**LAB 3. Inter-thread Interferences:   
Bandwidth Contention through the Memory Hierarchy**

**LEARNING GOALS**

* Realize to what extent the thread interference at main memory can impact on performance.
* Understand the impact of cache contention on performance.
* Work with synthetic applications (*microbenchmarks*) and understand how they can be used to model certain co-runner behaviors.
* Study the performance degradation due to bandwidth contention.

1. **Theoretical Concepts**

**Sharing the memory system**

Most microprocessors nowadays implement multiple cores on the same chip and they are called multicore processors. These processors provide higher performance than single-core processors while attacking their power, cooling, and package costs issues. The memory system in a multicore processor is shared by all the processor cores. These cores always share the main memory and usually the last level cache (LLC). The L2 cache can be private or shared and the L1 cache is always private to provide a fast access from the core to the stored data. Figure 1 presents the memory hierarchy of the processor used in our experimental platforms. As it can be observer, each core disposes of its private L1 and L2 caches, while the LLC and main memory are shared among all the cores of the processor.



Figure 1. Memory hierarchy of the Intel i5 4590 processor.

Applications running concurrently on a multicore processor can therefore interfere when accessing the shared LLC and main memory. In fact, the main memory is known to be one of the major performance bottlenecks of the system if many applications try to access the main memory frequently. Consequently, a lot of research effort has been devoted to this problem during the last years. In this lab session, we will quantify the impact of bandwidth contention at main memory and the LLC on a current commercial processor.

1. **Lab Setup**

In this lab, we monitor the performance of applications running on the system. To avoid any possible interference between the applications to be monitored and the user applications (e.g. the OS user interface, a web browser, or a spreadsheet software), the experiments will be launched on a remote server. To this end, a remote server will be assigned to each student. This server will be accessed through ssh and all the experiments should be launched on this system[[1]](#footnote-1).

|  |
| --- |
| $ ssh [user@semXXX.upv.es](mailto:user@semXXX.upv.es) |

*Download the scheduling framework and the files required to perform the lab (libpfm library, benchmark binaries and input files). The files can be downloaded from a github repository. Then, compile the libpfm library and the scheduling framework. These actions can be performed issuing the following commands.*

|  |
| --- |
| $ git clone https://github.com/jofepre/lab\_sched\_framework/tree/master/lab\_sessions/lab\_3/Scheduling\_framework  $ cd libpfm-4.8.0/  $ make  $ cd scheduling\_framework/ |

1. **The Mem-Bounded Microbenchmark**

When several applications run concurrently, each of them can only use a fraction of the shared resources implemented in the system (e.g., the last level cache capacity or bandwidth). Many times, we want to analyze the performance of the applications assuming that the use of a given resources is limited (due to sharing). These studies can be performed in a real system, with relative simplicity, using auxiliary applications also known as microbenchmarks.

A microbenchmark is a small application designed to use a given shared resource of the system. Typically used microbenchmarks are the ones that stress the memory system or the floating-point units. A microbenchmark should be able to achieve the highest possible utilization of the target resource, but without interfering (or reaching the minimum interference possible) in the other shared resources. This characteristic might seem straightforward but can be difficult to achieve when dealing with strongly connected resources such as the resources of the memory system.

In this lab session, we will work with the mem-bounded microbenchmark, which has been designed to stress the main memory bandwidth of the system. You can check its source code in the file mm-microbenchmark.c. To this end, mem-bounded allocates a huge array of data and repeatedly goes over the first element of each row of the matrix to increment its value. Assuming that the amount of data accessed exceeds the last level cache capacity, each access to a matrix row ends up causing a main memory access.

The matrix allocation is important to ensure that the microbenchmark behaves as expected. First of all, notice that if the matrix size does not exceed the LLC capacity, the microbenchmarks will hit many times at the LLC. In this case, it will not perform as many main memory accesses as possible, missing its goal. The second important parameter to succeed is the stride (distance in bytes between two consecutive accesses to the matrix). Since the microbenchmark only accesses the first element of each row, the stride matches the row length (in bytes). Short strides allow the hardware prefetcher to fetch the data before it is actually requested and thus, it might result in LLC hits, which is not the goal of the microbenchmark[[2]](#footnote-2). However, large strides reduce the data that is effectively accessed (only the first element of each row is accessed) and might allow the accessed data to fit the LLC.

**Microbenchmark details**

For our target architecture, with a 6 MB LLC cache, we could allocate a 12 MB matrix with 64 bytes rows. This row size matches the size of the cache blocks. Hence, given our access pattern, each access to the matrix would be done on a different cache line and, therefore, the data should always be replaced from the LLC cache before it is accessed again. With this data allocation, the microbenchmark would meet its goal if we do not take hardware prefetching into account. However, when prefetching is enabled we can experimentally see that the hit ratio at the LLC is by 65%, which means that the prefetch engine is able to fetch the data before the microbenchmark actually requests it.

Since our goal is to stress the main memory bandwidth, we want that all the matrix accesses miss on the LLC. To achieve this behavior without turning off prefetching, our best option is to increase the access stride to the data matrix. By increasing the access stride (row length) to 512 bytes, we prevent that the prefetchers fetch the data that the microbenchmark will access in advance. Notice that now, we are allocating a 96 MB data matrix, even though the microbenchmark only accesses 12 MB (the first cache block of each matrix row).

Finally, we introduce a loop of *nop*-operations after each access to the matrix to allow configuring the main memory bandwidth the microbenchmark uses. The higher the number of *nops* the microbenchmark perform between the matrix accesses, the lower its bandwidth utilization will be. The number of *nops* can be set through an input parameter and will allow us to carry experiments under different main memory bandwidth scenarios.

**Microbenchmark characterization**

*Study the main memory bandwidth utilization of the microbenchmark according to the number of nops performed per main memory access. The microbenchmark has two input parameters. The first one is the number of nops. The second one is the number of iterations through the matrix and defines the length of the experiment (a thirty-seconds experiment should be representative; notice that the execution time of the experiment grows with the number of nops). Represent the results in a chart showing how the bandwidth utilization decreases with the number of nops. Then, identify the number of nops required to achieve a bandwidth utilization by 100, 50, and 10 main memory accesses per microsecond.*

1. **Studying the Bandwidth Interference at Main Memory**

In this Section, we are going to analyze the performance degradation of the SPEC CPU2006 benchmarks due to main memory bandwidth contention. *Before performing the experiments, answer the following questions:*

* *Do you expect the performance of all the SPEC benchmarks to suffer a similar degradation? Or do you think there will be application that suffer higher performance degradation than others? In that case, can you guess which application will suffer higher performance degradation?*

To analyze the main memory bandwidth interference, we will run each SPEC benchmark with several instances of the mem-bounded microbenchmark. More precisely, the SPEC benchmark will run on a core of the processor and each remaining core will run an instance of the microbenchmark to maximize the bandwidth contention generated.

*Study the performance degradation of the SPEC benchmarks when setting the bandwidth utilization of each mem-bounded microbenchmark to 100, 50, and 10 accesses per microsecond. You can edit the script script\_deg\_mm.sh provided as a baseline to launch the required experiments. The variables nops\_bw\_100, nops\_bw\_50, and nops\_bw\_10 should be set to the number of nops required by the microbenchmarks to achieve a bandwidth utilization by 100, 50, and 10 transactions per microsecond, respectively.* Notice that the microbenchmarks are launched in background and allocated on the cores 1, 2, and 3. Notice also that the number of iterations of the microbenchmarks is set to 0, which means that they will be running until they are killed by the script (line 37). The subset of benchmarks to be studied in this experiment has already been set in the microbenchmark (line 24).

*Analyze the performance degradation suffered by the SPEC benchmarks and identify whether or not is there a connection between the bandwidth utilization of the benchmarks (measured in the “*Lab 1: Understanding the Basics on Cache Hierarchy Performance and System Performance”*) or not. IPC values for the isolated execution of the benchmarks can also be reused from this lab session.*

1. **EXTENSION: Studying the Bandwidth Interference at the Last Level Cache**

In this Section, we propose to study the performance degradation of the SPEC CPU 2006 benchmarks due to LLC bandwidth interference. *To this end, you should first design a LLC-bounded microbenchmark, which should always miss in the L2 cache and hit in the LLC. This microbenchmark can be developed from the main mem-bounded microbenchmark by editing the allocation of the data matrix and the access pattern. Notice that the amount of data that is effectively accessed should fit in the 6 MB of the LLC in our experimental platform. Once the microbenchmark is designed you should ensure its correct operation and characterize its bandwidth consumption depending on the number of nop operations performed per LLC access.*

*After the LLC-bounded microbenchmark has been characterized, study the performance degradation of the SPEC benchmarks due to LLC bandwidth contention. To this end, the scripts devolved in Section 4 can be edited and reused. Are the benchmarks that suffer higher performance degradation due to LLC bandwidth contention the same ones that suffer higher performance degradation due to main memory bandwidth contention? Which is the reasoning behind this behavior?*

1. This lab session (including the scheduling framework configuration) has been prepared to be run on an Intel i5 4590. Performance events might differ (in name or implementation) in other architectures. [↑](#footnote-ref-1)
2. A random-access pattern would also avoid the prefetching interference, but can reduce the bandwidth utilization due to the time required to get a random number. [↑](#footnote-ref-2)