Parallel Computing on the Berkeley NOW

David E. Culler, Andrea Arpaci-Dusseau, Remzi Arpaci-Dusseau, Brent Chun,
Steven Lumetta, Alan Mainwaring, Richard Martin, Chad Yoshikawa, Frederick Wong

Computer Science Division
University of California, Berkeley

Abstract: The UC Berkeley Network of Workstations (NOW) project demonstrates a new approach to large-scale system design enabled by technology advances that provide inexpensive, low latency, high bandwidth, scalable interconnection networks. This paper provides an overview of the hardware and software architecture of NOW and reports on the performance obtained at each layer of the system: Active Messages, MPI message passing, and benchmark parallel applications.

1 Introduction

In the early 1990's it was often said that the "Killer Micro" had attacked the supercomputer market, much as it had the minicomputer and mainframe markets earlier. This attack came in the form of massively parallel processors (MPPs) which repackaged the single-chip microprocessor, cache, DRAM, and system chip-set of workstations and PCs in a dense configuration to construct very large parallel computing systems. However, another technological revolution was brewing in these MPP systems – the single-chip switch – which enabled building inexpensive, low latency, high bandwidth, scalable interconnection networks. As with other important technologies, this "killer switch" has taken on a role far beyond its initial conception. Emerging from the esoteric confines of MPP backplanes, it has become available in a form that is readily deployed with commodity workstations and PCs. This switch is the basis for *system area networks*, which have performance and scalability of the MPP interconnects and the flexibility of a local area network, but operate on a somewhat restricted physical scale.

The Berkeley NOW project seeks to demonstrate that it is viable to build large parallel computing systems that are fast, inexpensive, and highly available, by simply snapping these switches together with the latest commodity components. Such cost-effective, incrementally scalable systems provide a basis for traditional parallel computing, but also for novel applications, such as internet services[Brew96].

This paper provides an overview of the Berkeley NOW as a parallel computing system. Section 2 gives a description of the NOW hardware configuration and its layered software architecture. In the following sections, the layers are described from the bottom-up. Section 3 describes the Active Message layer and compares its performance to what has been achieved on MPPs. Section 4 shows the performance achieved through MPI, built on top of Active Messages. Section 5 illustrates the application performance of NOW using the NAS Parallel Benchmarks in MPI. Section 6 provides a more detailed discussion of the world's leading disk-to-disk sort, which brings out a very important property of this class of system: the ability to concurrently perform I/O to disks on every node.

2 Berkeley NOW System

The hardware configuration of the Berkeley NOW system consists of one hundred and five Sun Ultra 170 workstations, connected by a large Myricom network[Bode95], and packaged into 19-inch racks. Each workstation contains a 167 MHz Ultra1 microprocessor with 512 KB level-2 cache, 128 MB of memory, two 2.3 GB disks, ethernet, and a Myricom "Lanai" network interface card (NIC) on the SBus. The NIC has a 37.5 MHz embedded processor and three DMA engines, which compete for bandwidth to 256 KB of embedded SRAM. The node architecture is shown in Figure 1.

The network uses multiple stages of Myricom switches, each with eight 160 MB/s bidirectional ports, in a variant of a fat-tree topology.

2.1 Packaging

We encountered a number of interesting engineering issues in assembling a cluster of this size that are not so apparent in smaller clusters, such as our earlier 32-node prototype. This rack-and-stack style of packaging is extremely scalable, both in the number of nodes and the ability to upgrade nodes over time. However, structured cable management is critical. In tightly packaged systems the interconnect is hidden in the center of the

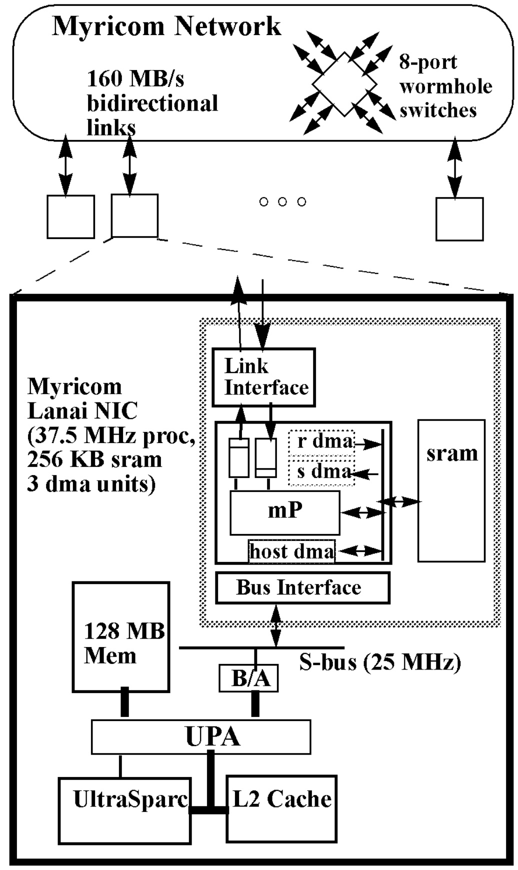

Figure 1. NOW Node Configuration

machine. When multiple systems are placed in a machine room, all the interconnect is hidden under the floor in an indecipherable mess. However, in clusters, the interconnect is a clearly exposed part of the design. (a bit like the service conduits in deconstructionist buildings). Having the interconnect exposed is valuable for working on the system, but it must stay orderly and well structured, or it becomes both unsightly and difficult to manage.

The Berkeley NOW has four distinct interconnection networks. First, the Myrinet provides high-speed communication within the cluster. We discuss this in detail below. Second, switched-Ethernet into an ATM backbone provides scalable external access to the cluster. The need for an external network that scales with the size of the cluster was not apparent when we began the project, but the traffic between the cluster and other servers, especially file servers, is an important design consideration. Third, a terminal concentrator provides direct console access to all the nodes via the serial port. This is needed only in situations when the node cannot be rebooted through the network, or during system development and debugging. Fourth, conventional AC lines provide a power distribution network. As clusters transition to the commercial mainstream, one engineering element will be to consolidate these layers of interconnect into a clean modular design. Figure 2 shows a picture of the NOW system.

Figure 2. NOW System

2.2 Network topology

The Myrinet switches that form the high-speed interconnect use source routing and can be configured in arbitrary topologies. The NOW automatic mapping software can handle arbitrary interconnect[Mai*97]; however, we wire the machine as a variant of a Fat-tree to create a system with more uniform bandwidth between nodes, thereby minimizing the impact of process placement. The topology is constrained by the use of 8-port (bidirectional) switches and wiring density concerns. Initially we planned to run cables from all the nodes to a central rack of switches; however, the cable cross-sectional area near the switches became unmanageable as a result of bulky, heavily-shielded copper network cables. Using fiber-optic cables that are now available, the cable density may be reduced enough to centrally locate the switches.

In building an indirect network out of fixed-degree switches, the number of upward links depends on the number of downward links. We elected to attach five hosts to each first level switch, which eliminates 40% of the cable mass. As shown in Figure 3, groups of seven of these switches are treated as a 35-node subcluster with the 21 upward links spread over four level-two switches. Three of these subclusters are wired together to comprise the NOW. We have found that as a rule of thumb, adding 10% extra nodes and extra ports greatly simplifies system administration, allowing for node failures, software or hardware upgrades, and system expansion.

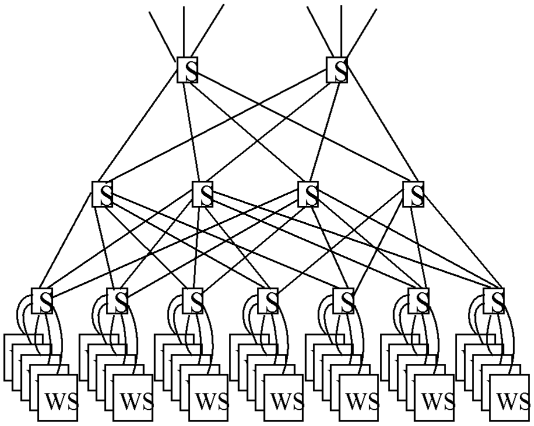

Figure 3. NOW Myrinet Network Topology

2.3 Software Architecture

The system software for NOW employs a layered architecture, as illustrated in Figure 4. Each node runs a complete, independent Solaris Unix with the associated process management, memory management, file system, thread support, scheduler, and device drivers. We extend Solaris at each of these interfaces to support global operations over the NOW.

Process Management: A global OS layer, called GLUnix, provides NOW-wide process management as a layer on top of Solaris (via sockets, daemons, and signals). Using either a global shell, `glush`, or the `glurun` command, sequential processes can be started anywhere on the NOW or parallel processes can be started on multiple nodes. Local `pids` are elevated to a global `pids`, and the familiar process control operations, such as ctrl-C or ctrl-Z, work on global processes. The Unix process information and control utilities, such as `ps` and `kill`, are globalized as well.

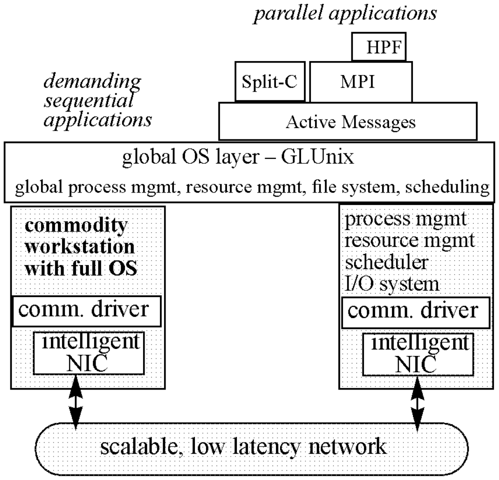

Figure 4. NOW software architecture

File System: A prototype file system, xFS, extends Solaris at the vnode interface to provide a global, high performance file system[And*95b]. Files are striped over nodes in a RAID-like fashion so that each node can read file data at the bandwidth of its interface into the network. The aggregate bandwidth available to nodes is that of all the disks. xFS uses a log-structured approach, much like Zebra[HaOu95], to minimize the cost of parity calculations. A single node accumulates enough of the log so that it can write a block to each disk in a stripe group. Before writing the blocks, it calculates a parity block locally and then writes it along with the data blocks.

An update-based file cache-coherence strategy is used, and the caches are managed cooperatively to increase the population of blocks covered by the collection of nodal caches. If a block about to be discarded is the last copy in the system, then it is cast off to a random remote node. Nodes take mercy on this block until it has aged to the point where it appears pointless to keep it in memory. This policy has the attractive property that actively used nodes behave like traditional clients while idle nodes behave like servers, so the cooperative file cache adapts dynamically to system usage.

Virtual Memory: Two prototype global virtual memory systems have been developed to allow sequential processes to page to the memory of remote idle nodes, since communication within the NOW has higher bandwidth, and much lower latency than access to local disks. One of these uses a custom Solaris segment driver to implement an external user-level pager which exchanges pages with remote page daemons. The other provides similar operation on specially mapped regions using only signals.

3 Active Messages

Active Messages are the basic communication primitives in NOW. This work continues our investigation of implementation trade-offs for fast communication layers[vE92*,Gol*96,Kri*96] and on NOW we have sought to generalize the approach and take full advantage of the complete OS on every node. The segment driver and device driver interface is used to provide applications with direct, protected user-level access to the network. Active Messages map to simple operations on queues and buffers that are shared between the user process and the communication firmware, which is executed on a dedicated processor embedded in the network interface card.

We have built two Active Message layers. The first, Generic Active Messages (gam) is oriented toward the traditional single-parallel-program-at-a-time style of parallel machines, and provides exactly the same API across a wide range of platforms[Cul*95].This serves as a valuable basis for comparison.

The newer AM layer[Main95], AM-II, provides a much more general purpose communication environment, which allows many simultaneous parallel programs, as well as client/server and system use. It is closely integrated with POSIX threads. The AM implementation is extremely versatile. It provides error detection and retry a the NIC-to-NIC level and allows the network to be reconfigured in a running system. A privileged mapper daemon explores the physical interconnection, derives deadlock-free routes, and distributes routes periodically[Mai*97]. AM-II provides a clean return-to-sender error model to support highly available applications.

The Active Messages communication model is essentially a simplified remote procedure call that can be implemented efficiently on a wide range of hardware. Three classes of messages are supported. Short messages pass eight 32-bit arguments to a handler on a destination node, which executes with the message data as arguments. Medium messages treat one of the arguments as a pointer to a 128 byte to 8 KB data buffer and invoke the handler with a pointer to a temporary data buffer at the destination. Bulk messages perform a memory-to-memory copy before invoking the handler. A request handler issues replies to the source node.

We have developed a microbenchmarking tool to characterize empirically the performance of Active Messages in terms of the LogP model[Cul*93, Cul*95]. Figure 5 compares the gam short message LogP parameters on NOW with the best implementations on a range of parallel machines. The bars on the left show the one-way message time broken down into three components: send overhead (o_s), receive overhead (o_r), and the remaining latency (L). The bars on the right shows the time per message (*g = 1/MessageRate)* for a sequence of messages. NOW obtains competitive or superior communication performance to the more tightly integrated, albeit older, designs.

The overhead on NOW is dominated by the time to write and read data across the I/O bus. The Paragon has a dedicated message processor and network interface on the memory bus; however, there is considerable overhead in the processor-to-processor transfer due to the cache coherence protocol and the latency is large because the message processors must write the data to the NI and read it from the NI. The actual time on the wire is quite small. The Meiko has a dedicated message processor on the memory bus with a direct connection to the network, but the overhead is dominated by the exchange instruction that queues a message descriptor for the message processor and the latency is dominated by the slow message processor accessing the data from host memory. Medium and bulk messages achieve 38 MB/s on NOW, limited primarily by the SBus.

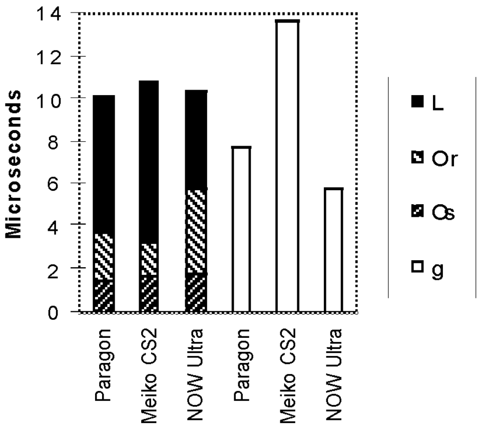

Figure 5. Active Messages LogP Performance

Traditional communication APIs and programming models are built upon the Active Message layer. We have built a version of the MPI message passing standard for parallel programs in this fashion, as well as a version of the Berkeley Sockets API, called Fast Sockets[Rod*97]. A shared address space parallel C, called Split-C[Cul*93], compiles directly to Active Messages, whereas HPF[PGI] compiles down to the MPI layer.

4 MPI

Our implementation of MPI is based on the MPICH reference implementation, but realizes the abstract device interface (ADI) through Active Message operations. This approach achieves good performance and yet is portable across Active Message platforms. The MPI communicator and related information occupy a full short message. Thus, a zero-byte control message is implemented as a single small-message request-response, with the handler performing the match operation against a receive table. The one-way time for an echo test is 15 µs. MPI messages of less than 8 KB use an adaptive protocol implemented with medium Active Messages. Each node maintains a temporary input buffer for each sender and senders keep track of whether their buffers are available on the destination nodes. If the buffer is available, the send issues the data without handshaking. Buffer availability is conveyed back to the source through the response, if the match succeeds, or via a request issued by the later matching receive. Large messages perform a handshake to do the tag match and convey the destination address to the source. A bulk operation moves the message data directly into the user buffer.

Figure 6 shows the bandwidth obtained as a function of message size using Dongarra's echo test on NOW and on recent MPP platforms[DoDu95]. The NOW version has lower start-up cost than the other distributed memory platforms and has intermediate peak bandwidth. The T3D/pvm version does well for small messages, but has trouble with cache effects. Newer MPI implementations on the T3D should perform better than the T3D/pvm in the figure, but data is not available in the Dongarra report.

5 NAS Parallel Benchmarks

An application-level comparison of NOW with recent parallel machines on traditional scientific codes can be obtained with the NAS MPI-based parallel benchmarks in the NPB2 suite[NPB]. We report briefly on two applications. The LU benchmark solves a finite difference discretization of the 3-D compressible Navier-Stokes equations. A 2-D partitioning of the 3-D data grid onto a power-of-two number of processors is obtained by halving the grid repeatedly in the first two dimensions, alternating between *x* and *y*, resulting in vertical pencil-like grid partitions. The ordering of point based operations constituting the SSOR procedure proceeds on diagonals which progressively sweep from one corner on a given z plane to the opposite corner of the same *z* plane, thereupon proceeding to the next *z* plane. This constitutes a diagonal pipelining method and is called a "wavefront" method by its authors [Bar*93]. The software pipeline spends relatively little time filling and emptying and is perfectly load-balanced. Communication of partition boundary data occurs after completion of computation on all diagonals that contact an adjacent partition.

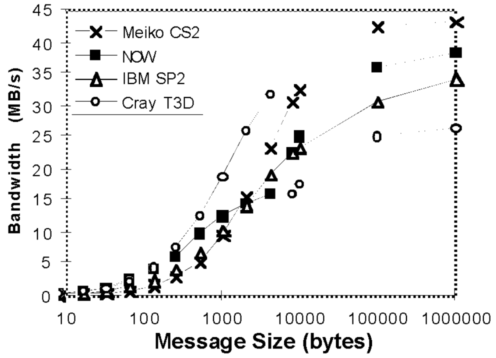

Figure 6. MPI bandwidth

The BT algorithm solves three sets of uncoupled systems of equations, first in the *x*, then in the *y*, and finally in the *z* direction. These systems are block tridiagonal with 5x5 blocks and are solved using a multi-partition scheme[Bru88]. The multi-partition approach provides good load-balance and uses coarse-grained communication. Each processor is responsible for several disjoint sub-blocks of points ("cells") in the grid. The cells are arranged such that for each direction in the line-solve phase, the cells belonging to a certain processor are evenly distributed along the direction of solution. This allows each processor to perform useful work throughout a line-solve, instead of being forced to wait for the partial solution to a line from another processor before beginning work. Additionally, the information from a cell is not sent to the next processor until all sections of linear equation systems handled in this cell have been solved. Therefore the granularity of communications is kept large and fewer messages are sent. The BT code requires a square number of processors.

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS

Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS

Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS

Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | **sales@docketalarm.com** | **1-866-77-FASTCASE**