# Tightly-integrated CPU-GPU architectures

with a detailed case study of analytic database workloads

Jason Power

Preliminary Examination Document

10am October 24, 2014

Computer Sciences Building room 4310

## 1 Introduction

New data is generated at an alarming rate, as much as 2.6 exabytes are created each day [25]. Users want to run complex queries on this abundance of data, sometimes with real-time constraints. This puts a great strain on our computational systems. Today we use large distributed systems to meet these demands. These data centers accounted for 1.1–1.5% of electricity use worldwide and 1.7–2.2% in the U.S in 2010 [20]. As the demand for data and processing capability increases, we must find a way to accomplish these computations more energy efficiently.

Prior to the the early 2000's, due primarily to Moore's law [26], computer architects were able to provide the necessary hardware to keep up with data scaling. In fact, because of Dennard scaling [12] processors were able to double in performance without increasing the total power dissipated. Thus, in addition to increasing performance, each new processor generation was twice as energy efficient (i.e., twice as many operations per joule).

However, because of the breakdown in threshold voltage scaling, and other technological hurdles, computer architects can no longer rely on Moore's law and Dennard scaling to increase energy efficiency [6]. For applications to continue to scale both their data footprint and query complexity without an exponential increase in power, computer architects need to take a radically different approach than in the era of strong Dennard scaling.

Luckily, there are many promising directions for future energy efficient computing. One solution used in many areas today is specializing the processor for a single or set of related tasks. Examples of specialization include digital signal processors, video encoding and decoding units, and graphics processing units (GPUs). GPUs have shown great promise in solving many problems with data-level parallelism efficiently, leading to an explosion of general-purpose GPU (GPGPU) computing.

GPUs have quickly evolved from domain-specific fixed function units to first-class general-purpose compute platforms. Today, almost all systems, from desktops and laptops to mobile phones and tablets have a GPU. GPUs have become more integrated with the system, and today can even be programmed using the same virtual address space as the CPU [9].

In this work, I aim to decrease the energy required for data-intensive workloads. Specifically, my work is broken into three parts.

1. Increase the programmability and efficiency of tightly-coupled CPU-GPU systems by using hardware to accelerate sharing the same virtual address space between the CPU and GPU (Section 3).

2. Increase the energy efficiency of an important class of database workloads, analytic queries, by leveraging current GPU hardware (Section 4).

3. Propose the analytic database machine (ADBM) a new system architecture which couples the CPU and GPU with DRAM to increase the efficiency of analytic database workloads (Section 5).

This document is organized as follows: First, I present background on both GPUs and the analytic database framework used in the latter two parts of my work. Then, I present the three parts of my work, first Section 3 covers enabling x86-64 address translation on GPUs, Section 4 covers my implementation of GPU analytic query processing, BitWarp, and its results, and Section 5 covers my proposed work to create an analytic database machine, ADBM. Section 6 presents a schedule to complete my thesis and the other work I have completed. Finally, Section 7 concludes.

## 2 Background

This section presents a brief background on GPU architecture and programming interface relevant to the rest of the document. Additionally, this section describes the workload and framework used to evaluate the efficacy of GPUs and my proposed database machine on analytic query processing.

### 2.1 GPU Architecture and Programming Interface

**GPU Architecture** — Graphic processing units (GPUs) were originally created to accelerate rendering graphics to the screen; however, in recent years they have become general purpose compute platforms. These general-purpose GPUs (GPGPUs) have a number of characteristics that differ from conventional CPUs:

- GPGPUs are optimized for instruction throughput, not instruction latency.
- GPGPUs have high-bandwidth and low-capacity on-chip caches.
- GPGPUs can exhibit much higher memory-level parallelism than CPUs.
- GPGPUs devote a higher percentage of area to compute functional units than CPUs.

Figure 1 shows an overview of a modern heterogeneous system where both a CPU and GPU are contained on the same silicon chip. The heterogeneous chip contains multiple CPU cores and multiple GPU compute units (CUs).

Within each CU, there are a large number of processing elements (PEs) that are simple functional units. These processing elements are controlled in lock-step by a shared instruction fetch and scheduling unit. Due to their lock-step and in-order nature, the front-end (instruction fetch and issue) of GPU CUs is less complex and more power efficient than the CPU core front-end.

In addition to these features, each CU of the GPU also contains two special memory system optimizations. First, between the PEs and the L1 data cache is a *coalescing unit* that takes the memory addresses generated by each of the PEs and attempts to coalesce them into the minimum number of memory references (e.g., if all 64 address are consecutive and aligned the coalescer will generate four 64-byte requests instead of 64 4-byte requests). Application performance depends on good coalescing behavior, especially for memory-bound workloads.

The second GPU-specific memory optimization is the addition of a directly addressed scratchpad cache (called local memory or group memory by AMD and shared memory by NVIDIA). This small (∼64 KB) cache is explicitly controlled by the programmer. Requests to the scratchpad cache do not need to access cache tags since the scratchpad is directly addressed. Memory requests to the scratchpad do not pass through the coalescer and do not access the low-bandwidth cache tags, and, therefore, the scratchpad cache is higher bandwidth than the data cache for requests that cannot be coalesced. It is common to load data into the scratchpad cache that has a poor data access pattern to avoid the high cost of uncoalesced accesses to the main memory system.

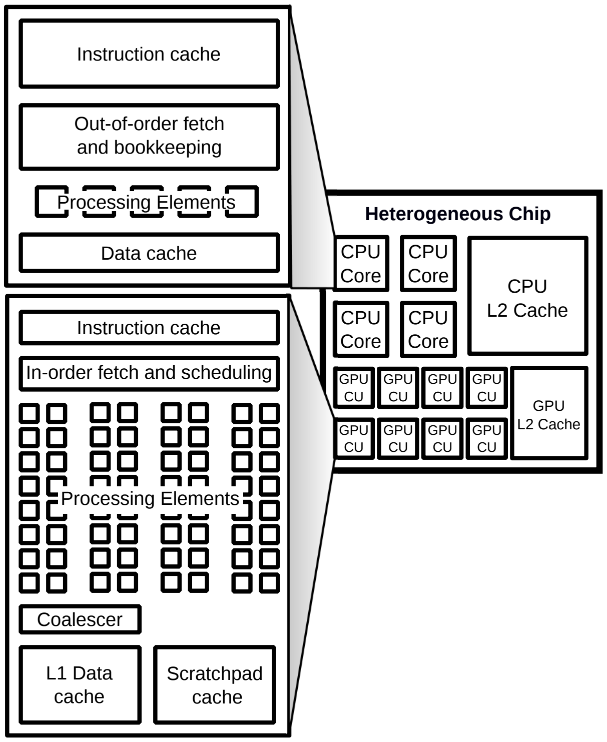

**Figure 1:** GPU architectural overview.

## 2.2 Analytic Database Workload

We use TPC-H as a proxy of analytic database workloads [39]. TPC-H is a decision support benchmark that examines large amounts of data with complex queries meant to model business questions. TPC-H consists of 22 queries and the total size of the database is configurable. The TPC-H benchmark is an industry standard test of database performance with many organizations submitting TPC-H performance results on real systems.

**BitWeaving and WideTable** — To test analytic database workloads on new architectures, I leverage a recently published database framework (WideTable [22]) and algorithm (BitWeaving [21]). WideTable is an in-memory database system which uses BitWeaving as its scan algorithm. Below is a brief overview of each.

WideTable is an in-memory database framework for accelerating analytic queries [22]. In WideTable, when the database is initially loaded, it is denormalized—converted from using multiple tables with pointers between them to a single large table, possibly with duplicated data. To accomplish this denormalization, an outer-join is performed on all database tables. This denormalization allows most queries to be completed with simple scan operations instead of requiring join operations. In a scan operation, each record is read sequentially and a field is evaluated against a predicate (e.g., "`SELECT salary WHERE salary > $100,000`" scans the whole database for records in which the salary is more than $100,000). Additionally, WideTable stores the data in columns instead of rows which increases the memory access locality for scan-like operations. In column-oriented formats, instead of storing the record contiguously (row-oriented), each field of sequential fields are stored together. Finally, WideTable uses dictionary compression to limit the size of the data in each column.

Since WideTable converts most database operations into simple scans, a fast scan implementation is important. WideTable uses BitWeaving as its scan implementation [21]. BitWeaving performs the scan on the compressed and compacted data stored in the WideTable columns. Thus, it significantly reduces the bandwidth required for scan operations and increases performance compared to previous scan algorithms. BitWeaving has two different algorithms for scanning columnar databases: horizontal and vertical. In horizontal BitWeaving, each coded record is stored sequentially with a single bit padding between each code. In vertical BitWeaving, the bits of each code are stored across separate words such that each bit of sequential records are stored together. The specific BitWeaving algorithms are described in Li and Patel's paper [21].

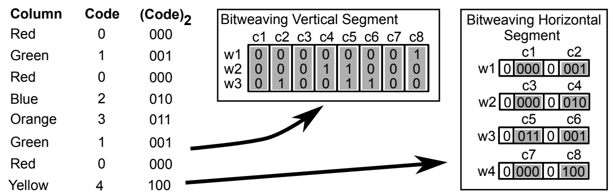

**Figure 2:** Overview of BitWeaving vertical and horizontal. Left shows a single column of a database and right shows the physical memory layout. w1–3 are three consecutive words in memory and c1–8 are eight consecutive records in a column.

Figure 2 shows an overview of BitWeaving. This figures shows three consecutive words in memory (w1, w2, and w3) of the two forms of BitWeaving. The codes are stored physically in words. Each word may hold multiple codes, and a single code may be split across multiple words. In BitWeaving vertical, each column (c1, c2, etc.) is *vertical*, and in BitWeaving horizontal each column is *horizontal*.

## 3 Unified Address Translation – largely completed work

CPUs and GPUs are currently tightly integrated physically with the CPU and GPU on the same silicon die (e.g., AMD Fusion, Intel desktop chips, most mobile chips). However, they are still logically separate. Most current GPGPU programming assumes separate physical memory address spaces and requires the programmer to explicitly move data from the CPU memory to the GPU memory.

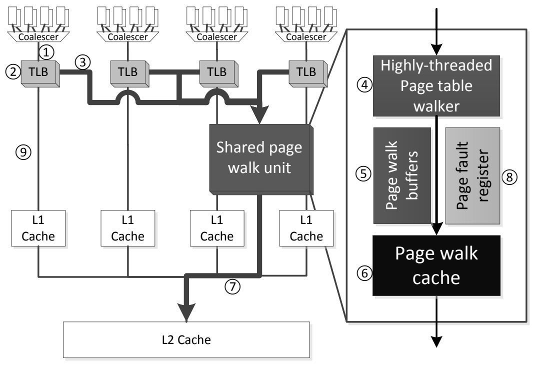

**Figure 3:** Proposed GPU MMU design.

Industry recognizes that a more logically integrated system can increase programmability and performance and is proposing solutions such as CUDA 6 from NVIDIA [17] and heterogeneous system architecture (HSA) from AMD and its partners [37]. These new paradigms allow GPGPU programmers to logically consider the CPU and GPU in the *same virtual address space*. This logical coupling of the CPU and GPU can be implemented multiple ways. In CUDA 6, the runtime transparently manages moving the data between the CPU and GPU. A potentially lower overhead solution targeted at tightly physically integrated systems is to use hardware to manage data movement. HSA uses this solution.

To implement a shared virtual address space between the CPU and GPU in hardware there are two key parts. First, the caches private to each device must be kept coherent (e.g., through judicious cache flushing or hardware mechanisms [32]). Second, the CPU and GPU must use the same address translation mechanism. This latter requirement is the focus of this work which is published in HPCA 2014 [33] and attached in Appendix A.

To unify the virtual address spaces of the CPU and GPU, we develop a GPU memory management unit (MMU) which implements the x86-64 address translation mechanism. Through a data-driven approach, we develop a proof-of-concept GPU MMU design (Figure 3) that negligibly impacts GPU performance compared to an ideal MMU (an average of 2% and maximum of 10%) and is fully compatible with x86-64 page table structure. We present three key findings and a design motivated by each finding. Our final proof-of-concept GPU MMU design, shown in Figure 3, puts L1 TLBs after the coalescing unit and scratchpad cache, contains a highly-threaded pagetable walker, and contains a page walk cache.

### 3.1 Key findings that drive our GPU MMU design

**Motivating Design 1: Post-coalescer MMU** — We find that accessing the GPU MMU after the scratchpad cache and the coalescer reduces total translation traffic by 85%. The reason for this reduction is that for a GPU application to perform well it must have regular memory access patterns that exhibit good coalescing behavior. Irregular access patterns that exhibit poor coalescing leverage the scratchpad cache to achieve high performance. Therefore, by putting the MMU after these hardware structures we can leverage the same optimizations programmers already make to reduce the impact of address translation.

**Motivating Design 2: Highly-threaded page table walker** — After investigating the performance bottlenecks in our first design, we find that bursts of TLB misses cause low performance. In fact, we find that there is an average of 60 concurrent page table walks at each CU and that over 90% of page table walk requests are issued within 500 cycles of the previous request. Therefore, we advocate for a non-blocking page table walker implemented with a highly-threaded design.

**Motivating Design 3: Page walk cache** — After alleviating the concurrent page walk bottleneck, we find that the L1 TLB miss rate is quite high. The poor temporal locality of GPU workloads cause this high miss rate. To reduce the impact of the high miss rate on performance, we add a page walk cache to our design that is accessed by the highly-threaded page table walker before the L2 data cache. This page walk cache design is similar in both implementation and in spirit to page walk cache structures used to accelerate CPU page table walks.

**Performance Results** — Figure 4 shows the performance of the three GPU MMU designs relative to an ideal, impossible to implement, MMU with minimal latency to the data cache and infinite L1 TLBs. Detailed methodology is in Appendix A. This figure shows that with Design 3, the GPU MMU is within 2% of the ideal MMU, on average for our workloads. Moving from Design 1 to Design 2 (adding the highly-threaded page table walker) increases performance for most workloads. nw and bfs are the exception to this and bring down the overall average for Design 2. These two workloads suffer from conflicts at the highly-threaded page table walker, which could be solved with a better queue priority algorithm, but we find that Design 3 resolves this issue. Design 3 increases performance over Design 2 for all of the workloads. The lowest performing workload is bfs because it has the worst coalescing behavior. We show in the full paper the performance of bfs can be closer to the ideal MMU with an optimized page walk cache design.

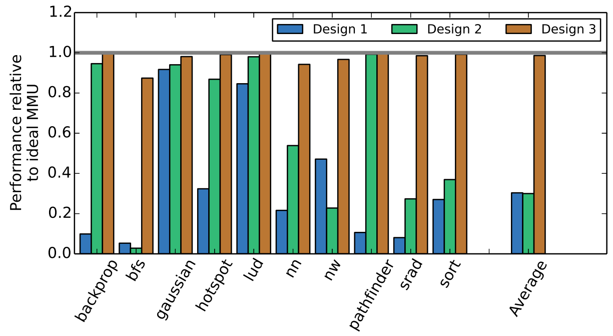

**Figure 4:** Results for proposed GPU MMU design.

The full paper in Appendix A also presents a discussion of correctness issues—page faults and TLB coherence; a sensitivity analysis comparing our proof-of-concept design to a design with a L2 TLB, different page walk cache designs, and TLB prefetching; a discussion on the impact of large pages; and an analysis of the energy and area overheads of our MMU design.

## 4 BitWarp: Using GPUs for Analytic Query Processing – previously submitted work-in-progress

Analytic query workload's performance and energy efficiency are increasingly important. We propose to leverage the efficiency gains of the GPU to increase the energy efficiency of analytic query workloads. Previous work, WideTable, showed that analytic queries (specifically evaluated with TPC-H) can be transformed such that most of the query can be executed using only a scan operation [21, 22]. Because of the transformations used in WideTable, executing the *scan* kernel is where most of the time is spent. Therefore, we focus on accelerating this kernel with GPGPUs. The scan kernel takes a column of data and performs a predicate computation for every data element and emits a vector of bits such that there is a one in every place where the data matches the predicate. Because of the regular nature of the memory access patterns in scan, the GPU is able to accelerate it significantly. A draft of this work is attached as Appendix B.

In BitWarp, we evaluate the scan kernel and full TPC-H queries on three different GPU generations: discrete GPUs, integrated GPUs, and a future tightly-coupled GPU (described in more detail in Section 4.3). Discrete GPUs have a high compute capability (many processing elements) and very high memory bandwidth but have a reduced capacity main memory and high CPU-GPU communication overheads. Integrated GPUs reduce the communication overhead and access expandable main memory (shared with the CPU). However, compared to discrete GPUs, integrated GPUs do not have as much compute capability and have a much smaller bandwidth to main memory. Future GPUs will combine the best aspects of discrete GPUs and integrated GPUs by leveraging 3D die-stacking. Future GPUs have the potential for high compute capability by stacking a GPU die on top of a CPU die, high memory bandwidth due to through-silicon vias, and low CPU-GPU communication overheads from new programming models and tight physical integration.

We show that the integrated GPU uses less than half the energy of a single core CPU, the discrete GPU uses half the energy of the integrated GPU and the future GPU could use 5× less energy than the discrete GPU. In fact, we show that a future GPU could reduce the scan energy by 60× compared to CPU platforms.

## 4.1 BitWarp Algorithm Overview

The BitWarp scan algorithm is a port of the BitWeaving algorithm discussed in Section 2.2 to run on GPUs built within the WideTable framework. There are three key differences between BitWeaving and BitWarp. First, BitWarp must manage data movement between the CPU and

GPU, which is handled transparently through modifications to the WideTable framework. Second, BitWarp uses a much larger underlying word size such that all PEs access consecutive memory words to leverage memory coalescing. Third, BitWarp executes aggregates on the GPU as well as the scan part of each query, which required updates to the WideTable framework which also significantly improved CPU aggregate performance.

## 4.2 TPC-H Performance

We evaluated BitWarp on a subset of the TPC-H queries. We were unable to include all of the TPC-H queries due to time constraints, but we plan to port all of the queries used in WideTable. Figure 5 shows both the energy and response time for each hardware configuration on each query relative to the single core CPU platform. Points to the lower-left imply lower energy and lower response time. Each color corresponds to the hardware used, and the shapes correspond to the TPC-H query.

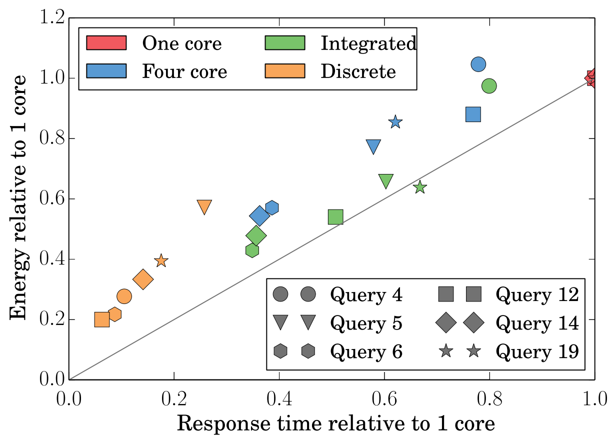

**Figure 5:** Energy and response time for each hardware configuration on each query relative to the single core CPU platform.

Figure 5 shows that for each TPC-H query the discrete GPU dominates all other hardware configurations in both energy consumption and response time. The integrated GPU dominates the one and four-core CPU in terms of energy for all queries, but for query 4 and query 19 the integrated GPU has a higher response time than the four-core CPU. Thus, in these cases there is a trade-off between energy efficiency and response time. Detailed performance and energy graphs for the TPC-H queries and an analysis of BitWarp's sensitivity to code size and column size is in Appendix B.

## 4.3 Future GPU platforms

Finally, in this work, we predict the effect of current technology innovations on this analytic database workload. We evaluate two technology innovations: tight CPU-GPU coupling and 3D die-stacking. Tight CPU-GPU coupling is in its infancy today with the first hardware support for heterogeneous system architecture (HSA) being released earlier this year. This tight coupling will decrease the overheads of using the GPU for general purpose computing by eliminating the driver bottleneck from the common compute path and allowing the CPU and GPU to share a virtual address space.

3D die-stacking is now a reality [5] due to advances in fabrication, cooling, and other technologies. Die-stacking will allow future GPUs to be more efficient in two different ways. First, die-stacking allows the manufacturing process to be optimized for CPUs and GPUs separately, increasing performance, increasing energy efficiency, and allowing more die area devoted to both. Second, die-stacking allows very high bandwidth to DRAM, up to 1 TB/s in some projections [1, 11, 30].

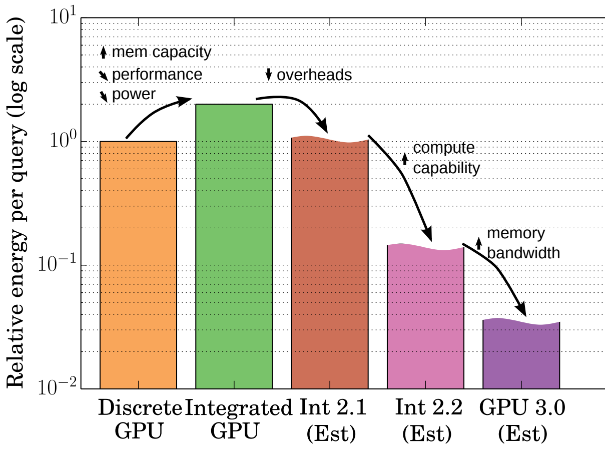

**Figure 6:** Relative energy per scan for predicted future GPU systems.

Figure 6 shows the predicted relative energy of these future architectures with the impact of each optimization. We used a three step process to estimate the performance of the future system. First, to gauge the impact of software overheads (e.g., memory copies and kernel launch overheads), we ran a number of batch scans on the integrated GPU without CPU involvement and found there was an approximate 2× performance improvement. Next, we assumed that future 3D-integrated GPUs will have a compute capability on par with today's discrete GPU and scaled the performance accordingly. Finally, to simulate higher memory bandwidth, we decreased the GPU clock and kept the memory clock constant. We found the performance to be unchanged, which means that in the converse case—increased memory bandwidth—the performance would increase proportionally. Therefore, we hypothesize that the scan workload will increase performance relative to the increase in bandwidth (up to 1 TB/s).

In our BitWarp work, we leave the details of this future architecture for future work. Developing this novel system to perform analytic query workloads as energy efficiently as possible is the focus of my proposal in the next section.

## 5 Analytic Database Machine (ADBM) – proposed work

This section describes a new system architecture specifically designed for analytic database queries that can provide significant efficiency gains compared to current platforms. This work is in its early stages, so we especially value your feedback.

### 5.1 Introduction

Running analytic queries on databases with terabytes of data or more is becoming important. In fact, the big data market is expected to grow to $16.9 billion in 2015 [40]. Analytic databases are an important part of many businesses and services. For instance, with the growth of Internet-connected devices, there is an explosion of data from sensors on these devices. The ability to analyze this data is important to many different industries from automotive to health care [29].

Not only does the latency of these queries matter, as the queries often have real-time constraints, but the total energy to execute the query is now a first-class design consideration. Today's solutions to high-performance analytic queries over a growing dataset is to scale-out the workload by adding more compute nodes. However, these compute nodes are not optimized for energy efficient analytic queries.

To execute queries with low latency, many database systems use an in-memory data store. With an in-memory database, queries complete faster since they do not need to access the disk, which is very high latency and low bandwidth. In addition to this trend towards low latency queries, memory capacity has increased to the point that the entire database can fit into DRAM or future byte-addressable memory technologies, such as phase change memory [44]. Today, you can buy a single system—a set of processors which all share the a physical memory space—with terabytes of main memory.

To demonstrate the inefficiencies of current systems, Figure 7 shows the percent of total memory capacity that can be accessed within a given amount of time (e.g., service level agreement, SLA) on a log-log scale for three different big memory systems: an AMD platform from Dell (PowerEdge M915), an Intel platform from Dell (PowerEdge M820), and a SPARC platform from Oracle (M6-32). The solid lines show the metric for systems configured for maximum memory capacity (i.e., 1 TB for an AMD blade, 1.5 TB for an Intel blade, and 32 TB for the SPARC server rack with 16 compute blades). This figure was generated by taking the peak theoretical bandwidth and multiplying by the SLA time to find the maximum amount of memory that can be accessed within the SLA time. This methodology will overestimate the amount of memory that can be referenced as it is not possible to achieve the theoretical memory bandwidth.

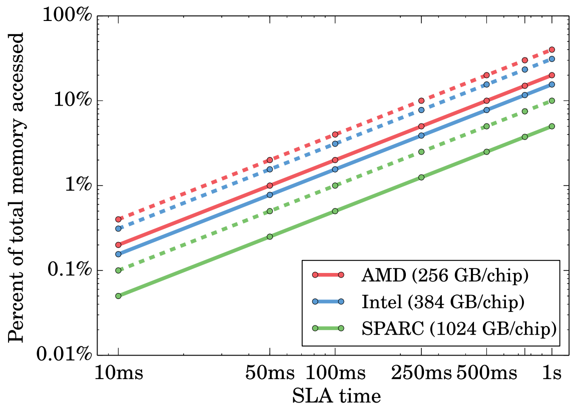

**Figure 7:** Percent of memory that can be accessed given an SLA requirement. Solid lines show today's systems configured for the highest memory capacity.

Figure 7 shows that for high capacity memory systems only a small percent of this capacity can be accessed quickly. For instance, less than 1% of the total memory capacity can be accessed in 50 ms for the three systems shown. The dashed lines in the figure show the percent memory accessed if the bandwidth per capacity is doubled. Bandwidth per capacity can be increased in two ways. Either the capacity per processor is halved, or the bandwidth per processor is doubled though new technology, like the DDR4 standard. Decreasing the capacity per processor implies that the total energy will increase for the same memory capacity. However, even with double the bandwidth per capacity, current systems can access only a small fraction of the total capacity in a reasonable service time.

To solve this memory-wall problem, I propose to take the specialization one step further from my previous work on BitWarp and customize the entire machine for analytic databases. I will leverage new technology trends in tightly-integrated CPU-GPU architectures and 3D die-stacking to significantly reduce the energy and increase the performance of this workload. Additionally, I believe this new architecture will scale better as data sizes increase than the naïve scale-out architecture using commodity components.

There are three main factors that are coming together to make this work possible. First, analytic workloads are becoming increasingly important and they have hit an energy efficiency wall. Second, due to 3D die-stacking, customizing a processor is more cost-effective compared to designing a system-on-a-chip from scratch for database systems. Third, 3D die-stacking is changing the memory bandwidth–compute capability trade-off. Now, DRAM or non-volatile memory can be accessed with latency and power more similar to on-chip caches than traditional off-chip memories.

## 5.2 Related Work

The idea of customizing the underlying architecture to increase performance for database systems is not new [13]. Many of these machines leveraged multiprocessing to increase performance over large mainframes. Boral and Dewitt discuss three different multiprocessor database machine designs: processor-per-track, processor-per-head, and off-the-disk machines [8]. However, Boral and Dewitt argue that the mid-1980's was not the right time for database machines and that the key bottleneck was disk bandwidth, not processing power. Today's trends are similar; compute performance is not the bottleneck. The main bottleneck is memory bandwidth. However, because of the increased importance of energy efficiency and the breakdown in Dennard scaling, it is time to rethink database machines.

There are many systems on the market today that are geared towards the workload we are targeting. For instance, Oracle exadata database machine [27] with hybrid columnar compression (HCC) [28] uses a compression scheme similar to WideTable, and targets big memory workloads. Oracle's exadata machine can be configured with 4 TB of memory and Oracle provides other solutions with up to 32 TB of memory. The Oracle M7 chip can be configured with 2 TB and 160 GB/s of memory bandwidth per chip, and it is expandable to up to 64 coherent chips [31]. Other manufacturers also produce similar machines, such as SAP HANA [41], IBM BLU [34] and

Vectorwise [45].

Other work has found that main-memory bandwidth is a performance bottleneck (e.g., [10, 36, 43]), including specifically for analytic database workloads [7]. Burger at al. show that adding memory latency tolerating hardware to a bandwidth constrained system can hurt performance [10]. This work discusses some of the reasons why current high-performance CPUs are inherently inefficient at high-traffic workloads. Rogers et al. show that using 3D die-stacking can mitigate some of the bandwidth bottleneck [36].

To combat some of these bandwidth and system-size inefficiencies, there has been many proposals for scale-out architectures. Workloads amenable to scale-out architectures were shown to perform inefficiently on high-performance CPUs [14, 15]. Thus, there are proposals to fundamentally change the CPU architecture to match these workloads [24]. This proposal is similar in spirit to this work, but instead of changing the CPU architecture I advocate for using GPGPUs, which is less disruptive to hardware manufacturers.

Wu et al. proposed the Q100 database accelerator which is a separate specialized processor that targets database kernels [42]. The Q100 is made of multiple tiles that each implement a different database function (e.g., select, filter, aggregate). The Q100 has a unique ISA which allows the programmer to string together database operations that are mapped directly to the hardware tiles. This system increases efficiency in two ways. First, it reduces communication cost by streaming data between tiles instead of to and from memory as a conventional CPU. Second, each tile is very specialized leading to increased efficiency. Due to its specialization, the Q100 will likely be more efficient than ADBM. However, the Q100 is less general and more disruptive to the design process for both hardware and software than ADBM.

The Oracle M7 chip contains a small accelerator for some database applications [31]. This accelerator can operate on compressed data or decompress and re-compress the data on-the-fly. The accelerator targets in-memory format conversions, value and range comparisons, and set membership [31]. This accelerator would likely be applicable to the WideTable database system we are using. However, it is designed for the low memory bandwidth to memory capacity ratio of the Oracle database system and will not provide the same performance as ADBM.

Kgil et al. proposed PicoSevers, a 3D-stacked architecture targeting tier 1 servers [18]. PicoServers stack many low-power cores on top of DRAM. The authors show that with this architecture, large L2 caches are no longer necessary due to the low-latency DRAM access, and they show that with this architecture the bottleneck is now the network interface receiving and sending requests and results. Similar to PicoServer, Ranganathan discussed Nanostores which stack non-volatile memory with the processor [35].

Recent implementations of similar 3D-stacked architecture ideas include 3D-stacked servers for key-value stores [16] and thin servers with smart pipes [23]. These two works focus on accelerating memcached which is a simple key-value store often used as a caching layer for web services. These works show great potential for 3D-stacked architectures, but no previous work has focused on using 3D-stacked architectures for analytic database workloads.

## 5.3 A Potential Solution

A key bottleneck in running analytic queries with in-memory databases is the memory bandwidth. As shown in BitWarp (Section 4), the increased bandwidth of the discrete GPU significantly improves performance. Therefore, when designing an architecture specifically for this workload, memory bandwidth is key.

To achieve this high bandwidth, I propose splitting the data into many different 3D-stacks with compute resources *co-located with the data.* This increases the bandwidth of the system in two ways. First, the intra-stack bandwidth is high since the interconnect is through-silicon vias (TSVs) instead of off-chip. Second, by using multiple stacks, the aggregate bandwidth is significantly increases, if most accesses are to local memory.

I leverage the GPGPU architecture instead of a CPU architecture to be able to take advantage of the increased intra-stack memory bandwidth. CPUs are not as efficient as GPUs for analytic database workloads for many reasons. First, CPUs require high-power hardware structures such as re-order buffers and load-store disambiguation units to implicitly expose memory-level parallelism in workloads. GPUs explicitly expose the memory-level parallelism through the SIMT programming model. Second, traditional CPU architecture includes deep cache hierarchies to

|  |  |  |  |
|---|---|---|---|
| DRAM Capacity | 8 Gb × 16 (2 W)[2, 19] | CPU Capability | 1 OOO Core (2–3 W) |
| DRAM Bandwidth | 256 GB/s [19] | Off-stack B/W | 50 GB/s |
| GPU Capability | 15 CUs (2–3 W × 15) | Total Peak Power | 40–60 W |

**Table 1:** Details of stack design.

decrease the effective memory latency. These deep hierarchies are inefficient when there is little temporal locality as seen in analytic workloads. Finally, for the same chip area and power budget, GPUs can have many more computational resources than CPUs.

Below I describe one potential ADBM architecture that leverages these ideas. During the evaluation of this system the final design may change (e.g., what makes up a single system, stack memory capacity and compute capability, etc.).

**Architecture** — Figure 8 shows a possible architecture for the analytic query database machine. In this figure, there is a logic die with a large GPU and smaller CPU with a stack of DRAM as the upper layers. These *stacks* are connected together via a board-level (off-chip) interconnect. Also shown is a side-view of the system. Figure 8 is only a logical representation of the system. An actual physical implementation has many constraints (e.g., cooling) which may require a slightly different physical design (e.g., flipped stacked).

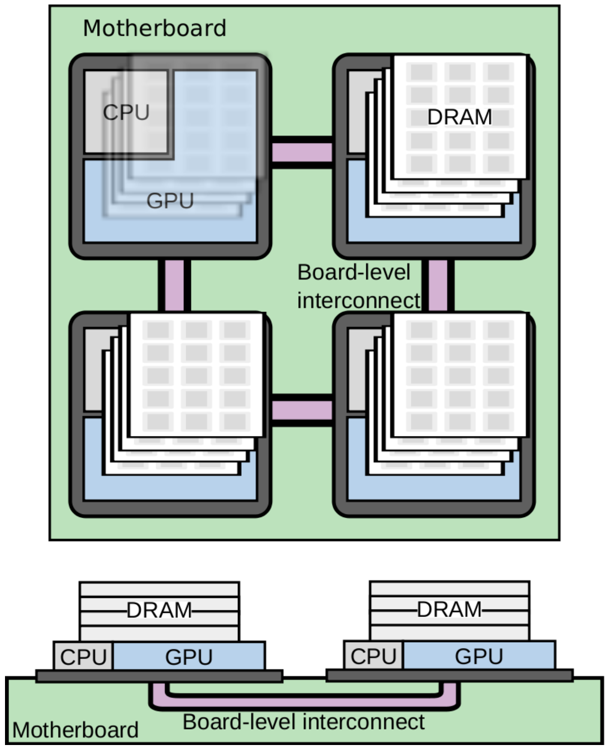

**Figure 8:** One possible architecture for the analytic query database machine.

Different from some of the previous 3D-stacked work, instead of considering each stack as a standalone system, there will only be one operating system running on the board (or blade) and all stacks will share the same physical address space with non-uniform memory access (NUMA). Each board will be made of a number of stacks. The main constraint to the number of stacks on a board will likely be the amount of heat that can be removed from the enclosure, about 800 W in current systems. I predict that each board will be able to have 16–64 stacks depending on the stack specifications.

Table 1 gives details of what is in a single stack assuming a next-generation process. DRAM stack and bandwidth numbers come from high-bandwidth memory [1, 19] assuming a doubling of bandwidth and capacity in the next generation. Power and capability numbers are roughly equivalent to today's systems. With this configuration, 16 of these stacks is within the power budget for a single blade (800 W).

Figure 9 shows the potential benefits this future architecture could provide. Figure 9 shows the energy consumed for a scan operation on today's CPU platforms, today's GPU platforms, and a future platform like the one described above. Methodology and system details are contained in the BitWarp paper in Appendix B. This figure shows that ADBM could provide a 60× decrease in energy compared to a CPU platform. The future GPU's energy is projected by simulating current GPU systems without any overheads as described in Section 4.3.

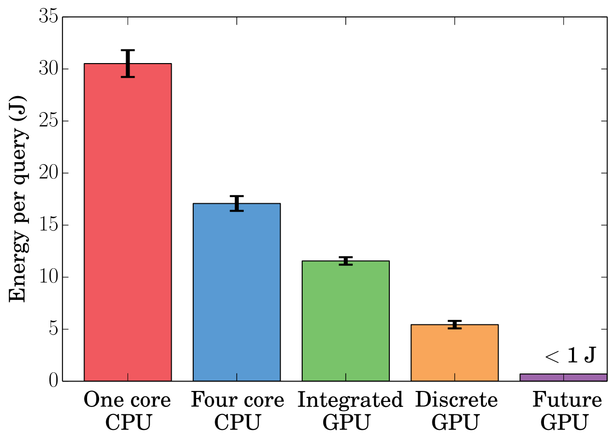

**Figure 9:** Average energy per scan for one billion ($10^9$) 10-bit codes. Future GPU is a predicted value.

**Inter-chip communication** — In ADBM, I propose to use a shared-memory interface between all of the stacks in the system for compatibility and ease of programming. Since ADBM can support terabytes of main memory, most of which is non-local, a shared-memory interface may have a large overhead for cache coherence in terms of performance and on-die storage for the directory. However, the key factor for ADBM to provide high performance is for most memory accesses to be local. By leveraging this fact, the inter-chip communication and coherence overheads can be minimized. For instance, given that most memory requests are to local memory, it may be plausible to implement a coherence protocol which disallows caching of remote data (e.g., the T3E [38]).

ADBM is best suited for a workload that has mostly-predictable memory access patterns, because for this architecture to be efficient, most memory accesses must be to the local memory and the inter-chip communication must be low. Luckily, the working set of the scan portion of the analytic database workload is easy to predict. For the other part of the workload (i.e., the aggregate), the working set cannot be predicted *a priori* as it is only known after the scan portion. However, most queries have a low selection rate in the scan phase; therefore, only a small percent of the data is referenced in the aggregate phase reducing the bandwidth requirement.

To eliminate any inter-chip communication during the scan computation, the data can be split across the chips in a specific way. Since the database is in a columnar format, each stack can have an equal number of rows of every column in the database. Each time a scan is performed, the work is split across many GPU workgroups. These workgroups are assigned a specific CU on some GPU in the system. The runtime system can statically analyze the kernel that the workgroups are going to execute and determine which stack the data resides. Then the runtime can assign that workgroup to a CU on the stack that has the data it is going to reference. This allows each scan to only access local data, and each stack will have an equal amount of work limiting load imbalance.

## 5.4 Evaluation Methodology

**Comparison platforms** — I am planning to compare ADBM to commercial servers which target this kind of big-memory workloads. E.g. Oracle M6-32 which is a cabinet system with up to 32 TB of memory, Intel and AMD multi-socket platforms configured with 512 MB–1 TB of memory per blade. These systems use about 800 W per blade, not counting cooling and power supply losses. Although it is unlikely I can directly compare to these systems due to prohibitive costs, I can possibly compare to a reduced memory capacity, single-blade, version of these systems.

**Evaluating the whole system** — I am going to use current hardware platforms to evaluate the system design as a whole as the system is much too large to evaluate with a simulator. There are two possible, non conflicting, ways to evaluate a system with many stacks on current hardware platform, time-multiplexed and simulated-parallel.

The time-multiplexed approach only uses a single hardware GPU. When a kernel is launched to the platform, the data that is in the simulated stack is loaded onto the GPU and the kernel is run to completion. Then the total time for the parallel phase can be computed with the serial times and the number of stacks. The inter-chip communication can be simulated by capturing the inter-chip memory references and applying a simple network model.

The simulated-parallel method is similar to the time-multiplexed method, except that instead of using a single GPU, I will use one GPU per stack on a cluster of GPUs. The simulated-parallel approach can decrease the time to simulate the system. Also, it may make simulating inter-chip communication easier. I can use similar methodology to simulate the inter-chip communication when using the simulated-parallel approach as in the time-multiplexed method.

Evaluating the effect of inter-chip communication will be challenging for two reasons. First, there is not a platform which has the bandwidth of the inter-chip communication network; GPU DRAM's bandwidth is too high and PCI-e bandwidth is too low. Second, the DRAM capacity required is that of the entire system (possibly terabytes). These problems can be overcome by using the simulated-parallel method discussed above. This method allows me to test these ideas on at least hundreds of gigabytes, if not terabytes of data depending on the number of hardware nodes used.

**Evaluating a single stack** — To evaluate the performance of a single stack, I can use current GPU platforms which approximate the DRAM capacity, DRAM bandwidth, and compute capability of a single stack. However, it may be impossible to fully examine the design space using real hardware. Therefore, to evaluate design trade-offs, I can use simulation (e.g., GPGPU-Sim [3] or gem5-gpu [33]). When using simulation, it may be necessary to reduce the size of the database that is executing. However, I do not believe that there will be a large impact on steady-state behavior with smaller inputs as long as they are significantly larger than the cache size.

**Evaluating energy consumption** — Estimating the energy and power of novel architecture systems is challenging. One solution is to use a first-order model that is guided by current hardware and data sheets for future hardware. I can collect hardware performance counters with utilization, number of local and non-local memory reference, etc. and apply a this first-order model. Additionally, I can construct a system with a predicted maximum power equal to other similar systems (e.g., the ones listed above) and compare performance to those similar-power systems.

## 5.5 Future research directions

We found in BitWarp that after accelerating the scan portion of the analytic workload the aggregate part of the computation became the bottleneck to performance. I believe there is a space to develop new algorithms and new hardware to accelerate this portion of the computation. The aggregate is made up of many random memory accesses and a few arithmetic operations. Therefore, if gather operations can be accelerated, the aggregate computation will benefit.

There are many interesting directions which this research could lead after initially developing the ADBM architecture. For instance, what would the impact of non-volatile memory be on the ADBM system? Could I replace the stacked-DRAM with stacked-FLASH or another non-volatile memory technology? How would the increased latency or write energy effect the overall system performance and energy efficiency? Specifically, racetrack or domain-wall memory may be a good fit. This new non-volatile memory technology can only read and write a single bit at a time similar to how the BitWeaving and BitWarp algorithms break the scan computation down bit-by-bit.

Also, would this ADBM architecture be applicable to other workloads? It is likely that other workloads that are amenable to scale-out style architecture, but need access to global data, will work well.

## 5.6 Research Deliverables

Below is an ordered list of the deliverables I am currently planning to create as I explore the ADBM research space.

1. Re-submit BitWarp work. This also includes getting the rest of TPC-H working correctly and implementing HSA support. These two points are also required for the ADBM work.

2. Develop ADBM simulation infrastructure. This requires two main parts, separating the execution into smaller subsets to run on each stack and creating the network simulation model.

| | |
|---|---|
| Oct 2014 | All TPC-H queries and HSA working write and collect data |
| Nov 2014 | Submit to SIGMOD |
| Dec 2014 | Work on cluster ADBM impl. |
| Jan 2015 | Break & revisions for SIGMOD |
| Feb 2015 | Initial ADBM numbers |
| Mar 2015 | Develop and evaluate energy model |
| Apr 2015 | Finalize ADBM numbers & design |
| May 2015 | Write ADBM paper (For MICRO or ASPLOS) |
| Jun 2015 | Begin exploring other ADBM ideas |
| Jul 2015 | continue |
| Aug 2015 | continue |
| Sep 2015 | HPCA (re)submission? |
| Oct 2015 | Write thesis |
| Nov 2015 | Write thesis |
| Dec 2015 | Defend thesis |

**Table 2:** Schedule to complete my thesis.

3. Write paper which introduces the ADBM architecture to an architecture conference (maybe ASPLOS). Key takeaways from this paper will be the fundamental inefficiencies of both CPUs and current GPU systems for analytic workloads and a new architecture to overcome these inefficiencies (ADBM).

## 6 Schedule and Other Work

### 6.1 Schedule

Table 2 contains my planned schedule to complete my thesis.

### 6.2 Other Work

This section highlights some work I have done as a graduate student that I currently plan to *not* include in my dissertation.

**Heterogeneous System Coherence** — As previously discussed, there are two challenges to design a memory system for tightly-integrated CPU-GPU platforms: unified virtual address space (Section 3) and coherence between the private caches. Heterogeneous system coherence (HSC) targets the latter challenge [32]. In the HSC paper we show that the central directory in modern coherence protocols is a bottleneck in future heterogeneous system. To alleviate this bottleneck, we leverage coarse-grained coherence to move most data requests off of the low-bandwidth coherence network, onto the high-bandwidth incoherent network present in heterogeneous systems. We find that HSC can reduce directory bandwidth by 94% and increase performance by 2×, on average. This work was largely done during an internship at AMD and published in MICRO 2013.

**gem5-gpu** — gem5-gpu [33] is a heterogeneous system simulator that combines the state-of-the-art simulators gem5 [4] and GPGPU-Sim [3] which simulate a CPU system and a GPGPU, respectively. gem5-gpu provides full-system simulation of the heterogeneous system. gem5-gpu also leverages Ruby in gem5 and provides detailed cache models opening the door to research in heterogeneous cache architectures. We released gem5-gpu as an open source project which can be found at `http://gem5-gpu.cs.wisc.edu`, and an overview paper was published in Computer Architecture Letters.

## 7 Conclusions

My thesis aims to increase the efficiency of data-intensive workloads in three ways. First, I provide a hardware mechanism for the CPU and GPU to share the same virtual address space, increasing the efficiency and easing the programming of general-purpose GPU applications. Second, I specifically target analytic database workloads and present a novel software system that targets heterogeneous hardware to significantly increase the efficiency of these workloads. And third, I propose a new system hardware architecture which tightly-couples the CPU and GPU with DRAM to further increase the efficiency of analytic database queries. With these and other architectural and software innovations, hopefully we will continue to see growth in data and application complexity for many more years.

## References

[1] 3D-ICs. http://www.jedec.org/category/technology-focus-area/3d-ics-0.

[2] 4Gb B-die DDR3 SDRAM. Technical report, Samsung Electronics, 2012.

[3] Ali Bakhoda, G.L. Yuan, W.W.L. Fung, Henry Wong, and Tor M. Aamodt. Analyzing CUDA workloads using a detailed GPU simulator. In *IEEE International Symposium on Performance Analysis of Systems and Software*, pages 163–174. IEEE, April 2009.

[4] Nathan Binkert, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, David A. Wood, Bradford M. Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, and Tushar Krishna. The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 39(2):1, August 2011.

[5] Bryan Black. MICRO 46 Keynote: Die Stacking is Happening, 2013.

[6] Mark Bohr. A 30 Year Retrospective on Dennard's MOSFET Scaling Paper. *IEEE Solid-State Circuits Newsletter*, 12(1):11–13, January 2007.

[7] Peter Boncz, Stefan Manegold, and Martin Kersten. Database Architecture Optimized for the new Bottleneck: Memory Access. In *Proceedings of the 25th VLDB Conference*, pages 54–65, 1999.

[8] Haran Boral and David J. Dewitt. Databse Machines: An idea whose time has passed? A critique of the future of database machines. Technical Report #504, UW-Madison Computer Sciences, 1983.

[9] Dan Bouvier and Ben Sander. Applying AMD's " Kaveri " APU for Heterogeneous Computing. In *Hot Chips 26*, number August. AMD, 2014.

[10] Doug Burger, James R. Goodman, and Alain Kägi. Memory bandwidth limitations of future microprocessors. In *Proceedings of the 23rd annual international symposium on Computer architecture - ISCA '96*, pages 78–89, New York, New York, USA, 1996. ACM Press.

[11] HMC Consortium. Hybrid Memory Cube Specification 1.0. http://hybridmemorycube.org/files/SiteDownloads/HMC_Specification%201_0.pdf, 2013.

[12] R.H. Dennard, F.H. Gaensslen, V.L. Rideout, E. Bassous, and A.R. Leblanc. Design Of Ion-implanted MOSFET's with Very Small Physical Dimensions. *IEEE Journal of Solid-State Circuits*, SC-9(5):256—-268, April 1974.

[13] David J. Dewitt, Robert H. Gerber, Goetz Graefe, Michael L. Heyrens, Krishna B. Kumar, and M. Muralikrishna. GAMMA - A High Performance Dataflow Database Machine. In *Proceedings of the 12th International Conference on Very Large Data Bases*, pages 228–237, 1986.

[14] Michael Ferdman, Babak Falsafi, Almutaz Adileh, Onur Kocberber, Stavros Volos, Mohammad Alisafaee, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, and Anastasia Ailamaki. Clearing the clouds. In *Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS '12*, page 37, New York, New York, USA, 2012. ACM Press.

[15] Michael Ferdman, Babak Falsafi, Almutaz Adileh, Onur Kocberber, Stavros Volos, Mohammad Alisafaee, Djordje Jevdjic, Cansu Kaynak, Adrian Daniel Popescu, and Anastasia Ailamaki. Quantifying the Mismatch between Emerging Scale-Out Applications and Modern Processors. *ACM Transactions on Computer Systems*, 30(4):1–24, November 2012.

[16] Anthony Gutierrez, Michael Cieslak, Bharan Giridhar, Ronald G. Dreslinski, Luis Ceze, and Trevor Mudge. Integrated 3D-stacked server designs for increasing physical density of key-value stores. *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems - ASPLOS '14*, pages 485–498, 2014.

[17] Mark Harris. Unified Memory in CUDA 6. http://devblogs.nvidia.com/parallelforall/unified-memory-in-cuda-6/, 2013.

[18] Taeho Kgil, Shaun D'Souza, Ali Saidi, Nathan Binkert, Ronald Dreslinski, Trevor Mudge, Steven Reinhardt, and Krisztian Flautner. PicoServer: Using 3D Stacking Technology To Enable A Compact Energy Efcient Chip Multiprocessor. In *Proceedings of the 12th international conference on Architectural support for programming languages and operating systems - ASPLOS-XII*, page 117, New York, New York, USA, 2006. ACM Press.

[19] Joonyoung Kim and Younsu Kim. HBM: Memory Solution for Bandwidth-Hungry Processors. In *Hot Chips 26*, number August. SK hynix Inc., 2014.

[20] Jonathan Koomey. Growth in Data center electricity use 2005 to 2010. Technical report, Analytics Press, 2011.

[21] Yinan Li and Jignesh M Patel. BitWeaving : Fast Scans for Main Memory Data Processing. *SIGMOD*, 2013.

[22] Yinan Li and Jignesh M Patel. WideTable: An Accelerator for Analytical Data Processing. *PVLDB*, 7(10):907—-918, 2014.

[23] Kevin Lim, David Meisner, Ali G Saidi, Parthasarathy Ranganathan, and Thomas F Wenisch. Thin servers with smart pipes. *ACM SIGARCH Computer Architecture News*, 41(3):36, July 2013.

[24] Pejman Lotfi-Kamran, Emre Ozer, Babak Falsafi, Boris Grot, Michael Ferdman, Stavros Volos, Onur Kocberber, Javier Picorel, Almutaz Adileh, Djordje Jevdjic, and Sachin Idgunji. Scale-out processors. In *Proceedings of the 39th Annual Internation Symposiom on Computer Architecture*, number 3, pages 500–511, September 2012.

[25] Andrew McAfee and Erik Brynjolfsson. Big Data: The Management Revolution. *Harvard Business Review*, October 2012.

[26] Gordon E. Moore. Cramming More Components Onto Integrated Circuits. *Electronics*, pages 114–117, January 1965.

[27] Oracle. A Technical Overview of the Oracle Exadata Database Machine and Exadata Storage Server. Technical Report An Oracle White Paper, Oracle, 2012.

[28] Oracle. Hybrid Columnar Compression (HCC) on Exadata. Technical Report An Oracle White Paper, Oracle, 2012.

[29] Oracle. Big Data Analytics. Technical Report March, Oracle, 2013.

[30] J Thomas Pawlowski. Hybrid Memory Cube (HMC). In *Hot Chips 23*, 2011.

[31] Stephen Phillips. M7: Next Generation SPARC. In *Hot Chips 26*, 2014.

[32] Jason Power, Arkaprava Basu, Junli Gu, Sooraj Puthoor, Bradford M. Beckmann, Mark D. Hill, Steven K. Reinhardt, and David A. Wood. Heterogeneous system coherence for integrated CPU-GPU systems. In *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture - MICRO-46*, pages 457–467, New York, New York, USA, 2013. ACM Press.

[33] Jason Power, Joel Hestness, Marc S. Orr, Mark D. Hill, and David A. Wood. gem5-gpu: A Heterogeneous CPU-GPU Simulator. *Computer Architecture Letters*, 13(1):1–4, 2014.

[34] Vijayshankar Raman, Guy M. Lohman, Tim Malkemus, Rene Mueller, Ippokratis Pandis, Berni Schiefer, David Sharpe, Richard Sidle, Adam Storm, Liping Zhang, Gopi Attaluri, Ronald Barber, Naresh Chainani, David Kalmuk, Vincent KulandaiSamy, Jens Leenstra, Sam Lightstone, and Shaorong Liu. DB2 with BLU acceleration: so much more than just a column store. *Proceedings of the VLDB Endowment*, 6(11):1080–1091, August 2013.

[35] Parthasarathy Ranganathan. From Microprocessors to Nanostores: Rethinking Data-Centric Systems. *Computer*, 44(1):39–48, January 2011.

[36] Brian M. Rogers, Anil Krishna, Gordon B. Bell, Ken Vu, Xiaowei Jiang, and Yan Solihin. Scaling the bandwidth wall: Challenges in and Avenues for CMP Scaling. In *Proceedings of the 36th annual international symposium on Computer architecture - ISCA '09*, number 1, page 371, New York, New York, USA, 2009. ACM Press.

[37] Phil Rogers. Heterogeneous System Architecture Overview. In *Hot Chips 25*, 2013.

[38] Steven L. Scott. Synchronization and communication in the T3E multiprocessor. *ACM SIGOPS Operating Systems Review*, 30(5):26–36, December 1996.

[39] Transaction Processing Performance Council. TPC Benchmark H. Revision 2.14.3, November 2011.

[40] Dan Vesset, Henry D Morris, Matthew Eastwood, Benjamin Woo, Richard L Villars, Jean S Bozman, and Carl W Olofson. Worldwide Big Data Technology and Services 2012–2015 Forecast. Technical Report March, International Data Corporation, 2012.

[41] Gereon Vey and Ilya Krutov. *SAP In-Memory Computing on IBM eX5 Systems*. IBM Corp., 2012.

[42] Lisa Wu, Andrea Lottarini, Timothy K Paine, Martha A Kim, and Kenneth A Ross. Q100: The Architecture and Design of a Database Processing Unit. In *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems - ASPLOS '14*, pages 255–268, New York, New York, USA, 2014. ACM Press.

[43] Wm. A. Wulf and Sally A. McKee. Hitting the memory wall. *ACM SIGARCH Computer Architecture News*, 23(1):20–24, March 1995.

[44] Yuan Xie. Modeling, Architecture, and Applications for Emerging Memory Technologies. *IEEE Design & Test of Computers*, 28(1):44–51, January 2011.

[45] Marcin Zukowski and Peter Boncz. Vectorwise: Beyond Column Stores. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, pages 1–6, 2012.