Epoch Profiles: Microarchitecture-Based Application Analysis and Optimization

Trevor E. Carlson, *Ghent University*
Siddharth Nilakantan and Mark Hempstead, *Drexel University*
Wim Heirman, *Intel Corporation*

Abstract—The performance of data-intensive applications, when running on modern multi- and many-core processors, is largely determined by their memory access behavior. Its most important contributors are the frequency and latency of off-chip accesses and the extent to which long-latency memory accesses can be overlapped with useful computation or with each other.
In this paper we present two methods to better understand application and microarchitectural interactions. An *epoch profile* is an intuitive way to understand the relationships between three important characteristics: the on-chip cache size, the size of the reorder window of an out-of-order processor, and the frequency of processor stalls caused by long-latency, off-chip requests (epochs). By relating these three quantities one can more easily understand an application's memory reference behavior and thus significantly reduce the design space. While epoch profiles help to provide insight into the behavior of a single application, developing an understanding of a number of applications in the presence of area and core count constraints presents additional challenges. *Epoch-based microarchitectural analysis* is presented as a better way to understand the trade-offs for memory-bound applications in the presence of these physical constraints.
Through epoch profiling and optimization, one can significantly reduce the multidimensional design space for hardware/software optimization through the use of high-level model-driven techniques.

Index Terms—Microarchitecture analysis, memory-level parallelism, visualization

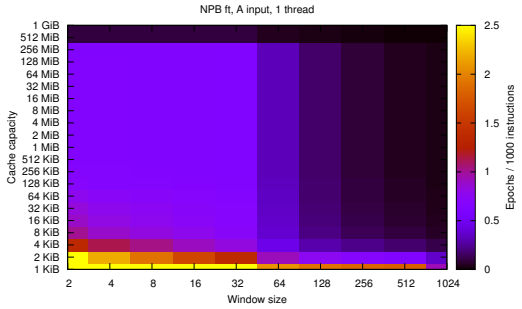

Fig. 1. This epoch profile plots the number of non-overlapped long-latency memory accesses (in epochs per 1,000 instructions) as a function of a processor's instruction window size and cache capacity. Epoch profiles show how long-latency effects impact overall performance.

1 Introduction

As Moore's Law continues to advance, we are seeing an increasing amount of computational power on a single chip. However, off-chip memory bandwidth is not growing accordingly, making applications increasingly bandwidth-constrained. Efficient use of the available on-chip cache capacity, thus minimizing pressure on off-chip bandwidth, is an important consideration for the performance of many applications, including HPC applications.

One key to improving application performance is the ability to extract as much memory-level parallelism (MLP) as possible. By exploiting MLP, the microprocessor can overlap off-chip requests instead of serializing them, which tends to improve application performance [1].

Although the raw MLP measurement of an application allows us to understand the amount of parallelism that exists (or can be extracted) over long time scales, this metric does not allow us to approximate application performance. The epoch model [1], on the other hand, directly relates to performance penalties observed by an out-of-order processor in response to off-chip accesses. Each epoch is composed of on-chip computation and cache accesses, followed by a potentially long period where the processor waits for a response from an off-chip data access. By counting the number of epochs that exist in an application, one can better understand the penalty caused by off-chip accesses for a particular hardware/software configuration. For memory-dominated applications, as the number of epochs in an application increases, so does the application runtime. Therefore, minimizing the epoch count across a selection of relevant microarchitectural parameters (such as cache and processor window size, or core count) can allow one to reduce the overall application runtime.

The goal of this work is to provide a method to better understand the closely-coupled hardware/software interactions that relate performance through bandwidth (and epochs). To do this, we abstract a microprocessor into two major performance-limiting components: the total on-chip cache capacity and the size of the reorder buffer, or window size, of the processor core. By comparing the number of epochs present when sweeping across these microarchitectural values, it becomes possible to determine which values provide the most benefit for a particular application. While completing such a study with detailed processor models is possible, our goal with this work is to do so without detailed timing simulation, but instead with microarchitecture analysis coupled with analytical modeling. This becomes even more important when varying architectural parameters not changed in this study, such as core frequency, or taking into account energy consumption, as cycle-level simulation across a large design space can take a significant amount of time.

Figure 1 shows an example epoch profile that takes into account each of these parameters. Considering a vertical slice of the graph, as the cache capacity (y-axis) increases past 16 KiB, the epoch frequency visible in the heat map decreases signifi-

. Manuscript submitted: 13-Feb-2014. Manuscript accepted: 23-Mar-2014. Final manuscript received: 31-Mar-2014.

cantly (changes from bright yellow and orange to dark purple) indicating that a cache size of 64 KiB allows for significant off-chip memory access reductions across small window sizes (x-axis). Cache capacities above 256 MiB provide another step in reduction of off-chip accesses across all window sizes. Along the horizontal axis is the size of the reorder buffer, which affects the amount of independent memory accesses that can be found. As the window size increases past 32 and 64 entries to 256, the off-chip access count drops significantly, indicating that there can be a significant application performance benefit when moving past 32 entries.

2 Background

2.1 MLP and the Epoch Model

Chou et al. [1] define both MLP and epoch modeling. MLP is the ability of a microprocessor to extract more than one outstanding off-chip memory access. As accessing off-chip memory takes significantly longer than performing a large number of local computations [1], parallel off-chip accesses reduce the total amount of time that a processor needs to stall waiting for long-latency load requests, improving performance.

The epoch model divides application execution into a number of time periods, or intervals, consisting of a period of active program execution (consisting of both instruction execution and local cache accesses), followed by a period of delay caused by a number of different factors, such as off-chip long-latency load requests blocking the reorder buffer (ROB) or serializing instructions. One can construct epochs by recording the time when these factors occur, and beginning a new epoch following each event. Due to the overlapping of events, like long-latency load accesses, each epoch will incur a single end-to-end latency for all overlapped requests. This differentiates epoch counts from miss rates because each epoch represents a known stall point for a processor core, while each miss can potentially be overlapped by the out-of-order core itself and is difficult to attribute to processor stalls without detailed simulation.

The epoch model effectively separates the off-chip CPI component from the rest of the CPI of an application. We can therefore approximate the off-chip CPI component as the number of epochs times the memory access latency (See Table 1). Higher epoch frequencies indicate larger memory penalties, and therefore a slower overall application.

2.2 Hardware Abstractions

One key component of the epoch profile is the size of the reorder buffer, or window size. Larger windows allow a processor to extract additional independent operations and memory accesses, reducing overall runtime.

In addition to the window size of the processor, the total amount of on-chip and relatively fast cache is the other major micro-architectural component under consideration. We abstract the amount of on-chip cache that is accessible by an application by monitoring the stack distance of memory accesses. By using stack distances, we emulate a single, fully-associative cache with LRU replacement. Accesses with a stack distance that is greater than the size of the cache will be long-latency misses. Hence, increasing cache capacity reduces the number of off-chip accesses required by the application.

3 Application Epoch Profiling

As described in Section 2.1, the epoch frequency provides a direct way to measure application performance when off-chip memory accesses dominate the time spent in the application. We have observed that application epoch frequency tends to decrease as window size or on-chip cache capacity increases. Nevertheless, we would like to better understand application characteristics and trends. Which combination of window size and cache capacity will benefit a particular application the most? When will application performance no longer improve as we increase core and cache resources? In this work we describe application analysis through epoch profiles as a way to more easily answer these questions. In addition to single-application epoch analysis, we also present a microarchitectural analysis based on epochs to aid in early microarchitectural design-space exploration.

3.1 Epoch Profiles

The application epoch profile is a method to determine inherent application performance characteristics with respect to cache size and the ability of the microarchitecture to extract MLP from the application. It consists of a heat map (See Figure 1) measuring the epochs per 1000 instructions (EPKI) that represent the frequency of long-latency, off-chip memory requests that stall the microprocessor. On the x-axis is the window size, and on the y-axis, the total capacity of the on-chip cache hierarchy of the processor. The benefit of measuring the epoch frequency instead of the MLP of an application is that it directly relates to application performance. The epoch profile, therefore, provides a method to compare the resulting performance of a number of different microarchitectural trade-offs.

3.2 Epoch-based Microarchitectural Analysis

While viewing a single application's epoch profile can be helpful to determine how it responds to a number of performance-critical microarchitectural changes, there is an additional need to understand how both single applications and classes of applications respond to the question: do we want additional cores or larger cache capacity for future processor designs?

For a given set of applications, there exists an optimal hardware-software configuration where an application achieves its highest performance. Unfortunately, there are a large number of parameters to consider that complicate this search. Naturally, as the cache size grows, so does its average latency. But, more complex relationships exist that require additional insight and knowledge about both the application as well as the interactions with the microarchitecture itself. The performance of bandwidth-bound applications on a particular microarchitecture, for example, depend both on the available bandwidth and frequency of off-chip memory accesses made by the application.

We identify three primary, but interdependent factors that contribute to determining aggregate application performance (which we define as the sum of the IPCs that contribute to useful work over all cores).

- LLC cache capacity. The total cache capacity affects the number of long-latency loads (off-chip accesses) seen.
- Number of cores and application threads. Thread count affects application data partitioning [4] directly, but higher core counts can provide a way to extract more application performance.
- Core window size. The window size of an application determines how many long-latency loads are overlapped, determining the number of epochs, and therefore the penalty caused by off-chip memory accesses (CPI_{mem}). It is this CPI component that will determine the access rate which allows us to compute the amount of requested off-chip bandwidth per thread.

Each of these three items are linked closely together. A larger LLC capacity allows for a larger on-chip working set, but

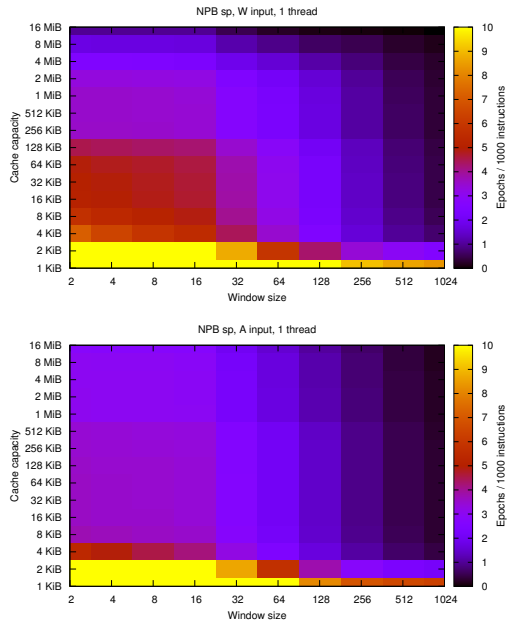

Fig. 2. Epoch profile comparison for the `W` (top) and `A` (bottom) input sets for the `sp` application.

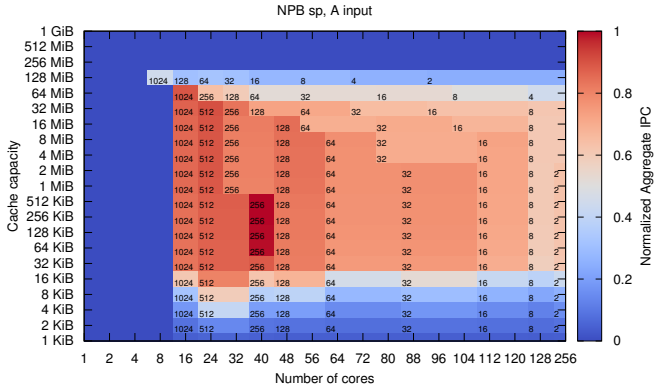

Fig. 3. Epoch microarchitecture analysis for `sp`, `A` input.

reduces either the number of cores or the window size of the cores available. The trade-off between window size and cores is a complex one, as larger windows (and therefore fewer cores) can potentially expose additional long-latency misses (reducing the number of epochs), but having fewer cores also reduces the amount of bandwidth needed to sustain non-bandwidth-limited performance.

To better understand these complex relationships, and to help microarchitects design next generation microprocessors, we have developed a model that uses the epoch as a performance measure to relate each of these primary factors to the other. For a given configuration (cache size, number of cores and application configuration), we can solve for the optimal window size as well as the total aggregate performance of the application. We partition the computation of maximum performance into bandwidth and non-bandwidth limited categories, and approximate performance for these two categories as defined in Table 1.

4 Experimental Setup

For this exploration, we provide a number of abstractions to allow us to relate the three primary variables to each other over a large sample space. Table 1 describes the constants,

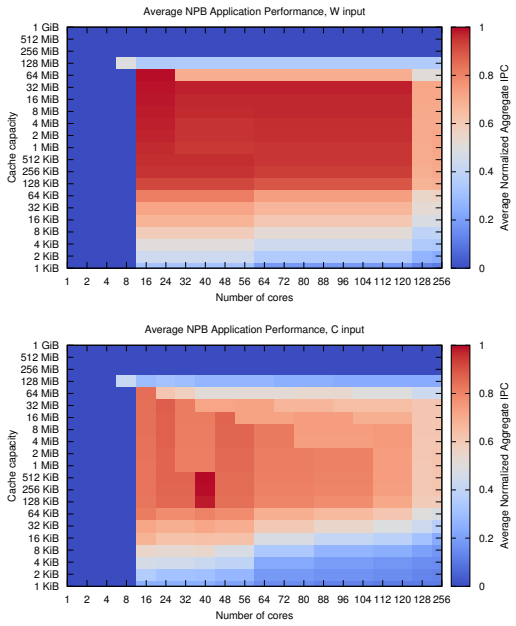

Fig. 4. Epoch microarchitecture analysis averages for `W` (top) and `C` (bottom) inputs of `npb`.

application-dependent variables and relationships used in this study. Base application IPCs are approximated with a power law: 1.0 for 32-entry window and up to 5.0 for a 1024 entry window.

In our experiments, we assume that the processor window size is the limiting factor for being able to extract MLP. Therefore we do not model the microprocessor's issue queues or load and store queues as we would like to determine the upper bound of how much these windows can extract additional MLP. We currently do not model a number of effects on epochs, such as: serializing instructions, instruction fetch misses, unresolvable branch mispredictions, memory-based dependencies and prefetching. These effects can be added to the model to provide additional fidelity.

We developed a Pintool [6] that monitors application memory references and register dependencies to track the epoch frequency. The resulting frequency is saved for each window size and cache capacity of interest and plotted as a heat map

TABLE 1
Variables and relationships used for modeling.

Variables and constants			
Core count	N	Total area (mm^2)	A_{proc}=700
Window size	W	Frequency (GHz)	$freq$=1
LLC capacity (KiB)	C_{LLC}	Memory BW (GB/s)	BW=32
App insn count	I	Memory latency (ns)	t_{mem}=45
App epoch count	e	Line size (KiB)	L=64
App mem accesses	m		

Relationships	
Core area (mm^2)	$A_{core}=2.0609 \cdot W^{0.3851}$
LLC area (mm^2)	$A_{LLC}=C_{LLC} \cdot 3.2305 \times 10^{-3}$
Processor area (mm^2)	$A_{proc}=A_{core}+A_{LLC}$
Base CPI	$CPI_{base}=1/2 \cdot \sqrt{W/160}$
Memory CPI	$CPI_{mem}=e \cdot freq/t_{mem}$
Core CPI	$CPI_{core}=CPI_{base}+CPI_{mem}$
Perf (compute-bound)	$IPC_{agg}=N \cdot \frac{1}{CPI_{core}}$
Perf (BW-bound)	$IPC_{agg}=BW \cdot N \cdot \frac{I}{m \cdot L \cdot freq}$

to form the epoch profile. Although the runtime overhead of this pintool is approximately 100 to 200× slowdown compared to native execution, a large number of configurations are being analyzed simultaneously (more than 200 for the experiments in this paper). An approximate calculation of the stack distance can reduce the tool runtime [2].

5 Results

5.1 Application Epoch Profile

Figure 2 plots the epoch profile for the `sp` application from the NAS Parallel Benchmarks suite [5], both for the class W input (top) and the larger class A input (bottom). Reading the epoch profiles in vertical slices, we see a result that is similar to a miss rate curve. Starting at the bottom of the graph, cache capacity is very small and most memory accesses result in off-chip memory requests which end an epoch of in-core execution. Moving to the top, representing ever larger on-chip cache capacities, sudden drops in miss rate become apparent. At these points, the cache had just became large enough to hold a new working set of the application. For both input sets, this happens at 4 KiB which is the size of thread-local stacks and data structures. Further drops are input-set specific. The class W input has a working set of the application's inner loop of less than 256 KiB, while at 8 MiB the complete input set fits in the cache leaving only very few (cold) cache misses. The larger A input has an inner loop working set of 512 KiB while the complete data set is does not fit the largest cache capacity considered.

Reading the graphs from left to right, we can gauge the impact of improved MLP extraction by larger out-of-order cores. Starting from the left, in-order or moderately out-of-order cores are not able to extract any MLP and hence serialize all off-chip accesses. Once the window size exceeds 32 instructions, however, a continuous decrease in the measured EPKI can be observed. For this application, the inner loop contains 146 instructions with three to four independent off-chip memory accesses, while subsequent loop iterations are independent of each other and therefore expose additional memory-level parallelism. Moving further to the right of the graph, each doubling of the window size also doubles the number of independent loop iterations that fit inside of it, further increasing MLP and reducing EPKI.

5.2 Epoch-based Microarchitectural Analysis

Figure 3 is an example the epoch-based microarchitectural analysis for the `sp` application. This figure shows the performance (in normalized aggregate application IPC) on the z-axis of the heat map for a number of configuration parameters. Across the core count (x-axis) and cache capacity (y-axis) options, we also display the largest window size (in text) for the optimal configurations. The configurations are chosen based on the best aggregate performance across all window sizes for that core count. Non-optimal configuration options do not list the window size, but use the results from the next smallest configuration for ease of viewing. Here we see, counterintuitively, that the cache capacities above 1 MiB do not perform the best when considering overall performance for our configuration parameters. The reason for this is because `sp` with the `A` input set is a bandwidth-bound application that is able to take advantage of a large number of cores (40) with a large window size (256 entries) to reduce the number of epochs sufficiently to exploit the bandwidth available (not shown). Fewer cores (even with a larger window size) are unable to reach the same performance. Adding more cores (with a smaller window size) is also unable to achieve the same level of performance because of bandwidth limitations. Finally, cache capacity is important for stack accesses which occur frequently, as memory capacities below 64 KiB reduce performance.

Figure 4 shows the average performance (normalized aggregate IPC) across all epoch-based microarchitecture analysis runs for applications found in the NAS Parallel Benchmarks. Here, the comparison between non-bandwidth-limited (top, `W`) and bandwidth limited (bottom, `C`) input sets becomes clear. Over a large range of parameters, the `W` input set shows close to maximum performance for many options because the working set fits into the available on-chip cache. Nevertheless, we can see with the `C` input set (and also with `A`, not shown) that there is a clear winner where MLP is sufficiently exposed with the relatively large window size (256 entries) to allow for sufficient forward progress even with a limited number of cores (40).

6 Conclusion

Off-chip bandwidth has become the primary factor for determining next-generation application performance. In this work, we introduce two new methods to better understand how the interaction between the microarchitecture and the application affect performance through the use of epochs, or out-of-order processor stalls due to long-latency off-chip accesses. We first describe the epoch profile, a unique, visual way to understand how much of an effect both larger cache capacity and growing window sizes have on modern applications. We also introduce a fast method of early microarchitectural design space exploration through epoch analysis.

While this work focuses on one main cause of epochs, namely long-latency loads stalling the ROB, taking into account additional events such as front-end stalls and prefetching, for example, could provide for better application understanding. Many of the larger input sizes of the NAS Parallel Benchmarks tend to be bandwidth bound, and therefore provide a good initial analysis target. Nevertheless, extending this work to evaluate additional benchmarks, including those outside of traditional data-parallel, HPC-style applications, is a next step required for evaluating the general applicability of this work.

As the microarchitectural design space grows, with parameters such as core clock frequencies and off-chip bandwidth, faster, more insightful means of analysis become increasingly important. Epoch-driven analysis provides an early, high-level application understanding, allowing the microarchitect to more easily compare future architectural options, in a fast, visual way before moving to detailed cycle-level simulations.

References

[1] Y. Chou, B. Fahs, and S. Abraham, "Microarchitecture optimizations for exploiting memory-level parallelism," in *ISCA*, Jun. 2004, pp. 76–87.

[2] C. Ding and Y. Zhong, "Predicting whole-program locality through reuse distance analysis," in *PLDI*, May 2003, pp. 245–257.

[3] S. Eyerman, L. Eeckhout, T. Karkhanis, and J. E. Smith, "A mechanistic performance model for superscalar out-of-order processors," *ACM TOCS*, vol. 27, no. 2, pp. 42–53, May 2009.

[4] W. Heirman, T. E. Carlson, K. Van Craeynest, I. Hur, A. Jaleel, and L. Eeckhout, "Undersubscribed threading on clustered cache architectures," in *HPCA*, Feb. 2014.

[5] H. Jin, M. Frumkin, and J. Yan, "The OpenMP implementation of NAS Parallel Benchmarks and its performance," NASA Ames Research Center, Tech. Rep., Oct. 1999.

[6] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: Building customized program analysis tools with dynamic instrumentation," in *PLDI*, Jun. 2005, pp. 190–200.