
Distributed Cluster Computing Environments

Mark A. Baker and Geoffrey C. Fox

Northeast Parallel Architectures Center
111 College Place
Syracuse University
New York 13244-4100
USA

Email: mab@npac.syr.edu
Tel: (315) 443 2083
Fax: (315) 443 1973

21st January 1996

Version 1.2

Chapter 1

1. Distributed Cluster Computing Environments (DCCE)

1.1 Introduction

This report aims to catalogue and briefly describe Distributed Cluster Computing Environments (DCCE) as well as newly established Cluster Management Software (CMS) projects. The document started out as part of a review commissioned by the UK government on Cluster Computing [1], and followed two other reports: a historic review of Cluster Computing produced by the Manchester Computing Centre [2], and a critical review of Cluster Computing projects funded by the NTSC [3] and produced by the University of Southampton.

Whilst gathering the information to produce the Cluster Computing Review it became evident that software for cluster computing fell into one of two camps:

1. Cluster Management Software (CMS) - this software is designed to administer and manage application jobs submitted to workstation clusters. It encompasses the traditional batch and queuing systems.

2. Distributed Cluster Computing Environments (DCCE) - with this software the cluster is typically used as an applications environment, similar in many ways to distributed shared memory systems. This chapter also includes descriptions of environments which are at an early stage of development stage.

This document deals with 2), the CMS is now reviewed in [4].

1.2 Organisation of this Document

This report is organised as follows:

Chapter 1 - introduces and then briefly discusses the scope and range of the review.

Chapter 2 - provides a description each of the cluster software packages.

Chapter 3 - has a comprehensive glossary of the terms used throughout report.

Chapter 4 - includes a list of references and a table of contents.

1.3 Some Words About Cluster Computing Environments

So called cluster computing systems and the means of managing and scheduling applications to run on these systems are becoming increasingly common place.

Software to manage and run applications on clusters of workstations comes in one of two basic forms:

- As a software layer positioned between the native operating system and user applications - CMS packages are typical examples of this type of approach.
- As a partial or complete replacement for the native operating systems - this approach encompasses a much wider range of software packages from micro-kernels and kernel enhancements through to implementations of virtual shared memory and active messages.

1.4 Clusters of Workstations: The Ownership Hurdle.

Generally a workstation will be "owned" by, for example, an individual, a groups, a department, or an organisation. They are dedicated to the exclusive use by the "owners". This ownership often brings problems when attempting to form a cluster of workstations. Typically, there are three types of "owners":

- Ones who use their workstations for sending and receiving mail or preparing papers, such as administrative staff, librarian, theoreticians, etc.
- Ones involved in software development, where the usage of the workstation revolves around the edit, compile, debug and test cycle.
- Ones involved with running large numbers of simulations often requiring powerful computational resources.

It is the latter type of "owner" that needs additional compute resources and it is possible to fulfill their needs by fully utilising spare CPU cycles from former two "owners". However, this may be easier said than done and often requires delicate negotiation to become reality.

Chapter 2

2. Distributed Cluster Computing Environments

<i>Ref No.</i>	<i>Package Name</i>	<i>Vendor</i>
1.	Amoeba	Vrije Universiteit, Amsterdam
2.	Beowulf	NASA Goddard Space Flight Center, USA
3.	BSP	University of Oxford, UK
4.	Calypso	New York University, USA
5.	Coherent Virtual Machine (CVM)	University of Maryland, USA
6.	Compiler Technology	University of Rochester, USA
7.	Cthreads	Georgia Tech, USA
8.	Data Diffusion Machine	Partnership in Advanced Computing Technologies, UK
9.	DCE	Open Software Foundation, USA
10.	DOME	Carnegie Mellon, USA
11.	Flash	Stanford University, USA
12.	GLU (Granular Lucid)	SRI International, USA
13.	LAM - Local Area Multicomputer	Ohio Supercomputer Center, USA
14.	Legion	Virginia, USA
15.	Linda	Yale University, USA
16.	Locust	State University of New York, USA
17.	MPI	Argonne National Laboratory, USA
18.	Mirage+	University of California, USA

19.	Networks Of Workstations (NOW)	Berkeley, USA
20.	PAPERS	Purdue University, USA
20.	Parallel Virtual Machine (PVM)	ORNL, USA
22.	The SHRIMP Project	Princeton University, USA
23.	THESIS	MCC, UK
24.	TreadMarks	Rice University, Texas, USA
25.	WANE	Florida State, USA
26.	Wisconsin Wind Tunnel Project - Tempest	Wisconsin, USA

Introduction

This chapter provides a brief description of each of the packages listed in the table above. These descriptions are based on information that has been coalesced from user guides and on-line (WWW) documents.

The majority of the software packages described here are ones that can be considered as DCCE, however, it also contains descriptions of some CMS packages which are at the early stages of development (see NOW and WANE) as well as other parallel environments such as the Parallel Virtual Machine (PVM), MPI and Linda.

2.1 Amoeba

URL <http://www.cs.vu.nl/vakgroepen/cs/amoeba.html>

Amoeba is a general purpose distributed operating system developed and supported by Vrije Universiteit in Amsterdam [5]. Amoeba is designed to take a collection of machines and make them act together as a single integrated system. In general users are unaware of the number and location of the processors that run their commands, nor of the number and location of the file servers that store their files.

Amoeba is an on-going research project and should be regarded as a platform for doing research and development in distributed and parallel systems, languages, protocols and applications. Amoeba provides some Unix emulation and has a Unix-like feel, but it is not a Unix replacement.

Amoeba was designed around a micro-kernel architecture - here each machine runs a kernel that supports the basic process, communications and objective primitives. It also handles raw I/O and memory management. Everything else is built on top of these fundamentals.

A typical Amoeba system will consist of three fundamental classes of machines. First, each user has a workstation for running the user interface, based on the X Windows system. Second, there exists a pool

of processors that are dynamically allocated to users as required. Third, there are specialised servers, such as file servers and directory servers that run all the time. All these components are connected via a fast LAN - at present only Ethernet is supported.

2.2 Beowulf

URL <http://cesdis.gsfc.nasa.gov/pub/people/becker/beowulf.html>

Beowulf [6] is a project to produce the software to control workstations based on commodity PC-class hardware and the Linux operating system. Beowulf couples the low cost, moderate performance of commodity PC subsystems with the emergence of *de facto* standards in message passing hardware and software, utilising local file storage capacity and bandwidth. This experimental system is driven by the requirements of NASA Earth and Space science applications including data assimilation, data set browsing and visualisation, and simulation of natural physical systems.

The Beowulf parallel workstation architecture comprises PC processor subsystems, each with a disk, controller and 16 Mbytes of DRAM. The processing elements are interconnected by two parallel Ethernet networks with a peak capacity of 10 Mbps per network. Two of the PCs have separate Ethernet interfaces to external LAN's for remote access and data transfer. Two additional subsystems have high resolution video controller/drivers, one of these also provides a keyboard and mouse interface for conventional single-user workstation access.

Most of the currently running parallel applications are written using PVM to communicate, others use RPC. Future work is expected to include:

- MPI support.
- Implementing a Network Virtual Memory system - like distributed shared memory.
- A distributed I/O file server.

2.3 The Oxford BSP Library

URL <http://www.comlab.ox.ac.uk/oucl/oxpara/bsplib1.htm>

The Oxford BSP library provides a mechanism for developing portable and predictably-efficient code across a wide range of distributed platforms. The programming model is based on the Bulk Synchronous Parallel (BSP) architectural model [7 & 8]. This abstract general model parameterises system characteristics in order to realise performance across distributed platforms without tying application code to a specific architecture. Predicting performance and determining optimal parallel strategies for applications are based on a corresponding cost model.

The library [9] consists of a small number of subroutines to implement process creation, remote data access, and bulk synchronisation. The library can be linked to programs written in standard sequential languages such as Fortran, C, and Pascal.

PORTABLE INTERFACE - SINGLE-SOURCE APPLICATION CODE

The library offers the same interface and semantics to all applications. The specification of the library

routines is portable, but their implementation is not necessarily so. Machine-specific instances of the library can be created to utilise the disparate facilities for parallelism native to each environment. Thus, the library is tuned for a machine rather than individually for the program, thereby achieving single-source application codes.

The BSP library embodies a static SPMD (Single Program, Multiple Data) programming model, in which every parallel process executes the same program text. Processes proceed together through a sequence of supersteps, all processes must reach the end of the superstep before anyone can proceed to the next. Each process has its own private data space; there is no globally shared data, but a process can explicitly access the variables of any other process. Remote data access is asynchronous; but is not guaranteed to be satisfied until the end of the current superstep, when all the processes are synchronised.

Predicting performance of programs is based on system parameters which characterise processor speed, bulk-synchronisation time and global communication bandwidth. The first prototype of the library was built to use either the PVM or PARMACS message-passing library, this implementation was portable but not particularly efficient. Higher-performance generic versions have been implemented using TCP/IP sockets (for workstation clusters), and System V shared-memory primitives (for shared-memory multiprocessors). For the highest performance, there are native library versions for specific machines such as the IBM SP1/SP2, SGI Power Challenge, Meiko CS-2 and Cray T3D.

2.4 Calypso

URL <ftp://cs.nyu.edu/pub/calypso/bdk95a.ps>

Calypso [10] is a prototype software system for writing and executing parallel programs on clusters of workstations using the generic operating systems and compilers. Calypso uses a novel architecture to provide fault tolerance, automatic load balancing and shared address space whilst incurring low runtime overheads for coarse-grained computations.

Calypso version 0.9 has been implemented in C++ using AT&T's CC compiler and at present runs under SunOS 4.1.3. A Calypso computation is based on a master process and a dynamically changing set of worker processes. Features of Calypso are:

PROGRAMMING: Programs are written in a language called Calypso Source Language (CSL). This is essentially C++ with additional constructs to express parallelism and is based on a shared memory model.

MAPPING: Programs are logical entities and as such the degree of parallelism expressed by the programmer is independent of the parallelism provided by the execution environment, for example the number of workstation available.

FAULT RESILIENCE: Calypso executions are fault resilient. Worker processes can fail and possibly recover at any point without affecting the final computation.

DYNAMIC LOAD BALANCING: Calypso automatically distributes work-load depending on the dynamics of the participating workstations.

2.5 Coherent Virtual Machine (CVM)

URL <http://www.cs.umd.edu/projects/cvm/>

The Coherent Virtual Machine (CVM) software Distributed Shared Memory (DSM) system project started in the Autumn of 1995. The projects goals have not yet been fully determined, but tentatively will include:

- Fault-tolerance - CVM will have a limited amount of fault tolerance, allowing it to recover from at least one node failure at a time.
- Multiple protocol support - CVM will provide at least four memory models in its initial configuration, single and multiple versions of lazy release consistency, sequential consistency, and eager release consistency.
- Extensibility - CVM will be extensible in that the source will be freely available, and the modules are being written in C++. New classes can easily be derived from a master protocol class, allowing new protocols to be easily incorporated.
- Multi-threading support - CVM will be multi-threaded, allowing for the overlap of computation and communication through context switching. Additionally, the multi-threading will efficiently support multiprocessor nodes.

2.6 Compiling for Distributed Shared-Memory Machines

URL <http://www.cs.rochester.edu/u/wei/dsm.html>

Existing work in parallelising compilers falls into two principal categories: compiling for uniform shared memory machines and compiling for distributed memory message-passing machines. Little work has addressed compiler techniques for distributed shared memory machines. The goal of this project is to develop a parallelising compiler for distributed shared memory systems.

2.7 Cthreads

URL <http://www.cc.gatech.edu/systems/projects/Cthreads/>

Cthreads [11] is a user-level threads package that runs on uni-processor and multi-processor machines. It offers support for shared-memory parallel programming. The uni-processor support is intended for development, debugging and classroom use.

Cthreads is a lightweight threads package that is configurable for a variety of parallel and sequential machines. Cthreads is based on Mach Cthreads and has user level functions for thread management, synchronisation and memory management.

2.8 Data Diffusion Machine

URL <http://www.pact.srf.ac.uk/DDM/welcome.html>

The Data Diffusion Machine (DDM) [12] is a virtual shared memory architecture where data is free to migrate through the machine. The DDM overcomes the problems of shared memory machines - which are generally thought to be convenient for programming but do not scale beyond tens of processors - by providing a virtual memory abstraction on top of a distributed memory architecture. To the user the DDM appears as a conventional shared memory machine - this approach is generally known as Virtual Shared Memory (VSM).

The DDM, like the KSR, are known as cache only memory architectures or COMA's. In this type of machine each data item migrates to the processor(s) where it is used. There are no fixed home locations where data will always be found, instead data is located by means of directories. Data addresses no longer correspond to physical locations, but simply represent names (or tags) for data. As the data is free to move around the system the programmer sees a uniform memory access (UMA) system in which all data is equally accessible.

The purpose of the DDM architecture is to provide a scalable shared memory architecture to the user. In order to do so the DDM uses a hierarchical processor structure. The leaves of the tree consist of processors with large set-associative memories that comprise the sole store for data. The nodes above the leaves are directories that keep track of the data items below. If necessary, ordinary caches can be placed between the processor and the DDM memory, or private memory can be attached to nodes to store, for example, program code.

This project is in the process designing a link-based DDM. Within this project the evaluation of the architecture by means of a (software) emulator that has been calibrated to reflect realistic timings is being undertaken. An accompanying tool gives a real-time animated graphic visualisation, allowing a user to observe the behaviour of the program and spot contention in parts of the memory. This visualisation tool has been demonstrated successfully on up to 72 nodes.

2.9 Distributed Computing Environment (DCE)

URL <http://www.osf.org/dce/index.html>

The OSF Distributed Computing Environment (DCE) [13] is a comprehensive, integrated set of services that supports the development, use and maintenance of distributed applications. It provides a uniform set of services, anywhere in the network, enabling applications to utilise the power of a heterogeneous network of computers.

The DCE aims to be operating system and network-independent, providing compatibility with users' existing environments. DCE includes a comprehensive set of distributed services which provides commercial-quality, inter-operable distributed computing environment.

DISTRIBUTED COMPUTING ENVIRONMENT OVERVIEW

The architecture of DCE tries to mask the physical complexity of the networked environment and provides a layer of logical simplicity for the user. This layer is composed of a set of services that can be used, separately or in combination, to form a DCE. The services are organised into two categories:

Fundamental Distributed Services (FDS) provide tools for software developers to create the user application needed for distributed computing. They include:

- Remote Procedure Call.
- Directory Service.
- Time Service.
- Security Service.
- Threads Service.

Data-Sharing Services provide users with capabilities built upon the FDS. These services require no programming on the part of the user and facilitate better use of information. They include:

- Distributed File System.
- Diskless Support.

DCE AND THE DESKTOP

DCE allows the processing power inherent in a network to be extended to large numbers of nodes. DCE extends the power of the network - and all its resources - to individual users on PCs and Macintosh computers. As a result, PCs and Macintosh machines interconnected by LANs or network servers are no longer isolated. With DCE, they become trusted peer-level client systems that can reach out to servers for information and processing services.

OPERATING SYSTEM AND NETWORK INDEPENDENCE

DCE can be used on a variety of networks and operating systems by system vendors, software developers or end-users. It can be used with any network hardware and transport software, including TCP/IP, OSI, X.25, and other similar products.

The DCE is a portable implementation, written in standard C, which makes use of standard interfaces for operating system services, such as POSIX and X/Open guidelines. It can be ported easily to OSF/1, AIX, DOMAIN OS, ULTRIX, HP-UX, SINIX, SunOS, and UNIX System V operating systems.

2.10 Dome

URL <http://parsys.cs.cmu.edu/dome/Dome.html>

Dome (Distributed Object Migration Environment) [14 & 15] is being used to develop Grand Challenge applications, such as molecular dynamics, nuclear physics, distributed simulation, gene sequencing, and speech recognition, on heterogeneous networks of workstations. Dome makes it possible to take advantage of multi-computers (clusters of workstations), MPPs (massively parallel processors), advanced operating systems, and gigabit networks being developed under the US HPCC program.

Dome provides this capability through libraries of distributed objects (i.e., scalable software) which can be used to program heterogeneous networks of computers as a single resource, obtaining performance that cannot be achieved on the individual machines.

SOME FEATURES OF DOME:

- Addresses the problems of load balancing, ease of programming, and fault tolerance.
- Objects distribute themselves automatically over multiple computers.
- Attempts to load balance the applications being run.
- Supports mechanisms to recover from resource failures.

Dome is available on eight platforms: DEC OSF/1, HP-UX, Intel Paragon, SunOS, Irix, DEC Ultrix, IBM AIX, and Cray C90 Unicos. Other Features of Dome include:

ADAPTIVE COLLECTIVE COMMUNICATIONS - A technique for performing collective communications operations is incorporated in Dome. The technique automatically derives network topology and uses this information to optimise collective operations. This is achieved by choosing message routing schemes that efficiently map to the underlying network.

CHECKPOINTING - Dome has architecture independent checkpoint and restart to include arbitrarily deep call sequences, making the mechanism more flexible. The current system allows checkpoints to be initiated from almost anywhere in a call sequence.

LOAD BALANCING - Dome uses dynamic load balancing techniques to include both global and local load information.

FUTURE WORK:

- Further development of the adaptive collective communications - based on the MPI collective communications operations
- Develop new distributed object classes and applications.
- Implement a streamlined distributed I/O interface that utilises the Scotch Parallel File system.

2.11 FLASH

URL <http://www-flash.stanford.edu/>

The FLASH [16] (Flexible Architecture for SHared memory) multi-processor will be a scalable multi-processor capable of supporting a variety of communication models, including shared memory and message passing protocols, through the use of a programmable node controller. It is planned to provide the architectural support necessary to use FLASH as a traditional "standalone" supercomputer, a compute server as well as a robust multi-user or distributed system.

The key design feature of FLASH is the node controller, MAGIC (Memory And General Interconnect Controller). MAGIC is a custom chip centred around the Protocol Processor, a programmable protocol engine based on the TORCH processor design.

FLASH consists of multiple nodes, each with a processor and caches, a local portion of main memory, and local I/O devices. The nodes communicate through a high-speed low-latency mesh network. Cache coherence is provided by a coherence controller on each node. A machine like this is called a CC-NUMA multi-processor (cache-coherent with non-uniform memory access time) since accesses to local memory are faster than accesses to remote memory on other nodes.

In a CC-NUMA machine, an important unit of failure is the node. A node failure halts a processor and has two direct effects on the memory of the machine: the portion of main memory assigned to that node becomes inaccessible, and any memory line whose only copy was cached on that node is lost. There may also be indirect effects that cause loss of other data.

For the operating system to survive and recover from hardware faults, the hardware must make several guarantees about the behavior of shared memory after a fault. Accesses to unaffected memory ranges must continue to be satisfied with normal cache coherence. Processors that try to access failed memory or retrieve a cache line from a failed node must not be stalled indefinitely. Also, the set of memory lines that could be affected by a fault on a given node must be limited somehow, since designing recovery algorithms requires knowing what data can be trusted to be correct.

These hardware properties collectively make up a memory fault model, analogous to the memory consistency model of a multi-processor which specifies the behavior of reads and writes. The FLASH memory fault model was developed to match the needs of the Hive OS: it provides the above properties, guarantees that the network remains fully connected with high probability (i.e. the operating system need not work around network partitions), and specifies that only the nodes that have been authorised to write a given memory line could damage that line due to a hardware fault.

The Hive prototype is a complete implementation of UNIX SVR4 and is targeted to run on the Stanford FLASH multi-processor. Hive is an operating system with a novel kernel architecture that addresses the issues of reliability and scalability. Hive is fundamentally different from previous monolithic and microkernel SMP OS implementations: it is structured as an internal distributed system of independent kernels called cells. This improves reliability because a hardware or software fault damages only one cell rather than the whole system, and improves scalability because few kernel resources are shared by processes running on different cells.

2.12 GLU Parallel Programming System

URL <http://www.csl.sri.com/GLU.html>

GLU (Granular Lucid) [17] is a high-level programming system for constructing parallel and distributed applications to run on diverse high-performance computing systems. GLU is based on a hybrid programming model that combines multi-dimensional dataflow (Lucid) with imperative models. GLU enables rapid parallel program development using existing sequential code, it helps produce programs that are portable, efficient, and fault tolerant.

The GLU toolkit is for constructing and executing high-performance applications on distributed systems including heterogeneous workstation clusters (over WAN, LAN and wireless networks), multiprocessors, and massively parallel processors. The GLU toolkit consists of a high-level application description notation, a compiler, a graphical application execution user-interface, and a comprehensive performance analysis tool. GLU can be used to develop new parallel applications or convert sequential, legacy, applications to parallel equivalents.

The GLU toolkit has been used to develop high performance applications from existing sequential ones, where the original applications were written in C, Fortran and C++. The GLU system is currently available for the following systems:

- **WORKSTATIONS:** Solaris 1.1 and 2.3, IRIX v4.0.5F, AIX v3.2, OSF1 v1.3 and Linux v1.1.58
- **SYMMETRIC MULTIPROCESSORS:** Solaris 1.1 and IRIX v4.0.5F
- **MASSIVELY PARALLEL PROCESSORS:** SunOS 4.1.3

See URL <http://www.csl.sri.com/Lucid.html> for a description of the Lucid language.

2.13 Local area Multicomputing (LAM)

URL <http://www.osc.edu/lam.html>

LAM [18] runs on each computer as a single Unix daemon structured as a micro-kernel and hand-threaded virtual processes. It provides a simple message-passing and rendezvous service to local processes. Some of the in-daemon processes form a network communication subsystem, which transfers messages to and from other LAM daemons on other machines. The network subsystem adds features like packetisation and buffering to the base synchronisation. Other in-daemon processes are servers for remote capabilities, such as program execution and parallel file access. Users can configure in or out services as necessary.

The features of LAM are transparent to users and system administrators, who only see a conventional Unix daemon. System developers can de-cluster the daemon into a daemon containing only the nano-kernel and several full Unix client processes. The network layer in LAM is a documented, primitive and abstract layer on which to implement a more powerful communication standard like MPI (PVM has also been implemented).

An important feature of LAM is hands-on control of the multi-computer. There is very little that cannot be seen or changed at runtime. Programs residing anywhere can be executed anywhere, stopped, resumed, killed, and watched the whole time. Messages can be viewed anywhere on the multi-computer and buffer constraints tuned as experience with the application dictates. If the synchronisation of a process and a message can be easily displayed, mismatches resulting in bugs can easily be found. These and other services are available both as a programming library and as utility programs run from any shell.

LAM installs anywhere and uses the shell's search path at all times to find LAM and application executables. A cluster is specified as a simple list of machine names in a file, which LAM uses to verify access, start the environment, and remove it.

MPI IMPLEMENTATION

The four main MPI synchronisation variables: context, tag, source rank, destination rank are mapped to LAM's abstract synchronisation at the network layer. MPI debugging tools interpret the LAM information with the knowledge of the LAM/MPI mapping and present detailed information to MPI programmers. A significant portion of the MPI specification is being implemented completely within the runtime system and independent of the underlying environment. MPI programs developed on LAM can be moved without source code changes to any other platform that supports MPI.

2.14 Legion

URL <http://www.cs.virginia.edu/~legion/>

The Legion research project at the University of Virginia aims to provide an architecture for designing and building system services that afford the illusion of a single virtual machine. This virtual machine will provide a secure shared object and name spaces, application adjustable fault-tolerance, improved response time, and greater throughput. Legion will enable fault-tolerance, wide area parallel processing, inter-operability, heterogeneity, a single file name space, protection, security, efficient scheduling, and comprehensive resource management.

Design Philosophy

The principles of the object-oriented paradigm are the foundation for the construction of Legion; the paradigm's encapsulation and inheritance properties are exploited, as well as benefits such as software reuse, fault containment, and reduction in complexity. Other systems require that the programmer address the full complexity of the environment; the difficult problems of load balancing, scheduling, fault management, etc., are likely to overwhelm all but the best programmers.

Design Objectives

SITE AUTONOMY: Legion will not be a monolithic system but be composed of diverse resources owned and controlled by an array of different organisations. These organisations will have control over their own resources, e.g., specifying how much resource can be used, when it can be used, and who can and cannot use the resource.

EXTENSIBLE CORE: The future needs of the many and varied users cannot be fully anticipated and so Legion is composed of extensible and replaceable components. This will permit Legion to evolve with time and allow users to construct mechanisms and policies to meet their own specific needs.

SCALABLE ARCHITECTURE: As Legion will consist of thousands of hosts, it must use a scalable software architecture. For this reason there are no centralised structures (which could become a bottleneck and/or single point of failure), the system is totally distributed.

EASY-TO-USE, SEAMLESS COMPUTATIONAL ENVIRONMENT: Legion tries to mask the complexity of the hardware environment as well as the means of communications and synchronisation of parallel processing. Machine boundaries should be invisible to users. As much as possible, compilers, acting in concert with run-time facilities, manage the environment for the user.

HIGH PERFORMANCE VIA PARALLELISM: Legion will support task and data parallelism and their combinations. Because of the nature of the inter-connection network, Legion must be latency tolerant.

SINGLE, PERSISTENT NAME SPACE: One of the most significant obstacles to wide area parallel processing is the lack of a single name space for file and data access. The existing multitude of disjoint name spaces makes writing applications that span sites extremely difficult. Legion will have a single persistent name space.

SECURITY FOR USERS AND RESOURCE OWNERS: The existing host operating systems can not be replaced, protection and security mechanisms cannot significantly be strengthened. Also, the existing mechanisms must not be weakened by Legion. Further, a mechanism must be provided for users to manage their own security needs; Legion does not define the security policy or require a "trusted" hosts.

MANAGEMENT AND EXPLOITATION OF RESOURCE HETEROGENEITY: Legion will support inter-operability between heterogeneous hardware and software components. In addition, Legion will try to exploit diverse hardware and data resources. Some architectures are better than others at executing particular kinds of code, e.g., vectorisable codes. These affinities, and the costs of exploiting them, must be factored into scheduling decisions and policies.

MULTIPLE LANGUAGE SUPPORT AND INTER-OPERABILITY: Legion applications will be written in a variety of languages. It will be possible to integrate heterogeneous source language application components in much the same manner that heterogeneous architectures are integrated. Legion will be able to support legacy codes.

FAULT-TOLERANCE: In a system as large as Legion, it is certain that at any given instant, several hosts, communication links, and disks will have failed. Fault tolerance will be considered at two levels; at the application level and of Legion itself.

Current Status

Legion is currently based on the Campus Wide Virtual Computer[1] (CWVC). This is a heterogeneous distributed computing system built on top of Mentat[2], an object-oriented parallel processing system. The CWVC demonstrates some of the benefits of a Legion-like system, providing departments across the campus with an easy-to-use interface to high performance distributed computing. The CWVC allows researchers at the University of Virginia to share resources and to develop applications that will be usable in a true Legion setting

2.15 Linda

URL <http://www.cs.yale.edu/HTML/YALE/CS/Linda/linda.html>

Linda was originally developed by D. Gelernter [19] at Yale University. The core of Linda is a small set of functions that can be added to a base language to yield a parallel dialect of that language. It consists mainly of a shared data collection called Tuplespace and a set of functions to access and modify the data stored in the Tuplespace. The unit of communication is a tuple, a list of typed fields which can be either actual or formal. Actual fields have a specific value whereas formal fields are place holders for values. Tuples stored in the Tuplespace can only have actual values. Apart from the limitations of the programming language, there are no limitations for both, the number of fields in a tuple and the field length. There must not be any empty fields in tuples. How tuples are represented in the Tuplespace is highly dependent on the implementation.

In addition to storing and retrieving tuples, the Linda functions provide process synchronization. The basic functions for accessing and modifying the contents of the Tuplespace are:

`out(tuple)` - The function `out()` puts a tuple into the Tuplespace. The tuple can be retrieved by all

other processes with the help of other Linda-functions. With `out()` you can only put data-tuples into the Tuplespace. `Out` never blocks.

`in(pattern-tuple)` - For retrieving tuples from the Tuplespace the function `in()` is used. It removes all tuples which match a given template from the Tuplespace and returns them. A template is a tuple with formal fields, to which actual values can be assigned during the matching process. However, no match exists in the Tuplespace, `in()` blocks until at least one matching tuple appears in the tuple-space.

`rd(pattern-tuple)` - Read is quite similar to `in()`. The difference is that `rd()` does not remove the matching tuples from the Tuplespace.

EVAL(FUNCTION-TUPLE) - Eval creates an active tuple, in fact a task-descriptor for which the Linda-server is responsible for computation. The active tuple may consist of ordinary elements and functions which need evaluation. The results are stored as a passive tuple in the Tuplespace.

2.16 Locust

URL <http://www.cs.sunysb.edu:80/~manish/locust/>

Locust is a distributed shared virtual memory system under development the State University of New York. Locust has been designed for a network of workstations and a prototype is under development for a set of PC's connected by an Ethernet. The key idea of Locust is the exploitation of data dependency information collected at compile time to hide message-latency and to reduce network traffic at the run time.

Pupa is the underlying communication sub-system of Locust. Pupa has been designed specially to support parallel computation in a LAN environment. It co-exists with TCP/IP, and its implementation on a network of PC's connected by a 10 Mbits/sec Ethernet and running FreeBSD. In the future Pupa will be ported to Pentium machines running FreeBSD 2.x and connected by a 100 Mbits/sec Ethernet.

2.17 Message Passing Interface (MPI)

URL <http://www.mcs.anl.gov/mpi/index.html>

The Message Passing Interface [20], commonly referred to as the MPI standard, is an effort to enhance the availability of efficient and portable software for addressing the current needs of the distributed computing community.

In early 1993 the MPI Forum was set up along the lines of the earlier HPF Forum to revise and expand an earlier MPI draft document. The meetings of the MPI Forum were held every six weeks and several mailing lists were created for discussions concerning the MPI Forum and its sub-committees. Both the meetings and the email discussions were open to all members of the high performance computing community. Members of over 40 academic, governmental and industrial institutions contributed to the MPI Forum. The final MPI Standard was released in May 1994 and was followed in June 1994 by a first list of errata and unresolved issues.

The scope of the MPI Standard has been deliberately limited to the message passing programming

model. There is no concept in MPI of a global address space shared by all processes. Some issues like parallel I/O and dynamic process creation were intentionally left aside by the MPI Forum because no standard practice had yet emerged at the time the document was written. Other issues, such as debugging and profiling tools, were felt to be outside the scope of MPI although standard interfaces are provided for the implementors of such tools. The handling of signals from either the communication network or the operating system running a process is not covered by the MPI Standard.

- MPI is intended as a message passing interface for both application and library software.
- The MPI standard is designed for programming multi-computers and also heterogeneous networks of computers. In general, implementations would have to be possible across a wide variety of platforms without any significant modifications to the vendor communication and system software.
- The semantics of the MPI interface were written to be language independent. Specific bindings are provided, in the MPI standard, for programs written in the C and Fortran 77 languages. The MPI interface was designed so that its usage would be as familiar as possible to anyone experienced in writing message passing programs.
- The MPI Forum strived to define a message passing interface that can be efficiently implemented. In particular, unnecessary buffering operations are avoidable. It is also possible for processes to exchange data while they executing tasks whose completion does not depend on that data.
- Reliability and ease of use were design goals of the MPI Forum. In particular, the MPI standard assumes a reliable communication network and most MPI procedures return an error code. Future generations of parallel computers are likely to allow multiple threads of computation within each process. The MPI Interface was designed to be reliable in a multithreading context.

2.18 Mirage+

URL <http://cs.ucr.edu/projects/mirage.html>

Mirage+ is a Distributed Shared Memory (DSM) system which provides memory management and communication across machines in a loosely coupled network environment. The system hides network boundaries to processes accessing shared memory, providing the illusion of a coarse grained multi-processor to the programmer. Mirage+ is designed to execute traditional System V Posix-compliant shared memory programs on commodity personal computers.

A key feature in Mirage+ is the simplicity with which powerful networking can be employed using solutions that have been designed for uni-processors and applied to a more powerful network of workstations platform. Mirage+ implements DSM using a paged segmentation scheme where segments are partitioned into pages. The system runs on a cluster of IBM PS/2 class machines.

The latest work includes a non-kernel invasive DSM system based on the Quarks prototype from Utah. Work is focusing on disconnected operation and mobility for DSM systems. The new research is based on advances and experiences with Mirage+. The project has also constructed the Riverside Mobile DSM Suite which is a set of application programs that exercise functionality in mobile systems that one would expect to implement using DSM. In addition, the project team has authored the DSM port of Quarks to the DEC Alpha AXP 150 personal computers and made the results of the port accessible to other

researchers.

2.19 Networks Of Workstations (NOW)

URL <http://now.cs.berkeley.edu/>

The Berkeley NOW project [21] is building system support for using a network of workstations (NOW) to act as a distributed supercomputer on a building-wide scale. It aims to utilise volume produced commercial workstations and switch-based networks, such as ATM, to show that NOW can be used for both the tasks traditionally run on workstations and large parallel programs.

In conjunction with complementary research efforts in operating systems and communication architecture, it is planned that 100 processor system will be demonstrated during the project lifetime and show that it will have better cost-performance for parallel applications than a massively parallel processing (MPP) machine, and better performance for sequential applications than an individual workstation. To attain these goals the NOW project is combining elements of workstation and MPP technology into a single system. The NOW project includes research and development into network interface hardware, fast communication protocols, distributed file systems, distributed scheduling and job control. The NOW system will consist of the following five packages:

- *Active Messages* - This interface generalises previous active messages interfaces to support a broader spectrum of applications such as client/server programs, file systems, operating systems, as well continuing support for parallel programs.
- *GLUnix* - the Unix co-ordinator - The operating system must support gang-scheduling of parallel programs, identify idle resources in the network (CPU, disk capacity/bandwidth, memory capacity, network bandwidth), allow for process migration to support dynamic load balancing, and provide support for fast inter-process communication for both the operating system and user-level applications. GLUnix is built as a layer on top of existing operating systems and will provide the functionality mentioned above.
- *Fast communication implementations* - The system needs a collection of low-latency, parallel communication primitives. These will include extensions of current ones, including sockets, remote procedure calls, and general messaging primitives on top of Active Messages. These communications layers will provide the link between the large existing base of MPP programs and fast but primitive communication layers.
- *xFS* - A serverless, distributed file system which attempts to have low latency, high bandwidth access to file system data by distributing the functionality of the server among the clients. The typical duties of a server include maintaining cache coherence, locating data, and servicing disk requests. The function of locating data in xFS is distributed by having each client responsible for servicing requests on a subset of the files. File data is striped across multiple clients to provide high bandwidth.
- *Network RAM* - On a fast network with Active Messages, fetching a page from a remote machine's main memory can be more than an order of magnitude faster than getting the same amount of data from the local disk. The system will be able to utilise the memory on idle machines as a paging devices for busy machines. This memory, called Network RAM, will appear as an additional memory hierarchy between the local RAM and the paging space on disk. The Network RAM system being designed is

serverless, in that there is no dedicated Network RAM. Any machine can be a server when it is idle, or a client when it needs more memory than the physical memory it has.

2.20 PAPERS

URL <http://garage.ecn.purdue.edu/~papers/Index.html>

PAPERS [22 & 23], Purdue's Adapter for Parallel Execution and Rapid Synchronisation, is inexpensive hardware designed to provide very low latency barrier synchronisation and aggregate communication operations to user-level unix processes running on a cluster of personal computers or workstations. PAPERS allows a cluster of unmodified PCs and/or workstations to function as a fine-grain parallel computer capable of MIMD, SIMD, and VLIW execution.

The PAPERS library is designed to provide both:

- Good support for hand-coding of parallel applications.
- An efficient target for optimising/parallelising compilers.

The PAPERS execution model is based on the properties of the generalised barrier synchronization mechanism. The typical barrier synchronization or aggregate communication operation using PAPERS is about 3 microseconds, including all hardware and software overheads; this is several orders of magnitude faster than using conventional networks, and is comparable to most commercial parallel supercomputers.

PAPERS consists of:

- Purdue's Adapter for Parallel Execution and Rapid Synchronization.
- Hardware support for parallel processing on a cluster.
- Typical latencies of a few micro-seconds.
- Efficient SIMD and VLIW emulation.
- Works with almost any workstations and/or PCs.
- Public domain hardware design and software.

2.21 Parallel Virtual Machine (PVM)

URL <http://www.epm.ornl.gov/pvm/>

PVM [24] is an integrated set of software tools and libraries that emulates a general-purpose, flexible, heterogeneous concurrent computing framework on interconnected computers of varied architecture. The overall objective of the PVM system is to enable such a collection of computers to be used cooperatively for concurrent or parallel computation. The principles upon which PVM is based include the following:

- User-configured host pool: A PVM application run on a set of machines that are selected by the user. Both shared and distributed-memory computers may be part of the host pool.
- Definable access to hardware: A PVM application views the hardware environment as an attributeless collection of virtual processing elements or user specified to take advantage of different machine

capabilities.

- Process-based computation: The unit of parallelism in PVM is an independent sequential thread of control that alternates between communication and computation. Multiple tasks may execute on a single processor.
- Explicit message-passing model: Collections of computational tasks, each performing a part of an application's workload using data, functional, or hybrid decomposition, cooperate by explicitly sending and receiving messages to one another.
- Heterogeneity support: The PVM system supports heterogeneity in terms of machines, networks, and applications.
- Multiprocessor support: PVM uses the native message-passing facilities on multiprocessors to take advantage of the underlying hardware. Vendors often supply their own optimized PVM for their systems, which can still communicate with the public PVM version.

The PVM system is composed of two parts. The first part is a daemon, that resides on all the computers making up the virtual machine. The daemon is designed so any user with a valid login can install this daemon on a machine. When a user wishes to run a PVM application, he/she first creates a virtual machine by starting up PVM. The PVM application can then be started from a Unix prompt on any of the hosts. Multiple users can configure overlapping virtual machines, and each user can execute several PVM applications simultaneously.

The second part of the system is a library of PVM interface routines. It contains a functionally complete repertoire of primitives that are needed for cooperation between tasks of an application. This library contains user-callable routines for message passing, spawning processes, coordinating tasks, and modifying the virtual machine.

2.22 The SHRIMP Project

URL <http://www.cs.princeton.edu/shrimp/>

SHRIMP (Scalable, High-Performance, Really Inexpensive Multi-Processor) [25] is a parallel machine being designed and built in the Computer Science Department at Princeton University. Shrimp is built from highly-integrated, commodity parts. The computing nodes of SHRIMP are Pentium PCs, and the routing network is the same one used in the Intel Paragon. A network interface card that connects the PCs to the routing network and the software to make SHRIMP a fully usable multi-computer is being designed and implemented.

SHRIMP uses an innovative model of communication called *virtual memory mapped communication*. This has several advantages over existing systems: it allows communication directly from user-level without violating protection, it enables very-low-overhead implementations of message-passing, and it is only a little bit harder to design and build than conventional DMA-based interfaces.

2.23 The THESIS Distributed Operating System Project

URL <http://info.mcc.ac.uk/MultiComp/dosp.html>

The aim of the THESIS (Transparent HETerogeneous Single Image System) [26 & 27] project is to exploit the fact that many commercial operating systems are now becoming micro-kernel based (i.e. using Mach), in order to create heterogeneous clusters of workstations. These workstations would be largely unchanged systems, requiring the replacement of existing pagers with distributed pagers, the introduction of distributed process managers and also translation mechanisms to enable the migration of text and data between heterogeneous processors.

The primary problems in the design of a distributed operating system for workstations which must be addressed are:

- Network latency. The advantages of the resources gained through collaboration must be offset against the overhead of the increase in communication times between machines.
- Coherency. The workstations must conspire together to produce the illusion of a single virtual memory.
- Heterogeneity. The collaborating machines will probably not all be code or data compatible. The translation of data and the use of 'fat8' binaries can be used to address this problem, but every effort must be made to avoid their costly use at runtime.
- Fault tolerance. Distributing a process and its memory over a number of machines increases its vulnerability to host and network failures. Provision for migrating processes away from failing or heavily loaded machines should be made.

2.24 TreadMarks

URL <http://www.cs.rice.edu/~willy/TreadMarks/overview.html>

TreadMarks [28] supports parallel computing across a heterogeneous workstation cluster by providing applications with a global shared memory interface. The shared memory primitives within TreadMarks facilitates the transition from sequential to parallel programs.

TreadMarks is implemented entirely as a user level library on top of the native Unix environment, which has all the necessary communications and memory-management functions required to implement TreadMarks. Programs written in C, C++ and Fortran are compiled and linked with the TreadMark library using any standard compiler for the language. TreadMark implements inter-machine communications using the Berkeley sockets interface.

The challenge in providing a shared memory interface is to do so efficiently. To this end, TreadMarks incorporates several innovative features, including release consistency and multiple-writer protocols. Current research on the TreadMarks project includes the integration of compiler and runtime techniques, the use of high-level synchronisation operators, multithreading - in particular on multi-processor nodes. TreadMarks runs on IBM, DEC, SUN, HP, and SGI hardware. TreadMarks was developed with support from the Texas Advanced Technology Program (TATP).

2.25 WANE

URL <http://www-wane.scri.fsu.edu/>

WANE is a comprehensive Wide Area Networked Environment being developed at the Supercomputer Computations Research Institute as part of a three year project funded by the US Department of Energy. It is designed to be a highly scalable and fault tolerant computing and information suite. WANE draws heavily from existing and ongoing software projects and attempts to integrate them into a single entity. Some of the key technologies exploited by WANE include:

- DQS - CPU and device scheduling (batch and interactive).
- PostGres - database accesses.
- Tcl/Tk - X-Window development.
- Mosaic/WWW/gopher - internet information services.
- FreeNet - information services.

The Wane Server

The WANE Server software package is a "turnkey" solution targeted at a diverse audience, which includes commercial, governmental, educational, and individual users. The system is designed to support thousands of users, is incrementally scalable, and fault tolerant. The package is modular, with services ranging from a simple mail or WWW server to a full fledged free net service package. The server package contains:

- Extensive user access controls.
- Hierarchical administrative levels.
- Multiple user domains.
- Support of separate administrative groups, allowing delegation of administrative tasks.
- User friendly graphical interface for user administration.
- A complete distribution of Linux.
- Multi-user operating system on inexpensive PC clones.
- Mail Hub service.
- World Wide Web multimedia server software.
- Internet connection tools/software.
- Internet Newsgroups.
- Comprehensive suite of interactive personal communication tools (Text, Audio, Video).
- Access to vast archives of software and data.

Client Software:

The client software includes a variety of public domain and shareware packages enabling users to connect to the Internet via modem or LAN. The WANE client distribution provides support for MacintoshOS, DOS, Windows, Windows NT, OS/2, and Linux.

Chapter 3

3. Conclusions

3.1 General Comments

The information collected and presented in this review is just a "snap-shot" of the DCCE packages available and found, via the Internet, during December 1995.

Anyone taking a brief look through this review will quickly establish that a large percentage of the packages described are interesting research projects that will probably go nowhere. But work on these projects will feed into other projects which will further our knowledge and understanding of the problems and needs associated with cluster computing.

3.2 The Future ?

The popularity of the WWW along with the increasingly functional and maturing tools [29] indicates that future successful cluster management systems will be based on this technology.

Presently no integrated WWW-based cluster management systems exists. But the fundamental infrastructure needed to produce such a system is already in-place. It should be a relatively simple exercise to use the WWW as a uniform interface to administer and run applications on a heterogeneous computing environment.

- Communications protocols are established (`http`, `ftp`)
- Distributed management daemons already run on most machines (`httpd`) - including PC and MAC servers.
- Well known user interfaces are already available (`netscape` and `mosaic`).
- A powerful scripting language for these systems is available (`Perl5`).
- User I/O is simple to set up (`forms`).
- Other tools are being added on a daily basis.

Finally, it seems likely that the experience learned from the packages reviewed in this document will be used to produce a WWW based system in the very near future.

Chapter 4

4. Glossary of Terms

AFS

AFS is a distributed filesystem that enables co-operating hosts (clients and servers) to share filesystem resources across both local area and wide area networks. AFS is based on a distributed file system originally developed at the Information Technology Center at Carnegie-Mellon University that was called the "Andrew File System". AFS is marketed, maintained, and extended by Transarc Corporation.

Amdahl's Law

A rule first formalised by Gene Amdahl in 1967, which states that if F is the fraction of a calculation

that is serial and $1-F$ the fraction that can be parallelised, then the speedup that can be achieved using P processors is: $1/(F + (1-F)/P)$ which has a limiting value of $1/F$ for an infinite number of processors. This no matter how many processors are employed, if a calculation has a 10% serial component, the maximum speedup obtainable is 10.

Application Programming Interface (API)

A set of library routine definitions with which third party software developers can write portable programs. Examples are the Berkeley Sockets for applications to transfer data over networks, those published by Microsoft for their Windows graphical user interface, and the Open/GL graphics library initiated by Silicon Graphics Inc. for displaying three dimensional rendered objects.

Bandwidth

The communications capacity (measured in bits per second) of a transmission line or of a specific path through the network. Contiguous bandwidth is a synonym for consecutive grouped channels in multiplexer, switch or DACS; i.e. 256kbps (4 x 64kbps channels).

Batching System

A batching system is one that controls the access to computing resources of applications. Typically a user will send a request the batch manager agent to run an application. The batch manager agent will then place the job (see definition below) in a queue (normally FIFO).

Cluster Computing

A commonly found computing environment consists of many workstations connected together by a local area network. The workstations, which have become increasingly powerful over the years, can together, be viewed as a significant computing resource. This resource is commonly know as cluster of workstations.

Distributed Computing Environment

The OSF Distributed Computing Environment (DCE) [13] is a comprehensive, integrated set of services that supports the development, use and maintenance of distributed applications. It provides a uniform set of services, anywhere in the network, enabling applications to utilise the power of a heterogeneous network of computers.

flops

Floating point operations per second; a measure of memory access performance, equal to the rate which a machine can perform single precision floating-point calculations.

Heterogeneous

Containing components of more than one kind. A heterogeneous architecture may be one in which some components are processors, and others memories, or it may be one that uses different types of processor together.

HPF

A language specification published in 1993 by experts in compiler writing and parallel computation, the aim of which is to define a set of directives which will allow a Fortran 90 program to run efficiently on a distributed memory machine. At the time of writing, many hardware vendors have expressed interests, a few have preliminary compilers, and a few independent compiler producers also have early releases. If successful, HPF would mean data parallel programs can be written portably for various multiprocessor platforms.

Homogeneous

Made up of identical components. A homogeneous architecture is one in which each element is of the same type - processor arrays and multicomputers are usually homogeneous. See also heterogeneous.

Homogeneous vs Heterogeneous

Often a cluster of workstations is viewed as either homogenous or heterogeneous. These terms are ambiguous, as they refer to not only the make of the workstation but also to the operating system being used on them. For example, it is possible to have a homogenous cluster running various operating systems (SunOS and Solaris, or Irix 4 and 5).

In this review we define homogenous as a cluster of workstations of the same make (i.e. Sun, HP, IBM, etc.). In contrast everything else is referred to as heterogeneous, i.e. a mixture of different makes of workstations.

I/O

Refers to the hardware and software mechanisms connecting a computer with its environment. This includes connections between the computer and its disk and bulk storage system, connections to user terminals, graphics systems, and networks to other computer systems or devices.

Interconnection network

The system of logic and conductors that connects the processors in a parallel computer system. Some examples are bus, mesh, hypercube and Omega networks.

Internet Protocol (IP)

The network-layer communication protocol used in the DARPA Internet. IP is responsible for host-to-host addressing and routing, packet forwarding, and packet fragmentation and reassembly.

Interprocessor communication

The passing of data and information among the processors of a parallel computer during the execution of a parallel program.

Job

This term generally refers to an application sent to a batching system - a job finishes when the application has completed its run.

Latency

The time taken to service a request or deliver a message which is independent of the size or nature of the operation. The latency of a message passing system is the minimum time to deliver a message, even one of zero length that does not have to leave the source processor. The latency of a file system is the time required to decode and execute a null operation.

Load balance

The degree to which work is evenly distributed among available processors. A program executes most quickly when it is perfectly load balanced, that is when every processor has a share of the total amount of work to perform so that all processors complete their assigned tasks at the same time. One measure of load imbalance is the ratio of the difference between the finishing times of the first and last processors to complete their portion of the calculation, to the time taken by the last processor.

Memory Hierarchy

Due to the rapid acceleration of microprocessor clock speeds, largely driven by the introduction of RISC technology, most vendors no longer supply machines with the same main memory access speeds as the CPU clock. Instead, the idea of having hierarchies of memory subsystems, each with different access speeds is used to make more effective use of smaller, expensive fast (so called cache) memories. In the context of computer networks, this hierarchy extends to storage devices, such as the local disk on a workstation (fast), to the disk servers, to off-line secondary storage devices (slow).

Message passing

A style of inter-process communication in which processes send discrete messages to one another. Some computer architectures are called message passing architectures because they support this model in hardware, although message passing has often been used to construct operating systems and network software for uni-processors and distributed computers..

Metacomputer

A term invented by Paul Messina to describe a collection of heterogeneous computers networked by a high speed wide area network. Such an environment would recognise the strengths of each machine in the Metacomputer, and use it accordingly to efficiently solve so-called Metaproblems. The World Wide Web has the potential to be a physical realisation of a Metacomputer. Also, see Metaproblem.

Metaproblem

A term invented by Geoffrey Fox for a class of problem which is outside the scope of a single computer architecture, but is instead best run on a Metacomputer with many disparate designs. An example is the design and manufacture of a modern aircraft, which presents problems in geometry, grid generation, fluid flow, acoustics, structural analysis, operational research, visualisation, and database management.

The Metacomputer for such a Metaproblem would be networked workstations, array processors, vector supercomputers, massively parallel processors, and visualisation engines.

MIPS

One Million Instructions Per Second. A performance rating usually referring to integer or non-floating point instructions. See also MOPS.

Message Passing

A style of interprocess communication in which processes send discrete messages to one another. Some computer architectures are called message passing architectures because they support this model in hardware, although message passing has often been used to construct operating systems and network software for uniprocessors and distributed computers.

Message Passing Interface (MPI)

The parallel programming community recently organised an effort to standardise the communication subroutine libraries used for programming on massively parallel computers such as Intel's Paragon, Cray's T3D, as well as networks of workstations. MPI not only unifies within a common framework programs written in a variety of existing (and currently incompatible) parallel languages but allows for future portability of programs between machines.

Massively Parallel Processing (MPP)

The strict definition of MPP is a machine with many interconnected processors, where 'many' is dependent on the state of the art. Currently, the majority of high-end machines have fewer than 256 processors. A more practical definition of an MPP is a machine whose architecture is capable of having many processors - that is, it is scalable. In particular, machines with a distributed memory design (in comparison with shared memory designs) are usually synonymous with MPPs since they are not limited to a certain number of processors. In this sense, "many" is a number larger than the current largest number of processors in a shared-memory machine.

Multicomputer

A computer in which processors can execute separate instruction streams, have their own private memories and cannot directly access one another's memories. Most multicomputers are disjoint memory machines, constructed by joining nodes (each containing a microprocessor and some memory) via links.

Multitasking

Executing many processes on a single processor. This is usually done by time-slicing the execution of individual processes and performing a context switch each time a process is swapped in or out - supported by special-purpose hardware in some computers. Most operating systems support multitasking, but it can be costly if the need to switch large caches or execution pipelines makes context switching expensive in time.

Network

A physical communication medium. A network may consist of one or more buses, a switch, or the links joining processors in a multicomputer.

NFS

Network Filing System is a protocol developed to use IP and allow a set of computers to access each other's file systems as if they were on the local host.

Network Information Services - NIS (former Yellow Pages)

Developed by Sun Microsystems, NIS is a means of storing network wide information in central databases (NIS servers), where they can be accessed by any of the clients. Typically, an NIS database will be used to store the user password file, mail aliases, group identification numbers, and network resources. The use of a single server avoids the problem of data synchronisation.

Parallel Job

This can be defined as a single application (job) that has multiple processes that run concurrently . Generally each process will run on a different processor (workstation) and communicate boundary, or other data, between the processes at regular intervals. Typically a parallel job would utilise a message passing interface, such as MPI or PVM, to pass data between the processes.

Process

The fundamental entity of the software implementation on a computer system. A process is a sequentially executing piece of code that runs on one processing unit of the system.

Queuing

Queuing is the method by which jobs are ordered to access some computer resource. Typically the batch manager will place a job the queue. A particular compute resource could possibly have more than one queue, for example queues could be set up for sequential and parallel jobs or short and long job runs.

Remote Procedural Call (RPC)

A mechanism to allow the execution of individual routines on remote computers across a network. Communication to these routines are via passing arguments, so that in contrast to using Sockets, the communication itself is hidden from the application. The programming model is that of the clients-servers.

Sequential computer

Synonymous with a Von Neumann architecture computer and is a "conventional" computer in which only one processing element works on a problem at a given time.

Sequential Job

This can be defined as a job that does not pass data to remote processes. Typically such a job would run on a single workstation - it is possible that for a sequential process to spawn multiple threads on its processor.

Single Point of Failure

This is where one part of a system will make the whole system fail. In cluster computing this is typically the batch manager, which if it fails the compute resource are no longer accessible by users.

Sockets

Also commonly known as Unix Berkeley Sockets, these were developed in the early 1980s as a means of providing application writers a portable means of accessing the communications hardware of the network. Since sockets allow point to point communications between processes, it is used in most of the networked workstation implementations of message passing libraries.

Speedup

The ratio of two program execution times, particularly when the times are from execution on 1 and P nodes of the same computer. Speedup is usually discussed as a function of the number of processors, but is also a function (implicitly) of the problem size.

Supercomputer

A time dependent term which refers to the class of most powerful computer systems world-wide at the time of reference.

Transmission Control Protocol (TCP)

TCP is a connection-oriented transport protocol used in the DARPA Internet. TCP provides for the reliable transfer of data as well as the out-of-band indication of urgent data.

Chapter 5

5. References

[1] M.A. Baker, H.W. Yau, *Cluster Computing Review*, NPAC Technical Report, SCCS-748, NPAC, Syracuse University, USA, October 1995.

[2] D. Hutchinson and P. Lever, *Uses of Computing Clusters for Parallel and Cooperative Computing*, A Report for JISC NTSC, Manchester University, December 1993.

[3] M. Baker, T. Hey and L. Robertson, *Cluster Computing Report*, A Report for the JISC NTSC, The University of Southampton, January 1995.

[4] M.A. Baker, G.C. Fox, *Cluster Computing Management Software*, NPAC Technical Report

SCCS-XXX, NPAC, Syracuse University, USA, January 1996.

[5] A.S. Tanenbaum, *The Amoeba Distributed Operating System*, Vrije Universiteit Technical Report, URL <ftp://ftp.cs.vu.nl/pub/amoeba/Intro.ps.Z>

[6] D. J. Becker, T. Sterling, D. Savarese, J. E. Dorband, U. A. Ranawake and C. V. Packer *Beowulf: A Parallel Workstation for Scientific Computation*, ICPP Paper, August 1995.

[7] W McColl, *Bulk Synchronous Parallel Computing*, 2nd Workshop on Abstract Machines for Highly Parallel Computing, Univ of Leeds, 14th-16th April 1993.

[8] R Miller, *A Library for Bulk-synchronous Parallel Computing*, British Computer Society Parallel Processing Specialist Group, General Purpose Parallel Computing, December 1993.

[9] R Miller and J Reed, *The Oxford BSP Library: Users' Guide*, version 1.0, Oxford Parallel, 1993.

[10] A. Baratloo, P. Dasgupta and Z.M Kedem, *CALYPSO: An Environment for Reliable Distributed Parallel Processing*, Technical Report, New York University USA, Sep. 1995.

[11] K. Schwan, H. Forbes, A. Gheith, B. Mukherjee and Y. Samiotakis, *A Cthread Library for Multiprocessors*, Tech report GIT-ICS-91/02, College of Computing, Georgia Institute of Technology, Atlanta, GA 30332. Jan 1991.

[12] D. H. Warren and S. Haridi, *The Data Diffusion Machine - A Scalable Shared Virtual Memory Multiprocessor*. In Proceedings of the 1988 International Conference on Fifth Generation Computer Systems. Tokyo, Japan, pp 943-952, December 1988.

[13] The OSF Distributed Computing Environment, Overview - URL <http://www.osf.org/comm/lit/OSF-DCE-PD-1090-4.html>

[14] J. Arabe, A. Beguelin, B. Lowekamp, E. Seligman, M. Starkey, and P. Stephan *Dome: Parallel programming in a heterogeneous multi-user environment*, Carnegie Mellon University, March 1995. Submitted for presentation at Supercomputing '95.

[15] E. Seligman and A. Beguelin, *High-Level Fault Tolerance in Distributed Programs*, Technical Report CMU-CS-94-223, School of Computer, Science, Carnegie Mellon University, December 1994

[16] J. Kuskin, et. al. *The Stanford FLASH Multiprocessor*, The Proceedings of the 21st International Symposium on Computer Architecture, pages 302-313, Chicago, IL, April 1994.

[17] *GLU Programmer's Guide v0.9*, SRI CSL Technical Report 94-06, 1994.

[18] LAM Overview - URL <http://www.osc.edu/Lam/lam/overview.html>

[19] R. Bjornson, N. Carriero, D. Gelernter, and L. Jerrold. *Linda, the Portable Parallel*. Technical Report 520, Yale University Department of Computer Science, Jan. 1988.

[20] J. J. Dongarra, S. W. Otto, M. Snir, and D. W. Walker, *An Introduction to the MPI Standard*, Communications of the ACM (submitted), January 1995.

- [21] T. Anderson, D. Culler, D. Paterson, et. at., *A Case for Networks of Workstations (NOW)*, Computer Science Division, EECS Department, University of California, Berkeley, 1994.
- [22] H. G. Dietz, T. M. Chung, and T. I. Mattox, *A Parallel Processing Support Library Based On Synchronized Aggregate Communication*, Submitted to 1995 Workshop on Languages and Compilers for Parallel Machines.
- [23] H. G. Dietz, T. M. Chung, T. I. Mattox, and T. Muhammad, *Purdue's Adapter for Parallel Execution and Rapid Synchronization: The TTL_PAPERS Design*, Purdue University School of Electrical Engineering, Technical Report, January 1995.
- [24] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek and V. Sunderam, *PVM: Parallel Virtual Machine - A Users' Guide and Tutorial for Networked Parallel Computing*, Scientific and Engineering Series, The MIT Press, ISBN 0-262-57108-0
- [25] M. A. Blumrich, K. Li, R. Alpert, C. Dubnicki, E. W. Felten, and J. Sandberg, *Virtual Memory Mapped Network Interface for the SHRIMP Multicomputer*, Int'l Symposium on Computer Architecture, April 1994, pp. 142 - 153.
- [26] I. McLean, *A Multiple pager approach to distributed memory coherence*. Dec 1994. URL <http://info.mcc.ac.uk/MultiComp/reports.html>
- [27] M. Kelly, *The Method Of Imaginary Machines, a Systems' Analysis Tool*. Dec 1994. URL <http://info.mcc.ac.uk/MultiComp/reports.html>
- [28] C. Amza, A.L. Cox, S. Dwarkadas, P. Keleher, H. Lu, R. Rajamony, W. Yu, and W. Zwaenepoel, *TreadMarks: Shared Memory Computing on Networks of Workstations*, to appear in IEEE Computer (draft copy) December 1995.
- [29] G.C. Fox, W. Furmanski, M. Chen, C. Rebbi, J. Cowie, *WebWork: Integrated Programming Environment Tools for National and Grand Challenges*, NPAC Technical Report SCCS-715, Syracuse University, June 14 1995.