ParaStation User Level Communication

Joachim M. Blum and Thomas M. Warschko and Walter F. Tichy

Institut für Programmstrukturen und Datenorganisation, Fakultät für Informatik,
Am Fasanengarten 5, Universität Karlsruhe, D-76128 Karlsruhe, Germany

Summary. PULC (ParaStation User Level Communication) is a user-level communication library for workstation clusters. PULC provides a multi-user, multi-programming communication library for user-level communication on top of high-speed communication hardware. This paper describes the design of the communication subsystem, a first implementation on top of the ParaStation communication adapter, and benchmark results of this first implementation.

PULC removes the operating system from the communication path and offers a multi-process environment with user-space communication. Additionally, it moves some operating system functionality to the user-level to provide higher efficiency and flexibility. Message demultiplexing, protocol processing, hardware interfacing, and mutual exclusion of critical sections are all implemented in user-level. PULC offers the programmer multiple interfaces including TCP user-level sockets, MPI [CGH94], PVM [BDG+93], and Active Messages [CCHvE96]. Throughput and latency are close to the hardware performance (e.g., the TCP socket protocol has a latency of less than 9 \(\mu s\)).

Keywords: Workstation Cluster, Parallel and Distributed Computing, User-Level Communication, High-Speed Interconnects.

1. Introduction

Common network protocols are designed for general purpose communication in a LAN/ WAN environment. These protocols reside in the kernel of an operating system and are built to interact with diverse communication hardware. To handle this diversity, many standardised layers exist. Each layer offers an interface through which the other layers can access its services. This layered architecture is useful for supporting diverse hardware but leads to high and inefficient protocol stacks. Protocols which use standardised interfaces of the operating system are unaware of superior hardware functionality and often reimplement features in software even if the hardware already provides them. Another inefficiency is due to copy operations between kernel- and user-space and within the kernel itself. To transmit a message the kernel has to copy the data from or to user-space. The copying between protected address space boundaries often adds more latency than the physical transmission of a message. In addition, the kernel copies the data several times from one buffer to another while traversing layers of the protocol stack. On the positive side, the traditional communication path with the kernel as single point of access to the hardware ensures correct interaction with the hardware and mutual exclusion of competing processes.
For parallel computation on clusters of workstations, many of the protocols which are designed for wide area networks are too inefficient. Therefore, cluster computing must take new approaches.

The most promising technique is to move protocol processing to user-level. This technique opens up the opportunity to investigate optimised protocols for parallel processing. With user-level protocols there is no need to use the standardised interfaces between the operating system and the device driver. Thus, the reimplementation of services in software which are already provided by the hardware can be avoided.

![Diagram of user-level communication highway](image)

**Fig. 1.1.** User-level communication highway

User-level communication removes the kernel from the critical path of data transmission. Figure 1.1 shows how user-level communication shortcuts the access to the communication hardware. High-performance communication protocols are based on superior hardware features to speed up communication. Copying data between kernel- and user-space is avoided and the implementation of true zero-copy protocols is possible. These key issues minimise latency and lead to high throughput.

But user-level communication has also its drawbacks, because now the single point of access to the communication hardware, namely the kernel, is missing. Therefore many user-level communication libraries restrict the number of processes on a node to a single process. Enabling multiple processes on one node in user-level raises difficulties, but also offers a lot of benefits. Once problems, such as demultiplexing of messages and ensuring correct interaction between multiple processes are solved, the high-speed communication