

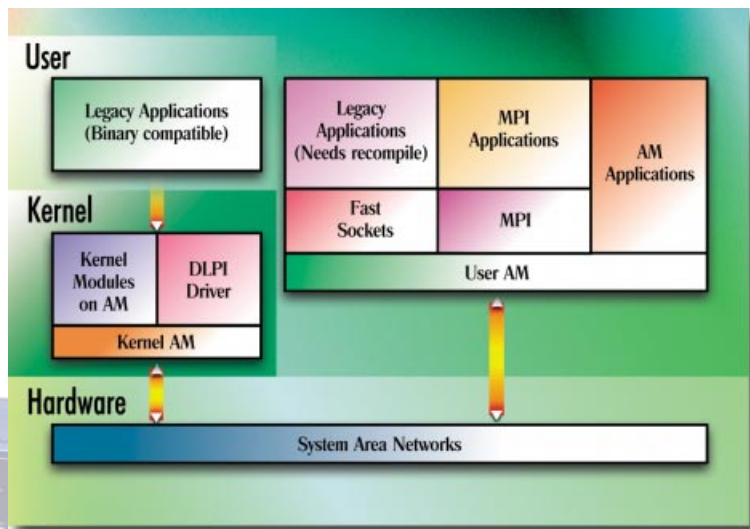
Scalable Communication Substrate for Cluster of Multi Processors

INTRODUCTION

KSHIPRA*, communication substrate designed to support low latency and high bandwidth is the key to the high level of aggregate system performance and scalability of C-DAC HPCC software. KSHIPRA caters to both parallel and distributed client server programming models.

The heart of KSHIPRA is the protected user level primitives - Active Messages (AM). It conforms to Active Messages II specifications given by University of California, Berkeley (UCB). Its implementation is an augmentation of the AM II software provided by UCB.

KSHIPRA Fast Sockets exports BSD sockets interface providing source code compatibility for legacy distributed / client server applications. Kernel Active Messages provide a high performance kernel to kernel, and kernel to user communication allowing kernel modules to leverage on AM technology. One such example is the Data Link Provider Interface (DLPI) driver which is layered over Kernel Active Messages. KSHIPRA thus provides binary compatibility to legacy distributed / client server applications. It also provides the Abstract Device Interface for layering Message Passing Interface (MPI) over AM.



KSHIPRA Communication Architecture

APPLICATION PROGRAMMING INTERFACES

KSHIPRA exports the following interfaces :

- ▶ Active Messages
- ▶ Fast Sockets
- ▶ DLPI
- ▶ MPI

HIGHLIGHTS

- Protected user level communication substrate with network virtualisation.
- Transparent support for multiple devices in the Cluster of Multi Processors.
- Active Message programming paradigm which exports the low latency and high bandwidth of the underlying system area networks.
- Fast Sockets interface for high performance distributed / client server programming.
- TCP/IP over Active Messages provides binary compatibility for legacy applications.
- MPI application programming interface for parallel programming.

*KSHIPRA (*kṣipra*) is a sanskrit word meaning *Fast and Alert*.

ACTIVE MESSAGES

Active Messages applies the RISC approach to communication architectures. It takes the entire system into consideration, from parallel programming paradigms down to the communication micro-architecture. The basic idea is to provide a small set of simple communication primitives, which are efficient and versatile.

The primitives are efficient in that they allow high-level programming paradigm to map well onto them and in-turn get mapped efficiently onto the processor and network hardware structures. The primitives are versatile in that they can be composed to implement variety of existing and emerging programming paradigms.

Programming Model

The Active Message interface allows applications to communicate among objects called communication endpoints. Endpoints are treated as single unit for communication, event

management and synchronisation. Any process can create multiple endpoints and gather related ones into bundles. Protection and message authentication is provided by per-endpoint message tags, which can be specified by the application. Traditional communication boundaries and domains allow interaction only between user-level parallel processes, whereas endpoints can communicate across these boundaries.

The key to both the versatility and efficiency of Active Messages lies in the ability to associate a small amount of user defined computation in the form of a handler with the reception of each message. The role of the message handler is to process the arriving packet as quickly as possible in order to allow an ongoing computation to continue. Handlers get the message out of the network and transfer the data directly into the application data structures. They may alternatively provide a small remote service and send a reply message back.

Multidevice Support

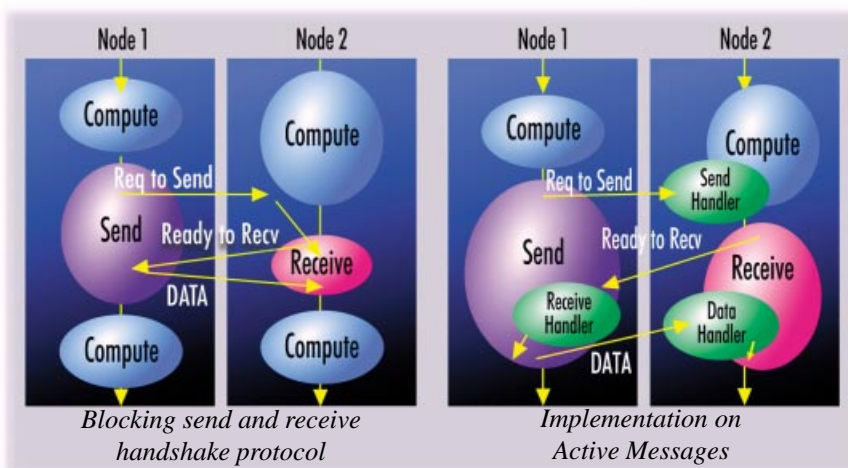
AM transparently directs messages through shared memory or the network depending on whether the destination endpoint is on local node or remote node. This allows parallel applications to fully exploit the Cluster of Multi Processors (CLUMPS) architecture by proper data and load partitioning.

Network Virtualization

A group of communicating endpoints form a virtual network with a unique protection domain. Traffic in one virtual network is never visible to a second virtual network, yet each virtual network retains the direct, user-level network access necessary for high performance. When distinct virtual networks share the same physical network resources, each continues to perceive private resources. This communication multiplexing is critical to high performance message passing with CLUMPS, since many processes are expected to be communicating at once.

Efficient Data Transfer

AM implementation leverages on a mechanism which bypasses operating system in the common case. This eliminates operating system overhead in frequent operations like send/receive and provides a communication abstraction which allows the users to exploit the underlying network to the fullest. Incoming messages are handled by explicit or implicit user-level polling and do not involve interrupts.



Why Active Messages ?

- ❑ **Designed for Clusters:** System Area Networks are capable of microsecond latencies and aggregate bandwidths of Gigabits per second with almost zero errors. However, traditional protocols like TCP/IP are designed for low bandwidth networks where error rates are high. AM is designed and implemented specifically for Cluster Communication over high speed system area networks.
- ❑ **Protected User Level Primitives :** AM bypasses operating system in the common case without compromising protection and network virtualization.
- ❑ **Success Oriented Protocols :** TCP/IP incurs high per packet processing cost to support end-to-end reliability. For example TCP uses an in-packet checksum, despite the presence of per-packet CRCs in most modern networks. AM is a *success oriented protocol* designed to exploit hardware features of highly reliable system area networks.
- ❑ **Reduced Stack :** Standard implementation of Sockets interface and the TCP/IP protocol suite separate the protocol and interface stack into multiple layers. AM integrates communication and programming interface to eliminate module-crossing costs.
- ❑ **Simple Buffer Management:** TCP/IP uses complicated buffer management schemes because of its multiple layers and implementation inside the kernel. AM uses simple yet efficient buffer management techniques, which reduces bookkeeping costs.
- ❑ **Decoupling Communication from Computation:** Traditional send and receive semantics do not allow communication to be overlapped with computation. In Active Messages handlers execute upon message arrival at destination, analogous to network interrupt.

Concurrency and Synchronization

Multiple threads may concurrently send requests and replies from an endpoint in a shared bundle and concurrently poll a shared bundle to handle individual incoming messages in parallel. While attempting to send a message the AM system can receive messages and dispatch their handlers.

The Active Message handlers provide very flexible means of synchronizing communication and computation in that a handler function enables arbitrary interaction between communicators and the ongoing computation.

Error Handling

Active messages guarantee reliable (at-most-once) message delivery. In the unlikely event that a message could not be delivered, it is returned to the application by invoking the per-endpoint error handler function so that the application may take appropriate recovery actions.

FAST SOCKETS

Fast sockets are a user-level sockets library layered over Active Messages. It allows legacy programs to leverage on low latency and high bandwidth offered by AM technology. Fast Sockets interface conforms to standard socket interface and hence provides source code compatibility to legacy applications. These applications need no code changes; only a relink to the Fast Sockets library. The implementation is based on the Fast Sockets implementation of the University of California, Berkeley.

KERNEL ACTIVE MESSAGES

Kernel Active Message (KAM) exports complete AM API to kernel modules. KAM can be used to develop kernel modules such as distributed virtual memory systems, file systems, or page caches. KAM gives KSHIPRA an extraordinary programming flexibility, which no other traditional communication model provides. It allows kernel endpoints to communicate with user application endpoints. KSHIPRA thus forms the basic building block for cluster system software.

DLPI

KSHIPRA makes novel use of KAM to export IP interface. It implements a Solaris 2.x DLPI compliant network driver layered over KAM. This driver looks like a standard network device to higher levels of the TCP/IP protocol stack but uses KAM transport operations to transport data. This allows the standard protocol stacks and legacy applications to use Active Messages transparently (providing binary compatibility) to communicate between hosts on the fast system area networks.

MPI

KSHIPRA caters to the needs of parallel computing community by providing an implementation of the Abstract Device Interface of MPI over the AM interface. The implementation effectively reflects the low-latency and high-bandwidth properties of AM to the MPI applications.

AVAILABILITY

Supported Hardware	:	Workstation Clusters
Supported Operating System	:	Solaris 2.5 and above
User Interfaces	:	Command Line
Supported Languages	:	C
Prerequisite Hardware	:	ParamNet, Myrinet
Supported Application Programme Interfaces	:	Active Messages, Fast Sockets, DLPI, MPI



A Scientific Society of the
Department of Electronics
Government of India

Additional Information

For more information on CDAC HPCC software, contact your CDAC marketing representative, access the CDAC Home Page on the internet World-Wide Web (www.cdac.org.in), or send an e-mail over the internet to : ssg@cdacb.ernet.in

C-DAC reserves the right to change or modify any of the product or service specifications or features described herein without notice. The product summary is for information only. C-DAC makes no express or implied representations or warranties in this summary

*All trademarks and brand names are owned by their respective owners.

Headquarters

University of Poona Campus,
Ganesh Khind, Pune - 411 007, INDIA
Tel : 352461 Fax : 91-212-357551
Tlx : 0145-7615 CDAC IN
email : business@cdac.ernet.in

Business Division

Ramanashree Plaza, 2/1 Brunton Road,
Bangalore - 560 025, INDIA
Tel : 5584271 Fax : 91-80-5584893
Tlx : 0845-8413 CDAC IN,
email : bdm@cdacb.ernet.in

Delhi Centre

E-13, 2nd Floor, Hauz Khas,
New Delhi - 110 016, INDIA
Tel : 6863428 Fax : 91-11-6863428