Performance Enhancements for HPVM in Multi-Network and Heterogeneous Hardware

Greg Bruno[1,2], Andrew A. Chien[1], Mason J. Katz[1], and Philip M. Papadopoulos[1]

[1] Department of Computer Science and Engineering; University of California; San Diego; USA

[2] INCEP Technologies, Inc.; San Diego; California; USA

Abstract. The Concurrent Systems Architecture Group (CSAG) has been investigating high-performance clustering of commodity PCs with gigabit interconnects and a commodity OS (Windows NT). This paper describes several of the major enhancements in the latest release of HPVM 1.9 (High Performance Virtual Machine). The first two of these are concurrent support of multiple gigabit fabrics and shared memory transport, and a self-tuning I/O scheme that manages performance disparities of various PCI implementations. In addition to these innovations, HPVM 1.9 also delivers both lower-latency (8.8 μs) and high-bandwidth (100+ MB/s) messaging. Another major contribution of HPVM 1.9 is the design and implementation a new unified polling strategy that allows simultaneous use of several networks (either VIA or Myrinet) and a shared memory transport without resorting to complex adaptive polling schemes. The implementation of our new high-performance shared memory transport is described in detail. Finally, several system and application benchmark results are given and interpreted to assess the value of these changes.

1 Introduction

High-performance PC clusters are rapidly becoming mainstream because of their favorable cost-to-performance ratios. Indeed, clusters of commodity PCs with gigabit networking (especially Myrinet, VIA, and to a lesser extent gigabit Ethernet) perform on par with significantly more expensive supercomputers. One focus of the CSAG group has been to explore methods for extracting high performance (100+ MB/s, 8.8 μs latency) out of the available hardware, and then translating that capability to applications through efficient layering of software. The hardware landscape has changed significantly during the past three years of this project. Low-latency gigabit hardware has achieved standardization through the Virtual Interface Architecture (VIA); SMPs (at least two-way) are increasingly attractive as nodes; and commodity motherboards with greater integration have significantly increased I/O performance variances across generations. Because of these rapid changes, greater flexibility and tunability was needed in the Fast Messages (FM) core messaging substrate of HPVM. This paper describes four enhancements to the latest version of HPVM, which improve its overall performance and increase performance stability over generations of motherboards.

Older versions of HPVM used the messaging hardware even for processes communicating on the same physical machine. Similar to older message passing machines like the Intel Paragon, this meant that node-to-node messaging performance was faster than on-node messaging performance. Furthermore, as the processor count per SMP increases, this design becomes increasingly disadvantageous. Instead of using the network hardware for every message, shared memory should be used for intra-machine transfers. While this optimization had been explored by other researchers [16], they employed complex adaptive polling schemes to manage the large cost differential between polling for intra- and inter-machine messages. The method presented here reduces the polling cost for inter-machine messages, eliminating the need for adaptive schemes without sacrificing performance on either transport. The resulting shared memory transport for FM is very efficient (delivering 3 μs latency, 200+ MB/s, see Sect. 5), with a poll cost of approximately 150 ns. A memory-based notification scheme for Myrinet network reduces polling cost to 100 ns, reducing FM point-to-point latency by nearly 15 percent. Parity in polling costs means that multiple interfaces can be polled in succession, eliminating the need for adaptive schemes and allowing additional transports to be easily incorporated. Section 4 describes the memory-based notification scheme and support for multiple network transports. Section 5 describes the design of FM's shared memory transport in detail.

Over the past few years, motherboard and specifically PCI chipset implementations have proliferated. In some cases, recent chipsets can deliver poorer programmed I/O performance than their predecessors. Because FM relies on efficient write combining and programmed I/O, these differences were directly reflected (negatively) in FM performance. To address this problem robustly, HPVM 1.9 includes self-tuning I/O code that empirically selects the optimum data burst size to obtain peak programmed I/O throughput. Section 6 describes this strategy and its associated performance.

In this paper, we summarize the relevant HPVM background and design in Sect. 2 and Sect. 3. We then describe the memory notification and shared memory implementation in Sect. 4 and Sect. 5. The observed performance differences across chipset generations and our adaptive I/O solution is described in Sect. 6, and Sect. 7 closes with a summary of our perspective on these results and conclusions.

2 Background

The plummeting price and increasing performance of both desktop computing systems and high-speed networks have made the use of cluster computing systems increasingly desirable. The Beowulf cluster approach [3], which relies on the pervasive Fast Ethernet technology (and essentially zero incremental cost) produces clusters that support coarse-grained, loosely coupled parallelism. The faster system-area network (SAN) technologies [1,2,4,10,13] provide hardware enablers for tightly coupled clusters. Significant advances from the research community in the past five years have made it possible to deliver hardware communication performance to the applications Fast Messages (FM) [17], Active Messages (AM) [5] U-Net [22], VMMC-2 [8], PM [20], BIP [18], MINI [12], and Osiris [7]. These efforts have forged a consensus on core requirements for network interfaces that was eventually crystallized in the Intel/Compaq/Microsoft standard for cluster interfaces—the Virtual Interface Architecture [21]. We briefly survey highlights from our efforts in this area during this period.

The HPVM research project has focused on crucial aspects of the design of high-performance messaging layers such as the trade-offs between protocol processing overhead and the functionalities exposed in the programming interface. The key design decisions taken for the original Fast Messages (FM) [17] for Myrinet implementation provided key communication services while retaining microsecond-level overhead. In April of 1995, Fast Messages 1.1 achieved short message latencies of 14 μs and a peak bandwidth of 17.6 MB/s Myricom's Myrinet. This performance represented 75 percent of the available 23 MB/s of I/O bandwidth on the Sparcstation 20 platform, as well as a performance improvement in both dimensions of approximately 5–10 times. The FM 1.1 implementation achieved $N_{\frac{1}{2}}$ of 54 bytes with a bandwidth of 17.5 MB/s available for messages as small as 128 bytes. FM 1.1 implemented the Berkeley Active Messages API [5].

Designed and implemented in autumn of 1996, Fast Messages 2.0 and 2.1 improved the API based on lessons learned from a Sockets-FM and MPI-FM implementation [14]. The new API provided streamed gather-scatter at both sender and receiver; per-message threaded handlers; and per-message flow control, which enabled the full performance of FM to be delivered to a wealth of application API's (MPI [14,15], Shmem Put/Get [11], Global Arrays, BSP). The performance achieved by FM 2.1 on 300 MHz Pentium II's and Myricom's Myrinet is 9 μs minimum latency, 92 MB/s peak bandwidth, with $N_{\frac{1}{2}} \approx 256$ bytes. These values represent high absolute performance, comparing to MPP interconnect performance and internal memory bandwidth.

Fast Messages 2.1 (as part of the HPVM 1.0 release) was used to build the 256-processor NT Supercluster, the world's largest Windows NT cluster. This system was demonstrated in April 1998, at the NCSA Alliance kickoff meeting, and has been in production use since autumn of 1998. The cluster has been upgraded several times, and current aggregate performance is approximately 200 gigaflops, 128 gigabytes of memory, and 2 terabytes of disk. Network bisection bandwidth delivered by Fast Messages running on Myrinet is $64 * 160$ MB/s $= 10$ GB/s.

The release of HPVM 1.2 in February 1999 added several important new functionalities to the system. First, we incorporated multiprocess and multiprocessor threading support, which allows FM to be used effectively in multiprocessor PC boxes. Separate processes using FM are safely isolated from each other. Second, peak bandwidth was increased from 92 MB/s to more than 100 MB/s. Third, improved dynamic configuration facilities (for messaging buffer pools) and an Installshield packaging made the entire system dramatically easier to install, manage, and use. Fast Messages and HPVM have also been used as a testbed for research on novel mechanisms for co-scheduling on a network of workstations [19], and for research on the introduction of QoS guarantees in a wormhole-routing interconnect [6]. HPVM is available for download on our Web site at <http://www-csag.ucsd.edu/>. The remainder of this paper summarizes the improvements to the HPVM system in HPVM 1.9.

3 HPVM Low-Level Design Review

Overall, the structure of FM is straightforward. Packets must traverse a pipeline across multiple I/O busses. Data integrity is ensured by guaranteeing that a receiving host has free memory for every incoming packet and by taking advantage of the hardware reliability of the network. Active-message–like semantics require that application programs are able to handle incoming messages while sending an outgoing message.

Reviewing the low-level architecture of the Myrinet LANai network interface and the basic structure of FM, the Myrinet LANai NIC houses a programmable 66 Mhz (Version 4.x) custom processor, and the code that runs on this processor is often called the LCP (LANai Control Program) or MCP (Myrinet Control Program). FM uses a custom LCP that only processes FM-formatted packets. The Myricom-supplied GM code also supports a TCP encapsulation. Working in conjunction with the LCP, a host library runs on the machine's CPU (x86 in this paper) to complete the software layout. Because the LANai is physically located on the PCI bus, memory transfers can occur in two ways: data can be DMAed *to/from* the LANai memory, or data can be PIOed (programmed I/O) *to* the NIC. The FM library always uses PIO for sending data to the LANai. DMA is always used to send data from the LANai to the host. One further feature/structure of the Myricom LANai is that packets must be DMAed to/from the physical network from NIC-local memory. This implies that a message packet must traverse a pipeline as shown in Fig. 1.

With proper software design, this entire process can be pipelined for high performance. All high-performance messaging implementations on Myrinet hardware must manage this pipeline to achieve their goals.

Data coming from the LANai to Host2 must use DMA to move data to host memory. DMA engines require that target memory be paged-in by the operating system. Usually, host memory is page-pinned so that the host operating system never pages-out this memory. While a variety of methods are possible, FM uses a static pinning policy. That is, the device driver pins one large region of memory when the device is loaded. The size of this region

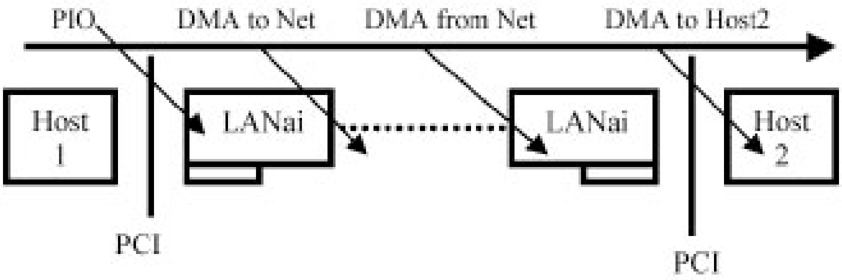

Fig. 1. Message packet traversing a pipeline.

is specified as a combination of maximum number of hosts in the cluster, the size of an FM packet (currently 2048 bytes of data for a total of 2080 bytes, including header), and the number of communication contexts. FM uses the term *communication context* to specify how many independent processes on a host must have simultaneous access to the high-speed network device. Special care (using the virtual memory protection mechanism of the host OS) keeps these contexts from interfering with each other. This is especially important because access to the LANai memory is memory-mapped into user space, allowing OS-bypass for common network interactions.

Because LANai memory is memory-mapped into one of multiple communication contexts, the LCP must distinguish incoming packets and place them in the correct location. This is accomplished by having the LCP inspect the FM header for a program ID. When a context first gains access to the device, its program ID (assigned from a global resource manager) and the instance number of that ID are placed into LANai memory.

The Myrinet network (and Giganet VIA networks) are considered to be hardware reliable, with very low bit-error rates (equivalent to memory bit-error rates). Taking advantage of this hardware reliability, FM is able to eliminate the software overhead of timeout/retransmission strategies. However, to ensure that data are received correctly, there must be available host memory at the receiver for every incoming packet. To ensure that memory is available, FM uses a credit-based flow control scheme. At startup, every node in a computation is assigned a certain amount of credit (measured in FM packets). All packets (full or partial) consume one credit. When a sender runs out of credit, it must wait for a credit refill. FM currently uses a static allocation policy, but the pinned receive region is actually a shared allocation. That is, the sum of all assigned credit across all nodes is shared from a single region. Changes to the FM credit allocation policy can be implemented without changing the MCP operation. In fact, the MCP knows nothing of credit, and dutifully acts as a packet forwarder. Incoming packets are placed in the next slot of the pinned memory region (with wrap at the region boundary). The FM credit policy ensures that packets will not be overwritten before being extracted from the network.

The FM library requires programs to extract packets from the network. FM is a handler-based system, so the packet type is associated with a "handler" code run on the destination host (active-message–like). The FM API call `FM_Extract()` allows the host library to examine packets that have arrived in the host's pinned memory region and to call the correct message handler. Because of this structure and in conjunction with the credit-based flow control scheme, when a sender runs out of credit, it is required to call `FM_Extract()`. This ensures message progression when two hosts are performing a head-to-head send. FM is completely thread-safe, so that multiple messages streams (say from different senders) can be handled safely.

4 Memory-Based Notification and Supporting Multiple NICs

4.1 Memory-Based Atomic Notification

In the previous versions of HPVM, applications detected the arrival of a new FM packet by calling an FM function that polled a memory location on the LANai interface. This is an expensive operation, because the request has to arbitrate for the CPU/memory bus, arbitrate for the PCI bus, then extract the value from the LANai's local memory and return it all the way back to a CPU register.

As described in Sect. 3, after the LANai control program receives a new packet from the network, it moves it into a main memory location, then the FM user-level library code extracts the packet from this location. The key is that the FM user-level library code and the LANai control program agree on the main memory location where the next packet is to be delivered. With this fact, it seemed clear that we could detect (atomic notification) by polling that location in memory rather than poll a control register across the PCI bus. This strategy has two key benefits. After the first poll, all subsequent polls can be satisfied in-cache; subsequent polls need not arbitrate or use the CPU/memory bus. The second benefit is that polling for FM packets doesn't require arbitrating for the PCI bus, so polls will not disturb any ongoing DMA transfers. The former ensures that polling scales on large SMPs, and the latter ensures fewer "turns" of the PCI busses, which is a critical factor when utilizing them at close to 100 percent (as FM does).

In essence, the hardware's cache coherence mechanism is used to signal the arrival of new packets. For example, if an application is actively polling for a new packet, on the first poll, the application will cache the poll value (which is a location in main memory where the LANai control program will deposit the next new packet). Subsequent polls will be satisfied by the cache—these polls will not generate any traffic outside the CPU the application is running on. When the LANai control program moves a new packet into the expected main memory location, the computer system's memory controller will determine that the application is caching the poll value and use the hardware-cache-invalidate mechanism to flush the poll value from the CPU cache. The next poll will cause the CPU to fetch the updated poll value from main memory. This value will be changed, thus indicating a newly arrived packet. This allows an application to actively poll for new packets in an efficient manner; regardless of the number of polls, at most two transfers from CPU to main memory will occur.

Finally, we achieve atomic notification with a single DMA transfer for variable-sized packets by using a novel custom packet layout. When a new packet arrives at the LANai, the LANai control program determines the packet's length and appends the length to the packet. The LANai control program then calculates the offset into the frame in main memory so that the newly appended packet length is aligned at the bottom of the frame. This allows applications to poll a common flag location at the end of the frame, regardless of the packet size. The use of the packet length allows the packet data to be contiguous with the flag, allowing all of it to be moved in a single DMA transfer. When the last value in the packet is written to main memory, the application knows that the entire DMA operation is complete. This simple optimization has a major impact on performance, improving latency for a zero-byte FM message over the previous HPVM implementation by 15 percent (10.4 μs vs. 8.8 μs).

4.2 Supporting Multiple NICs

HPVM 1.9 also allows several FM transports to be used concurrently: a shared memory transport (described in the next section) either a Myrinet transport or a GigaNet transport. In our current implementation, switching between the Myrinet and GigaNet transports only involves changing a registry variable and restarting the FM context manager service.

The shared memory transport enables efficient communication when FM tasks send messages to other FM tasks within the same SMP. The shared memory transport uses the same "memory notification" polling strategy as described above for the Myrinet interface. This has a subtle side effect, which is that polling for new packets costs the same for the shared memory and the Myrinet transports. The parity in polling costs allows multiple transports to be supported simply, and this architecture scales well for large SMPs, many network interfaces, and many different network types all within the same machine.

5 Shared Memory Transport

A deficiency of HPVM 1.2 was the implementation of the loopback message path, which traversed down to the Myrinet hardware layer. Although peak bandwidth between two Myrinet connected machines approached 100 MB/s, the bandwidth for two processes on the same machine was under 40 MB/s. The simple loopback design simplified the message layer; however, to better support larger SMP machines, we have designed and implemented a shared memory transport. In HPVM 1.9, each FM process is connected to Myrinet for inter-node communication, and to a shared memory region for intra-node communication. Both transport layers are supported concurrently, and because the FM design requires a central authority to map virtual node identifiers to physical nodes, each process can look up in the Global Resource Manager to decide which transport to use for peer communication. Further, due to the efficient receiver polling of the Myrinet transport, nothing more than adding a poll of shared memory is require to preserve low-latency communication for both transport layers.

The shared memory transport (hereafter referred to as SMT) uses the shared memory IPC mechanism provided by the host operating system. Although shared memory is straightforward, it does introduce the complication that pointer operations are unsafe, because there is no guarantee that separate processes will map the same memory region into the same address space. For this reason, all data structures used in SMT are array based. The basic design of SMT is for each process to control a message queue and permit other processes to atomically insert data into the queue. The implementation does not support any zero-copy IPC mechanism.

Mutual exclusion into shared message passing queues is low-latency and lock-free. The primary synchronization in SMT is a `fetch-and-increment` operation, which atomically fetches a value from memory and increments the value in place. By using a `fetch-and-increment` on the head and tail indices of a queue, lock-free mutual exclusion is guaranteed. The constraint that the queue length is a power of two removes any length overflow checks on indices when an appropriate bit mask is applied during index calculations. `Fetch-and-increment` is relatively inexpensive; on a 300 MHz Pentium II, our measurements show a single operation takes 85 nanoseconds (about 28 processor cycles).

The ordering property of `fetch-and-increment` is also used to assign SMT identifiers to processes. As processes attach to the shared memory region, they all `fetch-and-increment` a shared variable to determine their unique SMT identifiers. Just as a map of virtual node identifiers to physical node addresses is stored by the Global Resource Manager, a map of SMT identifiers to message queue locations is stored in the shared memory region. The map is used by senders to determine the message queue used by an SMT peer process.

Copying data into a frame in the message queue can be an expensive operation, and the host operating system may deschedule a process during this step. To allow for scheduler intervention without disrupting the sending and receiving in a message queue, the buffer filling and buffer pass-off steps are separated. Further, to allow for simple buffer reclamation, all buffers are controlled by the receiving process. Each process owns a free list and a receive queue. The free list is a ring buffer of 2K message frames, and the receive queue is a ring buffer of indices into the free list.

To send a message, a process must do the following.

1. Look up the receiving peer queue location using the SMT ID to queue location map.
2. `Fetch-and-increment` the head of the receiver's free list to consume a message frame.
3. `Test-and-set` an `inuse` flag in the frame. If the fetched value is non-zero, then go to step 2.
4. Copy data into the frame.
5. `Fetch-and-increment` on the tail of the receiver's receive queue.
6. Set the index at the fetched receive queue tail to the index of the consumed free list frame.

Note that under unfair scheduling, the sending process may traverse the free list in worst-case $\mathcal{O}(N)$ time, where N is the number of frames in the free list. The finite worst-case bound is a consequence of the fact that SMT is built on top of FM's credit-based flow control, which guarantees that a message frame will be present in the free list. However, assuming fair scheduling, or a single sender, the traversal time is $\mathcal{O}(1)$.

To receive a message a process must do the following.

1. Poll until the frame at the head of the receive queue is marked in-use.
2. `Fetch-and-increment` the head of the receive queue.
3. Index into the free list, and copy the data into the upper message layer if needed.
4. Unset the frame-in-use flag.

Although this scheme effectively reorders the receive queue according to the scheduling priority of the senders, it requires an additional `fetch-and-increment` and `test-and-set` operation, and adds some complexity to a single-queue scheme. This scheme will not protect from starvation when different process priorities are used. If a sender is scheduled at a higher priority, a lower priority sender may never acquire a message frame from the free list. The key advantage of this scheme is the reordering of senders to allow a sending process to effectively leapfrog over another, which keeps the receiver's message queue flowing. Without this property, a descheduled sender can block the receiver's message queue, thereby inducing bursts of high-latency communication directly correlated to the behavior of the host operating system scheduler.

To provide scalability to large SMP machines, the number of message queues a receiver owns is a parameter that can be adjusted to the amount of queue sharing desired. If the number of queues is set to one per receiver, then SMT has the same queuing structure as the Myrinet connectionless transport, where all senders transmit to the same device. The advantage of this scheme is that the receiver need only poll one receive queue, yielding low latency on the receiver side. The disadvantage is high contention due to numerous `fetch-and-increment` operations at the end points of the queue, which can result in high latency on the sending side. If the number of queues is set to 1 per receiver/sender pair, then SMT has the same queuing structure as the Giganet connection-oriented transport. The advantage of this scheme is that no synchronization primitives are needed in the queues, because this is a single writer/single reader scenario. The disadvantage is wasted memory, unless every permutation of process pairs communicates. More significantly, a receiver must poll a queue for every sender in order to extract messages, resulting in high latency on the receiver side.

If the number of queues can be set anywhere from one per receiver to N per receiver—where N is the number of SMT processes—then the amount of queue sharing, memory usage, and contention can be tuned for a given platform. If the number of receive queues is set at M—where $1 \leq M \leq N$—then the procedure for sending a message is the same as above, except that the receiver's free list and receive queue are now found as the receiver's $I \mod M$ free list and receive queue. Further, the receiver must now poll M receive queues.

The resulting performance of HPVM 1.9 with SMT improves on that of HPVM 1.2, both in delivered bandwidth (see Fig. 2) and in message latency (see Fig. 3). Incorporating the SMT into the FM system not only improves intranode communication performance, as discussed in Sect. 4, but it also does so with no penalty for internode communication.

Finally, to allow the SMT to run without a Myrinet network, we created a NULL transport to replace the Myrinet transport. This allows HPVM to run on a single workstation without any external transport hardware, resulting in a shared-memory-only implementation of HPVM. We envision this mode of operation becoming very common for HPVM users, because it allows them to develop HPVM software on their desktops.

6 PCI Chipset Pacing

HPVM (and FM in particular) was carefully tuned to use processor I/O and DMA asymmetrically to give highest performance. However, as PCI implementations have proliferated, we have found that PIO performance can be unpredictable. The following describes our experience and the adaptive tuning software we developed, which tests and selects good burst-size parameters to achieve good PIO performance.

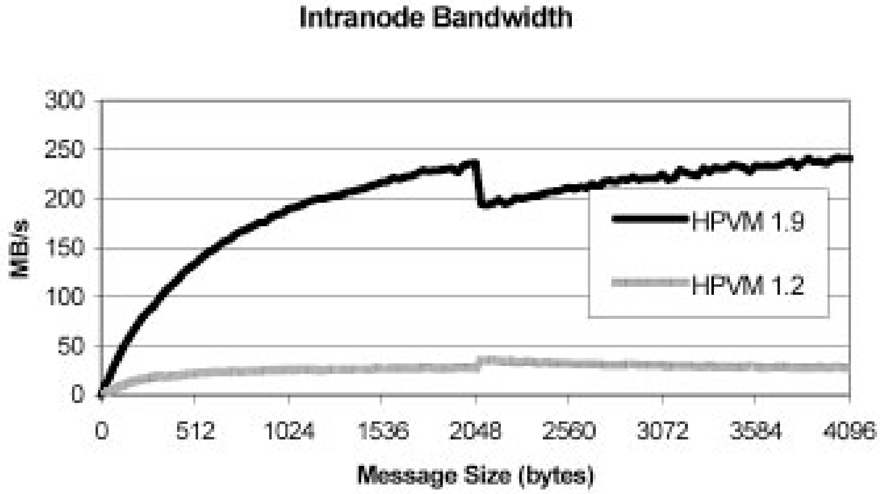

Fig. 2. Intranode FM messaging bandwidth, HPVM 1.2 vs. HPVM 1.9 (SMT).

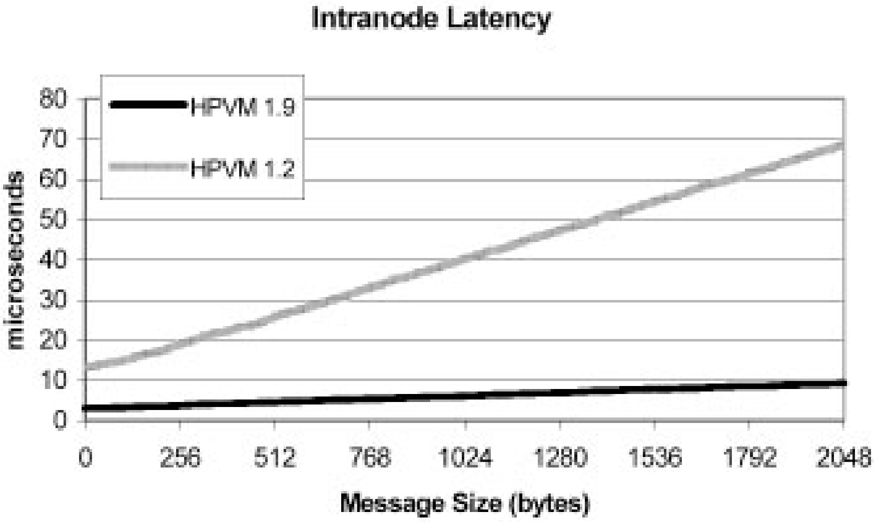

Fig. 3. Intranode FM messaging latency, HPVM 1.2 vs. HPVM 1.9 (SMT).

In December of 1998, we received a new 32-node cluster. Each node is an HP Netserver LPR that contains two 450 MHz Pentium IIs that are connected to a 100 MHz front-side bus and are coupled to main memory and I/O interfaces via the 440 BX chipset. Our previous cluster consisted of HP Kayak nodes. A Kayak node contains two 300 MHz Pentium IIs that are connected to a 66 MHz front-side bus and are coupled to main memory and I/O interfaces via the 440 LX chipset.

We were eager to compare FM performance for the clusters because the new cluster's processors and front-side buses are 50 percent faster (66 to 100 MHz). Preliminary FM micro-benchmarks (see Fig. 4) showed that the newer machines performance was better for smaller messages, but it also revealed that the newer machines performed worse for larger messages. The root of this behavior was inefficiencies of the PCI chipset at saturation, even though peak performance of the newer chipset is indeed higher than its older cousin. After adding sufficiently heavy instrumentation to the FM internals, it was discovered that the newer machines have worse PIO performance than the older machines for messages larger than 160 bytes. The slower machines' performance is robust, and performance continues to increase as message sizes increase; but the newer machines peak at 115 MB/s, and performance falls for messages of 160 bytes and greater. We verified that all configuration parameters (BIOS, chipset, and OS) between the two machines were identical.

Our working theory is that the queuing mechanism inside the 440 BX does not gracefully handle high-volume PIO scenarios (the queue in the 440 BX is 5 cache lines deep, that is, 160 bytes). With this theory, the following strategy was implemented: for messages more than 160 bytes, packets are "paced" so that the 440 BX chipset can perform optimally—that is, for messages larger than 160 bytes, data is PIOed across the bridge at 160 byte chunks. For example, if an application is sending a 400-byte message, the FM internal pacing code will PIO the message to the LANai in three transactions: two 160-byte chunks followed by an 80-byte chunk. This technique conditioned the newer machines' performance curves and expected performance as shown in Fig. 5.

To ensure that this performance is robustly attained in other systems, we inserted a PIO micro-benchmark in the distributed code (the same code used to uncover the 440 BX problem) that runs at boot time. This code analyzes the system's performance and chooses a pacing size for the system that attains the highest PIO performance.

7 Summary and Future Directions

The design of HPVM 1.9 builds on several generations of high-performance cluster software systems from the Concurrent Systems Architecture Group, originally at the University of Illinois and now at the University of California,

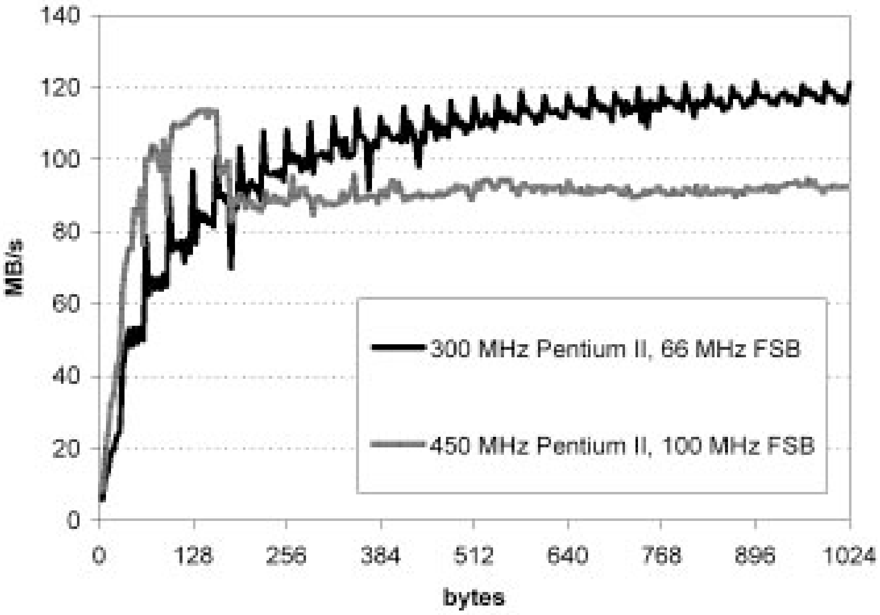

Fig. 4. FM micro-benchmarks without packet "pacing."

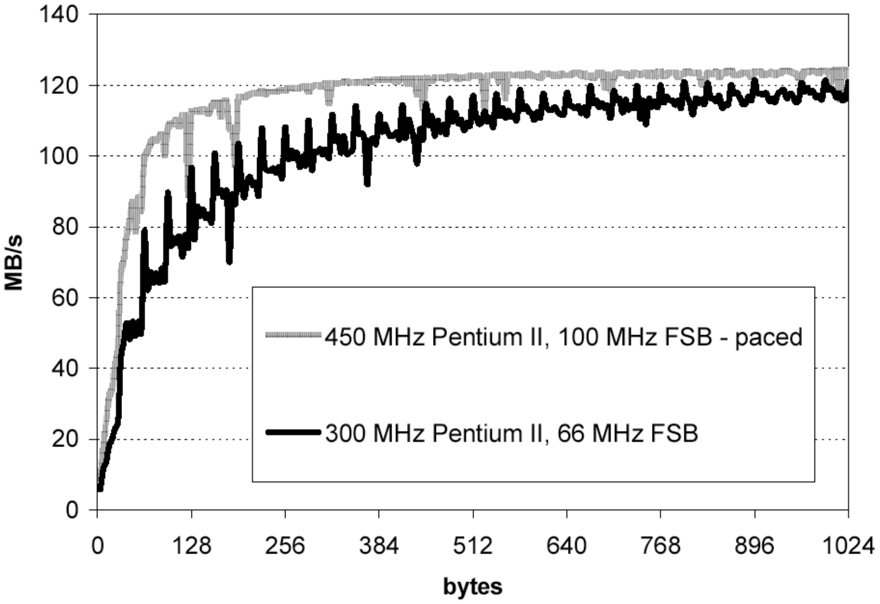

Fig. 5. FM micro-benchmarks results with packet "pacing."

San Diego. These previous systems have been used to build large production clusters and to deliver a wealth of application computational science. HPVM 1.9 adds a number of capabilities that further increase the flexibility, performance, and potential impact of the tools. Our efficient integration of multiple transports with high performance in a robust structure (a structure that is self-tuning and has low overhead) represents a new solution to an old problem [16,9]. Other specific new capabilities in HPVM 1.9 include support for multiple hardware network types (Giganet and shared memory), a workstation configuration for convenient software development, support for the Bulk-synchronous Programming API, and improved performance (8.8 μs latency, 100+ MB/s, see Fig. 2 and Fig. 3).

Currently, the NCSA Alliance NT Supercluster, which comprises more than five hundred processors, is using HPVM 1.9 (replacing HPVM 1.2 on that cluster). This cluster will continue to expand rapidly, because its delivered performance, reliability, and reasonable cost make it a compelling vehicle for computational science. We are exploring the use of HPVM 1.9 as a basis for a wide-area cluster federation (coupling over networks from OC-12 to OC-192, 10 gigabits/s and faster). We have successfully coupled the 128 processors of the UCSD cluster with the 256 processors of the NCSA NT Supercluster, using the OC-12 vBNS. We are exploring higher-speed network experiments, as well as the use of high-speed clusters to build a cost-effective network storage server. We have integrated a five-terabyte server and demonstrated a gigabyte per second of sustained external network bandwidth. We are working with other researchers to use this in computational grid applications.

Acknowledgments

The research described is supported in part by DARPA orders E313 and E524, through the US Air Force Rome Laboratory Contracts F30602-99-1-0534, F30602-97-2-0121, and F30602-96-1-0286, and NSF Young Investigator award CCR-94-57809. It is also supported in part by funds from the NSF Partnerships for Advanced Computational Infrastructure—the Alliance (NCSA) and NPACI. Support from Microsoft, Hewlett-Packard, Myricom Corporation, Intel Corporation, Packet Engines, Tandem Computers, and Platform Computing is also gratefully acknowledged.

References

1. American National Standard for Information Systems. Fiber-distributed data interface (FDDI): Token ring media access control (MAC). Technical Report ANSI X3.139-1987, American National Standard for Information Systems, July 1987. 2

2. T. M. Anderson and R. S. Cornelius. High-performance switching with fibre channel. In *Digest of Papers Compcon 1992*, pages 261–268. IEEE Computer Society Press, 1992. 2
3. D. Becker, T. Sterling, D. Savarese, J. Dorband, U. Ranawak, and C. Packer. Beowulf: A parallel workstation for scientific computing. In *International Parallel Processing Symposium Proceedings*, 1995. 2
4. N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W. Su. Myrinet: A gigabit-per-second local-area network. *IEEE Micro*, 15(1):29–36, February 1995. 2
5. B. Chun, A. Mainwaring, and D. Culler. Virtual network transport protocols for Myrinet. In *Hot Interconnects V Proceedings*, August 1997. 3, 3
6. K. Connelly and A. A. Chien. FM-QoS: Real-time communication using self-synchronizing schedules. In *SC97 Proceedings*, November 1997. 4
7. P. Druschel, L. L. Peterson, and B. S. Davie. Experiences with a high speed network adaptor: A software perspective. In *SIGCOMM '94 Symposium Proceedings*, pages 2–13, August 1994. 3
8. C. Dubnicki, A. Bilas, Y. Chen, S. Damianakis, and K. Li. VMMC-2: Efficient support for reliable, connection-oriented commnication. In *Hot Interconnects V Proceedings*, August 1997. 3
9. I. Foster, C. Kesselman, R. Olson, and S. Tuecke. Nexus: An interoperability layer for parallel and distributed computer systems. Technical Report Version 1.3, Argonne National Laboratory, December 1993. 14
10. D. Garcia and W. Watson. Servernet II. In *Parallel Computer Routing and Communications Workshop Proceedings*, LNCS. Springer-Verlag, 1997. 2
11. L. A. Giannini and A. A. Chien. A software architecture for global address space communication on clusters: Put/get on fast messages. In *High-Performance Distributed Computing Conference Proceedings*, 1998. 3
12. F. Hady and B. L. Menezes. The performance of crossbar-based binary hypercubes. *IEEE Transactions on Computers*, 44(10):1208–1215, October 1995. 3
13. IEEE. Standard for scalable coherent interface (SCI) specification. Std. 1596-1992, IEEE, August 1993. 2
14. M. Lauria and A. Chien. MPI-FM: High performance MPI on workstation clusters. *Journal of Parallel and Distributed Computing*, 40(1):4–18, January 1997. 3, 3
15. M. Lauria, S. Pakin, and A. A. Chien. Efficient layering for high speed communication: Fast messages 2.x. In *High-Performance Distributed Computing Conference Proceedings*, 1998. 3
16. S. Lumetta, A. M. Mainwaring, and D. E. Culler. Multiprotocol active messages on a cluster of SMPs. In *SC97 Proceedings*, San Jose, CA, November 1997. 2, 14
17. S. Pakin, M. Lauria, and A. Chien. High performance messaging on workstations: Illinois fast messages (FM) for Myrinet. In *SC95 Proceedings*, volume 2, pages 1528–1557, 1995. 3, 3
18. L. Prylli and B. Tourancheau. Protocol design for high performance networking: A Myrinet experience. Technical Report 97-22, LIP, Ecole Normale Superieure de Lyon, July 1997. 3
19. P. Sobalvarro, S. Pakin, A. Chien, and W. Weihl. FM-DCS: An implementation of dynamic coscheduling on a network of workstations. In *ASPLOS-VII NOW/Cluster Workshop*, Cambridge, MA, October 1996. 4

20. H. Tezuka, A. Hori, and Y. Ishikawa. PM: A high-performance communication library for multi-user parallel environments. Technical Report TR-96-015, Tskuba Research Center, Real World Computing Partnership, November 1996. 3

21. *The Virtual Interface Architecture Version 1.0*, December 1997. Promoted by Intel, Compaq, and Microsoft and available from `<http://www.viarch.org/>`. 3

22. T. von Eicken, A. Basu, V. Buch, and W. Vogels. U-Net: A user-level network interface for parallel and distributed computing. In *15th ACM Symposium on Operating Systems Principles Proceedings*, pages 40–53, December 1995. 3

Presenters' Biographies

Andrew A. Chien

SAIC Chair Professor
University of California, San Diego, USA

Andrew A. Chien's research involves networks, network interfaces, and the interaction of communication and computation in high-performance systems. Andrew is the Science Applications International Corporation (SAIC) Chair Professor in the Department of Computer Science and Engineering at the University of California, San Diego. He received his undergraduate, master's, and doctoral degrees from the Massachusetts Institute of Technology in 1984, 1987, and 1990 respectively. From 1990 to 1998, Professor Chien was a faculty member in the Department of Computer Science at the University of Illinois and a Senior Research Scientist in the National Center for Supercomputing Applications. He is currently working on large-scale clusters with both NCSA and NPACI. Andrew is also recipient of a 1994 National Science Foundation Young Investigator Award, a 1995 C. W. Gear Outstanding Faculty award, and a 1996 Xerox Outstanding Research Award.

You can find out more about Andrew A. Chien on the Web at <http://www-csag.ucsd.edu/individua