
GORDON: AN IMPROVED ARCHITECTURE FOR DATA-INTENSIVE APPLICATIONS

GORDON IS A SYSTEM ARCHITECTURE FOR DATA-CENTRIC APPLICATIONS COMBINING LOW-POWER PROCESSORS, FLASH MEMORY, AND DATA-CENTRIC PROGRAMMING SYSTEMS TO IMPROVE PERFORMANCE AND EFFICIENCY FOR DATA-CENTRIC APPLICATIONS. THE ARTICLE EXPLORES THE GORDON DESIGN SPACE AND THE DESIGN OF A SPECIALIZED FLASH TRANSLATION LAYER. GORDON SYSTEMS CAN OUTPERFORM DISK-BASED CLUSTERS BY 1.5× AND DELIVER 2.5× MORE PERFORMANCE PER WATT.

..... We live in a world overflowing with data. From handheld devices to data centers, we collect and analyze ever-greater amounts of information. Companies such as Google and Microsoft routinely process many terabytes of data, and users of desktop search engines routinely pose queries across hundreds of gigabytes of data stored on their hard drives. There's no reason to expect that our appetite for collecting and processing data will stop growing at its current breakneck speed.

To satiate our appetite for large-scale data processing, current technology must overcome three challenges. First, the recent slowdown in uniprocessor performance and the difficulties in programming their chip multiprocessor (CMP) replacements make it increasingly difficult to bring large computing power to bear on a single problem. Second, while hard-drive capacity continues to grow, the latency and bandwidth that hard drives can deliver do not. Third, power constraints due to cooling, economic, and ecological

concerns severely limit the range of possible solutions for both of these problems.

Designers have made some progress on each of these fronts: For data-centric applications, programming models such as Google's Map-Reduce¹ and Microsoft's Dryad² largely automate the task of parallelizing data-processing programs. Solid-state storage devices offer increased bandwidth and reduced latency for mass storage. Finally, processor manufacturers have developed capable, yet power-efficient processors.

For data-centric computing, the most fundamental of these three advances is the rise of solid-state storage. Flash memory's performance characteristics enable systems far outside the design space covered by existing technologies, such as conventional servers, processor-in-disk systems, and processor-in-memory systems. The highest-density flash memories available today (or in the near future) offer 16× the density per package of DRAM at 1/16 the power.^{3,4} In the near future, an array of four flash packages will

Adrian M. Caulfield

Laura M. Grupp

Steven Swanson

University of California,

San Diego

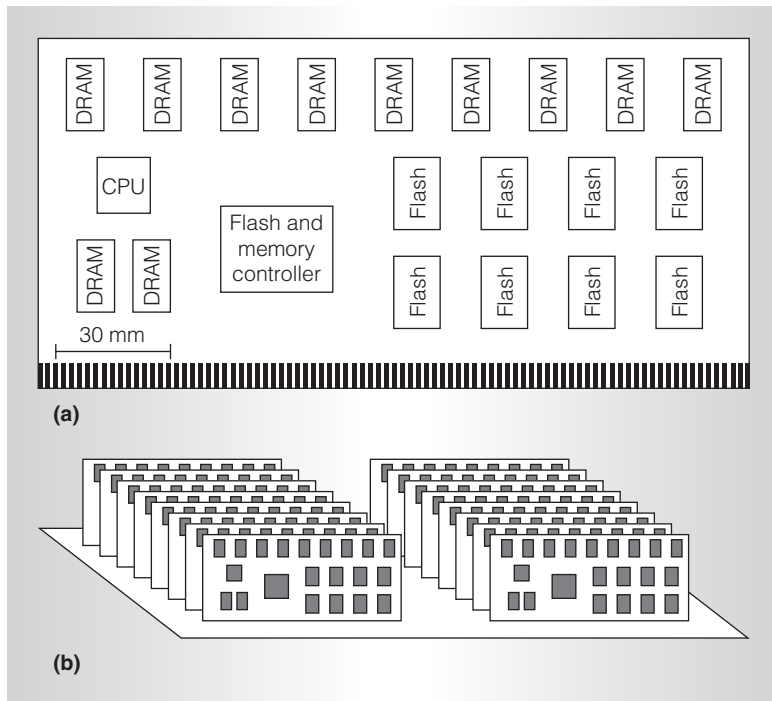


Figure 1. The Gordon system: A scale drawing of a Gordon node process control block (a) and a sketch of 16 nodes in a single enclosure (b). Gordon's nodes can be very compact. This configuration (with flash on both sides) holds 256 Gbytes of flash memory and 2.5 Gbytes of double-data-rate synchronous DRAM (DDR2 SDRAM). The enclosure holds 4 Tbytes of storage and provides 14.4 Gbytes per second of aggregate I/O bandwidth.

deliver $4\times$ the read bandwidth of a high-end disk at $1/30$ the power and a fraction of the latency. These advantages, combined with the fact that solid-state storage arrays comprise many discrete chips instead of a few large drives, provide much more flexibility in the architecture of a combined computing and storage platform.

With this in mind, we developed Gordon (see Figure 1), a flash-based system architecture for massively parallel, data-centric computing. Gordon leverages solid-state disks, low-power processors, and data-centric programming paradigms to deliver enormous gains in performance and power efficiency. (For a brief overview of these technologies, see the “Enabling technologies for efficient, data-centric computing” sidebar.) Gordon also offers a flash management software layer that allows a highly parallel operation of large arrays of flash devices.

As we describe how we designed Gordon, we'll review several considerations, including

- an evaluation of flash management techniques for data-centric applications,
- a thorough analysis of the Gordon system design space and the trade-offs between power, energy, and performance that Gordon must make, and
- a discussion of cost, virtualization, and system-level issues in Gordon machines.

Our results show that Gordon systems can deliver up to $2.5\times$ the computation per energy of a conventional cluster-based system while increasing performance by a factor of up to 1.5. We also demonstrate that our specialized flash management system can deliver up to 900 megabytes per second (Mbps) of read-and-write bandwidth for the applications we used.

Gordon's storage system

Gordon's flash-based storage system is the key to its power efficiency and performance advantage. Although Gordon's storage system targets data-intensive applications, several of the approaches we describe are applicable to more general-purpose storage systems as well. The following sections describe our storage-array hardware and management layer.

Storage hardware

The flash storage system comprises two components—a flash controller and the flash memory itself. The flash devices we model have program and read latencies of 200 microseconds (μs) and $25\ \mu\text{s}$, respectively. The maximum achievable sustained bandwidth on a 133-megahertz (MHz) bus is then 5 Mbps and 126.4 Mbps. Erases take 2 milliseconds (ms).^{3,5-7}

The flash controller implements Gordon's flash translation layer (FTL), which provides the link between the CPU and the flash array. We would like the controller to manage as much storage as possible, but hardware constraints limit its capacity. Flash chips connect to the controller over shared buses. Each bus supports up to four flash packages, each of which contains four dies. The physical organization of the flash memories and the logical organization that the FTL imposes on

Enabling technologies for efficient, data-centric computing

In addition to power-efficient microprocessors, Gordon relies on two technologies: large-scale data parallel-programming systems and solid-state storage. Here we provide a brief overview of these technologies.

Peta-scale data-centric parallel programming

Industrial researchers have recently developed several systems for processing extremely large data sets in parallel. Two of these systems, Microsoft Dryad¹ and Google MapReduce,² offer an alternative to parallel databases³⁻⁵ for managing and computing with massive data sets. Both systems provide simple abstractions for specifying data-parallel computations and then automating the task of making them run in parallel on commodity hardware.

Solid-state storage

Flash memory gained prominence in the last decade because of the rising popularity of mobile devices with large storage requirements (iPods, digital cameras, and so on). In response, flash manufacturers primarily pursued density and cost improvements and paid less attention to performance.

However, recent industrial efforts (see <http://www.onfi.org/specifications>) promise to raise peak bus bandwidth to at least 133 Mbytes per second, and researchers have sought to improve flash performance in general-purpose systems with new chip-level interfaces,⁶ solid-state disk organizations,⁷ flash translation layer designs,⁸ and system-level interfaces (see <http://www.fuionio.com>). These advances signal the beginning of flash memory's coming of age as a high-performance storage technology.

them significantly influence the storage system's performance.

Gordon's flash translation layer

Gordon's FTL is an extension of Birrell et al.'s FTL.⁸ (For an overview of FTLs, see the "Flash translation layer implementations" sidebar.) Birrell et al.'s FTL lets the application write to and read from any logical block address at random, hiding the flash's idiosyncrasies from the rest of the system and spreading erases evenly across the flash devices. It does this by using a write point to track the next location for writing data. As writes occur, the pointer progresses through a block and then is free to move to a new block anywhere in the array.

A key limitation of Birrell et al.'s FTL⁸ is that it only allows for only a single write point. As a result, it will never allow two

operations to proceed in parallel except in the case of cleaning blocks in the background. For small flash storage systems, such as a USB key, this is acceptable, but for Gordon it isn't.

We use two techniques to solve this problem. First, we extend our FTL to support multiple write points and spread accesses between them. Using multiple write points doesn't affect the read bandwidth significantly, but it can improve the write bandwidth dramatically. Our data show that increasing the number of write points per 133-MHz bus from one to four increases the write bandwidth by 2.8 \times .

The second approach is to combine physical pages from several dies into 2D superpages (and, therefore, superblocks for erase operations) and manage the flash array at this larger granularity. The 2D striping

References

1. M. Isard et al., "Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks," *ACM Special Interest Group on Operating Systems' (SIGOPS) Operating Systems Rev.*, vol. 41, no. 3, 2007, pp. 59-72.
2. J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Proc. 6th Conf. Symp. Operating Systems Design and Implementation*, ACM Press, 2004, p. 10.
3. D. DeWitt and J. Gray, "Parallel Database Systems: The Future of High Performance Database Systems," *Comm. ACM*, vol. 35, no. 6, 1992, pp. 85-98.
4. G. Graefe, "Encapsulation of Parallelism in the Volcano Query Processing System," *ACM Special Interest Group on Management of Data (SIGMOD) Record*, vol. 19, no. 2, 1990, pp. 102-111.
5. D.J. Dewitt et al., "The Gamma Database Machine Project," *IEEE Trans. Knowledge and Data Eng.*, vol. 2, no. 1, 1990, pp. 44-62.
6. R. Schuetz et al., "Hyperlink NAND Flash Architecture for Mass Storage Applications," *Proc. IEEE Non-Volatile Semiconductor Memory Workshop*, IEEE CS Press, 2007, pp. 3-4.
7. N. Agrawal et al., "Design Tradeoffs for SSD Performance," *Proc. Usenix 2008 Ann. Tech. Conf.*, Usenix Assoc., 2008; <http://research.microsoft.com/pubs/63596/USENIX-08-SSD.pdf>.
8. A. Birrell et al., *A Design for High-Performance Flash Disks*, tech. report MSR-TR-2005-176, Microsoft Research, 2005.

Flash translation layer implementations

Researchers have worked hard to improve the performance of flash-based solid-state drives by improving their flash translation layers. FTLs present a disk-like interface to flash memory by dynamically mapping logical block addresses (LBAs) to physical flash pages, providing wear leveling, ensuring reliability in the face of power failure, and maximizing performance. The granularity of the LBA map divides FTLs into three categories: page, block, and hybrid mapping.¹

Page mapping (or sector mapping)

These schemes map every LBA to its own physical page. The flexibility of this option allows for fast reading and writing as well as minimal erases, especially for small accesses. However, the memory required for the simplest version of this design balloons with drive size.

Block mapping

The granularity of the map in these schemes is that of flash blocks. This reduces the memory footprint of the FTL, simplifies the erasure algorithms, and is well-suited to large accesses on single chips.

Unfortunately, it does nothing to hide the disparity between erase and program operation granularities.

Hybrid mapping

Hybrid FTLs combine techniques from both approaches to capture benefits from each. In one example, the mapping is block-based but we can store each page anywhere within the block. This allows for a small DRAM footprint and the flexibility to modify individual pages several times before erasing.

With the striping technique, Gordon's FTL increases the bandwidth for large accesses and reduces the memory requirement by creating blocks out of pages that span chips. The large-granularity map, combined with the flexibility of page remapping, places it in the hybrid-mapping category.

Reference

1. T.-S. Chung et al., "A Survey of Flash Translation Layer," *J. Systems Architecture*, vol. 55, nos. 5-6, 2009, pp. 332-343.

combines horizontal (similar to *ganging* discussed in Agrawal et al.⁹) and vertical striping to generate even larger superpages. This approach lets us divide the array into rectangular sets of chips that are part of the same horizontal and vertical stripes. Our FTL provides one write point for each of these sets.

This 2D striping trades parallelism between operations for parallelism within a single operation. It also reduces management overhead by reducing the total number of superpages in the array. This is important, because the logical-to-physical map for a large flash array can be prohibitively large. For instance, for 256 Gbytes of flash and 2-Kbyte pages, the mapping is 512 Mbytes (as opposed to 16 Mbytes with 64-Kbyte superpages).

However, large superpages also increase the latency of subpage accesses, because the FTL will need to read or program more data than requested. Our workloads present a mix of large and small transfers. Whereas 88 to 93 percent of the bytes read and written are part of transfers of 120 Kbytes or larger, roughly half of the individual transfers are only 8 Kbytes. As a result, setting our page size to 128 Kbytes would roughly double the amount of data the FTL has to read and write.

Fortunately, the traces from our workloads show a clear pattern in accesses: An 8-Kbyte access aligned on a 128-Kbyte boundary is followed immediately by a 120-Kbyte access to the rest of that 128-Kbyte region. To exploit the pattern, we added a simple bypassing mechanism to our FTL that merges incoming read requests with pending requests to the same page and also caches the result of the last read. Our FTL also performs limited write combining by merging write requests to the same page when possible. This mechanism nearly eliminates the negative impact of large pages for small sequential reads.

Figure 2 shows how page size and bypassing affect overall storage performance of our data-intensive workloads, normalized to 8-Kbyte pages. In Figure 2a, bypassing is disabled and then large page sizes benefit the Index, Identity, RandomWriter, and Sort applications (we detail these applications' purposes in the following section). Figure 2b shows performance with bypassing. The 64-Kbyte pages provide between 1.1× and 6.3× speedups for all applications. For sequential accesses (and random accesses to superpages of a similar size) our storage array delivers 900 Mbps of bandwidth for reads and writes.

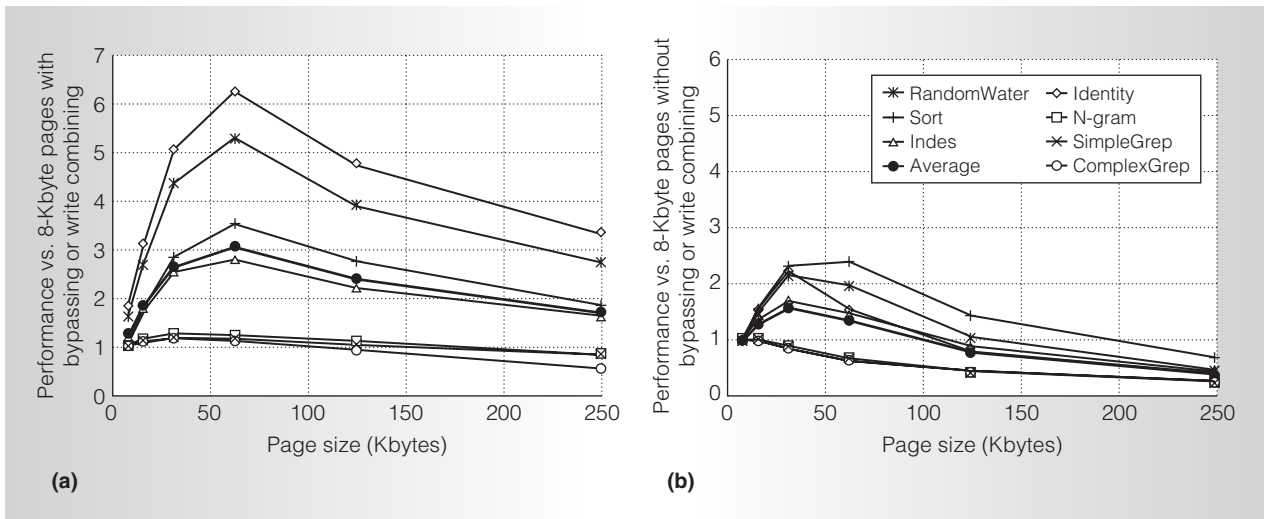


Figure 2. Application-level I/O performance with and without bypassing: Read bypassing and write combining allow all applications to benefit from larger page sizes (a). Without these two optimizations, large pages degrade performance (b).

Configuring Gordon

Having tuned Gordon’s flash storage system for data-intensive applications, we turn our attention to the remainder of the Gordon node design.

Workloads

To evaluate Gordon’s designs, we use a set of highly parallel, data-intensive applications with MapReduce benchmarks, each of which process up to 15 Gbytes of data:

- *RandomWriter* outputs random data,
- *Identity* copies all inputs to the output,
- *Sort* sorts random numbers,
- *SimpleGrep* searches for “the” in multilingual text,
- *ComplexGrep* searches for a complex regular expression in multilingual text,
- *N-Gram* finds frequently occurring N-word phrases in multilingual text, and
- *WebIndex* generates an index for a Web search engine.

Two of the benchmarks (*Identity* and *RandomWriter*) focus specifically on I/O. The other benchmarks represent more realistic applications from a range of domains. *WebIndex* is our most sophisticated application and *ComplexGrep* is the most computationally demanding.

To run the benchmarks, we use Apache Hadoop (see <http://hadoop.apache.org/core>), an industrial-strength open source MapReduce implementation written in Java. It provides many of MapReduce’s features,¹ including a distributed file system (Hadoop DFS), similar to the Google File System (GFS).¹⁰ All our experiments run under Linux 2.6.24 using Sun’s 64-bit Java implementation (JDK 1.6).

To characterize the workloads, we ran each application on a cluster of eight 2.4-GHz Core 2 Quad machines with 8 Gbytes of RAM and a single, large serial advanced-technology attachment (SATA) hard disk. A private 1-Gbit Ethernet network connects the machines.

Power model

To measure power consumption, we developed a power model that describes machine configurations as a set of identical nodes, each of which contains DRAM, a disk (or flash), and one or more processors. Within each node, we model each component’s power as $P = IdlePower \times (1 - ActivityFactor) + ActivePower \times (ActivityFactor)$. The activity factors come from our traces on our flash simulator.

For experiments in which we vary the CPU speed, we scale voltage with frequency across the range supported by each processor

Table 1. The parameters for our design space exploration. For all node configurations, we model a cluster of 32 nodes.

Parameter	Value
Processors	1, 2, 4
Flash dies	0, 64
Hard drives	0, 1, 2, 4
Processor type	Atom, Core 2
Atom frequency (in GHz)	0.8, 1.5, 1.9
Core 2 frequency (in GHz)	0.6, 1.2, 1.8, 2.4

(from 0.75 to 1.2 V for the Atom¹¹ and 0.8 to 1.6 V for the Core 2).¹²

Measuring cluster performance

To evaluate the architecture's performance for a cluster of machines, we use two simulators. The first is a high-level, trace-driven cluster simulator used to measure total system performance. The second simulator provides detailed storage system simulations that let us explore architectural options for flash storage arrays.

We calculate two sets of results from the simulators using two methods that model different amounts of internode synchronization. The first, called Sync, models tight synchronization between nodes during execution. The second method, NoSync, models no synchronization between nodes. These two models provide upper and lower limits on execution time.

High-level simulator. The high-level simulator measures overall performance. We collect traces from running systems that contain second-by-second utilization information for each component. The traces describe the cache, disk, CPU, and network use. They're collected from otherwise idle machines, but include all system activity (such as the operating system, system daemons, and so on).

We model a 32-node cluster by running four VMWare virtual machines (see <http://vmware.com>) on each of our eight servers (giving each virtual machine its own CPU and 2 Gbytes of memory) and gather independent traces for each one. Because

VMWare virtualizes performance counters, we gather instruction and second-level (L2) miss counts for the virtual machine itself.

The simulator processes a set of traces (one per node) in parallel. For each sample of data in the trace, it computes the time needed for instruction execution, disk accesses, and network transfers during that second. It then takes the maximum of these values as the simulated execution time for that sample, effectively modeling perfect parallelism between the disk, the network, and the CPU.

To measure power, the simulator uses the power model we previously described to compute the power consumption for each second of each node in the cluster. The Sync model includes power for the idle periods when one node is waiting for the other nodes to finish. Nodes are never idle in the NoSync model until they finish executing. Once execution on a node is complete, we assume it goes into a deep power-down state.

Storage simulator. We use two different simulators. For disk simulations we use DiskSim,¹³ configured to model a 1-Tbyte, 7,200-revolutions-per-minute (rpm) SATA-II hard drive with a 32-Mbyte disk cache. To model flash behavior we use a custom-built flash storage-array simulator. Both simulators process block-level traces taken from running systems and generate latency and bandwidth measurements for a high-level cluster simulator.

Design space survey

We carried out a systematic survey of the design space for Gordon's nodes, which includes both flash-based and hard-drive-based designs. Table 1 shows the parameters and values we varied in our survey. We restrict ourselves to a single storage type and processor type in each configuration, as well as a power budget of 300 watts (W) per node.

We use our simulators to measure the performance of a cluster of 32 of each node configuration using the Sync model. Figure 3 contains the results for the average across our benchmark suite. Each point represents a single node configuration and measures the energy consumption of the entire

Table 2. Optimal Gordon configurations. For all three design goals (performance, performance per watt, and power consumption) Gordon achieves substantially better results than disk-based designs.

Name	CPU configuration	Average power (in watts)		Power vs. MinP-Disk		Performance per watt vs. MaxE-Disk		Speedup vs. MinT-Disk	
		NoSync	Sync	NoSync	Sync	NoSync	Sync	NoSync	Sync
MinP*	1 Atom, 0.8 GHz	1.43	3.91	0.33	0.32	1.74	1.12	0.07	0.07
MaxE	1 Atom, 1.9 GHz	2.32	4.81	0.54	1.39	2.54	2.15	0.16	0.16
	1 Core 2, 2.4 GHz	9.14	19.89	2.11	1.63	2.31	1.88	0.59	0.56
	2 Core 2, 2.4 GHz	23.82	45.66	5.51	3.74	1.64	1.45	1.09	1.00
MinT	4 Core 2, 1.8 GHz	47.56	92.74	11.00	7.59	1.11	0.92	1.48	1.29
	4 Core 2, 2.4 GHz	58.50	106.18	13.54	8.69	1.08	0.93	1.77	1.49

* MinP isn't Pareto-optimal.

system versus the runtime. All data are normalized to a four-processor Core 2 configuration with one disk.

The circled points are the Pareto-optimal node designs. Note that shorter runtimes are impossible without increasing energy and lowering energy is impossible without increasing the runtime. For all workloads, the same designs are Pareto-optimal and flash-based. Table 2 summarizes the Pareto-optimal designs and the lowest-power design. The designs labeled MinT, MaxE, and MinP are the fastest (minimum time), most efficient (performance per watt), and minimum average power configurations, respectively.

Table 2 also summarizes (highlighted in bold) the improvements in performance, efficiency, and power consumption that flash provides for MaxE, MinP, and MinT relative to otherwise identical disk-based configurations. For instance, MaxE is between 2.2× and 2.5× more efficient than the most efficient disk configuration, while MinP saves more than 68 percent in power. The gains in performance are substantial as well: MinT is between 1.5× and 1.8× faster while expending nearly equal energy.

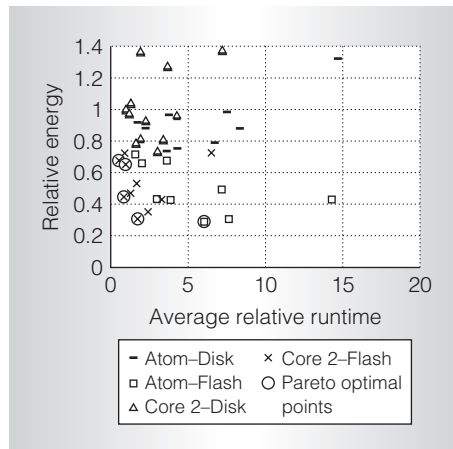


Figure 3. Pareto-optimal Gordon system designs: Results of our design space survey for the average across our benchmark suite. We circled the Pareto-optimal designs.

Gordon power consumption

Figure 4 shows the per-component energy consumption for the MaxE configuration relative to the MaxE-Disk (the most efficient disk-based configuration). Per-component energy consumption is mostly uniform

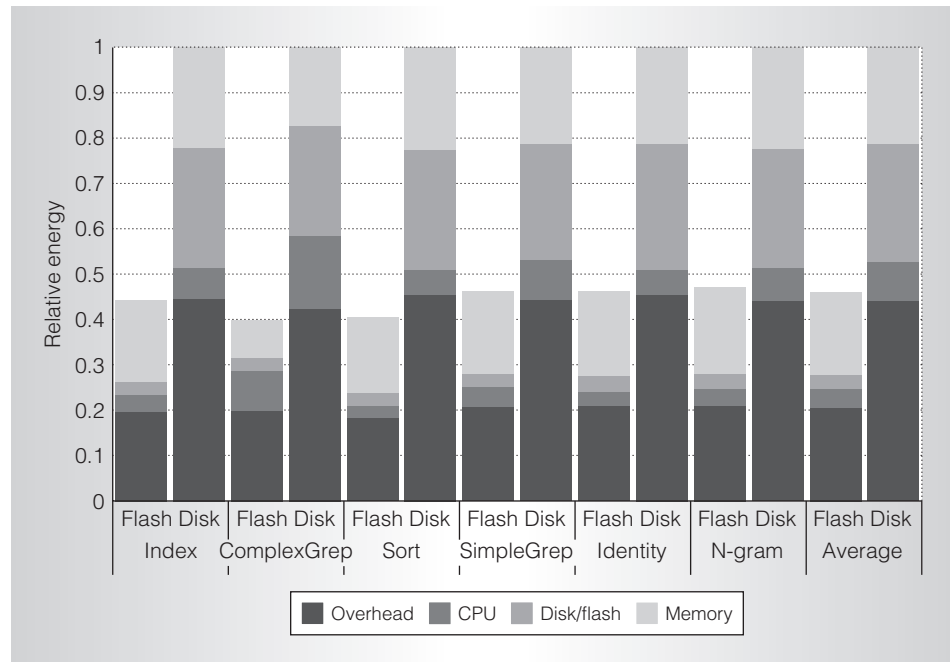


Figure 4. Relative energy consumption: Node energy consumption for MaxE relative to the disk-based configuration with the highest performance per watt.

across the applications. On average, the flash-based MaxE consumes 40 percent of the energy of the disk-based configuration, but execution times are longer, leading to a factor of two increases in performance per watt. The data show that the increased idle power of a disk has a twofold impact on efficiency: It causes the disk to burn excess power, but also encourages the design to use higher-performing, less-efficient processors (because these processors reduce the amount of time in which the disk sits idle). Flash eliminates much of the storage system's idle power, allowing the design to exploit more efficient processors.

Further reducing Gordon's power consumption is challenging. The DRAM and overhead power account for most of the remaining power. Reducing the amount of DRAM might be possible, especially if the in-memory working sets of our applications are small. Overhead power is a widely reported problem at the system level.^{14,15} Reducing this source of waste would benefit a wide range of systems, and make the power savings that flash can offer even more significant.

Utilizing Gordon

Incorporating flash memory into Gordon's system architecture required us to re-evaluate the trade-offs between power, performance, efficiency, and cost in the design of a Gordon node. At a larger scale, fully exploiting the efficiency gains that Gordon offers requires careful consideration of larger-scale trade-offs. We explore several of the trade-offs here and examine usage models for large Gordon systems and their potential roles in a large data center.

Exploit disks for cheap redundancy

Using a distributed, replicated file system increases the cost of storage for both disk- and flash-based systems. We can mediate this problem for some applications by combining Gordon nodes with other servers that have conventional hard drives. Gordon's file system keeps one replica in flash and redundant copies on disk. If a failure occurs, recovery will be automatic. When we don't need the disk-based replicas (the vast majority of the time), we can put the conventional servers into deep sleep with their hard drives spun down.

This system works well for reads, because the replicas don't need updating. For write-intensive workloads, we must store the replica in flash at least temporarily; otherwise the disk bandwidth for updating the hard-drive-based replicas would limit performance and significantly increase power consumption. One approach to mitigating this effect is to treat the flash storage array as a replica cache. We can keep all replicas of frequently updated data in flash, but keep replicas of less-frequently updated data on disk. If writes become less frequent, we can migrate the replicas to disk. Indeed, if a piece of data hasn't been accessed at all in a long time, we can move all replicas to disk.

Virtualizing Gordon

Gordon's strength is in providing high-bandwidth, highly parallel access to large volumes of data, not in storing that data. For storage, disks are more cost effective. This means that we're effectively using a Gordon system if we're frequently accessing the data Gordon stores. Consequently, it makes sense to manage Gordon's systems to maximize bandwidth use.

For some latency-critical workloads and workloads that need to process all the data stored by a group or organization, it makes sense to store data in a Gordon system and process it in place. In other scenarios, we imagine that the total data stored will be much larger than Gordon's capacity. In these cases, we can virtualize the Gordon system so that we can use its data-processing abilities on much larger data sets than it can store.

This usage model treats all or part of a Gordon system as a specialized coprocessor for performing computations on a small part of a much larger quantity of data stored in a disk-based data warehouse. Before a job runs, Gordon will load the necessary data into flash storage and, once the job completes, data for another job will take its place.

For instance, we could partition a 1,024-node system into 16 64-node slices, each with nonvolatile memory (NVM) storage for 16 Tbytes of data. Each slice provides a total of 112 Gbytes per second of I/O bandwidth. Assuming dual 1-Gbit

network connections between the Gordon slice and the data warehouse, loading 10 Tbytes of data (leaving room for scratch data in the flash storage array) would take about 11 hours. Gordon would perform the computations on the data, transfer the results to the warehouse, and load the data for the next job. Network limitations mean that transferring data between the Gordon array and the warehouse uses only 0.4 percent of Gordon's bandwidth resources. We can improve the situation by overlapping data transfer with execution and reducing the storage space allocated to each job by 50 percent (to accommodate storing both data sets simultaneously). The impact on execution time would be minimal and, if jobs spent at least four seconds processing each gigabyte of data, the time cost of loading the next job would be fully hidden by useful processing.

Gordon demonstrates that flash affords the opportunity to re-engineer many aspects of system design, and therefore enables a new class of computing systems. We believe that as flash performance improves, Gordon's performance gains will only increase. MICRO


Acknowledgments

We would like to thank the Architectural Support for Programming Languages and Operating Systems (ASPLOS) program committee reviewers as well Nathan Goulding and Joel Coburn for their helpful comments. This work is supported in part by US National Science Foundation awards NSF0811794 and NSF0643880.

References

1. J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Proc. 6th Conf. Symp. Operating Systems Design and Implementation*, ACM Press, 2004, p. 10.
2. M. Isard et al., "Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks," *ACM Special Interest Group on Operating Systems' (SIGOPS) Operating Systems Rev.*, vol. 41, no. 3, 2007, pp. 59-72.

3. Samsung, *Samsung K9f8g08uxm Flash Memory Datasheet*, 11 June 2007; http://www.samsung.com/global/system/business/semiconductor/product/2007/6/11/NANDFlash/SLC_LargeBlock/8Gbit/K9F8G08U0M/ds_k9f8g08x0m_rev10.pdf.
 4. Micron, *Micron DDR3 SDRAM Mt41j256m8 Datasheet*, 2008; <http://download.micron.com/pdf/datasheets/dram/ddr3/2GbDDR3SDRAM.pdf>.
 5. D. Kwak et al., "Integration Technology of 30nm Generation Multi-Level NAND Flash for 64GB NAND Flash Memory," *Proc. IEEE Symp. Very Large-Scale Integration (VLSI) Technology*, IEEE Press, 2007, pp. 12-13.
 6. Y. Park et al., "Highly Manufacturable 32GB Multi-Level NAND Flash Memory with 0.0098 μm^2 Cell Size Using Tanos (Si-oxide-al₂o₃-tan) Cell Technology," *Proc. IEEE Int'l Electron Devices Meeting*, IEEE Press, 2006, pp. 1-4.
 7. Open NAND Flash Interface Group, *Open NAND Flash Interface Specification 2.0*, 27 Feb. 2008; <http://www.onfi.org/specifications>.
 8. A. Birrell et al., *A Design for High-Performance Flash Disks*, tech. report MSR-TR-2005-176, Microsoft Research, 2005.
 9. N. Agrawal et al., "Design Tradeoffs for SSD Performance," *Proc. Usenix 2008 Ann. Tech. Conf.*, Usenix Assoc., 2008; <http://research.microsoft.com/pubs/63596/USENIX-08-SSD.pdf>.
 10. S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google File System," *ACM Special Interest Group on Operating Systems' (SIGOPS) Operating Systems Rev.*, vol. 37, no. 5, 2003, pp. 29-43.
 11. T.R. Halfhill, "Intel's Tiny Atom," *Microprocessor Report*, Apr. 2008.
 12. Intel, *Quad-Core Intel Xeon Processor 3200 Series Datasheet*, 2007; http://www.intel.com/Assets/en_US/PDF/datasheet/316133.pdf.
 13. G. Ganger, B. Worthington, and Y. Patt, *DiskSim*, tech. report, Parallel Data Lab, Carnegie Mellon Univ., 2009; <http://www.pdl.cmu.edu/DiskSim>.
 14. X. Fan, W.-D. Weber, and L.A. Barroso, "Power Provisioning for a Warehouse-Sized Computer," *Proc. Int'l Symp. Computer Architecture (ISCA 07)*, ACM Press, 2007, pp. 13-23.
 15. D. Economou et al., "Full-System Power Analysis and Modeling for Server Environments," white paper, Hewlett-Packard, June 2006; <http://whitepapers.zdnet.com/abstract.aspx?docid=347834>.
- Adrian M. Caulfield** is a doctoral student in computer engineering at the Nonvolatile Systems Laboratory at the University of California, San Diego. He's interested in all aspects of computer architecture, but focuses mainly on the integration of nonvolatile memory technologies into system architectures to enhance the performance and energy efficiency of computer systems. Caulfield received a BS in computer science from the University of Washington.
- Laura M. Grupp** is a third-year doctoral student in the department of Computer Science and Engineering at the University of California, San Diego. Her current research focuses on the intricacies of flash translation layers, the development of models for new types of nonvolatile memory, and the characterization of flash memory. Grupp received a BS in electrical engineering from the University of Washington.
- Steven Swanson** is an assistant professor in the Computer Science and Engineering Department at the University of California, San Diego, and leads the Nonvolatile Systems Laboratory. His research interests include architectural, systems, and security issues surrounding nonvolatile, solid-state memories as well as massively heterogeneous multiprocessor architectures. Swanson has a PhD in computer science from the University of Washington. He is a member of the ACM.
- Direct questions and comments to Steven Swanson, Computer Science and Engineering Dept., Univ. of California, San Diego, 9500 Gilman Dr. #0404, La Jolla, CA 92093-0404; swanson@cs.ucsd.edu.

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