Quantifying the Mismatch between Emerging Scale-Out Applications and Modern Processors

MICHAEL FERDMAN, Stony Brook University
ALMUTAZ ADILEH, ONUR KOCBERBER, STAVROS VOLOS, MOHAMMAD ALISAFAEE, DJORDJE JEVDJIC, CANSU KAYNAK, ADRIAN DANIEL POPESCU, ANASTASIA AILAMAKI, and BABAK FALSAFI, École Polytechnique Fédérale de Lausanne

Emerging scale-out workloads require 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. 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 workloads.

In this work, we introduce CloudSuite, a benchmark suite of emerging scale-out workloads. We use performance counters on modern servers to study scale-out workloads, finding that today's predominant processor microarchitecture is inefficient for running these workloads. We find that inefficiency comes from the mismatch between the workload needs and modern processors, particularly in the organization of instruction and data memory systems and the processor core microarchitecture. Moreover, while today's predominant microarchitecture is inefficient when executing scale-out workloads, we find that continuing the current trends will further exacerbate the inefficiency in the future. In this work, we identify the key microarchitectural needs of scale-out workloads, calling for a change in the trajectory of server processors that would lead to improved computational density and power efficiency in data centers.

Categories and Subject Descriptors: C.4 [**Performance of Systems**]: *Design studies*

General Terms: Design, Measurement, Performance

ACM Reference Format:
Ferdman, M., Adileh, A., Kocberber, O., Volos, S., Alisafaee, M., Jevdjic, D., Kaynak, C., Popescu, A. D., Ailamaki, A., and Falsafi, B. 2012. Quantifying the mismatch between emerging scale-out applications and modern processors. ACM Trans. Comput. Syst. 30, 4, Article 15 (November 2012), 24 pages.
DOI = 10.1145/2382553.2382557 http://doi.acm.org/10.1145/2382553.2382557

1. INTRODUCTION

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 data centers. With the industry rapidly expanding [Facebook 2012], 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

This work was partially supported by EuroCloud, Project No 247779 of the European Commission 7th RTD Framework Programme – Information and Communication Technologies: Computing Systems.
Authors' addresses: M. Ferdman, Stony Brook University, Stony Brook, NY 11794; A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi, École Polytechnique Fédérale de Lausanne, EPFL IC ISIM PARSA, INJ 239 (Bâtiment INJ), Station 14, CH-1015 Lausanne, Switzerland.

DOI 10.1145/2382553.2382557 http://doi.acm.org/10.1145/2382553.2382557

semiconductor manufacturing industry has reached the physical limits of voltage scaling [Hardavellas et al. 2011; Horowitz et al. 2005], 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 [Google 2012; OpenCompute 2012].

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 workloads. Processor vendors use the same underlying architecture for servers and for the general purpose market, leading to extreme inefficiency in today's data centers. Moreover, both general purpose (e.g., Intel and AMD) and traditional server processor (e.g., Sun Niagara, IBM Power7) designs follow a trajectory that benefits scale-up workloads, a trend that was established long before the emergence of scale-out workloads. Recognizing the space and power inefficiencies of modern processors for scale-out workloads, some vendors and researchers conjecture that even using processors built for the mobile space may be more efficient [EuroCloud 2012; Kgil et al. 2006; Reddi et al. 2010; SeaMicro 2011].

In this work, we observe that scale-out workloads share many inherent characteristics that place them into a distinct workload class 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 over-provisioning of the memory hierarchy and core microarchitectural resources for the scale-out workloads. Moreover, continuing the current processor trends will result in further widening the mismatch between the scale-out workloads and server processors. Conversely, we find that the characteristics of scale-out workloads can be leveraged to gain area and energy efficiency in future servers.

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

— *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- and Memory-Level Parallelism in Scale-Out Workloads is 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-Chip and Off-Chip Bandwidth Requirements of Scale-Out Workloads Are Low.* Scale-out workloads see no benefit from fine-grained coherence and high memory and core-to-core communication bandwidth.

2. MODERN CORES AND SCALE-OUT WORKLOADS

Today's data centers are built around conventional desktop processors whose architecture was designed for a broad 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 [Esmaeilzadeh et al. 2011; Hameed et al. 2010; Hardavellas et al. 2011; Venkatesh et al. 2010], 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.

2.1. Dominant Processor Architectures

Modern 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 scaling 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 into two roughly equal parts, with one part dedicated to the cores and tightly-coupled private caches and the other dedicated to a large shared last-level cache. The emergence of multicore processors offered the possibility to run computationally intensive multithreaded 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 cache access. While OoO execution can improve core resource utilization through instruction-level parallelism (ILP), the core's complexity 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. This functionality 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 OoO execution 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 referenced memory addresses. To maintain low access latency, the first-level cache capacity must remain small due to hardware constraints, whereas the last-level caches can have larger capacity, but with a higher access latency. 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 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 multithreading (SMT), enabling multiple 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 Workloads

To find a set of applications that dominate today's cloud infrastructure, we examined a selection of internet services based on their popularity [Alexa 2012]. For each popular service, we analyzed the class of application software used by the major 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 scale-out workloads have similar characteristics; all applications we examined (1) operate on large data sets that are split across a large number of machines, typically into memory-resident shards, (2) serve large numbers of completely independent requests that do not share any state, (3) have application software designed specifically for the cloud infrastructure where unreliable machines may come and go, and (4) use intermachine connectivity only for high-level task management and coordination.

Data Caching. Popular online applications rely on several tiers of servers. These services commonly have a frontend server that accepts the client requests and cooperates with backend servers to provide responses to the service users. The bulk of the dataset resides in the backend servers, where highly-specialized software is deployed to organize and maintain the dataset and to serve the frontend read and write requests to specific pieces of the data.

Frequently, the computational complexity and persistence requirements of the backend servers do not permit sufficiently high throughput or sufficiently low latency to operate large-scale services. In such cases, an additional layer of in-memory data caching servers is used to maintain a subset of the hot data, enabling high performance by quickly servicing the majority of the requests from an in-memory cache and avoiding sending these requests to the more complex backend servers. Although traditional data caching servers predominantly observed read requests, the rapid evolution and adoption of caching servers has resulted in their adaptation for other uses with higher frequency of write requests. Today, in addition to reducing the load on the backend data servers, data caching servers are also used for server-side storage of non-critical objects that do not require persistence in case of failure; for example, web client session data is frequently stored in the caching servers without saving the data to permanent storage at the backend.

Data Serving. The requirements of the backing stores in global-scale online applications differ from the requirements of databases deployed in traditional transaction processing applications (e.g., banking systems). The larger scale of client base results in higher demand for service throughput. The actions and contents generated by the large number of users results in a large volume of mostly unstructured data to be stored and processed in a timely manner. The data consistency requirements are usually more relaxed for online services. Moreover, popular services continuously add more features to enhance their services and to attract more clients. To meet all these demands, service operators started adopting non-traditional databases to achieve the scalability, speed, and flexibility requirements of their services.

Various *NoSQL* data stores [Cassandra 2012; Chang et al. 2006; DeCandia et al. 2007] have been explicitly designed to serve as the backing store for large-scale web applications such as the Facebook inbox, Google Earth, and Google Finance, providing fast and scalable storage with varying and rapidly evolving storage schemas. The NoSQL systems split hundreds of terabytes of data into shards and horizontally scale out to large cluster sizes. Most operations on requested objects are performed using indexes that support fast lookup and key range scans. For simplicity and scalability, these systems are designed to support queries that can be completely executed by a single machine, with all operations that require combining data from multiple shards relegated to the middleware.

MapReduce. Data is getting generated at extremely high rates from nontraditional sources (e.g., social media, blogs, sensors, and media streams) and in varying forms (e.g., text files, click streams, videos, sensor data, and log files). Finding insights in the available data opens opportunities to achieve significant benefits, such as business profits and security improvements, and to even answer analytics questions that were previously considered beyond reach. Finding those insights requires automated analytical processing to cluster, classify, and filter big data.

The map-reduce paradigm [Dean and Ghemawat 2004] has emerged as a popular approach to large-scale analysis, farming out requests to a cluster of machines 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 algorithms [Mahout 2012]. Users implement algorithms using *map* and *reduce* functions and provide these functions 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 to thousands of independent servers, communication between tasks is typically limited to reading and writing files in a distributed file system. For example, map tasks produce temporary files that are subsequently read by the reduce tasks, effectively rendering all map and reduce tasks architecturally independent. Whenever possible, the map-reduce infrastructures strive to further reduce communication by scheduling the processing of data on the servers that store that data.

Text analysis has emerged as one of the popular tasks applied at various stages of big data analysis such as filtering operations (e.g., log analysis), targeted crawling, spam identification, and sentiment analysis. As a typical machine learning task, text classification involves building a model of the data being classified and then using this model to classify the dataset using a cluster of map-reduce nodes.

Media Streaming. The availability of high-bandwidth connections to home and mobile devices has made media streaming services such as NetFlix, YouTube, and YouKu ubiquitous. The process of streaming the packetized video contents and the protocols involved in the streaming process vary among the service providers and client devices. For example, while the desktop version of YouTube streams contents over HTTP, the mobile version (i.e., m.youtube.com) uses RTSP (real-time streaming protocol). Regardless of the protocol details, all streaming service providers 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 diverse client base.

The various media streaming methods involve common tasks, starting with the clients sending requests to a frontend/load-balancer that accepts the client requests and balances the load on the various backend streaming servers. Once the connection is established between the streaming server and the client, the server starts to gradually send the necessary frames at the rate requested by the client, until the video duration is finished or according to the client interactions with the server (e.g., terminate, skip forward, or rewind).

To improve quality of service, the media dataset is both sharded and replicated among many servers. Sharding of media content ensures that servers frequently send the same content to multiple users, enabling in-memory caching of content by the servers. While in-memory caching is effective, the on-demand unicast nature of today's streaming services practically guarantees that the streaming server will work on a different piece of the media file for each client, even when concurrently streaming the same file to many clients.

SAT Solver. The ability to temporarily allocate, almost infinite compute resources in the cloud without purchasing the infrastructure has created an opportunity for engineers and researchers to conduct large-scale computations, which would be otherwise considered impractical. Cloud9 [Ciortea et al. 2010], provides software testing as a service in the cloud, relying on symbolic execution algorithms. Such complex algorithms become tractable when the computation is split into smaller subproblems and distributed within the cloud where a large number of SAT Solver processes are hosted.

Cloud environments offer dynamic and heterogeneous resources that are loosely connected over an IP network. The Cloud9 large-scale computation tasks are therefore adapted to a worker-queue model with centralized load balancing that rebalances 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.

Web Frontend. Accessing web pages has been the backbone of all internet-based services. Traditional web applications followed a model where the service published a mostly-static page that would be accessed by many users. The interactions between the users and the page contents were minimal. However, web serving technologies have seen critical changes over the past several years. For example, the wide adoption of Web 2.0 concepts has led to a richer user experience (e.g., widgets and interactive web applications) where online services are dominated by user-generated contents. Interactive web applications result in a higher number of smaller dynamic requests and affect the way web servers are designed and optimized.

Popular online services rely on fast, reliable, and scalable web frontend servers to provide their clients with a high-quality experience. Although a variety of web software stacks are used in datacenters, the underlying service architectures are similar. A load balancer distributes independent client requests (e.g., read and write request) across a large number of stateless web servers. The frontend web servers cooperate with the backend tiers (e.g., NoSQL or traditional databases) to serve pages that include dynamically retrieved and frequently changing contents.

Web Search. Web search engines, such as those powering Google and Microsoft Bing, have become fundamental to web users' experience. Although the major search engine operators compete using different technologies, the fundamental service architecture and tasks involved in a search operation are similar; search engines comprise

Table I. Architectural Parameters

Processor	32nm Intel Xeon X5670, operating at 2.93 GHz.
CMP width	6 OoO cores
Core width	4-wide issue and retire
Reorder buffer	128 entries
Load/Store buffer	48/32 entries
Reservation stations	36 entries
L1 cache	32KB, split I/D, 4-cycle access latency
L2 cache	256KB per core, 6-cycle access latency
LLC (L3 cache)	12MB, 29-cycle access latency
Memory	24GB, 3 DDR3 channels, delivering up to 32GB/s

several stages of nodes, starting with a frontend that accepts the user queries and ending with index serving nodes (ISN) that contain the dataset (i.e., a web index). The dataset is harvested from web pages using a background crawling process. The crawling process fetches web pages, extracts the terms from the fetched text, and creates an inverted index that maps each term to a list of web pages.

To support a large number of concurrent latency-sensitive search queries against the index, the data is split into memory-residents shards, with each index serving node (ISN) responsible for processing requests to its own shard. Hundreds of unrelated search requests are handled by each ISN every second, with minimal locality; shards are therefore sized to fit into the memory of the ISNs to avoid reducing throughput and degrading quality of service due to disk I/O. For performance scalability, ISNs may be replicated, allowing the frontend to load balance requests among multiple ISNs that are responsible for the same shard. ISNs communicate with the frontend machines and not with other ISNs. In an ISN, each request is handled by a software thread, without communicating with other threads.

A search operation begins when a user submits a search query to a frontend server. The frontend distributes the query to the ISNs. Each ISN is responsible for performing the query against its index shard and returning a list with the top N addresses (based on the service's specific ranking algorithm) to the frontend server. The frontend is then responsible for merging and sorting the addresses received from all ISNs. After selecting the top-ranked addresses, the frontend collects text captions for each of the top addresses from the corresponding ISNs. Finally, the frontend generates an HTML page that contains the top results formatted with their captions and sends the result page to the user.

3. METHODOLOGY

We conduct our study on a PowerEdge M1000e enclosure [Dell 2012] 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 cache hierarchy: the L1 and L2 caches are private to each core, while the LLC (L3) 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 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, two gigabit NICs are used in each blade. Table I summarizes the key architectural parameters of the systems.

In this work, we introduce CloudSuite, a benchmark suite of emerging scale-out workloads. We present a characterization of the CloudSuite workloads alongside traditional benchmarks, including desktop and engineering (SPEC2006), parallel (PARSEC), enterprise web (SPECweb09), and relational database server (TPC-C, TPC-E, Web Backend) workloads. For all but the TPC-E benchmark, we use CentOS 5.5 with the 2.6.32 Linux kernel. For the TPC-E benchmark, we use Microsoft Windows Server 2008 R2.

3.1. Measurement Tools and Methodology

We analyze architectural behavior using Intel VTune [Intel 2012], a tool that provides an interface to the processor performance counters. For all scale-out and traditional server workloads except SAT Solver, we perform a 180-second measurement after the workload completes the ramp-up period and reaches a steady state. Because SAT Solver has no steady-state behavior, we achieve comparable results across runs by reusing input traces. We create SAT Solver input traces that take 45 minutes to execute in our baseline setup and use the first 30 minutes as warmup and the last 15 minutes as the measurement window. We measure the entire execution of the PARSEC and SPEC CPU2006 applications.

For our evaluation, we limit 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 QoS requirements. To ensure that all application and operating system software runs on the cores under test, we disable all unused cores using the available operating system mechanisms.

We note that computing a breakdown of the execution-time stall components of superscalar out-of-order processors cannot be performed precisely due to overlapped work in the pipeline [Eyerman et al. 2006; Keeton et al. 1998]. We present execution-time breakdown results based on the performance counters that have no overlap. However, we plot memory cycles side-by-side rather than in a stack to indicate the potential overlap with cycles during which instructions were committed. We compute the memory cycles as a sum of the cycles when an off-core request was outstanding, instruction stall cycles contributed by L2 instruction hits, second-level TLB miss cycles, and the first-level instruction TLB miss cycles. Because data stalls can overlap other data stalls, we compute the number of cycles when an off-core request was outstanding using MSHR occupancy statistics,[1] which measure the number of cycles when there is at least one L2 miss being serviced. We do not include L1-D and L2 data cache hits in the computation because they can be effectively hidden by the out-of-order core [Karkhanis and Smith 2004].

We perform 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 100% hit ratio in the LLC, effectively reducing the cache capacity available for the workload running on the remaining cores of the same processor.

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.

[1]In the processors we study, we measure the occupancy statistics of the *super queue* structure, which tracks all accesses that miss in the L1 caches.

3.2. Scale-Out Workload Experimental Setup

Data Caching. We benchmark the Twemcache 2.5.0 memory object caching system (a scalable variant of the popular Memcached software) with a 2GB Yahoo! Cloud Serving Benchmark (YCSB) dataset. Server load is generated using the YCSB 0.1.3 client [Cooper et al. 2010] that sends requests following a Zipfian distribution with a 95:5 read to write request ratio using 1KB objects.

Data Serving. We benchmark the Cassandra 0.7.3 database with a 15GB Yahoo! Cloud Serving Benchmark (YCSB) dataset. Server load is generated using the YCSB 0.1.3 client [Cooper et al. 2010] that sends requests following a Zipfian distribution with a 95:5 read to write request ratio. Cassandra is configured with a 7GB Java heap and 400MB new-generation garbage collector space.

MapReduce. We benchmark a node of a four-node Hadoop 0.20.2 cluster, running the Bayesian classification algorithm from the Mahout 0.4 library [Mahout 2012]. The algorithm attempts to guess the *country* tag of each article in a 4.5GB set of Wikipedia pages. One *map* task is started per core and assigned a 2GB Java heap.

Media Streaming. We benchmark the Darwin Streaming Server 6.0.3, serving videos of varying duration, using the Faban driver [Faban 2012] to simulate the clients. We limit our setup to low bit-rate video streams to shift stress away from network I/O.

SAT Solver. We benchmark one instance per core of the Klee SAT Solver, an important component of the Cloud9 parallel symbolic execution engine [Ciortea et al. 2010]. Input traces for the engine are produced by Cloud9 by symbolically executing the command-line *printf* utility from the GNU CoreUtils 6.10 using up to four 5byte and one 10-byte symbolic command-line arguments.

Web Frontend. We benchmark a frontend machine serving Olio, a Web 2.0 web-based social event calendar. The frontend machine runs Nginx 1.0.10 with PHP 5.3.5 and APC 3.1.8 PHP opcode cache. We generate a backend dataset using the Cloudstone benchmark [Sobel et al. 2008] and use the Faban driver [Faban 2012] to simulate clients. To limit disk space requirements, we generate a 12GB on-disk file dataset and modify the PHP code to always serve files from the available range.

Web Search. We benchmark an index serving node (ISN) of the distributed version of Nutch 1.2/Lucene 3.0.1 with an index size of 2GB and data segment size of 23GB of content crawled from the public internet. We mimic real-world setups by making sure that the search index fits in memory, eliminating page faults and minimizing disk activity [Reddi et al. 2010]. We simulate clients using the Faban 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 Benchmark Experimental Setup

PARSEC 2.1. We benchmark the official applications with the *native* input, reporting results averaged across all benchmarks. We present results averaged into two groups, CPU-intensive (cpu) and memory-intensive (mem) benchmarks.

SPEC CPU2006. We benchmark the official applications, both integer and floating point, running with the first reference input, reporting results averaged across all benchmarks. Similarly to PARSEC, we separately present CPU-intensive (cpu) and memory-intensive (mem) applications.

SPECweb09. We benchmark the e-banking workload running on the Nginx 1.0.1 web server with the FastCGI PHP 5.2.6 interpreter and APC 3.0.19 PHP opcode cache. We disable connection encryption (SSL) when running SPECweb09 to allow for better comparison to the Web Frontend workload.

TPC-C. We benchmark the TPC-C workload on a commercial enterprise database management system (DBMS). The database load is generated by 32 clients configured with zero think time and running on a separate machine. The TPC-C database has 40 warehouses. The DBMS is configured with a 3GB buffer pool and direct I/O.

TPC-E. We benchmark the TPC-E 1.12 workload on a commercial enterprise database management system (DBMS). The client driver runs on the same machine, bound to a core that is not used by the database software. The TPC-E database contains 5000 customer records (52GB). The DBMS is configured with a 10GB buffer pool.

Web Backend. We benchmark a machine executing the database backend of the Web Frontend benchmark presented previously. The back-end machine runs the MySQL 5.5.9 database engine with a 2GB buffer pool.

3.4. I/O Infrastructure

Data-intensive scale-out workloads and traditional database server 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 overprovisions the I/O subsystem to avoid an I/O bottleneck. To avoid bringing up disk arrays containing hundreds of disks and flash devices, as it is traditionally done with large-scale database installations [TPC 2012], we construct a network-attached iSCSI storage array by creating large RAM disks in separate machines and connect the machine under test to the iSCSI storage via the high-speed Ethernet network. This approach places the entire dataset of our applications in the memory of the remote machines, creating an illusion of a large disk cluster with extremely high I/O bandwidth and low latency.

4. RESULTS

We begin exploring the microarchitectural behavior of scale-out workloads through 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. Overlapped with the execution-time breakdown, we show the *Memory* cycles bar, which approximates the number of cycles when the processor could not commit instructions due to outstanding long-latency memory accesses.

The execution-time breakdown of scale-out workloads is dominated by stalls in both application code and operating system. Notably, most of the stalls in scale-out workloads arise due to 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% of the cycles and experience only a fraction of the stalls due to memory accesses. Furthermore, although the execution-time breakdown of some scale-out workloads (e.g., MapReduce and SAT Solver) appears similar to the memory-intensive PARSEC and SPEC2006 benchmarks, the nature of the stalls of these workloads is different. Unlike the scale-out workloads, many PARSEC and

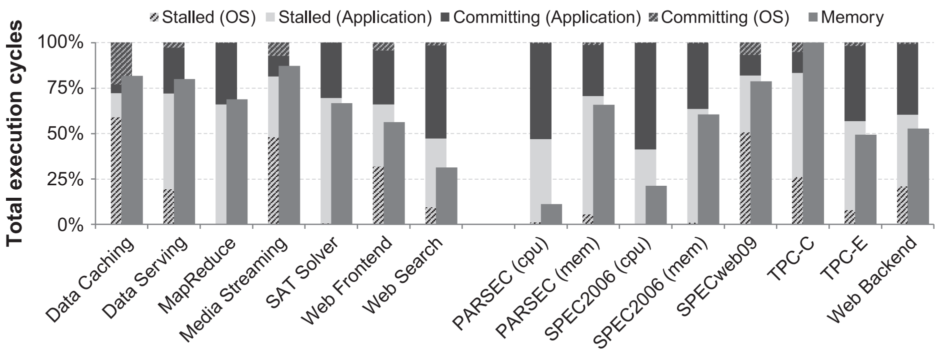


Fig. 1. Execution-time breakdown and memory cycles of scale-out workloads (left) and traditional benchmarks (right).

SPEC2006 applications frequently stall due to 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 (TPCC, TPCE, and Web Backend). However, we observe that scale-out workloads differ considerably from traditional online transaction processing (TPC-C) which spends over 80% of the time stalled due to dependent memory accesses.[2] 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 from the Web Frontend workload, representative of modern scale-out configurations. While the traditional web workload is dominated by serving static files and a small number of 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 the majority 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 (e.g., SPECweb09 and TPC-C) also cannot cater to scale-out workloads. In the rest of this section, we provide a detailed analysis of the inefficiencies of running scale-out workloads 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 critical role in processor 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

[2]In addition to instructions and data, TPC-C includes 14% Request-For-Ownership memory cycles. These accesses are not on the critical path, but are part of the memory cycles that appear in Figure 1.

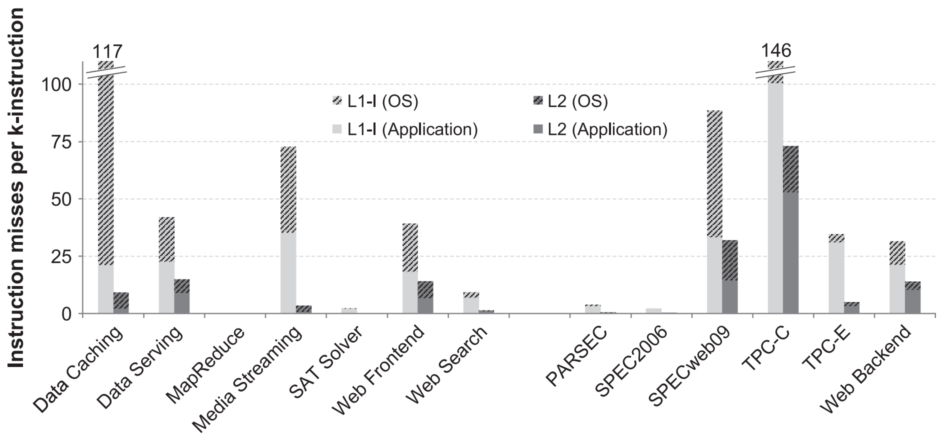


Fig. 2. L1-I and L2 instruction miss rates for scale-out workloads (left) and traditional benchmarks (right). The OS bars present the instruction cache miss rates of the workloads that spend significant time executing operating system code.

the core real-estate and power consumption are entirely wasted for the cycles that the frontend spends fetching instructions.

Figure 2 presents the L1-I and L2 instruction miss rates of our workloads. We find that, in contrast to desktop and parallel benchmarks, the instruction working sets of many scale-out workloads considerably exceed the L1-I cache capacity, 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 note that the operating system behavior differs between the scale-out workloads and traditional server systems. Although many scale-out workloads actively use the operating system, the functionality exercised by the scale-out workloads is more restricted. Both L1-I and L2 operating system instruction cache miss rates across the scale-out workloads are lower compared to traditional server workloads, indicating a smaller operating system instruction working set in scale-out workloads.

Stringent access-latency requirements of the L1-I caches preclude increasing the size of the caches to capture the instruction working set of many scale-out workloads, which is an order of magnitude larger than the caches found in modern processors. We find that modern processor architectures cannot tolerate the latency of L1-I cache misses, avoiding frontend 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-I capacity experiences a significant shortfall and cannot 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 retrieving 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 large uniform access latency, it contributes an unnecessarily large instruction-fetch penalty (29 cycles to access the 12MB cache).

Implications. To improve efficiency and reduce frontend 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 cores [Hardavellas et al. 2009] is likely to considerably reduce 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 prefetchers, high instruction-cache miss rates and significant frontend 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 non-sequential 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.

4.2. Core Inefficiencies

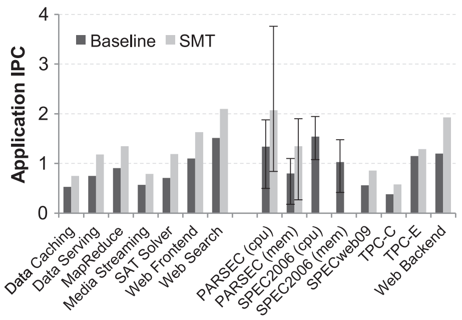
— *Low ILP precludes effectively using the full core width*
— *ROB and LSQ are underutilized due to low MLP*

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. 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.

Modern processors use 4-wide cores that can decode, issue, execute, and commit up to four instructions on each cycle. However, in practice, instruction-level parallelism (ILP) is limited by dependencies. The *Baseline* bars in Figure 3 (left) show the average number of instructions committed per cycle when running on an aggressive 4-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 workloads that can benefit from wide cores exist, 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 significant benefit.

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. The *Baseline* bars in Figure 3 (right) 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



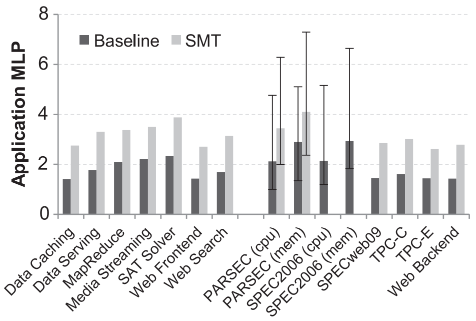


Fig. 3. Application instructions committed per cycle for systems with and without SMT out of maximum IPC of 4 (left) and memory-level parallelism for systems with and without SMT (right). Range bars indicate the minimum and maximum of the corresponding group.

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.

Support for 4-wide out-of-order execution with a 128-instruction window and up to 48 outstanding memory requests requires multiple-branch prediction, numerous ALUs, forwarding paths, many-port register banks, large instruction schedulers, highly-associative reorder buffers (ROB) and load-store queues (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 of 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. Entire buildings are packed with aggressive cores, designed to commit 4 instructions per cycle with over 48 outstanding memory requests, but executing applications with an average IPC of 0.8 and average MLP of 1.9. Moreover, current industry trends point toward 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 multithreading (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 IPC and MLP of an SMT-enabled core in Figure 3 using the bars labeled *SMT*. As expected, the MLP found and exploited by the cores when two independent application threads run on each core concurrently is nearly doubled 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 39-69% improvements in IPC.

Implications. The nature of scale-out workloads makes them ideal candidates to exploit multithreaded multicore architectures. Modern mainstream processors offer excessively complex cores, resulting in inefficiency through waste of resources. At the same time, our results corroborate prior work [Reddi et al. 2010], indicating that niche processors offer excessively simple (e.g., in-order [Davis et al. 2005; Kgil et al. 2006]) cores that cannot leverage the available ILP and MLP in scale-out workloads. We find that scale-out workloads 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 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 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 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 last-level cache (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 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 due to semiconductor manufacturing improvements. We investigate the utility of growing the LLC capacity for scale-out workloads in Figure 4. We plot the average system performance[3] of scale-out workloads as a function of the LLC capacity, normalized to a baseline system with a 12MB LLC. Unlike in the memory-intensive desktop applications (e.g., SPEC2006 mcf), we find minimal performance sensitivity to LLC size above 4-6MBs in scale-out and traditional server workloads. The LLC captures the instruction working sets of scale-out workloads, which are less than two megabytes. Beyond this point, small shared supporting structures may consume another one to two megabytes. 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.

Our results show that the on-chip resources devoted to the LLC are one of the key limiters of scale-out 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 scale-out application requirements by 2-3×, 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 [Hardavellas et al. 2007] 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

[3]User-IPC has been shown to be proportional to application throughput [Wenisch et al. 2006]. We verified this relationship for the scale-out workloads.

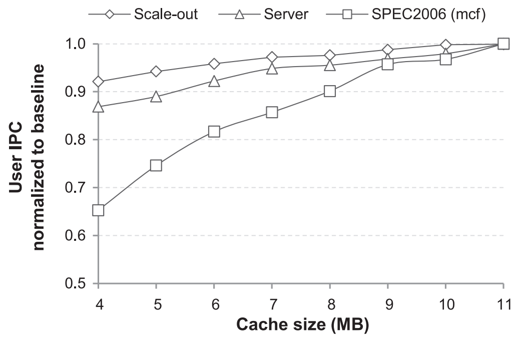


Fig. 4. Performance sensitivity to LLC.

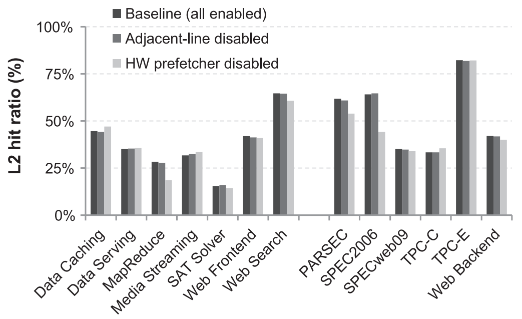


Fig. 5. L2 hit ratios of a system with enabled and disabled adjacent-line and HW prefetchers.

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 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 HW 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, SAT Solver). Finally, we note that the DCU streamer (not shown) provides no benefit to scale-out workloads, in some cases marginally increasing the L2 miss rate because it pollutes the cache with unnecessary blocks.

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 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 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 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 intercore communication.

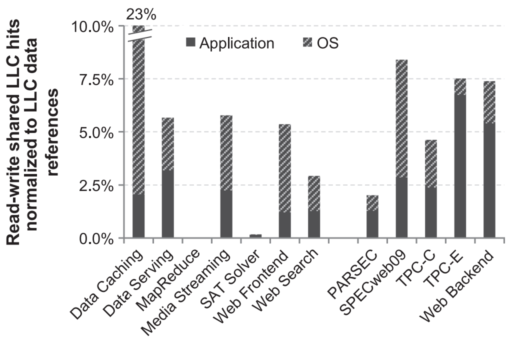


Fig. 6. Percentage of LLC data references accessing cache blocks modified by a thread running on a remote core.

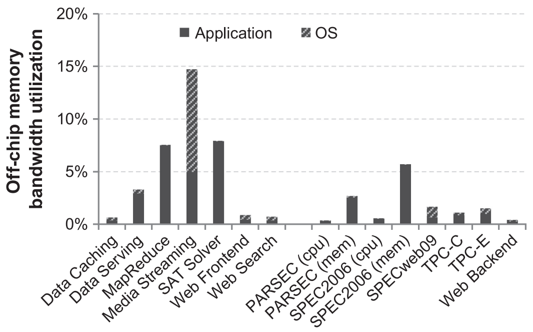


Fig. 7. Average off-chip memory bandwidth utilization as a percentage of available off-chip bandwidth.

Multithreaded 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. 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 *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. Multithreaded 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. We note that 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 over-provisioned 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 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 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 non-negligible 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. Among the scale-out workloads we examine, Media Streaming is the only application that uses up to 15% of the available off-chip bandwidth. However, we note that our applications are configured to stress the processor, actually demonstrating the worst-case behavior. Overall, we find that modern processors significantly over-provision off-chip bandwidth for scale-out workloads.

Implications. 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. Off-chip memory buses can be optimized for scale-out workloads by scaling back unnecessary bandwidth. 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.

5. RELATED WORK

Previous research has characterized the microarchitectural behavior of traditional commercial server applications when running on modern hardware, using real machines [Ailamaki et al. 1999; Keeton et al. 1998] and simulation environments [Hardavellas et al. 2007, 2009; Lo et al. 1998; Ranganathan et al. 1998]. We include traditional server applications in our study to compare them with scale-out workloads and to validate our evaluation framework.

The research community uses the PARSEC benchmark suite to perform experiments with chip multiprocessors [Bienia et al. 2008]. Bienia et al. [2008] characterize the PARSEC suite's working sets and the communication patterns among threads. In our work, we examine the execution time breakdown of PARSEC. Unlike the scale-out and traditional server applications, PARSEC has a negligible instruction working set and exhibits a high degree of memory-level parallelism, displaying distinctly different microarchitectural behavior compared to scale-out workloads.

Previous research analyzed various performance or power inefficiencies of modern processors running traditional commercial applications [Davis et al. 2005; Hameed et al. 2010; Hardavellas et al. 2007; Kgil et al. 2006; Ranganathan and Jouppi 2005; Tuck and Tullsen 2003]. Tuck and Tullsen [2003] showed that simultaneous multithreading can improve performance of scientific and engineering applications by 20–25% on a Pentium4 processor. Our results show similar trends for scale-out workloads. Kgil et al. [2006] 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. A similar observation has been made in the GPU domain [Guz et al. 2012]. Our results corroborate these findings, showing that, for scale-out workloads, the time spent accessing the large and slow last-level caches accounts for more than half of the data stalls [Hardavellas et al. 2007], calling for resizing and reorganizing the cache hierarchy. We draw similar conclusions to Davis et al. [2005] and Hardavellas et al. [2007] who showed that heavily multithreaded in-order cores are more efficient for throughput-oriented workloads compared to aggressive out-of-order cores. Ranganathan and Jouppi [2005] motivate the need for integrated analysis of microarchitectural efficiency and application service-level agreements in their survey of enterprise information technology trends. To address the power inefficiencies of current general-purpose processors, specialization at various hardware levels has been proposed [Hameed et al. 2010; Venkatesh et al. 2010].

As cloud computing has become ubiquitous, there has been significant research activity on characterizing particular scale-out workloads, either microarchitecturally [Reddi et al. 2010; Tang et al. 2011], or at the system level [Cooper et al. 2010; Kozyrakis et al. 2010; Li et al. 2010a, 2010b; Soundararajan and Anderson 2010]. To the best of our knowledge, our study is the first work to systematically characterize the microarchitectural behavior of a wide range of cloud services. Kozyrakis et al. [2010] presented a system-wide characterization of large-scale online services provided by Microsoft and showed the implications of such workloads on data-center server design. Reddi et al. [2010] characterized the Bing search engine, 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 scale-out workloads.

Much work focused on benchmarking the cloud and datacenter infrastructure. The Yahoo! Cloud Serving Benchmark (YCSB) [Cooper et al. 2010] is a framework to benchmark large-scale distributed data serving systems. We include results for the YCSB benchmark and provide its microarchitectural characterization running on Cassandra, a popular cloud database. Fan et al. [2007] discuss web mail, web search, and map-reduce as three representative workloads present in the Google datacenter. Lim et al. [2008] extend this set of benchmarks and add an additional media streaming workload. They further analyze the energy efficiency of a variety of systems when running these applications. Our benchmark suite similarly includes workloads from these categories.

CloudCmp [Li et al. 2010a, 2010b] is a framework to compare cloud providers using a systematic approach to benchmarking various components of the infrastructure. Huang et al. [2010] analyzed performance and power characteristics of Hadoop clusters using HiBench, a benchmark that specifically targets the Hadoop map-reduce framework. Our MapReduce benchmark uses the same infrastructure. However, we provide the microarchitectural, rather than a system-wide, characterization of the map-reduce workload. For benchmarking modern web technologies, we use CloudStone [Sobel et al. 2008], 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 microarchitecture of today's data centers is inherently inefficient for running scale-out workloads, resulting in low compute density and poor tradeoffs between performance and energy.

In this work, we used performance counters to analyze the microarchitectural behavior of a wide range of scale-out workloads. We analyzed and identified the key sources of area and power inefficiency in the instruction fetch, core microarchitecture, and memory system organization. We then identified the specific needs of scale-out workloads 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 workloads 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.

ACKNOWLEDGMENTS

The authors would like to thank the anonymous reviewers for their feedback and suggestions. 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 the SAT Solver benchmarks. We thank the DSLab for their assistance with Cloud9, Emre Özer and Rustam Miftakhutdinov for their feedback and suggestions, and Aamer Jaleel and Carole Jean-Wu for their assistance with understanding the Intel architecture and configuration. We thank the EuroCloud project partners for advocating and inspiring the CloudSuite benchmark suite.

APPENDIX A

This appendix describes the performance counter methodology used in Section 4. We break down events into Application and OS components, by considering kernel, network, filesystem, and other OS-related modules as part of the OS (i.e., vmlinux, bnx2, nfs, sep3_4, ext3, libiscsi, iscsi_tcp).

A.1. Execution-Time Breakdown and Memory Cylces

Components	Methodology
Stalled	UOPS_RETIRED.STALL_CYCLES
Committing	UOPS_RETIRED.ACTIVE_CYCLES
Memory	The sum of the following sub-components: LLC+Mem stalls: OFFCORE_REQUESTS_OUTSTANDING.ANY.READ_NOT_EMPTY
	Instruction stalls on L2: (6 / 35) * (L2_RQSTS.IFETCH_HIT / L2_RQSTS.IFETCH_MISS * OFFCORE_REQUESTS_OUTSTANDING.DEMAND.READ_CODE_NOT_EMPTY) DTLB misses that result in page walks: DTLB_LOAD_MISSES.WALK_CYCLES ITLB misses that result in page walks: ITLB_MISSES.WALK_CYCLES ITLB misses that hit in the second level of TLB: 10 * (ITLB_MISSES.ANY - ITLB_MISSES.WALK_COMPLETED)

Breakdown of the Memory component:

(1) The performance counter used in calculating the LLC+Mem stalls gives the number of cycles during which there is at least one outstanding L2 miss request (instruction or data).

(2) We calculate instruction stalls on L2 by scaling the stalls on LLC+Mem instruction accesses by a factor of (L2_fetch_latency * L2_instruction_hits) / (LLC_fetch_latency * L2_instruction_misses).
(3) We calculate stalls on ITLB misses that hit in the second-level TLB by multiplying the number of these events with the access latency of the second-level TLB.

A.2. L1 and L2 Instruction Miss Rate

Components	Methodology
L1-I miss rate	1000 * L1I.MISSES / INST_RETIRED.ANY
L2-I miss rate	L1-I miss rate * L2_RQSTS.IFETCH_MISS / (L2_RQSTS.IFETCH_HIT + L2_RQSTS.IFETCH_MISS)

A.3. Application IPC and MLP Per Hardware Context

Components	Methodology
Application IPC	INST_RETIRED.ANY / CPU_CLK_UNHALTED.THREAD
Application MLP	OFFCORE_REQUESTS_OUTSTANDING.DEMAND.READ_DATA / OFFCORE_REQUESTS_OUTSTANDING.DEMAND.READ_DATA_NOT_EMPTY

Comments:

(1) We exclude OS events as explained in Section A.
(2) The first performance counter used in calculating the application MLP gives the total number of outstanding L2 data miss requests. The second performance counter gives the number of cycles that there is at least one outstanding L2 data miss request.

A.4. Performance Sensitivity to Last-Level Cache (LLC)

We perform a cache sensitivity analysis by dedicating two cores to cache-polluting threads. The polluter threads traverse arrays of predetermined size in a pseudo-random 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 100% hit ratio in the LLC, effectively reducing the cache capacity available for the workload running on the remaining cores of the same processor.

Components	Methodology
User IPC	INST_RETIRED.ANY / CPU_CLK_UNHALTED.THREAD

Comments:

(1) We define User IPC as the fraction of the number of instructions committed by the application and the number of total cycles (include both application and OS).
(2) The first performance counter includes application instructions and the second performance counter includes both application and OS cycles.

A.5. L2 Hit Ratios of a System with Enabled and Disabled Adjacent-Line and HW Prefetchers

Components	Methodology
L2 hit ratio	MEM_LOAD_RETIRED.L2_HIT / (MEM_LOAD_RETIRED.L2_HIT + MEM_LOAD_RETIRED.LLC_MISS + MEM_LOAD_RETIRED.LLC_UNSHARED_HIT + MEM_LOAD_RETIRED.OTHER_CORE_L2_HIT_HITM)

Comment: The metric includes both application and OS events.

A.6. Percentage of LLC Data References Accessing Cache Blocks Modified by Other Core

Components	Methodology
LLC remote hits normalized to LLC data references	MEM_UNCORE_RETIRED.REMOTE_HITM / (MEM_LOAD_RETIRED.LLC_MISS + MEM_LOAD_RETIRED.LLC_UNSHARED_HIT + MEM_LOAD_RETIRED.OTHER_CORE_L2_HIT_HITM)

Comment: The numerator includes either application events or OS events. The denominator includes both application and OS events.

A.7. Off-Chip Bandwidth Utilization

Components	Methodology
BW requirements of one core in bytes/second	(64 * 2.926 * 10^9) * MEM_LOAD_RETIRED.LLC_MISS / CPU_CLK_UNHALTED.THREAD

Comment: We multiply the number of LLC misses per cycle with the number of bytes fetched (64 bytes) and the frequency of the processor in Hertz.

REFERENCES

AILAMAKI, A., DEWITT, D. J., HILL, M. D., AND WOOD, D. A. 1999. DBMSs on a modern processor: Where does time go? In *Proceedings of the 25th International Conference on Very Large Data Bases*.

ALEXA. 2012. The Web Information Company. http://www.alexa.com/.

BIENIA, C., KUMAR, S., SINGH, J. P., AND LI, K. 2008. The PARSEC benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*.

CASSANDRA. 2012. The Apache Cassandra Project. http://cassandra.apache.org/.

CHANG, F., DEAN, J., GHEMAWAT, S., HSIEH, W. C., WALLACH, D. A., BURROWS, M., CHANDRA, T., FIKES, A., AND GRUBER, R. E. 2006. Bigtable: A distributed storage system for structured data. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation*, vol. 7.

CIORTEA, L., ZAMFIR, C., BUCUR, S., CHIPOUNOV, V., AND CANDEA, G. 2010. Cloud9: A software testing service. *ACM SIGOPS Operating Systems Review 43*, 5–10.

COOPER, B. F., SILBERSTEIN, A., TAM, E., RAMAKRISHNAN, R., AND SEARS, R. 2010. Benchmarking cloud serving systems with YCSB. In *Proceedings of the 1st ACM Symposium on Cloud Computing*.

DAVIS, J. D., LAUDON, J., AND OLUKOTUN, K. 2005. Maximizing CMP throughput with mediocre cores. In *Proceedings of the 14th International Conference on Parallel Architectures and Compilation Techniques*.

DEAN, J. AND GHEMAWAT, S. 2004. MapReduce: Simplified data processing on large clusters. In *Proceedings of the 6th Conference on Symposium on Operating Systems Design and Implementation*, vol. 6.

DECANDIA, G., HASTORUN, D., JAMPANI, M., KAKULAPATI, G., LAKSHMAN, A., PILCHIN, A., SIVASUBRAMANIAN, S., VOSSHALL, P., AND VOGELS, W. 2007. Dynamo: Amazon's highly available key-value store. In *Proceedings of the 21st ACM SIGOPS Symposium on Operating Systems Principles*.

DELL. 2012. PowerEdge M1000e Blade Enclosure. http://www.dell.com/us/enterprise/p/poweredge-m1000e/pd.aspx.

ESMAEILZADEH, H., BLEM, E., AMANT, R. S., SANKARALINGAM, K., AND BURGER, D. 2011. Dark silicon and the end of multicore scaling. In *Proceeding of the 38th Annual International Symposium on Computer Architecture*.

EUROCLOUD. 2012. EuroCloud Server. http://www.eurocloudserver.com.

EYERMAN, S., EECKHOUT, L., KARKHANIS, T., AND SMITH, J. E. 2006. A performance counter architecture for computing accurate CPI components. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*.

FABAN. 2012. Faban Harness and Benchmark Framework. http://java.net/projects/faban/.

FACEBOOK. 2012. Facebook Statistics. https://www.facebook.com/press/info.php?statistics.

FAN, X., WEBER, W.-D., AND BARROSO, L. A. 2007. Power provisioning for a warehouse-sized computer. In *Proceedings of the 34th Annual International Symposium on Computer Architecture*.

GOOGLE. 2012. Google Data Centers. http://www.google.com/intl/en/corporate/datacenter/.

GUZ, Z., ITZHAK, O., KEIDAR, I., KOLOD, A., MENDELSON, A., AND WEISER, U. C. 2012. Threads vs. Caches: Modeling the behavior of parallel workloads. In *Proceedings of the International Conference on Computer Design*.

HAMEED, R., QADEER, W., WACHS, M., AZIZI, O., SOLOMATNIKOV, A., LEE, B.C., RICHARDSON, S., KOZYRAKIS, C., AND HOROWITZ, M. 2010. Understanding sources of inefficiency in general-purpose chips. In *Proceedings of the 37th Annual International Symposium on Computer Architecture*.

HARDAVELLAS, N., PANDIS, I., JOHNSON, R., MANCHERIL, N., AILAMAKI, A., AND FALSAFI, B. 2007. Database servers on chip multiprocessors: Limitations and opportunities. In *The 3rd Biennial Conference on Innovative Data Systems Research*.

HARDAVELLAS, N., FERDMAN, M., FALSAFI, B., AND AILAMAKI, A. 2009. Reactive NUCA: Near-optimal block placement and replication in distributed caches. In *Proceedings of the 36th Annual International Symposium on Computer Architecture*.

HARDAVELLAS, N., FERDMAN, M., AILAMAKI, A., AND FALSAFI, B. 2011. Toward Dark Silicon in Servers. *IEEE Micro 31*, 4, 6–15.

HOROWITZ, M., ALON, E., PATIL, D., NAFFZIGER, S., KUMAR, R., AND BERNSTEIN, K. 2005. Scaling, power, and the future of CMOS. In *Electron Devices Meeting, 2005. IEDM Technical Digest. IEEE International*.

HUANG, S., HUANG, J., DAI, J., XIE, T., AND HUANG, B. 2010. The HiBench benchmark suite: Characterization of the MapReduce-based data analysis. In *Proceedings of the 26th International Conference on Data Engineering Workshops*.

INTEL. 2012. Intel VTune Amplifier XE Performance Profiler. http://software.intel.com/en-us/articles/intel-vtune-amplifier-xe/.

KARKHANIS, T. S. AND SMITH, J. E. 2004. A first-order superscalar processor model. In *Proceedings of the 31st Annual International Symposium on Computer Architecture*.

KEETON, K., PATTERSON, D. A., HE, Q. Y., RAPHAEL, R. C., AND BAKER, W. E. 1998. Performance characterization of a quad Pentium Pro SMP using OLTP workloads. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*.

KGIL, T., D'SOUZA, S., SAIDI, A., BINKERT, N., DRESLINSKI, R., MUDGE, T., REINHARDT, S., AND FLAUTNER, K. 2006. 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*.

KOZYRAKIS, C., KANSAL, A., SANKAR, S., AND VAID, K. 2010. Server engineering insights for large-scale online services. *IEEE Micro 30*, 4, 8–19.

LI, A., YANG, X., KANDULA, S., AND ZHANG, M. 2010a. CloudCmp: Comparing public cloud providers. In *Proceedings of the 10th Annual Conference on Internet Measurement*.

LI, A., YANG, X., KANDULA, S., AND ZHANG, M. 2010b. CloudCmp: Shopping for a cloud made easy. In *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*.

LIM, K., RANGANATHAN, P., CHANG, J., PATEL, C., MUDGE, T., AND REINHARDT, S. 2008. Understanding and designing new server architectures for emerging warehouse-computing environments. In *Proceedings of the 35th Annual International Symposium on Computer Architecture*.

LO, J. L., BARROSO, L. A., EGGERS, S. J., GHARACHORLOO, K., LEVY, H. M., AND PAREKH, S. S. 1998. An analysis of database workload performance on simultaneous multithreaded processors. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*.

MAHOUT. 2012. Apache Mahout: Scalable machine-learning and data-mining library. http://mahout.apache.org/.

OPENCOMPUTE. 2012. Open Compute Project. http://opencompute.org/.

RANGANATHAN, P., GHARACHORLOO, K., ADVE, S. V., AND BARROSO, L. A. 1998. Performance of database workloads on shared-memory systems with out-of-order processors. In *Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems*.

RANGANATHAN, P. AND JOUPPI, N. 2005. Enterprise IT trends and implications for architecture research. In *Proceedings of the 11th International Symposium on High-Performance Computer Architecture*.

REDDI, V. J., LEE, B. C., CHILIMBI, T., AND VAID, K. 2010. Web search using mobile cores: Quantifying and mitigating the price of efficiency. In *Proceedings of the 37th Annual International Symposium on Computer Architecture*.

SEAMICRO. 2011. SeaMicro Packs 768 Cores Into its Atom Server. http://www.datacenterknowledge.com /archives/2011/07/18/seamicro-packs-768-cores-into-its-atom-server/.

SOBEL, W., SUBRAMANYAM, S., SUCHARITAKUL, A., NGUYEN, J., WONG, H., KLEPCHUKOV, A., PATIL, S., FOX, A., AND PATTERSON, D. 2008. Cloudstone: Multi-platform, multi-language benchmark and measurement tools for web 2.0. In *Proceedings of the 1st Workshop on Cloud Computing and Its Applications*.

SOUNDARARAJAN, V. AND ANDERSON, J. M. 2010. The impact of management operations on the virtualized datacenter. In *Proceedings of the 37th Annual International Symposium on Computer Architecture*.

TANG, L., MARS, J., VACHHARAJANI, V., HUNDT, R., AND SOFFA, M. L. 2011. The impact of memory subsystem resource sharing on datacenter applications. In *Proceeding of the 38th Annual International Symposium on Computer Architecture*.

TPC. 2012. Transaction Processing Performance Council. http://www.tpc.org/.

TUCK, N. AND TULLSEN, D. M. 2003. Initial observations of the simultaneous multithreading Pentium 4 processor. In *Proceedings of the 12th International Conference on Parallel Architectures and Compilation Techniques*.

VENKATESH, G., SAMPSON, J., GOULDING, N., GARCIA, S., BRYKSIN, V., LUGO-MARTINEZ, J., SWANSON, S., AND TAYLOR, M. B. 2010. Conservation cores: Reducing the energy of mature computations. In *Proceedings of the 15th International Conference on Architectural Support for Programming Languages and Operating Systems*.

WENISCH, T. F., WUNDERLICH, R. E., FERDMAN, M., AILAMAKI, A., FALSAFI, B., AND HOE, J. C. 2006. Simflex: Statistical sampling of computer system simulation. *IEEE Micro 26*, 18–31.

Received July 2012; accepted September 2012