Bandwidth Bandit: Understanding Memory Contention

David Eklov, Nikos Nikoleris, David Black-Schaffer and Erik Hagersten
Uppsala University, Department of Information Technology
{david.eklov,nikos.nikoleris,david.black-schaffer,eh}@it.uu.se

***Abstract*—Applications that are co-scheduled on a multicore compete for shared resources, such as cache capacity and memory bandwidth. The performance degradation resulting from this contention can be substantial, which makes it important to effectively manage these shared resources. This, however, requires insight into how applications are impacted by such contention.**

In this paper we present a quantitative method to measure applications' sensitivities to different degrees of contention for off-chip memory bandwidth on real hardware. This method is then used to demonstrate the varying contention sensitivity across a selection of benchmarks, and explains why some of them experience substantial slowdowns long before the overall memory bandwidth saturates.

I. Application Sensitivity to Memory Contention

From a performance point of view the memory hierarchy can be described by two metrics: its latency and bandwidth. These two metrics are intimately related. Using Little's law [1], the average bandwidth achieved by an application can be expressed as follows:

$$bandwidth = transfer\_size \times \frac{MLP}{latency}, \qquad (1)$$

where MLP is the application's average Memory Level Parallelism, that is, the average number of concurrent memory requests it has in-flight, and $latency$ is the average time to complete the application's memory accesses. The above equation clearly illustrates that the bandwidth consumed by an application is determined by both its memory access *latency* and its memory *parallelism*. While the bandwidth of an application largely determines how much contention it generates, its sensitive to off-chip memory contention is instead determined by its *latency*- and *bandwidth-sensitivity*.

A. Latency Sensitivity

An application is latency-sensitive if its performance significantly degrades when its average memory accesses latency is increased due to contention. For example, consider an application whose main computation is based on a linked list traversal (i.e. pointer chasing) and with a data working set larger than the last-level cache such that most memory accesses result in cache misses. The memory accesses to the list elements are data dependent on the memory access to the previous list element. The degree of latency-sensitivity of the application is therefore, at first order, determined by its computational intensity (i.e. the number of computations performed per memory access), and how much of the memory latencies can be hidden by out-of-order execution. If the computational intensity is too low, there are not enough computations to hide the memory latency. Consequently, the application's execution time is determined by the memory access latency, in which case the application is highly latency-sensitive.

B. Bandwidth Sensitivity

In addition to latency-sensitive applications, some applications are *bandwidth-sensitive*. For example, consider an application with a simple stride access pattern. Prefetchers in any modern processor will easily be able to detect this pattern and prefetch the data well in advance, thereby hiding its access latency. The performance of this application will therefore not be directly affected by increased access latencies as long as the prefetcher can fetch far enough ahead. However, if the access latency does increase, the prefetcher will require more memory parallelism to supply data at the same rate. But, if there is not enough memory parallelism available, the data will not be delivered at the required rate and the application's performance will suffer. Such bandwidth-sensitive applications are sensitive to the memory parallelism available in the memory hierarchy.

II. The Bandwidth Bandit

The Bandwidth Bandit method enables us to measure how an application's performance is impacted by contention for shared off-chip memory resources, and determine the application is degrees of latency- and bandwidth-sensitive. It works by co-running the application whose performance we want to measure (the *Target*) with a *Bandit* application that generates contention for the shared off-chip memory resources. To accomplish this, the Bandit accesses memory at a specified rate and in a controlled pattern that ensures that it generates the desired amount and type of contention. By measuring the Target's performance while varying the amount of contention the Bandit generates, we can obtain the Target's performance as function of contention for the memory system. A detailed discussion of how to implement the Bandit application can be found in [2].

III. Results

In this section we present the results of using the Bandwidth Bandit method on a set of applications from the SPEC2006 and PARSEC benchmarks suites. We selected eight benchmarks (five from SPEC and two from PARSEC) that have large bandwidth demands, as such applications are generally more sensitive to memory contention. All benchmarks were run to completion with their reference input sets. Our goal is to investigate how sensitive individual threads are to memory

978-1-4673-1146-5/12/$31.00 ©2012 IEEE

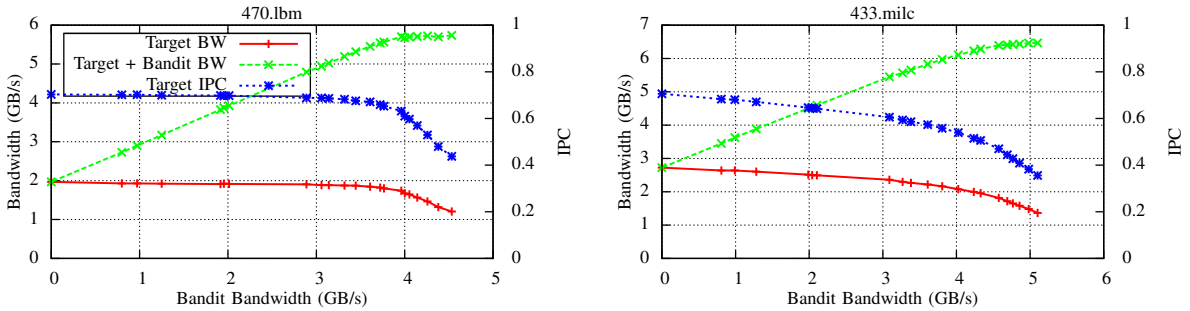

Fig. 1. Bandwidth Bandit data: Performance and bandwidth as a function of memory contention on a quad core Intel Xeon e5520 (Nehalem) machine. The graphs show Target's bandwidth (left, red) and IPC (right, blue), and total system bandwidth (right, green), as a function of bandwidth "stolen" by the Bandit (x-axis). The slope of the IPC data indicates the application's sensitivity to memory contention.

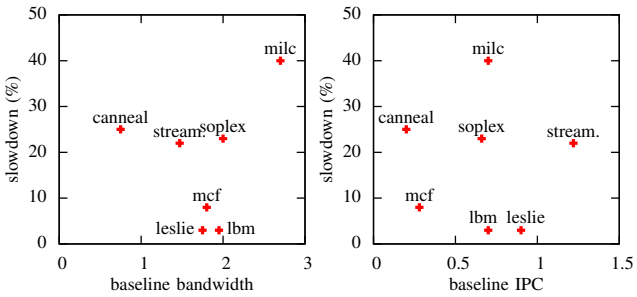

Fig. 2. Correlation of slowdown under memory contention with baseline bandwidth (left) and IPC (right) showing that both the baseline bandwidth and performance is a poor indicator of sensitivity to memory contention.

contention. We therefore ran the two PARSEC benchmarks with a single thread.

The data obtained using the Bandwidth bandit is shown in Figure 1 for two SPEC2006 benchmarks: Milc and Lbm. Lbm's bandwidth and IPC are virtually flat up to the point when the total bandwidth (Target + Bandit) levels off, which indicates that the off-chip bandwidth has saturated. Beyond the point when the bandwidth saturates, Lbm's bandwidth and IPC decrease proportionally to the amount of bandwidth the bandit steals. This indicates that Lbm is intially latancy-insensitive but becomes bandwidth-sensitive when the off-chip bandwidth saturates. On the other hand, Milc's bandwidth and IPC starts to decrease almost immediately when the Bandit starts stealing bandwidth, which indicate that it is latency-sensitive. Milc's performance (IPC) drops by almost 30% before the system bandwidth saturates.

Previous work [3], [4] has highlighted the correlation between an application's bandwidth demand and its sensitivity to contention. To investigate this we plot the correlation between application slowdown and baseline bandwidth in Figure 2. The baseline bandwidths are the applications' bandwidth demands when run alone on the machine. Here, we compute the slowdown as the baseline IPC relative to the IPC at the point when the total bandwidth (Target + Bandit) reaches 90% of the system saturation bandwidth. This operating point ensures that there is sufficient extra bandwidth in the system to avoid complete saturation.

The graphs in Figure 2 show hardly any correlation between the slowdowns and the baseline bandwidths. For example, Canneal and Streamcluster have the lowest (but different) baseline bandwidths, but large (and very similar) slowdowns, while Lbm and Soplex have virtually the same baseline bandwidth, but very different slowdowns. The right graph in Figure 2 shows the correlation between the slowdowns and the corresponding baseline IPCs. This graph shows even less correlation. These graphs indicate that neither the baseline bandwidth nor the baseline IPC are good indicators of an application's sensitivity to memory contention.

IV. Conclusions

This paper presented the Bandit Bandwidth, a quantitative method to measure applications' sensitivity to contention for off-chip memory bandwidth. Using the Bandwidth Bandit method we were able to draw the following conclusions: 1) There are two key properties of applications that determine their sensitivity to memory contention: their latency-sensitivity and their bandwidth-sensitivity; 2) Latency sensitive applications are likely to experience large slowdowns (up to 30%) before the off-chip memory bandwidth is saturated; 3) On the other hand, the performance of applications that are not latency sensitive is not significantly affected until the off-chip memory bandwidth becomes saturated; 4) Neither the baseline bandwidth nor performance is a good indicator of how sensitive an application is to off-chip memory contention.

References

[1] J. D. C. Little, "A proof for the queuing formula: $L = \lambda W$," *Operations Research*, vol. 9, 1961.

[2] D. Eklov, N. Nikoleris, D. Black-Schaffer, and E. Hagersten, "Design and evaluation of the bandwidth bandit," Tech. Rep. 2012-003, Department of Information Technology, Uppsala University, 2012.

[3] T. Dey, W. Wang, J. W. Davidson, and M. L. Soffa, "Characterizing multi-threaded applications based on shared-resource contention," in *Proc. of ISPASS*, 2011.

[4] L. Tang, J. Mars, N. Vachharajani, R. Hundt, and M. L. Soffa, "The impact of memory subsystem resource sharing on datacenter applications," in *Proc. of ISCA*, 2011.