The Architectural Costs of Streaming I/O: A Comparison of Workstations, Clusters, and SMPs

Remzi H. Arpaci-Dusseau, Andrea C. Arpaci-Dusseau, David E. Culler, Joseph M. Hellerstein, and David A. Patterson

Computer Science Division
University of California, Berkeley
{remzi, dusseau, culler, jmh, patterson}@cs.berkeley.edu

Abstract

We investigate resource usage while performing streaming I/O by contrasting three architectures, a single workstation, a cluster, and an SMP, under various I/O benchmarks. We derive analytical and empirically-based models of resource usage during data transfer, examining the I/O bus, memory bus, network, and processor of each system. By investigating each resource in detail, we assess what comprises a well-balanced system for these workloads.

We find that the architectures we study are not well balanced for streaming I/O applications. Across the platforms, the main limitation to attaining peak performance is the CPU, due to lack of data locality. Increasing processor performance (especially with improved block operation performance) will be of great aid for these workloads in the future. For a cluster workstation, the I/O bus is a major system bottleneck, because of the increased load placed on it from network communication. A well-balanced cluster workstation should have copious I/O bus bandwidth, perhaps via multiple I/O busses. The SMP suffers from poor memory-system performance; even when there is true parallelism in the benchmark, contention in the shared-memory system leads to reduced performance. As a result, the clustered workstations provide higher absolute performance for streaming I/O workloads.

Keywords: Clusters, SMPs, Balance, I/O

1 Introduction

```
A balanced computer system needs 1 MB
of main memory capacity and 1 Mbit per
second of I/O bandwidth per MIPS of CPU
performance.
            -Amdahl/Case rule of thumb
```

A well-known tenet of computer architecture suggests that systems should be *balanced* in terms of memory capacity, disk bandwidth, and processing power. This design principle reminds architects not to focus all engineering effort on any small subset of the system, for performance gains in one sub-system may be obviated by the lack of similar gains in another.

The Amdahl/Case rule of thumb provides a guideline for building such balanced systems [2]. However, since this rule originated, performance has increased by many orders of magnitude. Considering the fact that Amdahl made his balanced-system estimations from experience with an IBM 360, a time-shared, single-processor mainframe from 1964, does this rule of thumb still apply to today's vastly altered environment?

Of course, the term "balance" can have different interpretations in different contexts. For example, in the realm of scientific computing, balance is sometimes defined as the number of peak floating-point operations per cycle divided by sustained memory operations per cycle [13]. However, for I/O-based workloads, there is not a clear definition. What constitutes a well-balanced architecture for applications with large demands for streaming I/O?

For I/O-intensive workloads, we define a well-balanced system as one where all resources simultaneously reach near-peak utilization during input and output phases. To assess a particular architecture, we analyze the resource demands of a set of streaming (mostly sequential) I/O workloads, beginning at the disks, and moving up through the memory system and processor. We develop models of resource usage for the given applications, and compare the models with measured usage.

In the course of our study, we keep two questions in mind: first, when data is moving at its peak rate, what demand is placed on the various resources of the system? Second, how efficient are specific architectures at moving data? We approach these two questions via the technique of *disk scaling*. By adding disks to the system and monitoring resource usage, we not only understand how resources are taxed per byte transferred from disk, but also discover when a resource is likely to become a bottleneck.

Given the range of hardware platforms that are prevalent today, we do not wish to restrict ourselves to a particular class of machine. Therefore, we examine the costs of data movement on three diverse architectures. The first platform is the simplest and most common: the desktop workstation. We focus on the Sun Ultra 1 workstation, which forms a basis for comparison with the other two systems.

The second platform is a cluster of Ultra 1 workstations, an instance of larger-scale systems comprised of commodity workstations and high-speed networks such as ATM and Myrinet [3, 5, 10, 20]. By providing a low-latency, high-bandwidth interconnection, these switches have the potential of fusing workstations into a cohesive whole. Implicitly, we evaluate an underlying assumption of clustered systems: that the workstation in its current form is a good building block [3]. This assumption may be optimistic, as a machine that is well-balanced for the stand-alone case may not be properly architected for a tightly-integrated cluster environment.

The third platform is a small-scale symmetric multiprocessor (SMP), specifically, the Ultra Enterprise 5000. Built out of many of the same components as the Ultra 1 workstation, the SMP lends itself to direct comparison with the cluster architecture. The main difference between the two systems is the processor-to-processor interconnect: in the cluster architecture, a local-area network (Myrinet) connects the machines together, whereas in the SMP, the main memory bus (Gigaplane) provides a high-bandwidth, cache-coherent channel for communication between processors.

In order to drive the three architectures, we employ a set of I/O kernels. The first is a simple scan, which reads data from disk sequentially, selects matching records, and writes those records to disk. The second benchmark is an external sort. Sorting is a longtime database-industry standard benchmark, and has recently garnered much interest [1, 4, 15]. In addition to being useful in a classical database environment, sorting is also typical of the workload placed on systems performing decision support [9]. In this paper, we use the single-node and cluster versions of NOW-Sort, currently the world's fastest disk-to-disk sorting program [4]. The third benchmark is an external transpose. Whereas the first two benchmarks fit into the database domain, transpose is often found in external scientific codes. All benchmarks have been hand optimized for each platform.

For the set of benchmarks, we find that none of the systems are well balanced. Specifically, as we introduce more I/O into the system, the processor is likely to become the bottleneck before any other system resource. This is due to the lack of locality in streaming I/O workloads, which spend much time moving data, not operating upon it. Increasing the ability of the processor to move data in and out of the memory system (with increased support for block operations) will greatly aid these types of workloads.

Across all platforms, we find that memory traffic is quite high, often much higher than what is inherent in the applications. Part of this discrepancy is attributed to OS behavior (extra copies to the buffer cache and zeroing heap pages), part to the mismatch between the grain size of some benchmarks and the block size of the cache, and, in the cluster scenario, part to the communication layer. Despite the large amount of memory traffic, the memory interconnect for the stand-alone and clustered workstation is more than capable of handling the bandwidth demands.

For clustered workstations, I/O bus bandwidth is crucial; nearly three times the bus bandwidth is needed, when compared to a stand-alone workstation, due to the aggregation of network and disk traffic. An ideal clustered workstation would provide enough bandwidth for both network and disk I/O, perhaps via multiple I/O busses. Without sufficient I/O bus bandwidth, clusters can only improve aggregate disk performance by increasing the number of workstations in the system.

The SMP is perhaps the most well-balanced of the three architectures, but at the cost of lower absolute performance. This is partly attributed to an extra copy to avoid lock contention and false sharing. The other factor is a memory system that must be filled one bank at a time; thus, with less than the full memory configuration, the memory system performance of the SMP in the study is significantly decreased.

The rest of the paper is outlined as follows. We begin by explaining our methodology in Section 2. The hardware environment is described in Section 3, and the benchmarks are described and modeled in Section 4. Section 5 presents the experimental results, and in Section 6, we conclude.

2 Methodology

2.1 Methodological Approach

This study has two main objectives. The first is to provide a general characterization of the resource usage of the set of benchmarks. We believe the benchmarks are representative of applications that perform large amounts of streaming I/O, and thus the models can show us the resources an architecture must provide to efficiently execute such programs. The models are based on an understanding of the program code and are confirmed empirically on the range of machines.

The second objective is to evaluate the architectures at hand. We study a stand-alone workstation, a small cluster of workstations, and a small-scale SMP. How well do the different architectures execute the given benchmarks? How "balanced" is each of the systems for these types of applications?

We approach these dual objectives with a unified method: *disk scaling*. Each system begins with a single disk per processor. We run the benchmark on this configuration and monitor resources with various hardware and software counters (described below). We add disks to the system, introducing more "potential" I/O, and again run the benchmarks. By measuring resource usage during this process, we empirically determine how each benchmark behaves. We compare experimental results with the application models to separate their *inherent* data movement demands from the *actual* realization upon the given hardware. By measuring application behavior on the systems, we are able to recognize system bottlenecks, and can criticize the particular architectures.

Thus, the data is presented in two forms. The first form presents general relationships of resource usage for each benchmark on the different platforms. For example, on the cluster architecture, we find that the amount of I/O crossing the I/O bus during the read phase of the sort is roughly three times the amount of data read from disk. This is an algorithmic property of the sort, derived via model in Section 4, and confirmed empirically in Section 5.2. Comparing modeled versus actual usage also aids in understanding where the parameters of the architecture affect resource usage. For example, the cache block size does not match the natural grain size in some benchmarks, leading to extra memory traffic.

The second form presents the actual utilization of the resource in question. For example, how utilized was the I/O bus during the read phase of the sort, as disks were added to the system? This data can be used to gauge the effectiveness of the system under test, and to estimate when the resource in question would reach maximum utilization.

2.2 Experimental Apparatus

To measure resource utilization on our three platforms, we use a combination of software and hardware counters. While all UltraSPARCs have on-chip counters that track first- and second-level cache access statistics, the SMP in this study, the Enterprise 5000, also contains counters that track memory and I/O bus traffic. To obtain this information for the single workstation and cluster environments, we configure the Enterprise 5000 to emulate a single-processor system, by shutting down all but one of the processors on board. The resulting machine is quite similar to a single workstation, except for a memory system with an interconnect capable of much higher data transfer rates. For the cluster measurements, we attach a single network card to the SMP and connect it to seven UltraSPARC workstations. It should be noted that this configuration is *not* used to evaluate the memory bus of a workstation; rather, it is employed to gather information on program behavior, such as the amount of I/O traffic generated by the benchmark.

Small modifications were made to the Solaris kernel to enable counters only during non-idle periods, crucial when tracing kernel activity. For all counter-based measurements, we run the benchmarks 10 or more times. Our graphs present the mean results from those runs; the standard deviations are all less than 5%.

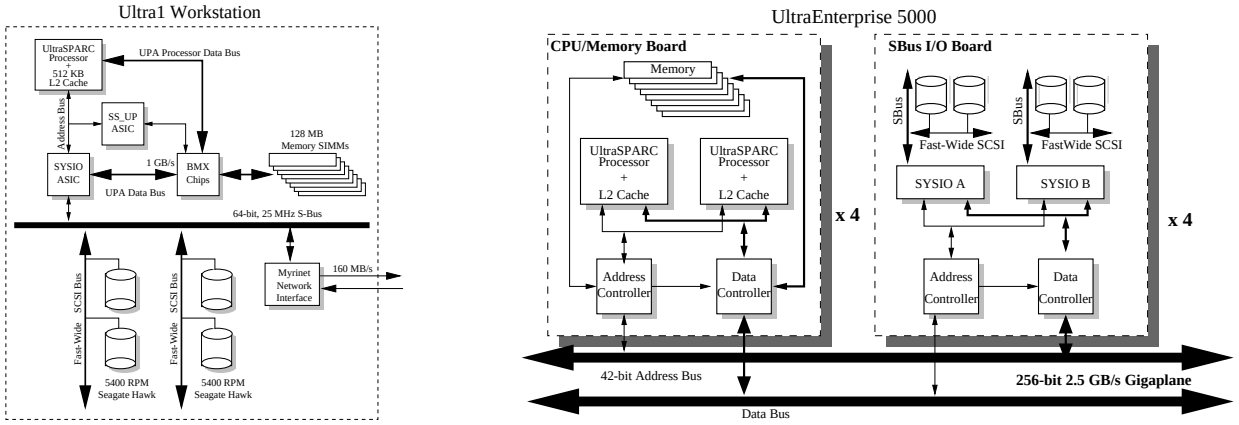

Figure 1: **Hardware Diagram.** *This figure depicts the internals of an Ultra 1 workstation and the Ultra Enterprise 5000.* **Workstation:** *Inside the workstation on the left, the BMX crossbar chips and SS_UP ASIC controller provide switched-based access to memory.* **SMP:** *On the right is the Enterprise 5000. Each CPU board houses two CPUs, which share an address and a data bus. Each I/O board consists of two S-Bus I/O busses. All boards are connected to the main memory interconnect (GigaPlane). Though memory is physically distributed, access time is uniform.*

3 Systems Overview

In this study we compare three platforms which represent important and distinct data interconnection architectures, yet hold most other hardware characteristics in common. All three platforms are based on the UltraSPARC processor and have the same CPU, caches, memory, I/O bus, peripheral busses, and disks. One platform is a single processor, the second is a cluster of eight workstations, and the third is an eight-way SMP.

3.1 Single Workstation

Our simplest platform is a single processor Ultra 1 Model 170 workstation, shown in Figure 1. Each machine contains a single 167 MHz UltraSPARC processor and off-chip second-level cache. The UPA provides a crossbar-like connection from the processor and caches to main memory for UltraSPARC systems, and can sustain bandwidths up to 1 GB/s, though a single processor can not drive it at that rate.

The main I/O bus of the Ultra 1 is the S-Bus[1]. Ethernet and a single fast-narrow SCSI bus connect to the S-Bus from the motherboard, and three S-Bus slots allow for additional devices. In our experimental setup, the Ultra 1 workstation has a single internal Seagate Hawk 2.1 GB, 5400-RPM disk attached to the narrow SCSI bus, used for paging activity. We extend the disk capacity of the system with one or more fast-wide SCSI controllers, each connected to two external disks.

Each of our three platforms runs Solaris 2.5.1, a modern, multi-threaded operating system [11]. Though we are presenting a study of architectural characteristics, operating system behavior often dictates usage patterns of the underlying hardware, as shown in [6, 16, 19]. Therefore, care must be taken to use the appropriate operating system interfaces.

Much of the data movement on the UltraSPARC is performed with special block copy hardware that is available as a part of the VIS instruction set. These block loads and stores move data directly into and out of the double-precision floating point registers, without polluting any of the caches. This hardware feature is accessible through library routines such as `memcpy`. Copy rates are roughly 170 MB/s (moving 340 MB/s of traffic over the memory bus).

[1]While the S-Bus is 64-bits wide, many devices are only available in 32-bit mode (including our network interface cards), reducing the potential bandwidth. In this study, we only examine 32-bit S-Bus products.

3.2 Cluster of Workstations

Our cluster consists of eight Ultra 1 workstations, identical to that described above. The workstations are connected with Myrinet, a 1280 MB/s switched-based network. Each machine has a single Myrinet card on the S-Bus, which is attached via cable to an eight-port switch; multiple switches can be linked together to form large, arbitrary topologies.

Parallel applications in our cluster communicate with Active Messages [21], a high-performance communication layer designed for low latency and high bandwidth switch-based networks. An Active Message is a restricted, lightweight remote procedure call, and is thus an appropriate communication layer for studying the base architectural costs of data movement, because unnecessary copying and buffering of data is avoided. In this paper, we use Active Messages 1.0 (GAM) over Myrinet, which has a round-trip latency of roughly 20 μs and a bi-directional sustained bandwidth of 40 MB/s (20 MB/s sending, 20 MB/s receiving) [7]. The primary unit of data transfer in our benchmark applications is 4 KB.

3.3 Symmetric Multiprocessor

The Ultra Enterprise 5000 is a small-scale symmetric multiprocessor with uniform memory access. The main hardware resource that distinguishes this system from the other two is the GigaPlane, a 256-bit wide packet-switched memory bus that connects up to eight CPU and I/O boards, as shown in Figure 1. Our system contains four 2-CPU processor boards and four I/O boards. A total of 1024 MB of memory is distributed across four of the processor boards.

Each I/O board contains two S-Busses. One S-Bus has a built-in fast Ethernet and fast-wide SCSI bus as well as an extra slot for other devices, whereas the other has two S-Bus slots. In most experiments, we add a fast-wide SCSI controller to each I/O card; each fast-wide SCSI has two disks attached.

All communication on the Enterprise 5000 is performed via loads and stores to shared memory. We use primitives such as mutexes and condition variables to safely access shared data, and barriers to synchronize threads across processors. Copy rates are slightly higher than the single Ultra 1 workstation, at 190 MB/s.

	I/O Bus		Memory Bus	
	Read	Write	Read	Write
Scan				
Workstation	D_r	D_w	$D_r + (\frac{key}{rec})D_r + (\frac{match}{total})(2D_r)$	D_w
Cluster	D_r	D_w	$D_r + (\frac{key}{rec})D_r + (\frac{match}{total})(2D_r)$	D_w
SMP	D_r	D_w	$D_r + (\frac{key}{rec})D_r + (\frac{match}{total})(2D_r)$	D_w
Sort				
Workstation	D_r	D_w	$D_r + (2D_r) + (\frac{key}{rec})D_r + (\frac{bucket}{rec})D_r$	$D_w + (\frac{bucket}{rec})D_w + (2D_w)$
Cluster	$2(\frac{P-1}{P})D_r + D_r$	D_w	$2(\frac{P-1}{P})D_r + (5D_r) + (\frac{key}{rec})D_r + (\frac{bucket}{rec})D_r$	$D_w + (\frac{bucket}{rec})D_w + (2D_w)$
SMP	D_r	D_w	$D_r + (4D_r) + (\frac{key}{rec})D_r + (\frac{bucket}{rec})(3D_r)$	$D_w + (\frac{bucket}{rec})D_w + (2D_w)$
Transpose				
Workstation	D_r	D_w	$D_r + (2D_r)$	D_w
Cluster	$2(\frac{P-1}{P})D_r + D_r$	D_w	$D_r + 2(\frac{P-1}{P})D_r + (2D_r)$	D_w
SMP	D_r	D_w	$D_r + (2D_r)$	D_w

Figure 2: **Benchmark Resource Models.** *This table presents models of resource usage for the three benchmarks across the workstation, cluster, and SMP platforms. All read phase resource usage is relative to the rate data is read from disk, D_r; similarly, for the write phase, all resource usage is relative to D_w. For example, the rate that data is expected to cross the I/O bus during the read phase of the single workstation sort is equal to the rate that data comes from disk, whereas in the read phase of the cluster sort, nearly three times the rate of the disk will cross the I/O bus, as P grows large.*

4 Benchmark Descriptions and Models

In this section, we give an overview of the three I/O kernels used throughout the study, and develop models of their resource usage. These benchmarks primarily perform sequential I/O; in future studies, we plan to examine non-sequential access patterns. The first two benchmarks, scan and sort, are typical of a data-processing environment. The third benchmark, transpose, is more commonly found in scientific codes.

The models presented in this section are of I/O and memory bus usage during the read and write phases of the benchmarks, derived from an understanding of the code. All models of resource usage are presented as ratios to the rate that data is read from or written to disk. For example, if data is read from disk at D_r MB/s, on the single workstation platform, we expect D_r MB/s to cross the I/O bus.

Benchmarks read data from disk via memory-mapped files. We use `mmap` in all benchmarks because the alternative `read` results in an extra copy to the buffer cache by the operating system, which is problematic for applications that stream through data [17]. By using `madvise` with a sequential access pattern, new pages are prefetched and old pages discarded appropriately. For writing, all benchmarks repeatedly call `write` with a large (64 KB) buffer, to avoid the high cost of repeated traps into the kernel. We do not use `mmap` here because it is not a natural match for writing (it can not extend the length of files).

Finally, all benchmarks have the capability to access multiple disks concurrently. We use a simple user-level striping library, similar to that described in [15]. This library spreads disk blocks across the disk sub-system with a user-specified block size (64 KB) and with minimal CPU overhead.

Each benchmark has been hand optimized for the platform in question. Therefore, for each of the three benchmarks, there are three versions of code. For the cluster and SMP, we present models from the perspective of a single processor, as each CPU performs identical tasks. We now describe each benchmark.

4.1 Scan

A sequential scan, modeled after a scan and selection in a database, is the simplest of the three benchmarks. The input set we use in both the scan and the sort is derived from the Datamation [8] and MinuteSort [15] sorting benchmarks: 100-byte records with 10-byte keys. In our terminology, key is the size of the key and rec is the size of a record. The scan selects and writes to disk records that match a user-specified set of criteria, where $(\frac{match}{total})$ is the fraction of matching records. In our benchmark, keys in a certain data range are selected, such that roughly half of the data is written back to disk.

I/O Model: The model of I/O bus usage for the scan on all three platforms is quite simple, since the only traffic crossing the I/O bus is traffic from the disks. Thus, when reading D_r MB/s from disk, D_r MB/s cross the I/O bus. Likewise, during the write phase, when writing D_w MB/s to disk, D_w MB/s cross the I/O bus.

Workstation Memory Model: During the read phase, the scan performs the following three steps. First, records from the memory-mapped file are transferred to memory, accounting for D_r MB/s across the memory bus. Second, the key portion of each record is examined to determine whether the key matches the criteria. Thus, a logical $(\frac{key}{rec})D_r$ crosses the bus, from memory into the processor caches. Third, if the key matches, it is copied into a separate buffer, for another $(\frac{match}{total})(2D_r)$ MB/s across the bus. The write phase writes the buffer to disk, accounting for only D_wMB/s.

Cluster Memory Model: The cluster version is nearly identical to the single-node scan. After P processes have been started across the nodes of the cluster, each node executes a single-workstation scan. This situation is the "best-case" scenario for a cluster, because no explicit data exchange occurs (*i.e.* it is embarrassingly parallel). Because of this complete parallelism, the models for resource usage for the cluster scan match the single node models exactly.

SMP Memory Model: The SMP version forks off P threads, each of which read and select records from an independent portion of the data file, and write them out to disk. Again, there is logically no sharing in this benchmark; thus, the resource usage models are identical to the single workstation.

4.2 Sort

The most complex benchmark used in this study is the external sort, described in detail in [4]. Sorting was chosen by database experts as an excellent test of the memory, I/O, and communication sub-systems of a machine [8]. As described in the scan, we use 10-byte keys within 100-byte records.

The basic sorting algorithm is similar on all three platforms. In the first step, the records must be converted from the layout on disk to a format more suitable for efficient sorting. As records are read from disk, the key and a pointer to the full record are placed into buckets based on the top few bits of the key; this improves the cache behavior of the sort in two ways. First, the sort operates on only <partial key, pointer> pairs, thus copying only 8-bytes rather than 100-byte records as keys are compared and swapped. Second, the number of keys in each bucket matches the size of the second-level cache. The next step sorts the keys in each bucket, using the algorithm described in [1]. Because this step accounts for only a very small fraction of the total execution time and performs no I/O, we do not discuss it further. Finally, the write phase scans the bucket array, gathering sorted records and writing them to disk.

I/O Model: The I/O bus usage for the single workstation and the SMP sort are identical to that in the scan: only disk traffic travels over the I/O bus. However, in the cluster, the I/O bus must also handle network communication. For every record read from local disk, the processor determines the destination workstation responsible for this record in the final sorted order. Assuming the initial data is randomly placed, $(\frac{P-1}{P})$ of the data is sent to remote processors; the equivalent amount is also received from all other processors. Thus, when reading from disk at D_r MB/s, the cluster sends and receives $2(\frac{P-1}{P})D_r$ MB/s; as P grows large, nearly $3D_r$ MB/s cross the I/O bus.

Workstation Memory Model: The memory bus model captures the extra complexity of the sort. During the read phase, the input file is mapped and read into the user's address space, accounting for D_r MB/s across the memory bus. To ensure these pages are not discarded by the operating system memory manager, the records are copied to an input buffer ($2D_r$ MB/s). To set up the sort phase, the keys are placed into buckets based on their top few bits. Therefore, each key is examined $((\frac{key}{rec})D_r)$, and the top four bytes of the key and a pointer to the associated record are written into a bucket array $((\frac{bucket}{rec})D_r)$; $bucket$ is the size of a <partial key, pointer> pair, or 8 bytes. The write phase includes a scan of the bucket array $((\frac{bucket}{rec})D_w)$, a copy into the write buffer $(2D_w)$, and the transfer from memory to disk (D_w).

Cluster Memory Model: The only difference in the cluster version from the basic algorithm occurs in the read phase. Instead of simply placing keys and pointers into a local bucket, the entire record is sent to the workstation that should hold it in the final sorted-order. After each processor has mapped the input file (D_r), the records are copied into one of P send buffers $(2D_r)$; as each buffer fills, it is sent to the appropriate destination processor. This buffering is necessary for efficient communication, since 100-byte messages cannot achieve peak transfer rates. As messages are sent and received, the I/O traffic described above must also cross the memory bus $(2(\frac{P-1}{P})D_r + D_r)$. Upon receipt, records are copied into a record buffer $(2D_r)$, each key is examined $((\frac{key}{rec})D_r)$, and a partial key and pointer are written into the bucket array $((\frac{bucket}{rec})D_r)$, as in the single workstation sort. The processors synchronize to complete the phase, and then each independently sort and write out their set of records, both of which are identical to the single workstation sort.

SMP Memory Model: The model of the SMP version of the sort must also account for communication between processors. Here, a global array of buckets and record buffers are allocated, and each processor begins reading from a separate file in parallel (D_r). As records are accessed, each processor could simply acquire a lock, copy the <key, pointer> pair into the global bucket array, copy the record into the global record array, and release the lock. However, this leads to high lock contention and poor performance. To avoid this problem, each processor keeps a small record and bucket buffer, into which it copies records $(2D_r)$ and <key, pointer> pairs $((\frac{key}{rec} + \frac{bucket}{rec})D_r)$. When a buffer fills, the processor grabs the proper lock, and copies the keys $(2(\frac{bucket}{rec})D_r)$ and records $(2D_r)$ into the global arrays. When the read phase is complete, the processors synchronize, divide the global bucket and buffer array among themselves, and sort the keys in parallel. Finally, during the write phase, each processor gathers its records and writes them to disk, as in the single workstation sort.

4.3 Transpose

Our final benchmark is transpose, similar to an operation found in external scientific codes, such as out-of-core FFT [12]. The basic operation reads in blocks in row-major order, and writes them to disk in column-major order. Blocks are 4 KB for this benchmark, a departure from the 100-byte records of the two previous benchmarks.

I/O Model: The I/O bus model for both phases of the single workstation and the SMP, and the write phase of the cluster, match the other benchmarks. The model for the read phase of the cluster corresponds to the sort.

Workstation Memory Model: The single node transpose reads the input set, and writes out the transpose of the blocks to disk. As it is reading the blocks from disk (D_r), it copies them into column-ordered buffers $(2D_r)$. For example, if the program transposes 64 blocks (an 8x8 array), the first block is read into buffer location 0, the next into location 8, and so forth. The write phase writes out blocks sequentially until the operation is complete, moving D_w MB/s across the bus.

Cluster Memory Model: On the cluster, after mapping the input disk into memory (D_r), each node repeatedly sends a block from its local, memory-mapped input file to a selected destination node $(2(\frac{P-1}{P})D_r)$. Upon receiving a message, the processor copies it into the proper buffer location, based on which processor sent the block $(2D_r)$. Once all the input data has been read into memory and sent to the proper destinations, each processor writes out the data to local disk in a phase with no communication (D_w), identical to the single-node case.

SMP Memory Model: The SMP version begins with each processor sequentially reading data from disk (D_r), and copying them into a buffer $(2D_r)$. The transpose is performed via shared memory, with each processor claiming the required portion of the other processors' data, and writing it to disk, moving D_w MB/s across the memory bus for the write.

4.4 Benchmark Summary

The models of resource usage are summarized in Figure 2. We have seen that for entirely parallelizable workloads such as the scan, the models are identical across platforms. For the cluster, I/O bus usage during the read phases of the sort and transpose is roughly three times that of a stand-alone workstation, due to the addition of network communication. Finally, memory bus usage for both the cluster and the SMP increases when communicating; in the cluster, an extra copy is performed to aggregate small messages into larger ones, whereas in the SMP, an extra copy is necessary to avoid false sharing and lock contention.

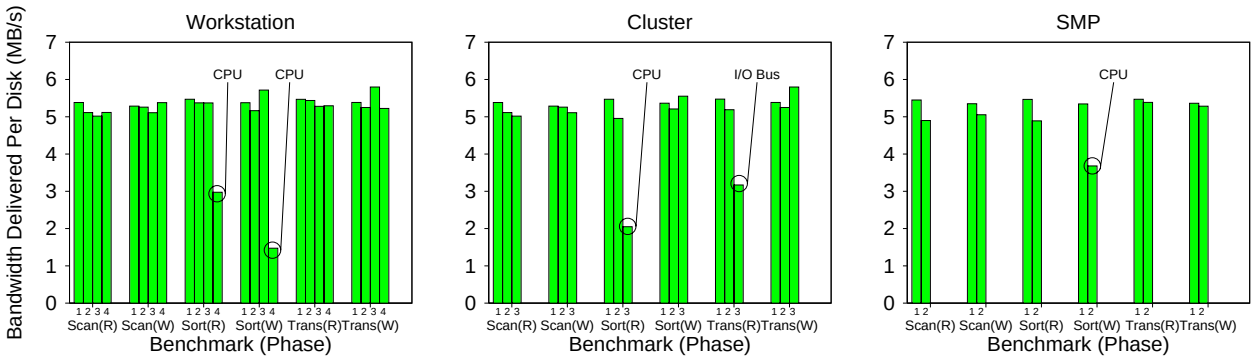

Figure 3: **Absolute Performance.** *These figures plot the incremental achieved disk bandwidth of the benchmarks. Each group of bars scales the number of disks per processor. If the measured rate does not match expectations, we denote the cause of the bottleneck (the CPU or I/O bus).*

5 Experimental Results

This section contains the main experimental results of the paper. We begin by presenting the absolute performance of each benchmark across platforms, shown as the amount of delivered disk bandwidth as disks are added to the system. This establishes two facts: first, the workloads are well-tuned and effectively use multiple disks, and second, the point at which some benchmarks reach a system bottleneck. We proceed by examining each resource on the path from disk to the CPU: the I/O bus, memory bus, processor interconnect, and processor.

5.1 Absolute Performance

We first show the absolute performance attained for each benchmark, across the three platforms, as disks are added to the system. In an ideal system, achieved bandwidth matches the peak disk rate multiplied by the number of disks in the system. In our system, each disk can deliver D_r = 5.5 MB/s when reading from disk, and D_w = 5.4 MB/s when writing to disk.

Figure 3 shows the increase in bandwidth achieved during each phase of the benchmarks across the platforms. For example, in the read phase of the sort for a single workstation, the first bar in the group indicates that data is read from one disk at about 5.5 MB/s. The next two bars show that adding two additional disks yields the expected benefit, with each disk adding roughly 5.4 MB/s. The sum of the three bars (not shown), 16.3 MB/s, is the total data rate achieved with three disks. When the fourth disk is added, the gain in read bandwidth of 3 MB/s is *below* what is expected, indicating we have reached a bottleneck in the system. In this case, the CPU is the bottleneck, as indicated on the graph.

To summarize, we see that at four disks per workstation, the single workstation sort reaches a CPU bottleneck. Cluster performance falls off in the read phases of the sort and transpose, due to a CPU and I/O bottleneck, respectively. Both of these occur with three disks per workstation. Finally, SMP performance degrades in the write phase of the sort with only two disks per processor, again due to the CPU. We now proceed by exploring each resource in detail.

5.2 The I/O Bus

We begin our exploration of system resources with the I/O bus. We find that our measurements of I/O bus traffic on each of the three platforms match the predictions of the models. Measurements of the utilization of the Sun I/O bus reveal that while the single workstation and SMP have sufficient bandwidth, performance in the cluster is sometimes limited.

5.2.1 I/O Bus Bandwidth

Workstation I/O Bus: The leftmost graph in Figure 4 shows the *measured* ratio of data crossing the I/O bus relative to data coming from disk, as disks are scaled from one to four, for each phase of the workloads. As predicted, for all applications on a single workstation, I/O bus usage matches disk traffic. For example, at four disks, when 22 MB/s are moving from disk, the same amount is crossing the I/O bus. Dividing the amount crossing the bus by the amount read from disk gives the ratio of bus bandwidth to disk bandwidth. In the figure, the modeled traffic is shown as a horizontal black line, and we can see that the models match the experimental results precisely.

Cluster I/O Bus: In the cluster, the communication phases, such as the read phases of the sort and transpose, are of particular interest. Both of these benchmarks are expected to move $2(\frac{P-1}{P})D_r + D_r$ MB/s across the I/O bus for every D_r MB/s read from disk. On our 8-node cluster, that comes to $2.75D_r$ MB/s. As shown in the middle graph of Figure 4, the experimental results confirm this behavior. Therefore, in a large cluster, the I/O bus must be able to handle roughly three times the traffic of a stand-alone workstation.

SMP I/O Bus: As in the single workstation, only data from disk crosses the I/O bus; any communication traffic only crosses the memory bus. The data in the figure confirms that D MB/s of bandwidth move across the I/O bus for every D MB/s that are read from or written to disk.

5.2.2 I/O Bus Utilization

We have established demands placed on the I/O bus via models and empirical measurement. Now we apply this knowledge to understand how well the three architectures in this study support streaming-I/O applications. Figure 5 plots the I/O bus utilization for the applications across the different architectures, as the number of disks are increased.

Workstation S-Bus: For a single-processor workstation, the S-Bus contains sufficient bandwidth to support the benchmarks up to a reasonably high disk transfer rate. The peak data bandwidth across the S-Bus operating in 32-bit mode is 80 MB/s; theoretically, this could be attained by moving the maximal 8-word (8-cycle) data burst across the bus after every 2-cycle arbitration phase. However, due to other control information (usually in the form of programmed I/O), peak utilization occurs at around 65%, or around 55-60 MB/s.

Figure 5 shows the utilization of the S-Bus as disks are added to the system. We can see that even at four disks, the bus is only about 25% utilized. Thus, via linear extrapolation,

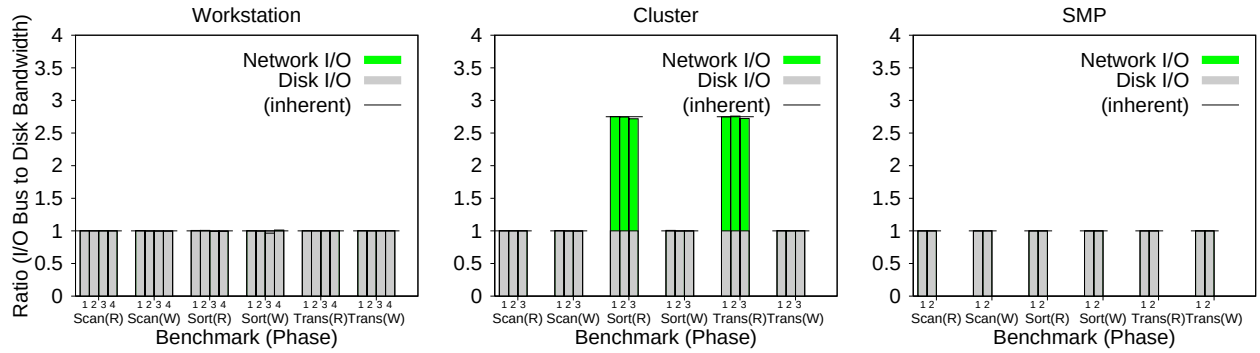

Figure 4: **I/O Bus Data Movement.** *These figures plot the ratio of I/O bus bandwidth to disk bandwidth for the workloads across platforms. The models are shown as horizontal black lines. Each set of bars represents a phase of one of the three benchmarks, and in each group, the number of disks is increased, showing what happens as the load on the system is increased.*

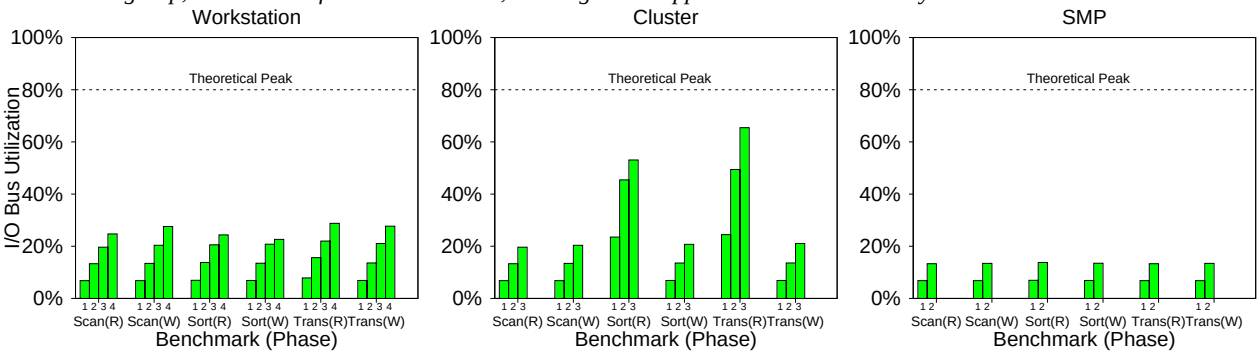

Figure 5: **I/O Bus Utilization.** *These figures plot the I/O bus utilization for the workloads across the three architectures, as disks are added to the system. Only the cluster reaches near peak utilization, in the read phase of the transpose.*

with no other bottlenecks in the system, the S-Bus would be able to support approximately 55 to 60 MB/s of disk bandwidth before hitting its peak utilization.

While the S-Bus in the Ultra 1 workstation contains sufficient bandwidth to support the disk subsystem, the internal SCSI bus does not. Although the workstation can house two internal disks, a fast-narrow SCSI bus with a peak bandwidth of 10 MB/s connects these disks to the motherboard. This configuration limits performance when two modern disks are attached, and should be avoided.

Cluster S-Bus: Because of the extra load placed on the I/O bus during phases of concurrent disk I/O and network communication, the S-Bus suffers high contention and peaks much more quickly than in the single workstation scenario. As seen in Figure 4, the utilization of the S-Bus during the read phase of the sort and the transpose is quite high, between 50% and 65% with three disks per workstation. We show in Section 5.5 that the sort does not reach peak S-Bus utilization because of a more severe CPU bottleneck. However, as indicated in Figure 3, the S-Bus limits the performance of the transpose.

The S-Bus was targeted to meet the bandwidth requirements of much slower devices, not today's high-speed disks and networks used in tandem. In Sun Microsystems' own words, "The S-Bus is optimized for the technologies expected to dominate in the late 1980s and early 1990s" [18]. It is evident that a new bus is needed to support I/O-intensive applications in a cluster. The 64-bit S-Bus partially solves this, but not without widespread availability of 64-bit cards.

SMP S-Bus: As stated above, the SMP S-Bus utilization is identical to the single-node utilization. By the same standards, the S-Bus can meet the I/O demands of the Enterprise system. However, let us examine the overall I/O architecture of the Enterprise. Filling 8 board slots of the Enterprise symmetrically with 4 CPU boards and 4 I/O boards gives the machine a total of 8 S-Busses, with a peak achievable bandwidth of 480 MB/s. However, four of the busses have only a single S-Bus slot (plus a built-in fast-wide SCSI slot); the other four have two S-Bus slots. In comparison, a single Ultra 1 workstation has three slots and an internal SCSI. If we attach fast-wide SCSI cards to all available slots, and drive each SCSI with 15 MB/s of disk bandwidth, each S-Bus will transfer 30 MB/s, or roughly *half* of the potential data rate. The lack of bus slots implies that more modern disk technologies such as UltraSCSI *must* be used to take full advantage of available bandwidth.

5.2.3 I/O Bus Analysis

For stand-alone workstations, there are no immediate problems with today's I/O bus technology. As workstations grow faster in the near future, standards such as PCI should provide the necessary bandwidth.

For clusters, the situation is much more serious, because the I/O bus must handle three times the bandwidth of stand-alone machines. Current S-Bus technology struggles under the aggregate demand of disk and network traffic. One straightforward solution is to provide *separate paths* for disk and network traffic, via multiple I/O busses. A recent machine from Sun, the Ultra30 workstation, is a good example of this: two PCI busses make this machine ideal for cluster computing. More radical solutions suggest placing the network interface on the memory bus [14]. This solves the problem by removing traffic from the I/O bus, while placing no additional load on the

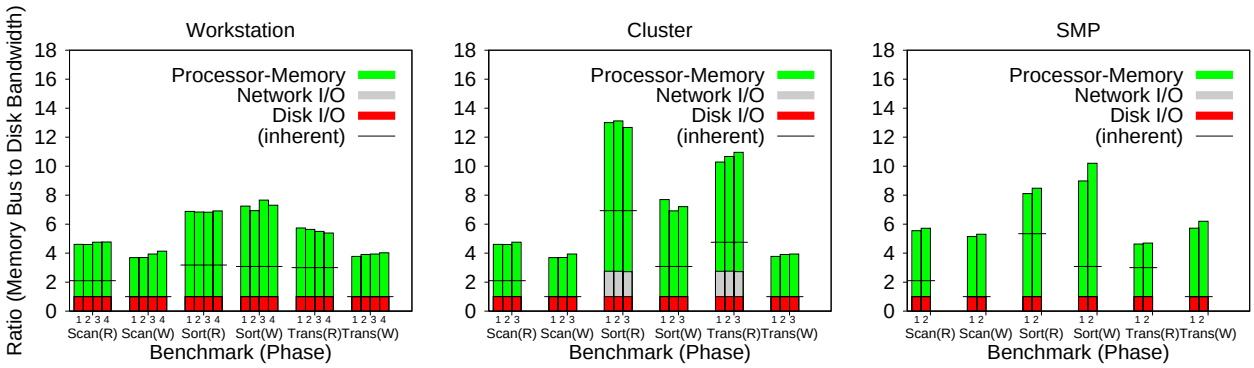

Figure 6: **Memory Bus Data Movement.** *These figures plot the ratio of memory bus bandwidth to disk bandwidth. The modeled amount of data movement, shown as horizontal black lines, under-predicts actual usage by a noticeable amount, due to interactions with the cache block size, operating system zeroing and copying, and, in the case of the cluster, a copy in the communication layer.*

memory bus. However, for the workloads in question, all of which are bandwidth (and not latency) sensitive, this solution may not justify the costs.

Finally, for SMP systems, standard I/O busses should provide plenty of disk bandwidth for the foreseeable future. However, the I/O architecture must provide an adequate number of slots for attaching disks.

5.3 The Memory Bus

Now that we have seen the I/O bus behavior of the benchmarks, we move on to the memory bus. Memory traffic is more difficult to model; we find that our models consistently underpredict memory bus usage. We show that the underlying cache architecture, operating system, and communication layer are responsible for these differences. As for the balance of our platforms, the Ultra 1 workstation memory bus is unlikely to be the bottleneck during streaming I/O; there is plenty of available bandwidth for all of the benchmarks. In the workstation cluster, where extra copies abound due to communication, the memory interconnect still suffices.

5.3.1 Memory Bus Bandwidth

Workstation Memory Bus: Figure 6 shows the measured memory bus traffic as a ratio to disk bandwidth; the inherent traffic is designated as a black horizontal line across each group of bars. There are two main reasons the models underestimate memory traffic: a mismatch between natural grain size and the 64-byte cache blocks, and operating system behavior. When these effects are taken into account, all memory traffic is explained.

Extra traffic occurs in the read phase of the scan due to two distinct interactions with the cache. First, as the scan examines the 10-byte key in every 100-byte record, a 64-byte block must be fetched into the cache. Thus, $(\frac{64}{100})D_r$ MB/s are transferred across the memory bus instead of the predicted $(\frac{key}{rec})D_r$. Second, when copying selected records to the output buffer, each 100-byte record lies on two or three cache blocks. In the worst-case, if every other record matches the scan criteria, what should generate $(\frac{1}{2})(2D_r)$ MB/s, actually generates $(\frac{4}{5})(2D_r)$ MB/s.

Similar granularity mismatches occur in the sort; however, it is more difficult to analytically determine its memory traffic because some repeated accesses to data may be cached. In the worst case, when examining 10-byte keys in 100-byte records, the full 64-byte cache block must be accessed. Also, when scattering 8-byte <partial key, pointer> pairs into random buckets, an entire cache line may need to be read and written. This could generate $(\frac{64}{100})D_r + 2(\frac{64}{100})D_r = 1.92D_r$ instead of $(\frac{10}{100})D_r + (\frac{8}{100})D_r = .18D_r$. Our empirical results show that the worst case is realized. Further work, including simulations of the user code and operating system, is required to understand this behavior in more detail, and is a part of our future work.

Operating system behavior, not represented in the models, generates significant traffic for all three benchmarks. During the read phase of the benchmarks, not only must the OS perform the work required to bring the blocks in from disk, it must also zero pages allocated by the user process. For the sort and transpose, the entire data set is copied into a user-allocated buffer, all of which must be zeroed on demand, whereas half as much zeroing takes place in the scan, because only $(\frac{match}{total})$ of the records are copied. We have found empirically that the total OS traffic on the memory bus during the read phase is $2.5D_r$ MB/s for the sort and transpose, and $1.5D_r$ MB/s for the scan.

Cluster Memory Bus: Figure 6 shows that memory traffic is under-predicted by the cluster models as well. The read phases of the sort and transpose are the only phases that perform communication and thus differ from the single-workstation implementation. Some of this extra traffic is captured in the models, whereas the rest arises from the behavior of the message layer. When the sort and transpose benchmarks send a buffer, the communication layer must copy the buffer to a pre-pinned, pre-mapped region of the address space from which DMA operations can be performed to the network device. As a result, significantly more traffic passes over the memory bus in the cluster than in any of the single-workstation benchmarks; *e.g.*, in the sort, $12D_r$ MB/s cross the bus for every D_r MB/s read from disk.

SMP Memory Bus: The memory traffic in all SMP benchmarks except the read phase of the sort should be identical to that in the single node; however, the measurements in Figure 6 show that at least an extra D MB/s is generated for every D MB/s transferred from disk. We believe this extra traffic is due to contention in the file system for buffers; while we avoid lock contention and false sharing in the application itself, we cannot avoid contention in the OS underneath. [2]

[2]The sort and transpose benchmarks in the SMP zero their input buffers before starting the measurements, which results in less memory traffic during the read phase. Without explicitly performing this operation, I/O performance suffered to the extent we could not fully utilize even two disks per CPU.

5.3.2 Memory Bus Utilization

We now present an evaluation of the Ultra memory systems. Since we can not measure the memory bus utilization on a workstation, we estimate utilization based on specifications.

Workstation UPA Interconnect: The UPA was designed to support small-scale SMP systems; therefore, it is somewhat over-engineered for the simple case of a single processor inside of a workstation. Though the 167-MHz UltraSPARC processor can only drive the memory system at about 340 MB/s (during `memcpy`), the interconnect is theoretically capable of sustaining roughly 1 GB/s. Even in the sort, the most memory-bus intensive applications, only $7D_r$ MB/s moves across the memory bus for every D_r MB/s transferred from disk; thus, to move 80% of peak across the memory interconnect, we would have to read from the disk sub-system at 114 MB/s.

Cluster UPA Interconnect: Again, due to the high capability of the UPA memory interconnect, the amount of memory traffic generated by the sort does not overload the memory system. Even though the sort moves $12D_r$ MB/s across the memory interconnect, this part of the system is not a bottleneck until D_r reaches 67 MB/s, well beyond what other system components can handle.

UltraEnterprise Memory System: The GigaPlane is the main system interconnect for Enterprise systems. The bandwidth of the interconnect is quite high, and is capable of delivering 2.5 GB/s with more memory banks. Because we wish to compare this to the cluster interconnect, we defer further discussion of it until the next section.

Interestingly, the memory bandwidth of the UltraEnterprise scales with the memory *capacity*. The system has eight memory banks, where each bank serves to increase the total memory bandwidth. However, before adding DRAM to the $n+1$st bank, the nth bank must be filled. Therefore, small capacity systems also have small performance capabilities. Indeed, in our early experiments, our server was configured with only two memory banks, limiting our sustainable performance to 1 GB/s on 8 processors (running a copy micro-benchmark similar to that in [13]).

5.3.3 Memory Bus Analysis

We have seen that subtleties in program interaction with the cache architecture, the operating system, and network communication, all lead to excess memory traffic. The mismatches between the grain size of the applications and the block size of the cache architecture are difficult to avoid without requiring application awareness of the underlying machine architecture. Reducing the number of copies performed by the operating system and the communication layer may be more tractable.

Many modern operating systems avoid extra copies by providing direct, non-buffered I/O, which allows applications to bypass the file system buffer cache. The next version of Solaris (2.6) also includes this functionality. Zeroing pages for protection is more difficult to avoid, and will continue to play an important role for streaming I/O workloads.

Cluster communication results in two extra copies in our current environment. Both could potentially be avoided. For example, in the cluster sort, the first copy is explicit in the program, where 100-byte records are copied into larger 4 KB blocks to amortize the overhead of sending a message. This copy could be avoided with tight integration between the network interface and the processor [14], lowering overheads and allowing applications to send small messages at peak rates. The second copy occurs when message layer must copy buffers into portions of the address space setup for DMA transfers. This copy could be avoided by exposing communication buffers to the application. The largest reduction in memory traffic in the cluster could come from direct disk-to-network transfers, completely avoiding the memory bus. However, an application such as the sort must dynamically determine the destination of each 100-byte record, perhaps requiring application-specific code on the disk controller.

5.4 The Communication Interconnect

We now examine the demands placed on the communication backplane for the cluster and SMP. We find that the interconnects are placed under significantly different demands, though both seem unlikely to be the bottleneck in real systems.

5.4.1 Interconnect Bandwidth

Cluster Interconnect: We analyze the communication rates both on the links attaching machines to the network and the total amount of traffic generated across all nodes. As established in Section 5.2, each workstation sends and receives roughly $(\frac{P-1}{P})D_r$ MB/s during the read phase of the sort and the transpose. Thus, the link that connects a machine to a switch has to support $(\frac{P-1}{P})D_r$ MB/s of traffic in each direction. With P processors sending, the aggregate bandwidth moving through the network in the read phase is $P(\frac{P-1}{P})D_r$ MB/s.

SMP Interconnect: The SMP interconnect must support all memory and I/O traffic. Therefore, it is not surprising that this resource is under heavy contention. For example, as seen in Figure 6, in both phases of the scan, each processor places roughly $4D_r$ MB/s on the memory bus for every MB/s read from disk. Therefore, the scan places roughly $4PD_r$ MB/s on the shared interconnect. The sort places up to $9PD_r$ MB/s on the bus, and the transpose roughly the same as the scan. The difference in aggregate bandwidth between the SMP interconnect and the cluster interconnect is roughly a factor of four to nine; however, bandwidth requirements of both mediums scale linearly with processors.

5.4.2 Interconnect Utilization

Myrinet Network: We now analyze the ability of the Myrinet hardware to support this traffic on our 8-node cluster, assuming a single 8-port switch is used. Myrinet links can sustain 160 MB/s in each direction [5], and thus has sufficient bandwidth until D_r reaches $(\frac{8}{7})$160 MB/s. Regarding the ability of Myrinet switches to handle the total communication bandwidth, each switch is a perfect crossbar, and can support 1280 MB/s of aggregate bandwidth when there is no port contention. Thus, the Myrinet switch is not a bottleneck for the total network traffic as long as D_r is under 180 MB/s, well more than the S-Bus can handle.

While the Myrinet links and switches provide ample bandwidth and capacity relative to today's disk speeds, other current networks are not in the proper performance regime. For example, a 100 Mbps (12.5 MB/s) fast Ethernet connection only provides enough link bandwidth to support about 6 MB/s of disk bandwidth per workstation.

GigaPlane: In the Enterprise, the GigaPlane can support roughly 2.5 GB/s of data transfer. In an 8-processor system, we established that up to $9 \cdot 8D_r$ MB/s must be transferred across the bus. Thus, the benchmarks place up to 72 D_r MB/s across the GigaPlane. If we could achieve peak utilization of the bus, the worst case would limit the amount of per-processor disk traffic to 35 MB/s.

5.4.3 Interconnect Analysis

Though the processor-to-processor interconnect is often a main concern for system designers, we can conclude that for streaming I/O applications on small-scale parallel systems, it is unlikely to be the bottleneck. The cluster interconnect needs to provide bandwidth that scales linearly with the number of disks on each workstation; for our streaming I/O benchmarks, the switch must essentially have the capability to handle the sum of all disk traffic. Because larger systems can consist of more switches, the interconnect should be able to scale to medium- and large-scale clusters. The SMP interconnect has to handle the aggregate of all memory and I/O data movement. However, for small-scale systems, modern, aggressive busses such as the GigaPlane seem sufficient.

5.5 The Processor

In this last section, we examine the processing requirements for streaming I/O. We begin by analyzing the number of instructions executed for each byte read from or written to disk. Though we study SPARC-based systems, we believe these relationships can be used as coarse estimates for other RISC-based machines. To understand how well the UltraSPARC processor executes the workloads, we measure processor utilization and CPI. Whereas instructions are implementation independent, utilization and CPI are directly determined by a number of architecture-specific factors, including branch behavior and memory latency. We shall see that the CPI for all platforms and benchmarks is quite high, and, as a result, the processor is often the bottleneck for our workloads. We should note that the characterizations in this section are preliminary, *e.g.* the exact breakdown of which instructions are executed and where time is spent is a part of our on-going research.

5.5.1 Processor Instruction Rates

Workstation Processor: Figure 8 shows the number of instructions (in the millions) executed in system and user mode for each MB/s of delivered disk bandwidth. We see that the number of instructions executed in system mode is almost constant across benchmarks and that writing with `write` requires more system instructions than reading with `mmap`, mainly due to an extra memory copy into the buffer cache.

The main difference across benchmarks arises in the work at user-level that must be performed. Transpose requires the least amount of work; in the read phase, the user code performs only a single copy of each 4 KB record. The read phase of the scan executes more instructions: not only must each key be compared to the selection criteria, but copies must be performed at a fine granularity. The write phases of both the transpose and scan perform essentially no work at the user-level, repeatedly calling `write` to move data to disk.

The sort is the most processor intensive, due to a mismatch between the disk representation of records and the form that is best for internal sorting. On disk, the records comprise a linear array, while the internal sort executes best when the 10-byte keys are separated from the larger records and when groups of keys fit in the second-level cache. Therefore, as records are read from disk, the keys are scattered into buckets, and pointers are set up to the full records. The write phase must then reverse the operation, gathering keys and records into a linear array before writing to disk.

Cluster Processor: The major increases in instruction rates relative to the single workstation occur when sending and receiving messages, in the read phases of the sort and transpose benchmarks. Because communication is performed at user level, the increase is seen in user instruction counts.[3]

		Workstation		Cluster		SMP	
		Sys	User	Sys	User	Sys	User
Scan	Read	1.2	1.8	1.2	1.8	1.7	1.9
	Write	1.6	0.0	1.6	0.0	2.0	0.0
Sort	Read	1.5	2.6	1.6	3.9	1.3	3.3
	Write	1.9	2.2	2.4	2.3	2.4	2.2
Trans	Read	1.5	0.5	0.8	1.3	0.9	0.5
	Write	1.6	0.1	1.6	0.1	1.9	0.1

Figure 8: **Processor Instruction Rate.** *This table shows the MIPS per MB/s of disk bandwidth.*

The transpose indicates the minimal instruction cost to perform bulk-message network communication, because it immediately sends a 4 KB block after it has been read from disk. The increase in the transpose read phase from the workstation to the cluster shows that slightly less than 1 MIPS are required to send and receive data at 1 MB/s: less work than the combined system and user cost of reading from or writing to disk. The cost of communicating is greater in the sort benchmark because the 100-byte records must be copied into 4 KB buffers to amortize the startup cost of communication.

SMP Processor: Similar to the cluster sort, some SMP versions of the benchmarks perform additional work to copy data: buffering is performed to avoid false-sharing and lock contention. For example, in the sort, copying keys into a temporary buffer before inserting them into the global bucket array increases instruction costs by 0.7 MIPS per MB/s. Higher instruction costs also appear in the read phase of the scan and the write phases of all programs. We believe this is due to lock contention within the kernel.

5.5.2 Processor Utilization and CPI

Workstation UltraSPARC-I: Figure 7 shows both the CPU utilization and CPI of each benchmark. From these graphs, we ascertain the sustainable disk bandwidth before each platform becomes CPU limited. For the single workstation, the sort places the most load on the processor, reaching 100% CPU utilization with four disks, or roughly 20 MB/s. This explains the drop in absolute performance seen in Figure 3, much earlier in this section. The other two benchmarks fair somewhat better; extrapolating the CPU utilization predicts that the CPU will not be a bottleneck until roughly 30 MB/s are transferred from disk. For all benchmarks, the CPU is expected to be the primary bottleneck: we predicted the I/O and memory busses could handle at least 55 MB/s and 114 MB/s, respectively, before saturating.

Cluster UltraSPARC-I: Our cluster measurements show that a fast communication layer places a heavy demand upon the UltraSPARC processor. For example, the processor is now 100% utilized in the read phase of the sort with only three disks; with this CPU bottleneck, adding more than two disks per workstation does not improve the performance of sorting. The transpose read fairs somewhat better, and would reach 100% utilization with just over three disks per workstation, or when

[3] The reader may notice that the number of system level instructions in the read phase of the transpose actually decreases in the cluster and SMP implementations. The cluster repeatedly uses small buffers rather than one large buffer, saving the operating system the cost of zeroing many pages; SMP versions zero the pages before the read phase begins. This has the result of reducing SMP instructions relative to the other platforms.

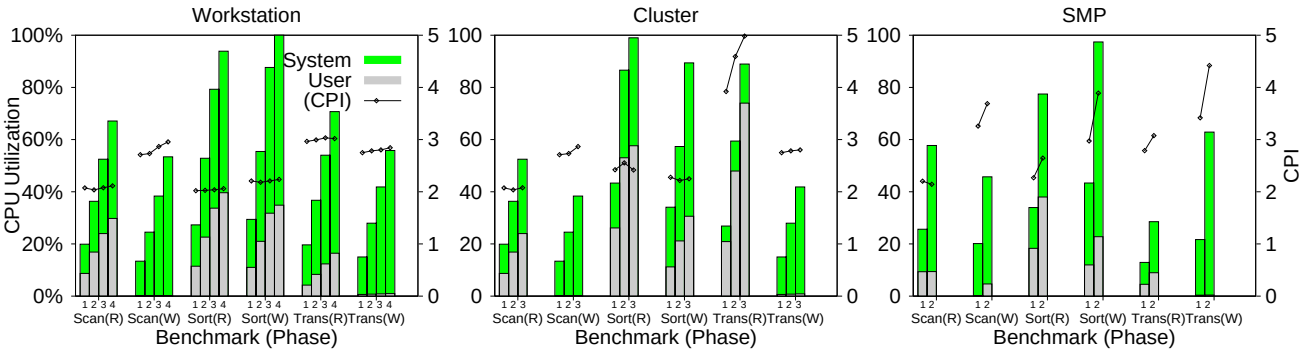

Figure 7: **Processor Utilization.** *These figures plot the processor utilization (with bars, corresponding to the left y-axis label) and CPI (with a line, corresponding to the right y-axis label) for the workloads across platforms. For most workloads, the processor is the main bottleneck of the system.*

moving 18 MB/s from disk. For sort and transpose, both the CPU and S-Bus reach peak utilization almost simultaneously.

The read phase of transpose has a high CPI for two reasons. First, the low data locality of accessing 4 KB blocks of data leads to high cache miss rates; this effect was also measured on the single workstation. Second, accessing the network interface card on the S-Bus is expensive; thus, CPI grows with the number of disks as the contention on the I/O bus increases.

SMP UltraSPARC-I: The CPU utilization shows that for the sort, the SMP can not leverage more than two disks per processor. In general, the CPU utilization on the SMP is much higher than the single workstation, even though the number of instructions executed per MB/s of I/O is similar.

The utilization graphs for all three platforms illustrate the importance of operating system performance for streaming-I/O workloads. For each benchmark on each platform, the time spent in the kernel dominates that spent at user-level, especially as the number of disks increases. Interestingly, the number of instructions executed is not as biased. This, along with the high CPI, indicate poor kernel cache performance.

5.5.3 Processor Analysis

The processor is the main bottleneck for all of the workloads across all of the platforms, except the few situations in the cluster environment where the I/O bus limits performance. There are two approaches to lowering the processor utilization of such workloads. The first is by reducing the number of instructions; communication layers such as Active Messages provide minimal cost primitives for network access; perhaps it is time to revisit the instruction costs of disk I/O. The high instruction cost of disk access may be exacerbated in the Solaris operating system, where generality, modularity, and support for threads, may all increase the common code path to disk. Instruction costs could also be reduced with additional support for large block operations in the instruction set.

The second approach focuses on lowering the CPI. Though the easiest solution may be in hardware via tighter integration with the memory system, a software lesson is perhaps more feasible for current systems: all operations must be "cache conscious". It is crucial to group data accesses into second-level cache-sized objects, thereby avoiding the high cost of repeated access to main memory. This style of programming was used extensively in [15], and we plan to investigate the support applications need to achieve better locality during I/O.

6 Conclusions

We have presented measurements of the resource costs of data movement on three machine architectures. Across all platforms, we developed models of benchmark resource usage, and validated the models empirically. We have also measured the utilization of each resource as the amount of disk bandwidth in the system is scaled, in order to evaluate each architecture from the perspective of the set of I/O kernels.

To summarize our results, we present a graphical representation of what it means to be a *balanced* system. Figure 9 plots the balance of the three platforms for each phase of the workloads in question. For each machine, the resources of the machine are shown on the x-axis. The y-axis plots the predicted per-processor disk bandwidth that could be added to the machine before that particular resource would reach saturation, for the given workloads. This value is based on a linear extrapolation of the usage characteristics found in earlier sections. A set of bars of the same height indicates a *balanced* system, and a set of higher bars indicate a *better* system.

From the figure, we see that none of the machines are well balanced. For all platforms, the first resource to become a bottleneck is the processor. Much of this is due to the lack of locality in streaming I/O benchmarks; in fact, none of the benchmarks had a CPI lower than 2, on a processor that could potentially execute 4 instructions per cycle.

For the single workstation, we also see that the memory interconnect for the stand-alone workstation is over-engineered; it is unlikely to be the bottleneck for these types of workloads.

For the cluster, the I/O bus limits performance during phases of network communication. Not surprisingly, for benchmarks with no communication, the balance of the cluster workstation defaults to the stand-alone case. The memory bus still provides plenty of bandwidth, but for especially memory-intensive benchmarks, it is fairly well utilized. The network backplane, Myrinet, has ample bandwidth, easily handling traffic rates proportional to those of the disk.

Finally, the SMP has the least absolute performance, but, as a result, appears to be the most balanced of the three architectures. The communication backplane, the GigaPlane, handles the high load placed upon it reasonably well. Under a streaming I/O application set, it should be adequate for small-scale parallel systems. However, the performance of the CPU is the weak link in the chain; because of the surprisingly high CPIs, processor utilization peaks rapidly. The end result is that the cluster is a better platform for performing workloads that are dominated by streaming I/O.

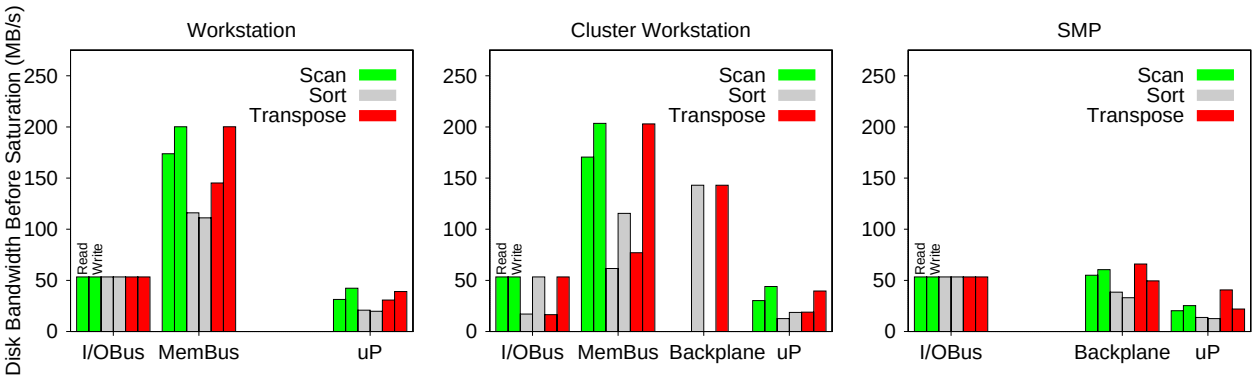

Figure 9: **Balanced Systems.** *These figures reflect the "balance" of each architecture. The x-axis shows the different resources of the system, and the y-axis the amount of (per-processor) disk bandwidth one can introduce into the system before the resource reaches peak utilization. A flat set of bars indicates a "balanced" system, and a higher set of bars indicates a "better" system.*

7 Acknowledgements

The authors would like to thank Jim Gray, Eric Anderson, Kim Keeton, Rich Martin, and Amin Vahdat for for valuable comments, feedback, and suggestions. We'd also like to thank Ashok Singhal of Sun for help with performance counters, Katherine Hartsell of Sun for information on Sun Enterprise Servers, and Sharad Mehrotra of Sun for help with a generous equipment donation. Special thanks go to Satoshi Asami for help with the TD disks. Finally, we thank the shephard, Jean-Loup Baer, for his direction and help in setting the work in context. This work was sponsored by DARPA contract F30602-95-C-0014 and the California State MICRO Program. Remzi Arpaci-Dusseau is currently funded by an Intel Graduate Fellowship.

References

[1] R. C. Agarwal. A Super Scalar Sort Algorithm for RISC Processors. In *1996 ACM SIGMOD Conference*, pages 240–246, June 1996.

[2] G. Amdahl. Storage and I/O Parameters and System Potential. In *IEEE Computer Group Conference*, pages 371–72, June 1970.

[3] T. E. Anderson, D. E. Culler, and D. A. Patterson. A Case for NOW (Networks of Workstations). *IEEE Micro*, February 1994.

[4] A. C. Arpaci-Dusseau, R. H. Arpaci-Dusseau, D. E. Culler, J. M. Hellerstein, and D. A. Patterson. High-Performance Sorting on Networks of Workstations. In *SIGMOD '97*, May 1997.

[5] N. Boden, D. Cohen, R. E. Felderman, A. Kulawik, and C. Seitz. Myrinet: A Gigabit-per-second Local Area Network. *IEEE Micro*, February 1995.

[6] J. B. Chen and B. Bershad. The Impact of Operating System Structure on Memory System Performance. In *Proceedings of the 14th Annual Symposium on Operating Systems*, pages 120–133, December 1993.

[7] D. E. Culler, L. T. Liu, R. P. Martin, and C. O. Yoshikawa. LogP Performance Assessment of Fast Network Interfaces. *IEEE Micro*, 2/1996.

[8] A. et. al. A Measure of Transaction Processing Power. *Datamation*, 31(7):112–118, 1985. Also in *Readings in Database Systems*, M.H. Stonebraker ed., Morgan Kaufmann, San Mateo, 1989.

[9] J. Gray. Personal Communication, June 1997.

[10] M. D. Hill, J. R. Larus, S. Reinhardt, and D. A. Wood. Cooperative-Shared Memory: Software and Hardware for Scalable Multiprocessors. *ACM Transactions on Computer Systems*, 11(4):300–318, 1993.

[11] S. Kleiman, J. Voll, J. Eykholt, A. Shivalingiah, D. Williams, M. Smith, S. Barton, and G. Skinner. Symmetric Multiprocessing in Solaris 2.0. In *Proceedings of COMPCON Spring '92*, 1992.

[12] C. L. Kuszmaul. Out-of-core FFTs in a Parallel Application Environment. Technical report, NAS Technical Report, RND-93-013, 1993.

[13] J. D. McCalpin. Sustainable Memory Bandwidth in Current High-Performance Computers. White Paper, 1995.

[14] S. S. Mukherjee and M. D. Hill. A Case for Making Network Interfaces Less Peripheral. In *Hot Interconnects V*, Aug. 1997.

[15] C. Nyberg, T. Barclay, Z. Cvetanovic, J. Gray, and D. Lomet. AlphaSort: A RISC Machine Sort. In *1994 ACM SIGMOD Conference*, May 1994.

[16] S. E. Perl and R. L. Sites. Studies of Windows NT Performance Using Dynamic Execution Traces. In *OSDI 2*, pages 169–184, October 1996.

[17] M. Stonebraker. The Case for Shared Nothing. *Database Engineering*, 9(1), 1986.

[18] SunMicrosystems. SBus Specification, Rev. B. White Paper, 1990.

[19] J. Torrellas, A. Gupta, and J. Hennessy. Characterizing the Caching and Synchronization Performance of a Multiprocessor Operating System. In *ASPLOS-V*, pages 162–74, October 1992.

[20] T. von Eicken, A. Basu, V. Buch, and W. Vogels. U-Net: A User-Level Network Interface for Parallel and Distributed Computing. In *Proceedings of the 14th ACM Symposium on Operating Systems Principles*, pages 40–53, December 1995.

[21] T. von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauser. Active Messages: a Mechanism for Integrated Communication and Computation. In *Proceedings of the 19th Annual Symposium on Computer Architecture*, Gold Coast, Australia, May 1992.