need to be executed to access all the bytes of a cache line. For example, to load a 64B line, you need to execute 4 SSE instructions where each instruction loads 16 bytes. With AVX2 instructions that load 32 bytes at a time, 2 instructions are executed to load the full cache line. However, to measure cache or memory b/w, there is no reason to load entire line as even partial line access would bring the entire line into cache. Loading the entire line measures the core execution b/w while partial loads (i.e loading just 16 bytes of the line) are sufficient to measure cache b/w. By default for buffer sizes < 50MB, this tool would automatically use partial loads/stores. If a value of 1 is specified for this option, only partial loads/stores would be used irrespective of the buffer size. However, if 0 is specified, full line would be accessed in all cases. This option is applicable only for 64 byte stride and 100% read, W2, W3 and W5 traffic types.

-l   Set the stride size in bytes. Default is 64 bytes. This option is valid in all modes except c2c_latency.

-L   Allocate the buffer using the hugetlbfs interface. In case of Windows o/s, large pages should be enabled appropriately. The system should have already reserved enough 2MB pages to satisfy the allocation request. This option is valid in all modes except c2c_latency.

-m   Specify the mask value (in hex) of CPUs to run the load generation threads. CPU 0 should be excluded from this mask as it is used to run the latency measuring thread. If Intel® Hyper-Threading Technology is enabled, the other CPU that is part of physical core 0 should also be omitted from this mask. The –m mask can include CPU 0 only if latency is not being measured by specifying –T option. For example, on a system with 8 logical processors, a mask –mfe will select all but CPU 0. Each bit set refers to the CPU number on which a throughput thread will be launched. If we want to run the load generation thread only on CPU 7 (counting from zero), then the mask would be –m80. This option is valid only in loaded_latency mode.

-n   Specify the maximum random number to be used in random access bandwidth generation. This option is valid only if specified along with –U option. For example, if a value of 8192 is specified, random numbers in the range of 1-8192 will be used. The randomization is not done over the entire buffer (-b size) to minimize the TLB misses. Instead, the entire buffer is divided into multiple blocks (each block will have –n number of cache lines) and then accesses will be randomized within each block. Once all the lines in a block have been accessed, the same process will be repeated for the next block and so on. This option is valid only in loaded_latency mode.

-o   Specify input file with options for per-thread controls during bandwidth measurements (supported only for loaded_latency mode). This provides fine grained control as to how each hardware thread should use memory during bandwidth measurements. For example, we may want a set of threads to be reading from DRAM and a set of threads to be writing to persistent memory. This can be specified in the input file. The control parameters specified by the user in the input file override the defaults assumed by the tool. Each line in the file has several fields separated by <space>. If any line starts with '#', then that line will be considered as a comment and ignored. *<cpu-range> <traffic-type> <seq/random> <buf size in KB> <dram/pmem> <node-id/pmem-folder-path> <per-thread-delay>.* The 1st field specifies the cpu-range. The next 5-6 fields are control parameters to be applied to the cpu(s) specified by the 1st field.

> *<cpu-range>:* The setting for the control parameters on each line can be for a single logical cpu# (hw thread) or a list of numbers separated by comma including a range specified by a dash between two numbers. If we need to apply the same setting for all 32 cores in a system, 0-31 can be used, instead of creating an entry for each of the cpu#s from 0 to 31. No blank is allowed. For example,
> 1-3,6,10-12 : logical cpu# 1 to 3, 6, and 10 to 12 all will have the same setting
> 6-12,20-22 : logical cpu# 6 to 12 and 20 to 22 all will have the same setting
>
> *<traffic-type>*: specifies the traffic type used by threads in <cpu_range>. Traffic types 'R' or 'Wx' (where x can take values defined in detail under –W option) can be specified.
>
> *<seq/random>*: specifies whether sequential or random traversal pattern is to be used by threads defined in <cpu_range>. Allowed values are 'seq' or 'rand'
>
> *<buf size in KB>* : specifies buffer size in KB to be allocated by each thread in <cpu_range>
>
> *<dram/pmem>*: specifies where each thread in <cpu_range> allocates memory (in regular dram or persistent memory). Allowed values are either 'dram' or 'pmem'
>
> *<node-id/pmem-folder-path>*: This field is optional when the previous field is selected as 'dram'. When this field is present, it can either be a number or a string. If the previous field is 'dram', then this field will be a number indicating the numa node# from where memory will be allocated. If the previous field is 'pmem', then this field specifies the path that points to a folder where file mapping to persistent memory is present. So for example, if in a 2S 32 cpu system, we want 1 socket to have random reads that go to local memory (numa node 0) and 1 socket to have 2 reads and 1 write traffic to numa node 1, the file would contain
>
> > *0-7,16-23 R rand 30000 dram 0*
> > *8-15,24-31 W6 seq 30000 dram 1*
>
> < *per-thread-delay* >: This field is optional. However, if this field is specified, the previous field < *node-id/pmem-folder-path* > should be present too. With this field we can specify load injection delays on a per-thread basis. Otherwise, option

–d in the command line would define the same load injection delays for all the threads

Typically, each load generation thread allocates memory from a single numa node. However, we can also configure a thread to load from two different address regions (either two numa nodes with dram or two persistent memory regions or a combination of dram and persistent memory). The format is as follows

<cpu-range> <traffic-type-2> <seq/random> <buf size in KB> <dram-1/pmem-1> <node-id/pmem-folder-path> <dram-2/pmem-2> <node-id/pmem-folder-path> <addr-mix-ratio>

<traffic-type-2>: Only a few traffic types are supported in this option and those can only take one of the following values

W21 : 100% reads (similar to –R)
W23 : 3 reads and 1 write (similar to –W3)
W27 : 2 reads and 1 non-temporal write (similar to –W7)

<dram-1/pmem-1> <node-id/pmem-folder-path>: Provide either dram and nodeid or pmem and the folder path for the 1st address stream

<dram-2/pmem-2> <node-id/pmem-folder-path>: Provide either dram and nodeid or pmem and the folder path for the 2nd address stream

< addr-mix-ratio >: This is a decimal number which specifies how many accesses would go to the 2nd address stream out of a total of 100 accesses. Currently, this can only take a value of 10 or 25. 10 means out of 100 addresses generated, 90 would target <dram1/pmem1> while only 10 would address <dram2/pmem2>. For a value of 25, out of 100 addresses generated, 75 would target dram1/pmem1 while only 25 would target dram2/pmem2. *<per-thread-delay>* is not supported in this two address mix option

The following is an example of each thread accessing data from two different address streams. Threads 0 to 2 allocate buffers from dram on numa node 2 and from persistent memory file in folder at /mnt/pmem1. The traffic type is 100% reads. The accesses are interleaved at a ratio of 3:1. That is 75 accesses would go to dram and remaining 25 accesses go to /mnt/pmem1. The 2nd line below selects a traffic type where 2 reads and 1 non-temporal write would be generated. The addresses are interleaved between dram and pmem at a ratio of 9:1.

0-2 W21 seq 30000 dram 2 pmem /mnt/pmem1 25
30-32 W27 seq 60000 dram 1 pmem /mnt/pmem2 10

-pn1,n2,n3,.. Specify a core number on each socket (in the form of comma separated CPU numbers) where a dummy thread is executed to keep that CPU 100% active. The dummy thread executes yield() system call in a loop. Since current generation processors support aggressive power management, sockets with all idle cores operate at a low frequency to save power. This results in higher latencies for snoop responses. So while measuring latencies, at least one CPU in each socket should be kept 100% busy. When this parameter is not specified, a dummy thread is automatically launched on the first cpu in each socket for latency measurements. This option is valid in idle_latency mode only.

-P Execute CLFLUSHOPT instruction after any stores to addresses in persistent memory region. If stores are not flushed immediately, they may be evicted from the last level cache in any order and that may not provide the most optimal b/w. When this option is specified, each store is followed by a clflushopt instruction to evict the line to persistent memory. This option is valid only in AVX512 mode. So, avx512 binary is required along with –Z option.

-Q Execute CLFLUSHOPT instruction after any stores to any addresses. This option is similar to the previous option (-P) but applies to all address range and not persistent memory range only.

-r Initialize the buffer with pseudo-random values so the access pattern to memory will be random for latency measurement. For random access in load-generation threads, use option –U. This option is valid in idle_latency and latency_matrix modes.

-R Select 100% read traffic. If this option is specified –W should not be used. This option is valid in bandwidth_matrix and loaded_latency modes.

-t Set time in seconds during which each measurement is captured. Default is 2 seconds. This option is valid in all modes except c2c_latency

-T Specify this flag if only b/w is desired without latency values. If this is selected, then the –m option can have a mask value covering all available CPUs including 0. Similarly, if –k is specified, it can include cpu 0 in the cpu range. This option is valid only in loaded_latency mode.

-U Specify this flag to enable random access for bandwidth generation (default: off, only sequential access). This means that the threads generating the traffic that is used for bandwidth measurements access memory with "randomness" – i.e. they do not sequentially stride through memory when this flag is specified. Use –n flag along with this to specify the level of randomness. This option is valid only in loaded_latency mode.

-v Specify this flag to print the verbose output: additional details about what MLC is doing behind the scenes. This is useful for debugging. This option is valid in all modes.

-w Specify the cpu id to pin hit/hitm writer thread. This option is valid only in c2c_latency mode.

-W Specify read to write ratio for the b/w generation thread in bandwidth_matrix and loaded_latency modes . –W*n* where *n* can take the following values (reads and writes are as observed on the memory controller): The last 3 values (21, 23 and 27) can only be specified in the *per-thread-config* file as part of –o option

- 2 - 2 reads and 1 write
- 3 - 3 reads and 1 write
- 5 - 1 read and 1 write
- 7 - 2 reads and 1 non-temporal write
- 8 - 1 read and 1 non-temporal write
- 10 - 2 reads and 1 non-temporal write (similar to stream triad)
- 21 – 2 address streams - 100% reads
- 23 – 2 address streams - 3 reads and 1 write
- 27 – 2 address streams - 2 reads and 1 non-temporal write

-x Set the number of iterations in millions. If this option is specified then –t should not be specified. A value of 0 (-x0) will perform only one iteration over the buffer allocated. This option is valid in idle_latency and latency_matrix modes.

-X Specify this flag to use only 1 hyper-thread per core for bandwidth measurements. When this flag is specified, MLC picks the first hyper-thread id out of all the hyperthreads of each core to run the bandwidth thread. The other hyperthread(s) of the cores will not be used to run anything. In many cases, this gives higher bandwidth when compared to running traffic on all the hyperthreads of each core. If this flag is specified as part of latency_matrix(with –a option), then only one thread from each of the cores will be used for latency measurements. This option is valid in latency_matrix, peak_injection_bandwidth, max_bandwidth, bandwidth _matrix and loaded_latency modes.

9 Example usages

9.1 Collecting all latencies and bandwidth data

9.1.1 Default invocation

mlc

When MLC is invoked without additional arguments, all the modes (latency and b/w matrix, peak b/w and loaded latencies) are automatically executed

9.1.2 Running MLC without root privileges

mlc -e

Since MSR accesses are privileged operations, users who don’t have root access may not be able to run Intel® MLC. In that case, –e flag can be used to prevent the tool from modifying the h/w prefetcher settings.

The above command will run all measurements without modifying the prefetchers.
-e flag can also be specified for any specific measurement as shown below

mlc --bandwidth_matrix –e
mlc --peak_injection_bandwidth –e

However, it is essential for the user to disable all the h/w prefetchers before starting the measurements through other means. Otherwise, by default, the tool would report really low latencies as only sequential strides are used for latency measurements. To get accurate latency measurements without changing the prefetcher settings, the following commands can be used by adding –r and –l128 parameters to force random accesses to beat the prefetchers that may be active.

mlc --latency_matrix –e –l128 –r
mlc --idle_latency –c2 –i2 –e –l128 –r

9.1.3 Using only one hyper-thread from each core

mlc -X

By default, MLC will use all the hyper-threads from each core for all b/w measurements. However, with higher core count processors, it is likely that the best b/w may be obtained by using only one hyper-thread from each core. –X option can also be used with other modes of operation as shown below

mlc --bandwidth_matrix –X
mlc --peak_injection_bandwidth –X
mlc --peak_max_bandwidth –X
mlc --loaded_latency –X

9.2 Measuring Idle latency

9.2.1 Measuring local memory latency from a specific CPU

mlc --idle_latency –c4 –i4

This command measures local memory latency from CPU 4. The –i parameter specifies a CPU that resides on the same socket. A dummy thread is launched on the 1st cpu on each of the sockets to keep at least one core in each socket active. This prevents the socket from going into low frequency and delaying snoop responses. By default, 64-byte sequential stride is used.

9.2.2 Measuring remote memory latency from a specific CPU

mlc --idle_latency –c0 –i8

The above command allocates a buffer with a default size of 200 Mbytes on CPU 8 and initializes the entire buffer. This ensures that all the memory for this buffer is allocated from memory that is present on the same socket as CPU 8. Then requests are generated by a thread running on CPU 0 to access this memory. This test will run for 2 seconds and the average latency measured will be reported. A dummy thread is automatically launched on the 1st cpu on each of the sockets to keep at least one core in each socket active.

9.2.3 Measuring latencies with different stride length

mlc --idle_latency –c4 –i4 –l256

This command measures local memory latency by using 256 byte stride instead of 64 byte stride.

9.2.4 Measuring cache hit latency

mlc --idle_latency –b3000 –c0 –t3

This command measures the L3 cache hit latency. Since the buffer allocated is only 3MB in size, repeated accesses to the same buffer ensures that all the lines in this buffer will reside in L3 cache (assuming the L3 size is more than 3MB). So, the latency measured would represent the cache hit latency.

9.2.5 Measuring latencies using fixed number of requests

mlc --idle_latency –x10

This command allocates a buffer with a default size of 200 MBytes by pinning to cpu# 1 (default) and executes 10 million references to memory and measures the average latency instead of running for a fixed amount of time. The special case for this (using only 1 iteration over the buffer) is –x0 and this can be useful for scenarios where we want the memory to be “accessed” only once (for example, while testing scenarios involving directory state – where state will change after the very first access).

9.2.6 Measuring idle latency to a specific NUMA node

mlc --idle_latency –c0 –j3

This command measures the idle latency from cpu# 0 to memory on numa node# 3. Note that numa node 3 may be a regular node (with CPU resources) or it may be a memory-only node with no CPU resources.

9.2.7 Measuring idle latency with random access

mlc --idle_latency –c0 –r –D8192

MLC disables h/w prefetchers and does sequential 64-byte accesses to measure latencies. However, if you are unable to run MLC with root privileges (as controlling h/w prefetchers require MSR write access), you can run MLC with –e option. But h/w prefetchers will lower the latencies reported due to sequential access. In that case, -r option can be used to force random accesses. To reduce TLB misses, random access over entire buffer size is not done. Instead, the buffer is divided into multiple blocks (each is sized to –D times 64 bytes) and all the lines in each block are first randomly accessed and after accessing all the lines in that block, the same is repeated for the remaining blocks. The default value for –D is 4096.

9.2.8 Measuring idle latency to persistent memory

mlc --idle_latency –c0 –J<path>

MLC can measure read latencies to persistent memory by providing a path name to the file system.

The first step is to expose the non-volatile memory (NVDIMM) to the OS. The BIOS describes the NVDIMM configuration to the OS via the ACPI-defined NVDIMM Firmware Interface Table (NFIT). Thus it is necessary to use a NFIT kernel, where the kernel is compiled with CONFIG_ACPI_NFIT enabled. Checking "ls /sys/firmware/acpi/tables/" can confirm the presence of an NFIT table. Upon detection of an NFIT table, the NVDIMM driver (pmem driver) is automatically loaded. For applications to directly access persistent memory utilizing a standard byte addressable load/store interface, the NVDIMM driver exposes the persistent memory through a persistent memory-aware file system. For example, let's say the persistent memory region is /dev/pmem0, it can be mounted as follows:

mkdir /mnt/pmem0
mkfs.ext4 /dev/pmem0
mount –o dax /dev/pmem0 ***/mnt/pmem0***

*mlc --idle_latency –c0 –J****/mnt/pmem0***

A file can now be created in /mnt/pmem0, mapped with mmap() and accessed with load/store references. The path /mnt/pmem0 would also be the path that we pass to MLC for measuring latency to that region of persistent memory. Note that multiple persistent memory pools can be configured in the BIOS using different sizes, from different DIMMs, with different interleaving options, etc. In such a case, multiple regions (/dev/pmem0, /dev/pmem1, etc) would show up when we use a NFIT kernel – and the procedure above can be used to create multiple mount points/files.

9.3 Measuring Latency Matrix

9.3.1 Default invocation

mlc --latency_matrix

The default invocation as specified above measures idle memory latency from each socket to every other socket in the system and reports the results in a matrix form. Note that the default invocation reports latencies to all the NUMA nodes in the system (NUMA-level reporting works only on Linux and on Windows, the reporting is only at the socket level). For example, in a system with 3 NUMA (memory) nodes and with two of them having CPU resources, the tool will report a matrix as follows:

```
Command line parameters: --latency_matrix

Using buffer size of 200.000MB
Measuring idle latencies (in ns)...
                Numa node
Numa node            0       1       2
       0          81.0   161.9   258.1
       1         161.5    81.0   255.3
```

9.3.2 Measuring latencies on all the cpus

mlc --latency_matrix –a

When –a option is specified, latencies will be measured from every core on the socket to memory on all the numa nodes.

9.4 Measuring Bandwidth Matrix

9.4.1 Default invocation

mlc --bandwidth_matrix

The default invocation as specified above measures the b/w available from each socket to every other socket in the system and reports the results in a matrix form. The traffic generated would be 100% reads in this example.

9.4.2 Measuring b/w matrix with different read/write ratios

mlc --bandwidth_matrix –W3

Instead of generating 100% reads as in the default case, -W3 will select 3 reads and 1 write to memory. The following are the possible options for –W*n* where *n* can take the following values (reads and writes are as observed on the memory controller):

- 2 - 2 reads and 1 write
- 3 - 3 reads and 1 write
- 5 - 1 read and 1 write
- 7 - 2 reads and 1 non-temporal write
- 8 - 1 read and 1 non-temporal write
- 10 - 2 reads and 1 non-temporal write (similar to stream triad)

For 100% reads, specify –R instead of –W option.

9.4.3 Using AVX 256-bit or 512-bit loads/stores

mlc --bandwidth_matrix –Y
mlc --bandwidth_matrix –Z (requires mlc_avx512 binary)

By default, MLC uses only 16 byte load/stores instructions to generate the b/w. However, if your processor supports 256-bit AVX load/store instructions, they you can append –Y option. Even better b/w may be obtained with –Z option if your processor supports AVX512 instructions. But you need to run mlc_avx512 binary for that.

9.5 Measuring Peak Injection Bandwidth

Peak injection bandwidth is measured by generating requests from the core at the fastest possible rate. Sometimes, this may not be the maximum b/w that is achievable due to other limitations. Use --max_bandwidth option if you want to measure maximum b/w

9.5.1 Default invocation

mlc --peak_injection_bandwidth

The default invocation as specified above measures total b/w using several different read/write ratios. A default buffer size of 100 MB (each for read and write) is allocated on each of the available CPUs for the measurement.

9.5.2 Using AVX 256-bit or 512-bit loads/stores

mlc --peak_injection_bandwidth –Y
mlc --peak_injection_bandwidth –Z (requires mlc_avx512 binary)

For best results, use –Y or –Z option. Z option requires mlc_avx512 binary

9.5.3 Measuring peak b/w for a subset of CPUs

mlc --peak_injection_bandwidth –mff

This command uses CPUs 0-7 and measures memory b/w available from those cores.

9.5.4 Measuring peak cache b/w

mlc --peak_injection_bandwidth –b100

This command measures the sum of L2 cache b/w available across all CPUs in the system. Here it is assumed that L1 cache size is < 100KB while L2 cache size is > 100KB. Then all the references will hit in L2 cache, thus providing L2 cache b/w aggregated across all the cores.

9.5.5 Measuring peak b/w with only one thread from each core

mlc --peak_injection_bandwidth –b100 –X

This command uses only one thread from each core (instead of all the available hyper threads on that core) to generate the b/w. On processors with 8 or more cores per socket, it is likely that best b/w may be achieved when only one hyper-thread from each core is utilized to generate the b/w.

9.6 Measuring Maximum Bandwidth

Typically, peak_injection_bandwidth may also be the maximum possible bandwidth. However, there may be some bandwidth drop at the highest possible injection rate. With *max_bandwidth* option, different injection rates are automatically tried to arrive at the maximum possible bandwidth. The parameters supported in this option are exactly the same as in --*peak_injection_bandwidth.* Please refer to previous section for more details.

9.7 *Measuring Loaded latencies*

9.7.1 Default invocation

mlc --loaded_latency

The above command line starts load generation threads on all the available CPUs except on 0 where the latency thread is started. If Intel Hyper-Threading Technology is enabled, then the other thread on core 0 is also avoided for load generation. The load injection delays are automatically changed every 2 seconds and b/w and latency measured at that level are reported. 100% read traffic is used for this measurement.

Sample output for the above command follows:

```
Inject  Latency Bandwidth
Delay   (ns)    MB/sec
==========================
 00000  196.54   76701.3
 00002  196.13   76784.2
 00008  196.14   77053.2
…….
 09000   71.74   2206.4
 20000   71.32   1489.7
```

Data in the 1st column provide load injection delay values (basically number of cycles) used. The 2nd column provides the latency values in nanoseconds for the measured b/w reported in the 3rd column. Data in the 1st column are for informational purposes only.

9.7.2 Measuring loaded latencies for different read/write ratios

mlc --loaded_latency –W2

This command is similar to the previous one except that loaded latencies are measured for 2:1 read/write traffic. All possible values for –W option are specified in section 8

9.7.3 Measuring each B/W data points for specified duration

mlc --loaded_latency –t10

This command executes each of the injection points for 10 seconds and outputs b/w and latencies measured over that interval. The default value for –t is 2 seconds.

9.7.4 Measuring total b/w without latency

mlc --loaded_latency –W2 –d0 –T

This command measures total b/w for 2:1 read/write traffic with an injection delay of 0 cycles. –T ensures that b/w generation thread will run on cpu#0 also instead of the latency thread. Since there is no injection delay, this is likely to provide the maximum b/w available

```
Inject  Latency Bandwidth
Delay   (ns)    MB/sec
==========================
 00000  0.00    69390.6
```

9.7.5 Measuring b/w available for a subset of CPUs

mlc --loaded_latency -W2 –d0 –T –mff

or

mlc --loaded_latency -W2 –d0 –T –k0-7

In the first illustration, the mask “ff” is used with –m to specify a bit mask of CPUs to be used. In this case, ff indicates eight ones, which corresponds to a one for each of the first eight CPUs. In other words, only CPUs 0-7 are being used to generate the b/w

An alternate means to do this is by using the –k flag instead of –m as shown above.

9.7.6 Measuring latency for a particular b/w

mlc --loaded_latency –d1000

B/W can be throttled by specifying a particular value for the –d option. In this case, 1000 cycles of delay is introduced between bursts of memory requests. Though there is no direct way of specifying or knowing the b/w that would be generated ahead of time, this option provides a simple and iterative way to arrive at the b/w of interest by trying different values of -d

```
Inject  Latency Bandwidth
Delay   (ns)    MB/sec
==========================
 1000   79.14   12363.3
```

9.7.7 Measuring latencies for specified load delay injection levels

mlc --loaded_latency –gdelay.txt

Default runs use a predefined set of load injection delays to generate latency vs. b/w values for different injection levels. However, the user can customize the injection values by providing an input file with those values (each injection value can be specified as a decimal number with one value per line in a text file, identifying the name of that input file through the –g option). For the data shown below, the input file delay.txt contained 3 lines with values 100, 800 and 4000

```
Inject  Latency Bandwidth
```

Delay (ns) MB/sec
==========================
00100 167.19 76958.8
00800 81.47 15132.2
04000 72.90 3824.4

9.7.8 Measuring latencies with random access

mlc --loaded_latency –r

By default, the latency thread is doing sequential 64-byte access. With this –r option, the latency thread does random accesses. This can reduce memory page hits and increase latency. The load generation threads always do sequential access only, unless –U and –n are specified (only supported for limited traffic types).

9.7.9 Measuring latencies with random access within specified window for randomness

mlc --loaded_latency –r –D8192

By default, the random accesses for the latency thread are within a 4096 cache line window (if –D is not specified). Specifying a value for –D can change the size of the window. Note that a very large window can have implications on TLB misses.

9.7.10 Measuring b/w for cache hierarchies

mlc --loaded_latency –b100 –T –K1

By selecting an appropriate value for the buffer size (-b option), we can measure b/w and latencies for any cache level. In the example above, a buffer size of 100 KB is specified. This buffer can fit within L2 cache and thus provide b/w and latencies for that level. All options that are applicable for measuring memory traffic (like read/write ratio, delay values) are also available irrespective of the buffer size. For best b/w measurements, -K1 option is specified wherein only partial loads/stores are executed (ie. only one 16 byte load/store per cache line). Using –K0 would force loads/stores to entire 64-byte line and likely reduce the b/w.

9.7.11 Measuring peak injection b/w with a mix of sequential and random *read-only* traffic

mlc --loaded_latency –T –d0 –o<perthreadfile>

We can achieve a mix of traffic by appropriately specifying which threads have sequential patterns and which threads have random patterns in *perthreadfile*. We use –T to make the measurement throughput-only (no latency thread) and we use –d0 to collect only the peak bandwidth measurement (no delay injection). Note that cpus not specified in *perthreadfile* would not be used to generate b/w. *Perthreadfile* might look something like this:

0-7 R rand 100000 dram

8-15 R seq 100000 dram

This makes h/w threads 0 to 7 use random access patterns, and h/w threads 8 to 15 use sequential access patterns. Since numa node# is not specified as a 6^{th} parameter in these lines, each thread will do local memory allocation

9.7.12 Measuring latency and b/w with a mix of local and remote traffic

mlc --loaded_latency –o<perthreadfile>

Through *perthreadfile*, each thread can be configured to get its memory from any of the numa nodes. The latency thread which runs on cpu#0 can also be controlled as to where the memory for latency thread will be allocated. A sample file is provided below. CPUs 0 to 3 allocate memory from numa node 1 and CPUs 4 to 7 allocate memory from numa node 0. CPUs 8 to 15 allocate memory from numa node 0.

0-3 R seq 200000 dram 1
4-7 R seq 300000 dram 0
8-15 R seq 100000 dram 0

Note that cpus not specified in *perthreadfile* would not be used to generate b/w. Cpu#0 is an exception and will always be used to measure latencies. If cpu#0 is configured in this file, then buffer size and memory location will be used as specified in the file (though traffic type will always be 100% read-only). If cpu#0 is not configured in this file, then by default, latency thread on cpu0 will allocate 200MB buffer from local memory. If b/w generation is preferred on cpu#0 as well, then –T option should be specified in which case no latency measurement will take place.

9.7.13 Measuring b/w with mix of dram and persistent memory

mlc --loaded_latency –T –d0 –o<perthreadfile>

We can achieve a mix of traffic by appropriately specifying which threads access DRAM and which threads access persistent memory in *perthreadfile*. We use –T to make the measurement throughput-only (no latency thread) and we use –d0 to collect only the peak injection bandwidth measurement (no delay injection). Assuming persistent memory is mounted at two mount points: /mnt/pmfs1 and /mnt/pmfs2, *perthreadfile* may look something like this:

0-3 R seq 100000 dram 0
4-7 R seq 100000 pmem /mnt/pmfs1
8-11 R seq 100000 dram 1
12-15 R seq 100000 pmem /mnt/pmfs2

This makes h/w threads 0 to 3 and 8 to 11 allocate memory from dram on node#0 and node#1 respectively. H/W threads 4 to 7 and 12 to 15 allocate memory from persistent memory mounted via mount points /mnt/pmfs1 and /mnt/pmfs2 respectively.

9.7.14 Measuring b/w with different injection delays

mlc --loaded_latency –T –o<perthreadfile>

By default, all load generation threads run with the same injection delays. However, if it is desired to have each thread run with different load injection delays, those delays can be added at the end of each line in the *perthreadfile* as shown below

0-3 R seq 100000 dram 0 ***2000***
4-7 R seq 100000 pmem /mnt/pmfs1 ***50***

Here threads 0 to 3 would operate with load injection delays of 2000 while threads 4 to 7 would operate with load injection delays of 50.

9.7.15 Measuring two traffic streams from each thread

mlc --loaded_latency –T –o<perthreadfile>

By default, each load generation thread accesses only one region of memory (either one NUMA node in dram or one persistent memory region). However, it may be desired that each thread be able to access two regions in interleaved fashion to better simulate real life usage models. We can configure *perthreadfile* to specify 2 regions of memory along with the ratio between accesses as the last field in each line

0-3 W21 seq 100000 dram 0 dram 1 ***25***
4-7 W21 seq 100000 dram 1 pmem /mnt/pmfs1 ***10***

Here threads 0 to 3 would allocate buffers from both numa node 0 and numa node 1 and the accesses would be interleaved on each of the threads. Out of 100 accesses on each thread, 25 accesses would be to numa node 1 and remaining 75 would be to numa node 0. Threads 4 to 7 would allocate buffers from both numa node 1 and persistent memory region mapped to a file in folder /mnt/pmfs1 and the accesses would be interleaved. Out of 100 accesses, 10 accesses would be to /mnt/pmfs1 and remaining 90 would be numa node 1. Currently, only ratios 10 and 25 are supported.

9.7.16 Measuring loaded latency with 1 Hyper-thread per core

mlc --loaded_latency –X

We can measure the loaded latency of a system using only 1 Hyper-thread per core for generating the bandwidth. This is useful because in several processors with high core count, the peak bandwidth is obtained when only 1 Hyper-thread per core is used. Specifying the -X flag results in bandwidth threads being scheduled on only one of the Hyper-threads in each core. In a 2-way SMT system, MLC uses only the first Hyper-thread among the two Hyper-threads in each core to generate bandwidth traffic. The second Hyper-thread is idle. In a 4-way SMT system (for example, Intel® Xeon® Phi® processors), specifying –X results in the first Hyper-thread being used and the remaining three Hyper-threads in the core being idle.

9.7.17 Measuring bandwidth from a set of cores to memory on a numa node

mlc --loaded_latency –k2-5,9-12,19,23-25 –j1 –T –d0

In this case, -k flag specifies the set of cores from which the threads measuring bandwidth are to run, and –j flag specifies the numa node from which memory is to be allocated for all the b/w generation threads. The list of numa nodes available in the system can be viewed with “numactl --hardware”. So in the above scenario, we measure the bandwidth from cores 2,3,4,5,9,10,11,12,19,23,24,25 to memory on numa node 1.

9.7.18 Measuring bandwidth with threads that do random accesses

mlc --loaded_latency –T –d0 –U

By default, the threads that generate the traffic for bandwidth measurements have only sequential accesses. For example “mlc --loaded_latency –T –d0” gives the peak bandwidth for read-only traffic. Adding the –U flag makes the threads follow random access patterns. Note that this is currently supported only for limited traffic types (-R, -W2 and W5). In addition to –U, –n parameter can be used to specify the range for random accesses (i.e. if we want them to be over a larger or smaller window).

9.8 Measuring cache to cache transfer latencies

9.8.1 Default invocation

mlc --c2c_latency

The above command measures the latency to transfer a modified or clean line from L2 cache of one core to another core on the same socket. Also, it measures the time taken to transfer a modified line from L3 of any socket to another socket.

9.8.2 Measuring HITM latency from remote L2 cache

mlc --c2c_latency –c2 –w22 –b200000 –C128

The above command is used to measure the time taken to transfer a modified line from L2 cache to another core on a different socket. Writer thread ‘w’ pinned to cpu 22 modifies 128KB of data (as specified in –C parameter) and transfers control to reader thread ‘c’ on cpu 2. Now, this thread reads the same 128KB of data that is currently resident in L2 of thread 22. Since those lines are in M state, the snoop responses would be Hit-modified (aka HITM) and the line would be transferred from the cache to the requester. Then the control is transferred back to the writer thread and this thread would move the window to another 128KB range in the buffer specified by –b parameter and the process will be repeated.

9.8.3 Measuring HIT latency from remote L2 cache

mlc --c2c_latency –c2 –w22 –b200000 –C200 –H

The above command is used to measure the time taken to transfer a clean line from L2 cache to another core on a different socket. Writer thread ‘w’ pinned to cpu 22 reads 200KB of data (as specified in –C parameter) and transfers control to reader thread ‘c’ on cpu 2. Now, this thread reads the same 200KB of data that is currently resident in L2 of thread 22. Since those lines are in E state, the snoop responses would be Hit-clean (aka HIT) and the line would be transferred from the cache to the requester. Then the control is transferred back to the writer thread and this thread would move the window to another 200KB range in the buffer specified by –b parameter and the process will be repeated.

9.8.4 Measuring HIT latency from remote L3 cache

mlc --c2c_latency –c2 –w22 –S24 –b200000 –C200 –H

In this case, we use a 3rd thread (selected by -S parameter) to force the cache line to be in L3. Both –w and –S parameters should select cpus on the same socket. Initially, thread on –w cpu reads the line into L2. Then, thread on –S cpu reads the same block of data (size being selected by –C). This will get the line to be present in L3 also in the shared state. Now, the control is transferred to thread on –c cpu. This thread will read the same data which will result in hits to L3 on the other socket and the data will be transferred from its cache.

Appendix

Intel® Memory Latency Checker uses code from Intel® Power Governor.
Intel Power Governor is distributed under the following license:
The BSD License

Intel® Memory Latency Checker uses code from Intel® Performance Counter Monitor. Intel Performance Counter Monitor is distributed under the following license:

Intel® Memory Latency Checker uses sample code from MSDN website (http://msdn.microsoft.com/en-us/library/windows/desktop/aa366543(v=vs.85).aspx)

The following license governs redistribution of this code :

(C) If you distribute any portion of the software, you must retain all copyright, patent, trademark, and attribution notices that are present in the software.

(D) If you distribute any portion of the software in source code form, you may do so only under this license by including a complete copy of this license with your distribution. If you distribute any portion of the software in compiled or object code form, you may only do so under a license that complies with this license.

(E) The software is licensed "as-is." You bear the risk of using it. The contributors give no express warranties, guarantees or conditions. You may have additional consumer rights under your local laws which this license cannot change. To the extent permitted under your local laws, the contributors exclude the implied warranties of merchantability, fitness for a particular purpose and non-infringement.

(F) Platform Limitation - The licenses granted in sections 2(A) and 2(B) extend only to the software or derivative works that you create that run on a Microsoft Windows operating system product.

Intel® Memory Latency Checker modified the sample IOCTL driver code from Windows Driver Kit installation for the MSR driver. The redistribution of the driver (binary only) is based on the following license

- Third Party Distribution. You may permit distributors of your programs to copy and distribute the Distributable Code as part of those programs.

ii. Distribution Requirements. For any Distributable Code you distribute, you must

- add significant primary functionality to it in your programs;
- require distributors and external end users to agree to terms that protect it at least as much as this agreement;
- display your valid copyright notice on your programs; and
- indemnify, defend, and hold harmless Microsoft from any claims, including attorneys' fees, related to the distribution or use of your programs.

iii. Distribution Restrictions. You may not

- alter any copyright, trademark or patent notice in the Distributable Code;
- use Microsoft's trademarks in your programs' names or in a way that suggests your programs come from or are endorsed by Microsoft;
- distribute Distributable Code to run on a platform other than the Windows platform;
- include Distributable Code in malicious, deceptive or unlawful programs; or
- modify or distribute the source code of any Distributable Code so that any part of it becomes subject to an Excluded License. An Excluded License is one that requires, as a condition of use, modification or distribution, that
- the code be disclosed or distributed in source code form; or
- others have the right to modify it.

3. SCOPE OF LICENSE. The software is licensed, not sold. This agreement only gives you some rights to use the software. Microsoft reserves all other rights. Unless applicable law gives you more rights despite this limitation, you may use the software only as expressly permitted in this agreement. In doing so, you must comply with any technical limitations in the software that only allow you to use it in certain ways. You may not

- work around any technical limitations in the software;
- reverse engineer, decompile or disassemble the software, except and only to the extent that applicable law expressly permits, despite this limitation;
- make more copies of the software than specified in this agreement or allowed by applicable law, despite this limitation;
- publish the software for others to copy;
- rent, lease or lend the software;
- transfer the software or this agreement to any third party; or
- use the software for commercial software hosting services.

4. BACKUP COPY. You may make one backup copy of the software. You may use it only to reinstall the software.

5. DOCUMENTATION. Any person that has valid access to your computer or internal network may copy and use the documentation for your internal, reference purposes.

6. EXPORT RESTRICTIONS. The software is subject to United States export laws and regulations. You must comply with all domestic and international export laws and regulations that apply to the software. These laws include restrictions on destinations, end users and end use. For additional information, see www.microsoft.com/exporting.

7. SUPPORT SERVICES. Because this software is “as is,” we may not provide support services for it.

8. ENTIRE AGREEMENT. This agreement, and the terms for supplements, updates, Internet-based services and support services that you use, are the entire agreement for the software and support services.

9. APPLICABLE LAW.

a. United States. If you acquired the software in the United States, Washington state law governs the interpretation of this agreement and applies to claims for breach of it, regardless of conflict of laws principles. The laws of the state where you live govern all other claims, including claims under state consumer protection laws, unfair competition laws, and in tort.

b. Outside the United States. If you acquired the software in any other country, the laws of that country apply.

10. LEGAL EFFECT. This agreement describes certain legal rights. You may have other rights under the laws of your country. You may also have rights with respect to the party from whom you acquired the software. This agreement does not change your rights under the laws of your country if the laws of your country do not permit it to do so.

11. DISCLAIMER OF WARRANTY. The software is licensed “as-is.” You bear the risk of using it. Microsoft gives no express warranties, guarantees or conditions. You may have additional consumer rights under your local laws which this agreement cannot change. To the extent permitted under your local laws, Microsoft excludes the implied warranties of merchantability, fitness for a particular purpose and non-infringement.

12. LIMITATION ON AND EXCLUSION OF REMEDIES AND DAMAGES. You can recover from Microsoft and its suppliers only direct damages up to U.S. $5.00. You cannot recover any other damages, including consequential, lost profits, special, indirect or incidental damages.

This limitation applies to

- anything related to the software, services, content (including code) on third party Internet sites, or third party programs; and
- claims for breach of contract, breach of warranty, guarantee or condition, strict liability, negligence, or other tort to the extent permitted by applicable law.

It also applies even if Microsoft knew or should have known about the possibility of the damages. The above limitation or exclusion may not apply to you because your country may not allow the exclusion or limitation of incidental, consequential or other damages.

Intel® Memory Latency Checker uses code from the Linux NVM Library https://github.com/pmem/nvml/).
The Linux NVM Library is distributed under the following license:

Everything in this source tree is covered by the previous license
with the following exceptions:

* src/jemalloc has its own (somewhat similar) license contained in
  src/jemalloc/COPYING.

* utils/cstyle (used only during development) licensed under CDDL.