Detecting Memory Bandwidth Saturation in Threaded Applications

<keywords: threading, scalability, Thread Profiler, VTune, memory>

Abstract

Memory sub-system components contribute significantly to the performance characteristics of an application. As an increasing number of threads or processes share the limited resources of cache capacity and memory bandwidth, the scalability of a threaded application can become constrained. Memory-intensive threaded applications can suffer from memory bandwidth saturation as more threads are introduced. In such cases, the threaded application won't scale as expected, and performance can be reduced. This article introduces techniques to detect memory bandwidth saturation in threaded applications.

This article is part of the larger series, "The Intel Guide for Developing Multithreaded Applications," which provides guidelines for developing efficient multithreaded applications for Intel® platforms.

Background

As modern processors include more cores and bigger caches, they become faster at a higher rate than the memory sub-system components. The increasing core count on a per-die basis has put pressure on the cache capacity and memory bandwidth. As a result, optimally utilizing the available cache and memory bandwidth to each core is essential in developing forward-scaling applications. If a system isn't capable of moving data from main memory to the cores fast enough, the cores will sit idle as they wait for the data to arrive. An idle core during computation is a wasted resource that increases the overall execution time of the computation and will negate some of the benefits of more cores.

The current generation of Intel® processors based on the Nehalem architecture moved from the traditional front-side-bus (FSB) approach to non-uniform memory access/architecture (NUMA) model to increase the available memory bandwidth to the cores and reduce the bandwidth saturation issues mentioned above. Figure 1 depicts the FSB to NUMA transition.

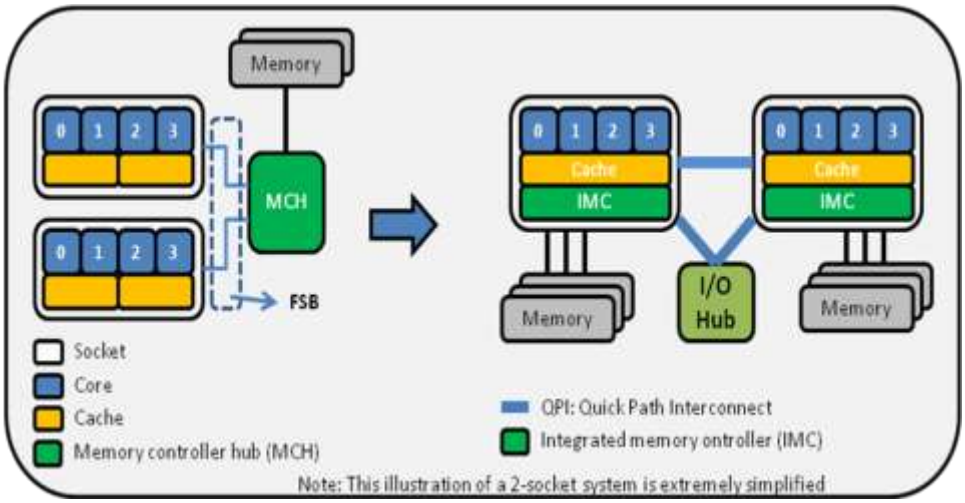

Figure 1. Transition from FSB to NUMA.

The clear symptom of bandwidth saturation for any parallel application is non-scaling behavior. In other words, an application that has saturated the available memory bandwidth will not scale effectively to more threads or cores. However, there are many causes for multi-threaded applications not to scale and some of these performance inhibiting factors include threading overhead, synchronization overhead, load imbalance, and inappropriate granularity. Intel® Thread Profiler is designed to identify such performance issues at the application level.

The following results are taken after running the STREAM benchmark version 5.6 with various numbers of threads (only triad scores are shown).

	Function	Rate (MB/s)	Avg time	Min time	Max time
1 Thread	Triad:	7821.9511	0.0094	0.0092	0.0129
2 Threads	Triad:	8072.6533	0.0090	0.0089	0.0093
4 Threads	Triad:	7779.6354	0.0096	0.0093	0.0325

It is easy to see that STREAM does not benefit from having more threads on this particular platform (a single-socket Intel® Core™ 2 Quad-based system). Closer inspection of the results shows that even though there was a slight increase in the triad score for the two-thread version, the four-thread version performed even worse than the single threaded run.

Figure 2 shows Intel Thread Profiler analysis of the benchmark. The timeline view reveals that all threads are perfectly balanced and have no synchronization overheads. While it is a powerful tool for identifying threading performance issues at application level, Intel Thread Profiler will not detect memory bandwidth saturation in threaded applications.

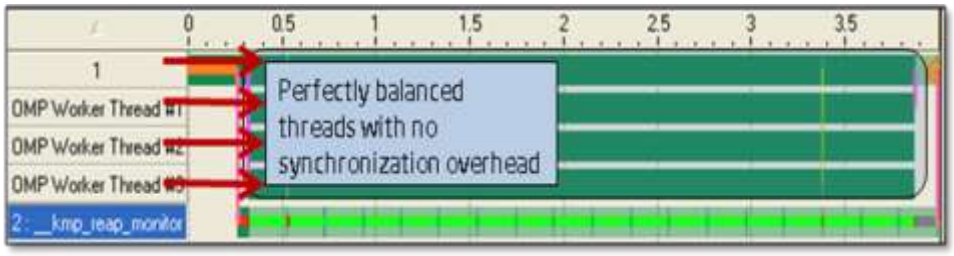

Figure 2. Intel Thread Profiler timeline view of the STREAM benchmark using four OpenMP* threads.

Advice

Intel® VTune™ Performance Analyzer and Performance Tuning Utility (PTU) used in combination with event-based sampling (EBS), can help developers measure application bandwidth usage, which can then be checked against the achievable (or theoretical) bandwidth on the system. Event-based sampling relies on the performance monitoring unit (PMU) supported by the processors.

VTune analyzer and PTU can help developers estimate the memory bandwidth usage of a particular application by using EBS. On Intel® Core™ 2 microarchitecture CPU_CLK_UNHALTED.CORE and BUS_TRANS_MEM.ALL_AGENTS performance events can be used to estimate the memory bandwidth.

- The CPU_CLK_UNHALTED.CORE event counts the number of core cycles while the core is not in a halt state. The core enters the halt state when it is running the HLT instruction.

- The BUS_TRANS_MEM.ALL_AGENTS event counts activity initiated by any agent on the bus. In systems where each processor is attached to a different bus, the count reflects only the activity for the bus on which the processor resides.

On Core 2-based systems memory bandwidth can be estimated by using the following formula:

(64 * BUS_TRANS_MEM.ALL_AGENTS * CPU Frequency) / CPU_CLK_UNHALTED.CORE

Module	Process	BUS_TRANS_ANY.ALL_AGENTS events	CPU_CLK_UNHALTED.CORE events
stream	stream	1,419,200,000	35,576,000,000

Figure 3. VTune analyzer EBS analysis of STREAM with four threads.

Figure 3 shows the EBS results of the STREAM benchmark when four threads were used. By using the above formula, it is possible to estimate the memory bandwidth usage of STREAM as 7.6Gb/sec.

Memory Bandwidth = (64 * 1,419,200,000 * 2.9GHz) / 35,576,000,000 = 7.6GB/sec

The STREAM-reported sustainable Triad score was 7.7GB/seconds, so the VTune analyzer-based calculation is quite reasonable. The STREAM benchmark was chosen to demonstrate how memory bandwidth measured using EBS can approximately measure the achievable memory bandwidth on a particular system.

If an application doesn't scale when more threads are added to take advantage of the available cores, and if Intel Thread Profiler doesn't show any application-level threading problems as mentioned above, then the following three steps can help the user determine whether or not a particular application is saturating the available memory bandwidth:

1. Run STREAM or similar benchmarks to get an idea of the sustainable memory bandwidth on the target system.

2. Run the target application under VTune analyzer or PTU and collect the appropriate performance counters using EBS. For Core 2 microarchitecture, these events are again CPU_CLK_UNHALTED.CORE and BUS_TRANS_MEM.ALL_AGENTS (Formula 1).

3. Compare VTune analyzer-measured memory bandwidth numbers to the sustainable or achievable memory bandwidth measured in step 1. If the application is saturating the available bandwidth, then this particular application won't scale with more cores.

Generally speaking, a memory-bound application (one whose performance is limited by the memory access speed) won't benefit from having more threads.

Usage Guidelines

The new Intel® Core™ i7 and Xeon® 5500 series processors are referred to as having an "uncore, " which is that part of the processor that is external to all the individual cores. For example, the Intel Core i7 processor has four cores that share an L3 cache and a memory interface. The L3 and memory interface are considered to be part of the uncore (see Figure 4).

Neither the VTune analyzer nor PTU support the sampling of events that are triggered in the uncore of the processor, and the memory bandwidth measurement must be performed differently. The relevant performance events used for measuring bandwidth are not sampled using EBS as is usual with VTune analyzer or PTU; rather, they are counted using time-based sampling. This means that the bandwidth is measured for the entire system over a designated time range, and it isn't possible to see how much of the bandwidth usage comes from specific functions, processes, and modules.

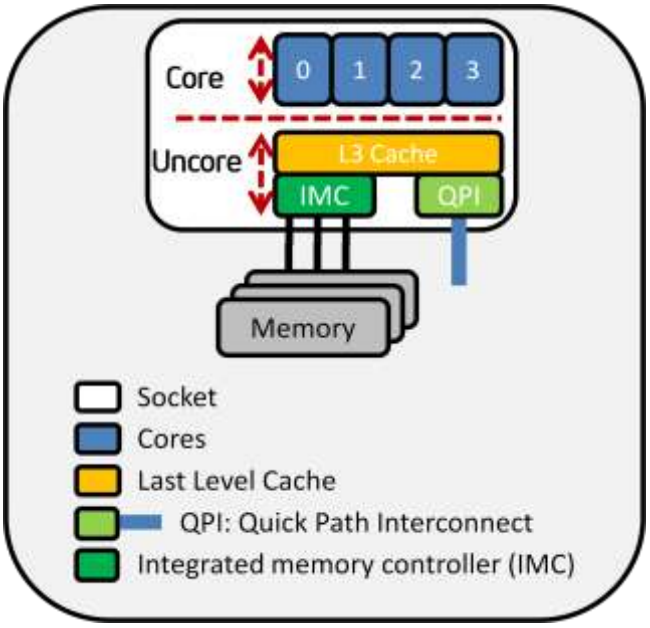

Figure 4. Simplified block diagram of a 4 core Nehalem processor.

The formula given above can be used to measure the memory bandwidth usage of any application, module, or function on Core 2 based systems except on Core 2 based Xeon MP processors, which also have uncore parts. The basic formula for measuring the memory bandwidth on Nehalem architecture-based systems can be given as follows:

> Memory Bandwidth = 1.0e-9 * (UNC_IMC_NORMAL_READS.ANY+UNC_IMC_WRITES.FULL.ANY)*64 / (wall clock time in seconds)

Additional Resources

Intel® Software Network Parallel Programming Community

Intel® VTune™ Performance Analyzer

STREAM: Sustainable Memory Bandwidth in High Performance Computers

Intel® Performance Tuning Utility

How Do I Measure Memory Bandwidth on an Intel® Core™ i7 or Intel® Xeon® Processor 5500 Series Platform Using Intel® VTune™ Performance Analyzer?