Improving Real-Time Performance by Utilizing Cache Allocation Technology

Enhancing Performance via Allocation of the Processor's Cache

White Paper

April 2015

Document Number: 331843-001US

Legal Disclaimer

By using this document, in addition to any agreements you have with Intel, you accept the terms set forth below.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked “reserved” or “undefined.” Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: http://www.intel.com/design/literature.htm

Any software source code reprinted in this document is furnished for informational purposes only and may only be used or copied and no license, express or implied, by estoppel or otherwise, to any of the reprinted source code is granted by this document.

This document contains information on products in the design phase of development.

Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. Go to: http://www.intel.com/products/processor_number/

Code Names are only for use by Intel to identify products, platforms, programs, services, etc. (“products”) in development by Intel that have not been made commercially available to the public, i.e., announced, launched or shipped.

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

Intel, Intel Xeon, Intel SpeedStep, Intel Hyper-Threading, and the Intel logo are trademarks of Intel Corporation in the U.S. and/or other countries.

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

Contents

Figures

Tables

Revision History

Date	Revision	Description
April 2015	001US	Initial release.

§

1 Summary

Intel continues to add additional cores to its processors, enabling the processor to run additional workloads simultaneously. As additional threads are being processed concurrently, data in the system's caches can become invalidated and evicted quickly.

For latency sensitive workloads, the added delay to fetch the data from system memory can negatively impact the performance. To help prevent this situation from occurring, Intel has developed Cache Allocation Technology (CAT) to enable more control over the LLC cache and how cores allocate into it.

Using CAT, the system administrator can reserve portions of the cache for individual cores so that only these cores can allocate into them. As a result, other applications may not evict cache lines from these reserved portions of the cache via general use of the caches. This paper describes the Intel cache hierarchy, how to configure CAT, and scenarios in which CAT can improve performance for latency sensitive applications.

2 Background

Generation after generation, Intel's Xeon® server processors continue to house more cores. With Intel® Hyper-Threading Technology, each of these cores is capable of executing two threads simultaneously. This provides customers with a great deal of computational power and the ability to execute many different tasks in parallel. The cache is one of the many mechanisms used to increase the overall performance of the processor and aid in the swift execution of instructions by providing high bandwidth low latency data to the cores.

With the additional cores, the processor is capable of executing more threads simultaneously. Depending on the workloads and the resources required by each of the threads, the various processes can interfere with the execution time of others. One such example is the invalidation and eviction of data from the cache. While this is a normal and important operation which occurs within the processor, it may introduce additional delays for some applications. Using CAT, system administrators can reserve portions of the cache for latency-sensitive applications so that its critical data is not evicted by other lower-priority processes.

Table 1. Fourth Generation Intel® Xeon® Processor which Support CAT

Processor	L3Cache	Brand String
Intel® Xeon® E5-2658 v3 Processor (12 Core, 2.20 GHz 105 W TDP) FC-LGA12A	30 MB 20 W	Intel® Xeon® CPU E5-2658 v3 @ 2.20 GHz
Intel® Xeon® E5-2648L v3 Processor (12 Core, 1.80 GHz 75 W TDP) FC-LGA12A	30 MB 20 W	Intel® Xeon® CPU E5-2648L v3 @ 1.80 GHz
Intel® Xeon® E5-2628L v3 Processor (10 Core, 2.00 GHz 75 W TDP) FC-LGA12A	25 MB 20 W	Intel® Xeon® CPU E5-2628L v3 @ 2.00 GHz
Intel® Xeon® E5-2618L v3 Processor (8 Core, 2.30 GHz 105 W TDP) FC-LGA12A	20 MB 20 W	Intel® Xeon® CPU E5-2618L v3 @ 2.30 GHz
Intel® Xeon® E5-2608L v3 Processor (6 Core, 2.00 GHz 105 W TDP) FC-LGA12A	15 MB 20 W	Intel® Xeon® CPU E5-2608L v3 @ 2.00 GHz

3 Introduction

This section introduces the Intel® Architecture, provides a summary of Intel cache policies, delineates the technical problem at hand and describes Cache Allocation Technology.

3.1 Acronyms and Terminology

Table 1 provides definitions for the acronyms and terminology utilized in this document.

Table 2. Acronyms and Terminology

Term / Acronym	Definition
CAT	Cache Allocation Technology
CPU	Central Processing Unit
FPGA	Field Programmable Gate Arrays
Intel® DDIO	Intel® Data Direct Input/Output (I/O) Technology
LLC	Last Level Cache
LRU	Least Recently Used
MS	Millisecond
MLC	Mid-Level Cache
MSI	Message Signaled Interrupt
PCIe*	Peripheral Component Interconnect Express
µs	Microseconds

3.2 Overview of the Intel® Architecture

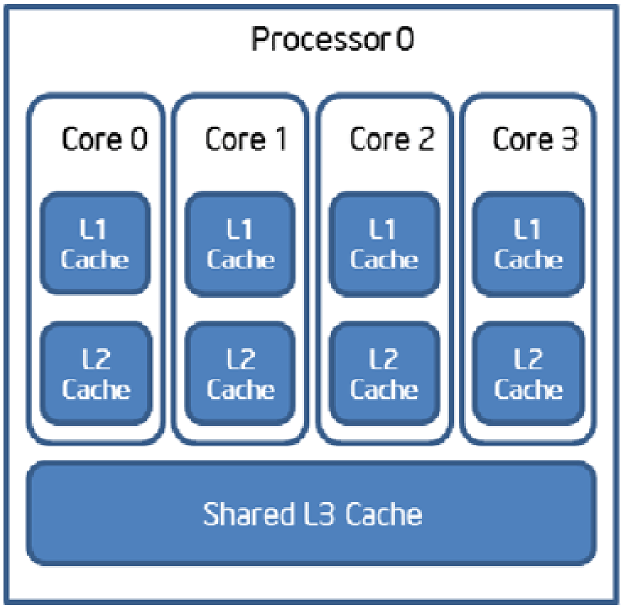
Intel® server processors are designed for heavy computational loads and include a large variety of features and technologies to improve general performance and accelerate particular tasks. Examples of these technologies include the processor's data and instruction caches, multiple cores, and Intel® Hyper-Threading technology. As the processor becomes capable of computing a growing number of tasks in parallel, there is a growing need for the ability to customize how resources are utilized within the processor. This customization can be very beneficial in use cases where certain tasks are utilizing resources, such as the cache, but not taking advantage of the possible performance gains provided by the resource.

3.3 Cache Hierarchy

Intel's Xeon® processors include three levels of cache: the L1, L2, and L3 caches. The L1 cache is the smallest, but fastest, cache and is located nearest to the core. The L2 cache, or mid-level cache (MLC), is many times larger than the L1 cache, but is not capable of the same bandwidth and low latency as the L1 cache. Similarly, the L3 cache, also known as the last level cache (LLC), is the largest and slowest cache on Intel Xeon processors and can be orders of magnitude larger than the MLC.

Figure 1 shows the cache hierarchy.

Figure 1. Cache Hierarchy



3.4 Intel Cache Policies

Each physical core contains its own private L1 and MLC caches. However, the LLC is shared and can be fully accessed and utilized by all cores in the system. On Intel Xeon v3 processors, the LLC is an inclusive cache. An inclusive cache includes all of the data that is stored in the lower level caches. If a request for data misses a core's L1 and MLC, the request then continues to the LLC to be serviced. If this results in an LLC hit, then snooping may be required to maintain coherency with another core which may have the data. Otherwise, in the case of an LLC miss, the request is serviced in memory.

In almost every case, as cores request data from memory a copy is placed in each of the caches as well. As a result, when a request misses the L1, it may hit in the MLC or LLC. However, as an LLC cache line is evicted the cache line must also be invalidated in the L1 and MLC if they exist. This can happen, for example, when a core has data in its L1 which it has not used for a time. As other cores utilize the LLC the least recently used (LRU) algorithm may eventually evict cache lines from the LLC. As a result, the cache line in the L1 and L2 cache of the core which had previously been used will become invalidated.

3.5 Problem

Xeon® processors are capable of running many threads in parallel. As a result, the contents of the shared LLC can quickly become overwritten with new data as it is requested from memory by the cores. However, this situation highly depends on the number of simultaneous threads and their memory workloads and patterns. With the right workloads and conditions, a large portion of the LLC can be quickly overwritten with new data causing significant portions of some L1 and L2 caches to be evicted and reduce the performance of the corresponding cores.

For example, assume an Intel® Xeon® powered server is running several processes and workloads. The server is also programmed with an interrupt service routine (ISR) to handle a high-priority interrupt which is very sensitive to latency. The other workloads on the server are low priority and generate a large amount of memory traffic.

During the time between interrupts, the low-priority processes generate memory traffic which may overwrite the entire LLC with new data and therefore invalidate everything in all other cores' L1 caches. In this situation, the next high-priority interrupt received will see a higher latency. This is because the code and data necessary to service the instruction are no longer in the cache and must be fetched from memory.

3.6 Cache Allocation Technology

The situation described above can be alleviated using a new feature in some new Intel processors called Cache Allocation Technology (CAT). CAT does not require any modifications to the operating system or kernel to take advantage of it. By means of defining and assigning a class of service to each core, the user can assign portions of the LLC to particular cores by limiting the amount of the LLC into which each core is able to allocate cache lines. Because the core is only able to allocate cache lines into its assigned portion of the cache, it is no longer possible for the core to evict cache lines outside of this region.

The cache is divided into ways and these ways can be divided or shared among the cores with CAT. Though a core may be limited to allocating, and therefore evicting, cache lines to a subset of the LLC, a read or write from a core may still result in a cache hit if the cache line exists anywhere in the LLC. For example, kernel code, shared data allocated by threads on other cores, data in the LLC written via Intel® Data Direct Input/Output (Intel® DDIO), may be found in the LLC outside of a core's limited region of the LLC and result in a cache hit.

CAT is particularly useful in scenarios where an offending application is requesting a large amount of data, putting pressure on other applications, but never reusing the data that is cached in the LLC. File hosting and video streaming programs are examples of these types of applications. On an otherwise idle system, these applications can consume the entire LLC, but never take advantage of the LLC because this offending application is not reusing most of the data that it requests. The core on which an offending application is running can be restricted to a small region of the cache. As a result, other applications have a better opportunity to benefit from the LLC but, at the same time, there is a potential for decreasing the offending application’s performance. The performance impact on the offending application depends on its workload, the size of its working set, and other factors.

Assume an application is processing data linearly on an otherwise idle system. The application has a 40MB working set and the LLC is 20MB in size. As the program processes the first 20MB of its working set, the data misses the LLC and becomes cached in the LLC after being fetched from memory. However, as the program makes requests for the next 20MB, the requests will again miss the LLC and must be serviced by memory. To make room for the next 20MB of data, the least recently used data is evicted as the following 20MB of the working set are requested. As the program continues, it thrashes the LLC as it requests new data and evicts old data in a continuous manner without hitting any of the data in the LLC.

In scenarios such as these, a user can configure the system such that the offending application is bound to a particular core(s). Additionally, the user can configure these same cores to only use, for example, ten percent of the LLC. Now the application can only thrash this ten percent of the cache, leaving the remaining ninety percent untouched. (Intel's Cache Monitoring Technology (CMT) can be used to track each core's LLC cache occupancy. For more information on CMT please see http://www.intel.com/content/www/us/en/communications/cache-monitoring-cache-allocation-technologies.html or chapter 17.14 of the *Intel Software Developer's Manual*.)

Furthermore, in this hypothetical scenario, with the program linearly processing 40MB of data, because the application never reused the data before being thrashed in the LLC, the application’s performance is unhindered despite the reduction in the amount of the LLC into which it can allocate. This is because the performance bottleneck is still found in the memory bandwidth and latency. By limiting the cache, which the core always misses and never hits, the performance is not reduced. The cache will still miss and the memory will continue to fulfill the requests for data. However, in other scenarios where the application is more dependent on a data set which is found in the LLC (rather than RAM), utilizing CAT in this way may result in decreased performance for the application. Refer to *Intel Software Developer's Manual* chapters 17.15 for more information on CAT.

4 Measurement and Analysis

This section provides comprehensive measurement and data analysis.

4.1 Performance Measurement Environment and Test Cases

The following test cases compare the performance of multiple programs when run with and without CAT enabled and configured to prioritize the LLC for sensitive applications. Various workloads will be used to highlight the effects of CAT in different use cases.

4.1.1 Test Case 1: MSI Latency

CAT can reduce MSI latency and jitter and as a result benefit applications that are sensitive to the performance of MSIs. As other workloads are run on the system between executions of the MSI handler, the system may evict the MSI handler code from cache. If this occurs, the next MSI received will result in the fetching of the handler from memory before the MSI can be serviced. This results in increased latency and jitter as the handler is required to be fetched from memory before processing the MSI. With CAT the user can prevent other cores, as they execute their workloads, from evicting the interrupt handler code from cache.

To demonstrate this, a system is configured with an external PCIe device which generates an MSI every 1ms. An interrupt handler is registered on the DUT to service interrupts and send PCIe writes back to the device. The card calculates the time between when the MSI was generated and when the PCIe write was received to determine how quickly the interrupt was handled. Simultaneously, other cores will be used to generate memory traffic to simulate a real-world scenario where other programs will be running and utilizing the cache and RAM. These applications read linearly from memory in a span much larger than the size of the LLC. To compare the effects of CAT, two tests will be run: one with CAT disabled and the other with CAT enabled.

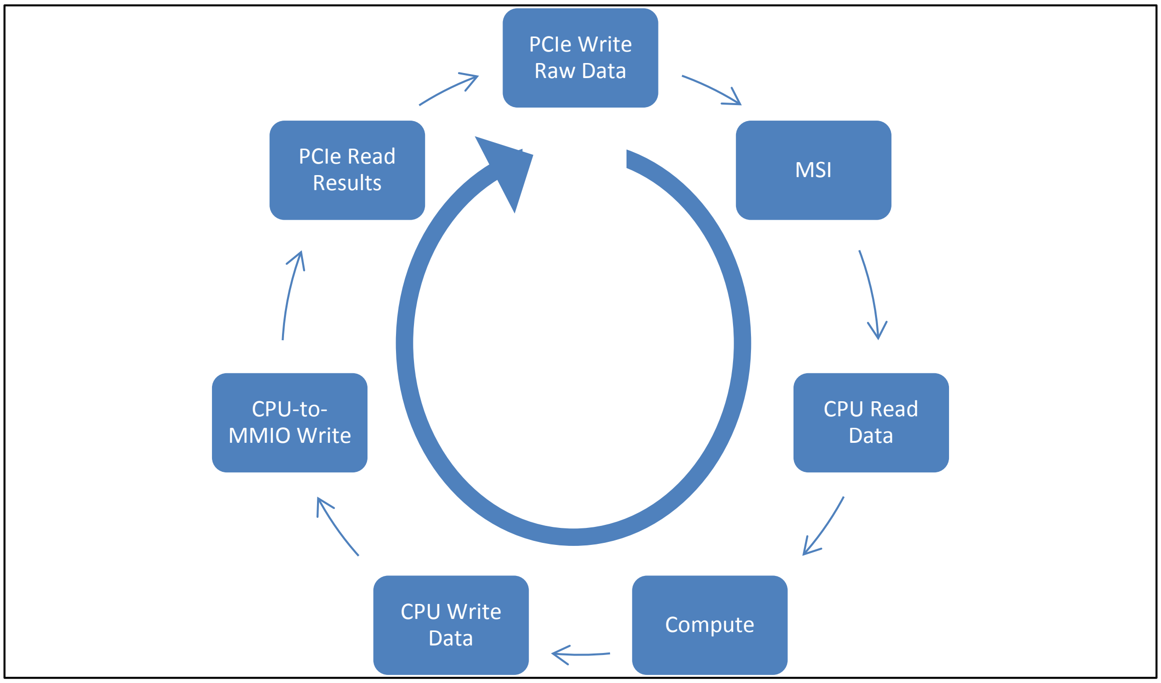
4.1.1.1 System Configuration

- Limit cores to a reduced set of the cache via CAT (the example below assumes a single core (core 1) as the core which will be generating CPU-to-MMIO traffic).
  - Set the default way mask to not include the last four cache ways
    - wrmsr 0xc90 0xFFFF0
  - Set a second way mask to only include the last four cache ways
    - wrmsr 0xc91 0xF
  - Set core 1 to use the second way mask
    - wrmsr –p 1 0xc8f 0x100000000

4.1.2 Test Case 2: A Real-Time Cycle

To simulate a real-time use case, the following cycle is the basis for these experiments. Several different utilities are executed simultaneously to simulate different portions of the cycle. Figure 2 illustrates the real-time cycle.

Figure 2. A Real-Time Cycle



An external PCIe* device writes current sensor, motor position, status values, etc., upstream to the system. After completing the writes, an MSI is used to notify the CPU that the data is available and complete. Upon receiving the interrupt, the CPU reads the data, computes new values based on the input, and then writes this data to cache/memory. The CPU sends a PCIe* write to the PCIe* device once these steps have been completed to notify of it of the updated data. After receiving the MMIO write from the CPU, the PCIe* device reads the results.

In this workflow, there are many situations where CAT can be utilized to improve performance. As seen from the previous tests, CAT can be used to improve performance when handling interrupts. In this test, other workloads can also benefit from CAT.

CAT can be used to preserve the cache lines utilized for sending/receiving data to/from the PCIe* device. In this case, CAT can be used to improve the performance of the "PCIe Read Results" step in Figure 2. This can be achieved by locking cache lines into a small portion of the cache that cannot be allocated by cores or Intel® DDIO. By ensuring these cache lines remain in cache they will always be available for immediate use in the LLC for Intel® DDIO instead of waiting for the cache lines to be fetched from main memory.

To add additional strain on the system, additional programs were executed in parallel to generate CPU to memory traffic; these applications read data linearly from memory in a span much larger than the size of the LLC.

4.1.2.1 System Configuration

- Lock target cache lines into the cache via CAT
  - Set the default way mask to not include the last two cache ways
    - wrmsr 0xc90 0xFFFFC
  - Set a second way mask to only include the last two cache ways
    - wrmsr 0xc91 0x3
  - Set core 1 to use the second way mask
    - wrmsr –p 1 0xc8f 0x100000000
  - Use core 1 to touch all of the memory locations one wishes to lock permanently into the cache
  - Set core 1 to use the default (first) way mask
    - wrmsr –p 1 0xc8f 0x000000000

4.2 Test Environment Configuration

The test environment for the data presented is described in Table 2.

Table 3. Test Environment

Platform	Grantley
Processor	Intel® Xeon® Processor E5-2658v3
CPU Frequency	2.20GHz

Intel® HT Technology	Disabled
C/P/T States	Disabled
Intel SpeedStep® Technology	Disabled
Memory	16GB (2x8GB) DDR4 Registered ECC 2133 MHz
Operating System	Red Hat* Enterprise Linux Server 6.3 (Kernel 3.7.10)
PCIe* Exerciser card	2 PCIe* Gen3 x4 cards

Note: The PCIe* exerciser card is an in-house hardware tool.

4.3 Measurement Results

The measurement results are detailed in this section.

4.3.1 Test Case 1: MSI Latency

For this test an external PCIe* device generates an interrupt every 1ms and the time taken to handle the interrupt is recorded. In the tables below, two test cases are shown. MSI latency when the system is loaded with concurrent CPU-to-Memory traffic, and loaded with concurrent CPU-to-Memory traffic with CAT configured. When the system is running concurrent CPU-to-Memory traffic with CAT configured, the average MSI latency is ~2.9 µs lower (~64% decrease). Minimum latency (decreased by ~27%) and maximum latency (decreased by ~32%) also saw improvements in their latency by utilizing CAT to optimize cache utilization.

Table 3 shows the MSI Latency.

Table 4. MSI Latency

MSI Latency	Minimum Latency	Maximum Latency	Average Latency
Loaded without CAT	1.66 µs	30.91 µs	4.53 µs
Loaded with CAT	1.22 µs	20.98 µs	1.62 µs

The aggregate bandwidth from the programs generating CPU-to-Memory traffic was also recorded with and without CAT. The results show that these workloads were unaffected by the reduced size of the cache into which they can allocate cache lines.

Table 4 shows the aggregate bandwidth.

Table 5. Aggregate Bandwidth

Loaded CPU-to-Memory Traffic	Aggregate Bandwidth
Without CAT	32.5 GB/s
With CAT	32.5 GB/s

4.3.2 Test Case 2: A Real-Time Cycle

In the second test case, several workloads which are commonly included in a real-time cycle are running concurrently. The maximum PCIe* read latency is recorded after running the workload for an hour.

Table 5 shows the maximum recorded PCIe* latency after 1 hour.

Table 6. Maximum Recorded PCIe* Latency after 1 Hour

Loaded CPU-to-Memory Traffic	Maximum Recorded PCIe Read Latency After 1 hour
Without CAT	1.360 µs
With CAT	1.200 µs

As with test case 1, the aggregate bandwidth from the additional programs generating CPU-to-Memory traffic was also recorded with and without CAT. The results show that these workloads were unaffected by utilizing CAT in this manner.

Table 6 shows the aggregate bandwidth from extra CPU-to-memory workloads.

Table 7. Aggregate Bandwidth from Extra CPU-to-Memory Workloads

Loaded CPU-to-Memory Traffic	Aggregate Bandwidth from Extra CPU-to-Memory Workloads
Without CAT	18.5 GB/s
With CAT	18.5 GB/s

5 Conclusion

In some scenarios, offending applications hinder the performance of other processes on a system by generating a large amount of CPU-to-Memory traffic and quickly allocating new cache lines into the LLC. This white paper describes Intel's Cache Allocation Technology, how to use it, and shows its effectiveness in test cases without modifying the operating system or kernel. The test cases described show the performance gain possible in these scenarios by the use and proper configuration of CAT.

In the first test case, MSI latency was decreased by utilizing CAT to limit the offending applications which are simultaneously generating CPU-to-Memory traffic. The minimum, average, and maximum MSI latencies saw a significant performance gain by utilizing CAT. In particular, the average MSI latency was decreased by ~64%. Additionally, this performance was achieved without degrading the CPU-to-Memory performance of the other applications used in this test case.

The second case was run in a much more involved use case with many applications running in parallel to simulate a real-time use case. In this scenario, without CAT the maximum PCIe* read latency measured in an hour was 1.36µs. By configuring CAT to preserve the cache lines containing the high-priority data, the maximum PCIe* read latency measured in an hour was reduced by ~12% to 1.2µs.

On systems with multiple workloads running simultaneously, which heavily utilize the cache; system administrators can utilize CAT to improve performance. CAT can be used to better utilize the caches and as a result some applications are more likely to hit the cache which results in fewer accesses to system memory. This results in lower latency and greater performance.

§