SMP PCs: A Case Study on Cluster Computing

Antônio Augusto Fröhlich*

GMD FIRST
Rudower Chaussee 5
D-12489 Berlin, Germany
guto@first.gmd.de

Wolfgang Schröder-Preikschat

University of Magdeburg
Universitätsplatz 2
D-39106 Magdeburg, Germany
wosch@cs.uni-magdeburg.de

Abstract

As commodity microprocessors and networks reach performance levels comparable to those used in massively parallel processors, clusters of symmetric multiprocessors are starting to be called the supercomputers of tomorrow. At the low-end of this technology are the clusters of SMP PCs, usually based on Pentium Pro or Pentium II processors. Many groups in the academia and in the industry are setting up such clusters with big expectations. However, how far can one go with a cluster of SMP PCs when the goal is high performance computing?

This paper discusses several aspects regarding the adoption of clusters of SMP PCs to support high performance computing, including software and hardware restrains to deliver processing power to parallel applications, as well as the most innovative alternatives to overcome these restrains. Besides discussing the current state of PC cluster computing, the authors identify further enhancements that will help to enable these low-cost machines to join the HPC universe.

1. Introduction

The current state of microprocessor and interconnection technologies is enabling the conception of low-cost, high-performance parallel machines based on commodity components. As microprocessors benefit from large scale production (and sales) to support technological improvement, the custom processors formerly used on parallel machines are losing their space. With high performance microprocessors available, most massively parallel processors (MPP) are now being made of off-the-shelf microprocessors. At the same time, the microcomputer industry is using this technology to make available symmetric multiprocessors (SMP) that show many architectural features of yesterdays supercomputers, and thus represent a meeting point for both technologies.

Together with powerful processors, powerful interconnection networks, sometimes very similar to those used in MPPs, are commercially available. Some commodity networks deliver higher bandwidth than many computer busses, frequently at very low latencies. Many network adapters also include enough circuitry to implement communication strategies without impacting main processor performance. This has encouraged several experiments on clustering SMPs to accomplish cost-effective, high-performance machines.

However, when we compare a Pentium based SMP PC to some other RISC based SMPs currently available, we realize that there is still a big gap between them. Moreover, the hardware represents just one part of the accomplishment of high performance computing (HPC) on clusters of SMP PCs. Appropriate software must be provided to deliver the computational power to applications. Developing low overhead software to support parallel computing in cluster environments constitutes the big challenge, specially when traditional programming interfaces are to be preserved.

In this paper we identify and discuss key points on this emerging technology. We start by discussing the main characteristics of current clusters of SMP PCs, including processors and interconnection networks. Then we discuss the software commonly adopted on such architectures, analyzing processing and communication strategies. We finish this paper with a balance of what can already be taken for grant and what is still missing to bring clusters of SMP PCs into the HPC scenery.

*Work partially supported by Federal University of Santa Catarina and by Fundação Coordenação de Aperfeicoamento de Pessoal de Nível Superior.

2. Hardware Aspects

As the purpose of this paper is to analyze the possibilities of using clusters of SMP PCs to support high performance computing and not the clusters themselves, we will concentrate on hardware aspects that can highly influence the software design, either by impacting performance or by imposing design restrictions.

The next sections discuss the fundamental components of a cluster of SMP PCs: the SMP PCs themselves and the interconnection networks. Often, high performance applications, besides processing and communication, will also demand high performance file systems. However, we will not discuss the theme here.

2.1. Symmetric Multiprocessors PCs

Traditional SMP PCs are comprised by up to 4 Pentium processors[1] sharing a single memory, as shown in figure 1. The expression symmetric multiprocessor is derived from the fact that there are no specific purpose processors and that all processors have the same privilege when accessing the memory.

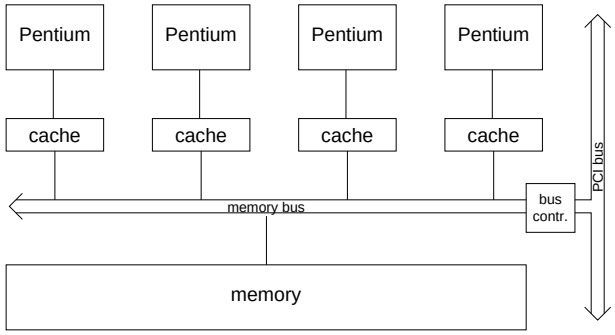

Figure 1. The structure of a typical SMP PC.

The symmetry among the processors in a SMP enables very simple process management strategies, as any ready to run process (thread) can be dispatched to any processor. In this way, porting traditional single processor operating systems is also made easy. As a matter of fact, however, achieving good performance in SMPs will imply in making the operating system itself a concurrent program, what is far more complex.

For purposes of performance analysis, it is possible to split a SMP PC into processor, memory and I/O. These two last, in combination with some glue circuitry, are referred to as chipset. Different chipsets, although showing the same interface, can present considerably different performance, due exactly to the glue circuitry. However, as currently only one chipset for the Pentium II processor, the Intel 440LX, is available, our analysis will disregard the coupling details of the chipset. Moreover, comparing bridging components of a chipset is a task that demands for special hardware analysis tools and, in our opinion, should be conducted by the industry.

2.1.1 Intel P6 Processors Family

The Intel P6 family of microprocessors, that currently includes Pentium Pro and Pentium II, is widely used on high-end microcomputers. The P6 family presents a highly optimized internal architecture that benefits from a complex pipeline with multiple parallel functional units and is able of out-of-order, speculative execution. Even if most of these characteristics are deeply hidden behind the old ix86 macro-architecture, which interface includes only 8 general purpose registers, benchmarks that disregard memory access latency show quite good results for the P6 family: the Pentium II at 300 MHz achieved 9.3 SPECint 95 and 8.9 SPECfp 95[2].

2.1.2 Memory

The P6 processors family performance has been encouraging many to adopt PCs to support HPC, but most of the enthusiasm vanishes when we get to memory latency. SMPs in general, when compared to more sophisticated parallel machines, present very high memory latencies, mainly due to its simple organization. Nevertheless, this weakness is frequently compensate on RISC based SMPs with the adoption of large register sets in combination to large private caches for each processor.

Unfortunately, the adoption of 512 Kbytes second level caches with P6 processors is not enough to compensate the poor memory buses of traditional PCs, which bandwidth, according to our measurements, is restricted to approximately 145 Mbytes/s (3.2 Gbytes/s in a UltraSPARC based Sun HPC 5500). Memory contention problems have been reported by several groups. In particular, the measurements of a four Pentium Pro SMP running Solaris, presented in [12] and summarized in the figure 2, show that the memory copy bandwidth available to each processor in the SMP decreases drastically with the number of processors. Moreover, the same paper reports that, for some data intensive applications running in a four processors configuration, two processors spend most of the time waiting for the memory subsystem.

[1] The Pentium II processor with the "Slot 1" connector supports 2-way SMP only.

[2] According to SPEC official results, the best performing processor in February 1998 was the Alpha 21164 at 600 MHz, with 18.8 SPECint and 29.2 SPECfp.

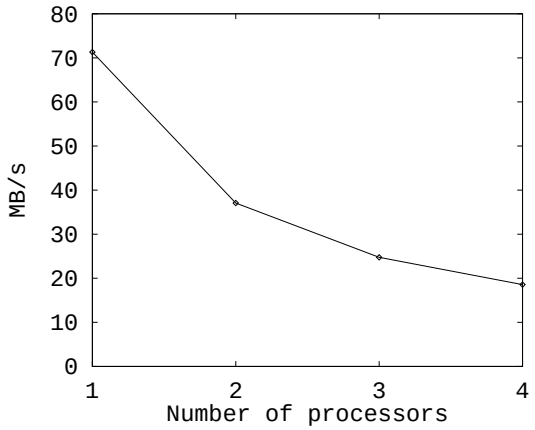

Figure 2. Per processor memory copy bandwidth in an ordinary SMP PC with four Pentium Pro processors.

2.1.3 I/O Bus

For current PCs, communication is just an ordinary I/O operation. Thus, in order to cluster PCs, one have to plug some sort of network interface card (NIC) into the I/O bus. Independently of the selected bus, remote communication will always go through it and, some times, will be restrained by it. Usually the operating system designers can rely on a DMA controller or on some sort of burst transfer in the I/O bus, nevertheless, these mechanisms are often insufficient to exploit the available network bandwidth.

A typical case where the bus restrains the network is a Myrinet NIC on a PC PCI bus. Myrinet will be described in more details latter, for while it is enough to know its raw bandwidth: 1.28 Gbits/s. There are two basic ways to operate a Myrinet NIC: programmed I/O and DMA. The decision of which technic to use should consider that, even if DMA is usually quite faster than programmed I/O, it also requires the physical address where the data is to be transfered to / from to be know in advance. Besides that, DMA requires the data to be contiguously stored in memory. Therefore, using DMA implies in moving the data to a well-known area or re-programming the DMA controller with buffer physical addresses translated by hand. Both alternatives can put DMA performance back to programmed I/O levels. In a standard 32-bits PC PCI bus at 33 MHz, the peak throughput is 355 Mbits/s for programmed I/O and 1066 Mbits/s for DMA, disregarding copies, re-programming and address translations. Even the theoretical DMA bandwidth is not enough to exploit the bandwidth of Myrinet and of other high speed networks.

2.2. Communication

The scenery of interconnection hardware to cluster SMP PCs is a quite large one, including, among others, Fast and Giga Ethernet, FDDI, ATM, Fiber-channel, Myrinet and SCI. In order to discuss it, we selected two very distinct representatives: Myrinet and SCI. Both are high speed networks well suited and often used to cluster not only PCs, but SMPs in general. The first aims to support message passing, while the second is designed to support a distributed shared memory scheme in hardware.

2.2.1 Myrinet

Myrinet [1] is a high speed network commercially produced by Myricon Inc. It has its roots on the Caltech Mosaic massive multiprocessor project, from which it inherited, among other characteristics, the worm-hole routing strategy. Myrinet's hardware and software interfaces, as well as protocols, are published and open, what has encouraged several research projects.

Myrinet NICs are currently based on the LANai 4.1 chip, which includes a RISC processor, two DMA controllers for pushing and pulling bit streams into / from the network and one DMA controller to transfer messages from / to the host memory. Besides the LANai, a Myrinet NIC also includes a full duplex link interface, a host bus interface (currently SBus and PCI) and from 256 Kbytes to 1 Mbytes of static RAM that can be used to store the control program for the LANai processor and also to store message buffers. These NICs are able of transferring arbitrary length messages, to which they automatically generate / verify a checksum, at a raw bandwidth of 1.28 Gbits/s. The layout of a Myrinet NIC is presented in figure 3.

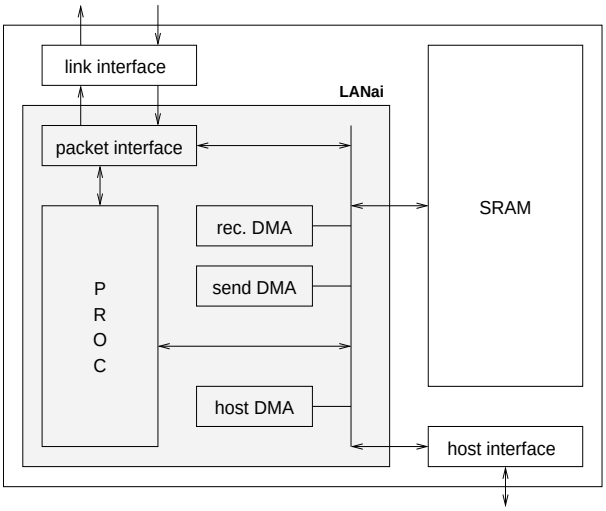

Figure 3. The layout of a Myrinet NIC.

A Myrinet switch is basically a 8x8 mesh, with a per-hop latency of around 0.5 μs, where worm-hole routing takes place. Switches can be arbitrarily connected to form virtually any topology, some of which are shown in figure 4. However, although arbitrary, the network topology must be realized in order to enable source routing. Other important characteristics of Myrinet switches are packet ordering preservation and flow control. Packets sent through the same path are delivered by the switch in the same order they were sent. Flow control is handle in hardware by blocking a packet when a selected output port is busy. Packets can be blocked for some short time, after what they are dropped. This hardware flow control propagates until the source NIC stops pushing packets into the network.

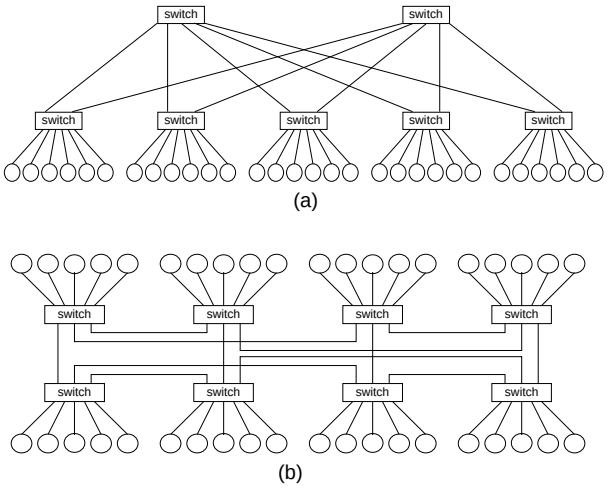

Figure 4. Two possible Myrinet topologies: a fat-tree (a) and a cube (b).

The Myrinet NIC organization is specially interesting for operating system designers because communication protocols and strategies are implemented in software and can be easily modified. Therefore, several communication packages are available for Myrinet, including emulations for standard networks, like Ethernet and ATM, standard protocols like TCP/IP and many experimental protocols. This flexibility is probably the reason why Myrinet has been widely used to cluster SMPs. Research projects adopting Myrinet include, among others: NOW from the University of California at Berkeley [3]; Fast Messages from the University of Illinois [10]; COMPaS from the Real World Computing Partnership [12].

2.2.2 Scalable Coherence Interface

The Scalable Coherence Interface (SCI), as the name indicates, is not a network, but an interface specified in the IEEE standard 1596 from 1992. The standard defines hardware and protocols to tightly connect up to 64 K nodes. The most remarkable difference from a traditional network to a SCI compliant one is that in SCI data is exchanged among nodes using implicit communication, via remote memory access, in stead of using explicit message passing.

SCI defines a global 64 bits address space shared by all nodes in the cluster and accessible through bus-like transactions for reading, writing, moving and locking memory locations. These transactions are atomic, dead-lock free and preserve memory and cache coherence along the cluster. Up to 64 transactions may be outstanding on each node, what allows for better bandwidth utilization. In this context, nodes can export segments of its local physical address space by mapping them into the SCI global address space. In brief, SCI specifies a hardware based distributed shared memory.

The scalability question is addressed in the standard by only employing unidirectional, point-to-point links to connect nodes. By doing so, the number of nodes in a SCI cluster in not limited by the length or by the speed of the network, as it would be in a SMP bus. Nodes can be gathered in rings or through switches, although the ring is usually adopted. Larger clusters can be build of rings of rings or of rings interconnected by switches. Some examples of SCI topologies are shown in figure 5.

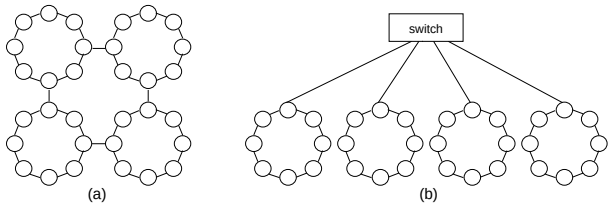

Figure 5. Two possible SCI topologies: a ring of rings (a) and a tree of rings (b).

Data integrity is maintained in hardware by means of checksums that are generated and verified for each transfered packet. If a bad packet is received, it is simply discarded. The associate transaction will eventually time-out, causing the packet to be retransmitted. This combination of checksum and time-out ensures reliable transactions. Nevertheless, SCI protocols do not ensure in-order delivering.

Current implementations of SCI NICs deliver raw bandwidth of 1 Gbits/s and are available for SBus and PCI buses. Usually only a subset of the SCI standard is implemented. Cache coherence implementation, for instance, is infeasible in NICs connected to the I/O bus and is only implemented in custom architectures. Lock transaction are often missing too. NICs connected to the I/O bus implement the hardware distributed shared memory scheme in two phases: at first

a contiguous segment, in the I/O range of the physical address space designated to the NIC, is allocated and mapped into the SCI global address space in the area reserved for the respective node; then, the segment is mapped into the address space of some local processes. Every time the segment is referenced, the NIC generates the appropriate SCI transaction.

The big appeal of SCI is that most of the effort to support application communication is done in hardware. Moreover, passing the SCI distributed shared memory abstraction to applications is a tempting solution to support software developed for shared memory machines. Good performing message passing layers have also been developed on top of SCI. Research projects adopting SCI include, among others: SMiLE from the Technical University of Munich [5]; HPVM from the University of Illinois [2]; Scintilla from the University of California at Santa Barbara [7].

3. Software Aspects

Knowing the characteristics and limitations of the underlying hardware is fundamental for a good software design, specially because, as a matter of fact, the hardware in a computer system is far more evolved than the respective software, which is responsible for the major limitations imposed on applications. Therefore, software design aspects must be carefully studied when aiming to provide flexible, high performance solutions.

In the next sections we discuss the different approaches to support processing and communication in clusters of SMP PCs.

3.1. Processes

The final goal of any computing system is to deliver computing power to applications. Whatever abstraction chosen to achieve this should aggregate a bare minimum overhead. Besides, to bring a cluster of SMPs to the parallel machines universe, the process abstraction delivered to application programmers should preserve many centralized characteristics of shared memory machines, yet implemented in a distributed fashion. That is, the process abstraction should make several points in the distributed nature of a cluster transparent to applications. Reaching this transparency implies in implementing some sort of global scheduling, synchronization and communication mechanisms. To what extent to implement each transparency mechanism is a compromise to the (high performance) applications. In which level to implement them, whether in kernel or user level, is a compromise between scalability and performance in one side and generality in the other.

However, independently of where to place the user / kernel barrier, if any, adopting multi-threading is a consensus in cluster computing. Besides the advantages of adopting it to hide latency in the client / server scenery, clusters of SMPs benefit from multi-threading as a programming paradigm that allows for implicit communication on each SMP. Parallel programs can thus be implemented as collections of multi-threaded processes where each process executes in a different SMP. Inter-process (inter-node) communication is then handled using message passing, while intra-process (intra-node, inter-processor, inter-thread) communication is handled via shared memory.

To discuss these questions, we divided processes, according to the level where the basic elements are defined, in two groups: kernel and user level.

3.1.1 Processes at Kernel Level

Although Unix-like operating systems are highly centralized and do not scale to distributed architectures, they are often employed in clusters, mainly because they fit perfectly in the idea of building up parallel machines out of commodity components, yet at a high overhead to applications. Many Unix implementations, like Solaris, BSD and Linux, are available in multi-threaded SMP versions for PCs.

The overhead imposed by Unix does not reside in the process implementation itself, but in the side-effects of running Unix. For many applications, having an operating system able of managing several peripheral devices, virtual memory, file systems, TPC/IP networks, etc, is nothing but overhead. Besides that, the absence of adequate cluster-wide scheduling, synchronization and communication mechanisms, asks for additional layers of software, like MPI or PVM. Even so, Unix is the most frequently used operating system in the cluster computing universe. The second most frequent choice, Windows NT, shares the same deficiencies.

A more sophisticate approach to implement the process abstraction at kernel level are the micro-kernels, which usually get rid of Unix overhead by pushing device drivers, file systems, virtual memory, etc, to user level. Unquestionably, many micro-kernels are more adequate to clusters of SMPs than monolithic Unix-like operating systems, however, the absence of a traditional user interface, like Posix and MPI, seems to constitute an intraversable obstacle. Therefore, several micro-kernels supply server or library based Posix interfaces. The fact that these interfaces often perform worst than Unix is the reason why micro-kernels are seldom saw in clusters of SMPs.

3.1.2 Processes at User Level

Implementing the process abstraction at user level is an approach motivated by the realization that different applications have distinct requirements regarding processes. When

processes are implemented at user level, its basic properties, like scheduling, grouping and synchronization, may be customized according to the application requirements.

The implementation of a user level operating system is usually based on libraries with several distinct implementations for each component. Applications programmers can then select the best choice to satisfy their needs. Library operating systems, as this approach is known, are extremely time consuming to develop and face two big challenges: customizing an operating system is a task out of the scope of most application programmers, what demands for automatic tools still to be implemented; the same interface restrain faced by micro-kernels verifies here.

An example of library operating system is MIT's Exo-Kernel [6]. An exo-kernel based operating system is composed by an exo-kernel that acts as a secure interface to the underlying hardware and by a library that implements the operating system abstractions. Although very flexible, the adoption of a Posix library makes Exo-Kernel to perform not significantly better than Unix.

A more effective approach is to use object orientation to specialize the operating system components. The implementation is also based on libraries, but, in the place of functions, one can find classes. Inheritance and composition can significantly reduce the efforts to supply broader sets of system components, and the applications themselves can still specialize most components. This strategy, adopted in the Peace [11] project from GMD-FIRST is one of the most adequate to support high performance computing, because the applications get exactly the operating system they need. Supplying a traditional API, like MPI or PVM, is also made easier in this approach, since the operating system can be specialized only till the point where it satisfies the API requirements, avoiding implementing a complete Unix layer.

3.2. Communication

The distributed nature of a cluster claims for effective inter-process communication. Fortunately, network technologies has been evolving in such a fashion to, sometimes, let the processors behind. In this context, developing efficient software and protocols became the real challenge.

Several studies based in the LogP[3] model [4] have shown that network bandwidth and latency are no longer the bottleneck of inter-node communication, while the overhead to carry out a message assumes the ungrateful role. Breakdowns like those presented in [9, 8, 12] confirm that the impact of increasing overhead is more destructive to most applications than the one caused by proportionally reducing bandwidth or increasing latency. In order to compare the overhead present in distinct communication implementations, one should distinguish between the overhead incurred when preparing the message (marshaling, encapsulating, fragmenting, ordering, etc) and that incurred in the path between the application and the communication handler. The first is inherent to the chosen communication strategy, while the second is usually eliminated when the communication is implemented at user level.

The demand for low overhead communications has led to several user level implementations. However, there are still cases in which the current user level implementations are not adequate. Following we discuss both possibilities.

3.2.1 Communication at Kernel Level

Implementing communication at kernel level is the natural approach for traditional operating systems, like Unix and Windows NT, because they consider the network adapter to be just another peripheral device. Interfacing the network to applications is normally done using the socket abstraction, an end-point for n-to-1 communication channels defined in the scope of a protocol, very likely TCP/IP. In this context, accessing the network implies in a system call that is dispatched by the operating system, through the TCP/IP stack, to the proper device driver. Moreover, the socket interface, in most cases, is not adequate to express communication in parallel programs, what leads for additional layers of software, like PVM or MPI, on top of it. Unnecessary to say the overhead on this scheme is much to high. In fact, traditional Unix and Windows NT communication is seldom adopted when the purpose of the cluster is to support high performance computing.

Micro-kernels, in the oder hand, export the network to applications through some particular view of links, ports, mailboxes or remote procedure calls. The basic abstraction is usually implemented inside the kernel, while the network specific code is implemented by an external (highly coupled) server. Moreover, TCP/IP is usually replaced by a lighter and more adequate protocol. Although significantly more adequate to parallel environments than monolithic systems, implementing communication inside a micro-kernel, due to the simple fact of crossing a system call barrier, still incurs in considerable overhead. The path from the user process to the network driver / server demands for around 10 μs in the fastest systems. This is the round-trip time for some high speed networks. Together with the user interface problem discussed previously, the high overhead restrain the use of micro-kernel based communication

[3] LogP is a theoretical model of a distributed memory multiprocessor that specifies the performance characteristics of the interconnection network through four parameters: *"L"* is the latency or delay incurred in sending a small message from its origin to its destination; *"o"* is the overhead or the time spent by the processor to transmit or receive a message; *"g"* is the gap or interval between consecutive message transmissions or receptions at a processor; *"P"* is the number of processors.

in clusters of SMPs.

3.2.2 Communication at User Level

Defining process at user level means to forget traditional operating systems and therefore is not a frequent approach in clusters of SMPs. Defining communication at user level, however, is far less traumatic. It is normally accomplished by implementing a device driver that initializes and then exports the network to applications. Once exported, the network can be accessed without kernel intervention. This scheme supports the implementation of communication abstractions, including protocols, at user level. Such abstractions can then be customized to satisfy the requirements of specific applications. User level communication can be adopted in traditional operating systems as well as in microkernels and library operating systems.

The major restriction to implement communication at user level appears in environments where the cluster is shared by mutually untrusted applications, what demands for secure multiplexing of the network. In this cases, some sort of kernel intervention is usually required. An interesting exception is the previously described SCI, that benefits from the MMU's protection capabilities to ensure secure multiplexing of the network.

Two frequently used strategies to support user level communication are active messages and asynchronous remote copy. **Active messages** is a communication model widely used in the parallel programming community due to its simplicity and efficiency. Active messages belong to the so called one-way communication group, because only the send part of a message exchange is explicitly expressed. Each sent message, besides data, carries along a reference to a handler, which is invoked when the message reaches its destination. The sent data is then treated by the specified handler in the scope of the receiving process. Explicit receives are not possible.

Interesting implementations of active messages came from the University of California at Berkeley and Santa Barbara. Berkeley implemented active messages on a cluster of Sun Enterprise 5000 servers interconnected by Myrinet, while Santa Barbara implemented active messages for a cluster of Sun UltraSPARC workstations interconnected by SCI . The table 1 summarizes the results reported by these two implementations in [8] and [7], respectively. Since the hardware platforms are not the same, the results can not be used for comparisons, however they give a notion of the current status of active messages implementations.

	Myrinet	SCI
Latency (μs)	13.8	9.3
Send overhead (μs)	5.6	3.1
Receive overhead (μs)	8.1	3.1
Gap (μs)	17.6	13.5
Bandwidth (Mbytes/s)	31.7	26

Table 1. LogP parameters and bandwidth for active messages implementations.

Asynchronous remote copy is also a one-way communication strategy, where asynchronous remote memory access is supported. The memory mapped in the address space of a process can be read or written by another, possibly remote, processes. Remote memory access can suppress buffer handling and unnecessary memory copies, nevertheless, a handshake to arrange for locations where to read data from or write data to in other processes' address spaces, as well as mechanisms to notify the receiver that a new message is available at the destination must be provided.

An example of asynchronous remote copy is Fast Messages from the University of Illinois. Fast Messages implementation on a cluster of Sun SparcStation workstations interconnected by Myrinet achieved bandwidth around 20 Mbytes/s and latency around 10 μs [10]. Another example is PM from the Real World Computing Partnership, which implementation for a cluster of Pentium Pro interconnected by Myrinet is reported to have achieved bandwidth of 82.5 Mbytes/s and latency of 16 μs [12].

4. Discussion

Much has been done in order to enable clusters of SMP PCs to support high performance computing. But, in the current development stage, what is satisfactory and what is not? In this section we try to evaluate some aspects of this complex question, again, splitting the subject into hardware and software.

4.1. Hardware Aspects

Many currently available SMPs, for instance SGI Origin and Sun Enterprise, present better price / performance ratios than those from traditional MPPs, like the Cray T3E and the IBM SP/2. The good performance of these SMPs is mainly due to the large register sets of its RISC processors, to its large caches and to its high speed memory buses. SMP PCs, however, are still far from this reality, basically because of the obsolete memory subsystem that imposes very high penalties on memory access.

SMP PCs can be clustered by "intelligent" high-speed networks that are getting closer to the custom interconnection systems of MPPs. Actually, many networks provide bandwidths that can not be completely used by current SMP PCs due to restrictions on the I/O bus to where the communication adapters are to be plugged. In clusters of SMP PCs, even the PCI bus appears as the bottleneck in the communication system.

According to our measurements, the barrier that keeps clusters of SMP PCs from joining the HPC universe is the memory bus. As soon as PCs with faster memory buses come to the market, a careful selection of components will certainly conduct to reasonably high performing machines at relatively low costs.

4.2. Software Aspects

The big challenge to support high performance computing in clusters of SMP PCs reside in the software, that does not evolve quickly enough to step along with the hardware. The main reason for this is the poor software design that does not promote encapsulation and thus makes system software updates a painful operation that usually propagates until the application software. Therefore, most cluster users prefer traditional system software to the detriment of performance. The penalty of adopting a standard, all-purpose operating system can be tolerated, in many cases, by distributed computing, but seldom by parallel computing.

Several important advances on removing software overhead from the way of applications to hardware have been achieved, specially with user level communication. However, we believe that a generic operating system can only represent a compromise to make resources available. It will not be able to deliver appropriate performance to specific classes of applications. In stead of a generic operating system, we propose the adoption of object orientation to make available, through inheritance and composition, a set of system software components that can be instantiated according to the requirements of applications.

We also believe that any attempt to remove software overhead, either by ”striping” conventional systems or by developing new alternatives, should be encapsulate into some standard user interface, allowing for system software revisions without impacting application software. A reasonable user interface could be composed by Posix file handling, standard floating point functions and MPI. Behind such interface, only the minimal components required by the application. A software package designed under this philosophy could drastically increase performance. Further information about this strategy can be found in [11].

5. Conclusion

In this paper we analyzed several aspects regarding the adoption of clusters of SMP PCs to support high performance computing. Although the interconnection systems analyzed deliver satisfactory performance, the currently available SMP PCs, mainly due to the memory bus, are behind the expectancies. The next generations of PCs are to solve this problem.

Overcoming the restrains imposed by the software usually adopted on clusters of SMP PCs, however, seems to be a harder challenge. Most of the software alternatives considered in this paper proved to be inadequate to support HPC, specially due to high overhead or to the lack of an standard user interface.

The authors proposal of a scalable object oriented set of system software components encapsulated by a standard user interface may help to diminish the gap between MPPs and clusters of SMP PCs. This a proposal is now under development at GMD-FIRST.

References

[1] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W.-K. Su. Myrinet: A Gigabit-per-Second Local-Area Network. *IEEE Micro*, 15(1):29–36, Feb. 1995.

[2] A. Chien, S. Pakin, M. Lauria, M. Buchanan, K. Hane, L. Giannini, and J. Prusakova. High Performance Virtual Machines (HPVM): Clusters with Supercomputing APIs and Performance. In *Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing*, 1997.

[3] D. E. Culler, A. Arpaci-Dusseau, R. Arpaci-Dusseau, B. Chun, S. Lumetta, A. Mainwaring, R. Martin, C. Yoshikawa, and F. Wong. Parallel Computing on the Berkeley NOW. In *Proceedings of the 9th Joint Symposium on Parallel Processing*, Kobe, Japan, 1997.

[4] D. E. Culler, R. M. Karp, D. Patterson, A. Sahay, E. E. Santos, K. E. Schauser, R. Subramonian, and T. von Eicken. LogP: A Practical Model of Parallel Computation. *Communications of the ACM*, 39(11), Nov. 1996.

[5] M. Eberl, H. Hellwagner, M. Schulz, and B. G. Herland. SISCI - Implementing a Standard Software Infrastructure on an SCI Cluster. In *Proceedings of the First German Workshop on Cluster Computing*, Chemnitz, Germany, Nov. 1997.

[6] D. R. Engler, M. F. Kaashoek, and J. O'Toole. Exokernel: An Operating System Architecture for Application-level Resource Management. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, pages 251–266, Copper Mountain Resort, U.S.A., Dec. 1995.

[7] M. Ibel, K. E. Schauser, C. J. Scheiman, and M. Weis. Implementing Active Messages and Split-C for SCI Clusters and Some Architectural Implications. In *Proceedings of the Sixth International Workshop on SCI-based Low-cost/High-performance Computing - SCIzzL-6*, Santa Clara, USA, Sept. 1996.

[8] S. S. Lumetta, A. M. Mainwaring, and D. E. Culler. Multi-Protocol Active Messages on a Cluster of SMP's. In *Proceedings of Supercomputing'97*, Sao Jose, USA, Nov. 1997.

[9] R. P. Martin, A. M. Vahdat, D. E. Culler, and T. E. Anderson. Effects of Communication Latency, Overhead, and Bandwidth in a Cluster Architecture. In *Proceedings of the 24th Annual International Symposium on Computer Architecture*, Denver, USA, June 1997.

[10] S. Pakin, V. Karamcheti, and A. A. Chien. Fast Messages: Efficient, Portable Communication for Workstation Clusters and Massively-Parallel Processors. *IEEE Concurrency*, 5(2), June 1997.

[11] W. Schrder-Preikschat. PEACE - A Software Backplane for Parallel Computing. *Parallel Computing*, 20(10):1471–1485, 1994.

[12] Y. Tanaka, M. Matsuda, M. Ando, K. Kubota, and M. Sato. COMPaS: A Pentium Pro PC-based SMP Cluster and its Experience. In *IPPS Workshop on Personal Computer Based Networks of Workstations'98*, pages 486–497, 1998.