Research statement - Architectural Considerations for Big Data

Gunjae Koo

1. Architectural Research for Big Data Applications

Architectural research on high-performance data processing systems is becoming more critical in Big Data era as demands for huge volumes of data processing grow rapidly. In order to meet the performance requirements by modern big data applications, modern datacenter servers accommodate parallel processors (such as GPGUs and Xeon Phi) to accelerate data processing throughput, and advanced storage devices for higher data transfer bandwidth.

My PhD research focuses on the architectural study for supporting high-performance data computing systems from storage to processor core levels as summarized in Figure 1. In order to tackle the performance bottleneck by high data movement cost from the storage devices, I worked on active storage systems that exploit computation power available in the embedded storage processors. I proposed *Summarizer* – a dynamic near data processing framework, which improves the performance of data-intensive applications by offloading data computation dynamically to the embedded storage processors [1]. In addition I explored an in-storage indexing mechanism for database management system (DBMS) [2], and a graph-semantic aware storage which actively manages large graph structures in storage systems [3]. In the processor architecture level, I worked on improving the performance of big data analytics applications running on graphics processing units (GPUs). I proposed an efficient data cache management approach which can improve the utilization of small data cache in GPUs [4]. I also investigated an accurate prefetch method working on modern GPU architecture [5].

1.1 Storage system architecture for big data analytics

Data movement cost from storage devices to compute nodes is extremely high for modern data processing applications. As such big data applications that rely storage resident data pay significant fraction of their execution time on the data input/output (I/O) time. Recently high-performance datacenter servers deploy solid state drives (SSDs) which enclose NAND flash memory as primary storage media to provide lower data transfer latency. Such non-volatile memory (NVM)-based storage systems equip general-purpose embedded processors to support NVM managements, hence this computation potential by the embedded storage processor can facilitate active storage systems, which enables computing near storage or application-specific data management.

**Dynamic near data processing framework:** Near data processing (NDP) is a promising computation model which can reduce data movement cost by computing near storage instead of transferring data to host compute nodes. As the performance of the embedded processors in SSDs steadily improves, NDP has become a viable option. However, the SSD embedded processors have limited computation capability, and the availability of even the small amount of computing resource is not consistent. Moving large amounts of general purpose computing near storage may reduce the data transfer cost, but it may incur significant performance penalties by placing undue burden on the wimpy embedded processors. Some of previous solutions have tackled this issue by offloading only simple functions on the storage processors, but such a manual fine tuning makes NDP unpalatable for many use cases. In order to solve these challenges I proposed a dynamic NDP framework – *Summarizer* [1]. Summarizer is a firmware architecture and a programming model for NDP. Summarizer firmware supports opportunistic NDP control, which executes a set of user-defined offloaded functions when the computation resources are available near storage. Evaluation on the industry-strength SSD reference platform reveals Summarizer’s dynamic NDP control improves the performance by up to 20% for data analytics applications.

**In-storage indexing mechanism:** In big data analytics indexing is a common approach to reach the specific data that is needed for requested data computation. Thus indexing is an effective method to reduce host processing for database scan and data communication from the storage systems. However, index maintenance generates heavy computation load and data traffic, and this index maintenance cost increases exponentially as database size grows. In addition index structures themselves need one or more I/O accesses before reaching the desired data. In order to resolve these issues
on the conventional host-side index structures, I proposed FLIXR, a page-level in-storage index mechanism [2]. FLIXR utilizes the existing FTL page-mapping structures to automatically create and organize data indexes. FLIXR provides a programming model to create indexing rules, and a set of API to define index lookup operations to be offloaded to SSD embedded processors thereby reducing index-related data movement.

**Graph semantic aware storage:** Graph data processing is a critical component of broad range of data analytics applications such as social networks, page ranking and traffic analysis. Large graph structures that represent relations of massive domains consist of billions of vertices and more edges. As huge volumes of graph data cannot reside in memory space, graph data transfer from storage devices is a significant performance bottleneck. Many large graphs are sparse and are stored in compressed formats of adjacency matrix such as compressed sparse row (CSR) format. This type of compressed formats utilize in-direct accesses to memory and storage, which lead to extra accesses to graph data. In order to reduce graph data access overheads, we proposed graph semantic aware SSD (GraphSSD) [3]. GraphSSD supports direct translation from vertex ids to page data address in FTL. We also proposed the efficient graph data format and the enhanced graph storage access commands to benefit from the graph translation on storage.

### 1.2 Parallel processor architecture for big data analytics

GPUs employ a large number of concurrent thread contexts to support massive parallel computation with single instruction and multiple thread (SIMT) architecture. Thus modern datacenter servers and scientific computing platforms deploy GPUs to handle huge computation demands from wide range of data-intensive applications. Provisioning sufficient compute cores and large unified register files, GPUs support quick context switching among available threads to mitigate pipeline stalls by long latency memory operation. However irregular memory accesses frequently observed in data analytics applications are critical performance bottlenecks since the context switching mechanism of SIMT architecture cannot cover long data fetch delays. In addition GPU’s memory system suffers from inefficient resource use since burst data requests from irregular data accesses lead to significant congestion in memory hierarchy.

In order to reveal the critical performance impacts by the diverged memory accesses, I investigated the characteristics of individual data fetch instructions observed in wide range of GPU applications [6]. I classified the load instructions in the GPU applications based on the memory address generation approaches, then I showed there is a significant performance impact disparity between different classes of data fetch instructions. This research also revealed the performance of GPU memory hierarchy is significantly influenced by the critical load instructions, which frequently consume up hardware resources of memory subsystem. I found such types of data requests are frequently observed in data analytics applications ported to GPU domains. Hence, optimizing GPU memory system for such critical memory operation can be a promising approach to improve the overall performance of GPUs.

**Access pattern-aware cache management:** Inefficient use of data cache is one of critical performance issues in GPU memory hierarchy. GPU data cache shared across hundreds of concurrent threads suffers significant cache contention and premature data eviction. Strong data locality frequently observed in data analytics applications is not utilized well in the local data cache since GPU data cache is designed to support regular address accesses. In order to improve the utilization of GPU data cache, I proposed Access Pattern-aware Cache Management (APCM), which applies locality-specific cache management approaches based on the data locality type of each global load instruction [4]. I made the observation that GPU applications exhibit strong data access similarity across all threads from the same kernel code. Using the dedicated tag array hardware, APCM dynamically detects the data access history from one selected warp. Then APCM selectively applies cache bypassing and cache pinning to all other warps based on the detected data locality type of each load. APCM gains 34% performance improvement for cache sensitive applications and on average 22% improvement for all types of applications. Combined with previously proposed warp throttling methods APCM significantly outperforms the state-of-the-art warp-based cache management schemes.

**GPU prefetcher:** Albeit GPUs are supposed to be tolerant to long latency of data fetch, our observations revealed SIMT pipeline stalls by memory operation is critical for GPU performance [7]. Prefetch can be one of prominent approaches to hide this long latency of load instructions. I proposed CTA-Aware Prefetcher and Scheduler (CAPS), which is an accurate and efficient prefetching approach for GPUs [5]. Due to GPU’s unique software execution model that assigns large array data computation to thousands of concurrent threads, prefetch on GPU architecture can be effective when prefetch requests is issued for demand requests of other threads. However, regular strides observed among threads within a cooperative thread array (CTA) are broken at CTA boundaries, thus estimating accurate prefetch addresses across CTAs is a challenge. CAPS hoists the computation of the base address of each CTA and strides between two neighbor warps by adjusting warp scheduling. Using the modified warp scheduler and inter-thread prefetcher, CAPS is able to issue timely and accurate prefetches.
2. Future Research Directions - Reinventing Computing Systems for Big Data

Recent rapid growth of data size demanded for big data analytics applications raises new challenges to computer architecture research. As data size exceeds memory scaling, data movement cost from storage to compute nodes has become more dominant. In addition, exascale data processing demands require high data transfer bandwidth and parallel data processing architecture. Some of these challenges have been tackled by my PhD research for smart storage systems and parallel architecture. By leveraging my knowledge and research experiences on computer architecture and systems, I will focus on reinventing data-centric computing systems.

The primary challenges on big data computing are extremely high data movement costs and high data processing throughput demands. I believe the backbone of the future data-centric computing systems is active data computation near storage thereby reducing data movement cost. Hence, as shown in Figure 2 the data compute processors can be tightly coupled with storage media (flash memory in the figure) along with the accelerator and other non-volatile memory. That means the storage platform can be expanded as an active data computing systems for big data processing. I believe following three thrusts are prominent approaches that reinforce this future data computing systems. I believe my PhD research towards storage systems and parallel processor architecture and industry research experiences towards digital systems design give me solid foundations for these research directions.

**Thrust 1: data processing accelerator near storage**

My PhD research has investigated several innovations in the storage systems, however, as mentioned in my prior work presented in [1] lower computation capability in the storage embedded processors can be performance hurdle in the data computing systems. Recent research has focused on accelerate architecture that improves the performance of application-specific functions by using co-processors connected via off-chip interconnection network. With the acceleration co-processors data processing can be improved, however, data movement cost is still high since required data for processing acceleration is transferred from host system’s main memory, and this resident data in the main memory is fetched from storage.

I believe this high data movement cost can be curtailed if the data computing systems accommodate accelerator units which directly accesses data from storage media. This accelerator architecture tightly coupled with storage media has several merits. First, data movement cost in storage systems is relatively small compared to the off-chip data transfer via PCIe interconnections. In addition the host processors do not need to manage complex data transfer between main memory and accelerator buffers. Second, accelerator units in the storage computing systems can utilize larger data bandwidth from storage media. Currently the internal bandwidth of SSDs is restricted by the external data bandwidth of storage interface. However, the storage data computing systems can provide wider internal data bandwidth if large fraction of data requests is consumed by accelerator units. Namely the active data computing systems can process large volumes of data with the accelerator units, and then deliver the final results of smaller size.

**Thrust 2: data-centric processor architecture**

Modern big data analytics applications require high data processing throughput, therefore this demand motivates needs of data-centric processor architecture. Current memory hierarchy is designed to exploit data locality observed in general purpose applications and low access latency from fast memory components. However, strong data locality is not frequently observed in modern big data analytics applications. In order to support high data processing throughput demands, reinventing parallel processing and memory hierarchy architecture is desirable. Low data processing latency is another important feature of the data-centric processor architecture. My PhD study reveals long data fetch latency is critical for parallel processor architecture. It is necessary to compute fetched data quickly in order to mitigate performance degradation by long latency data fetch. Hence, I propose a simple memory subsystem architecture applicable to the data-centric computing architecture. This memory subsystem consists of different types of data buffers based on data locality types – streaming and reused data. The address generation unit can be tightly coupled with DRAM or NVMEM controllers to provide high data bandwidth. This feature may allow more efficient memory access command management by preventing overhead of page conflicts and read/write interventions.

I believe my research focus on parallel processor architecture and memory hierarchy can help to investigate this research
proposal. Also through my internship research at Intel I explored next-generation memory interfaces and memory controller issues, which can be helpful to investigate the memory systems in the data-centric computing architecture.

**Thrust 3: non-volatile memory hierarchy**

Advances in non-volatile memory (NVM) technology achieve higher cell density and lower access latency in various degrees. For instance contemporary multi-bit cell and vertical NAND technology contributes to increase data storage density, thus modern flash memory packages can store more data in the same package area. This means the data storage cost of the NAND flash memory has descended continuously. However, flash memory supports only page-level accesses, and access latency of the flash memory is still much slower compared to DRAM. High data erasure cost is also one of critical performance challenges. On the other hand advanced NVM technology such as 3DXpoint supports byte-addressable accesses with much lower read/write latency. However, the storage cost of 3DXpoint memory is much higher compared to NAND flash memory. Consequently NVM hierarchy combining advanced and cost-effective technology is applicable to boost data accessibility in the data-centric computing systems. For instance the storage computing systems can utilize the advanced NVM as special-purpose storage space to manage key values, indexes and data structure information. I believe my research towards SSD development platforms has prepared me with knowledge of NVM to explore this new memory hierarchy architecture.

**Conclusion**

I believe that research on above proposed topics can contribute together to achieve my research vision – **reinventing computing systems for big data applications**. The proposed research topics can be primarily applied to reinvent storage systems as active near data processing systems that can accelerate big data applications. Furthermore, the proposed research ideas can also deployed to renovate processor architecture. For instance, high-density non-volatile memory hierarchy directly coupled with the host processor can be a promising approach to boost big data processing performance significantly.

**3. Other Research Interests**

I had an opportunity to work on other architectural research projects in both CPU and GPU domains during my academic career at USC and internship at Intel. Before pursuing PhD degree at USC, I primarily worked on system-on-chip (SoC) design research in industry. Research experiences in industry had broaden my knowledge on digital hardware architecture and systems design, FPGA verification platform design, and system firmware programming. The breadth of knowledge I gained will be valuable cornerstone of my future research.

**Advanced memory access control**: While working as a research intern at Intel, I had an opportunity to investigate the memory controller architecture for high-performance server processors coupled with the next-generation DRAM. Based on the performance analysis using a wide range of datacenter server application traces, I was able to figure out performance issues that lead to high data access latency. I proposed several solutions to alleviate read/write interventions, which leads to lower memory read latency.

**Energy efficient computing**: Power consumption is a critical cost issue in high-performance parallel processors. Huge register files in GPU consume significant static and dynamic power. *Warped-compression* is an approach to reduce the number of activated register banks using the simple data compression [8, 9]. Both dynamic and static power can be saved with the proposed idea by applying bank-level power gating to the unused register banks. I contributed to hardware modeling of the data compression and decompression logic.
References


