A Communication Kernel for Parallel Programming Support on a Massively Parallel Processor System

Benno Overeinder[*] Joep Vesseur Frank v/d Linden Peter Sloot[*]
Parallel Scientific Computing & Simulation Group
University of Amsterdam

Keywords: parallel and distributed computing, run-time support systems

1 Introduction

The programming interface offered by portable parallel programming environments, such as PVM or MPI, creates a gap between the offered functionality in the parallel programming environment and the native parallel operating systems such as PARIX.

Because the needed functionality is of generic interest to multiple programming environments, we have designed and implemented a generic programming interface, the Communication Kernel, that efficiently supports well-known portable parallel programming environments.

In this paper we describe how the generic programming interface bridges the functionality gap between PARIX and any message passing interface. As a case study, we tested this Communication Kernel by implementing PVM on top of it.

2 Generic PVM versus Native PARIX

In this section we identify the difference in functionality of the Parallel Virtual Machine system (PVM) [1] and the PARIX (PARallel extensions to UnIX) operating system [3]. This difference stems from the fact that the PVM system was developed as a software framework for heterogeneous parallel computing in networked environments whereas PARIX was designed for tightly coupled parallel architectures.

2.1 Process Management

In PVM applications are viewed as comprising several sub-algorithms, each of which is potentially different in terms of it most appropriate programming model. Basically the Multiple Program – Multiple Data (MPMD) programming model applies to PVM. PARIX however, primarily supports the Single Program – Multiple Data (SPMD) programming model, although facilities are available for the MPMD paradigm.

The creation of a PVM task is accomplished with the `pvm_spawn` call. The spawned process starts an executable specified as an argument to the call. The node on which the task is started is specified by the user or is left to the PVM system for heuristic load balancing.

In the PARIX programming environment, program startup loads the same executable on all the nodes in the allocated partition of the MPP architecture. On the nodes itself, threads are the only active objects of an application. The only way of executing any processes in parallel on one processor is by means of threads.

[*] email: `{overeind, peterslo}@fwi.uva.nl` and
URL: `http://www.uva.nl/fwi/research/vg4/pwrs/`

2.2 Basic Message Passing

The PVM model assumes that any task can send a message to any other PVM task. The PVM inter-processor communication provides typed, asynchronous buffered communication primitives. These primitives are asynchronous blocking send, asynchronous blocking receive, and non-blocking receive varieties. In the PVM terminology, a blocking send returns as soon as the send buffer is free for reuse regardless of the state of the receiver. A non-blocking receive immediately returns with either the data or a flag indicating that the data has not arrived, while a blocking receive returns only when the data is in the receive buffer.

The basic communication primitive in PARIX is synchronous non-buffered communication. On top of this basic communication layer, PARIX provides asynchronous non-buffered communication and mailbox based communication (close to the PVM communication model).

2.3 Group Communication

The programmability of massively parallel computations strongly depends on global communication and synchronization primitives such as broadcast and barrier synchronization, respectively.

PVM supports the concept of user named groups, where a task can dynamically join or leave a group at any time. Various collective communication primitives are defined on groups such as broadcast, barrier, or the reduce operation.

No similar group constructs are available in PARIX.

3 The Communication Kernel

The Communication Kernel (CK) levels the functionality gap between portable, parallel programming environments such as PVM and MPI, and PARIX. In Fig. 1 a global overview of the layered design is shown.

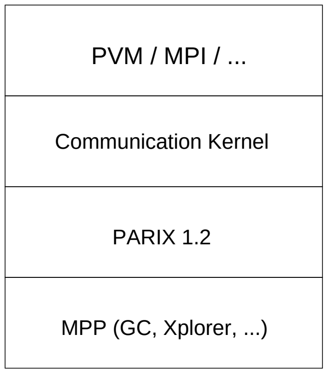

Figure 1: The Communication Kernel design overview.

With the design of the CK layer, the resulting message passing interface implementation intrinsics are clearly separated from the operating system dependent characteristics of PARIX. This improves the maintainability and portability to, for example, newer PVM or MPI versions.

3.1 Process Management

The PVM run-time support is composed of a set of system processes called pvm daemons (pvmd). A pvmd is installed on each node in the parallel virtual machine.

In the CK, the run-time support resides at each node in the MPP architecture and is composed of one run server at the “master node” and spawn servers at each node. The run server coordinates the process creation requests in a similar way as the pvmd does. It allows user processes to create (spawn) processes at specific nodes in the parallel system, or on a suitable node determined by a round-robin scheduling algorithm augmented with load information.

The task of the spawn server at each node is the actual creation of a new process (thread). After a request for process creation from the run server, a new thread is created that starts the requested executable.

3.2 Message Passing

One of the key issues to level the gap between the two environments is the efficient support of inter-process communication facility characterized by typed, buffered asynchronous communication. The most obvious alternative would be the use of mailbox based communication supported by PARIX. One serious disadvantage however is the communication performance.

Therefore, we have designed and incorporated a new generic asynchronous buffered communication in the CK. It uses the synchronous non-buffered communication primitives of PARIX as its transport layer, and implements both blocking and non-blocking communication for user-level processes.

3.3 Group Communication

In PVM a single task, the group server, is responsible for the dynamic group administration and for the coordination of barrier synchronizations. In the PVM design, the group server is a PVM task, just like any other task. In our implementation, the group server is also a task but as part of the run-time support system. This design allows a more efficient implementation of the group server that directly interacts with the Communication Kernel.

For collective communication we have implemented an optimized multicast with hypercube routing over the processor grid. The multicast messages are propagated to all the nodes in $\log_2(N)$ steps.

4 Results

The well-known “ping-pong” experiment is performed to measure the communication performance of the PVM implementation. In order to compare the PVM send and receive primitives with the underlying PARIX send and receive primitives, the PVM version of the ping-pong experiment is implemented similar to the PARIX version. By calling `pvm_setsbuf` the receive buffer becomes the new send buffer which is sent back immediately. Thus, the measurements do not include the unpacking and packing of data.

The overhead for PVM measured on the Parsytec PowerXplorer is shown in Table 1, which shows latencies for both PARIX 1.2 and the native PVM version, as well as the achieved throughput on both systems. In Fig. 2 a log-log plot depicts the communication performance of PVM versus PARIX.

	Latency μsec	Throughput (Kb/sec)
PARIX	91	1036
PVM	155	1033

Table 1: Communication latency and throughput measured on the PowerXplorer.

A more elaborate performance comparison between PVM and PARIX from an application perspective can be found in the paper by Hoekstra et al. [2].

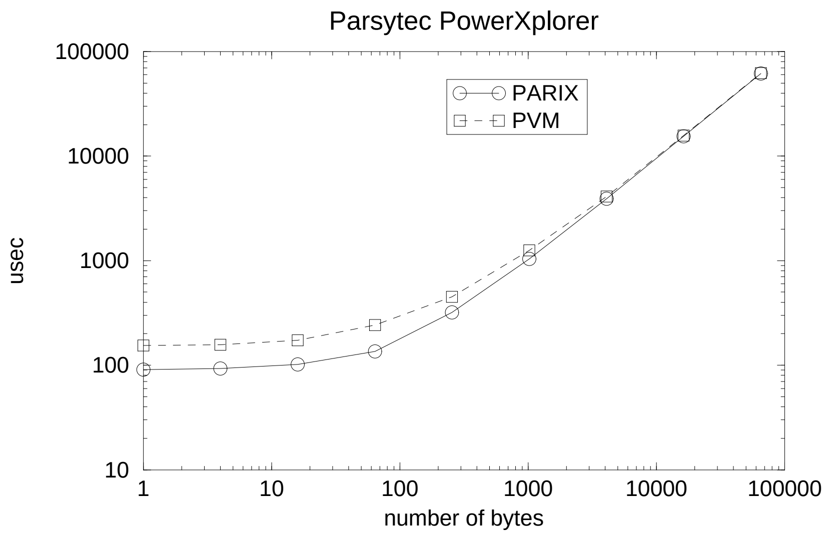

Figure 2: Communication performance of PVM versus PARIX on Parsytec PowerXplorer.

5 Conclusion

We have developed a generic Communication Kernel that supports popular portable parallel programming environments such as PVM and MPI, where our aim was to integrate the best of both worlds: the full functionality of PVM-like programming environments and the speed of native environments such as PARIX. The results clearly indicate the feasibility of our approach.

In the near future we will complete a full implementation of MPI on top of the CK layer.

References

[1] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, "PVM 3 user's guide and reference manual," Tech. Rep. ORNL/TM-12187, Oak Ridge National Laboratory, Oak Ridge, Tennessee, May 1994.

[2] A. G. Hoekstra, P. M. A. Sloot, F. van der Linden, M. van Muiswinkel, J. J. J. Vesseur, and L. O. Hertzberger, "Native and generic parallel programming environments on a Transputer and PowerPC platform," Accepted for publication in *Concurrency: Practice and Experience*.

[3] *PARIX 1.2 Documentation*. Parsytec GmbH.