IDE Ultra/33 Performance: Intel PIIX4E

Final Report 1/8/99
Bruce Worthington
NT Performance Group
Microsoft Corporation
bworth@Microsoft.com

Summary

The Intel 82371AB PCI-to-ISA / IDE Xcelerator, also known as the PIIX4E, provides a bridge from dual IDE peripheral buses to the PCI bus (which accesses main memory via another Intel chipset). A current IDE bus can burst data at 33 MB/s. A 32-bit 33 MHz PCI can burst data at 133 MB/s. For IDE→PCI transfers, the PIIX4E can pretty much keep up with an IDE bus (up to 32.6 MB/s). However, it appears that the PIIX4E is limited to 21.9 MB/s for bursting data for PCI→IDE transfers. Adding in the setup / cleanup DMA costs (per request), the PIIX4E can sustain up to 21.4 MB/s for 64KB disk writes (i.e., DMA reads) and 28.5 MB/s for 64KB disk reads (i.e., DMA writes) on a Pentium-II system running Windows 2000 Professional.

In the case of disk reads, the 14% performance difference between specified and realized bandwidth is largely due to DMA setup / cleanup activity. Between requests, a number of PIO's (programmed I/O transfers, indicative of register accesses in this case) and processing delays reduce raw throughput to 28.5 MB/s for 64KB requests (29.1 MB/s for 128KB requests).

In the case of disk writes, the performance degradation is much more severe. A third of the available bandwidth is lost to delays embedded in the actual data transfer, with DMA set / cleanup activity claiming another 1-2% of the remaining bandwidth. Synchronized traces of PCI and IDE activity conclusively show that the PIIX4E is responsible for the data transfer delays.

System Configuration

The test system is a Gateway workstation with a 350 MHz Pentium-II (512KB L2, 128MB Memory). The primary disk drive is a 9.55GB Quantum Fireball (EL10.2A) on the secondary IDE bus. When multiple drive activity was necessary, a second Fireball was attached (to the IDE bus appropriate to the experiment).

DMA Setup / Cleanup

In between each disk request, a series of PIO's is issued by the host (driver) in order to check the status of the finished request and set up the next request. Table 1 (see page 2) gives the sequence of PIO's and some indication of intermediate delays.

- When a request completes, an interrupt is signaled. Thus, the first PCI transaction after the last data transfer is an Interrupt Acknowledgement.
- The subsequent series of PIO's is basically the same for reads and writes, although the delays in between the PIO's may vary.
- If an individual PIO cannot be satisfied within 9 PCI cycles, it is retried immediately (only 2 idle cycles). In some cases it took up to 3 PCI transactions to successfully complete a PIO.
- After the last setup PIO, an 8-byte Memory Read is initiated by the PIIX4E to obtain the size and location of the next DMA request. If the request is scattered across several contiguous physical memory locations (e.g., page frames), a similar 8-byte Memory Read is initiated after each contiguous region has been transferred. E.G., a 64KB request is typically scattered over 16 physical pages, and therefore an 8-byte Memory Read is performed after each 4KB of data is transferred.
- If more than 25 Idle PCI cycles typically precede a particular PIO (or Memory Read), the table notes the range of observed delays.
- Addresses Ranges:

Address	Register Set
002X	Interrupt Controller #1
00AX	Interrupt Controller #2
017X	IDE Drive Control Block
10AX	IDE DMA Secondary Channel Controller

PCI Transaction	Target Address (0x00)	Bytes Transferred	Preceding Delay (µs)	Notes (e.g., Register Name, Data, Action, Reason for Delay)
Interrupt Acknowledge			1-2	
PIO Write	00A0	1	1-2	OCW3: Select ISR
PIO Read	00A0	1		ISR: Interrupts Serviced
PIO Write	00A0	1		OCW2: Clear Interrupt
PIO Write	0020	1		OCW2: Clear Interrupt
PIO Read	10AA	1	1-2	BMISX: Status (Interrupt Sent!)
PIO Read	10AA	1		BMISX: Status (Interrupt Sent!)
PIO Write	10A8	1		BMICX: Stop Bus Master
PIO Read	0177	1		Drive Status: Ready & ~Busy
PIO Read	0177	1	1-2	Drive Status: Ready & ~Busy
PIO Write	10A8	1	**15-125**	BMICX: Stop Bus Master
PIO Write	10AA	1		BMISX: Clear Interrupt
PIO Write	10AC	4		BMIDTPX: Set Descriptor Ptr
PIO Write	0176	1	1-3	Drive Drive/Head: Select Drive
PIO Read	0177	1		Drive Status: Ready & ~Busy
PIO Write	0172	1		Drive Sectors: 64KB
PIO Write	0176	1		Drive Drive/Head: LBA Mode & LBA [27:24]
PIO Write	0173	1		Drive LBA[0:7]
PIO Write	0174	1		Drive LBA[15:8]
PIO Write	0175	1		Drive LBA[23:16]
PIO Write	0177	1		Drive Cmd: DMA Read/Write
PIO Write	10AA	1		BMISX: Clear Interrupt
PIO Write	10A8	1		BMICX: Initiate DMA R/W
Memory Read	Descriptor Addr	8		DMA Address and Byte Count
Special Cycle (Disk Reads only)	Msg: 00120001	4	**100** (sum of before and after)	Special Cycle occurs only during Disk Reads (i.e., DMA writes)
First Data Transfer	DMA Address	32	1-2	MemWriteInv or MemRead

Table 1

Single Disk Read Performance

The PIIX4E has a 64-byte speed-matching FIFO. It accesses the PCI bus in bursts of 32 bytes, emptying half of the FIFO on each transfer. Table 2 shows a typical sequence of PCI activity (initiated by the PIIX4E) covering 32 bytes of data transfer.

PCI Phase	Duration (cycles)	Notes
Addr/Cmd: Memory Write	1	
Initiator Ready, Waiting on Target (Memory Ctlr)	1	Medium-speed DEVSEL
Data Transfer (32 bytes)	8	Xfer stopped by PIIX4E
Idle	22-23	Speed match IDE → PCI

Table 2

32 bytes transferred in 32 cycles == 21.9 MB/s. For a 64KB transfer, the resulting data transfer phase lasts almost exactly 2 ms. The DMA setup / cleanup activity takes approximately 200 µs, which results in an overall throughput of 29.8 MB/s, assuming no contention for the IDE or PCI buses.

Single Disk Write Performance

The PIIX4E has a 64-byte speed-matching FIFO. It starts off by requesting 64 bytes from memory[1], but after 32 bytes (a cache line) the target (i.e., the memory controller) stops the transfer. The remainder of the FIFO is filled in a subsequent request. Thus, the behavior observed on the PCI bus has a 64-byte cycle. Table 3 shows a typical sequence of PCI activity (initiated by the PIIX4E) covering 64 bytes of data transfer.

PCI Phase	Duration (cycles)[2]	Notes
Addr/Cmd: Memory Read	1	
Initiator Ready, Waiting on Target (Memory Ctlr)	13	P6 Bus & Memory Accesses
Data Transfer (32 bytes)	8	
Target (Memory Ctlr) Disconnection	2	Cache Line Boundary
Idle	12	
Addr/Cmd: Memory Read	1	
Initiator Ready, Waiting on Target (Memory Ctlr)	13	
Data Transfer (32 bytes)	8	Xfer stopped by PIIX4E
Idle	40	

Table 3

64 bytes transferred in 98 cycles (~2925 ns on this particular PCI bus) == 21.9 MB/s. For a 64KB transfer, the resulting data transfer phase lasts almost exactly 3 ms. The DMA setup / cleanup activity can be performed (best case) in less than 50 µs, which results in an overall throughput of 21.5 MB/s, assuming no contention for the IDE or PCI buses.

By simultaneously observing the PCI and IDE buses, it becomes obvious that the PIIX4E is not effectively pipelining the data transfer. Specifically, after each 64-byte transfer the PIIX4E is not reconnecting back to memory in time to keep the IDE bus transferring data at its maximum rate. Figure 1 (see page 4) shows a timeline of PCI and IDE activity covering one 64-byte cycle.

- The PIIX4E continues the ongoing IDE transfer as soon as it has received 32 bytes from the host.
- The host forces a PCI disconnection after 32 bytes have been transferred, but the PIIX4E reconnects quickly and the second 32 bytes are transferred to the PIIX4E soon enough such that only a minor disruption occurs in the IDE transfer (an additional 30 ns between the 31st/32nd bytes and the 33rd/34th bytes).
- The PIIX4E terminates the PCI transaction itself after the second 32 bytes have been received.
- At this point, the PIIX4E finishes the 64-byte IDE transfer (~800 ns remaining) and then waits an additional 400 ns before requesting more data from the host. This inserts a huge bubble (~1000 ns) in the data transfer pipeline, as evidenced by a long pause in the full-speed IDE transfer after every 64 bytes.

[1] PCI Read transactions do not actually request a specific amount of data. However, the PIIX4E documentation states that 64 bytes (enough to fill the PIIX4E's data FIFO) is the amount of data that the PIIX4E hopes to obtain from memory during a DMA Read. This is supported by the fact that the PIIX4E stops data transfer from memory after every 64 bytes of transferred data (even though the host /memory stops the data transfer after every 32 bytes).

[2] A 33 MHz PCI bus has a cycle time of ~30ns, as compared to the system bus which runs at 100 MHz (10ns) and the processor which runs at 450 MHz (2.2ns).

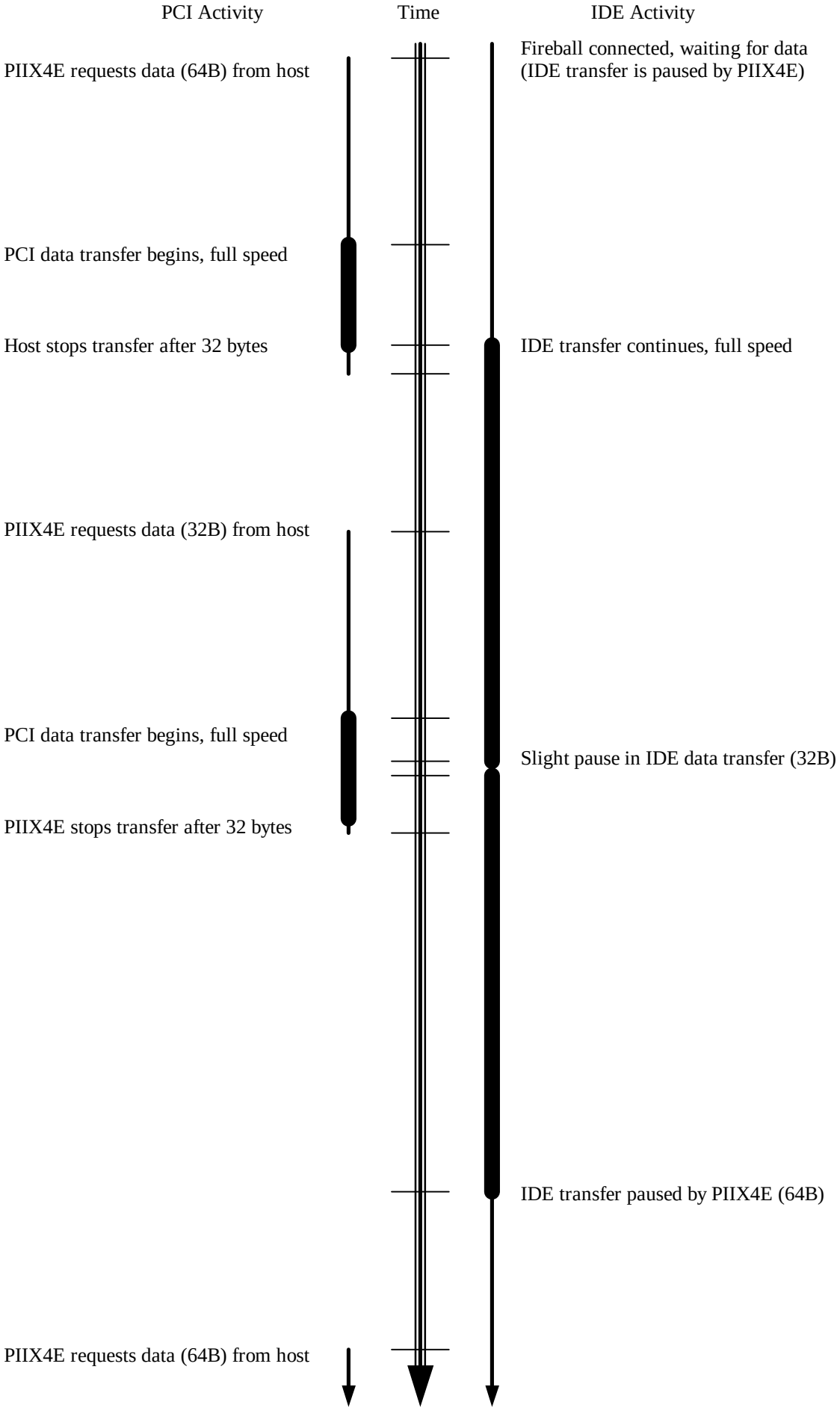

PCI Activity
Time
IDE Activity
PIIX4E requests data (64B) from host
Fireball connected, waiting for data
(IDE transfer is paused by PIIX4E)
PCI data transfer begins, full speed
Host stops transfer after 32 bytes
IDE transfer continues, full speed
PIIX4E requests data (32B) from host
PCI data transfer begins, full speed
Slight pause in IDE data transfer (32B)
PIIX4E stops transfer after 32 bytes
IDE transfer paused by PIIX4E (64B)
PIIX4E requests data (64B) from host

Figure 1

Multiple Disk Read Performance

The maximum read performance was observed for large requests that hit in the Fireball's on-board cache. For such a workload, a single disk is able to saturate the IDE bus and PIIX4E bridge. For smaller requests or requests that require disk media access, it is not so easy to saturate the IDE bus. A number of experiments were performed using the SQLIO disk request generator. The request size was varied from 2KB to 64KB; both sequential and random requests were generated; and a second Fireball was attached either to the same IDE bus as the primary Fireball or to the second IDE bus (hanging off the same PIIX4E chip).

Figure 2(a) shows throughput numbers for workloads made up of random disk reads. Mechanical latencies (i.e., seek and rotation times) dominate performance for this workload, reducing the single-disk throughput to less than 5 MB/s for 64KB requests. Throughput drops with request size (down to 0.27 MB/s for 2KB requests). Adding a second disk to the same IDE bus results in almost no performance improvement because each IDE bus can only have a single disk request in progress. On the other hand, adding a disk to the opposite IDE bus doubles the total throughput across the spectrum of request sizes.

Figure 2(c) shows throughput data for workloads made up of sequential disk reads. For a single Fireball, the overall throughput is limited by the speed at which the disk media can be accessed – just under 11 MB/s. For all but 2 KB requests (which have the highest overhead per byte transferred), adding a second disk on either IDE bus doubles the overall throughput. Since the Fireball prefetches sequential data into its on-board buffer/cache, even two disks on a single IDE bus are able to effectively work in parallel.

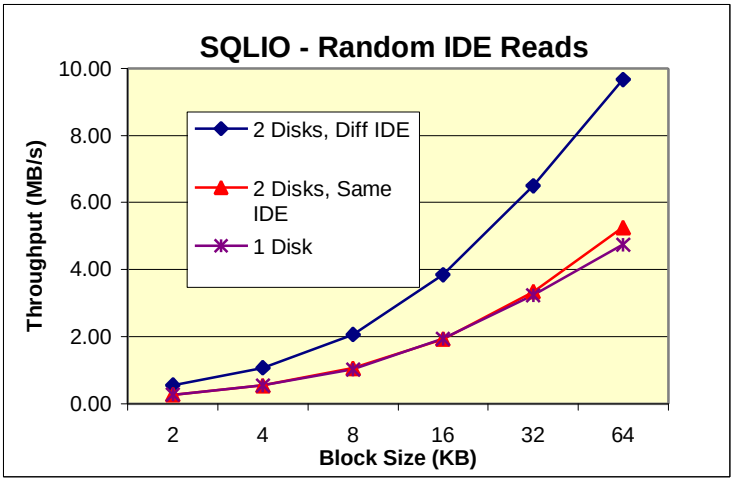
Figure 2(e) shows throughput data for workloads made up of disk reads to a single (repeated) disk location. This allows all requests to be serviced from the Fireball's on-board cache – no media access is required. The behavior of two disks under this workload is a bit less intuitive. An additional disk on the same IDE bus gives little added throughput, as might be expected since there is no opportunity for useful parallelism between the two disks' activity. But a second disk on the complementary IDE bus significantly improves performance for small request sizes. Since DMA setup / cleanup overheads are more significant for smaller requests, the parallelism between the two per-IDE DMA controllers provides a performance boost. However, throughput is **reduced** for this configuration when the request size grows larger than 16 KB. This would seem to indicate that there is a penalty involved in switching back and forth between the two IDE channels (between requests). This penalty is only visible in this particular scenario – the scenario with the highest potential throughput and therefore the scenario most sensitive to such penalties…

Multiple Disk Write Performance

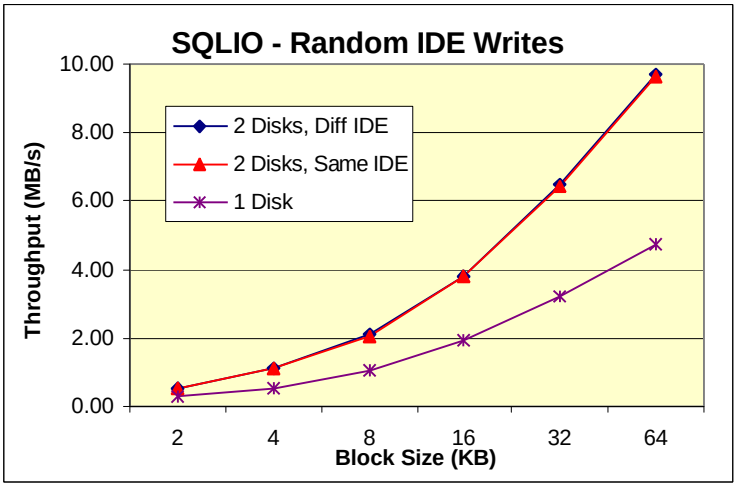
Figure 2(b) shows throughput numbers for workloads made up of random disk writes. The Fireball default configuration has write caching enabled, which allows the Fireball to disconnect from the IDE bus as soon as all of the write data has been transferred to the on-board buffer/cache. This allows a second drive on the same IDE bus to process a second write request while the first request is being written to the disk media. Thus, a second drive on the same or on the complementary IDE bus provides double the performance.

Figure 2(d) shows a similar level of performance improvement for sequential disk writes, although the performance improvement tapers off for smaller requests if the second disk is on the same IDE bus. This is due to the DMA setup / cleanup overhead taking a larger fraction of the request service time as the request size decreases.

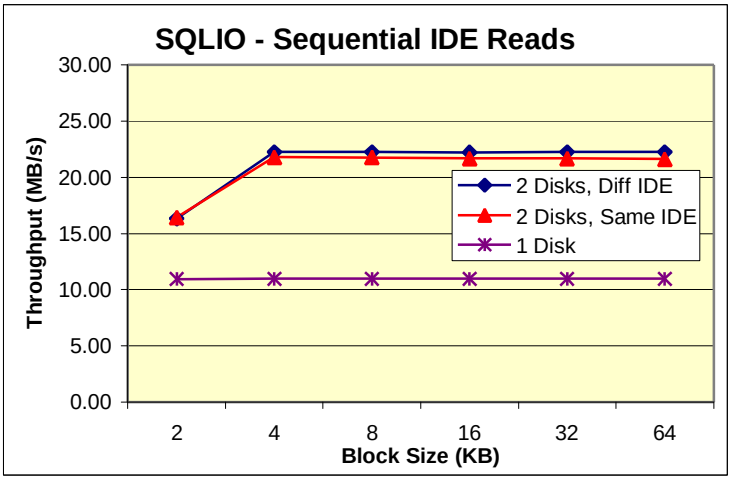
Figure 2(f) shows throughput data for workloads made up of disk writes to a single (repeated) disk location. The Fireball apparently allows data to be overwritten in the on-board cache, since the single-disk throughput for 64KB requests exceeds the maximum media transfer rate. Nevertheless, a second disk does improve performance for smaller request sizes – probably due to parallelism in DMA setup / cleanup and command processing at the disks themselves. The limitation of the PIIX4E in regards to disk writes (discussed above) puts a hard limit of approximately 21 MB/s on overall throughput.



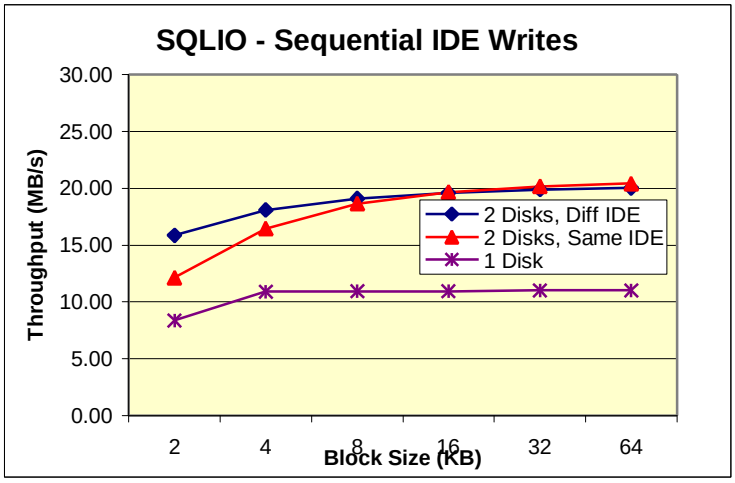
(a)



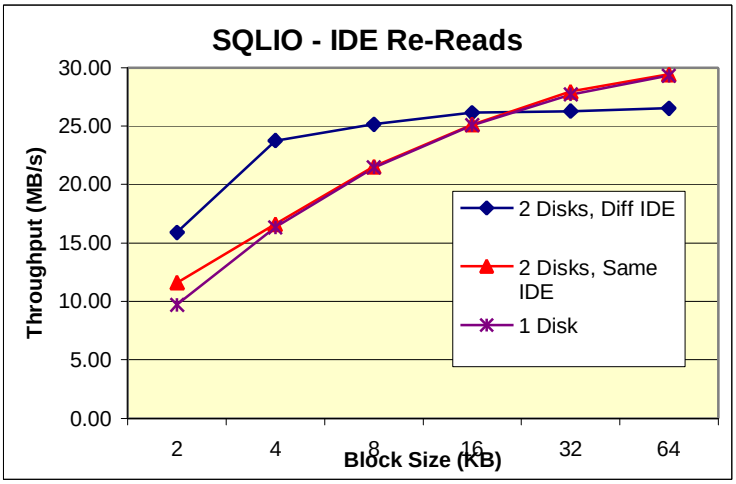
(b)



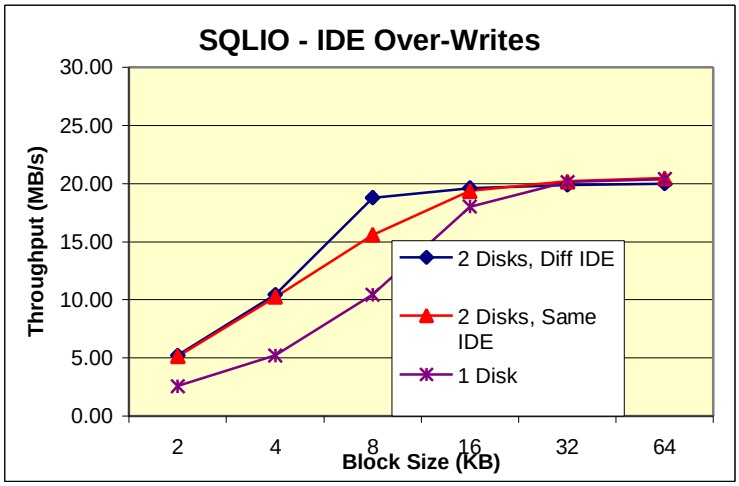
(c)



(d)



(e)



(f)

Figure 2

Final Notes and Acknowledgements

- Patrick Franklin pointed out another problem with the PIIX4E. Several of the I/O-space registers have fixed addresses (i.e., they cannot be relocated or assigned different addresses). As the PCI specification specifically provides for multiple base registers in the PCI configuration space (for ease in the relocation of addresses), there is no real excuse for this inflexibility. Furthermore, this prevents the possibility of including multiple PIIX4E chips to provide for additional parallelism between IDE peripherals. One may make the argument that IDE is only used for systems with few disks and low performance requirements; so multiple IDE bridges are not a realistic option. However, with Ultra66 DMA coming soon, and IDE disks making up the large majority of installed workstation drives, improving IDE performance via multiple bridges may be a realistic option after all...
- The results of this study will be passed along to Intel in the hope that future IDE bridges will not exhibit similar performance degradation. This prospect is especially worrisome as Ultra66 IDE is just around the corner; it would be a terrible shame to restrict a 66 MB/s peripheral bus to a mere 20 MB/s during disk writes.
- This study was prompted by the observations of Jim Gray (Microsoft Research). Initial analysis using his test system provided the data that focused my attention on the PIIX4E chip. Jim called in Bob Rapp from Quantum to perform some initial tracing of the IDE bus. Before we knew which hardware component was causing the performance degradation, the data they collected helped to exonerate the Fireball.