Efficient Implementation of PVM on the AN2 ATM Network

Matthias Hausner[1], Michael Burrows[2], and Chandramohan A. Thekkath[2]

[1] Eidgenössische Technische Hochschule, Institut für Computersysteme, ETH Zentrum, CH-8092 Zurich***

[2] Digital Equipment Corporation, Systems Research Center, 130 Lytton Ave, Palo Alto, CA 94301. U.S.A.

Abstract. Advances in processor and network technology have increased the attractiveness of using a cluster of workstations connected by a high-speed network as a coarse-grained multicomputer. This paper describes our attempts at building such a system. Specifically, our system consists of a high-speed ATM network connecting a set of Alpha workstations that host the PVM environment. Using commercially available products, our system demonstrates end-to-end PVM communication performance that is very close to ATM link bandwidth.

1 Introduction

Technology trends make it increasingly attractive to connect a cluster of high-performance workstations with a high-speed network to be used as a loosely-coupled supercomputer. For example, the processors in current high-end workstations are competitive with the fastest available uniprocessors, and more than competitive with the processors found in tightly-coupled machines. Also, the new generation of local-area networks narrows the gap with the specialized interconnection networks found in more tightly-coupled machines.

Workstation clusters used for parallel scientific applications have traditionally relied on programming environments such as PVM [?]. Thus, efficient support of these environments is important if the potential of loosely-coupled supercomputing is to be fully realized. Efficient implementations of PVM on networks like Ethernet or FDDI have been difficult because the underlying communication medium provides very little support for such implementations. These networks provide no reliability guarantees and since scientific applications typically do not have mechanisms to deal with packet loss, PVM implementations have typically relied on heavyweight transports like TCP/IP for reliability. This heavily penalizes communication relative to computation, making all but the most coarse-grained applications unattractive as candidates for execution on PVM-based clusters. Modern switch-based ATM networks are capable of providing low error rates. Given such a network, it is possible to efficiently layer PVM with minimal software over the hardware without sacrificing reliability. Since the software overheads are low, the performance of the resulting

*** M. Hausner's current address is Oberon microsystems, Inc., Pfingstweidstrasse 30, CH-8005 Zurich. email: hausner@oberon.ch

implementation can be close to what the hardware is capable of supporting. Such performance improvements are likely to make PVM attractive to a wider variety of applications.

In this paper we describe the current status of a system that we are building. It consists of a switch-based ATM network connecting a set of workstations that host the PVM environment. The rest of the paper is structured as follows. We first describe the hardware environment that we operate in. Then, we detail some of the important properties of the switch and the host-network controller that our implementation exploits. Next, we describe details of the PVM implementation followed by performance results. The paper concludes with a summary of current status and future work.

2 Hardware Environment

2.1 The Network

AN2 is a switch-based ATM network developed at DEC SRC [?, ?]. In AN2, as in typical ATM networks, data is carried over SONET fiber optic links using 53 byte ATM cells. Hosts on the network that wish to communicate with each other establish connections called *virtual circuits* (VCs) as specified by the ATM protocols. Since a full description of the capabilities of the network can be found elsewhere [?], we describe only those features that are particularly useful for PVM.

A key feature of AN2 is that the switches implement a credit-based flow control mechanism. A downstream switch sends back credits to the upstream switch for each cell it forwards on a particular VC. The upstream switch will not forward cells for a VC unless it has received credits for cells forwarded previously. This credit-based scheme has the advantage that cells will not be dropped in the network due to congestion. In modern fiber optic LANs with low error links, the main source of data loss is buffer overflow during congestion. The AN2 flow-control mechanism is a guarantee against this type of loss; in fact, under normal operation the network will not lose cells. In addition, data on the AN2 is protected by end-to-end checksums, which help to detect errors. These three factors: low-error fiber links, end-to-end checksums, and hardware flow control dramatically reduce the probability of an undetected error on the AN2. For the moment, until experience indicates otherwise, we therefore feel justified in treating data loss within the workstation cluster as an extremely rare occurrence, and regard it as a catastrophic event.

2.2 The Host and Network Controller

We use DEC Alpha machines [?] connected to AN2 using the OTTO TurboChannel controller. The machines we use run at 175 MHz and have caches that are kept coherent in the presence of DMA activity from external devices such as the OTTO. The OTTO controller has some features that facilitates an efficient implementation of PVM. We give a simplified description of these below.

The host transmits and receives *packets* on the cell-based ATM network. These packets are suitably formatted, checksummed, fragmented, and reassembled by the controller according to standard ATM Adaptation Layer (AAL5) specifications. The controller and host communicate via several data structures that are shared between them. The three key data structures that concern us are the *receive ring*, the *report ring*, and *transmit descriptors.*

The receive ring is a circular list of *packet descriptors*, each of which consists of a list of *fragment descriptors.* Each fragment descriptor describes the memory layout of host buffer to hold a fragment of an incoming packet. When the OTTO receives a packet on a connection, it uses the incoming VC number in the cells of the packet to index into a ring of receive packet descriptors. These descriptors are pre-initialized by the host to point to buffers in the receiver process for that particular VC. The controller uses a free packet descriptor in the ring to initiate DMA from the network to the address space, thereby eliminating unnecessary copy overheads. Caches are kept consistent by the Alpha workstation hardware.

The behavior of the transmit descriptors is similar. When the host has a packet to send, it initializes a descriptor with the host memory address and length information for each of the packet fragments. Then, the host writes a controller register, causing the OTTO to access the descriptors and perform DMA directly between host memory and the network.

When the controller has transmitted or received a packet, it writes a status report into a ring maintained in host memory. Host software can examine the report and do the necessary bookkeeping such as freeing up the packet memory on transmit or notifying the receiver process about packet arrival. There is a single report ring that is shared by all VCs.

3 PVM Software Structure

Figure **??** shows the canonical structure of two PVM applications communicating using the traditional facilities of the PVM library. Our goal was to replace the connection between the application that used TCP to use the facilities of AN2 directly, as indicated in Figure **??**. In our current version, we did not give high priority to speeding up the path between the PVM daemons. In a local area workstation cluster, we believe the error guarantees provided by AN2 enable us to avoid using heavy-weight TCP connections. We will be evaluating this decision as we gather more data about observed error rates in our environment.

To support PVM efficiently, we made changes to the operating system as well as the implementation of the PVM library routines. We did not change the programmer-visible interface to PVM library routines or the PVM daemon. In the next section, we describe our changes to the operating system followed by our changes to the internals of the PVM library.

3.1 Operating System Support For PVM

The AN2 network routinely carries TCP and NFS (i.e., UDP) traffic in our laboratory. All access to the device is mediated by a standard OSF/1 device driver that manages the OTTO data structures described earlier. We made some initial measurements of this driver to determine whether we could layer PVM over it. While

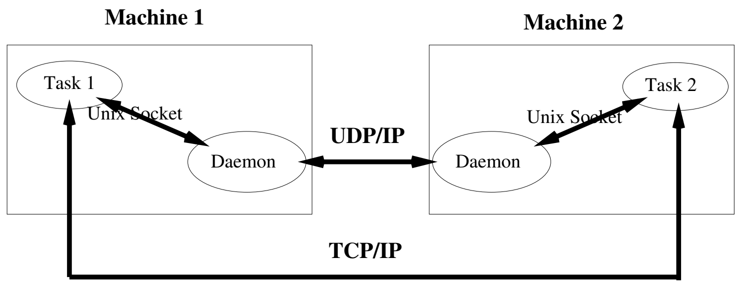

Fig. 1. Communication Structure of Traditional PVM Applications

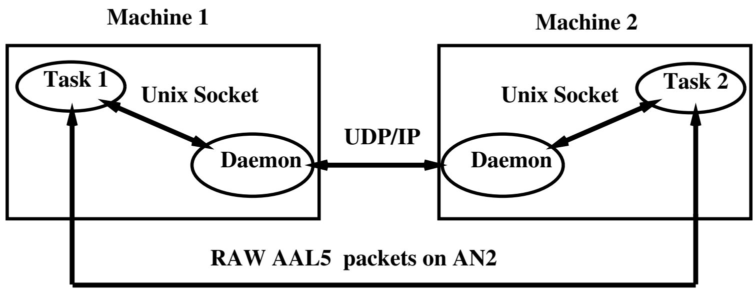

Fig. 2. Communication Structure of PVM Applications on AN2

we were happy with the performance, we felt that we could do better by devising a fast path through the driver without sacrificing compatibility with existing users of the driver, e.g., the internet protocols.

Ideally, we wanted each PVM application to be able to map the OTTO into its own address space and program the controller to transmit and receive data directly from its address space. This avoids all of the potentially expensive operating systems calls. The controller hardware does not support direct mapping of the device into multiple address spaces. However, with a small amount of software in the device driver we were able to achieve most of the benefits of direct mapping.

As described previously, the controller and the host share a region of memory that holds descriptors. This memory is common to all virtual circuits and thus to potentially many address spaces. When a PVM application starts up, the library requests the operating system device driver for a portion of descriptor memory, which it organizes into transmit and receive packet descriptors. The library can then directly initialize descriptors and prod the controller using programmed I/O to initiate the transfers. In our current implementation, the driver maps the entire memory into the application's address space. The application is given a unique offset identifying which portion of the memory it is allowed to use. However, the operating system does not enforce this restriction. Thus, a buggy (or malicious) application could potentially overwrite packet descriptors belonging to other applications. We are currently in the process of closing this protection loophole.

Receiving packets is a little more complicated. Recall that when the OTTO controller performs DMA using a receive descriptor, it writes a report into the report ring and interrupts the processor, which then executes the device driver code. How-