STiNG Revisited: Performance of Commercial Database Benchmarks on a CC-NUMA Computer System

Russell Clapp, Don DeSota, Carl Love and Adrian Moga
IBM
15450 SW Koll Parkway
Beaverton, Oregon 97006-6063

rclapp@us.ibm.com, desotad@us.ibm.com, carll@us.ibm.com, moga@us.ibm.com

Abstract

In 1996, the STiNG CC-NUMA architecture was introduced. Since that time, it has been implemented with several generations of Pentium Pro® and Xeon® processors, memory and I/O control chipsets, and ASICs that comprise the SCI-based interconnect. In this paper, we describe the performance of commercial OLTP and DSS database benchmarks as captured by hardware event counters embedded in the system. Comparisons are made between this data and the original projections that were made based on simulation models. This comparison shows that, despite some discrepancies between the simulated and executed workloads, the architectural simulations done during development were an accurate predictor of system performance.

1. Introduction

Introduced in 1996, the STiNG system was the first of a series of commercial Cache Coherent Non-Uniform Memory Access (CC-NUMA) architecture offerings of the late 1990's [1]. Other examples include the SGI Origin [2] and the Data General AViiON [3]. These systems were each influenced to some degree by the research done in the late 1980's and early 1990's on scalable shared-memory systems, including the Stanford DASH [4] and FLASH [5] projects, the MIT Alewife [6], the Wisconsin Wind Tunnel [7], and the Sun S3.mp [8]. In the years since the first CC-NUMA commercial systems were introduced, others have arrived or have been announced, including HP's Superdome [9] and IBM's Enterprise X Architecture® [10].

While this research and the commercial systems that followed were a significant departure from the bus-based Symmetric Multiprocessing (SMP) systems of the day, relatively little has been reported about the measured performance characteristics of these CC-NUMA systems, particularly for commercially important workloads. Most of the reported performance data has either been for scientific workloads or was the result of system simulation studies. While benchmark results have been published for some of the commercially available systems [11], detailed analysis like that found in [4] has not accompanied those results.

In this paper, we provide detailed performance data for multiple generations of the STiNG architecture, and compare it to the previously published simulation results found in [1]. This data was collected from the performance event counters embedded in the system's processors and custom controllers. The workloads executed for this study included the Transaction Processing Council's C Benchmark (TPC-C) and part of their D Benchmark (TPC-D) [12]. The TPC-C is the industry standard benchmark for determining On-Line Transaction Processing (OLTP) performance while the TPC-D benchmark is used for determining Decision Support System (DSS) performance.[1]

We were not able to exactly replicate the simulated configurations [1] in the laboratory. The discrepancies between the simulations and the lab experiments included workload differences, changes in cache sizes, and different clock speeds for the various components. However, we were able to closely correlate the measured data to the simulated results in most cases. This led us to conclude that the benchmarks we selected for architectural evaluation were accurate workload representatives, and that our architectural simulation model was an accurate description of the system performance characteristics. The data we present here supports these conclusions.

In the following sections we review the STiNG architecture through several of its implementations, the configurations of the various experiments executed, and the detailed results.

2. Architecture and Implementation

2.1 The STiNG Architecture

Figure 1 shows the high-level block diagram of the STiNG architecture. A system is comprised of some number of *quads*, each being a 4 processor SMP machine. In addition to the processors, each quad also contains Intel chipset components for memory control and bridges to PCI I/O busses, as well as a Lynx block that provides the interface between quads. The processors, some chipset components, and one component in the Lynx are interconnected via a bus in the quad, referred to as the front-side bus or quad bus. Every processor in every quad has a

[1] The TPC-D benchmark has been effectively replaced by the TPC-H and TPC-R benchmarks. We executed query 5 of the TPC-D benchmark in a manner consistent with the intent of the TPC-H benchmark, i.e. the query result was not pre-computed during the database load phase.

common view of system-wide memory and I/O address space. Within each quad, coherence is maintained using a standard snoop-based MESI coherence protocol as defined by the Pentium Pro processor and the front-side bus [13]. This protocol is bridged to a directory-based scheme for maintaining coherence throughout the system, which is based on the Scalable Coherent Interface (SCI) [14]. This functionality is implemented in the Lynx.

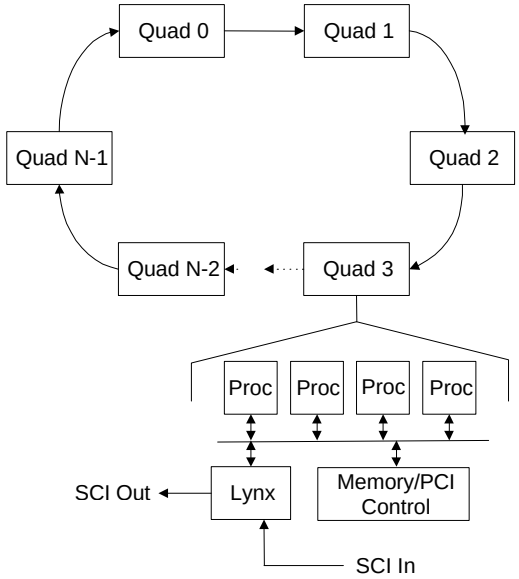

Figure 1: STiNG Block Diagram.

Figure 2 shows the high-level block diagram of the Lynx block.[2] The Lynx contains three logic components, several memory chips for tag arrays, and memory chips to implement a remote cache. The logic components provide the interface to the front-side bus, remote cache control, directory protocol processing, and SCI ring interface. These components make use of the tag arrays to determine the state of remote cache and local memory lines in order to execute the proper protocol actions. Duplicate sets of tags are provided, one each for the network side and the quad bus side. This reduces latency for incoming SCI requests for local memory lines modified on remote nodes. In this case, the directory controller consults the network side tags, determines the location of the modified cache line, and returns the information to the requestor. This avoids the additional latency of forwarding the request to the bus interface chip and then consulting the bus-side tags.

The bus interface controller is responsible for snooping every request on the quad bus and taking ownership of requests for remote memory lines or local memory lines that are cached remotely. Requests which do not require cooperation with remote quads are completed in order. All other processor requests to cacheable memory are completed out-of-order, with the bus interface controller taking the responsibility of sending a request to a remote quad for service. When a response is returned, the bus interface controller initiates one (or more) new bus transaction(s) to complete the original request. The bus interface controller makes requests to local memory and snoops to the processor's caches on behalf of requests originating on remote quads. The bus interface controller also manages the remote cache, supplying data to requests at a latency similar to that of local memory.

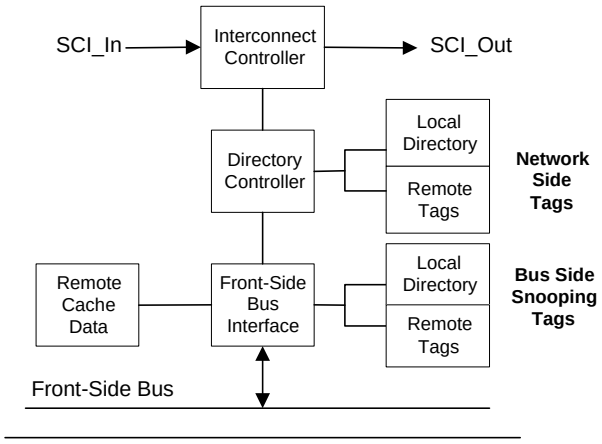

Figure 2: Lynx Block Diagram.

The directory controller is responsible for forwarding requests between the bus interface controller and the interconnect controller. This is not trivial, as the directory controller also manages the translation between the MESI protocol of the quad and the SCI-like protocol used to maintain system-wide coherence. Our implementation used an embedded microinstruction sequencer, so that "firmware" could be loaded to modify the coherence protocol after manufacture. This is discussed more in [1]. As discussed in the original paper, the occupancy of the microinstruction sequencer creates additional latency for OLTP workloads due to queuing. This latency can be mitigated by providing a second sequencer to increase microinstruction throughput.

The interconnect controller is responsible for providing the link and packet-level interface to the SCI ring. It takes SCI requests from the directory controller and sends them on the SCI ring. It receives incoming requests and either forwards them to its sending port or strips them from the ring if they are intended for the directory controller on this quad. It performs link-level functions including delivery acknowledgement and time synchronization.

2.2 NUMA-Q Implementations

Once the STiNG architecture was implemented, it was branded "NUMA-Q" for sale in the commercial marketplace. The NUMA-Q® brand represented several different implementations of the architecture. Each one had different component speeds and cache sizes, as well as some

[2] In the original paper, this was referred to as the *Lynx board.* We have renamed it here as this functionality has been implemented across multiple boards in later system releases.

differences in component features. A high-level summary of differences appears in Table 1.

The key enhancements over the life of the product were the processor updates, cache size increases, improvement in Intel supporting chipset, and improvements in the directory controller.

- Processor core speeds and cache size improvements occurred over time during migration from Pentium Pro® processors to Xeon® processors.
- Size of the remote cache also increased as the OBIC bus interface controller was enhanced to the DOBIC. The DOBIC supported an additional number of incoming and outgoing requests, enabling higher bandwidth through the Lynx. The COBIC enabled the front-side bus to operate at the maximum bus frequency of 100 MHz as compared to the 90% of the maximum frequency with previous generations of the bus interface controller.
- The most significant changes over time occurred to the directory controller, as the SCLIC with one microinstruction sequencer become the DSCLIC with two sequencer cores, one for odd cache line addresses and another for even cache lines. The DSCLIC had hardware assist features that implemented key instruction sequences in logic to further reduce instruction sequencer occupancy and latency. The CSLIC increased the operating frequency to 100 MHz.
- The supporting chipset that supplied memory and I/O bus bridge control was replaced with the second-generation 82450NX [15], significantly reducing memory latency, increasing sustainable bandwidth, and improving the I/O DMA bandwidth for DMA transfers to remote memory.

Table 1: Summary of NUMA-Q Implementations.

Code Name	NUMA-Q I "STiNG"	NUMA-Q II "Scorpion"	NUMA-Q III "Centurion"
Processor Core Speed	180MHz	495MHz	700MHz to 900MHz
Quad Bus/ Lynx Speed	60MHz	90MHz	100MHz
Processor L2 Cache Size	1MB, 4-way	2MB, 4-way	2MB, 4-way
Bus Interface Controller	OBIC	DOBIC	COBIC
Remote Cache Size	32MB, 4-way	128MB to 256MB, 4-way	128MB to 256MB, 4-way
Directory Controller	SCLIC – single core	DSCLIC – dual core w/ h/w assist.	CSCLIC – dual core w/ h/w assist.
Interconnect Controller	DataPump, 500MHz	DataPump, 500MHz	DataPump, 500MHz
Intel Support Chipset	82450GX (8 bus loads)	82450NX (6 bus loads)	82450NX (6 bus loads)

Clearly, the STiNG architecture supports many different combinations of components, many more than were considered in the simulation study published previously [1]. This simulation study was focused on the first implementation, and even then it differed from what was eventually shipped as product. This occurred due to the time lag between the study and completion of development. Although other simulations were done internally to consider other likely speed and cache size combinations for the various components, we have decided to compare our measured results in this paper with those simulation results previously published [1].

2.3 System-Level Considerations

In order to make database system benchmarks processor-bound and achieve good performance, a high-performance I/O subsystem is required. The hardware and software must combine to provide sufficient I/O bandwidth with little processor overhead for servicing request and interrupts. This was achieved for NUMA-Q platforms using fibre channel PCI host adapters attached to a fibre channel switched fabric. The fabric also contained fibre-to-SCSI bridges to attach large numbers of disk drives.

To provide the low processing overhead and bandwidth rates assumed by the simulation environment, two key features are required in the fibre channel-based I/O subsystem. First, *host-based queuing* was implemented to reduce processing overhead. This approach uses system memory to manage work queues for I/O adapters. The adapters consume the requests in the queue directly, thus preventing costly processor references to memory-mapped I/O space. The approach also has the added benefit of reducing completion interrupts, as adapters need only generate these when the work queue is empty.

The second key feature of the I/O subsystem is *multipath I/O*. This feature enables any quad in the system to access any disk in the system via the fibre channel host adapter in its local PCI bus. Combined with database buffer allocation from local memory, multipath I/O enables high locality rates for processor access to database buffers and provides high I/O bandwidth rates to be achieved by preventing DMA transfers across the longer latency SCI-based interconnect.

3. Simulated and Measured Configurations

3.1 OLTP Benchmarks

The standard benchmark for evaluating a system's OLTP performance has evolved from the TPC-B to the TPC-C during the past nine years. Prior to NUMA-Q development, the TPC-B benchmark was a well-understood and established benchmark for OLTP [17]. Data from hardware performance counters and address traces were available to feed simulation models of the STiNG architecture. By the time development of NUMA-Q was completed, however, the TPC-C had replaced the TPC-B as the OLTP benchmark of choice. While some early performance investigation was

done in the lab using the TPC-B benchmark, a full performance evaluation of a NUMA-Q system with a fully functional I/O subsystem was done using the TPC-C workload. Due to the expense of configuring and executing a large-scale TPC benchmark, it was not feasible to validate our earlier simulation results in the lab with the TPC-B workload.

While TPC-C benchmark results were published for several configurations of NUMA-Q systems, we will report in detail on performance results for one configuration that was not published externally. This internal result was for a single instance configuration that provided reasonable scalability to 16 processors, which we will show in Sec. 4. A single- instance configuration uses one DBMS engine that leverages lighter weight single-address space concurrency control and a large number of threads or processes to achieve scaling of performance with increased numbers of processors. We report on the single-instance result as it creates the most stress for the underlying quad-to-quad interconnect, and is most comparable to the single-instance TPC-B data used in the STiNG simulation studies.

3.2 DSS Benchmarks

A standard benchmark for Decision Support Systems appeared during the early development of NUMA-Q systems. While database query benchmarks had been available for some time (such as the Wisconsin benchmark [17]), the TPC-D was the first industry-standard benchmark put forth by the Transaction Processing Council for DSS. The "Power" metric for this benchmark measured a computer systems ability to apply all of its available resources toward execution of a single database query. For multiprocessor systems, this required a database management system capable of subdividing the work for a single query into subtasks and executing them in parallel. Because the TPC-D benchmark was comprised of a suite of queries each run in isolation for the Power test, there were many more computational phases to consider than there were for the TPC-B or TPC-C benchmarks. We chose Query 6 for study while evaluating the STiNG architecture as it had long steady state execution phase where it executed a full table scan, where each row of a database table is read from disk into memory. This query showed higher cache miss rates and a higher I/O bandwidth requirement than other long running TPC-D queries at the time, which made it a good candidate for study.

As time passed during NUMA-Q development, the nature of the TPC-D benchmark behavior changed rapidly. Once the benchmark was introduced, it stimulated a lot of work by DBMS architects and engineers to improve query-optimizers and their execution plans. It also resulted in new options for parallelization and optimization, including pre-computation of results during the database load phase, which did not exist while we were collecting data for Query 6. The end result was that, once a TPC-D benchmark was run on a complete NUMA-Q implementation, the behavior of Query 6 did not at all resemble the behavior of what we originally measured. For that reason, we have selected Query 5 from the set that was measured to compare results against, based on the fact that it did have a long running steady-state phase of execution. We could not compare these results to older results for Query 5, as it was not processor-bound in the early investigation of TPC-D behavior that occurred prior to NUMA-Q development.

Table 2: Workload Profiles for Simulated and Measured Benchmarks.

Event	TPC-B	TPC-C	TPC-D Q6	TPC-D Q5
System	STiNG	NUMA-Q II	STiNG	NUMA-Q I
Processor Count	16	16	32	32
Protocol Sequencers	2	2	1	1
Processor L2 Cache Size	512K	2M	512K	1M
L2 Cache Miss per inst	0.0223	0.0073	0.0018	0.0031
Remote Memory Access Rate per L2 miss	35%	24%	35%	27%
Remote Cache Size (MB)	32	128	32	32
RC Miss Rate per ref	11%	24%	15%	43%
I/O cache line per inst	0.0014	0.0003	0.0019	0.0017
I/O bits per inst	0.36	0.08	0.49	0.44

3.3 Workload Profiles

Table 2 shows the "workload profiles" for the simulation experiments and what was measured in the lab. The columns where System is labeled STiNG refer to the simulation inputs whereas the columns where System is labeled NUMA-Q (I or II) refer to the lab configurations and measured results.[3] The columns labeled TPC-B and TPC-C show what was used or measured for OLTP benchmarks, while the columns labeled TPC-D (Q06 or Q05) refer to the DSS benchmarks. For comparisons between the simulated results and the measured results for each workload type, the processor count and number of protocol sequencers were the same. The processor L2 cache sizes differed, and the effect is shown in the variation of L2 cache misses per instruction for each experiment. We can see from this data that the larger L2 cache reduced misses

[3] Recall that the differences between NUMA-Q I and II are shown in Table 1.

greatly for the OLTP benchmark as expected. The DSS benchmarks have lower miss rates in general, and in the case of our experiments, Q05 had a larger miss rate per instruction than Q06 despite a larger L2 cache.

The next 3 rows in the table show the remote memory access characteristics of the benchmarks. For the simulation studies, there was no hard data on how the memory footprint would be partitioned across quads in the NUMA system. During development, the DYNIX/ptx team was adding NUMA awareness to the OS and developing an API to enable application developers to leverage this capability. As a result, we made what we thought was a conservative assumption of a 35% rate of reference to remote memory for lines that were not modified elsewhere, and a uniform distribution for references to dirty lines. What we observed was a lower rate of remote reference that grew as more quads were added to the system.

In Table 2 we show the measured results for TPC-C at 4 quads and TPC-D Q05 at 8 quads which were both below 35% including references to dirty lines. It is interesting to note that TPC-C has a lower remote reference rate that what we assumed despite its much lower rate of L2 cache misses per instruction, while TPC-D Q05 has a lower rate despite its higher rate of L2 misses.

The remote cache miss rates measured in the lab were higher than what we assumed in our simulation model. For the simulation studies, we assumed that the remote cache would be sized large enough to avoid capacity and conflict misses. This proved largely to be true for OLTP, particularly when a 128MB remote cache was used, as was the case for TPC-C. Remote cache misses are dominated by the communication miss rate. As communication misses can not be prevented by the use of larger L2 caches, the number is significant for TPC-C even though the L2 miss per instruction is much lower than it was for TPC-B. Thus, the remote cache miss rate must be higher as fewer L2 capacity misses that reference remote memory are present for the TPC-C. However, a measure of the remote bandwidth requirement per instruction (as determined by multiplying the L2 misses by the remote memory reference rate by the remote cache miss rate) for both workloads shows that TPC-C has a much lower rate despite its higher remote cache miss rate. The actual remote bandwidth consumed was much closer however, as TPC-C had a lower

CPI than TPC-B. This is discussed further in Sec. 4.4. For TPC-D, the remote cache miss rate varies largely from query to query, as compulsory misses can be large when performing a join operation across large data sets held in memories on remote quads, as is the case for Q05. As a result, Q05 had a much higher remote memory bandwidth requirement than Q06.

While the I/O bandwidth consumption did not directly affect the performance and scaling of a processor-bound system, it is interesting to note how it varies between workloads. OLTP consumes much less I/O bandwidth, as it is typically configured with small disk blocks due to a lack of spatial locality. TPC-C also has a much lower rate of I/O bandwidth consumption than TPC-B, due to the effectiveness of the disk block buffer cache in memory. TPC-B was much less sensitive to memory size, as one very large table typically forced a disk access on nearly every transaction. For TPC-D, I/O bandwidth consumed was much higher, as complex queries typically require access to large amounts of data. In this case, the database is configured to use large disk blocks as spatial locality is present. Still, even for the DSS benchmarks where large amounts of I/O bandwidth is consumed, it is interesting to note that, when measured as bits of I/O per instruction, the amount is less than one-half of the traditional rule of thumb of 1 bit.

3.4 Review of Simulation Methodology

As described in [1], the simulation model used is straightforward. Given the workload profiles, the model determines instruction throughput for each processor and thus the entire system by determining the average processor clocks-per-instruction (CPI). It does this by issuing instructions at the rate of the *internal* CPI, and then generating cache misses for a subset of instructions according to the workload profile. Internal CPI is the time spent in the instruction execution pipeline as well as any stall time related to access of on-processor caches. This value is determined by a separate architectural simulation of the processor. The time the processor spends stalled[4] waiting on average for cache misses to be serviced is the *external* CPI. The STiNG simulation model determines the average cache miss latency and thus the external CPI by modeling all address and data traffic on all control and data paths in the system. The model is instrumented to provide detailed latency and bandwidth information for the workloads modeled. These results are presented in [1] and in this paper.

As stated above, the model generates events to be modeled according to the workload profile. The model is driven by these profile probabilities instead of using address trace-driven or execution-driven techniques. As a result, the model is unable to uncover software bottlenecks due to lock contention as higher levels of throughput are achieved. As stated in [1], more sophisticated modeling approaches are possible that should enable better accuracy and precision, but what we have observed is that our simpler model is indeed an accurate indicator of system performance. This is discussed further in the next section.

4. Experimental Results

4.1 Locality Rates

Figure 3 shows the L2 cache miss service distribution for the OLTP benchmarks. The distribution on the top

[4] Not all cache misses cause the processor to stall due to architectural techniques that allow the execution of multiple instructions in parallel.

represents what was simulated for TPC-B on the STiNG architecture. This distribution was derived from analysis of benchmark data collected on Sequent Symmetry [18] systems combined with expectations for STiNG's CC-NUMA architecture. The distribution is determined in part by the assumption for remote memory access rate and the expectation that all remote cache misses be communication misses. The distribution on the bottom shows what was measured for TPC-C. Because of the lower remote reference rate for TPC-C, many more L2 misses were satisfied with an access to local memory. There were also fewer misses that hit in the remote cache due to this effect. Overall, locality was higher for the TPC-C, with over 90% of the L2 misses being completed within the referencing quad as determined by adding the local memory hit, remote cache hit, and local cache-to-cache transfer categories.

The other three categories show the breakdown for the various types of remote operations. A "2 hop" remote reference refers to either a remote cache miss to a line which can be retrieved from the home quad or local memory reference to a line which is modified remotely. A "4 hop" reference refers to a remote cache miss to a line that is modified on a quad separate from the home quad.[5] "Local Hit/Remote Invalidate" refers to the case where a processor requests ownership for a line where the data is available on the requesting quad but a shared copy resides on a remote quad and must be invalidated. The fact that 2 hop and 4 hop are equivalent at 16 processors indicates that remote cache misses are indeed entirely for communication misses, and that these misses are equally likely to be on any quad in the system. This can be shown by multiplying the remote memory reference rate for modified lines by two-thirds (for a 4 quad system) to determine the rate of 4 hop communication misses. For TPC-B, this is 75% multiplied by 67%, which is about 50%. The fact that TPC-C showed almost the same breakdown suggests that our assumptions about communication misses were correct. We should also mention that the rate of communication misses overall for TPC-C is higher than that for TPC-B because the L2 cache size was larger (2MB vs. 1MB) in the TPC-C testbed.

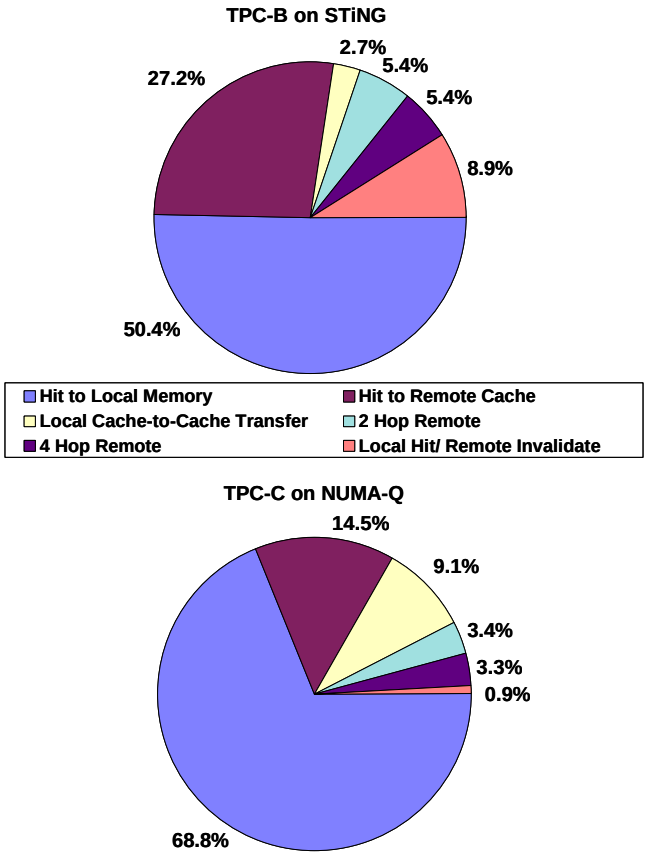

Figure 3: L2 Cache Miss Service Distribution for OLTP Benchmarks.

The interesting departure between the remote traffic rates for TPC-B and TPC-C is the rate of remote invalidations and the relative rates of local and remote communication misses. There were three software effects that enabled greater locality for the TPC-C for communication misses and invalidations. The first was multipath I/O, which enabled communication misses and invalidations related to kernel I/O activity to stay local to a given quad. The second was lowest priority processor interrupt servicing on a quad basis, which kept interrupts local to a given quad, but also created some additional communication misses within a quad. The third was a "buddy locking" scheme, which gave lock acquisition priority to processors on the same quad as the previous holder of a lock. This reduced migrations of lock lines and critical section data for highly contested locks.

Figure 4 shows the L2 cache miss service distribution for the DSS benchmarks. The diagram on the left of the figure represents what we simulated for Query 6 on the STiNG architecture, while the diagram on the right shows what was measured for Query 5 on the NUMA-Q implementation. Again, we see higher rates of requests satisfied by local memory on the measured system as well as higher rates of overall locality. There is also a higher rate of locality of communication misses and invalidations in the measured data for Query 5 as there was for TPC-C. The interesting departure here is the relative ratio of 2 hop and 4 hop remote operations. The assumptions for Query 6 were the same as for TPC-B, with the rate of 4 hop requests for an eight quad system becoming 7/8 * 6/7 = 75%, assuming all remote misses are communication misses whose addresses are uniformly distributed across quads. For Query 5, the majority of remote cache misses are not communication misses, as evidenced by the high rate of 2 hop misses.

[5] We call this "4 Hop" as the home quad responds to the requestor with the identity of the quad where the modified line resides, i.e. there was no request forwarding for NUMA-Q.

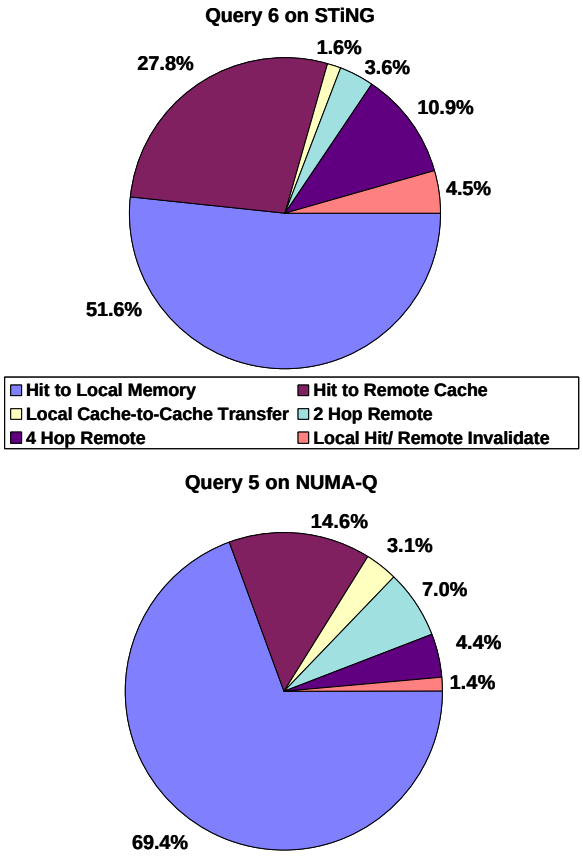

Figure 4: L2 Cache Miss Service Distribution for DSS Benchmarks.

Again, this is caused by a high rate of compulsory misses caused by processing a join operation using memory from all quads in the system.

4.2 Latency

Figure 5 shows the average cache miss service penalty for all simulated and executed benchmarks, normalized to bus clocks to make the different implementations more comparable. The figure shows that, for both workload types, the actual measured latency was lower than what was expected. This follows from the previous figures that showed that quad locality was higher overall than expected. The other thing to note on this diagram is the shape of the curve. The simulation model predicted that the rate of increase in average miss latency would decline as more quads were added to the system. This occurs because the rate of remote operations increases dramatically from zero as the second quad is added, but much less than that as each additional quad is added. This was shown to be true in the measured results. It is interesting to note that this is in sharp contrast to average miss penalty curves for SMP systems, where latency increases slowly at first as the bandwidth consumption increase, and then escalates very rapidly as the

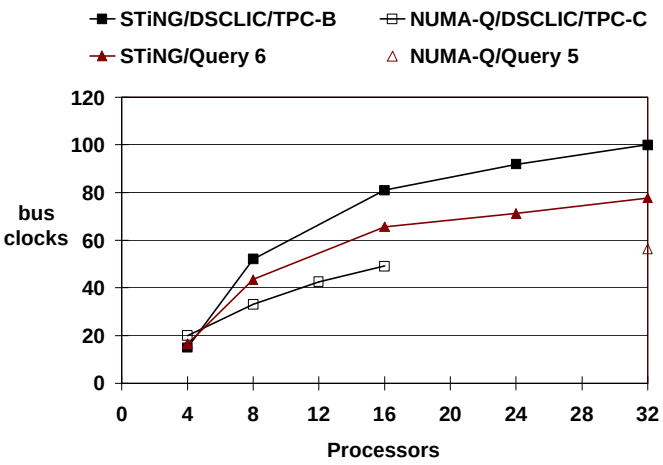

Figure 5: Average L2 Cache Miss Penalty.

memory resource saturates under heavy load. NUMA systems add additional memory subsystems with each quad, so this saturation effect can be avoided, provided there is sufficient bandwidth in the node-to-node interconnect.

Figure 6 shows the average latency for end-to-end operations that require access to at least one remote quad. Again, the curves show the rate of increase in remote latency declining as quads are added to the system. The figure also shows that, for OLTP, the measured remote latency was somewhat higher than what was predicted in simulation using the TPC-B workload, but less than 10%. This can be explained in part by the differences in the hardware configuration. Each system used the same speed SCI ring, but the measured result used a 90 MHz quad bus and Lynx while the simulation used a 66 MHz quad bus and Lynx. This made the relative contribution of the SCI ring to the overall remote latency greater for the measured result. Interestingly, the measured result for the 32-way TPC-D

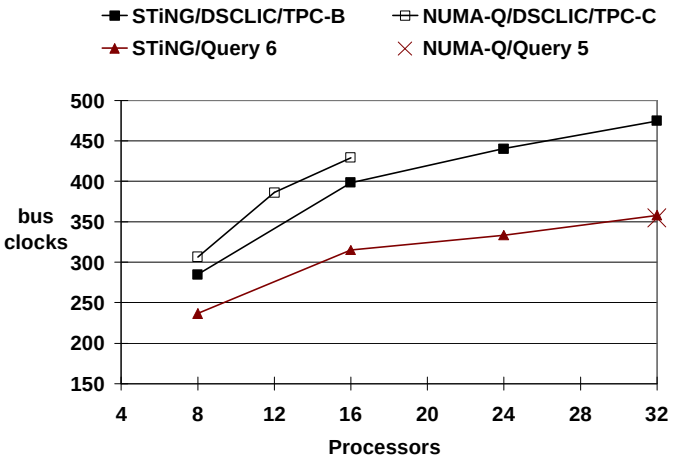

Figure 6: Average L2 Cache Miss Penalty for Requests Involving Access to Remote Quads.

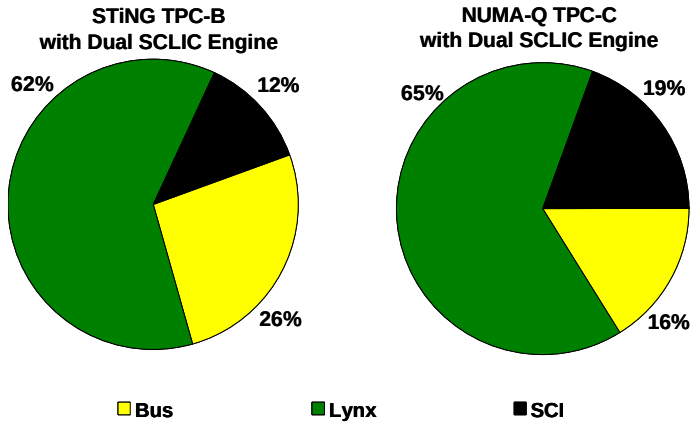

Figure 7: Breakdown of Average Remote Access Latency for OLTP.

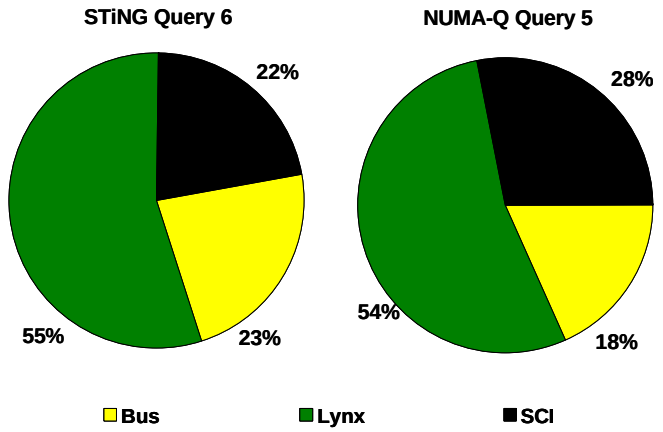

Figure 8: Breakdown of Remote Access Latency for DSS.

Query 5 matched the simulated result for Query 6, despite higher bandwidth consumption for Query 5. This can be explained by the SCI ring being relatively faster in this case as the measured system used a 60 MHz quad bus and Lynx as compared to 66 MHz for the simulation. This difference overcomes the small increase in remote latency for the measured result caused by queuing in the Lynx due to increased occupancy in the SCLIC instruction sequencer caused by higher bandwidth consumption.

Figure 7 and Figure 8 show the breakdown of remote access latency for the OLTP benchmarks into 3 main categories. The first is percentage of time spent on or waiting for the quad bus, the second is the time spent on the Lynx, and the third is the time spent on the SCI ring. For both workload types, the breakdowns are similar between what the simulator predicted and what was measured. There is a greater discrepancy for OLTP, but this is due to the greater speed differential between the SCI ring and the Lynx and quad bus for the measured case versus the simulated case as mentioned earlier. The percentage of time spent on the Lynx is greater for OLTP, given its higher quad-to-quad bandwidth requirement and thus increased queuing delays. These results show that the model of the system was indeed accurate, and with modest bandwidth consumption on a latency-limited system, the variations between workloads were fairly small.

4.3 Bandwidth

Figure 9 shows the data bandwidth utilization of the quad bus. For OLTP, the measured results track the simulated results quite closely, despite the much lower cache miss rate for TPC-C. The rate of bandwidth consumption is higher than expected given the lower miss rate for TPC-C due to a combination of lower relative CPI and a higher core processor speed. This lower relative CPI results from a lower L2 cache miss penalty for TPC-C when compared to the simulation results for TPC-B as shown in Figure 5, and is the result of greater degree of locality as shown in Figure 3. It is also interesting to note that the quad bus bandwidth consumption decreases as quads are added to the system, as predicted. Although an increase in remote traffic causes additional bus transactions to occur, this is more than offset by the increased latency of remote operations, which increases the CPI and reduces the processor's ability to consume bandwidth.

In the case of DSS, the bandwidth consumption is not following this regular pattern. The CPI for DSS is lower than for OLTP, and it increases at a much slower rate as quads are added. Most of the bus bandwidth for DSS is also consumed for I/O transfers, and small percentage of that bandwidth is assumed to require remote transfers in the simulation. This causes offsetting effects for quad bus bandwidth consumption. Unfortunately, the scaling results for Query 5 were not available for comparison. However, we can see that the quad bus bandwidth consumed for 32 processors for Query 5 is much higher than for Query 6. This is due to the much higher miss rate for Query 5

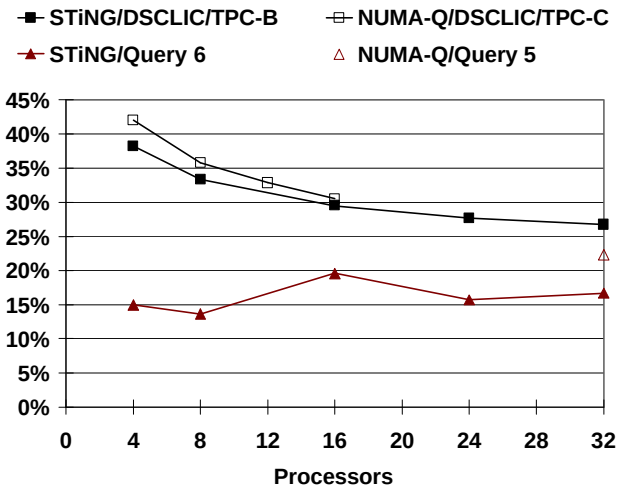

Figure 9: Average Data Bandwidth Utilization on Quad Bus.

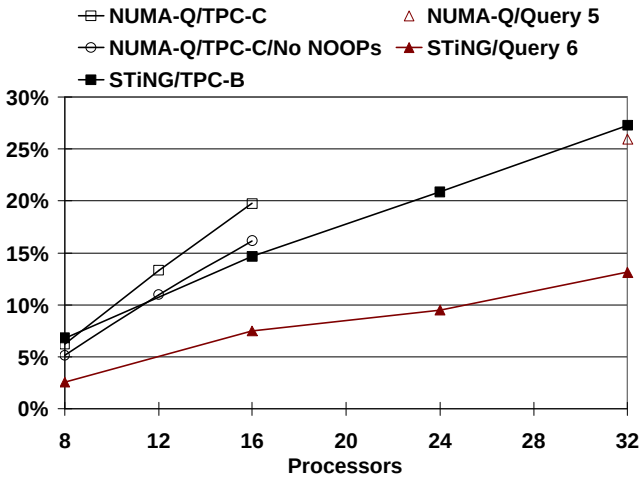

Figure 10: Interquad Bandwidth Consumption.

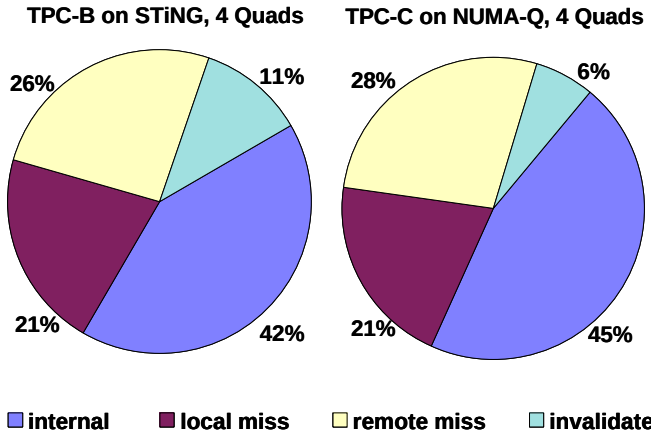

Figure 11: Internal and External Components of CPI for OLTP.

compared to Query 6 that is not offset by a much higher CPI. More details on the CPI for the workloads are provided in Sec. 4.4.

Figure 10 shows the data bandwidth utilization on the SCI ring. There are two measured results shown for OLTP, one labeled "No NOOPs". A NOOP response is essentially an SCI "retry" response that is sent to a requestor when the requested line is in a transition state. These responses were assumed to occur very infrequently, and were not part of the simulation model. However, the number of NOOPs can be noticeable, especially in the case of lock contention, where many processors are trying to access the same cache line simultaneously. Due to the high rate of locking in TPC-C, we measured the SCI ring bandwidth consumption with and without NOOPs, as we were surprised that the measured SCI ring bandwidth consumption was higher than what the simulation predicted, especially given the higher locality rate for TPC-C. The "No NOOPs" measurement showed good correlation with the simulated results, which we would expect given that the quad bus bandwidth consumption was very similar between TPC-C and TPC-B. As the effective bandwidth of the ring did not increase as quads were added to the system, the utilization increased steadily. Still, in this range, the SCI ring was not close to becoming a system bottleneck. The same trend was observed for TPC-D Query 6, but the bandwidth consumption was much lower. Query 5 on the other hand had remote bandwidth consumption closer to TPC-B, again due its higher L2 and remote cache miss rates as described above in Sec. 3.3.

4.4 Processor Clocks per Instruction (CPI)

Figure 11 shows the breakdown of the processor CPI into internal and external components for the OLTP benchmarks. The total CPI is time required on average to execute each instruction as measured in processor core clock cycles. The figure shows the relative contribution to this execution of the time for several categories. As stated above, internal CPI is the time spent in the instruction execution pipeline as well as any stall time related to access of on- processor caches. All of the other categories combine to add up to the external CPI, or that amount of time per instruction that the processor is stalled waiting for data from an external source. The figure shows that the breakdown between internal and external CPI was very similar for TPC-B and TPC-C, and that the relative contribution from external sources was also very similar. Despite the higher core speed for the measured TPC-C result compared to the simulated TPC-B result (495 MHz vs. 133 MHz), the CPI was actually very similar. This shows that the internal component of CPI was nearly the same between the two workloads, and its throughput scaled with processor cycle time. The external component was similar as the lower miss rates for TPC-C were offset by the increased processor cycle time, as described earlier in Sec. 3.3. The breakdown in the various external components was nearly the same due to the high

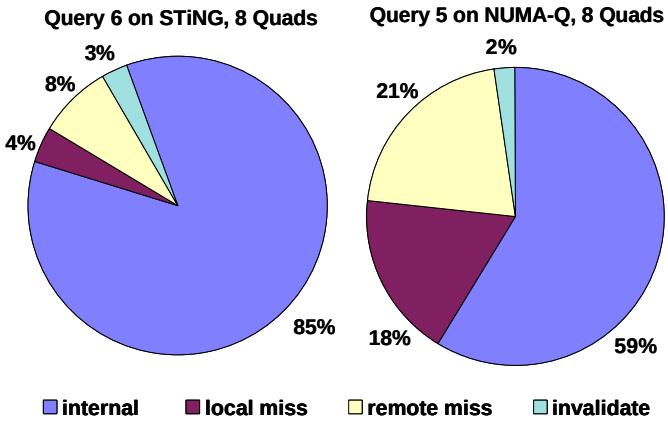

Figure 12: Internal and External Components of CPI for DSS.

rate of locality predicted and measured, as well as the similarity between predicted and measured local and remote access latencies.

Figure 12 shows the relative breakdown of CPI for the DSS benchmarks. Unlike the OLTP results, the DSS workloads had much different CPIs as well as different breakdowns, although the CPIs were much lower than what was observed for OLTP. Query 5 had a much higher CPI (67% higher) due to its higher L2 cache miss rate. The internal component of CPI, however, was almost the same between the Query 5 and Query 6. Again, this shows great similarity between the benchmarks as well as scaling of the internal CPI with processor core speed. The external CPI showed a different relative breakdown between remote and local misses and invalidates for the measured results when compared to simulation. This is caused by differing locality rates as shown in Figure 4.

4.5 System Throughput

Figure 13 shows the predicted scaling efficiency in throughput for TPC-B and TPC-D Query 6, as well the measured scaling for TPC-C. This chart captures the net effect of the miss rates, locality rates, and access latencies on system level performance for the latency-limited processor bound system. The scaling efficiency for TPC-C was very similar although somewhat better than what was predicted for TPC-B. A higher locality rate enabled better scaling initially, but this begins to be offset at 4 quads due to increased spinning on locks. The simulation model did not account for spinning on locks, and thus the scaling efficiency curve maintained its trend for 4 quads and beyond. With a lower CPI and very low external CPI, the TPC-D Query 6 curve shows very good scaling, as it is less sensitive to the average L2 cache miss penalty. Unfortunately, scaling results for Query 5 were unavailable for comparison. Although the scaling efficiency for OLTP is much lower, this has been demonstrated previously as an acceptable result for large-scale systems.

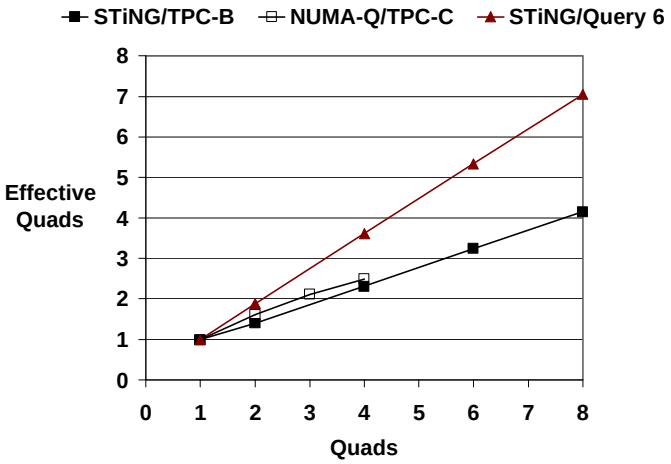

Figure 13: Efficiency of Scaling Throughput with Quads.

5. Summary and Conclusion

This paper analyzed the detailed performance aspects of the NUMA-Q implementation of the STiNG architecture and compared measured results to expectations determined by simulation studies. We have shown that, despite significant workload differences between what was simulated and what was measured, it is still possible to make sense of the data and draw meaningful conclusions.

In particular, we conclude that the simulated results for STiNG were accurate, despite known shortcomings in the model. The model predicted system throughput scaling within 15% of measured results, despite differences in workload characteristics and component speeds. This suggests that more precise simulation approaches based on trace or execution-driven methods may not be worth the significantly higher effort, especially when considering the reduced flexibility and increase in simulation turnaround time.

Furthermore, we have shown both in simulation and with measured data that the system level performance is determined by the rate of locality and the remote and local access latencies, and is not data bandwidth limited. Because of this, the behavior of the measured workloads is very similar, despite differences in miss rates and CPI. For this reason, our approach in using this simulation model to assess design trade-offs is valid, despite the inevitable evolution in benchmarks between the time a system is designed and the time it is deployed in the field.

Another conclusion we can offer is that CC-NUMA is a viable architecture from a systems performance standpoint. With reasonable locality, scaling of performance with processor count is achievable, as we have shown. Although some may argue that the remote access latencies of this system are too high, our conclusion on scaling performance still holds. As more aggressive designs have come to market with more to follow, we will undoubtedly see an increase in scaling efficiency for CC-NUMA systems running commercial workloads.

6. Acknowledgements

The authors would like to acknowledge the contributions of the NUMA-Q design team, without whose support this study would not have been possible. This includes a large number of people involved in both the hardware and software aspects of the system design. However, we would like to specifically thank Tommy Tse, Ruth Forester, and Mary Meredith for their role in executing experiments on the NUMA-Q systems and reporting the results, as well as providing insight into the behavior of the application software. We would also like to specifically thank Bob Safranek, Eric Lais, Rob Joersz, and Bruce Gilbert for their aid in determining the precise function of the hardware event counters.

This work represents the view of the authors and does not necessarily represent the views of IBM.

7. References

[1] T. Lovett and R. Clapp. STiNG: A CC-NUMA Computer System for the Commercial Marketplace. In Proceedings of the 23rd Annual Int'l Symposium on Computer Architecture, pages 308-317, May 1996.

[2] J. Laudon and D. Lenowski. The SGI Origin: A ccNUMA Highly Scalable Server. In Proceedings of the 24th Annual Int'l Symposium on Computer Architecture, pp. 241-251, June 1997.

[3] R. Clark and K. Alnes. SCI Interconnect Chipset and Adapter: Building Large Scale Enterprise Servers with Pentium Pro SHV Nodes. Hot Interconnects IV Symposium Record, pp. 221-245, August 1996.

[4] D. Lenoski, J. Laudon, T. Joe, D. Nakahira, L. Stevens, A. Gupta, and J. Hennessy. The DASH prototype: Logic overhead and performance. IEEE Transactions of Parallel and Distributed Systems, pages 41-61, vol. 4, no. 1, January 1993.

[5] J. Kuskin, D. Ofelt, M. Heinrich, J. Heinlein, R. Simoni, K. Gharachorloo, J. Chapin, D. Nakahira, J. Baxter, M. Horowitz, A. Gupta, M. Rosenblum, and J. Hennessy. The Stanford FLASH multiprocessor. In Proceedings of the 21st International Symposium on Computer Architecture, pages 302-313, April 1994.

[6] A. Agarwal, R. Bianchini, D. Chaiken, K. L. Johnson, D. Kranz, J. Kubiatowicz, B.-H. Lim, K. Mackenzie, and D. Yeung. The MIT Alewife machine: Architecture and performance. In Proceedings of the 22nd International Symposium on Computer Architecture, pages 2-13, June 1995.

[7] S. Reinhardt, J. Larus, and D. Wood. Tempest and typhoon: User-level shared memory. In Proceedings of the 21st International Symposium on Computer Architecture, pages 325-336, April 1994.

[8] A. Nowatzyk, G. Aybay, M. Browne, E. Kelly, M. Parkin, B. Radke, and S. Vishin. The S3.mp scalable shared memory multiprocessor. In Proceedings of the 1995 International Conference on Parallel Processing, August 1995.

[9] Hewlett Packard Company. Meet the hp superdome servers. http://www.hp.com/products1/servers/scalableservers/superdome/infolibrary/technical_wp.pdf

[10] M. Chapman. Introducing IBM Enterprise X-Architecture Technology. ftp://ftp.pc.ibm.com/pub/pccbbs/pc_servers_pdf/exawhitepaper.pdf

[11] Transaction Processing Council. TPC Results Listing. http://www.tpc.org/information/results.asp

[12] Transaction Processing Council. TPC Results Listing. http://www.tpc.org/information/benchmarks.asp

[13] Intel Corporation. Pentium Pro Family Developer's Manual Volume 1: Specifications, pages 7.1-7.2, January 1996.

[14] IEEE Computer Society. IEEE Standard for Scalable Coherent Interface (SCI), IEEE Std 1596-1992, New York, New York, August, 1993.

[15] Intel Corporation. Intel 450NX PCIset Revision 1.3, March 1999. http://www.intel.com/design/chipsets/datashts/24377102.pdf

[16] Intel Corporation. Intel 450KX/GX PCIset, 1996. http://developer.intel.com/design/chipsets/datashts/29052301.pdf

[17] J. Gray, Editor. The Benchmark Handbook for Database and Transaction Processing Systems, Morgan Kaufmann Publishers, San Mateo, CA, 1991

[18] T. Lovett and S. Thakkar. The Symmetry multiprocessor system. In Proceedings of the 1988 International Conference on Parallel Processing, pages 303-310, August 1988.

8. Trademarks