Gordon: Using Flash Memory to Build Fast, Power-efficient Clusters for Data-intensive Applications

PROBLEM STATEMENT

Data generation in recent years has been increasing at an insatiable rate. Combining this factor with the objective of decreasing global power consumption has presented us with three major challenges. First is the slowdown in uniprocessor performance in accordance with Moore's Law. Second is our debility of increasing the latency and bandwidth concomitantly with the hard drive capacity. And finally, the solution of the above two problems is trammeled by power constraints due to cooling, economic and ecological concerns. Hence, the aim of current technological front is to present a parallel, data-centric solution, preferably leveraging solid-state disks and low power processors. Gordon is exactly this.

PREVIOUS WORK

We have had some progress on each of the three problematic zones, described previously. First is MapReduce and Dyrad, which largely automate the task of parallelizing data-processing programs by providing simple abstractions for specifying data-parallel computations. MapReduce works on a set of key-value pairs while Dyrad is a generalization of UNIX pipes. Next, the bandwidth and latency issue can by largely tackled by introducing Solid-State Devices, which is fundamental in data-centric computing. Finally, capable and power-efficient processors have been developed recently by processor manufacturers.

Flash Memory, a type of Solid-State Device, is growing in popularity and is central to Gordon. Its application runs the gamut from storage in mobile device like iPods to digital cameras. It is cheap, has high density, requires low power and fall in the category of persistent storage. Although other aspects like bandwidth has not kept pace with its increasing density, over the years. Gordon provides a system architecture that exploits flash's unique capabilities.

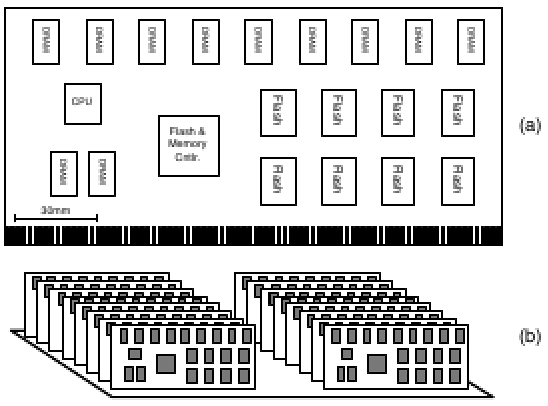
THE SOLUTION

As already said, Gordon is the answer to improving performance of data intensive applications by using low power processors over a Flash based storage. We will look at the Gordon's System architecture and its Storage system.

Gordon's System Architecture

A combination of a processor and its flash based storage is called a node. The high density of flash devices accedes compact Gordon nodes, which can run a complete operating system in itself (supporting minimal linux installation), acting as an independent computer. They communicate over conventional network interface (TCP/IP).

A schemetic of 16 nodes in a single enclosure and scale drawing a Gordon node PCB is as shown below.



Gordon's Storage System

The key to its performance and power efficiency is this Flash based Storage System. Let's look at current organization of flash devices and then we will look at how Gordon improvizes upon it.

We will consider the use of high density NAND flash devices. A block in a flash device is made up of 64 pages (SLC) or 128 pages (MLC). Each page consists of 2112 bytes for storing primary data, 'out of band' bad data and metadata. Each block is divided into two to four planes. Each plane has its own buffer to hold data for programs and reads, capable of performing some operations in parallel. A NAND flash device supports erase, program and read operation. Flash offers significant performance gains over conventional hard drives. Flash devices report erase and program failures due to wear out which can be minimized wear levelling, i.e. ensuring that blocks are erased with equal frequency.

Flash storage systems typically include a “flash translation layer” (FTL) that manages flash storage. The FTL provides wear leveling. It also maintains a layer of indirection between the logical block addresses (LBA) that the system uses to address data. Having looked at a general flash storage, lets look at Gordon's storage system.

The flash storage system comprises two components: a *flash controller* and *the flash memory* itself. Gordon’s FTL and its hardware interface to the flash storage array is implemented by the flash controller. Gordon's FTL is an extension of the FTL described above in the sense that it allows the application to write to and read from any logical block address (LBA) at random. The LBA table is held in volatile memory, but the FTL must keep a persistent version as well.

A *write point* is the position at which a program performs an operation. The aim is to allow multiple write points to enable operations in parallel, which is not possible in flash FTL. This improves write bandwidth significantly. A physical page from several dies are agglomerated to form a 'super page'. The pages can be combined in three ways: horizontal striping, vertical striping, and 2-dimensional (2D) striping. The size of a super page shouldn't be too large, otherwise it will increase the latency of sub-page accesses and wearouts will affect a much larger portion of the array.

We are done with describing the hardware and software of Gordon. Now, let's turn our attention to its performance and its capability of improving power efficiency.

ADVANTAGES OF THE SOLUTION

To motivate Gordon’s design, a set of benchmarks like MapReduce are used for parallel computation. Let's look at the performance and power consumption of Gordan on the used benchmark.

Cluster Performance

A high level trace driven simulator is used for measuring overall performance and storage system simulations is used for exploring architectural options for flash store arrays. The simulator processes a set of traces (one per node) in parallel. For one sample of data in the trace, it computes the time needed for instruction execution, disk accesses, and network transfers during that second. There Are two models for the purpose: *sync* and *nosync*. The simulator also measures the performance of p-way multi processor configuration. For latency and bandwidth storage simutations we use differennt simulations for flash and hard drives.

Power Consumption

Per component energy consumption is mostly uniform across the applications. The cost and power model for commodity systems is tabulated below.

	Capacity	Cost	Active power (W)
SAS disk	300GB	$340	15
SATA disk	1.5TB	$129	17
4GB DIMM	4GB	$193	6
FusionIO 160GB SLC	160GB	$2000	9
2.6GHz Intel Quad core	n/a	$500	60
Max DRAM enclosure	n/a	$1795	110
Max flash enclosure	n/a	$1025	90
Max disk enclosure	n/a	$1600	90

CONCLUSION

Fianlly, let's evaluate the trade-offs between power, performance, efficiency, and cost in the design of a Gordon node. To mitigate the issue of large cost involved in distributed and replicated file systems, some Gordon nodes can be combined to servers that have conventional hard disks and considering falsh storage as *replica cache*. Cost reduction is possible if the cluster based system uses commodity components. This is a relatively wise trade-off considering disks are slow and cheap. Thus there is little point in spending large sums to provide fast processors and exotic, high-speed networks. Flash offers huge gains relative to disk in terms of performance, efficiency, density and relative to DRAM in terms of density and power consumption. If these gains are sufficient to move a data-intensive application from off-line to on-line, flash could easily justify its extra cost.

Thus, Gordon demonstrates that flash affords the opportunity to re-engineer many aspects of system design, therefore enabling a new class of computing systems for data-intensive applications.