Reprinted from the

Proceedings of the Linux Symposium

July 23th–26th, 2003
Ottawa, Ontario
Canada

Conference Organizers

Review Committee

Proceedings Formatting Team

DMA Hints on IA64/PARISC

Optimizing DMA performance for HP Chip sets

Grant Grundler
Hewlett Packard
grundler@cup.hp.com

Abstract

Modern IO subsystems implement complex DMA transaction parameters, called DMA hints, which are not explicitly supported by the Linux DMA API. This paper investigates benefits of using non-default DMA hints and thus whether such hints should be abstracted into the DMA API. My conclusion is the implementation (ZX1) investigated does not warrant changing the DMA API. Other implementations need to be compared before proposing any changes.

HP PA-RISC (Astro[1]/Elroy[2]) and IA64 IO Controllers (ZX1) both support several types of DMA hints and both are commercially available. My primary interest was the ability to prefetch cache lines for PCI devices. The benefit is same as for CPU: bring the data closer to the consumer. But to my surprise, cache line prefetching is not the most important hint since default prefetching works well for all devices. Relaxing the PCI ordering rules turns out to be more important since firmware can't know when it's safe to do so.

Updated versions of this paper will be available from `http://iou.parisc-linux.org/ols2003/`

1 Introduction

DMA performance seems like such an obvious thing. Drivers just need to tell the device where to fetch something from memory, poke it, and life is good. Unfortunately, those days are over.

Modern SMP servers require multiple levels of bridges in order to support PCI-X Bandwidth (peak burst rate 133MHz/64-bits). In order to work well with CPUs and memory controllers, IO Devices participate in the CPU Cache Coherency protocols. They also need to minimize the number transactions used and use the appropriate type of transaction in order to optimally utilize available bandwidth.

Throughout this paper (and even in the title!) I use the word *Hint* which implies an "informational only" parameter. This isn't strictly accurate. Some platforms depend on certain parameters for correct operation. I.e. incorrect results may occur for some combinations of DMA hints. The DMA hints discussed in this paper *should* always provide correct results though I've crashed the ZX1 with some hints as noted.

And I recycled the ZX1 block diagram used in my 2002 OLS talk, "Porting Drivers to ZX1."[3] The diagram is useful to understand the routing of data between PCI devices, Memory, and CPU. [4]

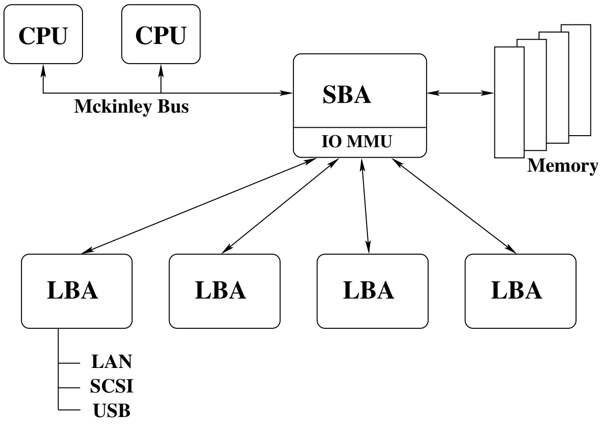

Figure 1: HP ZX1 Block Diagram

2 Overview of DMA

The following sections introduce some of the key concepts relating to Direct Memory Access.

2.1 Consistent vs. Streaming DMA Mappings

The Linux DMA mapping interface differentiates between two memory access patterns. A short summary of `DMA-mapping.txt`[5] follows.

Consistent DMA mappings are intended for data which is concurrently accessed by both CPU and PCI device(s) (i.e. Host RAM base device control structures like mailbox rings). Key feature is updates (writes) from either must be visible to the other based on PCI ordering rules. In short, fairly strict R/W ordering rules and transactions are typically less than a cache line in length.

Streaming DMA mappings are intended for memory regions exclusively accessed by the PCI device(s). This "exclusive" access begins when a host memory region is mapped and ends when the same region is unmapped.

The Streaming DMA interface provides two explicit hints: DMA direction and DMA length. From the length, we know the block size on which the DMA will terminate. But as noted in the introduction, DMA direction is required for correct operation on some platforms—but not ZX1 or PARISC IOMMUs.[1] The ZX1 *System Bus Adapter* (aka SBA; See `sba_iommu.c`) code does optionally use direction to optimize VM bits. Other hints regarding PCI Ordering compliance and DMA Read data consumption rate are not specified.

2.2 PCI DMA

A single DMA operation is fairly straight forward at the PCI bus level. The PCI Device asks the PCI arbiter for bus ownership. The Arbiter eventually grants the PCI device ownership of the bus. PCI device accepts ownership and sends the target address (possible two cycles worth for 64-bit addressing) followed by data. The transaction ends when either the PCI Controller asks the device or the PCI device volunteers to give ownership.

PCI supports several different types of Commands. Here are the ones relating to DMA along with summaries of their PCI Local bus definition:

- **Memory Read** command is used to read data from an agent mapped in the Memory Address Space. The target is free to do an anticipatory read for this command only if it can guarantee that such a read will have no side effects. Furthermore, the target must ensure the coherency (which includes ordering) of any data retained in temporary buffers after this PCI transaction is completed.

- **Memory Read Line** command is seman-

[1] PARISC platforms without IOMMU do require R/W direction hint.

tically identical to the Memory Read command. Use of MRL indicates the intention to read a full cache line of data.

- **Memory Read Multiple** command is semantically identical to the Memory Read command. Use of MRM indicates the intention to read more than one cache line of data before disconnecting. MRM is intended to be used with bulk sequential data transfers where the memory system (and the requesting master) might gain some performance advantage by sequentially reading ahead one or more additional cache line(s) when a software transparent buffer is available for temporary storage.

- **Memory Write** command is used to write data to an agent mapped in the Memory Address Space. When the target returns "ready," it has assumed responsibility for the coherency (which includes ordering) of the subject data.

- **Memory Write and Invalidate** command is semantically identical to the Memory Write command. The difference is MWI requires the device to write at least one complete cache line and the Host Cache controller can invalidate existing contents without having to send the contents (just reassign ownership) to the IO or Memory Controller. This avoids unnecessary cycles on the Front Side Bus for DMA writes. MWI requires `CACHELINE_SIZE` register in the device configuration space to (a) be implemented and (b) programmed by BIOS or PCI Initialization code to a suitable value.

The number of bytes transferred is constrained by the transfer type (MR, MRL, MRM, MW, MWI) and `LATENCY_TIMER` value. The `LATENCY_TIMER` is described briefly later in this paper and by the PCI Local Bus Specification.

Memory Writes are typically the simpler case from a software performance perspective. DMA Writes are buffered by the chip set and routed to the memory controller[6] at whatever rate the internal interconnect supports. Data throughput is typically limited by the PCI bus controller[7] or memory controller.

Reads are more complicated because the memory controller latency is harder to hide. All data handling systems (like disk IO) deal with this problem by "Read Ahead" (a.k.a. prefetching) or caching. However, large caches like those implemented in a CPU are expensive in many ways. And large caches don't help much since the read and write access patterns for IO devices typically aren't for the same cache lines repeatedly. Or in the case of shared data, the IO device competes at times with the CPU for the cache line.

Successive requests for bulk data can be prefetched by the I/O Controller. I was expecting prefetching to make a big difference for PCI devices. But the individual devices I tested (except 53c1010 Consistent DMA) did not perform better or worse for different levels of prefetching. Like disk IO, the effectiveness of the prefetching really depends on the data access pattern. I suspect this is because the PCI devices are designed to work well with out any prefetching and buffer enough data to keep the IO device from stalling.

2.3 DMA on PCI-X

PCI-X obsoletes much of what I was trying to accomplish in this paper. Cache line prefetching hint only applies to PCI. But three new PCI-X-only features are of interest:

- *Attributes* are part of the PCI-X command.

- *Split Transactions* replace the goofy "retry forever" schemes used when read data is not available.
- *Burst Transactions* replace MRM, MRL, and MWI.

2.3.1 Command Attributes

The PCI-X Specification[8] (drafts are available for free) has more details about command attributes in section *2.3 PCI-X Command Encoding*. I'll try to summarize below. Two attributes are currently defined for PCI-X commands.

Relaxed Ordering is also described in Section 4.1. In a nutshell, ignoring the PCI ordering rules regarding Programmed I/O (CPU R/W) and DMA (PCI R/W) yields measurable performance gains without sacrificing correct operation. The handful of drivers I've used under PARISC-Linux and IA64-Linux run just fine with outbound data[2] ordering rules relaxed.

Note the PCI-X spec doesn't require the PCI-X device to use Relaxed Ordering attribute when the Relaxed Ordering bit is set in the command register. ZX1 chip can override the Command Attribute for Relaxed Ordering behavior. And ZX1 chip set only implements this optimization for outbound data flow (PIO Write/DMA Read return). The PCI-X spec defines optimizations in both directions of data flow.

No Snoop attribute isn't relevant for most IO devices. Most Linux drivers expect DMA transactions under control of DMA mapping services to be coherent. "No Snoop" means the host driver guarantees the latest copy of a cache line is in the memory controller. And it implies the chip set can perform better if it doesn't have to Snoop. My understanding is non-coherent transaction are interesting for graphics devices, not the LAN/Storage devices I work with.

2.3.2 Split Transactions

Split Transactions just means the request for information and the completion (reply) of that request are separate transactions on the bus. This is a good thing for several reasons:

- Up to 32 transactions can be pending at the same time. Only 5 bits are defined in the *Tag* field of the PCI-X command. But the exact number of outstanding transactions supported depends on chip set implementation. This is identical in concept to *Tagged Queues* defined for SCSI protocol.
- More efficient: eliminates the need to poll (aka "retry forever") when the PCI Controller disconnects in the middle of a (e.g.) read transaction.
- Acts more like a Memory bus rather than an IO bus. Thus, it's easier for HW designers to route transactions across a larger fabric.

2.3.3 Burst Transactions

Memory Read Block (MRB) and **Memory Write Block** (MWB) are replacements for MRL/MRM and MWI respectively. The key difference between the above PCI and PCI-X commands is addition of a *Byte Count* field. By making the transfer length visible to the PCI-X controller, the chipset can prefetch cache lines appropriately. This is significant since not involving driver writers for 4 or 5 different OSs to program DMA hint bits is a good thing.

[2] Inbound data reordering causes Bad Things™ to happen. Discussed on ia64-linux mailing list.

3 DMA Parameters

Two additional parameters defined by PCI Local Bus specification affect DMA behavior. Both reside in the PCI device configuration space header: `LATENCY_TIMER` and `CACHELINE_SIZE`.

3.1 LATENCY_TIMER

`LATENCY_TIMER` constrains how long the PCI device will burst DMA before volunteering to give up the PCI bus. It should be long enough to transfer several cache lines of data if the device is capable. While this is a "tunable" parameter, I didn't feel it was necessary to experiment with this value since `LATENCY_TIMER` is a pretty well understood and described else where.

3.2 CACHELINE_SIZE

`CACHELINE_SIZE` is only required by PCI MWI command. `CACHELINE_SIZE` has to be the IO cache line size and not the CPU size. Typically this is either the line size of the memory controller or the line size of outer most CPU cache (e.g. L2 or L3 Cache). The CPU can use smaller lines for first and/or second level caches.

Since firmware is expected to program `CACHELINE_SIZE` with the appropriate value, it's a not really either a parameter nor tunable.

4 HP ZX1 DMA Hint bits

The HP ZX1 chip set implements several bits for DMA Hints. None of these are supported by the current Linux DMA mapping API. My original goal was to propose extensions to the Linux DMA mapping API. But I've concluded it's not absolutely necessary and I don't know which hints are of interest to other chip sets. Hopefully this paper will precipitate more discussion and comparison of chips and capabilities.

The primary reason it's not absolutely necessary is IA64 firmware is compensating for an ignorant OS. Firmware errs on the overly aggressive side in setting the cache line prefetching (for simple, single unit tests) and errs conservatively on the correctness case (Relaxed Ordering is disabled). Though it's not optimal, ZX1 IOMMU code could blindly set Relaxed Ordering hint bit (i.e. not enforce ordering) and some hacks can take care of the others.

And because PCI-X obsoletes the key PCI-only hint, I can't argue HP needs them. My pet architecture (PA-RISC) would benefit since HW shipped to date only supports PCI—but that's not of commercial interest. And PA-RISC alone is not a justification for extensions to the interface.

Even if I had HW descriptions for other IOMMUs, it would be a lot of work to independently abstract the DMA hints. Understanding platform IOMMU support well enough to abstract what's important is non-trivial. And since I'm fundamentally lazy (or "good at optimizing" as Bdale Garbee puts it), I'll pass for now.

Lastly, assuming hints are chip set specific (since no one has abstracted them), introducing hints for each chip set is the path to hell for driver writers. A relatively small number of people (per OS) understand how one IOMMU on one platform works. Trying to get a broader audience to understand several platform chip sets is unrealistic. Been there, done that.

4.1 Relaxed Ordering

Relaxed Ordering tells the HW it can ignore one PCI ordering rule. PCI-X specification offers this optimization in each direction (not just outbound) with its definition of Relaxed Ordering. HP's chip set is sufficiently well implemented in the inbound (DMA writes) path that the inbound optimization isn't helpful and thus not implemented.

HP also calls this optimization "PIOW/DMAR Ordering." The cryptic acronym means "Programmed IO Writes/DMA Reads" Ordering. Setting this hint indicates the driver and device don't depend on ordering of DMA Read returns and PIO Writes for correct operation. This hint allows DMA read returns to bypass PIO Writes in order to prevent an in-progress DMA burst from disconnecting on the PCI bus and force retries.

I didn't have any expectations for this hint. Mostly because of my ignorance when I started this investigation.

4.2 Read Current

Read Current transactions gets the most recent copy of cache line data *without changing the cache line state*. The key thing is the CPU can keep cache line ownership. By not giving up ownership, the CPU can continue to modify cache line contents without having to fight with the IO Controller (ping pong) for the cache line. However, the copy of the cache line is not maintained and can become stale. It should be consumed immediately (for some finite definition of *NOW*; parents will understand). Thus it's most useful when data is leaving the cache coherency domain (i.e. DMA reads).

Most driver writers will not need (or want) to worry about Read Current hint. First, different chip-sets have minor variations in implementation which may in fact still ping-pong the cache line. Secondly, Read Current hint has no effect on chip sets which already implement DMA reads by issuing Read Current transactions. And third, to date, none of the PCI devices that interest me obviously benefited by explicitly setting or clearing this hint bit on the platforms I've tested.

Read Current is implemented on both PARISC Runway[9] and McKinley bus.

4.3 Cache line Prefetching (PCI Only)

Cache line Prefetching for IO devices serves the same purpose as prefetching does for CPUs: avoid stalling by bringing data closer before it's needed. The amount of prefetch needed is a function of the device's *data consumption rate* and the actual memory controller latency.

For example, if the memory controller can deliver a cache line in 120ns and the device can consume a cache line in 120ns (8 * 15ns), we need to prefetch 1 cache line at any given moment in time. In other words, 2 cache lines of data will be in flight at any given moment in time. But as system workload increases, the average memory controller latency usually goes up too. It might really take 200 or 300ns to deliver a cache line. We need to compromise and pick the number of cache lines to prefetch so things work OK under worst case but perform optimally in the "expected work load" range.

ZX1 chip set can deliver a 128 byte cache line in about 110 ns[10] PCI devices can only consume 128 bytes every 240 ns (16 * 15ns) at best. The PCI device probably stalls less than 1/3 of the time waiting for data. This is substantially different than for the PARISC chip set which can only deliver a 64 byte cache line in about 180 ns.[11]

PCI-X mode of operation does NOT support cache line prefetching hints. It's not necessary because with split transactions, the device can have much more IO outstanding and in effect, perform its own prefetching.

4.4 DMA Block Size (PCI Only)

DMA Block Size tells the chip set when to stop prefetching. Prefetching will continue up to the block size boundary and resume when the first cache line of the next block is requested.

This hint also does not apply to PCI-X busses. It's not necessary because the PCI-X *Burst Transactions* specify the number of bytes being transferred and the chip set (or OS code) doesn't have to guess when to stop prefetching.

I didn't expect block size to matter much in single unit testing. It would be interesting to know how much it matters when testing a fully loaded system.

5 Case Study: BCM5701 (PCI)

In 2002, around the same time HP ZX1 products became available, HP started shipping Tigon3 NICs (designed and tested by HP). The BCM5701 NIC supported by HP-UX is shipped operating in PCI mode.

Test used is:

```
/opt/netperf/netperf -l 60 \
-H 10.0.30.0 -t UDP_STREAM -- \
-m 1024 -s 131072 -S 131072
```

UDP_STREAM is useful for testing output if the host networking stack only sends what the NIC can consume. I'm told this is the case for Linux. And while some applications really do run on top of UDP, I also ran TCP_STREAM test to get an idea of the workloads I'm familiar with.

Client (HP RX2600, 1GHz) was running 2.4.20-em19 + tg3 v1.5 over the built-in BCM5701.
Server (HP RX2600, 900MHz) was running the same kernel, same built-in BCM5701.

NICs were connected via cross-over cable and set to either 1500 or 9000 bytes MTU.

Firmware sets the default PCI Command Hint to 3 cache lines prefetch, Relaxed Ordering disabled, 4k block size, Read Current enabled.

I then varied the DMA Hints on the *Client* who was sending packets. While this sounds backwards, it's the netperf point of view. We want to observe the netperf "client" send performance.

5.1 Relaxed Ordering

Relaxed Ordering Hint is (on) enforced by default. I've turned it off selectively for MR, MRL and MRM PCI transactions in Table 1. Runs with 2.4.20+tg3 v1.5 only showed about 4% improvement with 1k messages.

Previous experience with UDP_STREAM testing on RHAS 2.1 (IA64, e.25?, using tg3 v0.99) demonstrated nearly 10% performance improvement with 1080 byte messages. Without ordering enforced, netperf reported 862 Mb/s[3] vs. around 775 Mb/s when ordering was enforced (default behavior).

TCP Stream test showed a smaller, but similar, sensitivity to this parameter. Clearing Relaxed Ordering hint in the MRM hint for Streaming DMA resulted in 778 Mb/s (vs. ~758 normally) using 1024 byte message and 1500 byte MTU.

[3] Or 848 Mb/s when the netperf client ran on the same CPU as the one interrupts were directed at.

PCI Cmd	Consistent	Streaming
NONE	759.61	758.74
MR	759.09	760.67
MRL	759.99	759.70
MRM	759.68	797.19
ALL	758.99	800.65

Table 1: BCM5701 UDP_STREAM Relaxed Ordering, 1k Msg

5.2 Cache line Prefetching

Neither TCP nor UDP showed any statistically significant differences as I varied the cache line prefetching for MR, MRL, or MRM commands. This was true for both 1k (1500 byte MTU) and 8k (9000 byte MTU) message sizes.

I suspect this is primarily because I'm measuring what the card is buffering, not PCI bus utilization. The card's ability to buffer is not affected by how inefficient the PCI bus is used. Unfortunately, I don't have the tools to measure PCI Bus utilization on the RX2600.

Again, like learning PCI-X obsoletes cache line prefetching, this is a disappointing but useful result.

5.3 DMA Block Size

Unlike cache line prefetching, I didn't expect much difference between the various block sizes. Once I knew prefetching makes no difference for the BCM5701, the fact that varying Block size hint also makes no difference was no surprise.

5.4 Read Current

Disabling Read Current Hint for Consistent mappings will crash the system. It's not clear to me why. I talked with the HW designers and it is clearly not an expected result due to how the read/write paths are implemented.

I wasn't expecting any measurable performance difference with (vs. without) Read Current for Streaming DMA. And in fact, I didn't see any.

6 Case Study: BCM5704S (PCI-X)

Since I only have one BCM5704S,[4] I ended up connecting both ports of the BCM5704S (tg3 v1.5) to the 82546EB (e1000 4.4.12-k1) in the other machine.

It's worth mentioning the BCM5704S sits behind an IBM PCI-X to PCI-X bridge:

```
...
+-[80]-+-01.0-[81]--+-04.0  QLA2312
|      |            +-04.1  QLA2312
|      |            +-06.0  BCM5704S
|      |            \-06.1  BCM5704S
|      \-1e.0     PCI Bus Controller
...
```

The bridge plays a bigger role in performance than people expect. For grins, I sent 4k messages out the client through both BCM5704S ports at the same time using default hints. Throughput was 990 ± 0.1 Mb/s for each port (total 1980 Mb/s). vmstat reports ~25% CPU (dual CPU systems) on the server (e1000 driver) and about 33% on the client (tg3 driver). Why was throughput so good? The IBM PCI-X bridge is prefetching data for the chip and also supports split transactions. The prefetching caused some heartburn for the IOMMU code since the IBM bridge ended up prefetching past page boundaries on early prototypes. Changes were made to the ZX1 PCI

[4] HP has no plans for productizing anything with BCM5704 on IA64 at this time. It happens to work and provides a nice comparison to the BCM5701 case study (PCI-X vs. PCI).

PCI Cmd	Consistent	Streaming
ALL	949.35	950.93
MR	952.97	951.79
MRL	952.64	952.66
MRM	953.97	952.58
NONE	951.19	945.10

Table 2: BCM5704 TCP_STREAM 8k Msg

Bus Controller (aka LBA) to stop the prefetching behavior.

6.1 Relaxed Ordering

With 4k messages, running TCP_STREAM gave consistent results around 990 ± 0.1 Mb/s. This wasn't the case for 8k messages. I'm not sure why since the MTU should have been 9k when running the tests. Table 2 is included for your amusement only.

It's irritating I don't know why the results are lower than with 4k messages or what's causing the variability. For the record, UDP_STREAM test was able to send 984.18 ± 0.01 Mb/s using 4k messages and 992.04 ± 0.01 Mb/s for 8k messages.

7 Case Study: 82546EB (PCI-X)

This is Intel's "4th Generation Gigabit MAC design with fully integrated, physical-layer circuitry to provide two standard IEEE 802.3 Ethernet interfaces. . . ."[5] Same setup as with the BCM5701 except an Intel add-on NIC is in both netperf client and server. Both NICs are configured to use 9000 byte MTU.

Msg Size	UDP TX	UDP RX
1024	937.77	492.27
1024	937.76	477.22
4096	983.70	959.55
4096	983.70	959.59
8192	991.80	991.80
8192	991.80	991.80

Table 3: e1000 v4.3.15 UDP (Mb/s)

Msg Size	TCP	UDP TX	UDP RX
1024	976.75	937.76	501.25
4096	978.16	983.72	983.70
8192	907.69	991.80	991.80

Table 4: e1000 v4.4.12 TCP/UDP (Mb/s)

Table 3 shows results for the driver Intel offered on their web site download area: e1000 v4.3.15 driver. But as Table 3 shows, TCP results varied from 639 to 660 Mb/s (1k messages) and got worse (540-565 Mb/s) for 8k messages. UDP results for smaller messages were very poor as well. Something is clearly wrong.

In contrast, TCP Streaming performance for v4.4.12-k1 e1000 driver was quite good. With default hints, both ports combined could send about 1770 Mb/s using 8k message.

7.1 Relaxed Ordering

Disabling ordering enforcement did not change performance in any statistically significant way. In fact, UDP results were identical to Table 4 except for slightly higher UDP RX result. TCP results also showed the same 70 Mb/s drop for 8192 byte messages. And combined port throughput stayed around 1770 Mb/s for 8k message size.

For grins, combined throughput with 4k message size achieved 1936 ± 1 Mb/s. Definitely an impressive result given both ports are shar-

[5] HP has no plans for productizing 82546EB on IA64 at this time. 82546EB only happens to work under Linux because e1000 driver uses I/O Port space. This chip has serious bugs when using MMIO space to access registers.

	Consistent		Streaming	
PCI Cmd	Order	Current	Order	Current
MR	223	NA	221	220
MRL	220	NA	221	214
MRM	220	NA	221	208
NONE[6]	222	NA	219	217

Table 5: 53c1010 Ordering/Current Hints (MB/s)

ing the PCI-X bus.

7.2 Read Current

Clearing Read Current bit for either Consistent or Streaming DMA resulted in a slight drop (890 Mb/s) for TCP Streaming test compared to Table 4. I'm suspicious of this result because afterwards, I could consistently only get 960 Mb/s (8 Mb/s less) for TCP Streaming using 4K messages.

I didn't run UDP tests for Read Current Hint.

8 Case Study: LSI 53c1010 (PCI)

LSI's 53c1010 (Ultra3 LVD) is pretty widely used along with 53c896 (Ultra2 LVD). Both are driven by the sym53c8xx_2 SCSI driver.

Since parallel SCSI busses are not duplex, testing this was fairly straightforward. I setup a MD RAID0 across both channels (alternating disks) with 10 odd-ball Ultra3 disks (9, 18, 36GB, mix of vendors). Then ran:

```
dd if=/dev/zero of=/dev/md4 \
     bs=64k count=200000
```

I learned later that running RAID0 was not such a good idea. More on this in the u320 (53c1030) case study.

Consistent DMA				
Prefetch Depth	0	1	2	3
MR	223	219	219	224
MRL	224	223	221	224
MRM	104	168	220	223
Block Size	512	1024	2048	4096
MR	223	223	222	223
MRL	220	218	218	221
MRM	220	221	219	220

Streaming DMA				
Prefetch Depth	0	1	2	3
MR	218	214	223	219
MRL	221	223	216	219
MRM	216	221	214	220
Block Size	512	1024	2048	4096
MR	216	220	208	219
MRL	215	221	221	222
MRM	218	210	224	223

Table 6: 53c1010 Cache line Prefetching, MB/s

8.1 Relaxed Ordering and Read Current

I've globbed both Relaxed Ordering & Read Current into Table 5 only because they are both boolean values. Differences of less than 3 MB/s are probably not significant.

Disabling Read Current for Streaming DMA clearly reduces performance for MRL and MRM transactions. I thought the "NONE" (217 MB/s) result is a weighted average of all three types of transactions but that is a logical fallacy. This result can't be better than the worst case unless some other interaction is taking place.

8.2 Cache line Prefetching

Of particular interest in Table 6 is the extent Consistent MRM prefetching affects throughput. I guessed this is because the 53c1010

[6] Well, this should really be "ALL" for Relaxed Ordering hint since all the bits are set.

"scripts" are kept in host memory (but cached locally) and all IO grinds to a halt when an uncached portion of script is not available. James Bottomley suggested the entire script fit in onboard RAM and was loaded under Host CPU control at init time. If true, then the scripts themselves are sequentially fetching control data and getting hurt badly by not having the control data available immediately.

9 Case Study: LSI 53c1030 (PCI-X)

Using the same methods (and the same u160 disk drives) as for 53c1010 didn't work. The results varied from 160 MB/s to 185 MB/s regardless of hint settings. I expected at least equivalent performance to the 53c1010 and suspect whatever is causing the variability is also limiting performance.

Trying a different method suggested by James Bottomley led to an interesting result. He was appalled I was using RAID0 because of issues with MD layer not coalescing IO requests again at the disk level. But using a 64k chunk, aka stride, I thought would provide big enough blocks.

To avoid RAID0, James suggested checking if multiple copies of `sg_dd` would (one per disk) would work. Well, I'd like to see multiple IOs outstanding per spindle. And fortunately `sgp_dd` man page suggests exactly that. Nice.

While the advantage of this method is it bypasses lots of kernel code related to buffer cache, the drawback is it also bypasses all the statistics gathering in the kernel. Neither vmstat nor iostat sees any of this disk activity. The solution is to measure the throughput of each disk individually (23 to 48MB/s) and then adjust the number of blocks transfered such that all 10 disks finished their sgp_dd process within about 1 second of each other. Then `date +%s` could time the cumulative I/O. Add up how much data each sgp_dd copied and divide by total time. This worked better than I expected and Table 7 shows how consistent that data was. The accuracy of the data is ± 1 second of 153 second (average, 266666 KB/s) run times. In retrospect, clearly a better method than using RAID0 and suggests roughly the performance RAID0 should be getting.

PCI Cmd	Consistent	Streaming
ALL	268421	266666
MR	266666	266666
MRL	266666	264935
MRM	268421	266666
NONE	264935	266666

Table 7: 53c1030 Relaxed Ordering, KB/s

The bad news is that despite contortions to collect reliable data, neither Read Current nor Relaxed Ordering hints made a statistical difference for the configuration I had. I still wonder if I misunderstand what the hint bits mean in the context of PCI-X. But I couldn't find anything in the chip set documentation to indicate otherwise. I worry the ZX1 chip set might "allow" (logical *And*) the ZX1 DMA Hint and PCI-X command attribute bits vs. "forcing on" (logical *Or*). My expectation was the latter based on documentation.

10 Case Study: qla2312 (PCI-X)

The qla2312 is a Qlogic PCI-X, dual port, 2Gb/s FC chip. Qlogic sells this chip for both dual port and single port FC HBAs. A single port is theoretically capable of 2 Gb/s (about 200MB/s) output and input (full duplex). The dual port HBA is theoretical capable 800 MB/s throughput. I tested the qla2312 in two configurations: with IBM PCI-X bridge and again without (thinking the PCI-X bridge was substantially impacting results).

I used the same methodology as for the 53c1030 Case Study with one of the two ports. Unfortunately, I didn't get a second DS2405 enclosure until much too late. And then I found out the second FC port on the card with the PCI-X Board was disfunctional. I was only able to run a few tests through both ports on a QLA2342 FC HBA (uses qla2312 chip).

The qla2312 HBA was running in PCI-X mode with Firmware version 3.01.18. Same 2.4.20-em19 kernel as before with qla2300 v6.04.00 driver.

10.1 Outbound vs. Inbound IO

To cut to the chase, setting Relaxed Ordering or disabling Read Current hints did not affect performance. With 8 disks, `sgp_dd` was consistently writing 190 ± 1 MB/s. Two things might have contributed to this result: No inbound load was saturating IO path or PCI-X to PCI-X bridge was "hiding" the effect.

Alone, `sgp_dd` inbound (IO reads) workload would get 198 MB/s consistently. Combined with the same outbound (IO writes) workload as above, the inbound rate drops to about 145 MB/s and the outbound workload hovers around 51 MB/s ± 1 MB/s.[7] Again, Relaxed Ordering and Read Current Hints made no difference.

Switching to the other RX2600 (900MHz) which had a qla2312 connected to the same set of disks, I reproduced the 198412 KB/s on the inbound-only workload as well. Bidirectional throughput was about the same: 143 MB/s in and 53 MB/s out (9% CPU utilization).

Having spent several days on this, I started to doubt this HBA was operating in full duplex mode despite all the marketing literature making such a claim. Scrounging through the 300 line `qla2x00_nvram_config()` function suggests full duplex mode is intentionally disabled:

```
  ...
/*
 * Setup driver firmware options.
 */
icb->firmware_options.enable_full_duplex = 0;
icb->firmware_options.enable_target_mode = 0;
  ...
```

Setting `enable_full_duplex` to 1 did not help.

10.2 Dual Port

I tried the same `sgp_dd` workload on both ports. Unfortunately, the 7 disks in the DS2405 I was loaned were ST336605FC (10k RPM) and not ST336753FC (15K RPM). This meant I had to compensate by adjusting the amount of data written to various disks again.

The bottom line is varying Read Current and Relaxed Ordering hints didn't matter for this workload. The outbound `sgp_dd` tasks managed 370 MB/s consistently.[8]

10.3 Summary of Lessons learned

The quote about the journey being more important than the destination comes to mind. Several things learned on this journey:

1. Firmware teams will compensate for stupid OSs. In this case performance

[7] Given the 3:1 bias of inbound:outbound throughput, I tried 6:7 (inbound:outbound) and 5:8 disks—yielded basically the same results.

[8] I can't help but wonder if I've got some piece of the puzzle wrong. But I've reviewed everything several times and if something is wrong, it's not obvious to me. I'll update the paper if I learn otherwise.

gains aren't what I expected because firmware was already setting aggressive cache line prefetching. On fully loaded systems performance could be worse ...but haven't measured that yet. However, Firmware couldn't use *unsafe hints* (e.g. Relaxed Ordering).

2. IO Card Vendors will compensate for stupid chip sets. It didn't initially occur to me high performance IO cards would buffer IO in order to compensate. But the tradeoff is latency.

3. PCI-X is a different bus protocol compared to PCI, not just a speedup. The differences in bus protocol obsoleted the key thing I was hoping to measure (cache line prefetching for DMA Reads).

4. Don't start by testing an adaptive driver. An adaptive driver will adjust its operating parameters after a period of time to optimize for the given workload. I wasted time trying to figure out why my tg3 performance measurements varied in unpredictable ways. Adding "sleep 30" between scripted test runs helped solve that problem.

5. The major weakness of this paper is methodology. I didn't know what I was measuring until I started investigating why I didn't get expected results. I need a PCI/PCI-X logic analyzer which can accurately measure the bus utilization. I believe HP has several such analyzers on site; they just won't fit in the RX2600. I would have to chop open the sheet metal so IO cards could stick out. I'm not willing to do that because airflow would be, uhm, dramatically altered.

10.4 Future Work

Several things come to mind that are still outstanding:

1. **Test fully loaded systems** The busier the memory controller is, the higher the latency memory fetches will be (2x-4x). We don't want to waste memory bandwidth (prefetching too much) or IO bandwidth (prefetch too little). Just enough to compensate for average latency.

2. **PARISC** implementation only supports PCI. Memory controller latencies are slower as is the IO MMU. It should benefit more from DMA Hints than IA64 does. I will update this paper (and remove this "Future work" item) with PARISC results when I have them.

3. **PCI-X DMA Hints** Not as much to do here but still worth exploring. Understand how different chip sets implement PCI-X DMA support.

4. **PCI/PCI-X Logic Analyzer** Perhaps in the future I can get access to an RX5670 with logic analyzer card installed and rerun the tests. Logistically it's non-trivial since RX5670 is not a machine I can walk around with under my arm.

5. **More 2Gb/s FC disks** would be useful. Need to figure out how to stress input and output at the same time. Maybe stripe across both controllers, two RAID0 md devices; one for reading and the other for writing.

10.5 And thanks to...

A fair number of people contributed to this paper. They provided support, ideas, or reviewed content. In no particular order:

Alan C. Meyer, James Bottomley, Erin Handgen, Thomas Bogendörfer, Kevin Carson, Stephane Eranian, David Mosberger, Alex Williamson, Dave Miller, Joe Cowan, Fred Worley, Mike Krause, Matthew Wilcox.

My apologies if I omitted other contributors.

References

[1] http://ftp.parisc-linux.org/docs/astro_intro.ps

[2] http://ftp.parisc-linux.org/docs/elroy_ers.ps

[3] http://iou.parisc-linux.org/ols2002/

[4] http://www.hp.com/products1/itanium/chipset/index.html

[5] http://cvs.parisc-linux.org/*checkout*/linux/Documentation/DMA-mapping.txt?rev=HEAD&content-type=text/plain

[6] http://h21007.www2.hp.com/dspp/files/unprotected/linux/zx1-mio.pdf

[7] http://h21007.www2.hp.com/dspp/files/unprotected/linux/zx1-ioa-mercury_ers.pdf

[8] http://www.pcisig.com/

[9] http://ftp.parisc-linux.org/docs/astro_runway.ps

[10] http://www.hp.com/products1/itanium/performance/architecture/lmbench.html

[11] http://lists.parisc-linux.org/pipermail/parisc-linux/2002-March/015966.html