

Clearing the Clouds: A Study of Emerging Workloads on Modern Hardware

Michael Ferdman^{‡†}, Almutaz Adileh[†], Onur Kocberber[†], Stavros Volos[†], Mohammad Alisafae[†],
Djordje Jevdjic[†], Cansu Kaynak[†], Adrian Daniel Popescu[†], Anastasia Ailamaki[†], Babak Falsafi[†]

[‡]CALCM
Carnegie Mellon University

[†]EcoCloud
École Polytechnique Fédérale de Lausanne

ABSTRACT

Emerging scale-out cloud applications need extensive amounts of computational resources. However, data centers using modern server hardware face physical constraints in space and power, limiting further expansion and calling for improvements in the computational density per server and in the per-operation energy use. Therefore, continuing to improve the computational resources of the cloud while staying within physical constraints mandates optimizing server efficiency to ensure that server hardware closely matches the needs of scale-out cloud applications.

We use performance counters on modern servers to study a wide range of cloud applications, finding that today’s predominant processor architecture is inefficient for running these workloads. We find that inefficiency comes from the mismatch between the application needs and modern processors, particularly in the organization of instruction and data memory systems and the processor core architecture. Moreover, while today’s predominant architectures are inefficient when executing scale-out cloud applications, we find that the current hardware trends further exacerbate the mismatch. In this work, we identify the key micro-architectural needs of cloud applications, calling for a change in the trajectory of server processors that would lead to improved computational density and power efficiency in data centers.

1 INTRODUCTION

Cloud computing is emerging as a dominant computing platform for delivering scalable online services to a global client base. Today’s popular services, such as search engines, social networks, and video sharing, all offer their services from cloud data centers. With the industry rapidly expanding [15], service providers are building new data centers, 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 [18, 19], no longer able to reduce power consumption or increase power density in new chips. Physical constraints have therefore become the dominant limiting factor for data centers because their sheer size and electrical power demands cannot be met.

Recognizing the physical constraints that stand in the way of further growth, cloud providers now optimize their data centers for compute density and power consumption. Cloud providers have already begun building server systems specifically targeting cloud data centers, improving compute density and energy efficiency by using high-efficiency power supplies and removing unnecessary board-level components such as audio and graphics chips [17, 33].

Although major design changes are being introduced at the board- and chassis-level of new cloud servers, the processors used in these new servers are not designed to efficiently run scale-out cloud applications. Processor vendors have moved toward producing more power-efficient processors targeting the server space, however, these processors use the same underlying architecture as the processors targeting the general purpose market. Unfortunately, this has led to extreme inefficiency in today’s data centers, as the trends of both general purpose (e.g., Intel and AMD) and traditional server processors (e.g., Sun Niagara, IBM Power7) target scale-up applications, having been established long before to the emergence of scale-out cloud workloads. Recognizing the space and power

inefficiency of modern processors for scale-out cloud applications, some vendors and researchers even conjecture that using processors built for the mobile space may be more efficient [13, 27, 37, 38].

In this work, we observe that scale-out cloud applications share many inherent characteristics that place them into a distinct workload class from desktop, scientific, and even traditional server workloads. We perform a detailed micro-architectural study of a range of cloud applications, finding a large mismatch between the micro-architectural demands of the cloud applications and today’s predominant processor architecture. We observe significant over-provisioning of the memory hierarchy and core micro-architecture for the scale-out cloud applications. Moreover, the mismatch between the scale-out cloud applications and server processors will grow if the current processor trends continue. At the same time, we find that the characteristics of scale-out cloud applications can be leveraged to gain micro-architectural area- and energy-efficiency in future servers.

We use performance counters to study the behavior of scale-out cloud applications running on modern server processors. Our results demonstrate:

- **Instruction-cache misses account for up to 60% of the stall time (35% of execution time) in cloud applications.** Instruction-caches and associated next-line prefetchers found in modern processors are inadequate for cloud applications.
- **Instruction- and memory-level parallelism in scale-out cloud applications is low.** Modern aggressive OoO cores are excessively complex, needlessly consuming power and on-chip area without providing performance benefits.
- **Data working sets of scale-out cloud applications 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 performance.
- **The scale-out nature of cloud applications avoids on-chip communication.** Cloud applications see no benefit from modern on-chip interconnects engineered for fine-grained coherence and high bandwidth core-to-core communication.

The rest of this paper is organized as follows. In Section 2, we provide an overview of the state-of-the-art server processors and scale-out cloud applications. We provide a detailed description of our benchmarking methodology in section Section 3. We present our results in Section 4, concentrating on the mismatch between the needs of cloud applications and modern processors. We summarize related work in Section 5 and conclude in Section 6.

2 MODERN CORES AND WORKLOADS

Today’s data centers are built around conventional desktop processors whose architecture was designed for a broad consumer market. The dominant processor architecture 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 [12, 18, 19, 44], 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 efficiency.

2.1 Dominant processor architectures

Today’s processors comprise several aggressive out-of-order cores connected with a high-bandwidth on-chip interconnect to a deep (three-level) cache hierarchy. While core aggressiveness and clock frequency enabled a rapid increase in computational performance, off-chip memory latency improvements were not as rapid. The “memory wall”—the gap between the speed at which cores could compute and the speed at which data could be delivered to the cores for computation—mandated that the data working set must fit into the on-chip caches to allow the core to compute at full speed. Modern processors therefore split the die area in two roughly equal parts, with half of the die dedicated to the cores and tightly-coupled private caches and the other half dedicated to a large shared last-level cache [8]. The emergence of multi-core processors offered the possibility to run computationally intensive multi-threaded applications, adding the requirement of fast and high-bandwidth core-to-core communication to allow cores to compute without incurring significant delays when operating on actively shared data.

To leverage the increasing number of transistors on chip for higher single-thread performance, cores are engineered to execute independent instructions out of order (OoO), allowing the processor to temporarily bypass instructions that stall due to a slow cache access. While OoO execution can improve core resource utilization through instruction-level parallelism (ILP), the core’s complexity (number of transistors and power) increases dramatically depending on the width of the pipeline and the size of the reorder window. Large windows require selecting and scheduling among many instructions while tracking all memory and register dependencies, functionality that requires a large and power-hungry scheduler, reorder buffer, and load-store structures. Moreover, the efficacy of growing the instruction reorder window rapidly drops off, resulting in diminishing returns at exponentially increasing area and energy costs with every processor generation. Notably, although wide pipelines and large reorder windows do not harm the core performance, low-ILP applications execute inefficiently because the area and power costs spent on these techniques do not yield a performance benefit.

The first-level instruction and data caches capture the primary working set of applications, enabling low-latency access to the most-frequently used data. However, to maintain low access latency, the cache capacity must remain small. As the size of the last-level cache (LLC) has reached tens of megabytes in modern processors, the access latency of the LLC has itself created a speed gap between the first-level caches and LLC, pushing processor designers to mitigate the gap by inserting an intermediate-size secondary cache. Additionally, to further mitigate the large LLC latency, the number of miss-status handling registers (MSHRs) is increased to allow for a large number of memory-reference instructions to be performed in parallel. Like the core structures for supporting ILP, the mechanisms to support memory level parallelism (MLP) use considerable area and energy. Increasing parallelism in the LLC and off-chip accesses can give a tremendous performance improvement when many independent memory accesses are available to execute, but results in poor efficiency when executing workloads with low MLP.

To increase the core utilization when MLP is low, modern processors add support for simultaneous multi-threading (SMT), enabling two software threads to be executed simultaneously in the same core. SMT cores operate like single-threaded cores, but introduce instructions from two independent software threads into the reorder window, enabling the core to find independent memory accesses and perform them in parallel, even when both software threads have low MLP. However, introducing instructions from multiple software threads into the same pipeline causes contention for core resources, limiting the performance of each thread compared to when that thread runs alone.

2.2 Dominant scale-out cloud applications

To find the set of applications that dominate the use of cloud infrastructure, we examined a selection of internet services based on their popularity [2]. For each popular service, we analyzed the class of application software used by the service providers to offer these services, either on their own cloud infrastructure or on a cloud infrastructure leased from a third party. We present an overview of the applications most commonly found in today’s clouds, along with brief descriptions of typical configuration characteristics and dataset sizes. Overall, we find that all scale-out cloud applications have functionally similar characteristics; all applications we examined operate on large data sets that are split across a large number of nodes, typically into memory-resident shards, serving large numbers of completely independent requests that do not share any state, having application software designed specifically for the cloud infrastructure where unreliable nodes may come and go and where inter-node connectivity is used only for high-level task management and coordination.

Data Serving. Various *NoSQL* data stores [5, 10, 41] have been explicitly designed to serve as the backing store for large-scale web applications such as the Facebook inbox [41] and Google Earth and Google Finance [5], providing fast and scalable storage with varying and rapidly evolving storage schema. The *NoSQL* systems split hundreds of terabytes of data into shards and horizontally scale to large cluster sizes, typically using indexes that support fast lookup and key range scans to retrieve the set of requested objects. For simplicity and scalability, these systems are designed to support queries that can be completely executed by a single storage node, with any operations that require combining data from multiple shards relegated to the middleware.

MapReduce. The explosion of accessible human-generated information necessitates automated analytical processing to cluster, classify, and filter this information. The map-reduce paradigm [9] has emerged as a popular

approach to handling large-scale analysis, farming out requests to a cluster of nodes that first perform filtering and transformation of the data (map) and then aggregate the results (reduce). A key advantage of the map-reduce paradigm is the separation of infrastructure and machine learning algorithms [3]. Users implement the algorithm as idempotent *map* and *reduce* functions and provide them to the map-reduce infrastructure, which is then responsible for orchestrating the work. Because of the generality of the infrastructure and the need to scale it to thousands of independent servers, communication between tasks is typically limited to reading and writing files in a distributed file system. For example, map jobs produce temporary files that are subsequently read by the reduce jobs, effectively rendering all map and reduce jobs architecturally independent.

Media Streaming. The availability of high-bandwidth connections to home and mobile devices has made media streaming services such as NetFlix, YouTube, and YuKu ubiquitous. Streaming services use large server clusters to gradually packetize and transmit media files ranging from megabytes to gigabytes in size, pre-encoded in various formats and bit-rates to suit a wide client base. Sharding of media content ensures that servers frequently send the same content to multiple users and enables in-memory caching of the content. While in-memory caching is effective, the on-demand unicast nature of most of today’s streaming services practically guarantees that even when streaming the same media file to many clients concurrently, the streaming server will work on disjoint pieces of the media file for all clients.

Simulation. The ability to temporarily allocate compute resources in the cloud without purchasing the infrastructure has created an opportunity to conduct large-scale simulations in the cloud. However, unlike the traditional super-computer environment with high-bandwidth low-latency dedicated interconnects with reliable and balanced memory and compute resources, the cloud offers dynamic and heterogenous resources that are loosely connected over an IP network. Large-scale simulation tasks must therefore be adapted to a worker-queue model with centralized load balancing that rebalances simulation tasks across a dynamic pool of compute resources, minimizing the amount of data exchanged between the workers and load balancers and practically eliminating any communication between workers. While each task may experience high data locality with small working sets within a task, as the workers proceed from task to task in their queue, simulation nodes move between completely unrelated and distant parts of the dataset in arbitrary order.

Web Frontend. Traditional web services with dynamic and static content are moved into the cloud to provide fault-tolerance and dynamic scalability by bringing up the needed number of servers behind a load balancer. Although many variants of the traditional web stack are used in the cloud (e.g., substituting Apache [34] with other web server software or using other language interpreters in place of PHP), the underlying service architecture remains unchanged. Independent client requests are accepted by a stateless web server process which either directly serves static files from disk or passes the request to a stateless middleware script, written in a high-level interpreted or byte-code compiled language, which is then responsible for producing dynamic content. All state information is stored by the middleware in backend databases such as cloud NoSQL data stores or traditional relational SQL servers supported by key-value cache servers to achieve high throughput.

Web Search. The largest data center applications are used to provide access to public internet indexes (e.g., Google, Bing, Yahoo!, Baidu), while many smaller web search clusters offer searches of specialized indexes of corporate or government data. Multi-terabyte indexes are split into shards, with each *index serving node (ISN)* responsible for processing requests to its own shard. A frontend node sends index search requests to all ISNs in parallel, collects and sorts the responses, and sends a formatted reply to the requesting client. Hundreds of unrelated search requests are handled by each ISN every second, with minimal locality; shards are therefore sized to fit into the RAM of the ISNs to avoid reducing throughput due to disk I/O. For performance scalability, ISNs may be replicated in case a single ISN per shard is unable to sustain the request throughput or meet the quality of service requirements. ISNs communicate only with the frontend nodes and never to other ISNs. Similarly, within a single node, processing threads do not communicate, independently working to process each request.

3 METHODOLOGY

We conduct our study on a PowerEdge M1000e cluster [11] with two Intel x5670 processors and 24GB of RAM in each blade. Each Intel x5670 processor includes six aggressive out-of-order processor cores with a three-level

Table 1.A. Architectural parameters

Processor	Intel Xeon 5670, 6 cores, 32nm @ 2.93GHz
CMP Size	6 OoO cores
Superscalar width	4-wide issue
Reorder buffer	128 entries
Load/Store buffer	48/32 entries
Reservation stations	36 entries
L1 Cache	split I/D, 32KB, 4-cycles access latency
L2 Cache	6-core CMP: 256KB per core, 12-cycles access latency
LLC (L3) cache	12MB, cycles 39-cycles access latency
Memory	24GB, 180/280 cycles access latency local/remote DRAM

Table 1.B. Performance counter methodology

Component	Definition	Estimation Method
Busy	Number of cycles at least one micro-op is retired	Corresponding counter
Stalled	Number of cycles without any micro-op retiring	Corresponding counter
Stalled on Memory	Number of cycles spent in memory	Outstanding Super Queue requests for instructions and data normalized by the number of corresponding cycles that Super Queue is not empty
Frontend	Stalls due to instruction fetching	Resource stall cycles subtracted from total issued stall cycles
Backend	Stalls due to backend	Resource stall cycles

cache hierarchy: the L1 and L2 caches are private to each core, while the LLC (L3) is shared among all cores. For all but one experiment, the nodes run CentOS 5.5 with the 2.6.18 Linux kernel. For the TPC-E benchmark, the node runs Microsoft Windows Server 2008 Release 2. Table 1.A summarizes the key architectural parameters of the systems.

3.1 Measurement tools and methodology

We analyze the behavior of scale-out cloud applications using the Intel VTune [24] software, a tool that provides an interface to the processor performance counters. We run each experiment multiple times so as to get a 95% confidence interval within a 5% error range. We note that computing a breakdown of the various execution-time stall components of superscalar out-of-order processors cannot be performed precisely due to overlapped work in the pipeline [14, 26]. Whenever possible, we present results based on the performance counters that have no overlap (cannot perform parallel work due to the organization of the pipeline); when non-overlapping counters are not available, we plot the results side-by-side rather than in a stack to indicate the potential overlaps. Table 1.B lists the specific performance counters used in our evaluation to obtain the execution time and core stalls breakdowns.

In order to break down the core stalls into frontend stalls and backend components, we calculate the frontend stalls (i.e., instruction fetch stalls only) by subtracting the resource stalls from the amount of cycles where no instructions were issued to the backend. We calculate cycles wasted to issuing wrong-path instructions but we don't include them in the core stall breakdown.¹

Data stalls overlap other data stalls and computation in the pipeline. Although prior work advocates a naïve approach of multiplying miss events by the approximate miss penalty to compute data stalls [1], we find that this approach is not sufficiently accurate to provide clear micro-architectural insight. Instead, we analyze the LLC and memory stalls due to long-latency data misses by leveraging the occupancy statistics of the L2 MSHRs.² The MSHR occupancy statistics enable us to measure the number of cycles when there is at least one L2 miss being serviced. We exclude the cycles to access the L1 and L2 caches from the data-stall analysis, as we find that the L1 and L2 cache hits are effectively hidden by the out-of-order core, a result that corroborates prior work [25].

-
1. On average across our workloads, less than 5% of cycles are wasted due to issuing wrong-path instructions.
 2. In our servers, the outstanding L2 misses are maintained in the *super queue* structure between the L2 and LLC.

Our cache sensitivity analysis is performed by dedicating two cores to a polluting micro-benchmark that traverses a static array of size equal to the polluted cache size in random patterns so as to ensure that its requests reach the LLC keeping the array cache blocks in the most recent place. Before running each experiment we verify that our micro-benchmark correctly pollutes the cache.

Our read-write sharing experiment is performed by splitting the cores evenly to both physical processors. In the dual-processor configuration, the LLC of each processor is private. In a case of an LLC miss request, the request is forwarded to the remote processor only in the case that the block is modified. Therefore, we count the number of LLC requests that hit in the remote LLC.

Our prefetching experiment is performed by disabling the adjacent-cache-line and stride prefetchers that fetch into the L2 cache. Because the L3 cache is inclusive, upon a prefetch request that misses in LLC the block is also allocated in the L3 cache. Therefore, we calculate both L2 and L3 miss ratios as an indication of the effectiveness of each prefetcher.

3.2 Cloud application experimental setup

Data Serving. We benchmark one node running the Cassandra 0.7.3 [41] storage system with a 30GB YCSB dataset that exceeds the node’s RAM capacity. Server load is generated using the YCSB 0.1.3 client [7] that sends requests following a Zipfian distribution with an equal number of reads and updates.

MapReduce. We benchmark one node of a Hadoop 0.20.2 cluster running the algorithm WordCount on a 4GB set of Wikipedia pages. Each core runs one map and one reduce job. We also evaluate a machine-learning text-classification algorithm operating on Wikipedia pages and observe that the behavior of the two systems were practically identical. For the rest of the paper we only show results for the WordCount algorithm.

Media Streaming. We benchmark a node running Darwin Streaming Server 6.0.3 to serve 50GB of videos encoded in several bit-rates ranging between 42Kbps and 60Kbps and use the Faban workload driver [22] to simulate the clients. We limit our setup to low bit-rate streams to shift stress away from network I/O.

Simulation. We benchmark one node running parallel symbolic execution to search for programming bugs in an application binary [6]. We use Cloud9 to analyze the command-line *printf* utility from the GNU CoreUtils 6.10.

Web Frontend. We benchmark a frontend node serving Olio, a Web 2.0 web-based social event calendar. Using the tools provided as part of the Cloudstone benchmark, we generate a backend dataset with a scale factor of 50000 (22GB) and use the Faban synthetic workload generator to simulate clients [22]. The frontend node runs Apache Web Server 2.2.19 with a built-in PHP 5.3.5 module and APC 3.1.8 PHP opcode cache. The backend node runs the MySQL 5.5.9 database engine.

Web Search. We benchmark one index serving node (ISN) of the distributed version of Nutch 1.1/Lucene 3.0.1 with an index size of 2GB and data segment size of 23GB of content crawled from the public internet. We make sure that the search index fits in memory to eliminate page faults and minimize disk activity to mimic real-world setups [37]. We simulate the clients using the Faban workload driver. The clients are configured to achieve the maximum search request rate while ensuring that 90% of all search queries complete in under 0.5 seconds.

3.3 Traditional application experimental setup

PARSEC 2.1. We benchmark one node running the official benchmark applications with the *native* input, reporting results averaged across all benchmarks.

SPEC CINT2006. We benchmark one core of a node running the official benchmark applications with the first *reference* input, reporting results averaged across all benchmarks.

TPC-C. We benchmark a node executing the TPC-C 5.11 workload on a commercial enterprise database management system (DBMS). The database load is generated by 160 clients configured with zero think time and running on a separate node. Our TPC-C database has 40 warehouses (4GB data and 2GB index). The DBMS is configured with a 3GB buffer pool and direct I/O.

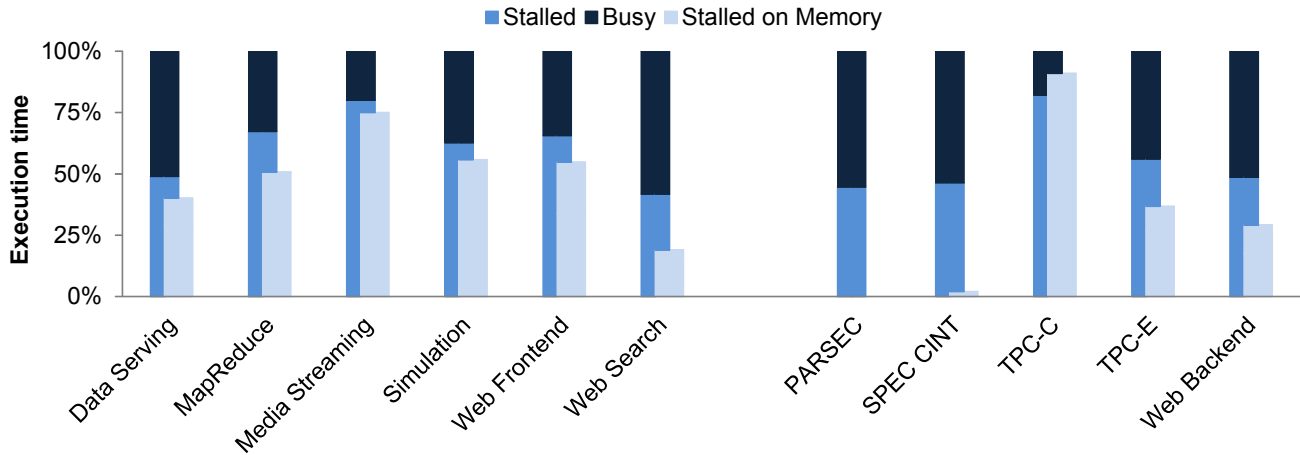


Figure 1. Execution time breakdown and memory stalls of scale-out cloud applications (left) and

TPC-E. We benchmark a node executing the TPC-E 1.12 workload on a commercial enterprise database management system (DBMS). The client driver runs on the same node but is bound to a core that is not used by the database system. Our TPC-E database contains 5000 customer records (55GB database). The DBMS is configured with a 22GB buffer pool.

3.4 I/O Infrastructure

Data-intensive scale-out cloud applications and traditional database workloads exhibit a significant amount of disk I/O, with a large fraction of the non-sequential read and write accesses scattered throughout the storage space. If the underlying I/O bandwidth is limited, either in raw bandwidth or in I/O operation throughput, the I/O latency is exposed to the system, resulting in an I/O-bound workload where the CPU is underutilized and the application performance unnecessarily suffers.

To isolate the CPU behavior of the applications, our experimental setup over-provisions the I/O subsystem to avoid an I/O bottleneck. To avoid bringing up disk arrays containing hundreds of disks and flash devices, as is traditionally done with large-scale database installations [42], we construct a network-attached iSCSI storage array by creating large RAM disks in separate nodes and connecting the node under test to the iSCSI storage via a high-speed ethernet network. This approach places the entire data set of our applications in the memory of the remote nodes, creating an illusion of a large disk cluster with extremely high I/O bandwidth and low latency.

4 RESULTS

The behavior and characteristics of scale-out cloud applications show a distinct difference from the desktop, parallel, and even traditional transaction processing workloads, indicating a disparity between processors designed for these workloads and the micro-architectural needs of cloud applications. Figure 1 illustrates the execution-time breakdown of cloud applications and popular processor benchmark suites.

We classify each cycle of execution as *Busy* if at least one instruction was retired during that cycle or *Stalled* otherwise. The execution-time breakdown of cloud applications shows slightly more stalls than conventional processor benchmarks, typically spending 50% or more cycles without retiring any instructions. Although the fraction of stalled cycles appears similar to desktop and parallel applications at a high level, the nature of the stalls of the cloud applications is radically different. Overlapped with the time breakdown, we plot *Stalled on Memory*, the fraction of cycles spent by the application waiting on long-latency memory accesses (L2 cache misses). While desktop and parallel applications stall due to a diverse set of resource limitations in the processor core (not shown), cloud applications show memory system behavior that more closely matches traditional online transaction processing workloads (TPC-C, TPC-E, and Web Backend). However, we also observe that cloud applications differ considerably from the traditional online transaction processing workload (TPC-C) which spends over 80% of its time stalled due to dependent memory accesses. Instead, we find that cloud applications are similar to the more

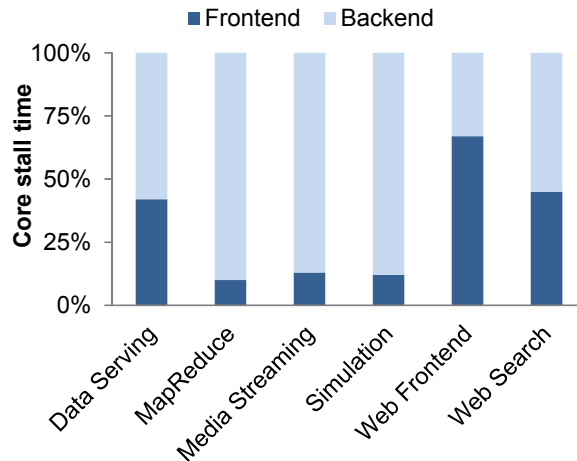


Figure 2. Core stall time breakdown

recent transaction processing benchmarks (TPC-E and Web Backend) that use more complex data schema and perform more complex queries than the traditional workload.

Although the behavior of cloud applications is similar, the class of cloud applications as a whole differs distinctly from other workloads. Processor architectures optimized for desktop and parallel workloads are not optimized for cloud applications that spend the majority of their time waiting for cache misses, resulting in a clear micro-architectural mismatch. At the same time, architectures designed for workloads that perform only trivial computation and spend all of their time waiting on memory (e.g., TPC-C) also cannot cater to cloud applications. In the remainder of this section, we provide a detailed analysis of the inefficiencies of running cloud applications on modern processors.

4.1 Frontend Inefficiencies

- *Cores idle due to high instruction-cache miss rates*
- *L2 caches increase average I-fetch latency*
- *Excessive LLC capacity leads to long I-fetch latency*

Instruction-fetch stalls play a pivotal role in system performance by preventing the core from making forward progress due to a lack of instructions to execute. Frontend stalls serve as a fundamental source of inefficiency for both area and power as the core real-estate and power consumption are entirely wasted for the cycles that the frontend spends fetching instructions. Figure 2 plots the core stalls divided into cycles stalled due to the frontend (instruction fetch) and backend (execution pipeline and memory) when running cloud applications. Cloud applications show considerable frontend stalls, from 10% to over 60% of the stalls.

L1 instruction-cache misses are the underlying cause of the majority of instruction-fetch stalls in cloud applications. Figure 3 presents the L1 instruction-cache miss ratios, indicating that the instruction working sets of the cloud applications exceed the L1-cache capacity. Furthermore, Figure 4 shows the ratios of instruction fetches that miss in the L2 cache are high. This indicates that the L1 instruction-cache capacity experiences a significant shortfall and cannot be mitigated by a minor increase in the L1-cache capacity or the addition of a moderately-sized L2 cache. Even for Media Streaming, whose active instruction working set is smaller than other applications, the aggregate number of L2 instruction misses is similar to other cloud applications due to the higher frequency of L2 instruction accesses.

Stringent access-latency requirements of the L1 instruction caches preclude increasing the size of the caches to capture the instruction working set of cloud applications, which is an order of magnitude larger than the caches found in modern processors. We find that today’s processor architectures cannot tolerate the latency of L1 instruction-cache misses, avoiding frontend stalls only for applications whose entire instruction working set fits into the L1 cache.

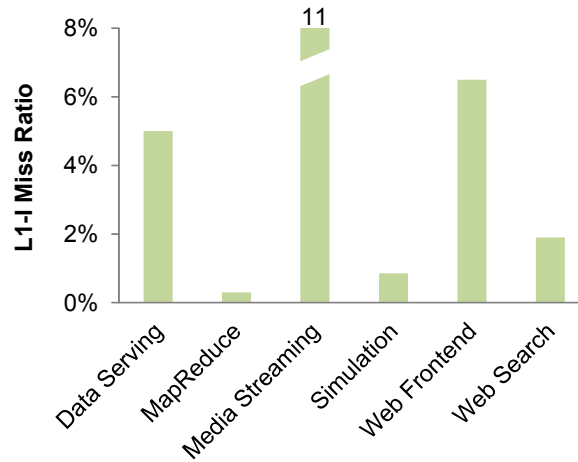


Figure 3. L1 instruction-cache miss ratio

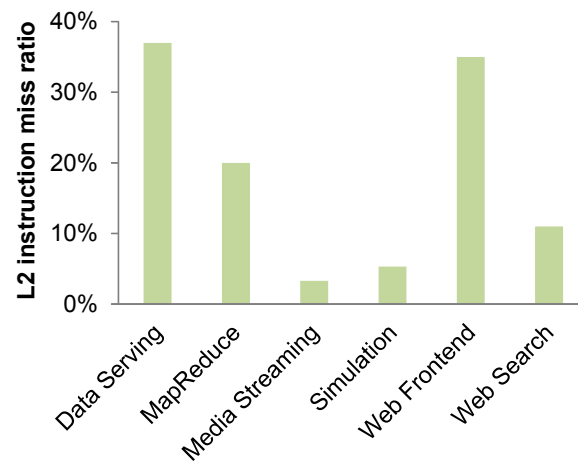


Figure 4. L2 cache instruction-miss ratio

The disparity between the needs of the cloud applications and the processor architecture are apparent when examining the instruction-fetch path. Although exposed instruction-fetch stalls would 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 misses for up to 37% of the fetch requests, increasing the average fetch latency by placing an additional intermediate lookup structure on the path to retrieving instruction blocks from the LLC. Moreover, the entire instruction working set of any cloud application is considerably smaller than the LLC capacity; however, because the LLC is a large cache with a large uniform access latency, it contributes an unnecessarily large instruction-fetch penalty (39 cycles to access the 12MB cache).

Implications: To improve efficiency and reduce frontend stalls, processors built for cloud applications 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 cores [20] is likely to considerably reduce the frontend 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 these blocks from the closest cores.

Furthermore, although modern processors include next-line instruction prefetchers, high instruction-cache miss rates and significant frontend stalls indicate that the prefetchers are ineffective for cloud applications. Cloud applications are written in high-level languages, use third-party libraries, and execute operating system code, thereby exhibiting complex non-sequential access patterns that are not captured by the simple next-line prefetchers.

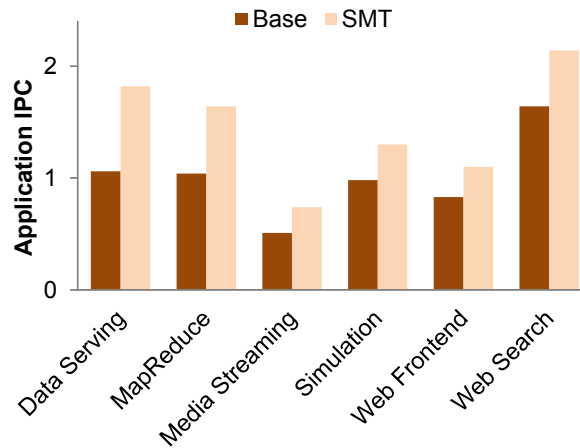


Figure 5. Application instructions executed per cycle for systems with and without SMT (out of max IPC of 4)

Including instruction prefetchers capable of predicting these complex patterns is likely to improve overall processor efficiency by eliminating wasted cycles due to frontend stalls.

4.2 Core Inefficiencies

- *Low ILP precludes effectively using the full core width*
- *ROB and LSQ are underutilized due to low MLP*
- *Resource sharing in SMT pipeline limits performance*

Modern processors execute instructions out of order to enable simultaneous execution of multiple independent instructions per cycle. 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 memory-level parallelism (MLP) by finding independent memory-accesses within the instruction window and performing the memory accesses in parallel. Exposed latency of LLC hits and off-chip memory accesses cannot be entirely hidden by out-of-order execution; achieving high MLP is therefore key to achieving high core utilization by reducing the data access latency.

Today’s processors use 4-wide cores that can decode, issue, execute, and retire up to four instructions on each cycle. However, in practice, instruction-level parallelism (ILP) is limited by dependencies. The *Base* bars in Figure 5 show the average number of instructions retired per cycle when running cloud applications on an aggressive 4-wide out-of-order core. Despite the abundant availability of core resources and functional units, cloud applications achieve a modest application IPC close to 1.0.

Modern processors have 48-entry load-store queues, enabling up to 48 memory-reference instructions in the 128-instruction window. However, just as instruction dependencies limit ILP, address dependencies limit MLP. We measure the average number of concurrently outstanding L2 cache misses in cloud applications. The *Base* bars in Figure 6 present the cloud application MLP, with the average MLP below 1.9 across all applications. These results show that the memory accesses in cloud applications are replete with complex dependencies, limiting the MLP that can be found in cloud applications by today’s aggressive processors.

Support for 4-wide out-of-order execution with a 128-instruction window and up to 48 outstanding memory requests requires multi-branch prediction, numerous ALUs, forwarding paths, many-port register banks, large instruction schedulers, wide-associativity reorder buffers and load-store queues, and many other complex on-chip structures. The complexity of such cores limits core count, leading to chip designs with several cores that consume

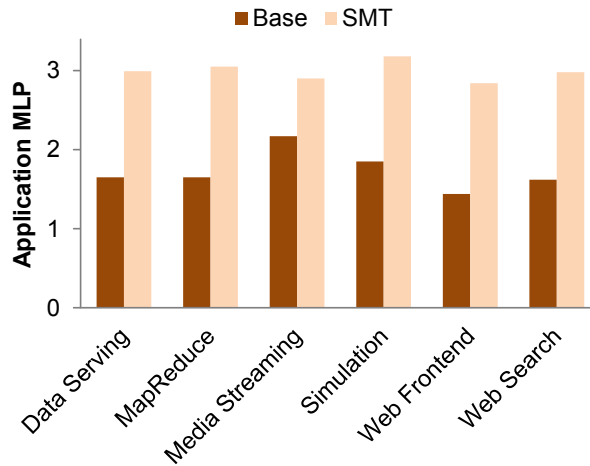


Figure 6. Memory-level parallelism for systems with and without SMT

half of the available on-chip real-estate and dissipate the vast majority of the chip’s dynamic power budget. However, our results indicate that cloud applications exhibit low ILP and MLP, deriving benefit only from a small degree of out-of-order execution and instruction reordering. As a result, the inherent nature of cloud applications cannot effectively utilize the available core resources. Both the die area and the energy are wasted, leading to data-center inefficiency when entire buildings are packed with aggressive cores, designed to retire 4 instructions per cycle with over 20 outstanding memory requests, but executing applications with an average UIPC of 0.9 and average MLP of 1.6. Moreover, current industry trends point toward even greater inefficiency in the future; over the past two decades, processors have gradually moved to increasingly complex cores, raising the core width from 2-way to 4-way and increasing the window size from 20 to 128 instructions.

The inefficiency of modern cores running applications without abundant IPC and MLP has led to the addition of simultaneous multi-threading (SMT) to the processor cores to enable the core resources to be simultaneously shared by multiple software threads, thereby guaranteeing that independent instructions are available to exploit parallelism. We present the UIPC and MLP of an SMT-enabled core running cloud applications in Figure 5 and Figure 6 using the bars labeled *SMT*. As expected, the MLP found and exploited by the core when two independent application threads run concurrently is nearly doubled compared to the system without SMT. However, because the core resources are shared, threads compete in the pipeline and block each other from retiring instructions. As a result, the SMT system achieves only a 30% performance improvement, despite a doubling of MLP.

Implications: The scale-out nature of cloud applications makes them ideal candidates to exploit multi-threaded multi-core architectures. Today’s mainstream processors offer excessively complex cores, resulting in inefficiency through waste of resources. At the same time, our results corroborate prior work [37], indicating that niche processors offer excessively simple (e.g., in-order [8, 27]) cores that cannot leverage the available ILP and MLP in cloud applications. We find that cloud applications would be well-suited by 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 4-way core, two independent 2-way cores would consume fewer resources while achieving higher aggregate throughput. Furthermore, each narrower core does not require a large instruction window, reducing the per-core area and power consumption compared to today’s processors, enabling higher compute density by integrating more cores per chip.

4.3 Data-Access Inefficiencies

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

More than half of today’s processor die area is dedicated to the memory system. Modern processors feature a three-level cache hierarchy where the last-level cache (LLC) is a large-capacity cache shared among all cores. To

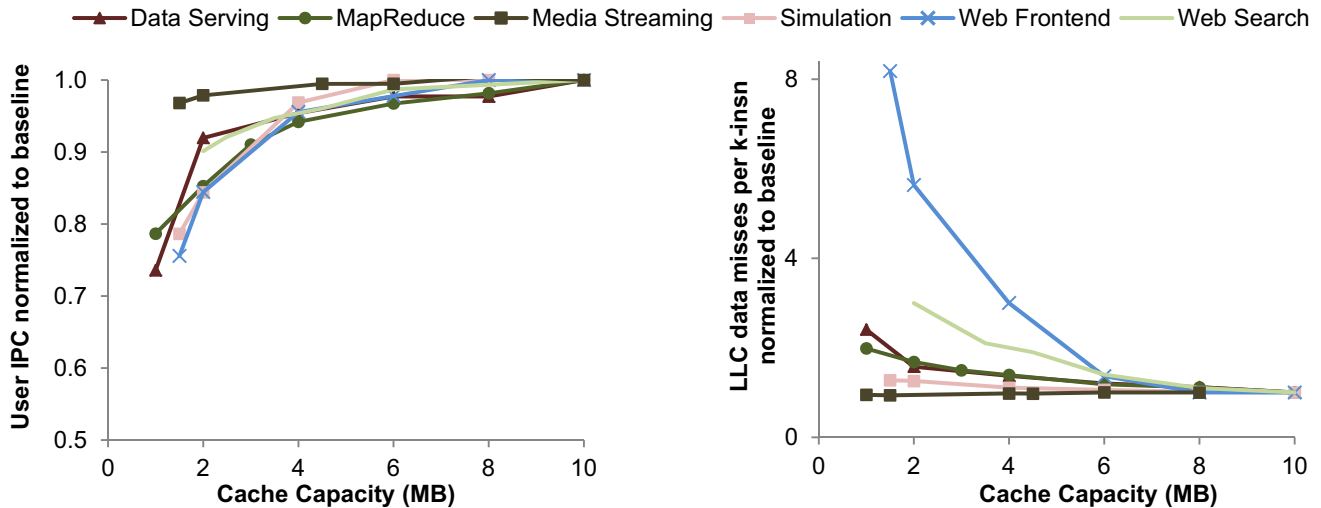


Figure 7. Sensitivity to LLC capacity with respect to performance (left) and cache miss ratio (right)

enable high-bandwidth data fetch, each core can issue up to 16 simultaneously outstanding L2 cache misses. The high-bandwidth on-chip interconnect enables cache-coherent communication between cores. To mitigate the capacity and latency gap between the L2 and LLC caches, 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 due to semiconductor manufacturing improvements. We investigate the utility of growing the LLC capacity for cloud applications in Figure 7. On the left, we plot the system performance¹ as a function of LLC capacity, normalized to the baseline system with a 12MB LLC. We find minimal performance sensitivity to LLC cache size beyond 4MB for most applications. On the right, we plot the normalized LLC miss rate. The LLC captures the instruction working sets of cloud workloads with only one or two megabytes of cache. Beyond this point, small shared supporting structures may consume another one to two megabytes. However, we find that capacity beyond 4MB is not effectively utilized even for the worst-case benchmark (Web Frontend), having minimal effect on the system performance despite an increase in the LLC miss rate. Because cloud applications operate on massive data sets and service large numbers 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.

Our results show that the on-chip resources devoted to the LLC are one of the key limiters of cloud-application compute 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 due to a reduction in average memory latency for those applications. However, because the LLC capacity already exceeds the cloud-application requirements by 2x-3x, 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 [21] 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.

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

1. User-IPC is proportional to application throughput [45].

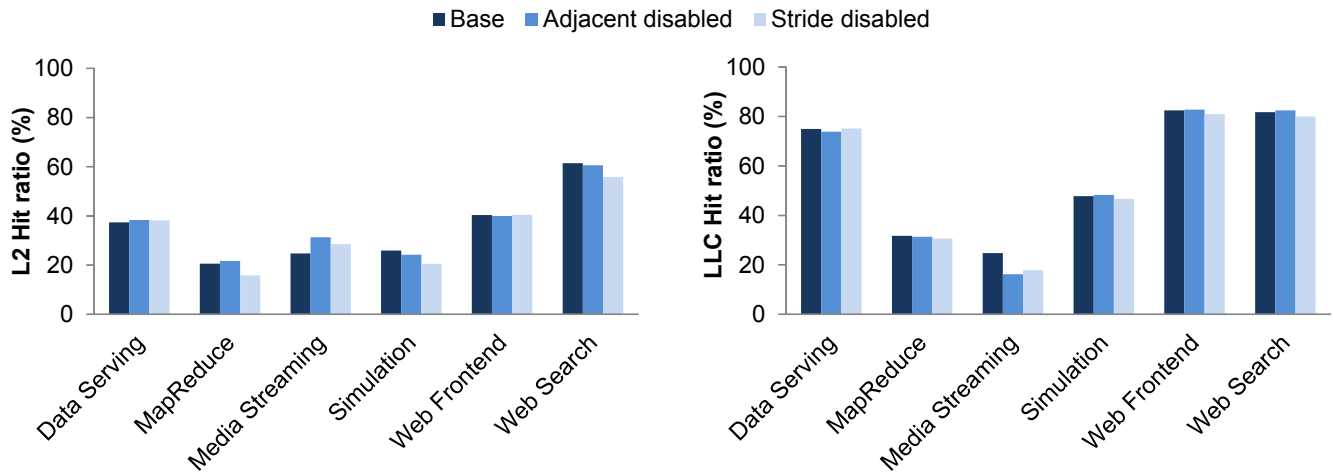


Figure 8. L2 (left) and LLC (right) hit ratios with and without the adjacent-line and stride prefetchers

processor’s demand, thereby hiding the access latency. In Figure 8, we present the cache hit ratios of the L2 and LLC of the base system with all prefetchers enabled, as well as the miss ratios after disabling the prefetchers. We observe that the adjacent-line prefetcher provides no benefit to cloud applications, in some cases marginally increasing the L2 miss rate because it pollutes the cache with an unnecessary adjacent block. The stride prefetcher does not increase the LLC hit ratio, however, it does result in a marginal increase in the L2 hit ratio. Overall, we find that the simple prefetchers used in modern processors are entirely ineffective for cloud applications.

Implications: While modern processors grossly over-provision the memory system, data-center efficiency can be improved by matching the processor design to the needs of the cloud applications. Whereas modern processors dedicate approximately half of the die area to the LLC, cloud applications would likely benefit from a different balance. A two-level cache hierarchy with a modestly sized LLC that makes 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 at the same time freeing up die area and power. The die area and power can be practically applied toward improving compute 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, while more advanced correlating data prefetchers could accurately prefetch complex access data patterns and increase the performance of all cores.

4.4 Bandwidth Inefficiencies

- *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 in core-to-core communication. We investigate the utility of the on-chip interconnect for cloud applications in Figure 9. We plot the fraction of L2 misses that access data most recently written by another thread running in a different physical processor.¹ We break down each bar into *OS* and *Application* components to offer insight into the source of the data sharing.

In general, we observe extremely limited read-write sharing across the cloud applications. We find that the OS-level data sharing is dominated by the network subsystem. Java-based applications (Data Serving, Map Reduce,

1. We explicitly bind application threads to processor cores in two different physical processor sockets on the server motherboard.

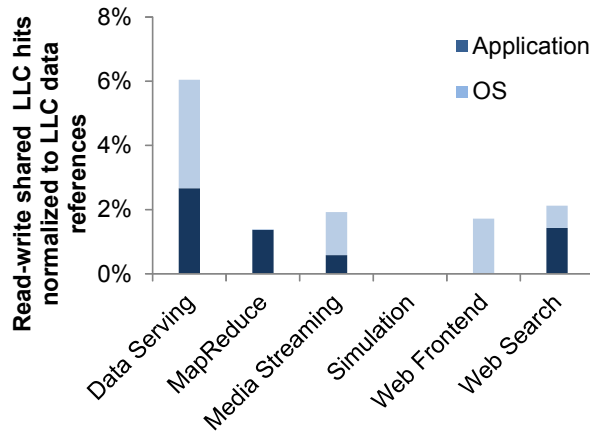


Figure 9. Percentage of LLC data references accessing cache blocks modified by a thread running on another core

and Web Search) exhibit a small degree of sharing from the use of a concurrent garbage collector that may run a collection thread on a remote core, artificially inducing application-level communication. The Media Streaming server application updates several global counters to track the total number of packets sent by the server process; we believe this to be an oversight on the part of the developers of the server software, as reducing the amount of communication by keeping per-thread statistics is trivial and would eliminate the mutex lock and shared-object update.

The low degree of sharing exhibited by most cloud applications indicates that wide and low-latency interconnects used in today’s processors are over-provisioned for cloud applications. Although with a small number of cores the overhead 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 over-provisioned 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 66MHz to dual-data-rate at over 1GHz, raising the peak theoretical bandwidth from 544MB/s to 17GB/s per channel, with the latest server processors having three independent memory channels. We plot the average off-chip bandwidth utilization of cloud applications in Figure 10. Cloud applications experience high off-chip miss rates, however, the MLP of the applications is low due to the complex data structure dependencies, leading to low aggregate off-chip bandwidth utilization even when all cores have outstanding off-chip memory accesses. Of the cloud applications we examine, Media Streaming is the only application that actively uses 38% of the available off-chip bandwidth. However, we note that our applications are configured to stress the processor, demonstrating the worst-case behavior. Our Media Streaming application streams low-bit-rate media to a large number of clients, a worst-case scenario that is unlikely to be observed in real deployment. We therefore conclude that modern processors over-provision off-chip bandwidth for scale-out cloud applications.

Implications: The on-chip interconnect and off-chip memory buses can be scaled back to improve processor efficiency. Because the cloud applications are written for scale-out and 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. Off-chip memory buses can be optimized for cloud applications by scaling back unnecessary bandwidth. Memory controllers consume a large fraction of today’s chip area and memory busses are responsible for a considerable fraction of the system power. Reducing the number of memory channels and the

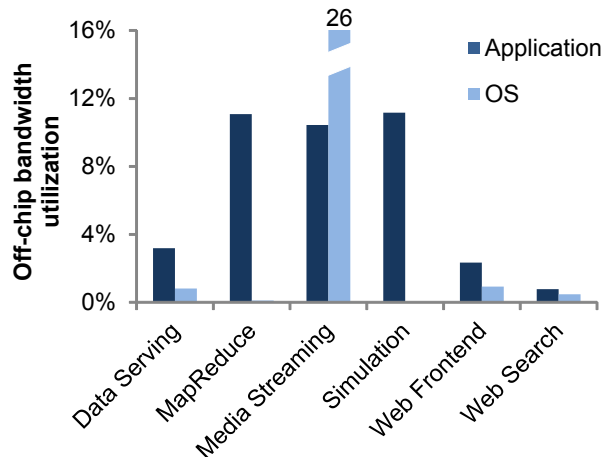


Figure 10. Average off-chip memory bandwidth utilization as a percentage of available off-chip bandwidth

power draw of the memory buses should improve cloud application efficiency without affecting application throughput.

5 RELATED WORK

Previous research has characterized the micro-architectural behavior of traditional commercial server applications when running on modern hardware, using real machines [1, 26] and simulation environments [20, 21, 32, 35]. We include traditional server applications in our study to compare them with scale-out cloud applications and to validate our evaluation framework.

The research community uses the PARSEC benchmark suite to perform experiments with chip multiprocessors [4]. Bienia et al. characterize the PARSEC suite’s working sets and the communication patterns among threads [4]. In our work, we examine the execution time breakdown of PARSEC. Unlike the cloud and traditional server applications, PARSEC has a negligible instruction working set and exhibits a high degree of memory-level parallelism, displaying distinctly different micro-architectural behavior compared with scale-out cloud applications.

Previous research analyzed various performance or power inefficiencies of modern processors running traditional commercial applications [8, 18, 21, 27, 36, 43, 44]. Tuck and Tullsen showed that simultaneous multithreading can improve performance of scientific and engineering applications by 20-25% on a Pentium4 processor [43]. Our results show similar trends for scale-out cloud applications. Kgil et al. [27] show that, for a particular class of throughput-oriented web workloads, modern processors are extremely power-inefficient, arguing that the chip area should be used for processing cores rather than caches. Our results corroborate these findings, showing that, for cloud workloads, the time spent accessing the large and slow last-level caches accounts for more than half of the data stalls [21], calling for resizing and reorganization of the cache hierarchy. We similar conclusions to Davis et al. [8] and Hardavellas et al. [21] who showed that heavily multithreaded in-order cores are more efficient for throughput-oriented workloads compared to aggressive out-of-order cores. Also corroborating prior work, we find that latency-sensitive and computationally intensive web search workloads favor more aggressive processing cores [37]. Ranganathan and Jouppi motivate the need for integrated analysis of micro-architectural efficiency and application service-level agreements in their survey of enterprise information technology trends [36]. To address the power inefficiencies of current general-purpose processors, specialization at various hardware levels has been proposed [18, 44].

As cloud computing has become ubiquitous, there has been significant research activity on characterizing particular cloud applications, either micro-architecturally [37], or at the system level [7, 28, 29, 30, 40]. To the best of our knowledge, our study is the first work to systematically characterizes the micro-architectural behavior of a wide range of cloud services. Kozyrakis et al. [28] presented a system-wide characterization of large-scale online ser-

vices provided by Microsoft and showed the implications of such workloads on data-center server design. Reddi et al. characterized the Bing search engine [37], showing that the computational intensity of search tasks is increasing as a result of adding more machine learning features to the engine. These findings are consistent with our results, which show that web search has the highest IPC among the studied cloud applications.

Much work focused on benchmarking the cloud and datacenter infrastructure. Yahoo! Cloud Serving Benchmark (YCSB) [7] is a framework to benchmark large-scale distributed data serving systems. We include results for the YCSB benchmark and provide its micro-architectural characterization running Cassandra, a popular cloud data serving application. Fan et al. discuss web mail, web search, and map-reduce as three representative workloads present in the Google datacenter [16]. Lim et al. extend this set of benchmarks and add an additional media streaming workload [31]. They further analyze the energy efficiency of a variety of systems when running these applications. Our benchmark suite also includes workloads from these categories.

CloudCmp [29, 30] is a framework to compare cloud providers. using a systematic approach to benchmarking various components of the cloud. Huang et al. analyzed performance and power characteristics of Hadoop clusters using HiBench [23], a benchmark that specifically targets the Hadoop map-reduce framework. Our analytics benchmark (MapReduce) uses the same map-reduce infrastructure. However, we provide the micro-architectural, rather than the system-wide, characterization of the map-reduce applications. For benchmarking modern web technologies, we use CloudStone [39], an open source benchmark that simulates activities related to social events.

6 CONCLUSIONS

Cloud computing has emerged as a dominant computing platform to provide hundreds of millions of users with online services. To support the growing popularity and continue expanding their services, cloud providers must work to overcome the physical space and power constraints limiting data-center growth. While strides have been made to improve data-center efficiency at the rack and chassis-levels, we observe that the predominant processor architecture of today’s data centers is inherently inefficient for running scale-out cloud workloads, resulting in low compute density and poor tradeoffs between performance and energy.

In this work, we used performance counters to analyze the micro-architectural behavior of a wide range of scale-out cloud applications. We analyzed and identified the key sources of area and power inefficiency in the instruction fetch, core micro-architecture, and memory system organization. We then identified the specific needs of cloud applications and suggested the architectural modifications that can lead to dense and power-efficient data-center processor designs in the future. Specifically, our analysis showed that efficiently executing scale-out cloud applications requires optimizing the instruction-fetch path for multi-megabyte instruction working sets, reducing the core aggressiveness and last-level cache capacity to free area and power resources in favor of more cores each with more hardware threads, and scaling back the over-provisioned on-chip and off-chip bandwidth.

7 REFERENCES

- [1] Anastassia Ailamaki, David J DeWitt, Mark D Hill, and David A Wood. DBMSs on a modern processor: Where does time go? In *Proceedings of the 25th International Conference on Very Large Data Bases*, 1999.
- [2] Alexa, The Web Information Company. www.alexacom.com.
- [3] Apache Mahout: Scalable machine-learning and data-mining library. <http://mahout.apache.org/>.
- [4] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. The PARSEC benchmark suite: characterization and architectural implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*, 2008.
- [5] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. Bigtable: a distributed storage system for structured data. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, 2006.
- [6] Liviu Ciortea, Cristian Zamfir, Stefan Bucur, Vitaly Chipounov, and George Candea. Cloud9: a software testing service. *ACM SIGOPS Operating Systems Review*, 43:5–10, January 2010.

- [7] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking cloud serving systems with YCSB. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, 2010.
- [8] John D. Davis, James Laudon, and Kunle Olukotun. Maximizing CMP throughput with mediocre cores. In *Proceedings of the 14th International Conference on Parallel Architectures and Compilation Techniques*, 2005.
- [9] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. In *Proceedings of the 6th Conference on Symposium on Operating Systems Design & Implementation - Volume 6*, 2004.
- [10] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: Amazon’s highly available key-value store. In *ACM SIGOPS Operating Systems Review*, 2007.
- [11] PowerEdge M1000e Blade Enclosure. <http://www.dell.com/us/enterprise/p/poweredge-m1000e/pd.aspx>.
- [12] Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam, and Doug Burger. Dark silicon and the end of multicore scaling. In *Proc. of the 38th annual international symposium on Computer architecture*, 2011.
- [13] EuroCloud Server. <http://www.eurocloudserver.com>.
- [14] Stijn Eyerman, Lieven Eeckhout, Tejas Karkhanis, and James E Smith. A performance counter architecture for computing accurate CPI components. *ACM SIGOPS Operating Systems Review*, 34, October 2006.
- [15] Facebook Statistics. <https://www.facebook.com/press/info.php?statistics>.
- [16] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. Power provisioning for a warehouse-sized computer. In *ACM SIGARCH Computer Architecture News*, 2007.
- [17] Google Data Centers. <http://www.google.com/intl/en/corporate/datacenter/>.
- [18] Rehan Hameed, Wajahat Qadeer, Megan Wachs, Omid Azizi, Alex Solomatnikov, Benjamin C. Lee, Stephen Richardson, Christos Kozyrakis, and Mark Horowitz. Understanding sources of inefficiency in general-purpose chips. In *Proceedings of the 37th Annual International Symposium on Computer Architecture*, 2010.
- [19] Nikos Hardavellas, Michael Ferdman, Anastasia Ailamaki, and Babak Falsafi. Power scaling: the ultimate obstacle to 1k-core chips. In *Technical Report NWU-EECS-10-05, Northwestern University, Evanston, IL*, March 2010.
- [20] Nikos Hardavellas, Michael Ferdman, Babak Falsafi, and Anastasia Ailamaki. Reactive NUCA: near-optimal block placement and replication in distributed caches. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*, 2009.
- [21] Nikos Hardavellas, Ippokratis Pandis, Ryan Johnson, Naju Mancheril, Anastassia Ailamaki, and Babak Falsafi. Database servers on chip multiprocessors: Limitations and opportunities. In *Conf. on Innovative Data Systems Research*, 2007.
- [22] Faban Harness and Benchmark Framework. <http://java.net/projects/faban/>.
- [23] Shengsheng Huang, Jie Huang, Jinqun Dai, Tao Xie, and Bo Huang. The HiBench benchmark suite: Characterization of the MapReduce-based data analysis. In *26th International Conference on Data Engineering Workshops*, 2010.
- [24] IntelV Tune Amplifier XE Performance Profiler. <http://software.intel.com/en-us/articles/intel-vtune-amplifier-xe/>.
- [25] Tejas S Karkhanis and James E Smith. A First-Order superscalar processor model. In *ACM SIGARCH Computer Architecture News*. IEEE Computer Society, 2004.
- [26] Kimberly Keeton, David A Patterson, Yong Qiang He, Roger C Raphael, and Walter E Baker. Performance characterization of a quad pentium pro SMP using OLTP workloads. *ACM SIGARCH Computer Architecture News*, 26:15–26, April 1998.
- [27] Taeho Kgil, Shaun D’Souza, Ali Saidi, Nathan Binkert, Ronald Dreslinski, Trevor Mudge, Steven Reinhardt, and Krisztian Flautner. Picoserver: using 3d stacking technology to enable a compact energy efficient chip

- multiprocessor. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2006.
- [28] Christos Kozyrakis, Aman Kansal, Sriram Sankar, and Kushagra Vaid. Server engineering insights for large-scale online services. *Micro, IEEE*, 30(4):8–19, july-aug. 2010.
- [29] Ang Li, Xiaowei Yang, Srikanth Kandula, and Ming Zhang. CloudCmp: comparing public cloud providers. In *Proc. of the 10th Annual Conference on Internet Measurement*, 2010.
- [30] Ang Li, Xiaowei Yang, Srikanth Kandula, and Ming Zhang. Cloudcmp: shopping for a cloud made easy. In *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, 2010.
- [31] Kevin Lim, Parthasarathy Ranganathan, Jichuan Chang, Chandrakant Patel, Trevor Mudge, and Steven Reinhardt. Understanding and designing new server architectures for emerging Warehouse-Computing environments. In *ACM SIGARCH Computer Architecture News*, 2008.
- [32] Jack L. Lo, Luiz André Barroso, Susan J. Eggers, Kourosh Gharachorloo, Henry M. Levy, and Sujay S. Parekh. An analysis of database workload performance on simultaneous multithreaded processors. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, 1998.
- [33] Open Compute Project. <http://opencompute.org/>.
- [34] Apache HTTP Server Project. <http://httpd.apache.org>.
- [35] Parthasarathy Ranganathan, Kourosh Gharachorloo, Sarita V Adve, and Luiz Andre Barroso. Performance of database workloads on shared-memory systems with out-of-order processors. *ACM SIGOPS Operating Systems Review*, 32:307–318, October 1998.
- [36] Parthasarathy Ranganathan and Norman Jouppi. Enterprise it trends and implications for architecture research. In *Proceedings of the 11th International Symposium on High-Performance Computer Architecture*, 2005.
- [37] Vijay Janapa Reddi, Benjamin C Lee, Trishul Chilimbi, and Kushagra Vaid. Web search using mobile cores: quantifying and mitigating the price of efficiency. In *ACM SIGARCH Computer Architecture News*. ACM, 2010.
- [38] SeaMicro Packs 768 Cores Into its Atom Server. <http://www.datacenterknowledge.com/archives/2011/07/18/seamicro-packs-768-cores-into-its-atom-server/>.
- [39] Will Sobel, Shanti Subramanyam, Akara Sucharitakul, Jimmy Nguyen, Hubert Wong, Arthur Klepchukov, Sheetal Patil, O. Fox, and David Patterson. Cloudstone: Multi-platform, multi-language benchmark and measurement tools for Web 2.0, 2008.
- [40] Vijayaraghavan Soundararajan and Jennifer M. Anderson. The impact of management operations on the virtualized datacenter. In *Proceedings of the 37th Annual International Symposium on Computer Architecture*, 2010.
- [41] The Apache Cassandra Project. <http://cassandra.apache.org/>.
- [42] TPC Transaction Processing Performance Council. <http://www.tpc.org/default.asp>.
- [43] Nathan Tuck and Dean M. Tullsen. Initial observations of the simultaneous multithreading pentium 4 processor. In *Proceedings of the 12th International Conference on Parallel Architectures and Compilation Techniques*, 2003.
- [44] Ganesh Venkatesh, Jack Sampson, Nathan Goulding, Saturnino Garcia, Vladyslav Bryksin, Jose Lugo-Martinez, Steven Swanson, and Michael Bedford Taylor. Conservation cores: reducing the energy of mature computations. In *Proceedings of the 15th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2010.
- [45] Thomas F. Wenisch, Roland E. Wunderlich, Michael Ferdman, Anastassia Ailamaki, Babak Falsafi, and James C. Hoe. Simflex: Statistical sampling of computer system simulation. *IEEE Micro*, 26:18–31, July 2006.