

A CASE FOR SPECIALIZED PROCESSORS FOR SCALE-OUT WORKLOADS

EMERGING SCALE-OUT WORKLOADS NEED EXTENSIVE COMPUTATIONAL RESOURCES, BUT DATACENTERS USING MODERN SERVER HARDWARE FACE PHYSICAL CONSTRAINTS. IN THIS ARTICLE, THE AUTHORS SHOW THAT MODERN SERVER PROCESSORS ARE HIGHLY INEFFICIENT FOR RUNNING CLOUD WORKLOADS. THEY INVESTIGATE THE MICROARCHITECTURAL BEHAVIOR OF SCALE-OUT WORKLOADS AND PRESENT OPPORTUNITIES TO ENABLE SPECIALIZED PROCESSOR DESIGNS THAT CLOSELY MATCH THE NEEDS OF THE CLOUD.

..... Cloud computing is emerging as a dominant computing platform for delivering scalable online services to a global client base. Today's popular online services, such as web search, social networks, and video sharing, are all hosted in large scale-out datacenters. With the industry rapidly expanding, service providers are building new datacenters, augmenting the existing infrastructure to meet the increasing demand. However, while demand for cloud infrastructure continues to grow, the semiconductor manufacturing industry has reached the physical limits of voltage scaling,^{1,2} no longer able to reduce power consumption or increase power density in new chips. Physical constraints have therefore become the dominant limiting factor, because the size and power demands of larger datacenters cannot be met.

Although major design changes are being introduced at the board and chassis levels of new cloud servers, the processors used in modern servers were originally created for desktops and are not designed to efficiently run scale-out workloads. Processor vendors use the same underlying microarchitecture

for servers and for the general-purpose market, leading to extreme inefficiency in today's datacenters. Moreover, both general-purpose and traditional server processor designs follow a trajectory that benefits scale-up workloads, a trend that was established for desktop processors long before the emergence of scale-out workloads.

In this article, based on our paper for the 17th International Conference on Architectural Support for Programming Languages and Operating Systems,³ we observe that scale-out workloads share many inherent characteristics that place them into a workload class distinct from desktop, parallel, and traditional server workloads. We perform a detailed microarchitectural study of a range of scale-out workloads, finding a large mismatch between the demands of the scale-out workloads and today's predominant processor microarchitecture. We observe significant overprovisioning of the memory hierarchy and core microarchitectural resources for the scale-out workloads.

We use performance counters to study the behavior of scale-out workloads running on

Michael Ferdman
Stony Brook University

Almutaz Adileh
Ghent University

Onur Kocberber
Stavros Volos
Mohammad Alisafaei
Djordje Jevdjic

Cansu Kaynak
Adrian Daniel Popescu
Anastasia Ailamaki

Babak Falsafi
École Polytechnique Fédérale
de Lausanne

modern server processors. On the basis of our analysis, we demonstrate the following:

- Scale-out workloads suffer from high instruction-cache miss rates. Instruction caches and associated next-line prefetchers found in modern processors are inadequate for scale-out workloads.
- Instruction-level parallelism (ILP) and memory-level parallelism (MLP) in scale-out workloads are low. Modern aggressive out-of-order cores are excessively complex, consuming power and on-chip area without providing performance benefits to scale-out workloads.
- Data working sets of scale-out workloads considerably exceed the capacity of on-chip caches. Processor real estate and power are misspent on large last-level caches that do not contribute to improved scale-out workload performance.
- On- and off-chip bandwidth requirements of scale-out workloads are low. Scale-out workloads see no benefit from fine-grained coherence and excessive memory and core-to-core communication bandwidth.

Continuing the current processor trends will further widen the mismatch between scale-out workloads and server processors. Conversely, the characteristics of scale-out workloads can be effectively leveraged to specialize processors for these workloads in order to gain area and energy efficiency in future servers. An example of such a specialized processor design that matches the needs of scale-out workloads is Scale-Out Processor,⁴ which has been shown to improve the system throughput and the overall datacenter cost efficiency by almost an order of magnitude.⁵

Modern cores and scale-out workloads

Today's datacenters are built around conventional desktop processors whose architecture was designed for a broad market. The dominant processor architecture has closely followed the technology trends, improving single-thread performance with each processor generation by using the increased clock

speeds and “free” (in area and power) transistors provided by progress in semiconductor manufacturing. Although Dennard scaling has stopped,^{1,2,6,7} with both clock frequency and transistor counts becoming limited by power, processor architects have continued to spend resources on improving single-thread performance for a broad range of applications at the expense of area and power efficiency.

In this article, we study a set of applications that dominate today's cloud infrastructure. We examined a selection of Internet services on the basis of their popularity. For each popular service, we analyzed the class of application software used by major providers to offer these services, either on their own cloud infrastructure or on a cloud infrastructure leased from a third party. Overall, we found that scale-out workloads have similar characteristics. All applications we examined

- operate on large data sets that are distributed across a large number of machines, typically into memory-resident shards;
- serve large numbers of completely independent requests that do not share any state;
- have application software designed specifically for the cloud infrastructure, where unreliable machines may come and go; and
- use connectivity only for high-level task management and coordination.

Specifically, we identified and studied the following workloads: an in-memory object cache (Data Caching); a NoSQL persistent data store (Data Serving); data filtering, transformation, and analysis (MapReduce); a video-streaming service (Media Streaming); a large-scale irregular engineering computation (SAT Solver); a dynamic Web 2.0 service (Web Frontend); and an online search engine node (Web Search). To highlight the differences between scale-out workloads and traditional workloads, we evaluated cloud workloads alongside the following traditional benchmark suites: Parsec 2.1 Parallel workloads, SPEC CPU2006 desktop and engineering workloads, SPECweb09 traditional web services, TPC-C traditional transaction processing workload, TPC-E modern transaction

Table 1. Architectural parameters.

Component	Details
Processor	32-nm Intel Xeon X5670, operating at 2.93 GHz
Chip multiprocessor width	Six out-of-order cores
Core width	Four-wide issue and retire
Reorder buffer	128 entries
Load-store queue	48/32 entries
Reservation stations	36 entries
Level-1 caches	32 Kbytes instruction and 32 Kbytes data, four-cycle access latency
Level-2 cache	256 Kbytes per core, six-cycle access latency
Last-level cache (Level-3 cache)	12 Mbytes, 29-cycle access latency
Memory	24 Gbytes, three double-data-rate three (DDR3) channels, delivering up to 32 Gbytes/second

processing workload, and MySQL Web 2.0 back-end database.

Methodology

We conducted our study on a PowerEdge M1000e enclosure with two Intel X5670 processors and 24 Gbytes of RAM in each blade, using Intel VTune to analyze the system’s microarchitectural behavior. Each Intel X5670 processor includes six aggressive out-of-order processor cores with a three-level cache hierarchy: the L1 and L2 caches are private to each core; the last-level cache (LLC)—the L3 cache—is shared among all cores. Each core includes several simple stride and stream prefetchers, labeled as “adjacent-line,” “HW prefetcher,” and “DCU streamer” in the processor documentation and system BIOS settings. The blades use high-performance Broadcom server network interface controllers (NICs) with drivers that support multiple transmit queues and receive-side scaling. The NICs are connected by a built-in M6220 switch. For bandwidth-intensive benchmarks, 2-Gbit NICs are used in each blade.

Table 1 summarizes the blades’ key architectural parameters. We limited all workload configurations to four cores, tuning the workloads to achieve high utilization of the cores (or hardware threads, in the case of the SMT experiments), while maintaining the workload quality-of-service requirements. To ensure that all application and operating

system software runs on the cores under test, we disabled all unused cores using the available operating system mechanisms.

Results

We explore the microarchitectural behavior of scale-out workloads by examining the commit-time execution breakdown in Figure 1. We classify each cycle of execution as Committing if at least one instruction was committed during that cycle, or as Stalled otherwise. We note that computing a breakdown of the execution-time stall components of superscalar out-of-order processors cannot be performed precisely because of overlapped work in the pipeline. We therefore present execution-time breakdown results based on the performance counters that have no overlap. Alongside the breakdown, we show the Memory cycles, which approximate time spent on long-latency memory accesses, but potentially partially overlap with instruction commits.

The execution-time breakdown of scale-out workloads is dominated by stalls in both the application code and operating system. Notably, most of the stalls in scale-out workloads arise because of long-latency memory accesses. This behavior is in contrast to the CPU-intensive desktop (SPEC2006) and parallel (Parsec) benchmarks, which stall execution significantly less than 50 percent of the cycles and experience only a fraction

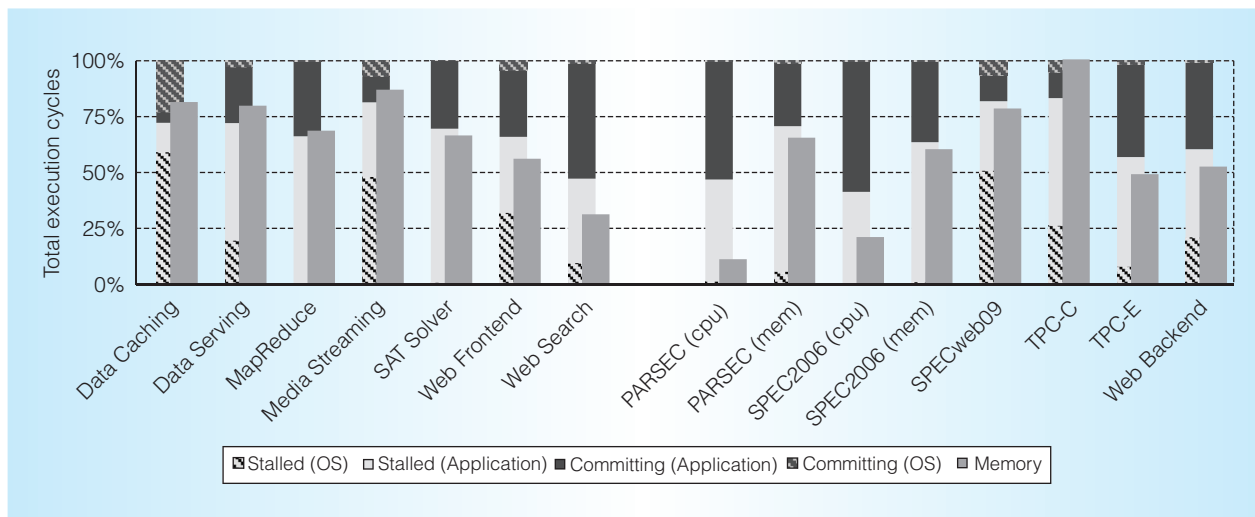


Figure 1. Execution-time breakdown and memory cycles of scale-out workloads (left) and traditional benchmarks (right). Execution time is further broken down into its application and operating system components.

of the stalls due to memory accesses. Furthermore, although the execution-time breakdown of some scale-out workloads (such as MapReduce and SAT Solver) appears similar to the memory-intensive Parsec and SPEC2006 benchmarks, the nature of these workloads' stalls is different. Unlike the scale-out workloads, many Parsec and SPEC2006 applications frequently stall because of pipeline flushes after wrong-path instructions, with much of the memory access time not on the critical path of execution.

Scale-out workloads show memory system behavior that more closely matches traditional online transaction processing workloads (TPC-C, TPC-E, and Web Backend). However, we observe that scale-out workloads differ considerably from traditional online transaction processing (TPC-C), which spends more than 80 percent of the time stalled, owing to dependent memory accesses. We find that scale-out workloads are most similar to the more recent transaction processing benchmarks (TPC-E) that use more complex data schemas or perform more complex queries than traditional transaction processing. We also observe that a traditional enterprise web workload (SPECweb09) behaves differently than the Web Frontend workload, representative of modern scale-out configurations. Although the traditional web workload is dominated by serving static files and a few dynamic scripts, modern scalable web

workloads like Web Frontend handle a much higher fraction of dynamic requests, leading to higher core utilization and less OS involvement.

Although the behavior across scale-out workloads is similar, the class of scale-out workloads as a whole differs significantly from other workloads. Processor architectures optimized for desktop and parallel applications are not optimized for scale-out workloads that spend most of their time waiting for cache misses, resulting in a clear microarchitectural mismatch. At the same time, architectures designed for workloads that perform only trivial computation and spend all of their time waiting on memory (such as SPECweb09 and TPC-C) also cannot cater to scale-out workloads.

Front-end inefficiencies

There are three major front-end inefficiencies:

- Cores are idle because of high instruction-cache miss rates.
- L2 caches increase average instruction-fetch latency.
- Excessive LLC capacity leads to long instruction-fetch latency.

Instruction-fetch stalls play a critical role in processor performance by preventing the core from making forward progress because of a lack of instructions to execute. Front-end

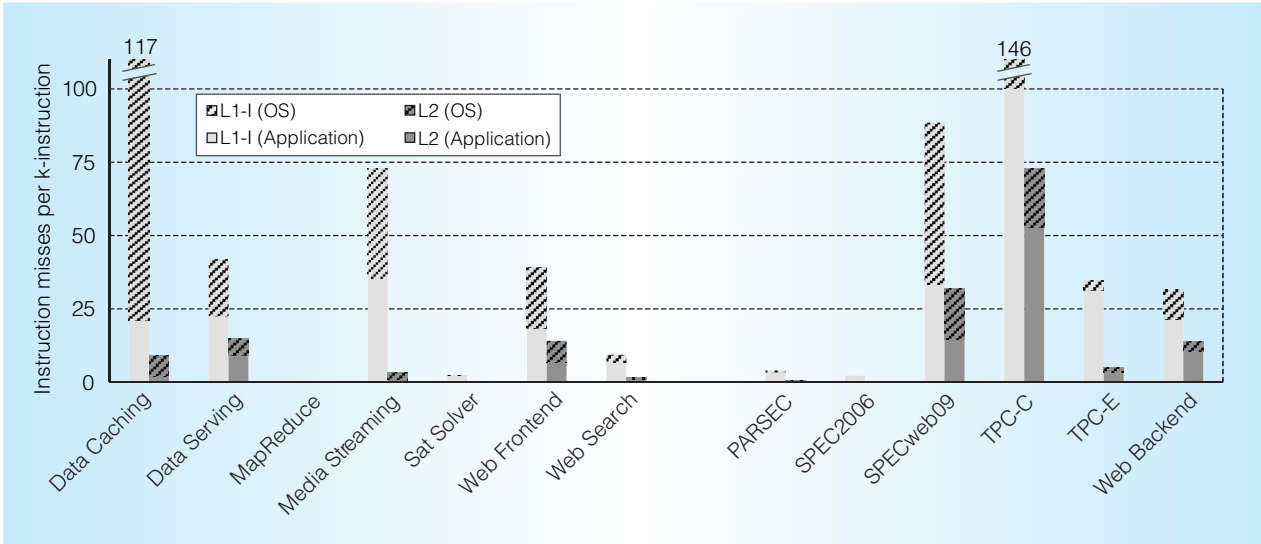


Figure 2. L1 and L2 instruction miss rates for scale-out workloads (left) and traditional benchmarks (right). The miss rate is broken down into its application and operating system components.

stalls serve as a fundamental source of inefficiency for both area and power, because the core real estate and power consumption are entirely wasted for the cycles that the front end spends fetching instructions.

Figure 2 presents the instruction miss rates of the L1 instruction cache and the L2 cache. In contrast to desktop and parallel benchmarks, the instruction working sets of many scale-out workloads considerably exceed the capacity of the L1 instruction cache, resembling the instruction-cache behavior of traditional server workloads. Moreover, the instruction working sets of most scale-out workloads also exceed the L2 cache capacity, where even relatively infrequent instruction misses incur considerable performance penalties. We find that modern processor architectures can't tolerate the latency of the L1 instruction cache's misses, avoiding front-end stalls only for applications whose entire instruction working set fits into the L1 cache. Furthermore, the high L2 instruction miss rates indicate that the L1 instruction cache's capacity experiences a significant shortfall and can't be mitigated by the addition of a modestly sized L2 cache.

The disparity between the needs of the scale-out workloads and the processor architecture are apparent in the instruction-fetch path. Although exposed instruction-

fetch stalls serve as a key source of inefficiency under any circumstances, the instruction-fetch path of modern processors actually exacerbates the problem. The L2 cache experiences high instruction miss rates, increasing the average fetch latency of the missing fetch requests by placing an additional intermediate lookup structure on the path to retrieve instruction blocks from the LLC. Moreover, the entire instruction working set of any scale-out workload is considerably smaller than the LLC capacity. However, because the LLC is a large cache with a high uniform access latency, it contributes an unnecessarily large instruction-fetch penalty (29 cycles to access the 12-Mbyte cache).

To improve efficiency and reduce front-end stalls, processors built for scale-out workloads must bring instructions closer to the cores. Rather than relying on a deep hierarchy of caches, a partitioned organization that replicates instructions and makes them available close to the requesting cores⁸ is likely to considerably reduce front-end stalls. To effectively use the on-chip real estate, the system would need to share the partitioned instruction caches among multiple cores, striking a balance between the die area dedicated to replicating instruction blocks and the latency of accessing those blocks from the closest cores.

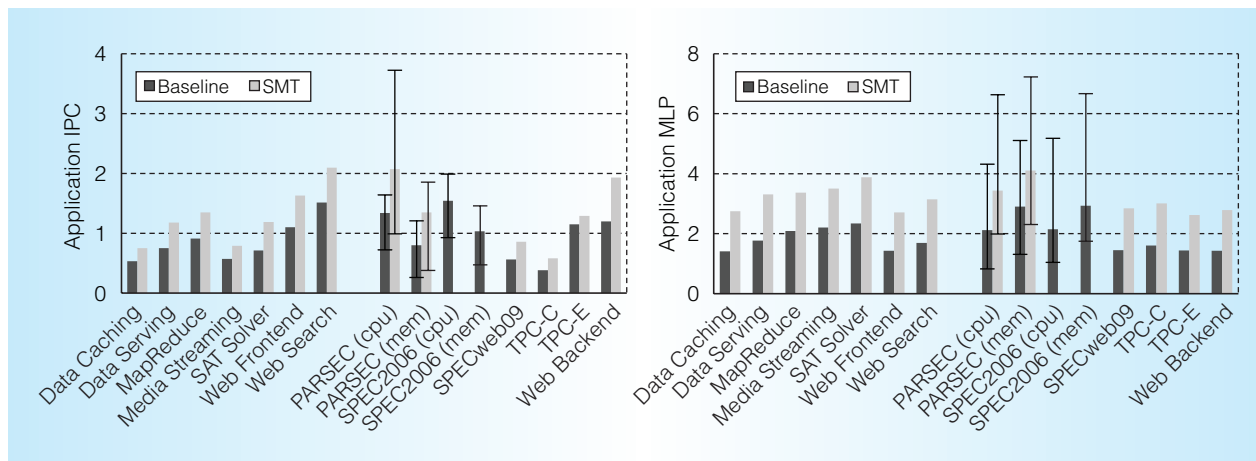


Figure 3. The instructions per cycle (IPC) and memory-level parallelism (MLP) of a simultaneous multithreading (SMT) enabled core. Application IPC for systems with and without SMT out of a maximum IPC of 4 (a). MLP for systems with and without SMT (b). Range bars indicate the minimum and maximum of the corresponding group.

Furthermore, although modern processors include next-line prefetchers, high instruction-cache miss rates and significant front-end stalls indicate that the prefetchers are ineffective for scale-out workloads. Scale-out workloads are written in high-level languages, use third-party libraries, and execute operating system code, exhibiting complex nonsequential access patterns that are not captured by simple next-line prefetchers. Including instruction prefetchers that predict these complex patterns is likely to improve overall processor efficiency by eliminating wasted cycles due to front-end stalls.

Core inefficiencies

There are two major core inefficiencies:

- Low ILP precludes effectively using the full core width.
- The reorder buffer (ROB) and the load-store queue (LSQ) are underutilized because of low MLP.

Modern processors execute instructions out of order to enable simultaneous execution of multiple independent instructions per cycle (IPC). Additionally, out-of-order execution elides stalls due to memory accesses by executing independent instructions that follow a memory reference while the long-latency cache access is in progress. Modern processors support up to 128-instruction windows, with the width of the processor dictating the number of instructions that

can simultaneously execute in one cycle. In addition to exploiting ILP, large instruction windows can exploit MLP by finding independent memory accesses within the instruction window and performing the memory accesses in parallel. The latency of LLC hits and off-chip memory accesses cannot be hidden by out-of-order execution; achieving high MLP is therefore key to achieving high core utilization by reducing the data access latency.

The processors we study use four-wide cores that can decode, issue, execute, and commit up to four instructions on each cycle. However, in practice, ILP is limited by dependencies. The Baseline bars in Figure 3a show the average number of instructions committed per cycle when running on an aggressive four-wide out-of-order core. Despite the abundant availability of core resources and functional units, scale-out workloads achieve a modest application IPC, typically in the range of 0.6 (Data Caching and Media Streaming) to 1.1 (Web Frontend). Although there exist workloads that can benefit from wide cores, with some CPU-intensive Parsec and SPEC2006 applications reaching an IPC of 2.0 (indicated by the range bars in the figure), using wide processors for scale-out applications does not yield a significant benefit.

Modern processors have 32-entry or larger load-store queues, enabling many memory-

reference instructions in the 128-instruction window. However, just as instruction dependencies limit ILP, address dependencies limit MLP. The Baseline bars in Figure 3b present the MLP, ranging from 1.4 (Web Frontend) to 2.3 (SAT Solver) for the scale-out workloads. These results indicate that the memory accesses in scale-out workloads are replete with complex dependencies, limiting the MLP that can be found by modern aggressive processors. We again note that while desktop and parallel applications can use high-MLP support, with some Parsec and SPEC2006 applications having an MLP up to 5.0, support for high MLP is not useful for scale-out applications. However, we find that scale-out workloads generally exhibit higher MLP than traditional server workloads. Noting that such characteristics lend themselves well to multithreaded cores, we examine the IPC and MLP of an SMT-enabled core in Figure 3. As expected, the MLP found and exploited by the cores when two independent application threads run on each core concurrently nearly doubles compared to the system without SMT. Unlike traditional database server workloads that contain many inter-thread dependencies and locks, the independent nature of threads in scale-out workloads enables them to observe considerable performance benefits from SMT, with 39 to 69 percent improvements in IPC.

Support for four-wide out-of-order execution with a 128-instruction window and up to 48 outstanding memory requests requires multiple-branch prediction, numerous arithmetic logic units (ALUs), forwarding paths, many-ported register banks, large instruction scheduler, highly associative ROB and LSQ, and many other complex on-chip structures. The complexity of the cores limits core count, leading to chip designs with several cores that consume half the available on-chip real estate and dissipate the vast majority of the chip's dynamic power budget. However, our results indicate that scale-out workloads exhibit low ILP and MLP, deriving benefit only from a small degree of out-of-order execution. As a result, the nature of scale-out workloads cannot effectively utilize the available core resources. Both the die area and the energy are wasted, leading to data-center inefficiency.

The nature of scale-out workloads makes them ideal candidates to exploit multi-threaded multicore architectures. Modern mainstream processors offer excessively complex cores, resulting in inefficiency through resource waste. At the same time, our results indicate that niche processors offer excessively simple (for example, in-order) cores that cannot leverage the available ILP and MLP in scale-out workloads. We find that scale-out workloads match well with architectures offering multiple independent threads per core with a modest degree of superscalar out-of-order execution and support for several simultaneously outstanding memory accesses. For example, rather than implementing SMT on a four-way core, we could use two independent two-way cores, which would consume fewer resources while achieving higher aggregate performance. Furthermore, each narrower core would not require a large instruction window, reducing the per-core area and power consumption compared to modern processors and enabling higher computational density by integrating more cores per chip.

Data-access inefficiencies

There are two major data-access inefficiencies:

- Large LLC consumes area, but does not improve performance.
- Simple data prefetchers are ineffective.

More than half of commodity processor die area is dedicated to the memory system. Modern processors feature a three-level cache hierarchy, where the LLC is a large-capacity cache shared among all cores. To enable high-bandwidth data fetch, each core can have up to 16 L2 cache misses in flight. The high-bandwidth on-chip interconnect enables cache-coherent communication between the cores. To mitigate the capacity and latency gap between the L2 caches and the LLC, each L2 cache is equipped with prefetchers that can issue prefetch requests into the LLC and off-chip memory. Multiple DDR3 memory channels provide high-bandwidth access to off-chip memory.

The LLC is the largest on-chip structure; its cache capacity has been increasing with each processor generation, thanks to

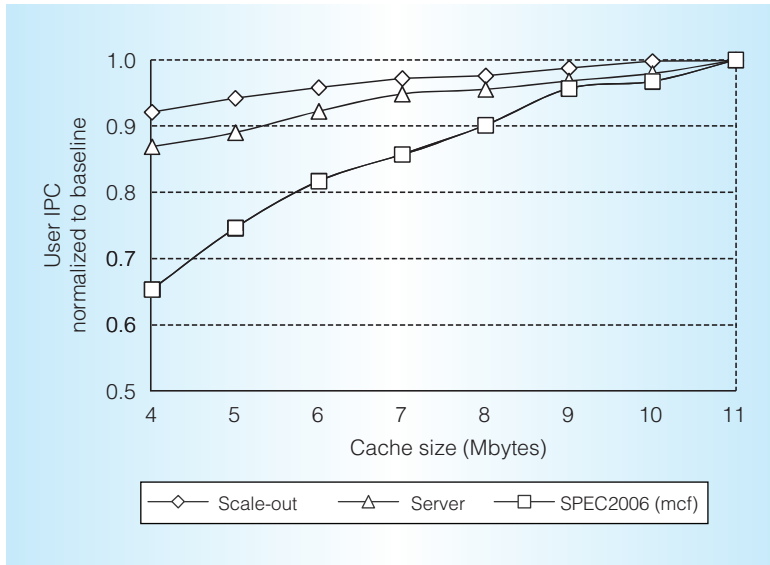


Figure 4. Performance sensitivity to the last-level cache (LLC) capacity. Relatively small average performance degradation due to reduced cache capacity is shown for the scale-out and server workloads, in contrast to some traditional applications (such as mcf).

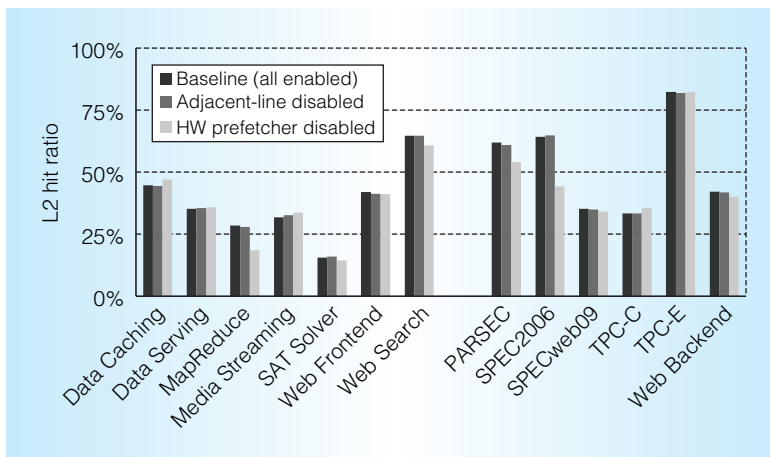


Figure 5. L2 hit ratios of a system with enabled and disabled adjacent-line and HW prefetchers. Unlike for Parsec and SPEC2006 applications, minimal performance difference is observed for the scale-out and server workloads.

semiconductor manufacturing improvements. We investigate the utility of growing the LLC capacity for scale-out workloads in Figure 4 through a cache sensitivity analysis by dedicating two cores to cache-polluting threads. The polluter threads traverse arrays of predetermined size in a pseudorandom sequence, ensuring that all accesses miss in the upper-level caches and reach the LLC. We use performance counters to confirm that

the polluter threads achieve nearly a 100 percent hit ratio in the LLC, effectively reducing the cache capacity available for the workload running on the remaining cores of the same processor.

We plot the average system performance of scale-out workloads as a function of the LLC capacity, normalized to a baseline system with a 12-Mbyte LLC. Unlike in the memory-intensive desktop applications (such as SPEC2006 mcf), we find minimal performance sensitivity to LLC size above 4 to 6 Mbytes in scale-out and traditional server workloads. The LLC captures the instruction working sets of scale-out workloads, which are less than 2 Mbytes. Beyond this point, small shared supporting structures may consume another 1 to 2 Mbytes. Because scale-out workloads operate on massive datasets and service a large number of concurrent requests, both the dataset and the per-client data are orders of magnitude larger than the available on-chip cache capacity. As a result, an LLC that captures the instruction working set and minor supporting data structures achieves nearly the same performance as an LLC with double or triple the capacity.

In addition to leveraging MLP to overlap demand requests from the processor core, modern processors use prefetching to speculatively increase MLP. Prefetching has been shown effective at reducing cache miss rates by predicting block addresses that will be referenced in the future and bringing these blocks into the cache prior to the processor's demand, thereby hiding the access latency. In Figure 5, we present the hit ratios of the L2 cache when all available prefetchers are enabled (Baseline), as well as the hit ratios after disabling the prefetchers. We observe a noticeable degradation of the L2 hit ratios of many desktop and parallel applications when the adjacent-line prefetcher and L2 hardware prefetcher are disabled. In contrast, only one of the scale-out workloads (MapReduce) significantly benefits from these prefetchers, with the majority of the workloads experiencing negligible changes in the cache hit rate. Moreover, similar to traditional server workloads (TPC-C), disabling the prefetchers results in an increase in the hit ratio for some scale-out workloads (Data Caching, Media Streaming, and SAT Solver). Finally, we note

that the DCU streamer (not shown) provides no benefit to scale-out workloads, and in some cases marginally increases the L2 miss rate because it pollutes the cache with unnecessary blocks.

Our results show that the on-chip resources devoted to the LLC are one of the key limiters of scale-out application computational density in modern processors. For traditional workloads, increasing the LLC capacity captures the working set of a broader range of applications, contributing to improved performance, owing to a reduction in average memory latency for those applications. However, because the LLC capacity already exceeds the scale-out application requirements by 2 to 3 times, whereas the next working set exceeds any possible SRAM cache capacity, the majority of the die area and power currently dedicated to the LLC is wasted. Moreover, prior research has shown that increases in the LLC capacity that do not capture a working set lead to an overall performance degradation; LLC access latency is high due to its large capacity, not only wasting on-chip resources, but also penalizing all L2 cache misses by slowing down LLC hits and delaying off-chip accesses.

Although modern processors grossly over-provision the memory system, we can improve datacenter efficiency by matching the processor design to the needs of the scale-out workloads. Whereas modern processors dedicate approximately half of the die area to the LLC, scale-out workloads would likely benefit from a different balance. A two-level cache hierarchy with a modestly sized LLC that makes a special provision for caching instruction blocks would benefit performance. The reduced LLC capacity along with the removal of the ineffective L2 cache would offer access-latency benefits while also freeing up die area and power. The die area and power can be applied toward improving computational density and efficiency by adding more hardware contexts and more advanced prefetchers. Additional hardware contexts (more threads per core and more cores) should linearly increase application parallelism, and more advanced correlating data prefetchers could accurately prefetch complex access data patterns and increase the performance of all cores.

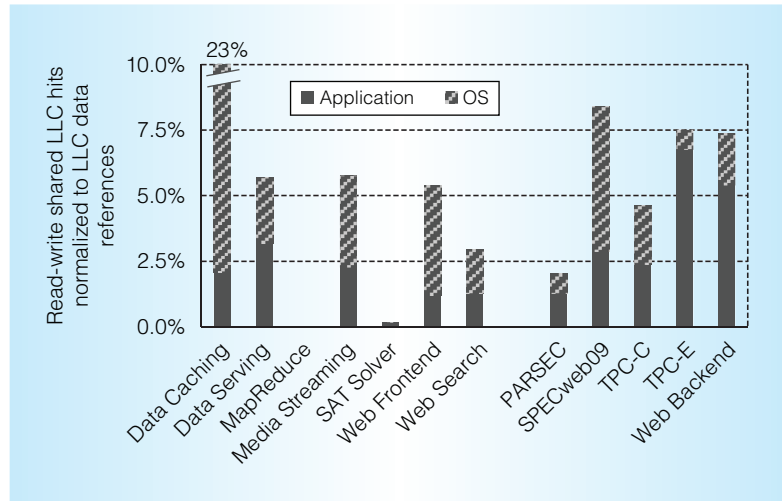


Figure 6. Percentage of LLC data references accessing cache blocks modified by a remote core. In scale-out workloads, the majority of the remotely accessed cache blocks are from the operating system code.

Bandwidth inefficiencies

The major bandwidth inefficiencies are

- Lack of data sharing deprecates coherence and connectivity.
- Off-chip bandwidth exceeds needs by an order of magnitude.

Increasing core counts have brought parallel programming into the mainstream, highlighting the need for fast and high-bandwidth inter-core communication. Multi-threaded applications comprise a collection of threads that work in tandem to scale up the application performance. To enable effective scale-up, each subsequent generation of processors offers a larger core count and improves the on-chip connectivity to support faster and higher-bandwidth core-to-core communication.

We investigate the utility of the on-chip interconnect for scale-out workloads in Figure 6. To measure the frequency of read-write sharing, we execute the workloads on cores split across two physical processors in separate sockets. When reading a recently modified block, this configuration forces accesses to actively shared read-write blocks to appear as off-chip accesses to a remote processor cache. We plot the fraction of L2 misses that access data most recently written by another thread running on a remote core, breaking down each bar into Application and

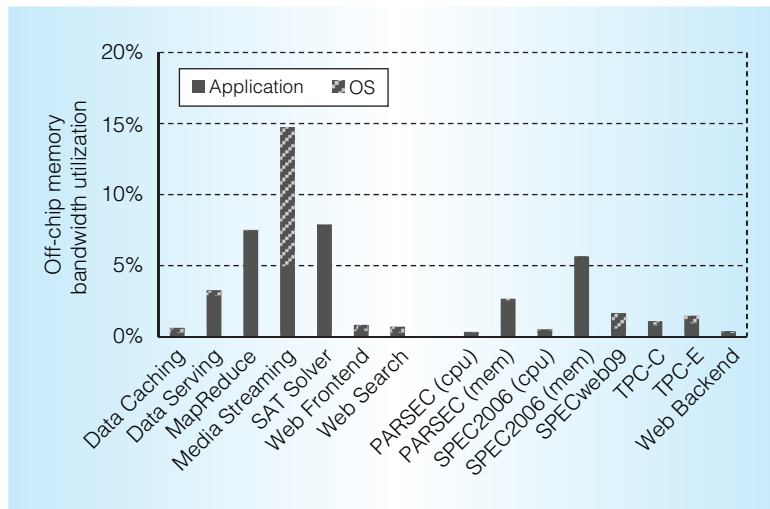


Figure 7. Average off-chip memory bandwidth utilization as a percentage of available off-chip bandwidth. Even at peak system utilization, all workloads exercise only a small fraction of the available memory bandwidth.

OS components to offer insight into the source of the data sharing.

In general, we observe limited read-write sharing across the scale-out applications. We find that the OS-level data sharing is dominated by the network subsystem, seen most prominently in the Data Caching workload, which spends the majority of its time in the OS. This observation highlights the need to optimize the OS to reduce the amount of false sharing and data movement in the scheduler and network-related data structures. Multi-threaded Java-based applications (Data Serving and Web Search) exhibit a small degree of sharing due to the use of a parallel garbage collector that may run a collection thread on a remote core, artificially inducing application-level communication. Additionally, we found that the Media Streaming server updates global counters to track the total number of packets sent; reducing the amount of communication by keeping per-thread statistics is trivial and would eliminate the mutex lock and shared-object scalability bottleneck—an optimization that is already present in the Data Caching server we use. The on-chip application-level communication in scale-out workloads is distinctly different from traditional database server workloads (TPC-C, TPC-E, and Web Backend), which experience frequent interaction between threads

on actively shared data structures that are used to service client requests.

The low degree of active sharing indicates that wide and low-latency interconnects available in modern processors are overprovisioned for scale-out workloads. Although the overhead with a small number of cores is limited, as the number of cores on chip increases, the area and energy overhead of enforcing coherence becomes significant. Likewise, the area overheads and power consumption of an overprovisioned high-bandwidth interconnect further increase processor inefficiency.

Beyond the on-chip interconnect, we also find off-chip bandwidth inefficiency. While the off-chip memory latency has improved slowly, off-chip bandwidth has been improving at a rapid pace. Over the course of two decades, the memory bus speeds have increased from 66 MHz to dual-data-rate at over 1 GHz, raising the peak theoretical bandwidth from 544 Mbytes/second to 17 Gbytes/second per channel, with the latest server processors having four independent memory channels. In Figure 7, we plot the per-core off-chip bandwidth utilization of our workloads as a fraction of the available per-core off-chip bandwidth. Scale-out workloads experience nonnegligible off-chip miss rates, but the MLP of the applications is low, owing to the complex data structure dependencies. The combination of low MLP and the small number of hardware threads on the chip leads to low aggregate off-chip bandwidth utilization even when all cores have outstanding off-chip memory accesses. Among the scale-out workloads we examine, Media Streaming is the only application that uses up to 15 percent of the available off-chip bandwidth. However, our applications are configured to stress the processor, actually demonstrating the worst-case behavior. Overall, modern processors are not able to utilize the available memory bandwidth, which is significantly over-provisioned for scale-out workloads.

The on-chip interconnect and off-chip memory buses can be scaled back to improve processor efficiency. Because the scale-out workloads perform only infrequent communication via the network, there is typically no read-write sharing in the applications; processors can therefore be designed as a collection of core islands using a low-bandwidth

interconnect that does not enforce coherence between the islands, eliminating the power associated with the high-bandwidth interconnect as well as the power and area overheads of fine-grained coherence tracking.⁴ Off-chip memory buses can be optimized for scale-out workloads by scaling back unnecessary bandwidth for systems with an insufficient number of cores. Memory controllers consume a large fraction of the chip area, and memory buses are responsible for a large fraction of the system power. Reducing the number of memory channels and the power draw of the memory buses should improve scale-out workload efficiency without affecting application performance. However, instead of taking a step backward and scaling back the memory bandwidth to match the requirements and throughput of conventional processors, a more effective solution would be to increase the processor throughput through specialization and thus utilize the available bandwidth resources.⁴

The impending plateau of voltage levels and a continued increase in chip density are forcing efficiency to be the primary driver of future processor designs. Our analysis shows that efficiently executing scale-out workloads requires optimizing the instruction-fetch path for multi-megabyte instruction working sets; reducing the core aggressiveness and LLC capacity to free area and power resources in favor of more cores, each with more hardware threads; and scaling back the overprovisioned on-chip and off-chip bandwidth. We demonstrate that modern processors, built to accommodate a broad range of workloads, sacrifice efficiency, and that current processor trends serve to further exacerbate the problem. On the other hand, we outline steps that can be taken to specialize processors for the key workloads of the future, enabling efficient execution by closely aligning the processor microarchitecture with the microarchitectural needs of scale-out workloads. Following these steps can result in up to an order of magnitude improvement in throughput per processor chip, and in the overall datacenter efficiency.⁵ MICRO

Acknowledgments

We thank the reviewers and readers for their feedback and suggestions on all earlier

versions of this work. We thank the PARSA lab for continual support and feedback, in particular Pejman Lotfi-Kamran and Javier Picorel for their assistance with the SPECweb09 and SAT Solver benchmarks. We thank the DSLab for their assistance with SAT Solver, and Aamer Jaleel and Carole Jean-Wu for their assistance with understanding the Intel prefetchers and configuration. We thank the EuroCloud project partners for advocating and inspiring the CloudSuite benchmark suite. This work was partially supported by EuroCloud, project no. 247779 of the European Commission 7th RTD Framework Programme—Information and Communication Technologies: Computing Systems.

References

1. M. Horowitz et al., "Scaling, Power, and the Future of CMOS," *Proc. Electron Devices Meeting*, 2005, pp. 7-15.
2. N. Hardavellas et al., "Toward Dark Silicon in Servers," *IEEE Micro*, vol. 31, no. 4, 2011, pp. 6-15.
3. M. Ferdman et al., "Clearing the Clouds: A Study of Emerging Scale-Out Workloads on Modern Hardware," *Proc. 17th Int'l Conf. Architectural Support for Programming Languages and Operating Systems*, 2012, pp. 37-48.
4. P. Lotfi-Kamran et al., "Scale-Out Processors," *Proc. 39th Int'l Symp. Computer Architecture*, 2012, pp. 500-511.
5. B. Grot et al., "Optimizing Data-Center TCO with Scale-Out Processors," *IEEE Micro*, vol. 32, no. 5, 2011, pp. 52-63.
6. H. Esmailzadeh et al., "Dark Silicon and the End of Multicore Scaling," *Proc. 38th Int'l Symp. Computer Architecture*, 2011, pp. 365-376.
7. G. Venkatesh et al., "Conservation Cores: Reducing the Energy of Mature Computations," *Proc. 15th Conf. Architectural Support for Programming Languages and Operating Systems*, 2010, pp. 205-218.
8. N. Hardavellas et al., "Reactive NUCA: Near-Optimal Block Placement and Replication in Distributed Caches," *Proc. 36th Int'l Symp. Computer Architecture*, 2009, pp. 184-195.

Michael Ferdman is an assistant professor in the Department of Computer Science at Stony Brook University. His research focuses on computer architecture, particularly on server system design. Ferdman has a PhD in electrical and computer engineering from Carnegie Mellon University.

Almutaz Adileh is a PhD candidate in the Department of Computer Science at Ghent University. His research focuses on computer architecture, particularly on improving performance in power-limited chips. Adileh has an MSc in computer engineering from the University of Southern California.

Onur Kocberber is a PhD candidate in the School of Computer and Communication Sciences at École Polytechnique Fédérale de Lausanne. His research focuses on specialized architectures for server systems. Kocberber has an MSc in computer engineering from TOBB University of Economics and Technology.

Stavros Volos is a PhD candidate in the School of Computer and Communication Sciences at École Polytechnique Fédérale de Lausanne. His research focuses on computer architecture, particularly on memory systems for high-throughput and energy-aware computing. Volos has a Dipl-Ing in electrical and computer engineering from the National Technical University of Athens.

Mohammad Alisafae performed the work for this article while he was a researcher in the School of Computer and Communication Sciences at École Polytechnique Fédérale de Lausanne. His research interests include multiprocessor cache coherence and memory system design for commercial workloads. Alisafae has an MSc in electrical and computer engineering from the University of Tehran.

Djordje Jevdjic is a PhD candidate in the School of Computer and Communication Sciences at École Polytechnique Fédérale de Lausanne. His research focuses on high-performance memory systems for servers, including on-chip DRAM caches and

3D-die stacking, with an emphasis on locality and energy efficiency. Jevdjic has an MSc in electrical and computer engineering from the University of Belgrade.


Cansu Kaynak is a PhD candidate in the School of Computer and Communication Sciences at École Polytechnique Fédérale de Lausanne. Her research focuses on server systems, especially memory system design. Kaynak has a BSc in computer engineering from TOBB University of Economics and Technology.

Adrian Daniel Popescu is a PhD candidate in the School of Computer and Communication Sciences at École Polytechnique Fédérale de Lausanne. His research focuses on the intersection of database management systems with distributed systems, specifically query performance prediction. Popescu has an MSc in electrical and computer engineering from the University of Toronto.

Anastasia Ailamaki is a professor in the School of Computer and Communication Sciences at École Polytechnique Fédérale de Lausanne. Her research interests include optimizing database software for emerging hardware and I/O devices and automating database management to support scientific applications. Ailamaki has a PhD in computer science from the University of Wisconsin-Madison.

Babak Falsafi is a professor in the School of Computer and Communication Sciences at École Polytechnique Fédérale de Lausanne and the founding director of EcoCloud, an interdisciplinary research center targeting robust, economic, and environmentally friendly cloud technologies. Falsafi has a PhD in computer science from the University of Wisconsin-Madison.

Direct questions and comments about this article to Michael Ferdman, Stony Brook University, 1419 Computer Science, Stony Brook, NY 11794; mferdman@cs.stonybrook.edu.

 Selected CS articles and columns are also available for free at <http://ComputingNow.computer.org>.