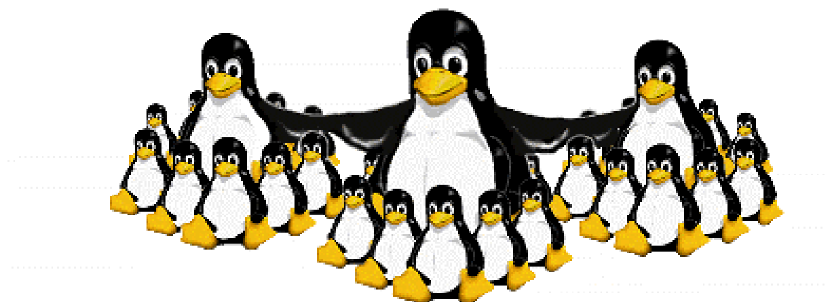
Cluster Computing

Architectures, Operating Systems, Parallel Processing & Programming Languages

Author Name: Richard S. Morrison



Revision Version 2.4, Monday, 28 April 2003
Copyright © Richard S. Morrison 1998 – 2003
This document is distributed under the GNU General Public Licence [39]
Print date: Tuesday, 28 April 2003
Document owner: Richard S. Morrison, r.morrison@ndy.com ✈ +612-9928-6881
Document name: CLUSTER_COMPUTING_THEORY
Stored: (\\RSM\FURTHER_RESEARCH\CLUSTER_COMPUTING)

Synopsis & Acknolegdements

My interest in Supercomputing through the use of clusters has been long standing and was initially sparked by an article in Electronic Design [33] in August 1998 on the Avalon Beowulf Cluster [24].

Between August 1998 and August 1999 I gathered information from websites and parallel research groups. This culminated in September 1999 when I organised the collected material and wove a common thread through the subject matter producing two handbooks for my own use on cluster computing. Each handbook is of considerable length, which was governed by the wealth of information and research conducted in this area over the last 5 years. The cover the handbooks are shown in Figure 1-1 below.

Figure 1-1 – Author Compiled Beowulf Class 1 Handbooks

Through my experimentation using the Linux Operating system and the undertaking of the University of Technology, Sydney (UTS) undergraduate subject Operating Systems in Autumn Semester 1999 with Noel Carmody, a systems level focus was developed and is the core element of this material contained in this document.

This led to my membership to the IEEE and the IEEE Technical Committee on Parallel Processing, where I am able to gather and contribute information and be kept up to date on the latest issues.

My initial interest in the topic has developed into a very practical as well as detailed theoretical knowledge. In Autumn semester 2001 I undertook to tutor the UTS Operating Systems subject which included guest lectures. This enabled me to further develop my ability to articulate my knowledge to students with no previous experience of the internals of operating systems or systems level programming and facilitate their learning.

Focus

This document reviews the current parallel systems theory with respect to Beowulf type clustering and experimentally investigates certain elements of parallel systems performance. The investigation work was carried out at the University of Technology, Sydney in Laboratory 1/2122E.

A separate document could be written on developing applications for parallel systems and currently it would be largely dependant on the target system.

I have however focused on the systems level as it has been apparent over the years that I have been researching the area that while Scientific Computing drove the HPCC to develop clustering, Corporations are now requiring the same level of advantages as possible with Clusters. As such, Operating system vendors are now taking the experience of the HPCC and integrating these features into their operating systems.

Next generation tools are available now to develop parallel programs and it is envisioned that parallel systems will be the only model in the future. Before this can happen, standardisation must be reached *(Formal or Pseudo*), as this will be an important step to minimise the transitional cost to parallel software. [Refer to Section 3.4.2 for further detail]

Executive Summary

Cluster Computing:

Architectures, Operating Systems, Parallel Processing & Programming Languages

The use of computers within our society has developed from the very first usage in 1945 when the modern computer era began, until about 1985 when computers were large and expensive.

Two modern age technologies, the development of high-speed networking and the personal computer have allowed us to break-down these price barriers and construct cost effective clusters of PCs which provide comparable performance to super-computers at a fraction of the cost. As PC's and networks are in common use, this allows most commercial organizations, governments, and educational institutions access to high performance super-computers.

The major difference between a network of PC's and a super-computer is the software which is loaded on each machine, and the way in which an application is processed, namely in parallel.

Parallel processing is the method of breaking down problems or work into smaller components to be processed in parallel thus taking only a fraction of the time it would take to run on a stand-alone PC.

The only drawback to this cost-effective way of computing is how can we effectively design these systems to meet our performance needs? Can widely used operating systems be used such as Windows? What software is available for users on this type of machine and how do we run this software on other machines built using the same technology? Can we use existing applications or do we need to develop new ones, if so how? How can we ensure that each PC is doing its fair share of work, or is not overloaded?

This document explores these issues from theory to practice, details a design methodology and shows by experimental investigation that from a structured design the speedup obtained with many PC's can be known within bounds prior to implementation.

To achieve our initial cost-effectiveness the Linux Operating system is used, however Windows NT can be used if desired while still maintaining a competitive edge over traditional super-computers. Additionally programming languages are available that abstract from the system and free the programmer up from worrying about system details.

Richard S. Morrison
B.E. (Computer Systems) Hons
MIEEE, MIEAust

February 2003

Contents

1. INTRODUCTION 12
1.1. BRIEF HISTORY OF COMPUTING AND NETWORKING 12
1.2. PARALLEL PROCESSING 12
1.3. MOTIVATION 13
1.3.1. Applications of Parallel Processing 14
2. ARCHITECTURES 17
2.1. COMPUTER CLASSIFICATION SCHEMES 17
2.2. CLUSTER COMPUTING CLASSIFICATION SCHEMES 21
2.3. BEOWULF 22
2.3.1. History 22
2.3.2. Overview 23
2.3.3. Classification 24
2.4. NOW/COW 25
2.5. DISTRIBUTED VS. CENTRALIZED SYSTEMS 26
3. SYSTEM DESIGN 28
3.1. PERFORMANCE REQUIREMENTS 29
3.1.1. The Need for Performance Evaluation 29
3.1.2. Performance Indices of Parallel Computation 30
3.1.3. Theoretical Performance of Parallel Computers 31
3.1.4. Performance Analysis and Measurement 36
3.1.5. Practical Performance of Parallel Computers 36
3.2. HARDWARE PLATFORMS 38
3.2.1. CPU 38
3.2.2. Symmetric Multiprocessing 38
3.2.3. Basic Network Architectures 40
3.2.3.1. Network Channel Bonding 42
3.2.4. Node Interconnection Technologies 43
3.3. OPERATING SYSTEMS 44
3.3.1. General 44
3.3.2. Towards Parallel Systems 44
3.3.3. Implementations 46
3.3.4. Redhat Linux 7.2 46
3.3.5. Microsoft Windows 2000 50
3.3.6. Sun Solaris 54
3.3.7. Other 54
3.4. MIDDLEWARE 55
3.4.1. Parallel Communications Libraries 56
3.4.1.1. PVM Overview 56
3.4.1.2. MPI Overview 57
3.4.2. Application Development Packages 59
3.4.2.1. BSP 61
3.4.2.2. ARCH 61
4. SYSTEM INSTALLATION & TESTING 63
4.1. BUILDING A BEOWULF 63
4.2. PERFORMANCE TESTING 66
4.2.1. Beowulf Performance Suite 66
4.2.2. The Linpack Benchmark 67

4.3. SYSTEM ADMINISTRATION 68
4.3.1. General 68
4.3.2. Mosixview 68
4.4. APPLICATIONS TESTING 69
4.4.1. Persistence of Vision 69

5. RESULTS 73

5.1. SUMMARY OF NUMERICAL DATA 73
5.2. RESULTS ANALYSIS 73

6. CONCLUSION 76

7. REFERENCES 78

8. APPENDIX 85

8.1. APPENDIX A – NODE INTERCONNECTION TECHNOLOGIES 86
8.1.1. Class 1 Network Hardware 86
8.1.2. Class 2 Network Hardware 90
8.2. APPENDIX B – CHANNEL BONDING 97
8.3. APPENDIX C – MPI IMPLEMENTATIONS 103
8.3.1. LAM 104
8.3.2. MPICH 105
8.4. APPENDIX D – POV-RAY 106
8.4.1. POV-Ray Benchmark 106
8.4.2. Alternate Bench Mark POV Files 108
8.4.3. POV-Ray Benchmark Performance data 117
8.5. APPENDIX E – RAW RESULTS 118
8.5.1. Lam MPI Cluster boot-up and tear-down 118
8.5.2. POV-Ray 119

Table of Figures

FIGURES

FIGURE 1-1 – AUTHOR COMPILED BEOWULF CLASS 1 HANDBOOKS 3
FIGURE 1-1 – INTERACTION AMONG EXPERIMENT, THEORY AND COMPUTATION 14
FIGURE 1-2 – ACSI WHITE AT THE LAWRENCE LIVERMORE NATIONAL LABORATORY 16
FIGURE 2-1 – THE FLYNN-JOHNSON CLASSIFICATION OF COMPUTER SYSTEMS 19
FIGURE 2-2 – ALTERNATE TAXONOMY OF PARALLEL & DISTRIBUTED COMPUTER SYSTEMS 19
FIGURE 2-3 – A PIPELINED PROCESSOR 20
FIGURE 2-4 – SPACE-TIME DIAGRAM FOR A PIPELINE PROCESSOR 20
FIGURE 2-5 – SPACE-TIME DIAGRAM FOR A NON-PIPELINED PROCESSOR 21
FIGURE 2-6 – CLUSTER COMPUTER CLASSIFICATION SCHEME 21
FIGURE 2-7 – CLUSTER COMPUTER ARCHITECTURE 22
FIGURE 2-8 – AVALON BEOWULF AT LANL 23
FIGURE 2-9 – LOGO FROM THE BERKLEY NOW PROJECT 25
FIGURE 3-1 – LAYERED MODEL OF A CLUSTER COMPUTER 28
FIGURE 3-2 – MAIN STEPS LEADING TO LOSS OF PARALLELISM 29
FIGURE 3-3 – VARIOUS ESTIMATES OF AN N-PROCESSOR SPEEDUP 35
FIGURE 3-4 – INTEL SMP SERVER BLOCK DIAGRAM 39
FIGURE 3-5 – SYSTEM COMPONENT RELATIONSHIPS IN THE LINUX OPERATING SYSTEM 47
FIGURE 3-6 – LINUX HIERARCHICAL FILE SYSTEM STRUCTURE 48
FIGURE 3-7 – WINDOWS 2000 ARCHITECTURE 50
FIGURE 3-8 – WINDOWS 2000 CLUSTER SERVER TOPOLOGICAL DIAGRAM 51
FIGURE 3-9 – WINDOWS 2000 CLUSTER SERVER BLOCK DIAGRAM 52
FIGURE 3-10 – SUN CLUSTER STRUCTURE 54
FIGURE 3-11 – THE PARALLEL PROGRAMMING MODEL EFFICIENCY ABSTRACTION TRADE-OFF 60
FIGURE 4-1 – LAYERED MODEL OF BEOWULF CLUSTER COMPUTER USED IN TESTING 63
FIGURE 4-2 – TEST CLUSTER COMPUTER IN LAB B1/2122E 66
FIGURE 4-3 – BPS RUNNING ON NODE1 OF THE TEST CLUSTER 66
FIGURE 4-4 – MAIN WINDOW OF MOSIXVIEW CLUSTER MANAGEMENT SOFTWARE 68
FIGURE 4-5 – RAY TRACING BENCH MARK TEST OUTPUT 69
FIGURE 4-6 – SAMPLE OUTPUT OF POVRAY'S ABILITIES 70
FIGURE 4-7 – ALTERNATE POV-RAY TEST CASE 1 70
FIGURE 4-8 – ALTERNATE POV-RAY TEST CASE II 71
FIGURE 5-1 – TEST RESULTS & THEORETICAL PERFORMANCE COMPARISON 75

TABLES

TABLE 3-1 – TOPOLOGICAL PARAMETERS OF SELECTED INTERCONNECTION NETWORKS 41
TABLE 5-1 – RENDERING TEST RESULTS 73
TABLE 8-1 – LAM MPI 2 IMPLEMENTATION DETAILS 104
TABLE 8-2 – MPICH MPI 2 IMPLEMENTATION DETAILS 105
TABLE 8-3 – POVBENCH POV-RAY PERFORMANCE BENCHMARK DATA EXTRACT 117

Definitions and Acronyms

The following words and acronyms have been used throughout this document:

Word List	Definition
AI	Artificial Intelligence
API	Application Programming Interface
ATM	Asynchronous Transfer Mode
BSP	Bulk Synchronous Parallel
CAD	Computer Aided Design
CAI	Computer Assisted Instruction
CAM	Computer Aided Manufacturing
CAPERS	Cable Adapter for Parallel Execution and Rapid Synchronization
CESDIS	Centre of Excellence in Space Data and Information Sciences
CORBA	Common Object Request Broker Architecture
COTS	Commodity off the shelf
COW	Cluster of Workstations
CPU	Central Processing Unit
DEC	Digital Equipment Corporation (now Compaq)
DLM	Distributed Lock Manager
DMMP	Distributed Memory / Message Passing
DMSV	Distributed Memory / Shared Variables
DOE	US Department of Energy
ESS	Earth and Space Sciences Project – NASA
Full-Duplex	Simultaneous Bi-directional (*transmitting and receiving*) communications over the same physical communication link.
GFLOPS	1×10^9 FLoating Point OPerations per Second
GMMP	Global Memory / Message Passing
GMSV	Global Memory / Shared Variables
GSFC	Goddard Space Flight Centre
GUI	Graphical User Interface
HAL	Hardware Abstraction Layer
Half-Duplex	Bi-directional communications however transmitting or receiving is mutually exclusive.
HPCC	High Performance Computer Community

Word List	Definition
IEEE	Institution of Electrical and Electronics Engineers
IA32	Intel Architecture, 32 bit
IMPI	Interoperable MPI
LAM	Local Area Multicomputer
LAN	Local Area Network
LANL	Los Alamos National Laboratory
Linux	Unix style Operating system with open source code arrangement. Linux is a trademark of Linus Torvalds.
LIPS	Logic Inferences Per Second
LWP	Light Weight Process
MFLOPS	$1x10^6$ FLoating Point OPerations per Second
MIPS	Million Instructions Per Second
MPI	Message Passing Interface Standard
MPP	Massively Parallel Processors
NASA	National Aeronautical & Space Administration
NFS	Networked File System
NIC	Network Interface Card
NOW	Network of Workstations
NPB	NASA Parallel Benchmark
OA	Office Automation
OS	Operating System
POSIX	Portable Operating System Interface Standard uniX like
PVM	Parallel Virtual Machine
RED HAT	Is a registered trademark of Red Hat, Inc.
RFC	Request For Comments - Internet standard
RPC	Remote Procedure Call
SAN	System Area Network
SMP	Symmetric Multi Processing
SPP	Standard Parallel Port
SSI	Single System Image
Stream	Serial sequence of instructions or data
Telco	Telecommunications Company
TFLOPS	$1x10^{12}$ FLoating Point OPerations per Second
ULT	User Level Thread
W2K	Microsoft Windows 2000

Conventions used within this document

The following formatting, structure and comments are given below to assist with the reading and interpretation of this document:

Advantages – Highlight a particular word, statement, or section of a paragraph.

Distributed Systems – Keyword within the document.

(*For reference purposes*) – Information provided to assist explanation, but only if required by the reader.

Focus Box

Focus Boxes have been added to provide additional information such as supporting fundamental concepts and related material of interest.

1. Introduction

1.1. Brief History of Computing and Networking

The use of computers within our society has developed from the very first usage in 1945 when the modern computer era began, until about 1985 when computers were large and expensive. At that time, even mini-computers normally cost tens of thousands of dollars. So as a result, many organizations were limited to a handful of computers. These mini-computers were also limited in the way Organizations could utilize them as they operated independently from each other, as there was limited ability to interconnect them.

During the mid 1980's, however, two advances in technology began to change that situation. The first was the development of powerful microprocessors. Initially these were only 8-bit, but 16, 32 and eventual 64 bit have followed. Many of these had the computing power of mainframe systems (*i.e. the larger systems*) but at a fraction of the price.

The second development was the invention of the high speed LAN. Networking systems allowed tens to hundreds of machines to be interconnected in such a way that small amounts of information could be transferred between machines in only a few milli-seconds. As the speed of networks increased larger amounts of data could be transferred.

The net result of these two technologies is the ability to construct 'networks of machines' composed of a large number of CPU's interconnected via high-speed networks. These systems are commonly known or referred to as **Distributed Systems,** in contrast to the centralized systems consisting of a single CPU, local memory and I/O devices (*such as mainframes that have dominated prior to the development of low cost CPU's and high speed networks*). [5]

1.2. Parallel Processing

Parallel processing is the method of breaking large problems down into smaller constituent components, tasks or calculations that are solvable in parallel. Parallel processing has emerged as a key enabling technology in modern computing. The past several years have witnessed an ever-increasing acceptance and adoption of parallel processing, both for high performance **scientific computing** and for more "general purpose" applications. This has been a result of a demand for higher performance, lower cost, and sustained productivity. The acceptance of parallel processing has been facilitated by two major developments: massively parallel processors (MPPs) and the widespread use of distributed computing. [2]

MPPs are now the most powerful computers in the world. These machines combine a few hundred to a few thousand CPUs in a single large cabinet with hundreds of Giga-bytes of memory.

MPPs offer enormous computational power and are used to solve computational Grand Challenge problems such as global climate modelling and medicine design. As simulations become more realistic, the computational power required to produce them grows rapidly. Thus, researchers on the cutting edge turn to MPPs and parallel processing in order to get the most computational power possible.

The second major development affecting scientific problem solving is distributed computing. Distributed computing is a process whereby a set of computers connected by a network is used collectively to solve a single large problem. As more and more organizations have high-speed local area networks interconnecting many general-purpose workstations, the combined

computational resources may exceed the power of a single high-performance computer. In some cases, several MPPs have been combined using distributed computing to produce unequalled computational power.

The most important factor in distributed computing is **cost**. Large MPPs typically cost more than $10 million. In contrast, users see very little cost in running their problems on a local set of existing computers. It is uncommon for distributed-computing users to realize the raw computational power of a large MPP, but they are able to solve problems several times larger than they could using one of their local computers.

A common architectural element between distributed computing and MPP's is the notion of **message passing**. In all parallel processing, data must be exchanged between cooperating tasks. In this area of research, several paradigms have been experimentally evaluated including shared memory, parallelising compilers, and message passing. The message-passing model has become the paradigm of choice, from the perspective of the number and variety of multiprocessors that support it, as well as in terms of applications, languages, and software systems that use it.

This document focuses on an area of Distributed systems known as **clustering**, where the many interconnected machines work collectively to process instructions and data. This area can also be described as **parallel systems** (*or thought of as using distributed systems for parallel processing – as it encompasses both hardware, operating systems, middleware and software applications*).

As centralised machines dominate the low end of computing, for reference purposes section 2.5 outlines the benefits/drawbacks of distributed systems are compared with centralised systems.

1.3. Motivation

Where do we obtain the drive or necessity to build computers with massively parallel processing subsystems or large processing capabilities?

Fast and efficient computers are in high demand in many scientific, engineering, energy resource, medical, artificial intelligence, basic research areas and now the corporate environment.

Large-scale computations are often performed in these application areas and as such Parallel processing computers are needed to meet these demands.

My research in this area has shown that the requirements of a parallelised computer can be drawn from one of two areas and is derived from two separate requirements. These two areas are listed below:

1. **Production Oriented Computing** – This is where data is processed in real time applications where timeliness, availability, reliability, and fault tolerance is a necessity. This type of computing can require both serial and inherently or parallelisable applications.
2. **Non-Production research computing –** This is where a large amount of computer power is required to work on mainly inherently parallel problems, or parallelisable applications.

This document concentrates on the non-production oriented research side of computing, however the requirements of production oriented computing are detailed. My personal motivation in this area relates to Plasma research and hence will need to be able to develop algorithms to work with Clusters and MPPs.

1.3.1. Applications of Parallel Processing

Large-scale scientific problem solving involves three interactive disciplines as shown in Figure 1-1 below:

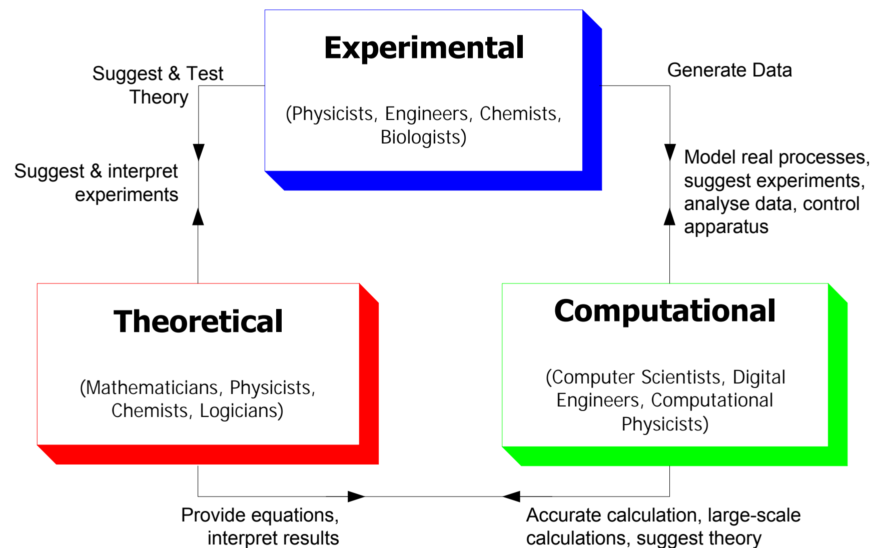


Figure 1-1 – Interaction among Experiment, Theory and Computation

As shown above theoretical scientists develop mathematical models that computer engineers solve numerically; the numerical results may then suggest new theories. Experimental science provides data for computational science, and the latter can model processes that are hard to approach in the laboratory. Using computer simulations has several advantages:

1. Computer simulations are far cheaper and faster than physical experiments.
2. Computers can solve a much wider range of problems than specific laboratory equipment can.
3. Computational approaches are only limited by computer speed and memory capacity, while physical experiments have many practical constraints.

Presented below are five distinct areas in which parallel processing can be grouped according to the required processing objective. Within each category several representative application areas have been identified. Further detail on these application areas can be found in Hwang [6]. It is interesting to note that the majority of the top 500 supercomputers are running one of the applications listed below. [29]

Predictive Modelling and Simulations

Multidimensional modelling of the atmosphere, the earth environment, outer space, and the world economy has become a major concern of world scientists. Predictive Modelling is done through extensive computer simulation experiments, which often involve large-scale computations to achieve the desired accuracy and turnaround time. Such numerical modelling requires state-of-the-art computing at speeds approaching 1 GFLOPS and beyond.

Applications include:

- Numerical Weather Forecasting.
- Semiconductor Simulation.
- Oceanography.
- Astrophysics (*Modelling of Black holes and Astronomical formations*).
- Sequencing of the human genome.
- Socio-economic and Government use.

Engineering Design and Automation

Fast supercomputers have been in high demand for solving many engineering design problems, such as the finite-element analysis needed for structural designs and wind tunnel experiments for aerodynamic studies. Industrial development also demands the use of computers to advance automation, artificial intelligence, and remote sensing of earth resources.

Applications include:

- Finite-element analysis.
- Computational aerodynamics.
- Remote Sensing Applications.
- Artificial Intelligence and Automation – This areas requires parallel processing for the following intelligence functions:
  - Image Processing
  - Pattern Recognition
  - Computer Vision
  - Speech Understanding
  - Machine inference
  - CAD/CAM/CAI/OA
  - Intelligent Robotics
  - Expert Computer Systems
  - Knowledge Engineering

Energy Resources Exploration

Energy affects the progress of the entire economy on a global basis. Computers can play an important role in: the discovery of oil and gas, the management of their recovery, development of workable plasma fusion energy and in ensuring nuclear reactor safety. Using computers in the high-energy area results in less production costs and higher safety measures.

Applications include:

- Seismic Exploration.
- Reservoir Modelling.
- Plasma Fusion Power.
- Nuclear Reactor Safety.

Medical, Military and Basic Research

In the medical area, fast computers are needed in computer-assisted tomography (CAT scan), imaging, artificial heart design, liver diagnosis, brain damage estimation and genetic engineering studies. Military defence need to use supercomputers for weapon design, effects simulation, and other electronic warfare. Almost all basic research areas demand fast computers to advance their studies.

Applications include:

- Medical Imaging
- Quantum Mechanics problems.
- Polymer Chemistry.
- Nuclear Weapon Design.

Visualization

Films such as The Matrix, Titanic, and Toy Story all made extensive use of cluster computers to generate the huge amount of imagery that was required to make these films. The cost of generating physical scenery for fantasy realms and historic recreations can be replaced by rendering on computers, not to mention the ability to violate the laws of physics.

Applications include:

- Computer-generated graphics, films and animations.
- Data Visualization.

World's Fastest Computers

The power of the last decade's supercomputers are now available in affordable desktop systems. However, the demand for ever-increasing computational power has not abated. For example, the US Department of Energy (*DOE*) established a program in 1996 called the Accelerated Strategic Computing Initiative (ASCI). The goal of ASCI is to enable 100 trillion floating point operations per second (teraflop/s or TFLOPS) sustained performance on key simulation codes by the year 2005. The DOE's aggressive schedule is due to the need to replace a test-based process for maintaining the nuclear stockpile with a simulation-based process. The change from test-based to simulation-based processes must be completed before the retirement of the aging population of nuclear scientists who can verify the conversion. At the same time industries from oil-and-gas exploration to aeronautics to pharmaceuticals are beginning to discover the enormous power inherent in high fidelity, three-dimensional simulations. [28]

In previous years, the world's most powerful computational machine was the ASCI White [29], an IBM-designed RS/6000 SP system that marked a breakthrough in computing at 12.3 teraflops. ASCI White uses MPP technology and is powered by 8,192 copper microprocessors, and contains 6 Tera bytes of memory with more than 160 TB of disk storage capacity.

Figure 1-2 – ACSI White at the Lawrence Livermore National Laboratory

IBM is currently in the process of building a new supercomputer for the DOE called Blue Gene/L. The computer is due for completion by 2005 and will operate at about 200 teraflops which has in previous years been larger than the total computing power of the top 500 supercomputers.

While the upper echelon of supercomputing relies on MPP technology, which will remain dominant for a long time, it is interesting to note that MPP and clusters are based around the same information theory and use familiar programming paradigms as discussed in this document, namely the message passing standard MPI.

As a point of comparison, at the time of writing, 63% of the world's top 500 supercomputers use MPP, 7% use SMP and 6% are cluster based. [29]

2. Architectures

2.1. Computer Classification Schemes

In general, digital computers may be classified into four categories, according to the multiplicity of instruction and data streams. Michael J. Flynn introduced this scheme or taxonomy for classifying computer organizations in 1966.

The essential element of the computing process is the execution of a sequence of instructions on a set of data.

Computer organizations are characterized by the multiplicity of the hardware provided to service the instruction and data streams. Listed below are Flynn's four machine organizations:

- **SISD** – Single Instruction stream / Single Data stream
- **SIMD** – Single Instruction stream / Multiple Data stream
- **MISD** – Multiple Instruction stream / Single Data stream (*No real application*)
- **MIMD** – Multiple Instruction stream / Multiple Data stream

Distributed systems can be classified by the **MIMD** organization but as each node can be classified as SISD both these schemes are presented here:

SISD – This organization represents most serial computers available today. Instructions are executed sequentially but may be overlapped in their execution stages (*using pipelining – refer to focus box below*). Most uniprocessor systems are pipelined. A SISD computer may have more than one functional unit in it. All the functional units are under the supervision of one control unit.

MIMD – Most multiprocessor systems and multiple computer systems can be classified in this category. An intrinsic MIMD computer implies interactions among the ***n*** processors because all memory streams are derived from the same data space shared by all processors.

MIMD can be summarised as follows:

- Each processor runs its own instruction sequence.
- Each processor works on a different part of the problem.
- Each processor communicates data to other parts.
- Processors may have to wait for other processors or for access to data.

If the ***n*** data streams were derived from disjointed subspaces of the shared memories, then we would have the so-called multiple SISD (MSISD) operation, which is nothing but a set of ***n*** independent SISD uniprocessor systems.

An intrinsic MIMD computer is tightly coupled if the degree of interactions among the processors is high. If not we consider them loosely coupled.

An example of the two:

Loosely Coupled multiprocessor systems do not generally encounter the degree of memory conflicts experienced by tightly coupled systems. In such systems, each processor has a set of input-output devices and a large local memory where it accesses most of the instructions and data. We refer to the processor, its local memory and I/O interfaces as a computer module.

Processes that execute on different computer modules communicate by exchanging messages through a message transfer system. (MTS). The degree of coupling in such a systems is very loose.

The determinant factor of the degree of coupling is the communication topology of the associated message transfer system. Loosely coupled systems are usually efficient when the interactions between tasks are minimal.

However due to the large variability of inter-reference times, the throughput of the hierarchical loosely coupled microprocessor may be too low for some applications that require fast response times. If high-speed or real time processing is required tightly coupled systems may be used.

An example of a loosely coupled multiprocessor machine is the Beowulf architecture, which is the subject of this document. Each processor has its own machine (*a separate PC*) connected through an Ethernet channel.

Tightly Coupled multiprocessor systems generally benefit over loosely coupled systems in performance however cost significantly more to achieve this performance.

Tightly coupled multiprocessors communicate through a shared main memory. Hence the rate at which processors can communicate from one processor to the other is on the order of the bandwidth of the memory. Tightly coupled systems can tolerate a higher degree of interactions between tasks without significant deterioration in performance. [6]

An example of a tightly coupled multiprocessor machine is a Transputer™ or a machine where two or more processors share the same memory such as with SMP (*Symmetric Multiprocessing*).

As the MIMD category includes a wide class of computers, in 1988 E. E. Johnson [9] proposed a further classification of such machines based on their memory structure (*global or distributed*) and the mechanism used for communications/synchronisation (*shared variables or message passing*). These are presented below:

- **GMSV** – Global Memory / Shared Variables.
- **GMMP** – Global Memory / Message Passing (*No real application*)
- **DMSV** – Distributed Memory / Shared Variables.
- **DMMP** – Distributed Memory / Message Passing.

It can be seen from the above classification that two of these architectures; DMSV and DMMP are loosely coupled and the remaining two; GMSV and GMMP are tightly coupled. As message passing is a key component to the Beowulf cluster technology, its architecture is best described or defined by the **DMMP** classification.

This leads us to the Flynn-Johnson classification of computer systems:

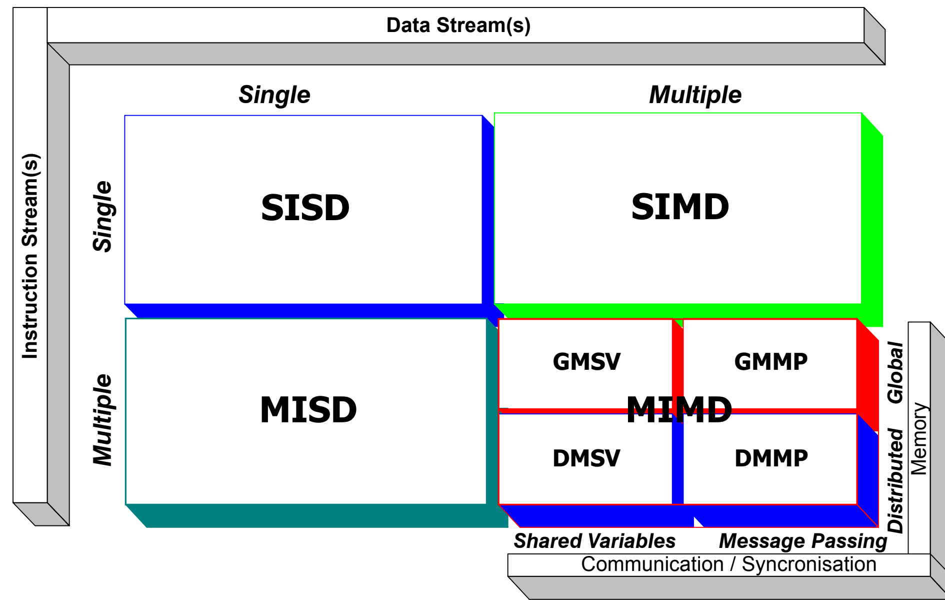


Figure 2-1 – The Flynn-Johnson Classification of Computer Systems

An alternate breakdown of the MIMD category proposed by Johnson is detailed by Tanenbaum and is shown in Figure 2-2 below. [5]

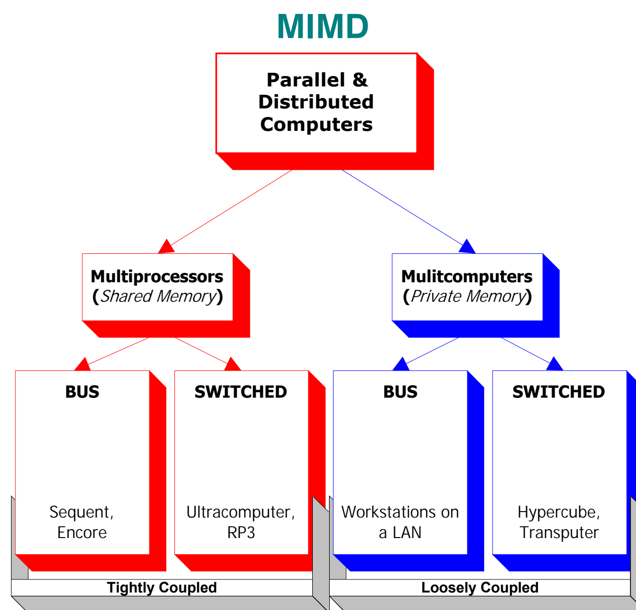


Figure 2-2 – Alternate Taxonomy of Parallel & Distributed Computer Systems

Tannebaum's taxonomy differs from the Flynn-Johnson model as it classifies a computer system based on the architecture of the interconnection network where the Flynn-Johnson model differentiates the programming model of the system. Both models are valid and present the similar information, however the Flynn-Johnson model is more descriptive as it describes the logical connections between processors rather than the physical connections and is thus less hardware specific and therefore is a better classification model.

As such further discussion is based of the Flynn-Johnson Classification model.

 Copyright © 2003

As stated above all SISD computers generally use pipelining. This is the case with all INTEL 40x86 and later CPU generations. As these processors will make up the cluster computer that will be tested the concept is explained below.

A pipelined computer performs overlapped computations to exploit ***temporal parallelism***.

Normally the process of executing an instruction in a digital computer involves four major steps:

1. Instruction Fetch (IF) from main memory.
2. Instruction Decoding (ID) identifying the operation to be performed.
3. Operand Fetch (OF) if needed in the execution.
4. Execution (EX) of the decoded arithmetic logic operation.

In a non-pipelined computer the four steps must be completed before the next instruction can be issued.

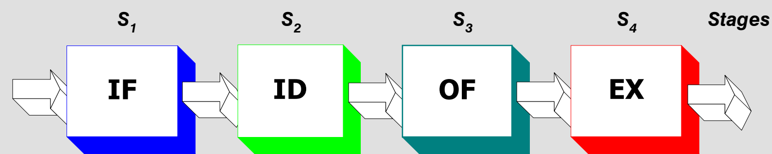


Figure 2-3 – A Pipelined Processor

In a pipelined computer, successive instructions are executed in an overlapped fashion, as illustrated in Figure 2-4. Four pipeline stages, IF, ID, OF and EX, are arranged into a linear cascade. The two space-time diagrams as depicted in Figure 2-4 and Figure 2-5 show the difference between overlapped instruction execution and sequentially non-overlapped execution.

This explanation depicts an Instruction Pipeline only – integer pipelines are also common with the INTEL Pentium containing two of them, potentially allowing two instructions to execute at the same time.

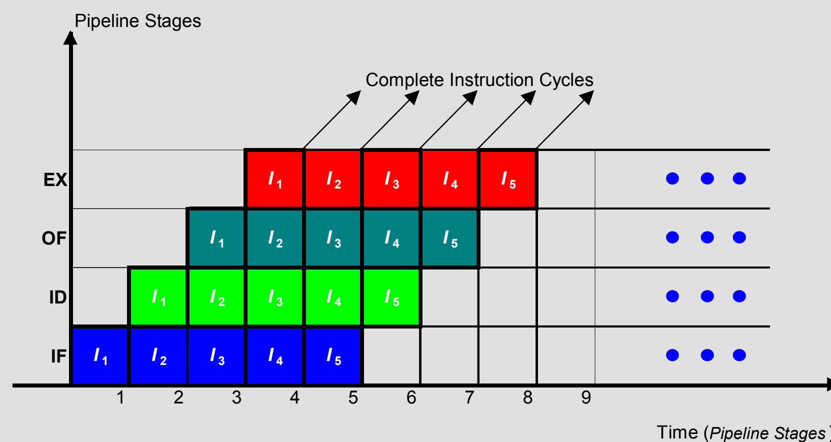


Figure 2-4 – Space-Time Diagram for a Pipeline Processor

PTO

Pipelining (cont).

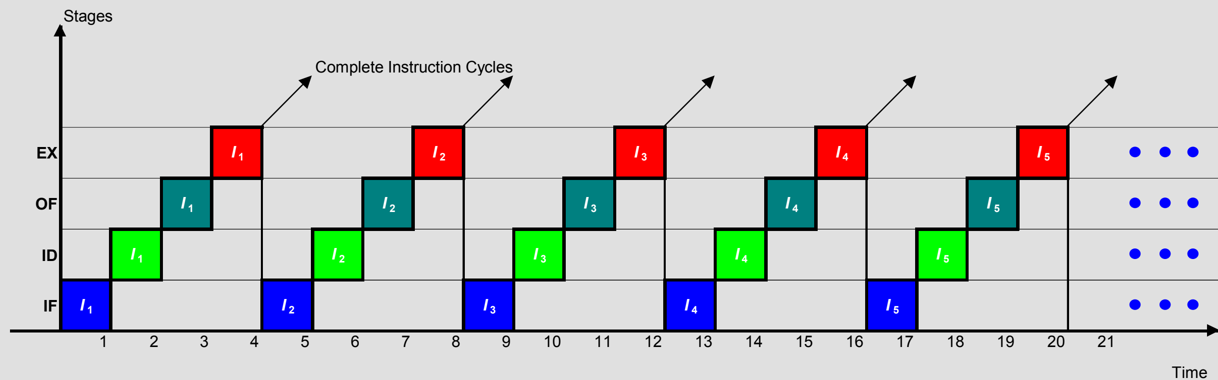


Figure 2-5 – Space-Time Diagram for a Non-pipelined Processor

Theoretically a ***k***-stage linear pipeline processor could be at most ***k*** times faster. However due to memory conflicts, data dependency, branch and interrupts, this speedup may not be achieved for out-of-sequence computations. [6] [7]

2.2. Cluster Computing Classification Schemes

Whilst the Flynn-Johnson DMMP model represents a number of different computer implementations, DMMP is also is a high level description for the area of computing that this document focuses on, which is the area of **Cluster Computing**.

Should we which to define DMMP machines further than the Flynn-Johnson model by way of classification of architectures, we would be defining actual implementation specific machines. One such classification is proposed by Thomas L. Sterling et al [12]:

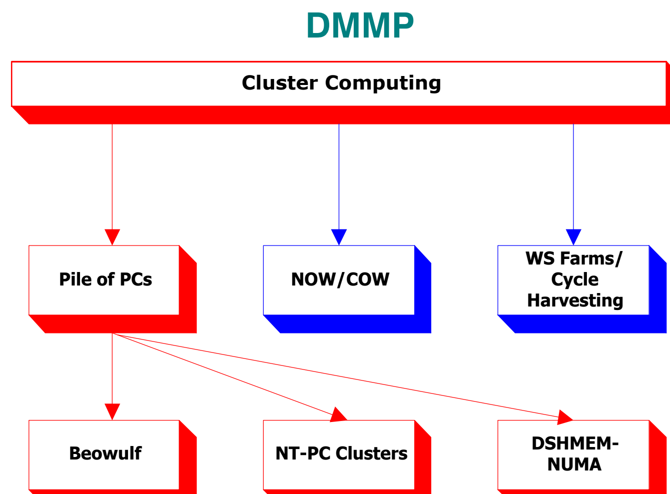


Figure 2-6 – Cluster Computer Classification Scheme

This document explicitly focuses on the Beowulf Cluster Computer, but does look at NOW (*Network of Workstations*) and COW (*Cluster of Workstations*) as well as NT-PC Clusters all of which are similar.

A cluster computer is a type of parallel or distributed processing system, which consists of a collection of interconnected stand-alone computers working together as a single integrated resource. The typical architecture of a cluster is shown in Figure 1-1 below. [31]

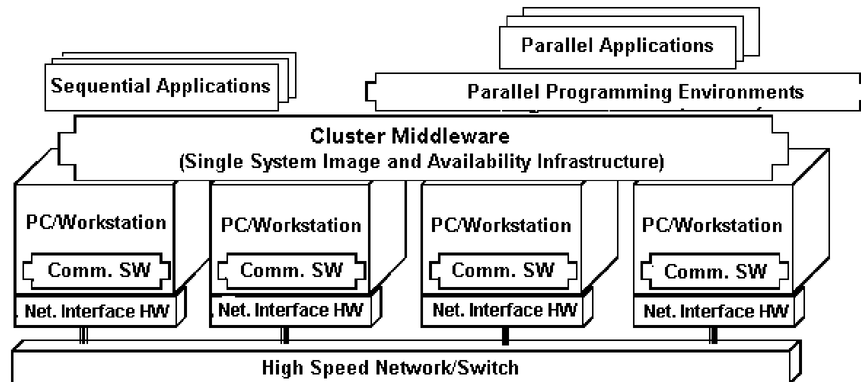


Figure 2-7 – Cluster Computer Architecture

The following are some prominent components of cluster computers:

- Multiple high performance computers (*PCs, Workstations, SMP servers*).
- Operating systems (*Layered or Micro-Kernel based*).
- High Performance Networks (*Switched network fabrics, Gigabit Ethernet*).
- Cluster Middleware (*Single System Image, and System Availability Infrastructure*).
- Parallel Programming Environments.
- Applications.

2.3. Beowulf

2.3.1. History

In the summer of 1994 Thomas Sterling and Don Becker, working at CESDIS under the sponsorship of the ESS project, built a cluster computer consisting of 16 Intel DX4 processors connected by channel bonded Ethernet. They called their machine Beowulf. The machine was an instant success and their idea of providing COTS (*Commodity off the shelf*) based systems to satisfy specific computational requirements quickly spread through NASA and into the academic and research communities. The development effort for this first machine quickly grew into what we now call the Beowulf Project. A non-technical measure of success of this initial project is the observation that researchers within the High Performance Computing community now refer to such machines as "Beowulf Class Cluster Computers". That is, Beowulf clusters are now recognized as genre within the HPC community. [4.2]

Thomas Sterling named the project 'Beowulf' after an old English tale of a legendary sixth-century hero from a distant realm who freed the Danes of Heorot by destroying the oppressive monster Grendel. As a metaphor, "Beowulf" was applied to this new strategy in high performance computing that exploits mass-market technologies to overcome the oppressive costs in time and money of supercomputing, thus freeing scientists, engineers, and others to devote themselves to their respective disciplines. Beowulf, both in myth and reality, challenges and conquers a dominant obstacle, in their respective domains, thus opening the way to future achievement. [12]

2.3.2. Overview

A Beowulf is distinct from other cluster computers as each node (*PC*) does not have keyboards, mice, video cards or monitors. For example, shown below is the LANL Beowulf machine named Avalon. Avalon was built in 1998 using the DEC Alpha chip, resulting in very high performance [24].

Figure 2-8 – Avalon Beowulf at LANL

Avalon

Beowulf is a machine usually dedicated to parallel computing, and optimised for this purpose. It gives a better **price/performance ratio** than other cluster computers built from **Commodity Components** and runs mainly software that is available at no cost. Beowulf has also more single system image features that assists users to utilize the Beowulf cluster as a single computing workstation.

Beowulf utilizes the **client/server** model of computing in its architecture as does many distributed systems (*with the noted exception of peer-to-peer*). All access to the client nodes is done via remote connections from the server node, dedicated console node or a serial console. As there is no need for client nodes to access machines outside the cluster, nor for machines outside the cluster to access client nodes directly, it is common practice for the client nodes to use private IP addresses such as the following IP address ranges:

10.0.0.0/8 or

192.168.0.0/16

The Internet Assigned Numbers Authority (*IANA*) has reserved these IP address ranges for private Internets, as discussed in RFC 1918 [13].

Usually the only machine that is connected to an external network is the server node. This is done using a second network card in the server node itself. The most common ways of using the system is to access the server's console directly, or either telnet or remote login to the server node from a personal workstation. Once on the server node, users are able to edit and compile source-code, and also spawn jobs on all nodes in the cluster. [1]

2.3.3. Classification

Beowulf systems can be constructed from a variety of distinct hardware and software components. In order to increase the performance of Beowulf clusters some non-commodity components can be employed. In order to account for the different types of hardware systems and to simplify references about Beowulf systems architecture, Radjewski et al [1] proposed the following simple classification scheme:

CLASS I BEOWULF:

Beowulf Class I machines are built entirely from commodity "off-the-shelf" parts. A Class I Beowulf is a machine that can be assembled from parts found in at least 3 nationally/globally circulated advertising catalogues.

The **advantages** of a Class I system are:

- Hardware is available form multiple sources (*low prices, easy maintenance*)
- No reliance on a single hardware vendor.
- Driver support from Linux commodity.
- Usually based on standards (*SCSI, EIDE, PCI, Ethernet, etc*).

The **disadvantages** of a Class 1 system are:

- Best performance is attained using CLASS II hardware.

CLASS II BEOWULF

A Class II Beowulf is simply any machine that does not pass the ease of availability certification test.

This class of Beowulf generally delivers higher performance at a cost penalty.

The **advantages** of a CLASS II system are:

- Performance is generally better than class one depending on the design and configuration (*i.e. the use of a 1.2GBps LAN will only improve performance for highly coupled applications*).

The **disadvantages** of a CLASS II system are:

- Driver support may vary.
- Reliance on single hardware vendor for parts and pricing.
- Generally more expensive than Class I systems.

Either class is not necessarily better than the other. It depends on the cost/performance requirements of the system that is required.

The classification system is only intended to assist in the discussion and specification of Beowulf systems.

This document focuses on Class I Beowulf systems as parts are highly available to construct clusters.

2.4. NOW/COW

Networks of Workstations (NOW) and Clusters of Workstations (COW) differ physically from Beowulf, as they are essentially complete PCs connected via a network. In most cases COWs are used for parallel computations at night, and over weekends when people are not actually using the workstations for every day work.

Another instance of this type of network is when people are using their PC but the central server is using a portion of their overall processing power and aggregating this across the network, thus utilising idle CPU cycles. In this instance programming a NOW is an attempt to harvest unused cycles on each workstation. Programming in this environment requires algorithms that are extremely tolerant of load balancing problems and large communication latency. Any program that runs on a NOW will run at least as well on a cluster. [1] [4.2]

Figure 2-9 below is the logo from the Berkley NOW project, that demonstrates the principle of building parallel processing computers from COTS components. [26]

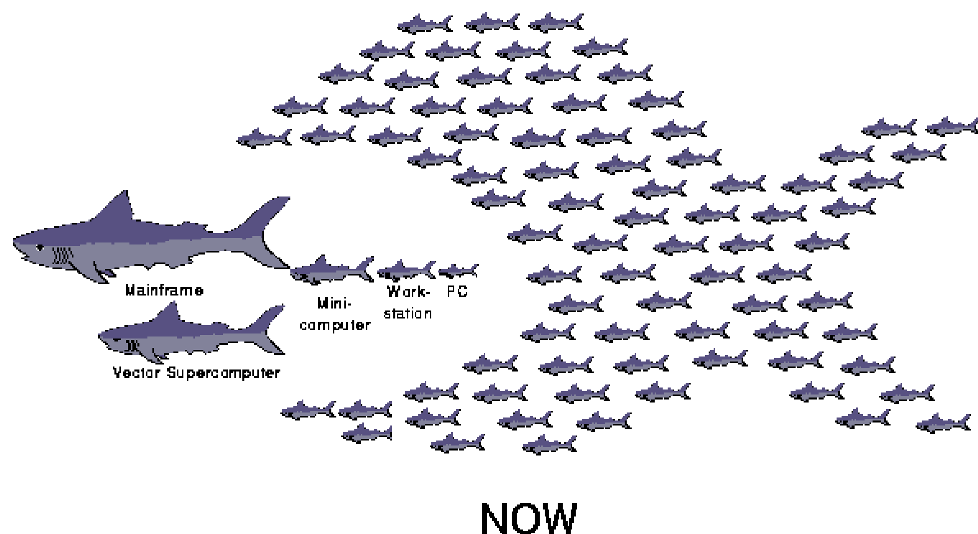


Figure 2-9 – Logo from the Berkley NOW Project

Previous research on cluster computing indicated that there are strong limits on the applications that will perform effectively on a COW or NOW. Beowulf improves on this philosophy by the ability to make the following assumptions which improves performance and/or the functionality:

- Dedicated processors.
- Private networks.
- High performance networks.
- Customisable system software.
- Systems Software can be imaged, ‘cloned’ or bootstrapped over the network.

2.5. Distributed vs. Centralized Systems

Distributed Systems have many advantages and are discussed here to outline the motivations of this area of work. Each advantage listed below is generally applicable to Beowulf clusters as they are encompassed by distributed systems.

Although distributed systems have their strengths, they also have their weakness, the majority of which are also outlined below. Despite these problems, it is considered that the advantages outweigh the disadvantages. [5]

ADVANTAGES

✓ **High Performance, High Throughput –** With multiple interconnected systems, high performance processing is attainable through the use of clustering where individual machines work as one, or appear as one transparently to the user to do work.

✓ **High Availability –** With multiple interconnected systems, the loss of any one system should have only minimal impact. Key systems and components can be replicated so that a backup system can take up the load after a failure.

✓ **High Reliability –** Should one machine fail, the system as a whole will not fail.

✓ **Economies of Scale –** The real driving force behind the trend towards decentralization is economic. Distributed systems have a better price performance ratio.

✓ **Expandability & Scalability –** When additional computer power is required, more CPU's can be added onto the network and the software 'cloned' or copied from another machine in a simple process. With centralised systems, upgrading a mainframe machine is cost prohibitive as well as complex.

✓ **Technology –** Today Clusters can be built out of a thousand CPUs yielding 20,000 MIPS, a comparable centralised system cannot currently be found. Further, to build a system of similar processing power would require a CPU that could execute an instruction in 0.005nsec (*50 picosecond*). From Einstein's theory of relativity we know that the speed of electrons have an upper limit of the speed of light (*practically much lower than this due to resistance and other factors*). This places an implication on the system to be a 1.5cm cube as we can only cover 1.5cm in 50picosec. Whilst current manufacturing processes will advance and produce faster and faster systems, parallelisation is the more cost effective alternative as we further get closer to the physical signal limit of the wavelength of light. Considering this, the sooner we embrace SMP and Cluster technology the sooner we will really see speed enhancements in our computer systems.

✓ **Suits Applications** – Some applications are inherently distributed in nature, such a geographically diverse operations or suit multiple processors working in parallel such as inherently parallel problems. Examples of these:

> **Geographically distributed:** such as a Factory where Robots and machines are distributed along the assembly line. It is considered optimal to have individual local computers to control and manage the immediate functions of each robot. Each computer is interconnected via a network where the overall coordination, control and statistics monitoring can be remotely administered.
>
> **Parallel:** such as a multi-threaded server application, running on a multi-tasking operating system, where each spawned thread can be allocated its own processor until such time as multiple threads are running on each processor. In this scenario the server will have a faster response time, as work can be done in parallel.

DISADVANTAGES

- **Available Software is Limited –** Little software is commercially available that offers functionality, reliability and can truly exploit a distributed environment. Additionally it can be quite expensive to develop new software applications for this architecture.
- **Communications Network can Introduce Unreliability –** It can potentially lose messages, become saturated, or too bandwidth limited to effectively serve the machines connected to it. To rectify these problems requires additional software and communications protocols and can require an expensive investment in new hardware.
- **Introduces Security Problems –** The ease of sharing data can be noted as an advantage of a distributed system. However as people and other systems can conveniently access data from any node in the system, it is equally possible to be able to access data that they have no business accessing. Security concerns in a distributed system are more far ranging that a centralised system as there is multiple points to, and of, attack and with distributed data systems, a greater risk of internal accidental data retrieval/deletion. This situation is possible to counter with rigorous lock-down procedures if followed by the network/systems integrators and administrators.

3. System Design

As previously stated; Fast, efficient, highly available and reliable computers are in high demand in many scientific, engineering, energy resource, medical military, artificial intelligence, research and now this requirement is seen in the corporate environment.

Whilst the requirements of each specific industry is different the key requirement is the same:

Design a cost effective super-computer.

With this chief requirement in mind, it is quite impossible to look past clustering as the solution.

This section approaches the system design requirements of a Beowulf cluster computer using a bottom-up design methodology as follows:

1. **Performance Requirements** – What performance is desired? What Performance is required? To effectively design a cluster computer system we need to know what application(s) will be running on the cluster and what is the intended purpose. This section reviews the theory behind parallel processing performance issues.
2. **Hardware –** What Hardware will be required or is available to build the Cluster Computer.
3. **Operating Systems –** What operating systems are available for Cluster Computing and what are their benefits.
4. **Middleware –** Between the Parallel application and the operating system we need a way of load balancing, and managing the system, users and processes, unless this is incorporated in to the Operating System.
5. **Applications –** What Software is available for Cluster Computing and how can applications be developed?

To assist in presentation of the material, this section has been structured following the bottom-up approach using the basic layered model of a cluster computer as shown in Figure 3-1 below. The hardware is shown at the bottom (*i.e. discussed first*) and the successive building blocks (*dependencies*) are discussed in later headings. This structure has been paid particular attention in the section on middleware as many packages can make up this layer.

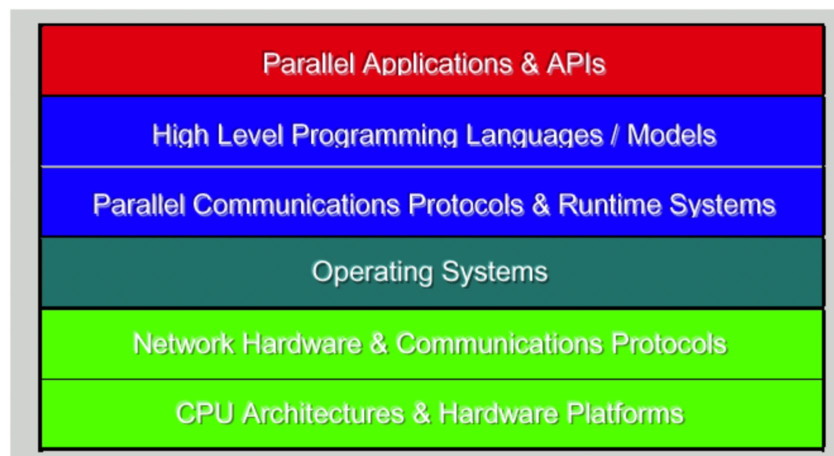


Figure 3-1 – Layered Model of a Cluster Computer

3.1. Performance Requirements

3.1.1. The Need for Performance Evaluation

One of the most important issues in parallel processing is how to effectively utilize parallel computers due to their inherent complexity. It is estimated that many modern supercomputers and parallel processors deliver only 10% or less of their peak performance potential in a variety of applications. Yet high performance is the very reason why super-computers are built.

The causes of performance degradation are many. Performance losses occur because of mismatches among:

1. **Hardware** – Idle processors due to conflicts over memory access & communications paths.
2. **Operating System** – Inefficient internal scheduler, file systems and memory allocation/de-allocation.
3. **Middleware** – Inefficient distribution and coordination of tasks, high inter-processor communications latency due to inefficient middleware.
4. **Applications** – Inefficient algorithms that do not exploit the natural concurrency of a problem.

The main steps leading to loss of parallelism, ranging from problem domain to hardware, are shown in Figure 3-2 below.

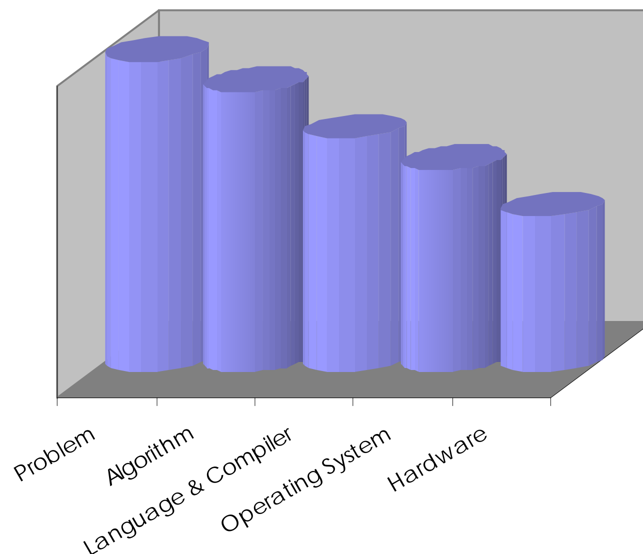


Figure 3-2 – Main Steps Leading to Loss of Parallelism

The **degree of parallelism** is the number of independent operations that may be performed in parallel. In complex systems, mismatches occur even among software and hardware modules. For example, the communications network bandwidth may not correspond to the processor speed or that of the memory.

Mapping applications to parallel computers and balancing processor loads is indeed a very difficult task.

Acquisition of the following knowledge motivates performance evaluation:

1. **Evaluation and comparison** of new and existing parallel computers.
2. **Increasing efficiency** in processor utilization by matching algorithms and architectures.
3. **Acquisition of information** for the design of new computers systems.

In general, computer performance can be studied by analytical means, by measurements and analysis. However with respect to parallel computers due to the complexity, analytical models are extremely complex and too many simplifying assumptions need to be made such that the end result does not accurately represent the operation of a parallel computer.

At present empirical results are the only methods that can be relied upon to assess the performance of parallel computers.

Performance benchmarking is another way to assess the performance of a particular parallel system. Benchmarking programs for Beowulf systems such as the LINPACK or the NASA developed NPB package are methods to compare Beowulf systems. Whilst benchmarks are useful, such work does not address the problem of how to tune the architecture, hardware, system software, and application algorithms to improve the performance. We need to gain an understanding of why a machine is performing the way it is through analysing how a machine performs across a wide range of benchmarks. [14]

3.1.2. Performance Indices of Parallel Computation

Within the research and development work that has been conducted in the field of parallel systems, some performance indices have been defined to measure the performance of parallel computation. Due to the level of complexity involved, no single measure of performance can give a truly accurate measure of a computer systems performance. Different indices are needed to measure different aspect. Some indices for global measurements follow [9] [14]:

1. **Execution rate** – The execution rate measures the machine output per unit of time. Depending on how machine output is defined, several execution rate measurements can be used. The concept of instructions per second, measured in MIPS (*million instructions per second*), is often used. While this measurement is meaningful for uniprocessors and multiprocessors it is inappropriate for SIMD machines, in which one instruction operates on a large number of operands. Additionally it does not differentiate between true results and overhead instructions. A good measure for arithmetic operations is **Megaflops (Mflops).** While this measure is appropriate for many numeric applications, it is not very useful for AI programs. A measure that can be used for logic programs and which can also be used for AI is LIPS (*logic inferences per second*).

2. **Speedup (S_p)** – The speedup factor of a parallel computation using p processors is defined as the ratio:

$$S_p = \frac{T_1}{T_p}$$

Where T_1 is the time taken to perform the computation on one processor and T_p is the time taken to perform the same computation on p processors. Hence S_p is the ratio of the sequential processing time to the parallel processing time. Normally the speedup factor is normally less than the number of processors because of the time lost to synchronisation, communication time, and other overheads required by the parallel computation:

$$1 \le S_p \le p$$

However there are cases where this does not apply. Refer to Super-Linear Speedups focus box below for details.

3. **Efficiency (E_p)** – The efficiency of a parallel computation is defined as the ratio between the speedup factor and the number of processors:

$$E_p = \frac{S_p}{P} = \frac{T_1}{pT_p}$$

Efficiency is a measure of the cost-effectiveness of computations.

4. **Redundancy (R_p)** – The redundancy of a parallel computation is the ratio between the total number of operations O_p executed in performing some computation with p processors and the number of operations O_1 required to execute the same computation with a uniprocessor:

$$R_p = \frac{O_p}{O_1}$$

R_p is related to the time lost because of overhead, and is always larger than 1.

5. **Utilization (U_p)** – The utilization factor is the ratio between the actual number of operations O_p and the number of operations that could be performed with p processors in T_p time units:

$$U_p = \frac{O_p}{pT_p}$$

6. **Quality (Q_p)** – The Quality factor of a parallel computation using p processors is defined as the equation:

$$Q_p = \frac{T_1^3}{pT_P^2 O_P}$$

Super-Linear Speedups

Superlinear speedups have been reported by Molavan [14] for non-deterministic AI computations, especially search operations. With respect to parallel processing with search operations, some paths leading to wrong results may be eliminated earlier than with sequential processing. Thus by avoiding some unnecessary computations, the speedup may increase by more than the number of processors.

One example of an algorithm that exhibits a super-linear speedup is A* - a widely known branch-and-bound algorithm used to solve combinatorial optimisation problems. [3.4.2.2][34]

3.1.3. Theoretical Performance of Parallel Computers

In this section, some basic mathematical models of parallel computation are presented. They are useful for understanding the limits of parallel computation.

The ideal speedup that can be achieved by a parallel computer with n identical processors working concurrently on a single problem is at most n times faster than a single processor. In practice, the speedup is much less, due to the many factors as outlined above.

The lower bound $\log_2 n$ is known as Minsky's conjecture, that the speedup is proportional to the logarithm of the number n of processors. This conjecture has its roots in analysis of data access conflicts assuming random distribution of addresses. These conflicts will slow all processes down to the point that quadrupling the number of processors only doubles the performance. However, data access patterns in real applications are far from random. Most applications have a good percentage of data access regularity and locality (*refer to the locality of reference focus box below*) that help improve the performance.

This law is considered to be pessimistic or the lower bound of the speedup. Depending on the application, real speed-up can range from **$\log_2 n$** to **n.** However, with respect to the upper bound **n**, in practice the formula (*and accompanying derivation*) below tends to model the performance of the upper bound more accurately.

Consider a computing problem, which can be executed by a uniprocessor in:

Unit time $T_1 = 1$

Let f_i be the probability of assigning the same problem to ***i*** processors

Each processor is working equally with average load $d_i = 1/i$ per processor

Assume **equal probability** of each operating mode using i processors i.e. $f_i = 1/n$, ***for n** operating modes: I = 1, 2,.....n*

The average time required to solve the problem on an n-processor system is given below, where the summation represents ***n*** operating modes:

$$T_n = \sum_{i=1}^{n} f_i . d_i = \frac{\sum_{i=1}^{n} \frac{1}{i}}{n} \qquad \text{Equation 3-1}$$

The average speedup ***S*** is obtained as the ratio of $T_1 = 1$ to T_n; that is,

$$S = \frac{T_1}{T_n} = \frac{n}{\sum_{i=1}^{n} \frac{1}{i}} \le \frac{n}{\ln n} \qquad \text{Equation 3-2}$$

The **Weighted Harmonic Mean Principle** relates the execution rate to a weighted harmonic mean. Let T_n be the time required to compute ***n*** tasks of ***k*** different types. Each type consists of n_i tasks requiring t_i seconds each, such that:

$$T_n = \sum_{1}^{k} n_i . t_i$$

And

$$n = \sum_{1}^{k} n_i$$

By definition, the execution rate ***R*** is the number of events or operations in unit time, so:

$$R_n = \frac{n}{T_n} = \frac{n}{\sum_{1}^{n} n_i . t_i}$$

Let us define f_i as the fraction of results generated at rate R_i, and $t_i = 1/R_i$ as the time required to generate a result. Then:

$$R_n = \frac{1}{\sum_{1}^{k} f_i . t_i} = \frac{1}{\sum_{1}^{k} f_i / R_i} \qquad \text{Equation 3-3}$$

Where

$f_i = n_i / n$

$$\sum_i f_i = 1$$

$$R_i = 1/t_i$$

Equation 3-3 represents the basic principle in computer architecture some times loosely referred to as a **bottleneck**, or the weighted harmonic mean principle. The importance of this principle and Equation 3-3 is the following:

If a single rate is out of balance** (i.e. lower than the others)**, then this will dominate the overall performance of the machine.

This is the case for designing general-purpose Beowulf cluster client-nodes with homogenous hardware. This is as opposed to the use of heterogeneous collections of nodes communicating over various different networks. A heterogeneous network and node base are systems that are out of the range of mathematical modelling or determining the performance prior to implementation.

Amdahl introduced Amdahl's law in 1967. This law is a particular case of the weighted harmonic mean principle. Two rates are considered:

1. The high, or parallel execution rate R_H
2. The low, or scalar execution rate R_L

If f denotes the fraction of results generated at the high rate R_H and 1- f is the fraction generated at the low rate, then the Equation 3-3 becomes:

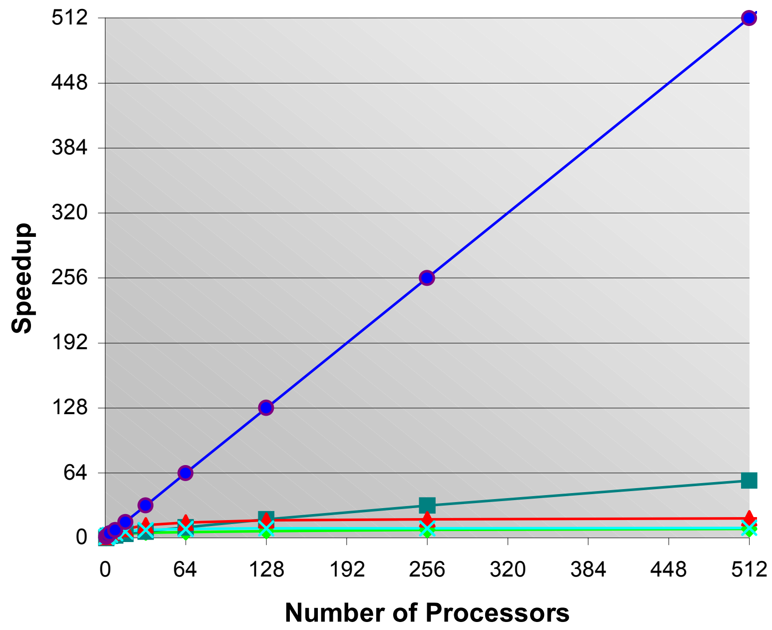
$$R_f = \frac{1}{f/R_H + (1-f)/R_L} \qquad \text{Equation 3-4}$$

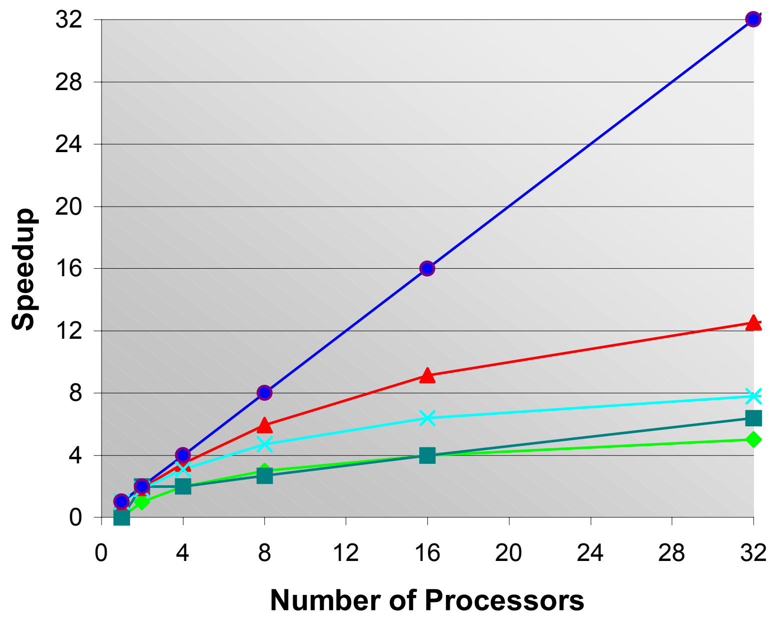
This formula is known as Amdahl's law. It is useful for analysing system performance that results from two individual rates of execution, such as vector or scalar operations or parallel or serial operations. It is also useful for analysing a complete parallel system in which one rate is out of balance with the others. For example, the low rate may be caused by I/O or communication operations, and the high rate may be caused by vector, memory, or other operations.

Amdahl's law can also be applied to the execution of a particular program. From the law we see that a small fraction f of inherently sequential or unparalisable computation severely limits the speedup that can be achieved with p processors. Consider a unit time task, for which the fraction f is unparalisable (*so it takes the same time* f *on both parallel and sequential machines*) and the remaining $1-f$ is fully paralisable [so it runs in time $(1-f)/p$ on a p-processor machine].

Of note is the fact that with Amdahl's formula when f=0 this represents the ideal case of and n times speedup.

Hence to plot these results will give an indicative idea of what the performance a Beowulf cluster will produce depending on the number of processors that are used. In practice this will vary, however, if a good communications network is chosen the speedup as shown will be achieved. Shown below are three views from the same data derived from the principles above. The three views show; the over all trend and, a low number of processors.


Speedup
0
64
128
192
256
320
384
448
512
Number of Processors


Speedup
0
4
8
12
16
20
24
28
32
Number of Processors

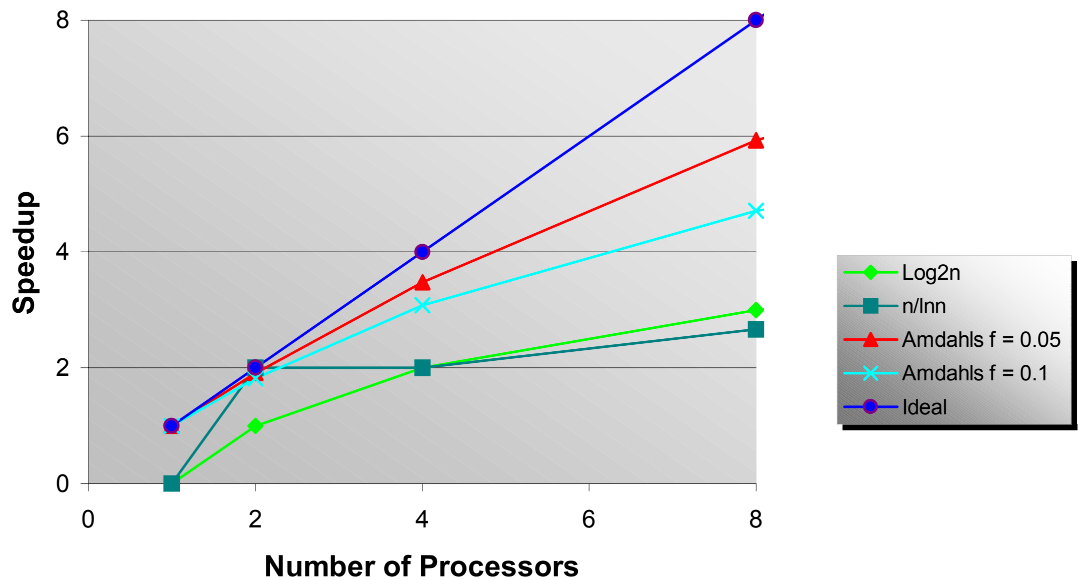


Figure 3-3 – Various Estimates of an n-processor Speedup

From the above analysis and plot it can be seen why typical commercial multiprocessor systems consist of only two or four processors. Designers can achieve a better speed up with a smaller number of fast processors than a larger number of slower processors.

One important note that should be made that is closely related to Amdahl's law is the fact that some applications lack inherent parallelism, thus limiting the speedup that is achievable when multiple processors are used. [6] [9] [14]

Locality of Reference

The success of memory hierarchy is based upon assumptions that are critical to achieving the appearance of a large, fast memory. The foundation of these assumptions is termed **locality of reference**.

There are three components of the locality of reference, which coexist in an active process which are:

- **Temporal –** A tendency for a process to reference in the near future the elements of instructions or data referenced in the recent past. Program constructs that lead to this concept are loops, temporary variables, or process stacks.
- **Spatial –** A tendency for a process to reference instructions or data in a similar location in the virtual address space of the last reference.
- **Sequentiality –** The principle that if the last reference was $r_j(t)$, then there is a likelihood that the next reference is to the immediate successor of the element $r_j(t)$.

It should be noted that each process exhibits an individual characteristic with respect to the three types of localities. [6] [18]

3.1.4. Performance Analysis and Measurement

To understand complex interactions of many factors contributing to the performance of parallel processors, performance measurement and analysis must be done. These are achieved with specialised hardware and software. The traditional approach for measuring the performance of a program running on a parallel computer is to time its execution from beginning to end and compute the Megaflops (Mflops) and now more recently in **Gigaflops (Gflops)** and **Teraflops (Tflops)**. However timing alone is inadequate to fully characterize the program's behaviour. It is also necessary to detect and record data related to the dynamic occurrences of hardware, system software, and application programming events and interactions.

Measurements can be made on both hardware and software in both static and dynamic ways. This information in this document concentrates on measuring Hardware and Software performance in a dynamic way.

Some factors are discussed below that classify the types of performance measurement available.

Hardware and Software Measurements

Hardware measurements focus on the physical events taking place within various machine components. They are usually made with hardware monitors connected directly to hardware devices, and are triggered by certain signals. Measuring devices need to be designed such that they do not interfere with the systems being measured. For example when designing microprocessors, it is important to make critical signals accessible for measurement. In this instance chip manufacturers such as Intel have specific hardware registers that are able to measure out-of-band performance. Additionally measurement instrumentation has been integrated into the design of a few MPP systems.

Software Measurements are directed more towards logical events that occur during program execution. These include individual timing of routines or tasks, and observations of events related to loop-level parallelism, synchronisation, and other measurements. Special software is needed to start, accumulate, and interpret records.

Static and Dynamic Analysis

The goal of *static analysis* is to extract, quantify, and analyse various characteristics of benchmark code and it's mapping to a particular architecture at compile time without actually executing the code. Through static analysis we can measure the effectiveness of the compiled portion of a system. In general, the purpose is to detect whether or not the software component of the systems is responsible for poor performance. Conclusions may be reached about the appropriateness of the hardware model, load algorithms, code restructuring, mapping and allocation. The Paraphase compiler, developed at the University of Illinois, illustrates how static analysis tools are used to perform tradeoffs.

The goal of *dynamic analysis* is to monitor the hardware and software while the application program is running. Many performance losses are associated with inefficiencies in processor scheduling, load balancing and data communication. [14]

3.1.5. Practical Performance of Parallel Computers

While the mathematical models of section 3.1.3 serve to educate on the general performance of parallelisation of sequential problems as well as inherently paralisable applications, it does not take in to account Super-linear speedups and the high performance achievable with parallel computers when dealing with search algorithms or highly parallel applications.

Beowulf systems are particularly well suited to, and may easily exploit process-level parallelism that is exposed by running multiple, independent processes. [12]

When applicable in an application, process-level parallelism is often the easiest way to achieve speedup in parallel computations. The only requirement for process-level parallelism is that there must exist sequential code or codes that need to be run many times to produce a result. This is otherwise known as **parametric computing**.

In general, the mathematical models discussed strictly dictate that to maximise speedup, parallel systems should be built out of a small number of high performance processing elements. Whilst this is true for many applications, there are some notable examples where this is not the case.

Eadline et al [38.1] describe their results with respect to I/O-dominant applications in a general law that states:

"For two given parallel computers with the same cumulative CPU performance index, the one which has slower processors has better performance for I/O-dominant applications"

The work shows that for I/O dominant applications it is better to have more data blocks being processed at one time (*by larger number of slower processors*) than a smaller number of fast processors.

Eadline's work demonstrates that applications perform differently and performance cannot be modelled effectively with mathematics at this stage, reinforcing the use of testing and experimentation. Although this is the case, the popularisation of parallel computers will be through the development of general-purpose, cost effective, fault-tolerant parallel clusters such as Beowulf that can be tuned to a particular algorithm's natural requirements or used in a standard configuration.

3.2. Hardware Platforms

3.2.1. CPU

A choice of CPU should be made from two families: Intel x86 [IA32] compatible (*such as the Pentium4*) or Compaq Alpha systems [Formerly DEC]. Alternate vendor CPUs are supported by Linux and cluster-able operating systems (*such as the IBM Power PC chip and the SUN SPARC*), however there is only limited support and accompanying software distributions available for these architectures. In general, Beowulf systems are mainly built with Intel or Alpha architectures. Intel based systems are considered as commodity systems because there are multiple sources (*Intel, AMD, Cyrix*) and obviously ubiquitous. Compaq Alpha on the other hand, is a clear performance winner, but does represent a single source part, and therefore can be hard to source at a good price.

Within the Intel based systems, the Pentium 3 and 4 [23] have the best floating point performance and they support SMP motherboards. In addition, Intel has confirmed that the Pentium 4 micro-architecture will scale to 10Ghertz over the next decade.

3.2.2. Symmetric Multiprocessing

As previously discussed, Symmetric Multiprocessing (SMP) are tightly coupled multiprocessors that communicate through a shared main memory. Hence, the rate at which processors can communicate from one processor to the other is on the order of the bandwidth of the memory. Depending on the hardware a small local cache may be used in each processor.

Symmetric Multiprocessor boards are commonly used in Beowulf clusters. In architectural terms by adding SMP to cluster nodes we are using tightly coupled nodes in a loosely coupled interconnection methodology.

The main advantages of using SMP in a cluster are:

1. Lower price per performance.
2. Greater communication speeds between processors on the same board (*usually a dual-processor configuration is used*).
3. Requires less space and draws less power.

The first advantage is important when building a large cluster. By using dual CPU SMP systems across the whole cluster, only 50% of the network cards, cases, power supplies, and motherboards are required. The only cost increase is a more expensive, SMP motherboard, however the cost cuts far outweighs the increase.

If it is decided that SMP systems will not be used across the cluster, it is worthwhile to consider an SMP server. The Topcat Research project [11] at the University of Technology, Queensland has shown that it is quite common for a Beowulf master node to run at load of two or higher, fully utilising both SMP CPUs. As the master node serves file systems to the client nodes, it is important that the NFS server has enough CPU cycles left to perform its duties. A workaround is to lower the nice level of nfsd (*NFS Daemon*), but to have spare CPU cycles is also important. If during the operation of the cluster the server node will be loaded by users, it should be considered to use a fast SMP system.

With SMP each node relies on the Linux (*or the operating system of choice*) internal scheduler, which determines how the CPUs get shared. The user cannot (*at this point*) assign a specific task to a specific SMP processor. The user can however, start two independent processes or a threaded process on a particular SMP node and expect to see a performance increase over a single CPU system.

Symmetric Multiprocessing

A typical SMP block diagram for an Intel system is shown below. Designed for scalability and availability, 8-way Pentium III Xeon processor-based servers provide high performance and price/performance for mid-range and back-end enterprise solutions – at a fraction of the cost of proprietary RISC-based solutions [23.1]

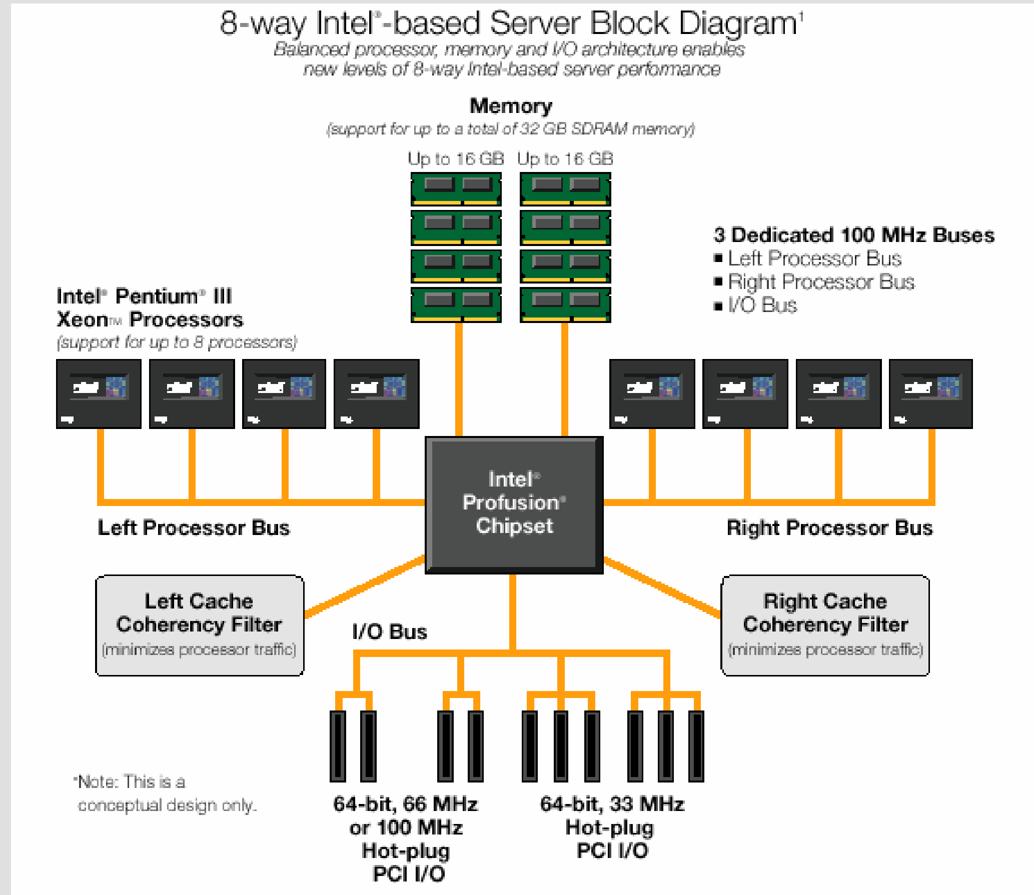


Figure 3-4 – Intel SMP Server Block Diagram

Aside from the hardware architecture, the Software utilization of an SMP machine is through the use of threads. Linux supports POSIX threads as do most other operating systems, but most OS's generally have their own enhanced thread packages.

3.2.3. Basic Network Architectures

The design of a network system to suit a Beowulf cluster must take into account what type of applications will be running on a cluster such as:

- Degree of intercommunication required between each node during the running of an application.
- The effect of processor intercommunication on the throughput of the Cluster.

Once this is determined, it is a matter of deciding on the performance level required before selecting an implementation topology and then technology.

Given the internal processor and memory structures of each node, distributed memory architectures are primarily characterized by the network used to interconnect the nodes.

This network is usually represented as a graph, with:

- *Vertices* corresponding to *processor-memory nodes*.
- *Edges* corresponding to *communications links*.

If *unidirectional* communications links are used then *directed-edges* are used.

If *bi-directional communications* links are used then *bi-directional edges* are used. While bi-directed edges signify bi-directional communication this represents both half-duplex and full duplex communications.

Important parameters of an interconnection network include:

1. **Network Diameter** – The longest of the shortest paths between various pairs of nodes, which should be relatively small, is the latency is to be minimized. The network diameter is more important with store-and-forward routing (*when a message is stored in its entirety and retransmitted by intermediate nodes*) than with wormhole routing (*when a message is quickly relayed through a node in small pieces*).
2. **Bisection (band)width** – The smallest number (*total capacity*) of links that need to be cut in order to divide the network into two sub networks of half the size. This is important when nodes communicate with each other in a random fashion. A small bisection (band)width limits the rate of data transfer between the two halves of the network, thus affecting the performance of communications intensive algorithms.
3. **Vertex or node Degree** – The number of communications ports required of each node, which should be a constant, independent of network size if the architecture is to be readily scalable to larger seizes. The node degree has a direct effect on the cost of each node, with the effect being more significant for parallel ports containing several wires or when the node is required to communicate over all ports at once.

The following table lists these three parameters for some commonly used interconnection networks. This table is by no means exhaustive but gives an idea of the variability of parameters across different networks. [9]

Network Topology	No. of Nodes	Network Diameter	Bisection Width	Node Degree	Local Links
1D Mesh (linear array)	k	k–1	1	2	Yes
1D Torus (ring, loop)	k	k/2	2	2	Yes
2D Mesh	k^2	2k–2	k	4	Yes
2D Torus (k-ary 2 cube)	k^2	k	2k	4	Yes[1]
3D Mesh	k^3	3k-3	k^2	6	Yes
3D Torus (k-ary 3 cube)	k^3	3k/2	$2k^2$	6	Yes[1]
Pyramid	$(4k^2-1)/3$	$2\log_2 k$	2k	9	No
Binary Tree	$2l-1$	$2l-2$	1	3	No
4-ary hypertree	$2^l(2^{l+1}-1)$	$2l$	2^{l+1}	6	No
Butterfly	$2^l(l+1)$	$2l$	2^l	4	No
Hypercube	2^l	l	2^{l-1}	l	No
Cube-connected cycles	$2^l l$	$2l$	2^{l-1}	3	No
Shuffle-exchange	2^l	$2l-1$	$\geq \frac{2^{l-1}}{l}$	4 unidir	No
De Bruijn	2^l	l	$\frac{2^l}{l}$	4 unidir	No

[1] With Folded Layout

Table 3-1 – Topological Parameters of Selected Interconnection Networks

While direct interconnection networks of the type shown in Table 3-1 have led to many important classes of parallel systems, bus based architectures such as Ethernet dominate the smaller scale parallel machines. However since a single bus can quickly become a performance bottleneck as the number of processors increases, a variety of multiple-bus architectures and hierarchical schemes are available for reducing bus traffic or increasing the internode bandwidth.

Due to the ever-dropping prices of 100 Mbps Ethernet switches/NICs, fully meshed, hypercube and the network topologies as outlined in Table 3-1 above are no longer economical for Clustering.

Figure 2-7 depicts a typical Ethernet bus configuration interconnecting each Beowulf Cluster node. 100 Mbps switched, full duplex Ethernet is the most commonly used network in Beowulf systems, and gives almost the same performance as a fully meshed network. Unlike standard Ethernet where all computers connected to the network compete for the cable and cause collisions of packets, switched Ethernet provides dedicated bandwidth between any two nodes connected to the switch.

As outlined in the following section should higher internode bandwidth be required, we can use channel bonding to connect multiple channels of Ethernet to each node.

3.2.3.1. Network Channel Bonding

Overview

One of the goals of a Beowulf system is to utilize scalable I/O using commodity subsystems. For scaling network I/O a method was devised by NASA to join multiple low-cost networks into a single logical network with higher bandwidth. The only additional work over a using single network interface is the computationally simple task of distributing the packets over the available device transmit queues.

Initially Beowulf clusters were designed to implement this method using two 10Mbps Ethernets, later expanding it to three channels. Recent Beowulf systems have parallel 100Mpbs Fast Ethernet channels or the latest Gigabit Ethernet channels.

Except for the performance increase, channel bonding is transparent to the applications running on the cluster.

A by-product of the use of channel bonding is the redundancy of having two or more networks interconnecting each node. Through the use of modified communications protocols this network configuration can add fault tolerance and greater reliability to a cluster. [36]

A user-level control program called '**ifenslave**' is available from NASA's website [4]. Refer Appendix B – Channel Bonding for the source listing and commentary.

The system-visible interface to "channel bonding" is the 'ifenslave' command. This command is analogous to the 'ifconfig' command used to set up the primary network interface. The 'ifenslave' command copies the configuration of a "master" channel, commonly the lowest number network, to slave channels. It can optionally configure the slave channels to run in a receive-only mode, which is useful when initially configuring or shutting the down the additional network interfaces.

Implementation

To minimize protocol overhead and allow the latest possible load-balancing decision, Beowulf channel bonding is implemented at the device queue layer in the Linux kernel, below the IP protocol level. This has several advantages:

- Load-balancing may be done just before the hardware transmit queue, after logical address (*e.g. IP address*) and physical address (*e.g. Ethernet station address*) are added to the frame.
- Fragmented packets, usually large UDP/IP packets, take advantage of the multiple paths.
- Networks that fail completely first fill their hardware queues, and are subsequently ignored.

This implementation results in the following limitations:

- All connected machines must be on the same set of bonded networks. The whole cluster must be channel bonded as communication between a channel bonded node and a non-channel bonded node is very slow if not impossible.
- It's difficult to detect and compensate if one network segments (*splits in the middle*).
- Channel bonding shutdown is not as simple. The safest way to restore single channel performance is to either reboot each system or use the network manager (*part of the control panel*) to shutdown and restart each interface.

The software implementation is divided into two parts, the kernel changes needed to support channel bonding, and the user-level control program 'ifenslave'.

- The user-level control program 'ifenslave' is compiled from the source ifenslave.c.

- Apply the kernel patch (*linux-2.0.36-channel-bonding.patch from Paralogic [38]*), run xconfig and enable Beowulf Channel bonding then rebuild and install the kernel.

Typical requirements of the hardware configuration is shown as follows:

- ***n*** Ethernet NICs per system.
- ***n*** hubs (*one for each channel*) OR ***n*** switches (*one for each channel*) OR a switch that can be segmented into ***n*** virtual LANs.

Once a cluster is channel bonded it can be tested by running the network performance benchmark **netperf** or similar. [4.1] [38]

3.2.4. Node Interconnection Technologies

A node interconnection technology is the hardware and software interface that connects each node and allows node intercommunication, as distinct from the interconnect topology as discussed in 3.2.3. This document concentrates on Class 1 Beowulf clusters, which has implications for the node interconnect used [*Refer to 2.3.3 and Appendix 8.1 for details*].

Some design considerations for the selection of a node interconnect are as follows:

- **Linux support:** yes/no, kernel driver or library (*kernel drivers are preferred*).
- **Maximum bandwidth:** The higher the better.
- **Minimum latency:** The lower the better.
- **Available as:** Single Vendor / Multiple Vendor Hardware.
- **Interface port/bus used:** High performance, included as a standard node port and matched bandwidth to the dedicate node network fabric bandwidth.
- **Network structure:** Bus/Switched/Topology.
- **Cost per machine connected:** The lower the better.

A full analysis of the available node interconnection technologies are presented in Appendix 8.1.

As previously stated 100 Mbps switched, full duplex Ethernet is the most commonly used network in Beowulf systems as it:

1. Gives almost the same performance as a fully meshed network.
2. Is one of the most commonly available networks.
3. Cost effective.

As such it will be used for all testing and experimentation in this document. Refer to Appendix 8.1 for technical details.

3.3. Operating Systems

3.3.1. General

A modern operating system provides two fundamental services for users:

1. **Provides a Virtual Machine** – A programming and user interface that hides the obscurities of using/programming the various hardware components that make up a system.
2. **Shares hardware resources among users or executing processes** – One of the most important resources is the processor. In all multitasking operating systems (*which are the only type of operating system dealt with in this document*) the computer performs multiple tasks at once by scheduling jobs using various scheduling algorithms which allows many processes and threads to run concurrently in pseudo parallel (*as with the case of a one processor machine*) or parallel (*as with the case with an SMP machine*).

These services are provided to the user by the operating system through the following operating system functions:

1. **Process Management** – The operating system must allocate resources to processes, enable processes to share and exchange data, protect the resources of each process from one another, enable synchronisation among processes, and start, stop and suspend processes according to different priorities and scheduling algorithms.
2. **Memory Management** – The operating system needs to handle the relocation, protection, sharing, logical and physical organization of memory through various schemes.
3. **File Systems Management** – The primary consideration for an operating system is the need to manage local files including structure, organization, naming, access, allocation and locking.
   Distributed systems as detailed in this document are a special case where the operating system and/or an additional layer of middleware is required to manage file systems across a network, where files can be distributed across disjointed memory spaces.
4. **I/O** – An operating system needs to interface with many different forms of hardware to collect data and communicate results. This requires integration, logical structuring, control, intercommunication and programming abilities. I/O can be roughly grouped into three subcategories:
    1. **Human Readable –** i.e. Video display terminals, printers, keyboard, mouse and speakers.
    2. **Machine Readable –** i.e. disk drives, digital camera and other devices.
    3. **Communication –** i.e. Modems & network cards.

3.3.2. Towards Parallel Systems

The operating system is a critical factor to the development of parallel systems. In all Beowulf systems, a required software layer called middleware augments the operating system. Middleware handles source code programming, compilation, and parallel execution and runtime requirements.

However, the latest generation of operating systems are emerging with support for parallel systems paradigms by integrating middleware functions into the operating system. Three of the most commonly used operating systems (*Microsoft Windows, SUN Solaris and Redhat Linux*) are including kernel level support for parallel programming, inherent parallelisation and load

balancing characteristics. This is the next leap forward in parallel systems as it caters for the corporate environment and will eventually pervade into desktop machines.

The main issues that these so-called next generation operating systems need to address (*and currently do so to varying degrees*) to exploit the cluster environment further than scientific computing capabilities, are listed below: [32]

- **Failure Management** – These include fault tolerant or highly available solutions. Fault tolerant solutions will recover transparently to an application any internal node or component failure. In contrast, highly available solutions offer a high probability that all nodes will be in service. However, with highly available solutions any transactions lost in the process of a node or component failure need to be handled or recovered at the applications layer (*such as with atomic transactions*).
- **Load Balancing** – Clusters require an effective capability for balancing the load among available computers. This includes the ability for the system to cater in terms of dynamic task scheduling and reconfiguration for the addition or removal of nodes from the cluster.
- **Parallelising Computation** – In some cases, as with the case with scientific computing, effective use of a cluster requires executing software from a single application in parallel. There are three general approaches to this problem:
    1. **Parallelizing Compiler** – A parallel compiler determines at compile time, which parts of an application can be executed in parallel. These are then split off and assigned to different computers in the cluster. Performance depends on the nature of the problem and how well the compiler is designed.
    2. **Parallel and Cluster Aware Applications** – In this approach the programmer write an application from the outset to run on a cluster, and uses message passing to move data, as required, between nodes. This places a high burden on the programmer but may be the best approach for exploiting clusters for some applications. Another technique that can be used is to write a multi-threaded cluster aware application and let the cluster manage the threads.
    3. **Parametric Computing** – This approach can be used if the problem is inherently parallel, where an algorithm or program is executed a large number of times, each time with a different set of starting conditions or parameters. A good example is simulation programs, however tools are required to organise, run and manage jobs in an orderly manner.

These next generation Operating Systems use one of the following two cluster implementation models, currently used in the computer industry – the shared device and shared nothing models which are described below:

Shared Device Model

In the shared device model, software running on any computer in the cluster can gain access to any hardware resource connected to any computer in the cluster (*for example, a hard drive*). If two applications require access to the same data, much like a symmetric multiprocessor (SMP) computer, access to the data must be synchronized. In most shared device cluster implementations, a component called a Distributed Lock Manager (DLM) is used to handle this synchronization.

The DLM is a service that is provided to applications running on the cluster that keeps track of references to hardware resources within the cluster. If multiple applications attempt to reference a single hardware resource, the DLM recognizes and resolves the conflict. However, using a DLM introduces a certain amount of overhead into the system in the form of additional message traffic between nodes of the cluster as well as the performance loss due to serialized access to hardware resources.

Shared Nothing Model

The shared nothing model is designed to avoid the overhead used by the DLM in the shared device model. In this model, each node of the cluster owns a subset of the hardware resources that make up the cluster. As a result, only one node can own and access a hardware resource at a time. When a failure occurs, another node takes ownership of the hardware resource so that the hardware resource can still be accessed.

In this model, requests from applications are automatically routed to the system that owns the resource. For example, if an application needs to access a database on a hard drive owned by a node other than the one it is running on, the node passes the request for the data to the other node. This allows the creation of applications that are distributed across nodes in a cluster.

3.3.3. Implementations

The rest of this section introduces three operating system implementations that constitute the majority of clusters currently being implemented:

1. Linux Redhat version 7.2 based on the 2.4.x series kernel.
2. Microsoft Windows 2000.
3. SUN Solaris.

Within each section the standard version of the OS is described, as well as the clustering version. It is of note that in all cases the clustering version costs more than the standard version and hence, all scientific computing clusters are built using the standard versions (*to maximise computing power for cost*) and corporations tend towards the cluster versions (*as they require the inbuilt clustering functions, reduced administration overheads, support and cost is not such a significant issue*).

The last heading in the section refers briefly to other operating systems that have been used in clustering.

3.3.4. Redhat Linux 7.2

Redhat Linux is a distribution of Linux that has received over a billion dollars worth of capital from companies such as IBM to produce an operating system to rival the Windows platform.

Whilst there are many distributions of the Linux Operating system, Redhat Linux is the most popular according to International Data Corp. (IDC) research [22]. The Linux Redhat distribution also introduced the Redhat Package Manager and according file packages denoted by the `.rpm` extension. This system allows precise control of installing and removing packages (*programs utilities, etc*).

As each different Linux distribution uses its own numbering schema is can be difficult to know which Linux version is actually implemented. Hence, the crucial number to check is the Linux Kernel version.

The latest version available is Redhat Linux 8.0 [Kernel 2.4.18]

For laboratory testing purposes the Linux Redhat 7.2 distribution will be used, implementing the 2.4.7-10 kernel. For further implementation details of the Linux Operating system refer to 4.1 for further details.

Features of the Linux Operating Systems include:

- Every line of the source code for the kernel and utilities is available at no cost.
- There are no licensing fees to run Linux. Refer to the GNU General Public License [39].
- Demand Paging.
- Virtual Memory.

- Multitasking.
- Multiple File Systems Support.
- Networking Support.
- Program Development Tools.
- Dynamically loadable, shared libraries.
- Graphical User Interface.
- Robust

Refer to Figure 3-5 for a basic block diagram of the structure of Linux.

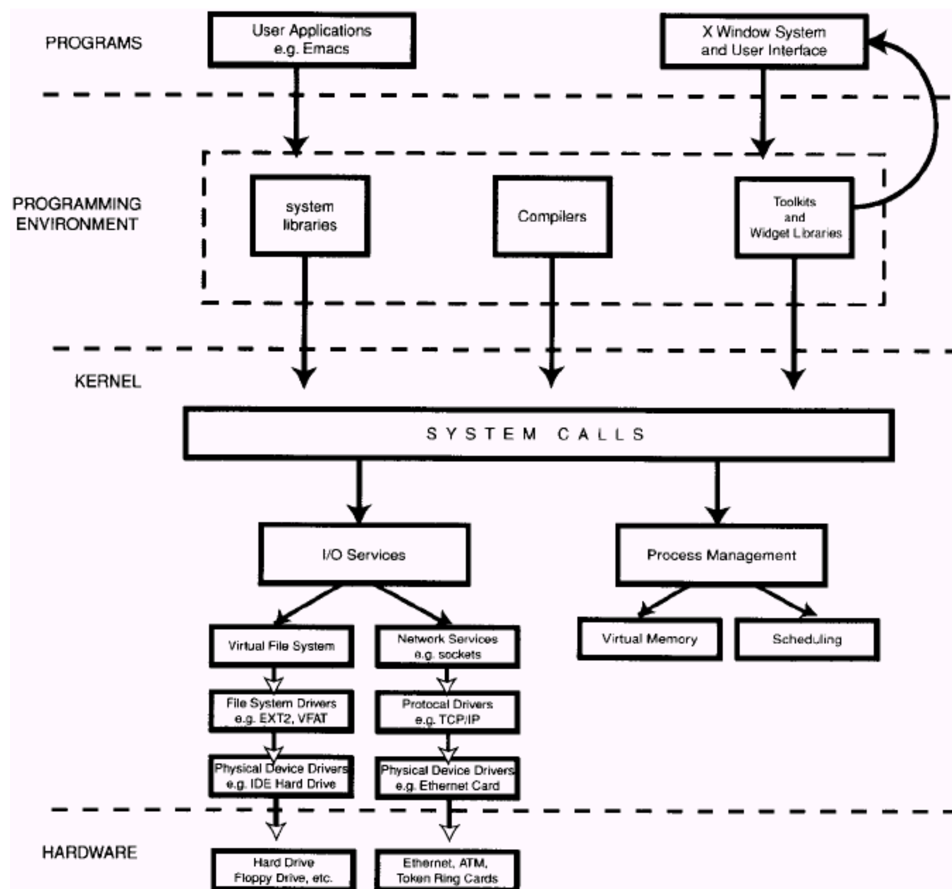


Figure 3-5 – System Component Relationships in the Linux Operating System

The Linux kernel interacts with file systems through a layer called the Virtual File System (VFS). This allows Linux to easily incorporate new file systems as the VFS provides the operating system a standard interface layer which mounts, reads, and writes file systems without requiring individual specific file formats.

Linux uses the Extended-2 File system, EXT-2, however with the release of Linux Redhat 7.2 and the 2.4.7-10 Kernel, the EXT3 file system is available.

File systems are fundamentally tree structures, but EXT2, like most Unix file systems, allows the file system to form directed graphs with hard linking and symbolic links as shown in Figure 3-6. [12] [31]

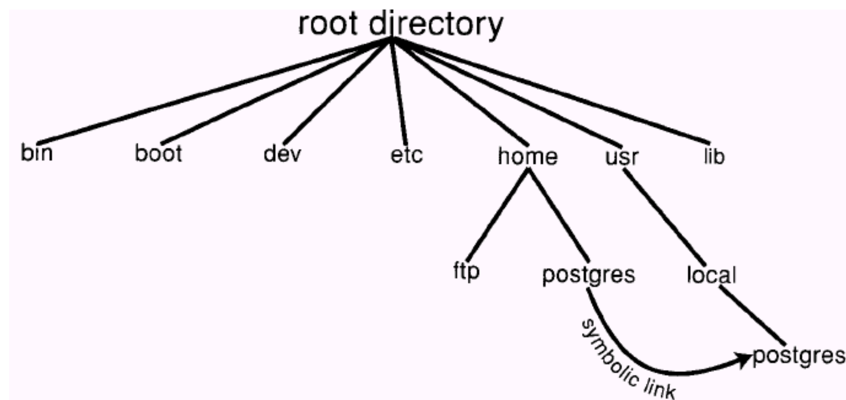


Figure 3-6 – Linux Hierarchical File System Structure

With respect to parallel systems, currently a parallel file system is available for Linux (*at the time of writing only available in beta format*) known as Parallel Virtual File System (PVFS).

The Parallel Virtual File System (PVFS) project is an effort to provide a high-performance and scalable parallel file system for PC clusters. PVFS is open source and released under the GNU General Public License [39]. It requires no special hardware or modifications to the kernel. PVFS provides four important capabilities in one package: [40]

- A consistent file name space across the machine.
- Transparent access for existing utilities.
- Physical distribution of data across multiple disks in multiple cluster nodes.
- High-performance user space access for applications.

Redhat High Availability Server

The investment of IBM in Redhat Linux, has allowed the vendor to develop a Clustering solution to rival that of Windows Cluster Server to deliver advanced security, Scalability, availability and reliability.

Known as Red Hat™ High Availability Server, it is a specialized version of the commonly used Red Hat Linux distribution, however, with a price of USD $1995. [22]

Red Hat High Availability Server is an out-of-the-box clustering solution that delivers dynamic load balancing, improved fault tolerance and scalability of TCP/IP based applications. It lets users combine individual servers into a cluster, resulting in highly available access to critical network resources such as data, applications, network services, and more. If one server in the cluster fails, another will automatically take over its workload. The Red Hat High Availability Server is ideally suited to Web servers, ftp servers, mail gateways, firewalls, VPN gateways and other front-end IP-based applications where virtually uninterrupted service is required.

The product supports heterogeneous network environments, allowing individual members of the cluster to run Red Hat Linux or virtually any other OS including Solaris®, and Windows NT®. Because the Red Hat High Availability Server is an open source product, customers are free from expensive technology lock-in that often occurs with proprietary solutions.

Security Features

The Red Hat High Availability Server has a number of inherent security features designed specifically for high availability Web front-end applications. Remote system access is disabled by default, and unused network services are not installed or started in the standard installation.

The Red Hat High Availability Server can be configured in two main ways. In Failover Services (FOS) mode, the system can be configured as a two node cold failover cluster ideally suited for applications where simple, affordable redundancy is needed such as firewalls, static Web servers, DNS, and mail servers. In Linux Virtual Server (LVS) mode, the system can be configured as an n-node cluster consisting of a two node load balancer, which accepts requests and directs those request to one of any number of IP-based servers based on a configurable traffic management algorithm.

Red Hat High Availability Server Features and Benefits

- **Ease of Installation** – the installation manager installs only those packages that are needed with the clustering packages.
- **High Performance and Scalability –** The Red Hat High Availability Server supports the scalability that meets the growth demands of highly dynamic IP environments. The number of cluster nodes is limited only by the hardware and network used. The product has advanced cluster features that provide high levels of performance including an ability to configure servers to bypass the load balancers when returning traffic back to the client, increasing the overall performance of the cluster. Additionally, because individual nodes can be taken off-line for maintenance or upgrades without interruption of service costly downtime can be eliminated.
- **Maximized Flexibility** – The Red Hat High Availability Server offers System Administrators a high degree of flexibility. In LVS mode the product supports four load balancing methods (*Round Robin, Weighted Round Robin, Least Connections, and Weighted Least Connections*) and three traffic forwarding techniques (*IP Masquerading, Tunnelling and Direct Routing*). Virtually every popular IP service is supported including Web (http), email, FTP, LDAP, DNS, as well as others.
- **Increased Security** – The Red Hat High Availability Server has built-in security features designed to withstand common attacks. System Administrators can set-up sand traps, providing for redirection of IP traffic from a potential attacker to a secure address. Out of the box, finger, talk, wall, and other daemons are disabled or not installed. In addition, multiple traffic routing and scheduling techniques along with virtual IP addresses allow creation of a security barrier for the network.
- **Availability** – The Red Hat High Availability Server dramatically reduces the likelihood of system failure by quickly detecting component server and application failures and redirecting workload to the remaining servers in the cluster. If one or more servers fail, others take over with minimal interruption.
- **Cost Effective** – Uses inexpensive commodity hardware to lower your overall cost of purchasing and maintaining the system.
- **Support** – A one-year support package that includes standard hours installation and configuration assistance and 24x7 server-down support.

3.3.5. Microsoft Windows 2000

Microsoft Windows is a dominant operating system in the PC marketplace. It is based on the Windows NT architecture and it is a stable, multitasking, priority-based pre-emptive scheduling, and multi-threaded 32-bit operating system.

Some of the features of W2K are listed below:

- Supports multiple CPU's and provides multi-tasking using SMP.
- Supports different CPU's using the HAL (*NT previously supported Intel x86, DEC Alpha, MIPS and it is understood this is set to continue*) and multiprocessor machines using POSIX threads.
- Uses an object-based security model and its own file system (NTFS) that allows permissions to be set on a file and directory basis.
- Built-in Networking protocols such as IPX/SPX, TCP/IP, and NetBEUI.

Figure 3-7 below illustrates the overall structure of W2K.

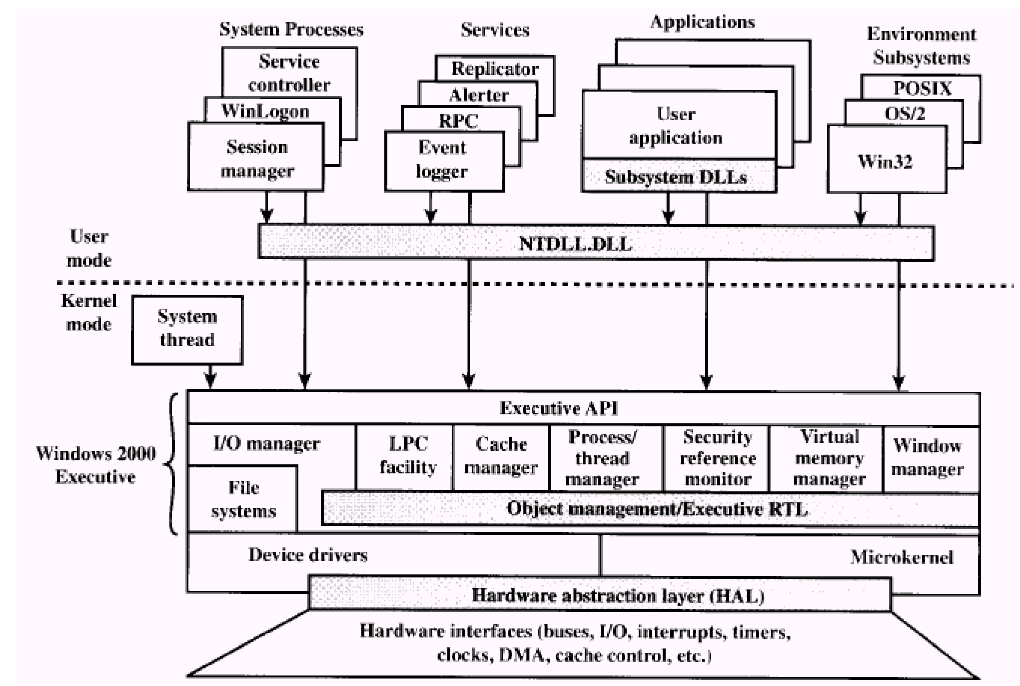


Figure 3-7 – Windows 2000 Architecture

As with the majority of operating systems, W2K separates application-oriented software from operating systems software. The latter, which includes the Executive, the micro-kernel device drivers, and the HAL, runs in kernel mode. Kernel mode software has access to system data and to the hardware. The remaining software, running in user mode, has limited access to system data.

W2K has what is referred to as a modified micro-kernel architecture, which is highly modular. Each system function is managed by just one component of the operating system and all applications access that function through the responsible component using a standard interface. In principle any module can be removed, upgraded, or replaced without rewriting the OS or any of its standard API's.

However, unlike a pure micro-kernel system, W2K is configured so that many of the system functions out-side the micro-kernel run in kernel mode. The reason is to boost the performance.

The most important sub-system is the Win32 API. The Win32 API allows the applications programmer access to the many functions of the Windows platform and is the point of interface for the cluster middleware layer of software. [31] [32]

One of the drawbacks of using the Windows 2000 platform (*standard and clustering versions*) for clustering is cost. A cluster is designed to function as a single machine, however in most cases we use a local operating system on each node of the system and with the W2K licensing model this would mean we would need to pay a licence cost per node. One possible way around this is to use diskless nodes. However, with operating systems such as Linux that are freely available under the GNU General Public License model [39] this is not required.

Windows 2000 Cluster Server

Windows 2000 (W2K) Cluster Server (*formerly codenamed Wolfpack*) is a shared-nothing cluster, in which each disk volume and other resources are owned by a single system at a time.

A typical set-up for a W2K cluster is shown in Figure 3-8 below.

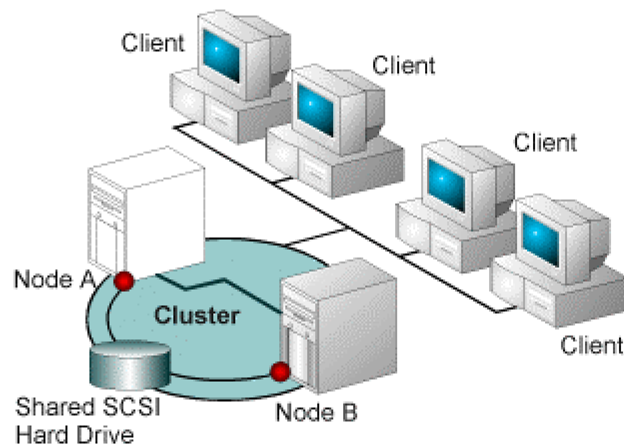


Figure 3-8 – Windows 2000 Cluster Server Topological Diagram

W2K is the next generation from Windows NT 4.0 and is the second generation of cluster operating systems from Microsoft.

W2K Cluster Server is based on the following concepts: [32]

- **Cluster Service** – The collection of software on each node that manages all cluster-specific activity.
- **Resource** – An item managed by the cluster service. All resources are objects representing actual resources in the system, including physical hardware devices such as disk drives, network cards and logical items such as logical disk volumes, TCP/IP addresses, entire applications, and databases.
- **Online** – A resource is said to be online at a node when it is providing service on that specific node.
- **Group** – A collection of resources managed as a single unit. Usually, a group contains all of the elements needed to run a specific application and for client systems to connect to the service provided by that application.

The Cluster Service (*Clussvc.exe*) and Resource Monitors (*Resrcmon.exe*) are the keys to a Cluster Server cluster. Each node in a cluster contains all of the components shown in Figure 3-9 and described in the architectural overview below. [34]

Architectural Overview

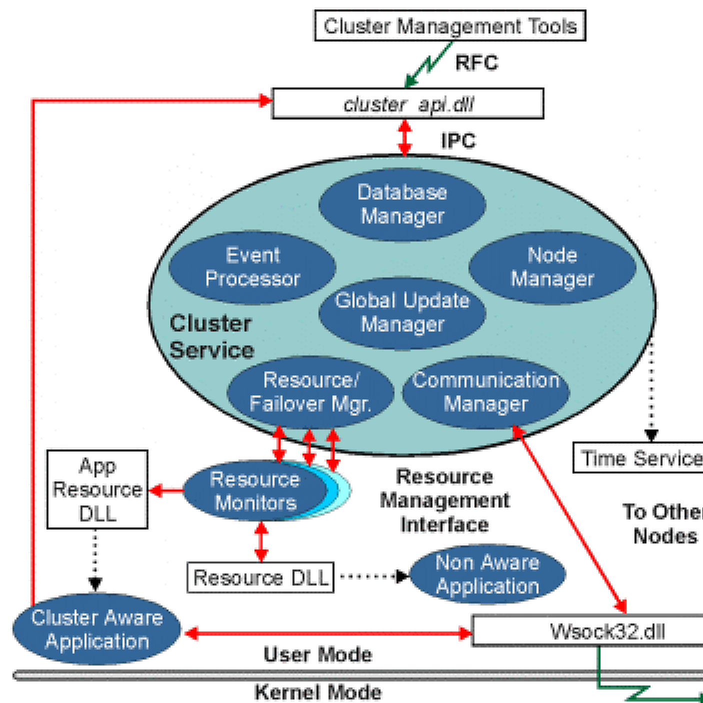


Figure 3-9 – Windows 2000 Cluster Server Block Diagram

The Database Manager is the component of the Cluster Service that implements the cluster database. This database contains information about all of the entities in the cluster, such as the cluster itself, resource types, groups, and resources. The cluster database is stored in the registry on each node of the cluster.

The Database Managers, one on each node in the cluster, cooperate with each other to maintain consistent cluster configuration information.

In addition, the Database Manager also provides an interface to the configuration database that is used by the other components of the Cluster Service. Cluster services coordinate updates to the registry to maintain consistency and provide atomic updates across the nodes in the cluster.

The Node Manager on one node in the cluster communicates with the Node Manager on the other node in order to detect node failures in the cluster. This is accomplished by sending "heartbeat" messages between the nodes.

If the Node Manager on a node does not respond, the active resources must be failed over to the node that is still running. The Resource/Failover Manager does the failover portion of this process. If the Cluster Service fails on a node, all of the resources on the node will failover even if the resources are still all online and functioning. This happens as a result of the Node Manager initiating failover of the resources since it is unable to communicate with the Node Manager on the other node.

Quorum Resource Interaction – If the communication link between the nodes becomes unavailable but both nodes are still functioning, the Node Manager on each node would try to fail the resources over to the node on which the Node Manager was running. This would result in each node thinking it is the surviving node of the cluster and would try to bring all the resources online. To prevent this situation, the Cluster Server relies on the quorum resource to ensure that only one node has a resource online. If communication between nodes fails, the node that has control of the quorum resource will bring all resources online. The node that

cannot access the quorum resource and is out of communication will take any resources it had offline.

The Event Processor is the communications centre of the Cluster Service. It is responsible for connecting events to applications and other components of the Cluster Service. This includes:

- Maintenance of cluster objects.
- Application requests to open, close, or enumerate cluster objects.
- Delivery of signal events to cluster-aware applications and other components of the Cluster Service.

The Event Processor is also responsible for starting the Cluster Service and bringing the node to the "offline" state. The Event Processor then calls the Node Manager to begin the process of joining or forming a cluster.

Communication Manager – The components of the Cluster Service communicate with the Cluster Service on other nodes through the Communication Manager. Communication Manager is responsible for the following:

- KeepAlive protocol. Checks for failure of the Cluster Service on other nodes.
- Resource group push. Initiates failover of resources from one node to another node.
- Resource ownership negotiation. Determines the new owner of resources from a failed node.
- Resource state transitions. Nodes notify the rest of the cluster when a resource goes offline or comes online. This is used to help track the owner of a resource.
- Enlist in a cluster. Initial contact with a cluster.
- Join a cluster. Synchronization with the other node in the cluster after going offline and coming back online.
- Database updates. Two-phase commit of cluster database updates.

The Global Update Manager provides an interface for other components of the Cluster Service to initiate and manage updates. The Global Update Manager allows for changes in the online/offline state of resources to be easily propagated throughout the nodes in a cluster. In addition, notifications of cluster state changes are also sent to all active nodes in the cluster.

Centralizing the global update code in a single component allows the other components of the Cluster Service to use a single, reliable mechanism.

The Resource/Failover Manager is responsible for:

- Managing resource dependencies.
- Starting and stopping resources, by directing the Resource Monitors to bring resources online and offline.
- Initiating failover and failback.

In order to perform the preceding tasks, the Resource/Failover Manager receives resource and cluster state information from the Resource Monitors and the Node Manager. If a resource becomes unavailable, the Resource/Failover Manager either attempts to restart the resource on the same node or initiates a failover of the resource.

3.3.6. Sun Solaris

The Solaris operating system from SunSoft is a UNIX-based multithreaded and multi-user operating system. It supports Intel x86 and SPARC-based platforms. Its network support includes a TCP/IP protocol stack and layered features such as Remote Procedure Calls (RPC), and the Network File System (NFS). The Solaris programming environment includes ANSI-compliant C and C++ compilers, as well as tools to profile and debug multithreaded programs.

The Solaris kernel supports multithreading, multiprocessing and has real-time scheduling features that are critical for multimedia applications. Solaris supports two kinds of threads: Light Weight Processes (LWPs) and user level threads (ULTs). [31]

In addition to many other features, Solaris has an in built Clustering feature set, which is of interest in both parallel processing and for reliability purposes.

Sun Cluster

Sun Cluster is a distributed operating system built as a set of extensions to the base Solaris UNIX system. It provides the cluster with a single-system image; that is the cluster appears to the user and applications as a single computer running the Solaris operating system.

Figure 3-10 shows the overall architecture of Sun Cluster. The major components are as follows: [32]

- **Object and communication support** – The SUN Cluster implementation is object oriented. The CORBA object model is used to define objects and the remote procedure call mechanism.
- **Process management** – Global process management extends process operations so that the location of a process is transparent to the user. Process migration is possible, i.e. a process can move from one node to another during its lifetime to achieve load balancing or for failover.
- **Networking** – SUN Cluster implements networking systems to support the cluster implementation.
- **Global distributed file system –** SUN Cluster implements a global file system to simplify file system operations and process management over the entire cluster.

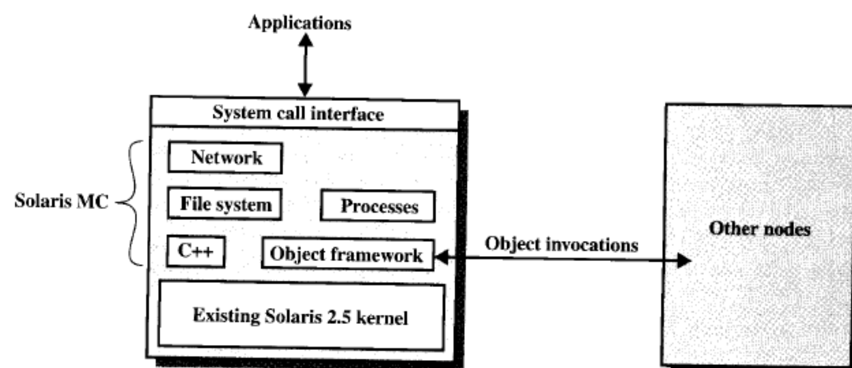


Figure 3-10 – Sun Cluster Structure

3.3.7. Other

Whilst Linux, Windows NT and SUN Solaris have been discussed with respect to clustering, this is not to say that other Operating systems are not available that can be used in a cluster configuration.

For reference, some of the other operating systems that have been used in this configuration are listed below:

- **Free BSD** – Based on the Berkley Standard Distribution of Unix produced by the University of California at Berkley. This operating system is freely available.
- **MAC OS** – The Macintosh Operating system for use with Macintosh Hardware.
- **Scyld Beowulf Cluster Operating System** – The original Beowulf creators and researchers [2.3] formed Scyld Corporation and are producing pre-configured cluster suited Linux distributions. As can be expected these Linux distributions cost as much as $8,000 US dollars for a 32-processor cluster as opposed to plain Linux available free under the GPL [27].

3.4. Middleware

Middleware is the key differentiator between a standard network PC's and a parallel processing supercomputing facility. Middleware is a software layer that is added on top of the operating system to provide what is known as a **Single System Image** (SSI).

Middleware provides a layer of software that enables uniform access to different nodes on a cluster regardless of the operating system running on a particular node. This is a similar concept to the Hardware Abstraction Layer (HAL) in Windows 2000 that provides uniform and portable access to a PC's hardware.

The middleware is responsible for providing high availability, by means of load balancing and responding to failures in individual components.

The following lists the desirable objectives of cluster middleware services and functions [32]:

1. **Single Entry Point –** A user logs onto the cluster rather than on to an individual computer.
2. **Single File Hierarchy –** The user sees a single hierarchy of file directories under the same root directory.
3. **Single Control Point –** There is a default workstation used for cluster management and control. This is usually known as the server.
4. **Single Virtual Networking –** Any node can access any other point in the cluster, even though the actual cluster configuration may consist of multiple interconnected networks. There is a single virtual network operation.
5. **Single Memory Space –** Distributed Shared Memory enables programs to share variables.
6. **Single job-management system –** Under a cluster job scheduler, a user can submit a job without specifying the host computer to execute the job.
7. **Single User Interface –** A common graphic interface supports all users, regardless of the workstation from which they enter the cluster.
8. **Single I/O Space –** Any node can remotely access any I/O peripheral or disk device without knowledge of its physical location.
9. **Single Process Space –** A uniform process-identification scheme is used. A process on any node can create or communicate with any other process on a remote node.
10. **Checkpointing –** This function periodically saves the process state and intermediate computing results, to allow rollback recovery after a failure.
11. **Process Migration –** This function enables load balancing.

Items 1 through 7 are concerned with providing a Single System Image, and items 8 through 11 are concerned with enhancing the availability of the cluster.

This section details the recent advances in Cluster Middleware.

3.4.1. Parallel Communications Libraries

A cluster is a collection of local memory machines. The only way for **Node A** to communication to **Node B** is through the network. Software built on top of this architecture 'passes messages' between nodes. While message-passing codes are conceptually simple, their operation and debugging can be quite complex.

A major goal of middleware for message passing systems is to ensure the degree of portability across different machines. The expectation is that the same message passing code should be able to be executed on a variety of machines as long as the message-passing library is available. Whilst this is the case, some tuning may be required to take best advantage of the features of each system.

There are two popular message passing libraries (*or lowest layer of middleware*) that are commonly used are PVM and MPI. Both PVM and MPI provide a portable software API that supports message passing.

From a historical stand point PVM appeared first and was designed to work on networks of workstations (*to create a Parallel Virtual Machine*). It has since been adapted to many parallel supercomputers (*using both distributed and shared memory*). Control of PVM is primarily with its authors.

In contrast, MPI is a standard that is supported by many hardware vendors (*such as IBM, HP, HITACHI, SUN, & Cray*). It provides more functionality than PVM and has versions for networks of workstations (*such as Beowulf clusters).* Control of MPI is with the standards committee known as the **MPI forum**. The most current release of the MPI standard is MPI2. [3] [11]

As MPI2 is the current standard for message passing systems, the experimental testing phase of this documentation focuses on this library although both PVM and MPI are discussed.

3.4.1.1. PVM Overview

The PVM (*Parallel Virtual Machine*) software provides a unified framework within which parallel programs can be developed in an efficient and straightforward manner using existing hardware. PVM enables a collection of heterogeneous computer systems to be viewed as a single parallel virtual machine. PVM transparently handles all message routing, data conversion, and task scheduling across a network of incompatible computer architectures.

The PVM computing model is simple yet very general, and accommodates a wide variety of application program structures. The programming interface is deliberately straight forward, thus permitting simple program structures to be implemented in an intuitive manner. The user writes his application as a collection of cooperating tasks. Tasks access PVM resources through a library of standard interface routines. These routines allow the initiation and termination of tasks across the network as well as communication and synchronization between tasks. The PVM message-passing primitives are oriented towards heterogeneous operation, involving strongly typed constructs for buffering and transmission. Communication constructs include those for sending and receiving data structures as well as high-level primitives such as broadcast, barrier synchronization, and global sum.

PVM tasks may possess arbitrary control and dependency structures. In other words, at any point in the execution of a concurrent application, any task in existence may start or stop other tasks or add or delete computers from the virtual machine. Any process may communicate and/or synchronize with any other. Any specific control and dependency structure may be implemented under the PVM system by appropriate use of PVM constructs and host language flow-control statements. Owing to its ubiquitous nature (*specifically, the virtual machine concept*) and because of its simple but complete programming interface, the PVM system has gained widespread acceptance in the high-performance scientific computing community. [2]

PVM contains **fault tolerant** features that become more important as the cluster size grows. The ability to write long running PVM applications that can continue even when hosts or tasks

fail, or loads change dynamically due to outside influence, is quite important to heterogeneous distributed computing. [21]

PVM documentation and distributions for both Windows NT and Unix/Linux are available from the PVM home page at: http://www.epm.ornl.gov/pvm/.

The most current version of PVM is 3.4.3.

It is also noted that due to PVM's long running history there are many other websites that contain PVM source code and programming examples.

3.4.1.2. MPI Overview

MPI (*Message Passing Interface*) is a standardized and portable message-passing system designed by a group of researchers from academia and industry to function on a wide variety of parallel computers. The standard defines the syntax and semantics of a core of library routines useful to a wide range of users writing portable message-passing programs in Fortran 77, C and now C++ and Fortran 90. Several well-tested and efficient implementations of MPI already exist, including some that are free and in the public domain. These are beginning to foster the development of a parallel software industry, and there is excitement among computing researchers and vendors that the development of portable and scalable, large-scale parallel applications is now feasible.

10 Reasons for use of MPI over PVM

Many comparisons of the two message passing models have been undertaken in the Research Community [19.1], [21] and the salient reasons for using MPI are outlined below:

1. **MPI has more than one freely available, quality implementation** – There are at least LAM, MPICH and CHIMP. The choice of development tools is not coupled to the programming interface.
2. **MPI defines a 3rd party profiling mechanism** – A tool builder can extract profile information from MPI applications by supplying the MPI standard profile interface in a separate library, without ever having access to the source code of the main implementation.
3. **MPI has full asynchronous communication** – Immediate send and receive operations can fully overlap computation.
4. **MPI groups are solid, efficient, and deterministic** – Group membership is static. There are no race conditions caused by processes independently entering and leaving a group. New group formation is collective and group membership information is distributed, not centralized.
5. **MPI efficiently manages message buffers** – Messages are sent and received from user data structures, not from staging buffers within the communication library. In some cases Buffering may be totally avoided.
6. **MPI synchronization protects the user from third party software** – All communication within a particular group of processes is marked with an extra synchronization variable, allocated by the system. Independent software products within the same process do not have to worry about allocating message tags.
7. **MPI can efficiently program MPP and clusters** – A virtual topology reflecting the communication pattern of the application can be associated with a group of processes. An MPP implementation of MPI could use that information to match processes to processors in a way that optimises communication paths.
8. **MPI is highly portable** – Recompile and run on any implementation. With virtual topologies and efficient buffer management, for example, an application moving from a cluster to an MPP could even expect good performance.

9. **MPI is formally specified** – Implementations have to live up to a published document of precise semantics.
10. **MPI is a standard** – Its features and behaviour were arrived at by consensus in an open forum. It can change only by the same process.

The most current MPI standard, **MPI2** can be obtained from the MPI forum website [20] at:

http://www.mpi-forum.org/

The most notable additions to the MPI2 standard (*from the original MPI standard*) are listed below:

- **Parallel I/O**
- **Remote Memory Operations**
- **Dynamic Process Management**
- **Supports the use of threads** – It is noted that POSIX Threads (*as defined by the POSIX standard)* should be used to ensure portability.

While the MPI standards are freely available from the MPI forum, distributions for different operating systems and platforms are available from various groups. A reliable source that lists all current commercial and freely available distributions is the LAM-MPI home page at:

http://www.lam-mpi.org/mpi/implementations/

Two of freely available distributions of MPI are known as MPICH and LAM. Many performance comparisons of the two have been undertaken in the Research Community [19.4]. For testing purposes the **LAM distribution** will be used as in many cases LAM exhibits better performance [19.4] and has greater compliance with the MPI 2 standard [19.2]. Refer to Appendix C for Standard compliance details and the LAM-MPI website [19.4] for performance details.

LAM – Local Area Multicomputer

LAM (*Local Area Multicomputer*) is an MPI programming environment and development system for heterogeneous computers on a network. With LAM, a dedicated cluster or an existing network computing infrastructure can act as one parallel computer solving one problem.

The latest stable version of LAM/MPI is 6.5.9 [Released 28 Jan 2003]

LAM features extensive debugging support in the application development cycle and peak performance for production applications. LAM features a full implementation of the MPI communication standard.

There are currently two versions of LAM available: LAM 6.5.9 and LAM 6.6 beta:

- LAM version 6.5.9 includes many new features (*such as support for Interoperable MPI [IMPI]*) and bug fixes. It is the most recent stable release. More information on IMPI can be found in [3].
- LAM version 6.6 beta is the next generation LAM software, except it is only available in beta release and hence is not able to be used for a production cluster.

LAM on Linux

As reported by the MPI-LAM Development Team [19.3] the 2.2.x series of the Linux kernel has bugs in part of its TCP/IP implementation. This has caused performance problems in the LAM/MPI software. Linux 2.2.10 appears to have rectified this problem, since LAM performance in 2.2.10 is comparable to LAM performance under 2.0.36. The LAM development team have not found a way to explain the performance differences noted in the TCP test program, even in 2.2.10.

All Linux versions that shall be used will be Redhat Linux Version 6.2, kernel version 2.2.14-5.0 and later. [22]

3.4.2. Application Development Packages

On of the major roadblocks to parallel processing are the billions of dollars worth of existing software that are only able to be run on serial SISD machines.

The cost of the change to switch to parallel programs and retraining of programmers is prohibitive. Whilst this objection is true and it is noted that not all programs needed in the future have been written. New applications will be developed and many new problems will become solvable with increased performance through the use of parallel systems. Students of operating systems are already being trained to think in parallel and use methodologies that can parallelise serial applications. Additionally, tools are both currently available and being developed to transform sequential code into parallel code automatically (*Fortran and C tools are available* [11]).

In a pre-emptive initiative, it has been suggested by Behrooz [9] that it may be prudent to develop programs in parallel languages even if they are to be run on sequential computers. In doing this the added information with respect to concurrency and data dependencies would allow a sequential computer to improve its performance by instruction prefetching, data caching, and so forth. [9]

To this end research into parallel systems has focused on developing a cost effective hardware platform such as Beowulf, and cost effective message passing platforms. These message-passing platforms have been available freely for commercial operating systems, however do not parallelise a serial problem, and great effort is required by the programmer to do this.

Parallel programs can be expected to have a lifespan of decades, while the platforms on which they execute have life spans of much less than a decade. Accordingly, such software must be designed to run on more than one computer over its lifetime.

Whilst there are many parallel languages available for parallel programming, not all are discussed here. Additionally it is preferable for the programming systems software to utilize the lower layer of **middleware** **(***i.e. a standard message passing library***)** as described above. The reason for this is two fold:

1. **Software needs to be portable –** An overall goal of parallel and distributed systems is to ensure that when a piece of software or an application is written that it has portability, i.e. the ability to execute on many different machines. (*As discussed in Section 3.4*).
2. **A programming system must have the ability to abstract** and hide most of the work involved in parallel development and execution. Without this feature, the switch to parallelisation of programming will be arduous.

Skillicorn [37] argues that parallel programming models and languages should:

1. **Be easy to program –** the exact structure of the executing program should be inserted by the translation mechanism (*compiler and run-time system*) rather than by the programmer. This implies that a model should conceal:
    - Decomposition of a program into parallel threads.
    - Mapping of threads to processors.
    - Communication among threads.
    - Synchronization among threads.
2. **Have a software development methodology –** To bridge the gap between the information provided by the programmer about the semantic structure of the program, and the detailed structure required to execute it, requires a firm semantic foundation on which transformation techniques can be built.

3. **Be architecture-independent -** so that programs can be migrated from parallel computer to parallel computer without having to be redeveloped, or indeed modified in any non-trivial way. This requirement is essential to permit a widespread software industry for parallel computers as computer architectures have comparatively short life spans.
4. **Be easy to understand** – A model should be easy to understand and to teach, since otherwise it is impossible to educate existing software developers to use it.
5. **Be efficiently implementable** – A model should be efficiently implementable over a useful variety of parallel architectures. Note that efficiently implementable should not be taken to mean that implementations extract every ounce of performance out of the target architecture.
6. **Provide accurate information about the cost of programs –** Any program's design is driven, more or less explicitly, by performance concerns. Execution time is the most important of these, but others such as processor utilization or even cost of development are also important.

These criteria reflect the belief that developments in parallel systems must be driven by a parallel software industry based on portability and efficiency.

Skillcorn evaluated programming models in six categories, depending on the level of abstraction they provide. Those that are very abstract conceal even the presence of parallelism at the software level. Such models make software easy to build and port, but efficiency is usually hard to achieve. At the other end of the spectrum, low-level models make all of the messy issues of parallel programming explicit (*how many threads, how to place them, how to express communication, and how to schedule communication*), so that software is hard to build and not very portable, but is usually efficient.

Most recent models are near the centre of this spectrum, exploring the best trade-offs between expressiveness and efficiency. However, there are models that are both abstract and are able to be implemented efficiently, opening the prospect of parallelism as part of the mainstream of computing, rather than high-performance computing only.

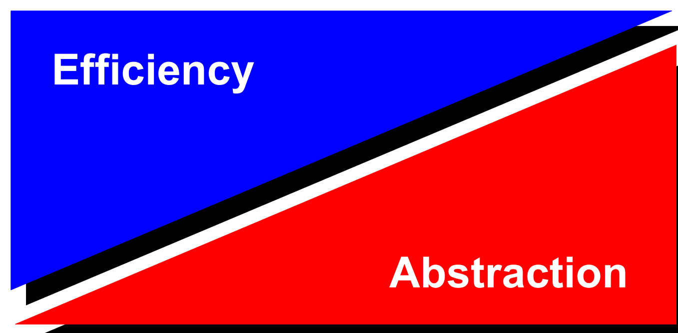


Figure 3-11 – The Parallel Programming Model Efficiency Abstraction Trade-off

Skillcorn's six categories are as follows:

1. **Nothing Explicit, Parallelism Implicit –** Models that abstract from parallelism completely. Such models describe only the purpose of a program and not how it is to achieve this purpose. Software developers do not need to know even if the program they build will execute in parallel. Such models are abstract and relatively simple, since programs need be no more complex than sequential ones.
2. **Parallelism Explicit, Decomposition Implicit –** Models in which parallelism is made explicit, but decomposition of programs into threads is implicit (*and hence so is mapping, communication, and synchronization*). In such models, software developers are aware that parallelism will be used, and must have expressed the potential for it in programs, but do not know even how much parallelism will actually be applied at

run-time. Such models often require programs to express the maximal parallelism present in the algorithm, and then reduce that degree of parallelism to the target architecture, at the same time working out the implications for mapping, communication, and synchronization.

3. **Decomposition Explicit, Mapping Implicit –** Models in which parallelism and decomposition must both be made explicit, but mapping, communication, and synchronization are implicit. Such models require decisions about the breaking up of available work into pieces to be made, but they relieve the software developer of the implications of such decisions. (*i.e. BSP*)
4. **Mapping Explicit, Communication Implicit –** Models in which parallelism, decomposition, and mapping are explicit, but communication and synchronization are implicit. Here the software developer must not only break the work up into pieces, but must also consider how best to place the pieces on the target processor. Since locality will often have a marked effect on communication performance, this almost inevitably requires an awareness of the target processor's interconnection network. It becomes very hard to make such software portable across different architectures.
5. **Communication Explicit, Synchronization Implicit –** Models in which parallelism, decomposition, mapping, and communication are explicit, but synchronization is implicit. Here the software developer is making almost all of the implementation decisions, except that fine-scale timing decisions are avoided by having the system deal with synchronization.
6. **Everything Explicit –** Models in which everything is explicit. Here software developers must specify all of the detail of the implementation. As noted earlier, a programming model needs the ability to abstract, as it is extremely difficult to build software using such models, because both correctness and performance can only be achieved by attention to vast numbers of details. (*i.e. PVM, MPI*)

Two high-level applications development packages, BSP and ARCH are briefly discussed below. Further information on the design and implementation of parallel programs can be obtained from the online book Designing and Building Parallel Programs. [42]

3.4.2.1. BSP

Bulk Synchronous Parallelism (BSP) is a parallel programming model that abstracts from low-level program structures in favour of supersteps. A superstep consists of a set of independent local computations, followed by a global communication phase and a barrier synchronization. Structuring programs in this way enables their costs to be accurately determined from a few simple architectural parameters, namely the permeability of the communication network to uniformly-random traffic and the time to synchronize. Although permutation routing and barrier synchronizations are widely regarded as inherently expensive, this is not the case. As a result, the structure imposed by BSP comes free in performance terms, while bringing considerable benefits from an application-building perspective. [25]

3.4.2.2. ARCH

ARCH is an extension to MPI relying on a small set of programming abstractions that allow the writing of multi-threaded parallel codes according to the object-oriented programming style. ARCH has been written on top of MPI with C++. Very little knowledge of MPI is required to use ARCH as the latter entirely redefines the user interface in an object-oriented style and with new thread-compatible semantics. C++ was not simply used as a development language. Instead, it was attempted to transmit the object-oriented method for program development.

ARCH consists of several sets of C++ classes supplying tools for the writing of multi-threaded parallel codes.

The first set deals with threading and supplies two classes to this purpose: the Thread and S_Thread classes. The Thread class provides functions for thread construction, destruction, initialisation, scheduling, suspension and so forth. S_thread is defined by private derivation from the previous one and allows the writing of multi-threaded programs in a structured style. Illustration cases may be found in [35] where each multi-threaded program is presented in a layout similar to an electronic board design. The library contains three additional sets of classes for thread synchronization and communication. Each set relates to a well-identified communication model:

1. Point-to-point synchronous.
2. Point-to-point asynchronous.
3. One-sided via global write/read function calls.

4. System Installation & Testing

4.1. Building a Beowulf

Whilst this section is not intended to be a complete or comprehensive guide to building a Beowulf Cluster, it does list the salient points on the configuration process as well as the explicit configuration used for testing in the laboratory.

Building a Beowulf cluster requires a thorough working knowledge of the operating system including networking, file systems, system configuration, daemons and services. In relation to Linux this requires a working knowledge of up to 30 different configuration files and their individual format requirements to get a system up and running. This detailed configuration knowledge is learned through reading and implementing (*by trial and error*) the various How-Tos.

With respect to my knowledge and experience with Linux, it dates back to Redhat Linux 5.2 (October 1998) and the documentation that is particularly useful for building a cluster is noted in [11] and [38.1]

Linux Installation

Initially the Redhat 6.2 and 7.1 (*codename: Seawolf, Linux Kernel Version 2.4.7-2*) distributions were initially tested. It was found that both contained many bugs, especially to do with NFS and EXT2.

As soon as the Redhat 7.2 distribution was available, all nodes in the cluster were upgraded to RH7.2 as it implements the 2.4.x Kernel and the EXT3 file system that offers greater stability and performance. The RH7.2 distribution also includes versions of LAM and PVM that can be installed as part of the initial installations.

For laboratory-testing purposes the Linux Redhat 7.2 distribution will be used implementing the 2.4.7-10 kernel.

It is of note that the 2.4.x series kernels do not yet implement NFS over TCP instead UDP is used (*which Linux uses by default as opposed to Sun Solaris using TCP*).

In total eight nodes were used, the specifications of which is shown in Figure 4-1

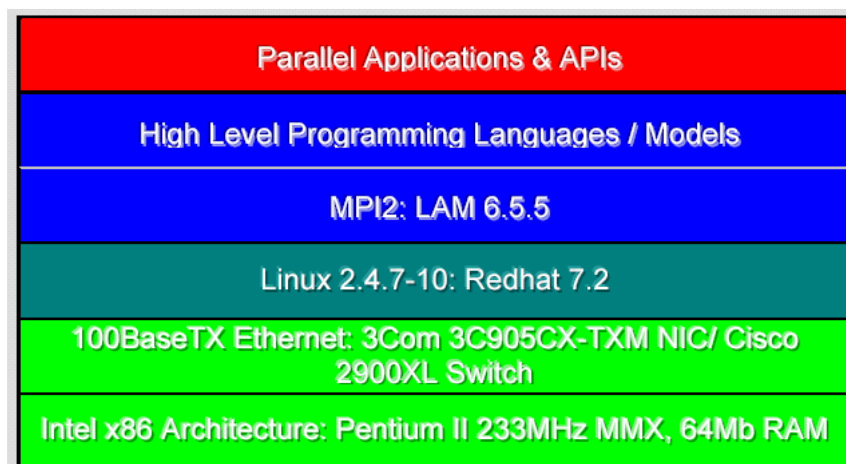


Figure 4-1 – Layered Model of Beowulf Cluster Computer Used in Testing

In addition to the configuration as detailed in Figure 4-1, the master server was installed with 128Mb of Ram, an additional network card for external cluster access, and a 2Gb hard-drive for the `/home` directory which will be cross mounted within the cluster.

Installation

The following process was under taken in installing the system:

1. Install CD-ROM on the master node (*server node*).
2. Install Linux Redhat 7.2 on the master node from the CD-ROM using a CD-ROM boot image floppy (*use the dos utility* `rawrite` *from CD-ROM 1*).
3. Configure using Linuxconf and start the required services using serviceconf. (*Note: if serviceconf is not available then symbolic linking is required to the Run Level 3 and 5 startups – however I prefer to use a GUI for system services operations, hence serviceconf was used*).
4. Create the following accounts to be used throughout the whole cluster:

Name	Password	Purpose
`root`	`cluster`	`# for administration`
`beowulf`	`beowulf`	`# cluster operation`

   The root account is local to each machine, however the `beowulf` account is global (*using the same group and user ID*) and hence a change on any node to this account will be instantaneously reflected across all nodes. Additionally the beowulf account and home directory are located under `/home/beowulf` which is exported from the NFS server, hence only one set of config files exist for this account. The beowulf account was configured using the `csh` (*C Shell*) for compatibility and ease of configuration (*requires conf files for POV-Ray, LAM and other parallel applications*).

5. Start NFS server, exporting `/home` and `/mnt/cdrom` to all nodes in the cluster.
6. Ensure NFS, RSH, NTP, TCP/IP services are working. Use `rpcinfo -p` to make sure the NFS server is working (*rpc.portmap, rpc.mountd, rpc.nfsd, rpc.statd, rpc.lockd, rpc.rquotad should be listed*).
7. Copy Linux Redhat 7.2 CDs (*1 and 2*) from the CD-Rom to `/home`.
8. Install Linux Redhat 7.2 on each node over the network using an NFS image and a network boot image floppy – do not add accounts other than the root account at this stage.
9. Configure using Linuxconf and start the required services using serviceconf (*all required files can be configured with these utilities or manually. Using these utilities is through recommended, however knowledge of the file formats is still required*).
10. Add all nodes to the `etc/hosts` file, such as:

```
node1   192.168.0.1 # master server, NFS server and /home dir
node2   192.168.0.2
node3   192.168.0.3
node4   192.168.0.4
node5   192.168.0.5
node6   192.168.0.6
node7   192.168.0.7
node8   192.168.0.8 # backup LAM server
```

11. DNS and NIS were not used to reduce daemon overhead. However could be used in future clusters to ease of maintenance and management.

12. Get Telnet up and running, as well as Linuxconf web access services for remote management of all nodes. Linuxconf web access allows remote configuration through a web-browser via port 98. For example:

    http://192.168.0.x:98

13. Edit all Beowulf clients `/etc/ntp.conf` to add the server address `192.168.0.1` and set the Beowulf master clock to the correct time.
14. Cross mount `/home` and `mnt/cdrom` from the NFS server on each node of the system.
15. Remove internal passwords for each node in the cluster using a `.rhosts` file in the top level beowulf account directory.
16. Ensure `rsh` and `rlogin` access from the server to each node is possible.
17. Stop all unused daemons running on every node in the cluster. This includes `lpd` and `sendmail` for printer and email services respectively.
18. Stop all applications that consume resources such as processing power and memory. Boot each node into Run-level 3 (Multi-User/Text/Full Network).
19. Install LAM, if not installed in the initial installation. Configure the `/etc/lam/lam-bhost.lam` file to contain the node names on the system.
20. Login to each node using a user account, rather than the `root` account. LAM does not allow the use of the `root` account as it could crash and destroy the system (*for testing purposes, it is actually more convenient to log into each node as root and only the server node as beowulf, the server logs into each node using the beowulf operating account, even though passwords are not required*).
21. Start LAM using the `lamboot` command on the server [Refer to 8.5.1 for details].

Notes on the Installation:

The following notes are for historical reference purposes:

1. Linux's default implementation of NFS is over unreliable data transport UDP rather than reliable data transport such as TCP. This is in contrast to Sun Solaris. Linux can be configured to use TCP however the Linux 2.4.x series kernels do not provide support for this at the time of writing. Should reliability be a design requirement then the 2.2.x series kernels should be used such as Redhat 6.2. Another benefit of using earlier distributions is that they have smaller size-footprint. [41]
2. Whilst the test installation was carried out using a network installation, nodes can be cloned which reduces the overall work involving in setting up or in adding additional nodes in a large cluster. Information on cloning nodes can be obtained at: ftp://ftp.sci.usq.edu.au/pub/jacek/beowulf-utils/disk-less/ [11]
3. Whilst PVM was not recommended in this document to be used as the lower layer of middleware, it does provide some noteworthy features such as fault-tolerance and an easy to use GUI that can be installed with Redhat 7.2.

All nodes in the cluster were located at the University of Technology in the ITS research laboratory in Building 1, Level 21 room 1/2122E. The Ethernet switch was located in room 1/2122A. As can be seen in Figure 4-2 below, all machines have monitors, keyboards and mouses, an attribute that is not required in a Beowulf, cluster. Although not required, this does make configuration, in various test environments easier.

Figure 4-2 – Test Cluster Computer in Lab B1/2122E

4.2. Performance Testing

The test suites as outlined in this section were installed and run on the test-cluster.

4.2.1. Beowulf Performance Suite

The Beowulf Performance Suite (BPS) was developed by Paralogic [38.2] as a graphical front end to a series of commonly used performance tests for Parallel Computers. It was designed explicitly for the Beowulf Class of computers running Linux and the suite is packaged as an easy to install `.rpm` file. The suite contains seven different but well-known tests, which were commonly installed separately.

The suite can be run from the command line by invoking `bps` or `xbps` with the GUI as shown in Figure 4-3 below. The suite generates all test results and graphs (*using Ggnuplot*) in .html format hence documentation can be maintained on line, which is in-line with the use of web-based configuration and reporting mechanisms to streamline system administration (*such as CISCO Ethernet switch management and node management with linuxconf web access*).

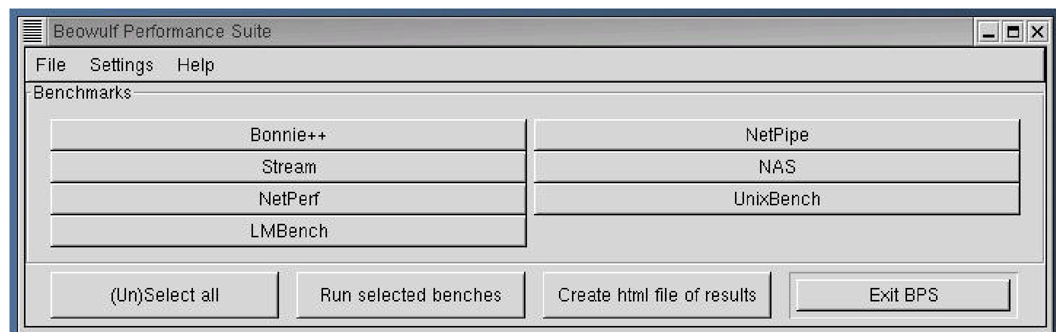


Figure 4-3 – BPS Running on node1 of the Test Cluster

Details of each of the seven tests are noted below:

1) **Bonnie++** – is a benchmark suite that is aimed at performing a number of simple tests of hard drive and file system performance.
2) **Stream** – The STREAM Benchmark is the de-facto industry standard benchmark for the measurement of computer memory bandwidth. The STREAM benchmark measures "real world" bandwidth sustainable from ordinary user programs - not the theoretical "peak bandwidth".
3) **NetPerf** – Netperf is a benchmark that can be used to measure the performance of many different types of networking. It provides tests for both unidirectional throughput, and end-to-end latency. The environments currently measurable by netperf include:
   - TCP and UDP via BSD Sockets
   - DLPI
   - Unix Domain Sockets
   - Fore ATM API
   - HP HiPPI Link Level Access
4) **LMBench** – is a benchmark suite that tests system bandwidth and latency:
   - Bandwidth benchmarks (*Cached file read, Memory copy, Memory read/write, Pipe, TCP*)
   - Latency benchmarks (*Context switching, Networking: connection establishment, pipe, TCP, UDP, and RPC hot potato, File system creates & deletes, Process creation, Signal handling, System call overhead, Memory read latency, Miscellaneous, Processor clock rate calculation*)
5) **NetPipe** – NetPIPE (*A Network Protocol Independent Performance Evaluator*) is a protocol independent performance tool that encapsulates the best of ttcp and netperf and visually represents the network performance under a variety of conditions. By taking the end-to-end application view of a network, NetPIPE clearly shows the overhead associated with different protocol layers. Netpipe assists with identifying ways of optimising a communication channel to boost the overall system performance.

6) **NAS** – The NAS Parallel Benchmarks (NPB) are a set of eight programs designed to assist in the performance evaluation of parallel supercomputers. The benchmarks, which are derived from computational fluid dynamics (CFD) applications, consist of five kernels and three pseudo-applications.
7) **UnixBench** – tests the file I/O and kernel multitasking performance resulting in a system index.

Package Dependencies: Gnuplot (*graph plotting tools*)

4.2.2. The Linpack Benchmark

Each of the top 500 supercomputers is measured by the LINPACK benchmark taking the `best' case performance [29.1]. LINPACK was chosen as it is widely used and performance numbers are available for almost all relevant systems and is hence a relevant benchmark for clusters.

The LINPACK Benchmark was introduced by Jack Dongarra. The benchmark used in the LINPACK Benchmark is to solve a dense system of linear equations.

Package Dependencies: blas (*basic linear algebra sub-programs*)

4.3. System Administration

The following system administration tools were used to manage and monitor the test-cluster in real-time.

4.3.1. General

Packaged with the RH7.2 distribution are two useful operating system level tools that were widely used to set-up and manage the workstations:

1. **Linuxconf** – Linuxconf is a utility similar to Windows NT control panel. The majority of settings for the local workstation or server, network configuration, file systems and user accounts are available to be set in a GUI environment. This is in stark contrast to the norm of configuring 30 different text files, each with there own different formats. Another benefit for clustering and networking in general is that nodes can be managed by Linuxconf over a network through a web browser.
2. **Serviceconf** – Serviceconf is a utility again similar to Windows NT service manager. Serviceconf allows the various services to be started, stoped and set-up for each run-level in a GUI environment. Within a cluster environment, this is important as there are many services that need to be run from the `xinetd` service. Whilst Linuxconf does have the ability to manage services, it does not extend to the sub-protocols of `xinetd.`

A third package BCS (*basic cluster scripts*) is a package of specially developed parallel tools [38] and was added for cluster administration purposes. The scripts are implemented in the Shell BASH, which was used for all root accounts on the cluster.

The scripts were designed so that any group of Linux machines with `rcp` and `rsh` can be acted upon as a group. This assists configuration, i.e. copying a configuration file to each machine or installing an `.rpm` across each machine.

4.3.2. Mosixview

Mosxiview is an extensible management and monitoring tool for Beowulf Clusters, developed to simplify the management tasks for a large Beowulf Cluster. [1.1]

Real-time monitoring is critical in a clustering environment, as we need a way to ensure that each node is up and running, has enough resources to run, and is not overloaded.

As shown in Figure 4-4 below, the main window of Mosixview lists all the vital statics of each machine. Moxixview additionally has a service control window for managing services on remote nodes. The services manager can start, stop, and allow remote execution of services.

Package Dependencies: QT 2.3.0 (*window development libraries*)

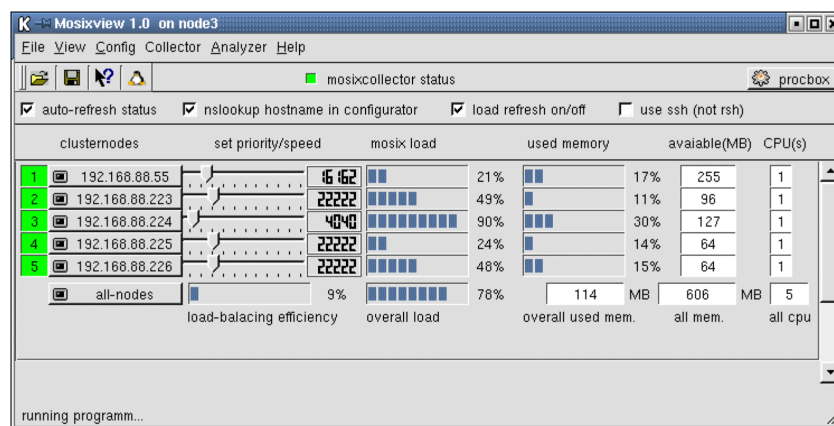


Figure 4-4 – Main Window of Mosixview Cluster Management Software

4.4. Applications Testing

4.4.1. Persistence of Vision

Persistence of Vision Ray-tracer or POV-Ray is a ray-tracing program that creates three-dimensional, photo-realistic images and animations using the ray tracing rendering technique. It reads in a text file denoted with extension `.pov`, containing information describing the objects and lighting in a scene and generates an image of that scene from the viewpoint of a camera also described in the text file.

Ray-tracing is a rendering technique that calculates an image of a scene by simulating the way rays of light travel in the real world. However, it does its job backwards by starting with a simulated camera and traces rays backwards out into the scene. It is done backwards as the vast majority of rays never hit an observer, hence it would take forever to trace a scene. Ray tracing is not a fast process by any means, but it produces very high quality images with realistic reflections, shading, perspective, and other effects as shown in Figure 4-6.

Standard Package Dependencies: libpng (*library functions for PNG image manipulation*)

Animation rendering is an excellent example of **process-level parallelism** and as POV-Ray is freely available for Linux (*as well as many other platforms*), it is an excellent application for parallel execution and benchmarking. [12] [3.1.5]

In fact, much data is available on benchmarking computer systems with POV-Ray. Shown in Figure 4-5 is the result of the standard script `skyvase.pov` [8.4.1], which was introduced by Andrew Haveland-Robinson as a benchmark for POVRay on many different computer systems.

Figure 4-5 – Ray Tracing Bench Mark Test Output

Table 8-3 [refer Appendix E] contains a data extract of the performances of various clusters, similar in power to the test-cluster, on this benchmark.

The test-cluster will be benchmarked using the `skyvase.pov` as shown in Figure 4-5, `poolballs.pov` as shown in Figure 4-7, and `tulips.pov` as shown in Figure 4-8. [43]

Figure 4-6 – Sample Output of Povray's abilities

Figure 4-7 – Alternate POV-Ray Test Case 1

Figure 4-8 – Alternate POV-Ray test Case II

POV-Ray is an open source project, and hence there are many different versions available, including a wide range of plugins and patches for various compatibility and extensibility.

Some of the plugins available include:

- Graphical User Interface.
- Plugin for compatibility with RenderMan.
- Plugin for compatibility with AutoCAD.
- Patch for PVM to operate POV-Ray in parallel with PVM.
- Patch for MPI to operate POV-Ray in parallel with MPI/MPICH.

One of the patches that was not available was explicit support for the LAM implementation of MPI. Since this was the desired distribution due to performance and compatibility, it was decided to use the MPICH version of the patch [MPICH/POV-Ray patch by Leon Verrall

www.verrall.demon.co.uk/mpipov/] and modify for LAM compatibility. However, it was found that no modifications were required and compilation of the new version of MPI-POV-Ray using the existing Linux source code to suit LAM was successfully completed.

Theory of MPI-Povray

Using the new version of MPI-POV-Ray, the system operates with one master task and many slave tasks. The master has the responsibility of dividing the image into small blocks that are assigned to slaves. When a slave has finished rendering a block, it is sent back to the master. The master combines them to form the final image. The master task does not render anything itself, and hence does not use much CPU power. For better utilization of the master, a slave task (*i.e. rendering*) can be run on the master.

MPI POV-Ray adds the following additional commands line options to standard POV-Ray:

```
+NHxxx Height of MPI Chunks

+NWxxx Width of MPI chunks
```

The options control the dimensions of the chunks that get assigned to slave tasks. In general, this allows customisation to suit the particular architecture and system performance that MPI-PO-Ray is operating on. In general, it is a trade-off between message passing overhead and division of labour.

If the chunk size is set too high, it is likely that one PE (*processing element*) will get a difficult section of the image to render, and this will be come rate-limiting for the while job. If the chuck size is too small, much processing time is spent passing messages (*or blocking on Sen/Receive messages*), hence under utilizing CPU power.

As a guide if the image is equally difficult to render in all parts of the image and less than 16 processors are available, then a larger chunk size is recommended (*try to divide the image equally between the available PE's*).

If the image varies in difficulty or there are a large number of processors available, then the default chunk size of 32x32 (*in a checker configuration*) is recommended. [43] [1]

With animations, although not fully tested in this document, each PE is assigned its own frame.

5. Results

5.1. Summary of Numerical Data

This section details the results of applications testing and includes a results analysis. The raw results obtained from the test cluster, including scripts of command line input/output can be found in Appendix E.

From the large amount of data and test results obtained in the laboratory, the most interesting are detailed in this section (*applications level only*) to show the various speedup indices of a cluster computer. Shown in Table 5-1 below, are the results of the rendering test cases as detailed in section 4.4.1.

	skyvase.pov		poolballs.pov			tulips.pov	
No. of CPUs	**Render Time** *(seconds)*	**Speedup**	**Render Time** *(seconds)*	**Speedup over X-POV-Ray[1]**	**Speedup on MPI-X-POV-Ray[2]**	**Render Time** *(seconds)*	**Speedup**
1	129	1.0	2541[1] / 1786[2]	1.0	1.0	279264	1.0
2	72	1.8	906	2.8	2.0		
3	48	2.7	605	4.2	3.0		
4	36	3.6	457	5.6	3.9		
5	29	4.4	364	7.0	4.9		
6	25	5.2	309	8.2	5.8		
7	22	5.9	262	9.7	6.8		
8	20	6.5	229	11.1	7.8	Failed	

Table 5-1 – Rendering Test Results

5.2. Results Analysis

With respect to Table 5-1 the following analysis of the numerical results is presented:

Skyvase

The `skvase.pov` test case is the benchmark for a cluster computer using parallel rendering. The results show that a near ideal speedup was achieved.

From the POVBENCH benchmark data [Table 8-3] it can be seen that for a single node the test cluster achieved a POVmark of 114.73 which is considered above average for the hardware used (*based on the POVmark data*). This can be attributed to the use of structured design and implementation of Linux on each node as well as efficiencies included in the implemented Linux kernel version 2.4.7-10.

When rendered on all eight nodes the test cluster achieved a POVmark of 740.00. When comparing the cluster test results with the POVBENCH cluster data, the test cluster takes the 86th spot with a parallel rendering time of 20sec, out of the 236 parallel results submitted. This is just below the 85th spot with a 16 node cluster of Pentium Pro 200MHz machines with a rendering time of 19sec.

These high-performing results can be attributed to the high-performing nodes, a 100BaseTX switched Ethernet fabric and the use of the high performance MPI-library LAM for POV-Ray that is not commonly used.

Poolballs

Of all of the test-cases the Poolballs test results contained the most interesting trend.

The test case was run under two scenarios, as follows:

1. **X-POV-Ray** – Poolballs was first run on one node using the standard serial version of POV-Ray. This resulted in a best execution time of 2541 seconds. All further results were obtained using the parallel version as required for multiple CPU's. It can be seen from the results that a **super-linear speedup** was achieved over the serial version.
2. **MPI-X-POV-Ray** – Whilst the first scenario was run many times, each time showing a super-linear speedup was achieved, it was decided to test the case when the Parallel version is run on one node using a master and slave process. This required the operating system to multitask between the two processes and it was expected that a similar result to the serial version would be achieved. However as shown in Table 5-1, the parallel version out-performed the serial version on one node. This changes the speedup results when compared to the parallel version, showing a close to ideal speedup.

Tulips

The tulips test case has a high degree of difficulty and was used to test the stability of the cluster. Additionally it was intended to provide a longer test case which could show the trend as many messages are sent and received over the network, increasing the load on the server, the communications medium, and each node.

Unfortunately, the test case took over 77 hours to render on one node and as such results were not obtained for each combination of processors. The test case was run on all eight nodes, however it was apparent that after five days that it had crashed the cluster, and due to time constraints further testing to get it running to completion was not possible. Hence, this test case has not provided any information on speedup index.

From the work that was done using the rendering test cases, in particular Tulips, it is theorized, that as the resources are heavily utilized the speedup index decreases to the lower bound Log_2n.

Shown in Figure 5-1 below is a plot of the salient test results: skyvase, poolballs super linear speedup and the applicable theoretical expectations as derived in section 3.1.3.

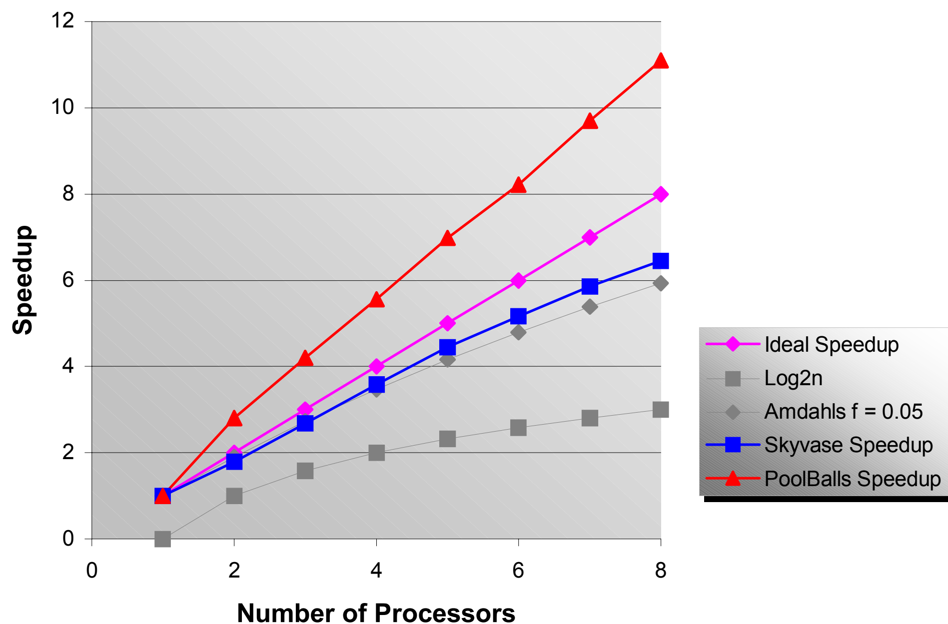


Figure 5-1 – Test Results & Theoretical Performance Comparison

It was noted from the raw test results produced by MPI-POV-Ray, that each node was not contributing an equal percentage of work. Whilst MPI-POV-Ray divides an image up equally, it does not evaluate the difficulty of each chunk. Each chunk is different and takes a different time to render. As compensation to this, MPI-POV-Ray distributes small chunks (*also chunk size can be modified by the user to suit the particular architecture*) such that the disparity is minimised. However, this disparity does lead to a decrease in the overall speedup index. Whilst this is minimal for the rendering algorithm, as it can be considered symmetric, this is not the case for all algorithms, and applications.

[Refer to Appendix E – 8.5.2: Within each multi-CPU test case are PE (PE) Distribution Statistics. These statics detail the percentage of work that each node completed].

The following conclusions can be draw from these results:

- The performance of a Beowulf Cluster can be modelled by a pessimistic Lower bound **Log_2n** for **$n \leq 8$** and the upper bound, which is the ideal case **n.**
- **Amdahl's law** with f=0.05 models more closely the lower bound than **Log_2n** however it is noted that both are highly pessimistic models when trended out to larger numbers of processors such as 64, 128 and 512.
- In some instances super-linear speedups can be achieved, however only with tree-search type applications or inherently parallel algorithms.
- Through the process of structured design, a high-performing, cost-effective Cluster Computer can be built with known performance bounds.
- Load Balancing for nodes is generally required. Should this feature not be available it is possible to manually balance work using the weighted harmonic mean principle [as detailed in 3.1.3]. The outcome of this principle is that each node should have the same physical hardware specification and performance. This enables algorithms that are naturally well balanced (*i.e. symmetric such as parallel rendering*) to not require a load-balancing facility to work effectively and achieve a high speedup index. This methodology can cater effectively for the general-purpose case with a symmetric algorithm. Whilst rates will invariably become out of balance, even in an application such as MPI-POV-Ray [43], the imbalance will be minimised.

6. Conclusion

The Beowulf cluster computer is classified as a MIMD, loosely coupled class computer using the message-passing model for node intercommunication. It is built from commodity class components enabling it to be cost effective, in comparison to traditional super-computers.

The motivation to build cluster computers can be seen from the industries that use these systems. They generally require large amounts of cost-effective computing power, to derive results in a timely fashion.

The Beowulf class of computers were started in 1994 by NASA and have grown immensely since, which can be seen by the proliferation in the world top500 supercomputers as well as the available software for the architecture.

Beowulf clusters can be effectively designed to suit different applications depending on the performance requirements and the commodity hardware available. Cost-savings are introduced with the use of commodity components available in the open market as well as the many system choices that can be made (*such as Hardware and Operating Systems*) rather than specific vendor solutions.

Beowulf-class computers can be described as:

- Cost-effective.
- Easy to assemble.
- Reliable.
- Deliver on wide range of carefully chosen applications.

However, Beowulf-class computers are not:

- Appropriate for all supercomputer applications.
- Infinitely scalable.
- Not for all applications and every situation.
- Not user-intuitive to set-up and use.

This document has shown that the best approach to building high performance clusters is using a layered approach. Each layer can be looked at individually selecting the highest performing and most cost-effective solution. The results of this analysis are:

- The Linux operating system has the features and abilities of commercially available operating systems, in some cases out-performing them with speed and the ability to reduce the footprint to a minimal size. Linux one of very few operating systems to offer these features with the added benefit of an open licensing arrangement, reducing the overall cost of each node for a Beowulf cluster.

- MPI offers many features for the parallel programmer and functionality as the lowest layer of middleware. The most notable is the portability between systems and performance. In the coming years the MPI standard will completely overrun the use of PVM, when it includes the fault-tolerance, load-balancing and other mature features it currently lacks. As with Linux, open source and free distributions of MPI are available for use with Linux clusters. One of the highest performing, with the highest degree of compatibility with the current version of the MPI standard is the Local Area Multicomputer (LAM) distribution.

- Broad testing of the Beowulf cluster is required to optimise the platform and provide verification of the design objectives. Many test suites are available for clusters,

however applications level testing should be completed such as Parallel rendering with MPI-POV-Ray.

As part of the work outlined in this document, theoretical performance figures were taken in to the laboratory for verification. This process included the design, set-up and commissioning of a high-performance Beowulf cluster using the Linux Operating System, the LAM MPI message passing library and a pseudo-novel implementation of the ray-tracing rendering and animation program POV-Ray that was built for LAM/MPI.

During the testing process it was found that in some instances running an application in parallel can result in high performance, and that parallel applications can gain super-linear speedups over their serial counterparts. These efficiencies can only be tapped with parallel systems and for certain applications.

The results show great variability in the performance of an application, based on the input data. Whilst there is variance in performance, it was shown that it is possible to determine the bounds of performance prior to implementation, thus leading to a meaningful design process.

Load balancing, the technique of ensuring that each node is processing it fair share of work and is not under or over utilised, is not necessarily required to produce high-performance speedups on symmetric algorithms. It is possible through system design and application implementation to minimise the requirement for load balancing. However, it is a recommended facility, integral to the middleware layer of software and implemented in packages such as PVM and in the future MPI. In some instances, it is part of the Operating System (*as with the Cluster Versions of Windows 2000 and Linux Redhat*).

The coming parallel software generation will see a myriad of languages utilized, all offering varying degrees of abstraction and efficiency. In the long term it will be most likely that a few specialised languages will become dominant in the main stream parallel applications development market. Efficiency will be compromised for ease of development. In contrast the high performance scientific community will continue to use and develop specialised scientific, portable and highly efficient languages for their own use. These languages will have abilities to tweak applications for different platforms to maximise the performance without code-level modifications.

The Future of Clusters

Clusters have an ever growing base in the following areas:

- Scientific and Engineering research (*Universities, Government and Industry*).
- Education.
- Computer Science.
- Corporations.
- Media Development Industries.

As the cost of hardware and storage continues to decline, the use of clusters will increase dramatically. Already most database vendors support SMP machines and several, including Oracle, are starting to release versions of their software for Linux clusters. As these mainstay packages become available, they will drive the availability of a whole new class of applications that run on these machines, ranging from serious business applications to entertainment applications such as online gaming systems.

Another effect of the ever-increasing capability of computer hardware is its ever-decreasing size. Traditional supercomputers are very large and until recently, clusters were even larger since they are, by definition, collections of rack-mounted workstations. As system sizes decrease, the physical size of a cluster will decrease as well, while the overall computational capabilities will increase. With the current availability of two nodes in one RU rack space, a 64-node cluster can fit in a single 19” rack, and with single-board computers, the same cluster is able to fit under a desk.

7. References

Other documents referenced within this document:

1. Beowulf How-to

Beowulf UnderGround Website
Version 1.1.1, November 1998
Jacek Radjewski and Douglas Eadline
Available on the Internet at:
http://www.beowulf-underground.org/

1) Mosixview – http://www.beowulf-underground.org/software.html

2. PVM: Parallel Virtual Machine

A Users' Guide and Tutorial for Networked Parallel Computing
Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, Vaidy Sunderam
The MIT Press
Cambridge, Massachusetts, London, England
1994 Massachusetts Institute of Technology
Available on the Internet at:
http://www.netlib.org/pvm3/book/pvm-book.html

3. MPI: The Complete Reference

MPI: The Complete Reference
Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker, Jack Dongarra
The MIT Press
Cambridge, Massachusetts
London, England

4. Beowulf at NASA/GSFC

Beowulf at NASA/GSFC
Earth and Space Sciences Project
NASA Goddard Space Flight Center
http://beowulf.gsfc.nasa.gov/

Website now maintained by Scyld Computing Corporation:
1) http://www.beowulf.org/software/bonding.html – Beowulf Channel Bonding
2) http://www.beowulf.org/intro.html – Beowulf Introduction

5. Modern Operating Systems

Andrew S. Tanenbaum
Vrije Universiteit
International Edition 1992
Prentice-Hall International, Inc

6. **Computer Architecture and Parallel Processing**
Kai Hwang and Fayĕ A. Briggs
International Edition 1985
McGraw Hill

7. **An Introduction to the Intel Family of Microprocessors**
A Hands-On Approach Utilzing the 8088 Microprocessor
James L. Antonakos
Second Edition 1996
Prentice Hall

P41 – Pipelining in the Pentium and 486 chip

8. **Programming Distributed Systems**
Henri Bal
First Edition 1990
Prentice Hall

9. **Introduction to Parallel Processing**
Algorithms and Architectures
Behrooz Parhami
First Edition 1999
Plenum Press

P15 – 1.4 Types of Parallelism: Taxonomy
P78 – Table 4.2 Topological Parameters of Selected Interconnection Newtorks

10. **Pocket Glossary of Computer Terms**
Including Technical Reference Material
Black Box Corporation
Third Edition 1999

11. **Beowulf Installation and Administration How-to**

Beowulf UnderGround Website
Version 0.1.2, June 1999
Jacek Radjewski and Douglas Eadline
Available on the Internet at:
http://www.beowulf-underground.org/

12. **How to Build a Beowulf**
A Guide to the Implementation and Application of PC Clusters
Thomas L. Sterling, John Salmon, Donald J. Becker, Daniel F. Savarese
First Edition 1999
The MIT Press
Cambridge, Massachusetts
London, England

P11 – Figure 2.1 Classification Scheme for Parallel Computing

13. RFC 1918

Address Allocation for Private Internets
Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, E. Lear
Network Working Group
February 1996
Obsoletes RFC: 1627, 1597
BCP: 5 Category: Best Current Practice

Available on the Internet at:
http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc1918.html

14. Parallel Processing

From Applications to Systems
Dan I. Moldovan
First Edition 1993
Morgan Kaufmann

P26 – 1.4 Performance of Parallel Computations

15. Orca

Report on the Programming Language Orca
Henri E. Bal
Dept. of Mathematics and Computer Science
Vrije Universiteit
Amsterdam, The Netherlands
May 1994

16. A Programmers Guide to ZPL

Lawrence Snyder
Department of Computer Science and Engineering
University of Washington
Seattle WA 98195
Version 6.3, January 6, 1999

Available on the Internet at:
http://www.cs.washington.edu/research/zpl/

17. Bulk Synchronous Parallel

BSP Clusters: High performance, Reliable & Very Low Cost
Stephen Donaldson and Jonathan M.D. Hill
Programming Research Group, University of Oxford, U.K.

Questions & Answers About BSP
Stephen Donaldson and Jonathan M.D. Hill

Programming Research Group, University of Oxford, U.K.

18. Logic and Computer Design Fundamentals

M. Morris and Charles R. Kime
International Edition 1997
Prentice Hall

P572 – 12-2 Locality of Reference

19. LAM / MPI Parallel Computing

Website http://www.lam-mpi.org/

1) http://www.lam-mpi.org/mpi/mpi_top10.php – MPI vs. PVM
2) http://www.lam-mpi.org/mpi/implementations/ – MPI implementations
3) http://www.lam-mpi.org/linux/ – LAM/Linux issues
4) http://www.lam-mpi.org/performance.php – LAM vs. MPICH performance comparisons

20. MPI Forum

Message Passing Interface Standards Website
MPI 2 Standard available online
Website http://www.mpi-forum.org/

21. PVM and MPI

A Comparison of Features
G. A. Geist, J. A. Kohl, P. M. Papadopoulos
May 30, 1996
Note this paper compares MPI 1.2 with PVM 3.4
http://www.epm.ornl.gov/pvm/PVMvsMPI.ps

22. Redhat

Commercial and Free Version of Linux
Software Website
http://www.redhat.com/

23. Intel

Corporate Website
http://www.intel.com/

2) http://www.intel.com/eBusiness/products/enterprise/8way/8wayQRG.htm – SMP Block diagram

24. Los Alamos National Laboratory

Centre for Non-Linear Studies
Avalon Beowulf Cluster
http://cnls.lanl.gov/Internal/Computing/Avalon

25. University of Oxford Parallel Applications Centre

The Bulk Synchronous Parallel Model
University Website:
http://oldwww.comlab.ox.ac.uk/oucl/oxpara/bsp/bspmodel.htm

26. University of California NOW Project

Conducted by the Computer Science Division
University of California, Berkeley
University Website: http://now.cs.berkeley.edu/

27. Scyld Computing Corporation

Corporate Website
http://www.scyld.com/

28. A System Software Architecture for High-End Computing

David S. Greenberg et al
Sandia National Laboratories
http://www.supercomp.org/sc97/proceedings/TECH/GREENBER/INDEX.HTM

29. Top 500 Supercomputer Sites

Benchmark performance and ranking the of the top 500 Supercomputers
University of Mannheim
University of Tennessee
All information was extracted from the November 2001 Top 500 List
http://www.top500.org/
1) http://www.top500.org/lists/linpack.html - Linpack Config
2) http://www.top500clusters.org/ - Top 500 Clusters

30. Linux Parallel Processing How-to

Purdue University
Version 980105, 5 January 1998
Hank Dietz
Available on the Internet at:
http://yara.ecn.purdue.edu/pplinux/

31. High Performance Cluster Computing

Volume 1 – Architectures & Systems
Volume 2 – Programming & Applications
Edited By Rajkumar Buyya
Monash University Melbourne, Australia
First Edition 1999
Prentice Hall PTR

P10 – Figure 1.2 Cluster Computer Architecture

32. Operating Systems

Internals and Design Principles
William Stallings
Fourth International Edition 2001
Prentice-Hall International, Inc

33. Electronic Design

Technology-Applications-Products-Solutions
http://www.elecdesign.com/
Penton

August 17, 1998 Issue
Page 30 - Mail-Order Supercomputer available for a mere $150,000

34. Supporting Microsoft Cluster Server

Microsoft Official Curriculum
CD-ROM Version
December 1, 1997

35. Multi-Threaded Object-Oriented MPI-Based Message Passing

The ARCH Library
Jean-Marc Adamo
First Edition 1998
Kluwer Academic Publishers
http://www.cpe.fr/~arch/ARCH-index.htm

Chapter 8 – Parallel A Algorithm*

36. Parallel Algorithms/Architecture Synthesis

The Second Aizu International Symposium
March 17-21 Japan
IEEE Computer Society Press

Page 340 – A Parallel & Fault-Tolerant LAN with Dual Communications Subnetworks

37. Models and Languages for Parallel Computation

David B. Skillicorn & Domenico Talia
October 1996

38. Paralogic Inc.

Corporate Website
http://www.plogic.com/index.html

http://www.xtreme-machines.com/x-cluster-qs.html - Cluster Quick Start
ftp://ftp.plogic.com - Ethernet Channel Bonding Kernel Patch

1) Paper: Performance Considerations for I/O Dominant Applications on Parallel Computers
2) Beowulf Performance Suite – http://www.plogic.com/bps

39. GNU General Public License

Version 2, June 1991
Copies of the license can be down found at
http://www.fsf.org/copyleft/gpl.html

40. The Parallel Virtual File System

Research Website
Parallel Architecture Research Laboratory
Clemson University, Clemson, South Carolina
http://parlweb.parl.clemson.edu/pvfs/index.html

41. Linux NFS How-to

28 December 2001
Tavis Barr, Nicolai Langfeldt, Seth Vidal
Available on the Internet at:
http://nfs.sourceforge.net

42. Designing and Building Parallel Programs

28 December 2001
Ian Foster
Online book available on the Internet at:
http://www-unix.mcs.anl.gov/dbpp/

43. Persistence of Vision Ray-tracing

1) Main website – http://www.povray.org/
2) Benchmarking – http://www.haveland.com/index.htm?povbench/index.htm
3) Examples inc .pov files – http://www.xlcus.com/povray/

All documentation is available (*depending on issue status*) upon request via e-mail from Richard Morrison (r.morrison@ndy.com) quoting the document name and number.

 Copyright © 2003

8. Appendix

This section of the document contains the appendices that support the work presented in the main document.

 Copyright © 2003

8.1. Appendix A – Node Interconnection Technologies

This appendix details commonly used networking technologies [30] used to interconnect cluster nodes. It specifically contains the information required to evaluate, design and implement a Beowulf cluster network for a specific requirement. This section is divided into two categories as follows:

- **Class 1** – Commodity components (*for use in Beowulf Class 1 Clusters*)
- **Class 2** – Vendor specific components (*for use in Beowulf Class 2 Clusters*)

[1] Denotes at the time of writing the information has not been confirmed.

8.1.1. Class 1 Network Hardware

CAPERS

CAPERS (*Cable Adapter for Parallel Execution and Rapid Synchronization*) is a spin-off of the PAPERS project, http://garage.ecn.purdue.edu/~papers/, at the Purdue University School of Electrical and Computer Engineering. In essence, it defines a software protocol for using an ordinary "LapLink" parallel SPP-to-SPP cable to implement the PAPERS library for two Linux PCs. The network do not scale, but the price cannot be beaten. As with TTL_PAPERS, to improve system security, there is a minor kernel patch recommended (but not required) and is available at: http://garage.ecn.purdue.edu/~papers/giveioperm.html

Design Considerations:

Linux support: AFAPI library
Maximum bandwidth: 1.2 Mb/s
Minimum latency: 3 microseconds
Available as: commodity hardware
Interface port/bus used: SPP
Network structure: cable between 2 machines
Cost per machine connected: $2

10Mb Ethernet

For some years now, 10 Mbits/s Ethernet has been the standard network technology. Good Ethernet interface cards can be purchased for well under $100, and a fair number of PCs now have an Ethernet controller built-into the motherboard. For lightly-used networks, Ethernet connections can be organized as a multi-tap bus without a hub; such configurations can serve up to 200 machines with minimal cost, but are not appropriate for parallel processing. Adding an unswitched hub does not really help performance. However, switched hubs that can provide full bandwidth to simultaneous connections cost only about $150 per port. Linux supports an extensive range of Ethernet interfaces, but it is important to consider that variations in the interface hardware can yield significant performance differences. See the Linux Hardware Compatibility HOWTO for comments on which are supported and how well they work; also see http://cesdis1.gsfc.nasa.gov/linux/drivers/.

An interesting way to improve performance is offered by the 16-machine Linux cluster work done in the Beowulf project, http://cesdis.gsfc.nasa.gov/linux/beowulf/beowulf.html, at NASA CESDIS. There, Donald Becker, who is the author of many Ethernet card drivers, has developed support for load sharing across multiple Ethernet networks that shadow each other (*i.e., share the same network addresses*). This load sharing is built-into the standard Linux distribution, and is done invisibly below the socket operation level. Because hub cost is significant, having each machine connected to two or more hubless or unswitched hub

Ethernet networks can be a very cost-effective way to improve performance. In fact, in situations where one machine is the network performance bottleneck, load sharing using shadow networks works much better than using a single switched hub network.

Design Considerations:

Linux support: kernel drivers

Maximum bandwidth: 10 Mb/s

Minimum latency: 100 microseconds

Available as: Commodity hardware

Interface port/bus used: PCI

Network structure: switched or unswitched hubs, or hubless bus

Cost per machine connected: $150 (hubless, $100)

100Mb Ethernet (Fast Ethernet)

Fast Ethernet refers to a hub-based 100 Mbits/s Ethernet that is somewhat compatible with older "10 BaseT" 10 Mbits/s devices and cables. As might be expected, anything called Ethernet is generally priced for a volume market, and these interfaces are generally a small fraction of the price of 155 Mbits/s ATM cards. A drawback is that having a collection of machines dividing the bandwidth of a single 100 Mbits/s "bus" (*using an unswitched hub*) yields performance that is not as good on average as using 10 Mbits/s Ethernet with a switched hub that can give each machine's connection a full 10 Mbits/s.

Switched hubs that can provide 100 Mbits/s for each machine simultaneously are expensive, but prices are dropping rapidly per quater, and these switches do yield much higher total network bandwidth than unswitched hubs. The thing that makes ATM switches so expensive is that they must switch for each (*relatively short*) ATM cell; some Fast Ethernet switches take advantage of the expected lower switching frequency by using techniques that may have low latency through the switch, but take multiple milliseconds to change the switch path. Should the design routing pattern be expected to change frequently, then these switches should be avoided. See http://cesdis1.gsfc.nasa.gov/linux/drivers/index.html for information about the various cards and drivers.

Channel Bonding is also available as described in Section 3.2.3.1.

Design Considerations:

Linux support: kernel drivers

Maximum bandwidth: 100 Mb/s

Minimum latency: 80 microseconds

Available as: commodity hardware

Interface port/bus used: PCI

Network structure: switched or unswitched hubs

Cost per machine connected: $400[1]

1000Mb Ethernet (Gigabit Ethernet)

Gigabit Ethernet, http://www.10gea.org/Tech-whitepapers.htm, does not have a good technological reason to be called Ethernet. However the name does accurately reflect the fact that this is intended to be a cost effective, mass-market, computer network technology with native support for IP. However, current pricing reflects the fact that Gb/s hardware is still complex hardware to manufacture.

Unlike other Ethernet technologies, Gigabit Ethernet provides for a level of flow control that should make it a more reliable network. Full-Duplex Repeaters (FDRs) or, simply multiplex lines, using buffering and localized flow control to improve performance. Most switched hubs are being built as new interface modules for existing gigabit-capable switch fabrics. Switch/FDR products have been shipped or announced by the majority of networking vendors such as:

- http://www.acacianet.com/
- http://www.baynetworks.com/
- http://www.cabletron.com/
- http://www.networks.digital.com/
- http://www.extremenetworks.com/
- http://www.foundrynet.com/
- http://www.gigalabs.com/
- http://www.packetengines.com/
- http://www.plaintree.com/
- http://www.prominet.com/
- http://www.sun.com/
- http://www.xlnt.com/

NIC’s and Switches are now available using a Category 5e-cabling infrastructure, rather than requiring the use of optical fibre. This is significantly driving the cost down as copper cabling is less expensive and existing copper-cable plants can be reused.

There is a Linux driver, http://cesdis.gsfc.nasa.gov/linux/drivers/yellowfin.html, for the Packet Engines "Yellowfin" G-NIC, http://www.packetengines.com/. Early tests under Linux achieved about 2.5x higher bandwidth than could be achieved with the best 100 Mb/s Fast Ethernet; with gigabit networks, careful tuning of PCI bus use is a critical factor. There is little doubt that driver improvements, and Linux drivers for other NICs, will follow.

Design Considerations:

Linux support: kernel drivers

Maximum bandwidth: 1,000 Mb/s

Minimum latency: 300 microseconds[1]

Available as: multiple-vendor hardware

Interface port/bus used: PCI

Network structure: switched hubs or FDRs

Cost per machine connected: $2,500[1]

10G Ethernet (10 Gigabit Ethernet)

10 Gigabit Ethernet, http://www.10gea.org/Tech-whitepapers.htm is currently the latest evolution of the Ethernet protocol.

NIC’s and Switches are now available using single and multimode optical fibre interfaces.

On 20 March 2003, the IEEE approved the development of 10GBASE-CX4 to be IEEE 802.3ak, which will allow the operation of 10Gb/s Ethernet over 15metres of copper cable. The current draft of this standard specifies shielded 100Ω twisted pair cabling with specialist connectors. It is considered that clustering will be an application of the new standard when products are released. Follow updates at: http://grouper.ieee.org/groups/802/3/ak/index.html

Additionally by approximately 2006 it is envisaged that 10GBaseT will be released by the IEEE, providing copper interfaces and an ability to run 100m on TIA/EIA 568-A or ISO/IEC 11801 cabling infrastructure. It is understood at this stage that possibly a modification will be released by ISO/IEC to Category 6, producing perhaps a Category 6e (similar to Category 5e) or a new category of cabling suited to the implementation of the protocol over copper. Follow the developments at: http://grouper.ieee.org/groups/802/3/10GBT/index.html

Design Considerations:

Linux support: yes

Maximum bandwidth: 10,000 Mb/s

Minimum latency: tba

Available as: multiple-vendor hardware

Interface port/bus used: PCI-X

Network structure: switched hubs or FDRs

Cost per machine connected: $tba[1]

PLIP

For the minimal cost of a "LapLink" cable, PLIP (*Parallel Line Interface Protocol*) allows two Linux machines to communicate through standard parallel ports using standard socket-based software. In terms of bandwidth, latency, and scalability, this is not a very serious network technology; however, the near-zero cost and the software compatibility are useful. The driver is part of standard Linux kernel distributions.

Design Considerations:

Linux support: kernel driver

Maximum bandwidth: 1.2 Mb/s

Minimum latency: 1,000 microseconds?

Available as: commodity hardware

Interface port/bus used: SPP

Network structure: cable between 2 machines

Cost per machine connected: $2

SLIP

Although SLIP (*Serial Line Interface Protocol*) is firmly planted at the low end of the performance spectrum, SLIP (*or CSLIP or PPP*) allows two machines to perform socket communication via ordinary RS232 serial ports. The RS232 ports can be connected using a null-modem RS232 serial cable, or they can even be connected via dial-up through a modem. In any case, latency is high and bandwidth is low, so SLIP should be used only when no other alternatives are available. It is worth noting, however, that most PCs have two RS232 ports, so it would be possible to network a group of machines simply by connecting the machines as a linear array or as a ring. There is also load-sharing software called EQL.

Design Considerations:

Linux support: kernel drivers

Maximum bandwidth: 0.1 Mb/s

Minimum latency: 1,000 microseconds[1]

Available as: commodity hardware

Interface port/bus used: RS232C

Network structure: cable between 2 machines

Cost per machine connected: $2

USB (Universal Serial Bus)

USB (*Universal Serial Bus,* http://www.usb.org/) is a hot-pluggable conventional-Ethernet-speed, bus for up to 127 peripherals ranging from keyboards to video conferencing cameras. Computers are interconnected using a USB hub (*similar to Ethernet*) or interconnected to the adjacent node(s) (*PC's generally come with two USB ports, these can be split to interconnect more than two nodes if required*).

USB is considered the low-performance, zero-cost, version of FireWire.

USB ports are a defacto standard on PC motherboards similar to as RS232 and SPP. Development of a Linux driver is discussed at http://peloncho.fis.ucm.es/~inaky/USB.html

Design Considerations:

Linux support: kernel driver

Maximum bandwidth: 12 Mb/s

Minimum latency: Data not available

Available as: commodity hardware

Interface port/bus used: USB

Network structure: bus

Cost per machine connected: $5 – depending on network topology

8.1.2. Class 2 Network Hardware

Myrinet

Myrinet http://www.myri.com/ is a LAN designed to also serve as a 'System Area Network' (SAN), i.e., the network within a cabinet full of machines connected as a parallel system. The LAN and SAN versions use different physical media and have different characteristics; generally it is recommended to use the SAN version within a cluster.

Myrinet is conventional in structure and has a reputation for being a particularly good network implementation. The drivers for Linux are said to perform very well, although large performance variations have been reported with different PCI bus implementations for the host computers.

Currently, Myrinet is a preferred network technology by cluster-groups (*as it has one of the smallest latencies*) that do not have significant budgetary constraints.

Design Considerations:

Linux support: Library

Maximum bandwidth: 1,280 Mb/s

Minimum latency: 9 microseconds

Available as: single-vendor hardware

Interface port/bus used: PCI

Network structure: Switched hubs

Cost per machine connected: $1,800

Parastation

The ParaStation project http://wwwipd.ira.uka.de/parastation at University of Karlsruhe Department of Informatics is building a PVM-compatible custom low-latency network. They first constructed a two-processor ParaPC prototype using a custom EISA card interface and PCs running BSD UNIX, and then built larger clusters using DEC Alphas. Since January 1997, ParaStation has been available for Linux. The PCI cards are being made in cooperation with a company called Hitex (see http://www.hitex.com/parastation/). Parastation hardware implements both fast, reliable, message transmission and simple barrier synchronization.

Design Considerations:

Linux support: HAL or socket library

Maximum bandwidth: 125 Mb/s

Minimum latency: 2 microseconds

Available as: single-vendor hardware

Interface port/bus used: PCI

Network structure: Hubless mesh

Cost per machine connected: > $1,000

ArcNet

ARCNET is a local area network that is primarily intended for use in embedded real-time control systems. Like Ethernet, the network is physically organized either as taps on a bus or one or more hubs, however, unlike Ethernet, it uses a token-based protocol logically structuring the network as a ring. Packet headers are small (*3 or 4 bytes*) and messages can carry as little as a single byte of data. Thus, ARCNET yields more consistent performance than Ethernet as it has bounded delays. Unfortunately, it is slower than Ethernet and less popular,

making it more expensive. More information is available from the ARCNET Trade Association at http://www.arcnet.com/

Design Considerations:

Linux support: kernel drivers

Maximum bandwidth: 2.5 Mb/s

Minimum latency: 1,000 microseconds[1]

Available as: multiple-vendor hardware

Interface port/bus used: ISA

Network structure: unswitched hub or bus (logical ring)

Cost per machine connected: $200

ATM

ATM (*Asynchronous Transfer Mode*) has been heralded over the past few years to take over Ethernet, however this is unlikely to happen. ATM is cheaper than HiPPI and faster than Fast Ethernet, and it can be used over the very long distances and as such is used in Telco applications. The ATM network protocol is also designed to provide a lower-overhead software interface and to more efficiently manage small messages and real-time communications (*e.g., digital audio and video*). It is also one of the highest-bandwidth networks that Linux currently supports. The drawback is that ATM is not cheap, and there are still some compatibility problems across vendors. An overview of Linux ATM development is available at http://lrcwww.epfl.ch/linux-atm/

Design Considerations:

Linux support: kernel driver, AAL* library

Maximum bandwidth: 155 Mb/s (*soon, 1,200 Mb/s*)

Minimum latency: 120 microseconds

Available as: multiple-vendor hardware

Interface port/bus used: PCI

Network structure: switched hubs

Cost per machine connected: $3,000

FC (Fibre Channel)

The goal of FC (*Fibre Channel*) is to provide high-performance block I/O (*an FC frame carries a 2,048 byte data payload*), particularly for sharing disks and other storage devices that can be directly connected to the FC rather than connected through a computer. Bandwidth-wise, FC is specified to be relatively fast, running anywhere between 133 and 1,062 Mbits/s. If FC becomes popular as a high-end SCSI replacement, it may quickly become a cost-effective technology; for now, it is not cost-effective and is not supported by Linux. The Fibre Channel Association at maintains a good collection of FC references:

http://www.amdahl.com/ext/CARP/FCA/FCA.html

Design Considerations:

Linux support: no

Maximum bandwidth: 1,062 Mb/s

Minimum latency: Data not available

Available as: multiple-vendor hardware

Interface port/bus used: PCI?

Network structure: Data not available

Cost per machine connected: Data not available

FireWire (IEEE 1394)

FireWire, http://www.firewire.org/, the IEEE 1394-1995 standard, is destined to be the low-cost high-speed digital network for consumer electronics. The showcase application is connecting DV digital video camcorders to computers, however FireWire is intended to be used for applications ranging from being a SCSI replacement to interconnecting the components of your home theatre. It allows up to 64K devices to be connected in any topology using busses and bridges that does not create a cycle, and automatically detects the configuration when components are added or removed. Short (*four-byte "quadlet"*) low-latency messages are supported as well as ATM-like isochronous transmission (*used to keep multimedia messages synchronized*). Adaptec has FireWire products that allow up to 63 devices to be connected to a single PCI interface card, and also has good general FireWire information at http://www.adaptec.com/serialio/

Although FireWire will not be the highest bandwidth network available, the consumer-level market (*which should drive prices very low*) and low latency support might make this one of the best Linux PC cluster message-passing network technologies within the next year or so.

Design Considerations:

Linux support: No

Maximum bandwidth: 196.608 Mb/s (soon, 393.216 Mb/s)

Minimum latency: Data not available

Available as: multiple-vendor hardware

Interface port/bus used: PCI

Network structure: random without cycles (self-configuring)

Cost per machine connected: $600

HiPPI And Serial HiPPI

HiPPI (*High Performance Parallel Interface*) was originally intended to provide very high bandwidth for transfer of huge data sets between a supercomputer and another machine (*a supercomputer, frame buffer, disk array, etc.),* and has become the dominant standard for supercomputers. Although it is an oxymoron, Serial HiPPI is also becoming popular, typically using a fibre optic cable instead of the 32-bit wide standard (parallel) HiPPI cables. Over the past few years, HiPPI crossbar switches have become common and prices have dropped sharply; unfortunately, serial HiPPI is still quite expensive, and that is what PCI bus interface cards generally support. Further to this, Linux doesn't yet support HiPPI. A good overview of HiPPI is maintained by CERN at http://www.cern.ch/HSI/hippi/; they also maintain a rather long list of HiPPI vendors at http://www.cern.ch/HSI/hippi/procintf/manufact.htm

Design Considerations:

Linux support: no

Maximum bandwidth: 1,600 Mb/s (serial is 1,200 Mb/s)

Minimum latency: Data not available

Available as: multiple-vendor hardware

Interface port/bus used: EISA, PCI

Network structure: switched hubs

Cost per machine connected: $3,500 (serial is $4,500)

IrDA (Infrared Data Association)

IrDA (*Infrared Data Association, http://www.irda.org/*) is the infrared device on the side of many laptop PCs. It is inherently difficult to connect more than two machines using this interface, so it is highly unlikely to be used for clustering. Don Becker did some preliminary work with IrDA.

Design Considerations:

Linux support: no[1]

Maximum bandwidth: 1.15 Mb/s and 4 Mb/s

Minimum latency: Data not available

Available as: multiple-vendor hardware

Interface port/bus used: IrDA

Network structure: None required

Cost per machine connected: $0

SCI

The goal of SCI (*Scalable Coherent Interconnect, ANSI/IEEE 1596-1992*) is to essentially provide a high performance mechanism that can support coherent shared memory access across large numbers of machines, as well various types of block message transfers. It is considered that the designed bandwidth and latency of SCI are both far in advance in comparison to most other network technologies. The drawback is that SCI is not widely available as cost-effective production units, and there is not any Linux support.

SCI is used primarily in various proprietary designs for logically shared physically distributed memory machines, such as the HP/Convex Exemplar SPP and the Sequent NUMA-Q 2000 (see http://www.sequent.com/). However, SCI is available as a PCI interface card and 4-way switches (up to 16 machines can be connected by cascading four 4-way switches) from Dolphin, http://www.dolphinics.com/, as their CluStar product line. A good set of links overviewing SCI is maintained by CERN at http://www.cern.ch/HSI/sci/sci.html

Design Considerations:

Linux support: no

Maximum bandwidth: 4,000 Mb/s

Minimum latency: 2.7 microseconds

Available as: multiple-vendor hardware

Interface port/bus used: PCI, proprietary

Network structure: Data not available

Cost per machine connected: > $1,000

SCSI

SCSI (*Small Computer Systems Interconnect*) is essentially an I/O bus that is used for disk drives, CD ROMS, image scanners, and other peripherals. There are three separate standards SCSI-1, SCSI-2, and SCSI-3; Fast and Ultra speeds; and data path widths of 8, 16, or 32 bits *(with FireWire compatibility also mentioned in SCSI-3*). It is quite a complicated set of standards, however SCSI is somewhat faster than EIDE and can handle more devices more efficiently.

What many people do not realize is that it is fairly simple for two computers to share a single SCSI bus. This type of configuration is very useful for sharing disk drives between machines and implementing fail-over - having one machine take over database requests when the other machine fails. Currently, this is the only mechanism supported by Microsoft's PC cluster product, Windows 2000 Cluster Server (formerly known as WolfPack). However, the inability to scale to larger systems renders shared SCSI limited for parallel processing in general.

Design Considerations:

Linux support: kernel drivers

Maximum bandwidth: 5 Mb/s to over 20 Mb/s

Minimum latency: Data not available

Available as: multiple-vendor hardware

Interface port/bus used: PCI, EISA, ISA card

Network structure: inter-machine bus sharing SCSI devices

Cost per machine connected: Data not available

ServerNet

ServerNet is the high-performance network hardware from Compaq (*technology obtained from Tandem*). Especially in the online transaction processing (OLTP) world, Compaq is well known as a leading producer of high-reliability systems, so it is not surprising that their network claims not just high performance, but also "high data integrity and reliability". Another interesting aspect of ServerNet is that it claims to be able to transfer data from any device directly to any device; not just between processors, but also disk drives, etc., in a one-sided style similar to that suggested by the MPI remote memory access mechanisms. While a single vendor only offers ServerNet, that vendor is powerful enough to potentially establish ServerNet as a major standard.

Design Considerations:

Linux support: no

Maximum bandwidth: 400 Mb/s

Minimum latency: 3 microseconds

Available as: single-vendor hardware

Interface port/bus used: PCI

Network structure: hexagonal tree/tetrahedral lattice of hubs

Cost per machine connected: Data not available

SHRIMP

The SHRIMP project, http://www.CS.Princeton.EDU/shrimp/, at the Princeton University Computer Science Department is building a parallel computer using PCs running Linux as the processing elements. The first SHRIMP (*Scalable, High-Performance, Really Inexpensive Multi-Processor*) was a simple two-processor prototype using a dual-ported RAM on a custom EISA card interface. There is now a prototype that will scale to larger configurations using a custom interface card to connect to a "hub" that is essentially the same mesh routing network used in the Intel Paragon (see http://www.ssd.intel.com/paragon.html). Considerable effort has gone into developing low-overhead "virtual memory mapped communication" hardware and support software.

Design Considerations:

Linux support: user-level memory mapped interface

Maximum bandwidth: 180 Mb/s

Minimum latency: 5 microseconds

Available as: research prototype

Interface port/bus used: EISA

Network structure: mesh backplane (as in Intel Paragon)

Cost per machine connected: Data not available

TTL_PAPERS

The PAPERS (*Purdue's Adapter for Parallel Execution and Rapid Synchronization*) project, http://garage.ecn.purdue.edu/~papers/, at the Purdue University School of Electrical and Computer Engineering is building scalable, low-latency, aggregate function communication hardware and software that allows a parallel supercomputer to be built using unmodified PCs/workstations as nodes.

There have been over a dozen different types of PAPERS hardware built that connect to PCs/workstations via the SPP (Standard Parallel Port), roughly following two development lines. The versions called "PAPERS" target higher performance, using whatever technologies are appropriate; current work uses FPGAs, and high bandwidth PCI bus interface designs are also under development. In contrast, the versions called "TTL_PAPERS" are designed to be easily reproduced outside Purdue, and are remarkably simple public domain designs that can be built using ordinary TTL logic. One such design is produced commercially, http://chelsea.ios.com/~hgdietz/sbm4.html

Unlike the custom hardware designs from other universities, TTL_PAPERS clusters have been assembled at many universities from the USA to South Korea. Bandwidth is severely limited by the SPP connections, but PAPERS implements very low latency aggregate function communications; even the fastest message-oriented systems cannot provide comparable performance on those aggregate functions. Thus, PAPERS is particularly good for synchronizing the displays of a video wall, scheduling accesses to a high-bandwidth network, evaluating global fitness in genetic searches, etc. Although PAPERS clusters have been built using IBM PowerPC AIX, DEC Alpha OSF/1, and HP PA-RISC HP-UX machines, Linux-based PCs are the platforms best supported.

User programs using TTL_PAPERS AFAPI directly access the SPP hardware port registers under Linux, without an OS call for each access. To do this, AFAPI first gets port permission using either iopl() or ioperm(). The problem with these calls is that both require the user program to be privileged, yielding a potential security hole. The solution is an optional kernel patch, http://garage.ecn.purdue.edu/~papers/giveioperm.html that allows a privileged process to control port permission for any process.

Design Considerations:

Linux support: AFAPI library

Maximum bandwidth: 1.6 Mb/s

Minimum latency: 3 microseconds

Available as: public-domain design, single-vendor hardware

Interface port/bus used: SPP

Network structure: tree of hubs

Cost per machine connected: $100

WAPERS

WAPERS (*Wired-AND Adapter for Parallel Execution and Rapid Synchronization*) is a spin-off of the PAPERS project, http://garage.ecn.purdue.edu/~papers/, at the Purdue University School of Electrical and Computer Engineering. If implemented properly, the SPP has four bits of open-collector output that can be wired together across machines to implement a 4-bit wide wired AND. This wired-AND is electrically touchy, and the maximum number of machines that can be connected in this way critically depends on the analog properties of the ports (*maximum sink current and pull-up resistor value*); typically, up to 7 or 8 machines can be networked by WAPERS. Although cost and latency are very low, so is bandwidth; WAPERS is much better as a second network for aggregate operations than as the only network in a cluster. As with TTL_PAPERS, to improve system security, there is a minor kernel patch recommended, but not required: http://garage.ecn.purdue.edu/~papers/giveioperm.html

Design Considerations:

Linux support: AFAPI library

Maximum bandwidth: 0.4 Mb/s

Minimum latency: 3 microseconds

Available as: public-domain design

Interface port/bus used: SPP

Network structure: wiring pattern between 2-64 machines

Cost per machine connected: $5

8.2. Appendix B – Channel Bonding

```
/* Mode: C;
 * ifenslave.c: Configure network interfaces for parallel routing.
 *
 *      This program controls the Linux implementation of running multiple
 *      network interfaces in parallel.
 *
 * Usage:      ifenslave [-v] master-interface slave-interface [metric <N>]
 *
 * Author:     Donald Becker <becker@cesdis.gsfc.nasa.gov>
 *             Copyright 1994-1996 Donald Becker
 *
 *             This program is free software; you can redistribute it
 *             and/or modify it under the terms of the GNU General Public
 *             License as published by the Free Software Foundation.
 *
 *      The author may be reached as becker@CESDIS.gsfc.nasa.gov, or C/O
 *      Center of Excellence in Space Data and Information Sciences
 *         Code 930.5, Goddard Space Flight Center, Greenbelt MD 20771
 */

static char *version =
"ifenslave.c:v0.07 9/9/97  Donald Becker (becker@cesdis.gsfc.nasa.gov)\n";
static const char *usage_msg =
"Usage: ifenslave [-afrvV] <master-interface> <slave-interface> [metric <N>]\n";

#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <errno.h>
#include <fcntl.h>
#include <getopt.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <linux/if.h>
#include <linux/if_arp.h>
#include <linux/if_ether.h>

struct option longopts[] = {
 /* { name  has_arg  *flag  val } */
    {"all-interfaces", 0, 0, 'a'},    /* Show all interfaces. */
    {"force",       0, 0, 'f'},              /* Force the operation. */
    {"help",          0, 0, '?'},            /* Give help */
    {"receive-slave", 0, 0, 'r'},     /* Make a receive-only slave.  */
    {"verbose",       0, 0, 'v'},            /* Report each action taken.  */
    {"version",       0, 0, 'V'},            /* Emit version information.  */
    { 0, 0, 0, 0 }
};

/* Command-line flags. */
unsigned int
opt_a = 0,                                   /* Show-all-interfaces flag. */
opt_f = 0,                                   /* Force the operation. */
opt_r = 0,                                   /* Set up a Rx-only slave. */
verbose = 0,                                 /* Verbose flag. */
opt_version;
int skfd = -1;                               /* AF_INET socket for ioctl() calls.  */

static void if_print(char *ifname);

int
main(int argc, char **argv)
{
       struct ifreq  ifr2, if_hwaddr, if_ipaddr, if_metric, if_mtu, if_dstaddr;
       struct ifreq  if_netmask, if_brdaddr, if_flags;
       int goterr = 0;
       int c, errflag = 0;
       char **spp, *master_ifname, *slave_ifname;

       while ((c = getopt_long(argc, argv, "afrvV?", longopts, 0)) != EOF)
              switch (c) {
              case 'a': opt_a++; break;
              case 'f': opt_f++; break;
              case 'r': opt_r++; break;
```

```
            case 'v': verbose++;          break;
            case 'V': opt_version++;      break;
            case '?': errflag++;
            }
        if (errflag) {
            fprintf(stderr, usage_msg);
            return 2;
        }

        if (verbose || opt_version)
            printf(version);

        /* Open a basic socket. */
        if ((skfd = socket(AF_INET, SOCK_DGRAM,0)) < 0) {
            perror("socket");
            exit(-1);
        }

#ifdef notdef
        /* Find options scattered throughout the command line.
           I should change this to use getopt() sometime. */
        argc--; argv++;
        while (*argv[0] == '-') {
            char *argp = *argv++;
            argc--;
            while (*++argp) {
                switch (*argp) {
                default:
                    fprintf(stderr, "Unrecognized option '%c'.\n%s",
                            argp[0], usage_msg);
                    return 2;
                }
            }
        }
#endif

        if (verbose)
            fprintf(stderr, "DEBUG: argc=%d, optind=%d and argv[optind] is %s.\n",
                    argc, optind, argv[optind]);

        /* No remaining args means show all interfaces. */
        if (optind == argc) {
            if_print((char *)NULL);
            (void) close(skfd);
            exit(0);
        }

        /* Copy the interface name. */
        spp = argv + optind;
        master_ifname = *spp++;
        slave_ifname = *spp++;

        /* A single args means show the configuration for this interface. */
        if (slave_ifname == NULL) {
            if_print(master_ifname);
            (void) close(skfd);
            exit(0);
        }

        /* Get the vitals from the master interface. */
        strncpy(if_hwaddr.ifr_name, master_ifname, IFNAMSIZ);
        if (ioctl(skfd, SIOCGIFHWADDR, &if_hwaddr) < 0) {
            fprintf(stderr, "SIOCGIFHWADDR on %s failed: %s\n", master_ifname,
                    strerror(errno));
            return 1;
        } else {            /* Gotta convert from 'char' to unsigned for printf().  */
            unsigned char *hwaddr = (unsigned char *)if_hwaddr.ifr_hwaddr.sa_data;

            /* The family '1' is ARPHRD_ETHER for ethernet. */
            if (if_hwaddr.ifr_hwaddr.sa_family != 1 && !opt_f) {
                fprintf(stderr, "The specified master interface '%s' is not"
                        " ethernet-like.\n  This program is designed to work"
                        " with ethernet-like network interfaces.\n"
                        " Use the '-f' option to force the operation.\n",
                        master_ifname);

                return 1;
            }

            if (verbose)
                printf("The hardware address (SIOCGIFHWADDR) of %s is type %d  "
                       "%2.2x:%2.2x:%2.2x:%2.2x:%2.2x:%2.2x.\n", master_ifname,
                       if_hwaddr.ifr_hwaddr.sa_family, hwaddr[0], hwaddr[1],
```

```
                                hwaddr[2], hwaddr[3], hwaddr[4], hwaddr[5]);
        }

        {
                struct ifreq *ifra[6] = { &if_ipaddr, &if_mtu, &if_dstaddr,
                                                                &if_brdaddr, &if_netmask,
&if_flags };
                const char *req_name[6] = {
                        "IP address", "MTU", "destination address",
                        "broadcast address", "netmask", "status flags", };
                const int ioctl_req_type[6] = {
                        SIOCGIFADDR, SIOCGIFMTU, SIOCGIFDSTADDR,
                        SIOCGIFBRDADDR, SIOCGIFNETMASK, SIOCGIFFLAGS, };
                int i;

                for (i = 0; i < 6; i++) {
                        strncpy(ifra[i]->ifr_name, master_ifname, IFNAMSIZ);
                        if (ioctl(skfd, ioctl_req_type[i], ifra[i]) < 0) {
                                fprintf(stderr,
                                                "Something broke getting the master's %s: %s.\n",
                                                req_name[i], strerror(errno));
                        }
                }
        }

        do {
                strncpy(ifr2.ifr_name, slave_ifname, IFNAMSIZ);
                if (ioctl(skfd, SIOCGIFFLAGS, &ifr2) < 0) {
                        int saved_errno = errno;
                        fprintf(stderr, "SIOCGIFFLAGS on %s failed: %s\n", slave_ifname,
                                        strerror(saved_errno));
                        return 1;
                }
                if (ifr2.ifr_flags & IFF_UP) {
                        printf("The interface %s is up, shutting it down it to enslave it.\n",
                                   slave_ifname);
                        ifr2.ifr_flags &= ~IFF_UP;
                        if (ioctl(skfd, SIOCSIFFLAGS, &ifr2) < 0) {
                                int saved_errno = errno;
                                fprintf(stderr, "Shutting down interface %s failed: %s\n",
                                                slave_ifname, strerror(saved_errno));
                        }
                }

                strncpy(if_hwaddr.ifr_name, slave_ifname, IFNAMSIZ);
                if (ioctl(skfd, SIOCSIFHWADDR, &if_hwaddr) < 0) {
                        int saved_errno = errno;
                        fprintf(stderr, "SIOCSIFHWADDR on %s failed: %s\n", if_hwaddr.ifr_name,
                                        strerror(saved_errno));
                        if (saved_errno == EBUSY)
                                fprintf(stderr, "  The slave device %s is busy: it must be"
                                                " idle before running this command.\n",
slave_ifname);
                        else if (saved_errno == EOPNOTSUPP)
                                fprintf(stderr, "  The slave device you specified does not
support"
                                                " setting the MAC address.\n  Your kernel likely
does not"
                                                " support slave devices.\n");
                        else if (saved_errno == EINVAL)
                                fprintf(stderr, "  The slave device's address type does not
match"
                                                " the master's address type.\n");
                        return 1;
                } else {
                        if (verbose) {
                                unsigned char *hwaddr = if_hwaddr.ifr_hwaddr.sa_data;
                                printf("Set the slave's hardware address to "
                                           "%2.2x:%2.2x:%2.2x:%2.2x:%2.2x:%2.2x.\n", hwaddr[0],
                                           hwaddr[1], hwaddr[2], hwaddr[3], hwaddr[4],
hwaddr[5]);
                        }
                }

                if (*spp  &&  !strcmp(*spp, "metric")) {
                        if (*++spp == NULL) {
                                fprintf(stderr, usage_msg);
                                exit(2);
                        }
                        if_metric.ifr_metric = atoi(*spp);
                        if (ioctl(skfd, SIOCSIFMETRIC, &if_metric) < 0) {
                                fprintf(stderr, "SIOCSIFMETRIC: %s\n", strerror(errno));
                                goterr = 1;
```

```
                }
                spp++;
        }

        if (strncpy(if_ipaddr.ifr_name, slave_ifname, IFNAMSIZ) <= 0
                || ioctl(skfd, SIOCSIFADDR, &if_ipaddr) < 0) {
                fprintf(stderr,
                                "Something broke setting the slave's address: %s.\n",
                                strerror(errno));
        } else {
                if (verbose) {
                        unsigned char *ipaddr = if_ipaddr.ifr_addr.sa_data;
                        printf("Set the slave's IP address to %d.%d.%d.%d.\n",
                                   ipaddr[0], ipaddr[1], ipaddr[2], ipaddr[3]);
                }
        }

        if (strncpy(if_mtu.ifr_name, slave_ifname, IFNAMSIZ) <= 0
                || ioctl(skfd, SIOCSIFMTU, &if_mtu) < 0) {
                fprintf(stderr, "Something broke setting the slave MTU: %s.\n",
                                strerror(errno));
        } else {
                if (verbose)
                        printf("Set the slave's MTU to %d.\n", if_mtu.ifr_mtu);
        }

        if (strncpy(if_dstaddr.ifr_name, slave_ifname, IFNAMSIZ) <= 0
                || ioctl(skfd, SIOCSIFDSTADDR, &if_dstaddr) < 0) {
                fprintf(stderr, "Error setting the slave with SIOCSIFDSTADDR: %s.\n",
                                strerror(errno));
        } else {
                if (verbose) {
                        unsigned char *ipaddr = if_dstaddr.ifr_dstaddr.sa_data;
                        printf("Set the slave's destination address to %d.%d.%d.%d.\n",
                                   ipaddr[0], ipaddr[1], ipaddr[2], ipaddr[3]);
                }
        }

        if (strncpy(if_brdaddr.ifr_name, slave_ifname, IFNAMSIZ) <= 0
                || ioctl(skfd, SIOCSIFBRDADDR, &if_brdaddr) < 0) {
                fprintf(stderr,
                                "Something broke setting the slave broadcast address:
%s.\n",
                                strerror(errno));
        } else {
                if (verbose) {
                        unsigned char *ipaddr = if_brdaddr.ifr_broadaddr.sa_data;
                        printf("Set the slave's broadcast address to %d.%d.%d.%d.\n",
                                   ipaddr[0], ipaddr[1], ipaddr[2], ipaddr[3]);
                }
        }

        if (strncpy(if_netmask.ifr_name, slave_ifname, IFNAMSIZ) <= 0
                || ioctl(skfd, SIOCSIFNETMASK, &if_netmask) < 0) {
                fprintf(stderr,
                                "Something broke setting the slave netmask: %s.\n",
                                strerror(errno));
        } else {
                if (verbose) {
                        unsigned char *ipaddr = if_netmask.ifr_netmask.sa_data;
                        printf("Set the slave's netmask to %d.%d.%d.%d.\n",
                                   ipaddr[0], ipaddr[1], ipaddr[2], ipaddr[3]);
                }
        }

        if ((if_flags.ifr_flags &= ~(IFF_SLAVE | IFF_MASTER)) == 0
                || strncpy(if_flags.ifr_name, slave_ifname, IFNAMSIZ) <= 0
                || ioctl(skfd, SIOCSIFFLAGS, &if_flags) < 0) {
                fprintf(stderr,
                                "Something broke setting the slave flags: %s.\n",
                                strerror(errno));
        } else {
                if (verbose)
                        printf("Set the slave's flags %4.4x.\n", if_flags.ifr_flags);
        }

        /* Do the real thing: set the second interface as a slave. */
        if ( ! opt_r) {
                strncpy(if_flags.ifr_name, master_ifname, IFNAMSIZ);
                strncpy(if_flags.ifr_slave, slave_ifname, IFNAMSIZ);
                if (ioctl(skfd, SIOCSIFSLAVE, &if_flags) < 0) {
                        fprintf(stderr,       "SIOCSIFSLAVE: %s.\n", strerror(errno));
                }
```

```
                }
        } while ( (slave_ifname = *spp++) != NULL);

        /* Close the socket. */
        (void) close(skfd);

        return(goterr);
}

static short mif_flags;

/* Get the inteface configuration from the kernel. */
static int if_getconfig(char *ifname)
{
        struct ifreq ifr;
        int metric, mtu;                      /* Parameters of the master interface. */
        struct sockaddr dstaddr, broadaddr, netmask;

        strcpy(ifr.ifr_name, ifname);
        if (ioctl(skfd, SIOCGIFFLAGS, &ifr) < 0)
                return -1;
        mif_flags = ifr.ifr_flags;
        printf("The result of SIOCGIFFLAGS on %s is %x.\n",
                   ifname, ifr.ifr_flags);

        strcpy(ifr.ifr_name, ifname);
        if (ioctl(skfd, SIOCGIFADDR, &ifr) < 0)
                return -1;
        printf("The result of SIOCGIFADDR is %2.2x.%2.2x.%2.2x.%2.2x.\n",
                   ifr.ifr_addr.sa_data[0], ifr.ifr_addr.sa_data[1],
                   ifr.ifr_addr.sa_data[2], ifr.ifr_addr.sa_data[3]);

        strcpy(ifr.ifr_name, ifname);
        if (ioctl(skfd, SIOCGIFHWADDR, &ifr) < 0)
                return -1;

        {              /* Gotta convert from 'char' to unsigned for printf().  */
                unsigned char *hwaddr = (unsigned char *)ifr.ifr_hwaddr.sa_data;
                printf("The result of SIOCGIFHWADDR is type %d  "
                           "%2.2x:%2.2x:%2.2x:%2.2x:%2.2x:%2.2x.\n",
                           ifr.ifr_hwaddr.sa_family, hwaddr[0], hwaddr[1],
                           hwaddr[2], hwaddr[3], hwaddr[4], hwaddr[5]);
        }

        strcpy(ifr.ifr_name, ifname);
        if (ioctl(skfd, SIOCGIFMETRIC, &ifr) < 0) {
                metric = 0;
        } else
                metric = ifr.ifr_metric;

        strcpy(ifr.ifr_name, ifname);
        if (ioctl(skfd, SIOCGIFMTU, &ifr) < 0)
                mtu = 0;
        else
                mtu = ifr.ifr_mtu;

        strcpy(ifr.ifr_name, ifname);
        if (ioctl(skfd, SIOCGIFDSTADDR, &ifr) < 0) {
                memset(&dstaddr, 0, sizeof(struct sockaddr));
        } else
                dstaddr = ifr.ifr_dstaddr;

        strcpy(ifr.ifr_name, ifname);
        if (ioctl(skfd, SIOCGIFBRDADDR, &ifr) < 0) {
                memset(&broadaddr, 0, sizeof(struct sockaddr));
        } else
                broadaddr = ifr.ifr_broadaddr;

        strcpy(ifr.ifr_name, ifname);
        if (ioctl(skfd, SIOCGIFNETMASK, &ifr) < 0) {
                memset(&netmask, 0, sizeof(struct sockaddr));
        } else
                netmask = ifr.ifr_netmask;

        return(0);
}

static void if_print(char *ifname)
{
        char buff[1024];
        struct ifconf ifc;
        struct ifreq *ifr;
        int i;
```

```
        if (ifname == (char *)NULL) {
                ifc.ifc_len = sizeof(buff);
                ifc.ifc_buf = buff;
                if (ioctl(skfd, SIOCGIFCONF, &ifc) < 0) {
                        fprintf(stderr, "SIOCGIFCONF: %s\n", strerror(errno));
                        return;
                }

                ifr = ifc.ifc_req;
                for (i = ifc.ifc_len / sizeof(struct ifreq); --i >= 0; ifr++) {
                        if (if_getconfig(ifr->ifr_name) < 0) {
                                fprintf(stderr, "%s: unknown interface.\n",
                                                ifr->ifr_name);
                                continue;
                        }

                        if (((mif_flags & IFF_UP) == 0) && !opt_a) continue;
                        /*ife_print(&ife);*/
                }
        } else {
                if (if_getconfig(ifname) < 0)
                        fprintf(stderr, "%s: unknown interface.\n", ifname);
        }
}

/*
 * Local variables:
 *  version-control: t
 *  kept-new-versions: 5
 *  c-indent-level: 4
 *  c-basic-offset: 4
 *  tab-width: 4
 *  compile-command: "gcc -Wall -Wstrict-prototypes -O ifenslave.c -o ifenslave"
 * End:
 */
```

8.3. Appendix C – MPI Implementations

The following MPI Implementations were considered for use [19].

8.3.1. LAM

LAM/MPI			
Supplier:	University of Notre Dame		
Version:	6.5.1		
Last update:	05/06/2001 at 08:58:22		
URL:	http://www.lam-mpi.org/		
E-mail:	lam@lam-mpi.org		
Platforms:	Just about all POSIX platforms.		
Source code available:	Yes		
Commerical:	No		
Free:	Yes		
	Topic	**Availability**	**Notes**
MPI-1:	All (excluding MPI_CANCEL)	Supported	
	MPI_CANCEL	Some Support	Canceling sends is not supported
	Topic	**Availability**	**Notes**
MPI-2: Miscellany	mpiexec	No Support	
	MPI_INIT(NULL, NULL)	Supported	
	TYPE_INDEXED_BLOCK	No Support	
	STATUS_IGNORE	Supported	
	Keyval error class	Supported	
	Committed datatype	Supported	
	User functions @ termination	No Support	
	MPI_FINALIZED	Supported	
	MPI_Info	Supported	
	MPI_MALLOC	Supported	
	Language interoperability	Supported	
	Error handlers	Supported	
	Datatype manipulations	Supported	
	New Datatypes	Some Support	Have MPI_WCHAR and MPI_UNSIGNED_LONG_LONG; missing MPI_SIGNED_CHAR
	Canonical [UN]PACK	No Support	
	Topic	**Availability**	**Notes**
MPI-2: Process Creation and Management	Process manager	Supported	
	Establishing communication	Supported	
	Other	Supported	
	Topic	**Availability**	**Notes**
MPI-2: One-sided Communication	Initialization	Supported	
	Communication	Supported	
	Synchronization	Some Support	Missing MPI_Win_test, MPI_Win_lock, and MPI_Win_unlock
	Error handling	Some Support	Have MPI_ERR_WIN and MPI_ERR_BASE
	Topic	**Availability**	**Notes**
MPI-2: Extended Collective Operation	Intercommunicator constructors	No Support	
	Extended collectives	No Support	
	Topic	**Availability**	**Notes**
MPI-2: External Interfaces	Generalized requests	No Support	
	Status information	No Support	
	Naming objects	Supported	
	Error classes	No Support	
	Decoding datatypes	Supported	
	Threads	Some Support	MPI_INIT_THREAD only supports MPI_THREAD_SINGLE
	Attribute caching	Supported	
	Duplicating datatypes	Supported	
	Topic	**Availability**	**Notes**
MPI-2: I/O	File manipulation	Supported	
	File views	Supported	
	Data access	Supported	
	File interoperability	Some Support	
	Consistency	Supported	
	Error handling	No Support	
	Error classes	No Support	
	Topic	**Availability**	**Notes**
MPI-2: Language	C++ for MPI-1.2	Supported	
	C++ for MPI-2	No Support	
	Fortran 90 module	No Support	
	Topic	**Availability**	**Notes**
IMPI	IMPI support	Some Support	Point to point functionality supported

Table 8-1 – LAM MPI 2 Implementation Details

8.3.2. MPICH

MPICH			
Supplier:	Argonne National Laboratory		
Version:	1.2.1		
Last update:	11/13/2000 at 08:17:26		
URL:	http://www-unix.mcs.anl.gov/mpi/mpich/		
E-mail:	mpi-maint@mcs.anl.gov		
Platforms:	MPP's, IBM SP, Intel Paragon, SGI Onyx, Challenge and Power Challenge, Convex (HP)		
Source code available:	Yes		
Commerical:	No		
Free:	Yes		
	Topic	**Availability**	**Notes**
MPI-1:	All (excluding MPI_CANCEL)	Supported	
	MPI_CANCEL	Supported	
	Topic	**Availability**	**Notes**
MPI-2: Miscellany	mpiexec	No Support	
	MPI_INIT(NULL, NULL)	No Support	
	TYPE_INDEXED_BLOCK	Supported	
	STATUS_IGNORE	No Support	
	Keyval error class	Supported	
	Committed datatype	Supported	
	User functions @ termination	No Support	
	MPI_FINALIZED	Supported	
	MPI_Info	Supported	
	MPI_MALLOC	No Support	
	Language interoperability	Supported	
	Error handlers	No Support	
	Datatype manipulations	Some Support	
	New Datatypes	Some Support	
	Canonical [UN]PACK	No Support	
	Topic	**Availability**	**Notes**
MPI-2: Process Creation and Management	Process manager	No Support	
	Establishing communication	No Support	
	Other	No Support	
	Topic	**Availability**	**Notes**
MPI-2: One-sided Communication	Initialization	No Support	
	Communication	No Support	
	Synchronization	No Support	
	Error handling	No Support	
	Topic	**Availability**	**Notes**
MPI-2: Extended Collective Operation	Intercommunicator constructors	No Support	
	Extended collectives	No Support	
	Topic	**Availability**	**Notes**
MPI-2: External Interfaces	Generalized requests	No Support	
	Status information	Supported	
	Naming objects	Some Support	
	Error classes	No Support	
	Decoding datatypes	Supported	
	Threads	Some Support	
	Attribute caching	No Support	
	Duplicating datatypes	No Support	
	Topic	**Availability**	**Notes**
MPI-2: I/O	File manipulation	Supported	
	File views	Supported	
	Data access	Supported	
	File interoperability	Some Support	
	Consistency	Supported	
	Error handling	No Support	
	Error classes	No Support	
	Topic	**Availability**	**Notes**
MPI-2: Language	C++ for MPI-1.2	Supported	
	C++ for MPI-2	No Support	
	Fortran 90 module	Some Support	
	Topic	**Availability**	**Notes**
IMPI	IMPI support	No Support	

Table 8-2 – MPICH MPI 2 Implementation Details

8.4. Appendix D – POV-Ray

The following appendix contains some of the POV-Ray test script files or `.pov` files used on the test cluster as well as performance data extracted from the dataset on comparable clusters. [43.2]

8.4.1. POV-Ray Benchmark

Test Settings

For command line versions:
povray -i skyvase.pov +v1 -d +ft -x +a0.300 +r3 -q9 -w640 -h480 -mv2.0 +b1000 > results.txt
(*For Unix/Linux versions prepend with "time" and note the CPU time used*)

For GUI versions:
Go to menu: Render --> Edit settings/Render (or press Alt+C) and paste:
+v1 -d +ft -x +a0.300 +r3 -q9 -w640 -h480 -mv2.0 +b1000
into the "Command line options" box.
This will temporarily override your render settings.
If you're running a multi-tasking OS, try to give POV full attention.

skyvase.pov file

```
// Persistence Of Vision raytracer version 2.0 sample file.

// By Dan Farmer
//    Minneapolis, mn

//   skyvase.pov
//   Vase made with Hyperboloid and sphere {, sitting on a hexagonal
//   marble column.  Take note of the color and surface characteristics
//   of the gold band around the vase.  It seems to be a successful
//   combination for gold or brass.
//
// Contains a Disk_Y object which may have changed in shapes.dat

#include "shapes.inc"
#include "shapes2.inc"
#include "colors.inc"
#include "textures.inc"

#declare DMF_Hyperboloid = quadric {  /* Like Hyperboloid_Y, but more curvy */
   <1.0, -1.0,  1.0>,
   <0.0,  0.0,  0.0>,
   <0.0,  0.0,  0.0>,
   -0.5
}

camera {
   location <0.0, 28.0, -200.0>
   direction <0.0, 0.0, 2.0>
   up  <0.0, 1.0, 0.0>
   right <4/3, 0.0, 0.0>
   look_at <0.0, -12.0, 0.0>
}

/* Light behind viewer postion (pseudo-ambient light) */
light_source { <100.0, 500.0, -500.0> colour White }

union {
   union {
      intersection {
         plane { y, 0.7 }
```

```
            object { DMF_Hyperboloid scale <0.75, 1.25, 0.75> }
            object { DMF_Hyperboloid scale <0.70, 1.25, 0.70> inverse }
            plane { y, -1.0 inverse }
         }
         sphere { <0, 0, 0>, 1 scale <1.6, 0.75, 1.6 > translate <0, -1.15, 0> }

         scale <20, 25, 20>

         pigment {
            Bright_Blue_Sky
            turbulence 0.3
            quick_color Blue
            scale <8.0, 4.0, 4.0>
            rotate 15*z
         }
         finish {
            ambient 0.1
            diffuse 0.75
            phong 1
            phong_size 100
            reflection 0.35
         }
      }

      sphere {  /* Gold ridge around sphere portion of vase*/
         <0, 0, 0>, 1
         scale <1.6, 0.75, 1.6>
         translate -7*y
         scale <20.5, 4.0, 20.5>

         finish { Metal }
         pigment { OldGold }
      }

      bounded_by {
         object {
            Disk_Y
            translate -0.5*y  // Remove for new Disk_Y definition
            scale <34, 100, 34>
         }
      }
   }

   /* Stand for the vase */
   object { Hexagon
      rotate -90.0*z             /* Stand it on end (vertical)*/
      rotate -45*y               /* Turn it to a pleasing angle */
      scale <40, 25, 40>
      translate -70*y

      pigment {
         Sapphire_Agate
         quick_color Red
         scale 10.0
      }
      finish {
         ambient 0.2
         diffuse 0.75
         reflection 0.85
      }
   }

   union {
      plane { z, 50  rotate -45*y }
      plane { z, 50  rotate +45*y }

      pigment { DimGray }
      finish {
         ambient 0.2
         diffuse 0.75
         reflection 0.5
      }
   }
```

 Copyright © 2003

8.4.2. Alternate Bench Mark POV Files

poolballs.pov

```
/*
DEMONSTRATION FILE
POOLBALL PLUGIN (version 1.0)

(C) Nathan G B O'Brien  1997
no13@ozemail.com.au
http://www.ozemail.com.au/~no13
*/


// ==== Standard POV-Ray Includes ====
#include "colors.inc"  // Standard Color definitions
#include "textures.inc"       // Standard Texture definitions

camera {
    location <0,20,-30>
    look_at  <0,10,0>
}

light_source {<10,50,-20> color White}

// Step one.
// Declare variables that will be the same for all poolballs

// POOLBALL INFORMATION
#declare Poolball_radius = 10
#declare Poolball_font = "crystal.ttf"
#declare Poolball_insert = 1
#declare Poolball_finish = finish {phong 1 reflection 0.25}

// FONT ADJUSTMENTS
#declare Small_numberx = 0
#declare Small_numbery = 0.1
#declare Large_numberx = 0.05
#declare Large_numbery = 0.1

// TABLE SURFACE
box {<500,0,500><-500,-10,-500>
    pigment{Green}
    finish {phong 1}
    normal {bumps .2 scale .5}
}

// EIGHT BALL
#declare Poolball_texture = texture{pigment{red 0 green 0 blue 0}}
#declare Poolball_number = 8
#declare Poolball_band = 0
#declare Pool_rotatex = 0
#declare Pool_rotatey = 0
#declare Pool_rotatez = 0
#include "13_Pball.inc"
object {Poolball translate <0,0,10>}

// THIRTEEN BALL
#declare Poolball_texture = texture{pigment{red 1 green 1 blue 0}}
#declare Poolball_number = 13
#declare Poolball_band = 1
#declare Pool_rotatex = 10
#declare Pool_rotatey = -20
#declare Pool_rotatez = 10
#include "13_Pball.inc"
object {Poolball translate <-15,0,0>}

// FOURTEEN BALL
#declare Poolball_texture = texture{pigment{red 0 green 0 blue 1}}
```

```
#declare Poolball_number = 14
#declare Poolball_band = 1
#declare Pool_rotatex = 20
#declare Pool_rotatey = 35
#declare Pool_rotatez = -10
#include "13_Pball.inc"
object {Poolball translate <15,0,15>}

// SIX BALL
#declare Poolball_texture = texture{pigment{red 1 green 0 blue 0}}
#declare Poolball_number = 6
#declare Poolball_band = 0
#declare Pool_rotatex = 0
#declare Pool_rotatey = 60
#declare Pool_rotatez = 0
#include "13_Pball.inc"
object {Poolball translate <15,0,-10>}
```

Poolballs.inc

```
/*
DEMONSTRATION FILE
POOLBALL PLUGIN (version 1.0)

(C) Nathan G B O'Brien  1997
no13@ozemail.com.au
http://www.ozemail.com.au/~no13
*/


// ==== Standard POV-Ray Includes ====
#include "colors.inc"  // Standard Color definitions
#include "textures.inc"       // Standard Texture definitions

camera {
    location <0,20,-30>
    look_at  <0,10,0>
}

light_source {<10,50,-20> color White}

// Step one.
// Declare variables that will be the same for all poolballs

// POOLBALL INFORMATION
#declare Poolball_radius = 10
#declare Poolball_font = "crystal.ttf"
#declare Poolball_insert = 1
#declare Poolball_finish = finish {phong 1 reflection 0.25}

// FONT ADJUSTMENTS
#declare Small_numberx = 0
#declare Small_numbery = 0.1
#declare Large_numberx = 0.05
#declare Large_numbery = 0.1

// TABLE SURFACE
box {<500,0,500><-500,-10,-500>
    pigment{Green}
    finish {phong 1}
    normal {bumps .2 scale .5}
}

// EIGHT BALL
#declare Poolball_texture = texture{pigment{red 0 green 0 blue 0}}
#declare Poolball_number = 8
#declare Poolball_band = 0
#declare Pool_rotatex = 0
#declare Pool_rotatey = 0
#declare Pool_rotatez = 0
#include "13_Pball.inc"
object {Poolball translate <0,0,10>}
```

```
// THIRTEEN BALL
#declare Poolball_texture = texture{pigment{red 1 green 1 blue 0}}
#declare Poolball_number = 13
#declare Poolball_band = 1
#declare Pool_rotatex = 10
#declare Pool_rotatey = -20
#declare Pool_rotatez = 10
#include "13_Pball.inc"
object {Poolball translate <-15,0,0>}

// FOURTEEN BALL
#declare Poolball_texture = texture{pigment{red 0 green 0 blue 1}}
#declare Poolball_number = 14
#declare Poolball_band = 1
#declare Pool_rotatex = 20
#declare Pool_rotatey = 35
#declare Pool_rotatez = -10
#include "13_Pball.inc"
object {Poolball translate <15,0,15>}

// SIX BALL
#declare Poolball_texture = texture{pigment{red 1 green 0 blue 0}}
#declare Poolball_number = 6
#declare Poolball_band = 0
#declare Pool_rotatex = 0
#declare Pool_rotatey = 60
#declare Pool_rotatez = 0
#include "13_Pball.inc"
object {Poolball translate <15,0,-10>}
```

tulips.pov

```
#declare TulipHeadTexture = texture
{
  pigment
  {
    bumps

    color_map {
      [0.0 colour rgb<0.9, 0.0, 0.0>]
      [1.0 colour rgb<0.8, 0.0, 0.2>]
    }

    scale <0.005, 0.02, 0.005>
  }

  normal
  {
    bumps 0.25
    scale <0.025, 1, 0.025>
  }

  finish
  {
    phong 0.5
    phong_size 25
  }
}

#declare TulipStemTexture = texture
{
  pigment
  {
    bumps

    color_map {
      [0.0 colour rgb<0.5, 0.5, 0.4>]
      [1.0 colour rgb<0.3, 0.4, 0.1>]
    }

    scale <0.005, 0.08, 0.005>
  }

  normal
  {
    bumps 0.25
    scale <0.025, 1, 0.025>
  }

  finish
  {
    ambient 0.0
  }
}

#declare GrassTexture = texture
{
  pigment
  {
    bumps

    color_map {
      [0.0 colour rgb<0.6, 0.6, 0.6>]
      [1.0 colour rgb<0.3, 0.7, 0.2>]
    }

    scale <0.005, 0.5, 0.005>
  }

  normal
  {
    bumps 0.25
```

```
    scale <0.03, 2, 0.03>
  }

  finish
  {
    ambient 0.0
    diffuse 0.5
  }
}

#declare WindmillBladesTexture = texture
{
  pigment
  {
    colour rgb<0.2, 0.2, 0.0>
  }
}

#declare WindmillBaseTexture = texture
{
  pigment
  {
    bumps

    color_map {
      [0.0 colour rgb<0.5, 0.5, 0.4>]
      [1.0 colour rgb<0.8, 0.8, 0.8>]
    }

    scale 2
  }
}


#declare PetalBlob = blob
{
  threshold 0.5

  sphere{<0.00, 0, 0> 1.0,  1}
  sphere{<0.03, 0, 0> 0.6, -1}

  bounded_by{box{<-0.55,-0.4,-0.4>, <-0.3, 0.4, 0.4>}}
}

#declare Petal = union
{
  difference
  {
    object
    {
      PetalBlob
      scale <1, 2, 1>
    }

    plane
    {
      <0, 1, 0>, 0
    }
  }

  difference
  {
    object
    {
      PetalBlob
      scale <1, 1, 1>
    }

    plane
    {
      <0, -1, 0>, 0
    }
  }
```

```
  rotate <0, 0, 10>
  translate <0.05, 0, 0>
}

#declare TulipHead = union
{
  #declare PetalLayerLoop = 0;
  #while (PetalLayerLoop < 4)
    #declare PetalScale = 1-PetalLayerLoop / 5;
    #declare PetalLoop = 0;
    #while (PetalLoop < 3)
      object
      {
        Petal
        rotate <0, PetalLoop * 120 + PetalLayerLoop * 59, 0>
        scale <PetalScale, 1, PetalScale>
      }
      #declare PetalLoop = PetalLoop + 1;
    #end
    #declare PetalLayerLoop = PetalLayerLoop + 1;
  #end
  texture
  {
    TulipHeadTexture
  }
}

#declare TulipStem = intersection
{
  torus
  {
    10, 0.06
    rotate <90, 0, 0>
    translate <10, 0, 0>
  }

  box
  {
    <-2, -5, -2>
    <2, 0, 2>
  }

  texture
  {
    TulipStemTexture
  }
}

#macro Tulip(HeadAngle)
  union
  {
    object {TulipStem}
    object {TulipHead rotate <0, HeadAngle, 0>}
//object{sphere {<0, 0, 0>, 0.5 texture{TulipHeadTexture}}}
  }
#end

#declare Grass = difference
{
  torus {
    0.8, 0.1
  }
  torus {
    0.8, 0.11
    translate <0.02, 0, 0>
  }
  plane {
    < 0, 0, -1>, 0
  }
  plane {
    <-1, 0,  0>, 0
  }
```

```
  rotate <90, 0, 0>
  translate <0.9, 0, 0>
  scale <1, 2, 1>

  texture
  {
    GrassTexture
  }

  bounded_by{box{<-0.08, 0, -0.1>, <0.13, 1.7, 0.1>} rotate <0, 0, -12>}
}


#declare WindmillBlades = union
{
  box {< 2.0, -2.5, 0>, < 16.0,   0.5, 0.2>}
  box {< 2.5,  2.0, 0>, < -0.5,  16.0, 0.2>}
  box {<-2.0,  2.5, 0>, <-16.0,  -0.5, 0.2>}
  box {<-2.5, -2.0, 0>, <  0.5, -16.0, 0.2>}

  box {<-8.0, -0.5, 0>, <8.0, 0.5, 0.2>}
  box {<-0.5, -8.0, 0>, <0.5, 8.0, 0.2>}

  cylinder {<0, 0, 0>, <0, 0, 2>, 1}

  texture
  {
    WindmillBladesTexture
  }
}

#declare WindmillBase = union
{
  cone
  {
    <0, 0, 0>, 8, <0, 20, 0>, 2
    texture
    {
      WindmillBaseTexture
    }
  }

  cylinder {<  0, -2,   0>, <  0,  0.0,   0>, 12.0}
  cylinder {<  0, -2,   0>, <  0, -8.0,   0>,  8.0}
  cylinder {< 11, -2,   0>, < 11,  3.5,   0>,  0.2}
  cylinder {<-11, -2,   0>, <-11,  3.5,   0>,  0.2}
  cylinder {<  0, -2, -11>, <  0,  3.0, -11>,  0.2}

  texture
  {
    WindmillBladesTexture
  }
}

#declare Windmill = union
{
  object
  {
    WindmillBase
  }
  object
  {
    WindmillBlades
    rotate < 0, 0, -15>
    rotate <15, 0,   0>
    translate <0, 18, -6>
    rotate <0, 20, 0>
  }
}

#declare Sky = sky_sphere
{
```

```
  pigment
  {
    bozo
    turbulence 0.65
    octaves 6
    omega 0.7
    lambda 2
    color_map {
      [0.0 color rgb<0.5, 0.5, 0.5>]
      [1.0 color rgb<0.5, 0.7, 1.0>]
    }
    rotate <0, 30, 0>
    rotate <73, 0, 0>
    scale <0.2, 0.1, 0.1>
  }

  pigment
  {
    bozo
    turbulence 0.65
    octaves 6
    omega 0.7
    lambda 2
    color_map {
      [0.0 color rgbt<1.0, 1.00, 1, 0>]
      [1.0 color rgbt<0.5, 0.66, 1, 0.2>]
    }
    rotate <0, 30, 0>
    rotate <70, 0, 0>
    scale <0.2, 0.1, 0.1>
  }
}

camera
{
  right  x
  up 4/3*y

  location <0, 0, -3.0>
  look_at <0, 0, 0>

  aperture 0.25
  blur_samples 256
  focal_point<0,0,0>
  confidence 0.9999
  variance 0
}

#declare  R1 = seed(123);
#declare LightLoop = 0;
#while (LightLoop < 100)
  light_source
  {
    <800-rand(R1)*1600, 400, 800-rand(R1)*1600>
    color rgb<0.037, 0.037, 0.037>
  }

  #declare LightLoop = LightLoop + 1;
#end

sky_sphere {Sky}

union
{

  plane
  {
    <0, 1, 0>, -4

    pigment {colour rgb <0, 0.5, 0>}
    finish {ambient 1.0 diffuse 0.0}
  }
```

```
  object {Tulip(65) rotate <0, 0, -25> rotate <0, -25, 0> translate <-0.7, -
0.1,  0.0>}
  object {Tulip(30) rotate <0, 0, -40> rotate <0, -10, 0> translate <-0.2, -
1.2,  0.0>}
  object {Tulip(70) rotate <0, 0, -15> rotate <0,   0, 0> translate < 1.8, -
2.8,  3.8>}
  object {Tulip(20) rotate <0, 0,   0> rotate <0,   0, 0> translate < 3.0,
1.2, 20.0>}
  object {Tulip( 0) rotate <0, 0,   0> rotate <0,   0, 0> translate < 4.2, -
2.0,  7.0>}

  #declare R1 = seed(129);
  #declare TulipLoop = 0;
  #while (TulipLoop < 640)
    object
    {
      Tulip(60+rand(R1)*360)
      rotate <0, 0, -25-rand(R1)*10> // Lean
      rotate <0, -rand(R1)*10, 0>    // Rotate
      translate <rand(R1)*20, -rand(R1)*1, rand(R1)*40>
      rotate <0, -45, 0>
      rotate <-5, 0, 0>
      translate <2, -0.7, 3>
    }
    #declare TulipLoop = TulipLoop + 1;
  #end

  #declare TulipLoop = 0;
  #while (TulipLoop < 64)
    object
    {
      Tulip(60+rand(R1)*360)
      rotate <0, 0, -30-rand(R1)*10> // Lean
      rotate <0, -rand(R1)*10, 0>    // Rotate
      translate <-rand(R1)*20, -rand(R1)*1, rand(R1)*10>
      rotate <0, -45, 0>
      rotate <-5, 0, 0>
      translate <-0.5, -1.5, 5>
    }
    #declare TulipLoop = TulipLoop + 1;
  #end

  #declare R1 = seed(696);
  #declare GrassLoop = 0;
  #while (GrassLoop < 4096)
    object
    {
      Grass
      rotate <-30 + 60*rand(R1), 0, -30 + 60*rand(R1)>
      rotate <0, 360*rand(R1), 0>
      scale <1 + rand(R1), 0.5 + rand(R1), 1 + rand(R1)>
      translate <-10 + 20*rand(R1), -4, rand(R1)*20>
    }
    #declare GrassLoop = GrassLoop + 1;
  #end


  object
  {
    Windmill
    translate <18, 9, 80>
  }

  #declare R1 = seed(60);
  #declare GrassLoop = 0;
  #while (GrassLoop < 48)
    object
    {
      Grass
      rotate <0, -140+100*rand(R1), 0>
      scale <20, 1+1.5*rand(R1), 8>
      translate <GrassLoop - 22, (48-GrassLoop)/24, 40>
    }
```

```
    #declare GrassLoop = GrassLoop + 1;
  #end


  #declare R1 = seed(19);
  #declare GrassLoop = 0;
  #while (GrassLoop < 128)
    object
    {
      Grass
      scale <2, (64 - abs(GrassLoop - 64))/64+1, 2>
      rotate <0, -90, -90 + GrassLoop * 2>
      translate <-20+GrassLoop/24+4*rand(R1), 2+6*rand(R1), 36>
    }
    #declare GrassLoop = GrassLoop + 1;
  #end
}
```

8.4.3. POV-Ray Benchmark Performance data

The following data is an extract of the POVVBENCH database detailing only those machines that have significance to the testing done in the laboratory.

Note: All costs are in US dollars.

POVBench Result Extract

15-Jan-02

Found 1922 matches (serial & parallel)

Cluster test results as discussed in this document

Time	POVmark	Machine	Processor	Clock Rate	Operating System	Compiler	POV Version	Release
0:00:01	14800	DARK STAR 866/24	Intel P-III	866 MHz	kernel 2.2.18_Cluster	gcc 2.95	Parallel: POV 3.1	May-01
0:00:02	7400	MK-81	Intel Pentium III (96 total)	500 MHz	Linux 2.2.2	gcc	Parallel: POV 3.01	Jun-99
0:00:03	4933.33	Fianna	dual intel pIII	450 MHz	redhat linux 6.2	egcs 2.91.66	Parallel: POV 3.1	May-01
0:00:04	3700	FAST	25 x Intel Celeron	525 MHz	Red Hat 6.1	egcs 2.91.66	Parallel: POV 3.1	Feb-00
0:00:19	778.95	Monolith Cluster	4 X Dual Celeron 500	500 MHz	FreeBSD 4.1.1-Stable	GCC	Parallel: POV 31	Jan-01
0:00:19	778.95	Quad MMC Daemon	16 x Pentium Pro 256k cache	200 MHz	Linux 2.1.127	gcc	Parallel: POV 3.02	Jan-99
0:00:20	740	Beowulf Test Cluster	Intel Pentium II (Cluster of 8 Machines)	233 MHz	Linux Redhat 7.2 (2.4.7-10 Kernel)	GCC / MPICC	Parallel: POV 3.1	Jan-02
0:00:21	704.76	chloe	2x550 & 2x466 Celeron	550 MHz	SuSe Linux 6.3	egcs-2.91.66	Parallel: POV 3.1	Jan-00
0:00:21	704.76	SMILE Beowulf Cluster	Pentium II	350 MHz	Linux Redhat 5.2	GCC	Parallel: POV 3.02	Apr-99
Single Processor Results								
0:02:08	115.63	Foremost	Pentium II	333 MHz	Microsoft NT 4.00.1381	??	Single: POV 3.02	Jun-98
0:02:09	114.73	Beowulf Test Cluster Node 1 Only	Intel Pentium II	233 MHz	Linux Redhat 7.2 (2.4.2-10 Kernel)	GCC	Single: POV 3.1	Jan-02
0:02:11	112.98	bigone	PII-300	300 MHz	Linux 2.2.5 glibc	egcs 1.1.2	Single: POV 3.02	May-99
0:02:11	112.98	Osborne PowerATX PII 300	PII 300MHz / 64MB EDO DRAM	300 MHz	Windows NT 4 (sp3)	??	Single: POV-Ray for Windows 3.02	Nov-97
0:02:12	112.12	MaxCom PC	AMD-K6	350 MHz	Windows NT 4.0 SP3	??	Single: POV 3.1	May-99
0:02:12	112.12	Compaq Deskpro 6000	Pentium Pro (256K)	200 MHz	Windows NT 4.0 SP3	Microsoft Visual C++ v5.0	Single: POV 2.2	Apr-97

Table 8-3 – POVBENCH POV-Ray Performance Benchmark Data Extract

Copyright © 2003

8.5. Appendix E – Raw Results

8.5.1. Lam MPI Cluster boot-up and tear-down

```
Script started on Tue Nov 20 22:47:31 2001
[beowulf@node1 ~]$ lamboot -v /etc/lam/lam-bhost.lam
LAM 6.5.5/MPI 2 C++/ROMIO - University of Notre Dame

Executing hboot on n0 (node1 - 1 CPU)...
Executing hboot on n1 (node2 - 1 CPU)...
Executing hboot on n2 (node3 - 1 CPU)...
Executing hboot on n3 (node4 - 1 CPU)...
Executing hboot on n4 (node5 - 1 CPU)...
Executing hboot on n5 (node6 - 1 CPU)...
Executing hboot on n6 (node7 - 1 CPU)...
Executing hboot on n7 (node8 - 1 CPU)...
topology n0(o)...n1...n2...n3...n4...n5...n6...n7...done
[beowulf@node1 ~]$ wipe -v /etc/lam/lam-bhost.lam
LAM 6.5.5/MPI 2 C++/ROMIO - University of Notre Dame

Executing tkill on n0 (node1)...
Executing tkill on n1 (node2)...
Executing tkill on n2 (node3)...
Executing tkill on n3 (node4)...
Executing tkill on n4 (node5)...
Executing tkill on n5 (node6)...
Executing tkill on n6 (node7)...
Executing tkill on n7 (node8)...
[beowulf@node1 ~]$ exit
Script done on Tue Nov 20 22:48:17 2001
```

8.5.2. POV-Ray

The following script files show the raw input/output from the command line. Each file has had trivial data removed in part to reduce the overall size.

Skyvase on 1 node

```
Script started on Fri Nov 23 23:02:24 2001
[beowulf@node8 povray31]$ -__[Kx-povray -i skyvase.pov +v1 -d ___[K+ft -x +a0.300 +r3 -q9  _-
w640 -h480 -mv2.0 +b1000 > results.txt Persistence of Vision(tm) Ray Tracer Version
3.1g.Linux.gcc
  This is an official version prepared by the POV-Ray Team(tm). See the
    documentation on how to contact the authors or visit us on the
    internet at http://www.povray.org.
Copyright 1999 POV-Ray Team(tm)
Parsing Options
  Input file: skyvase.pov (compatible to version 2.0)
  Remove bounds........On  Split unions........Off
  Library paths: /usr/local/lib/povray31 /usr/local/lib/povray31/include
Output Options
  Image resolution 640 by 480 (rows 1 to 480, columns 1 to 640).
  Output file: skyvase.tga, 24 bpp Targa, 1000 KByte buffer
  Graphic display.....Off
  Mosaic preview......Off
  CPU usage histogram.Off
  Continued trace.....Off  Allow interruption..Off  Pause when done.....Off
  Verbose messages.....On
Tracing Options
  Quality:  9
  Bounding boxes.......On  Bounding threshold: 25
  Light Buffer.........On  Vista Buffer.........On
  Antialiasing.........On  (Method 1, Threshold 0.300, Depth 3, Jitter 1.00)
  Radiosity...........Off
Animation Options
  Clock value....   0.000  (Animation off)
Redirecting Options
  All Streams to console..........On
  Debug Stream to console.........On
  Fatal Stream to console.........On
  Render Stream to console........On
  Statistics Stream to console....On
  Warning Stream to console.......On


Parsing...........skyvase.pov:83: warning: CSG union unnecessarily bounded.


Creating bounding slabs.
Scene contains 4 frame level objects; 3 infinite.

Rendering...
  -:--:-- Rendering line    0 of  480Camera is inside a non-hollow object.
Fog and participating media may not work as expected.
 supersampled 0 times.
  -:--:-- Rendering line    1 of  480 supersampled 0 times.
  -:--:-- Rendering line    2 of  480 supersampled 0 times.
  0:00:01 Rendering line    3 of  480 supersampled 0 times.
.
.
.
  0:02:09 Rendering line  479 of  480 supersampled 10 times.
  0:02:09 Rendering line  480 of  480 supersampled 10 times.        Done Tracing
skyvase.pov Statistics, Resolution 640 x 480
----------------------------------------------------------------------------
Pixels:          307840   Samples:          396160   Smpls/Pxl: 1.29
Rays:           1519523   Saved:                 9   Max Level: 5/5
----------------------------------------------------------------------------
Ray->Shape Intersection          Tests       Succeeded  Percentage
----------------------------------------------------------------------------
Cone/Cylinder                  2382542         1281629     53.79
CSG Intersection               3664171          396353     10.82
CSG Union                      2563258          738033     28.79
```

Copyright © 2003

```
Plane                         26388678        14489601     54.91
Quadric                        2563258         1571892     61.32
Sphere                         2563258          652132     25.44
Bounding Object                2382542         1281629     53.79
----------------------------------------------------------------------------
Calls to Noise:            1445139   Calls to DNoise:        2816194
----------------------------------------------------------------------------
Shadow Ray Tests:          3726692   Succeeded:                90399
Reflected Rays:            1123363
----------------------------------------------------------------------------
Smallest Alloc:                 12 bytes   Largest:          1024008
Peak memory used:          1107606 bytes
----------------------------------------------------------------------------
Time For Trace:    0 hours  2 minutes   9.0 seconds (129 seconds)
    Total Time:    0 hours  2 minutes   9.0 seconds (129 seconds)
[beowulf@node8 povray31]$ x-povray -i skyvase.pov +v1 -d +ft -x +a0.300 +r3 -q9
-w640 -h480 -mv2.0 +b1000 > results.txt Persistence of Vision(tm) Ray Tracer Version
3.1g.Linux.gcc
  This is an official version prepared by the POV-Ray Team(tm). See the
    documentation on how to contact the authors or visit us on the
    internet at http://www.povray.org.
Copyright 1999 POV-Ray Team(tm)
Parsing Options
  Input file: skyvase.pov (compatible to version 2.0)
  Remove bounds........On  Split unions........Off
  Library paths: /usr/local/lib/povray31 /usr/local/lib/povray31/include
Output Options
  Image resolution 640 by 480 (rows 1 to 480, columns 1 to 640).
  Output file: skyvase.tga, 24 bpp Targa, 1000 KByte buffer
  Graphic display.....Off
  Mosaic preview......Off
  CPU usage histogram.Off
  Continued trace.....Off  Allow interruption..Off  Pause when done.....Off
  Verbose messages.....On
Tracing Options
  Quality:  9
  Bounding boxes.......On  Bounding threshold: 25
  Light Buffer.........On  Vista Buffer.........On
  Antialiasing.........On  (Method 1, Threshold 0.300, Depth 3, Jitter 1.00)
  Radiosity...........Off
Animation Options
  Clock value....   0.000  (Animation off)
Redirecting Options
  All Streams to console..........On
  Debug Stream to console.........On
  Fatal Stream to console.........On
  Render Stream to console........On
  Statistics Stream to console....On
  Warning Stream to console.......On


Parsing...........skyvase.pov:83: warning: CSG union unnecessarily bounded.


Creating bounding slabs.
Scene contains 4 frame level objects; 3 infinite.

Rendering...
  -:--:-- Rendering line    0 of  480Camera is inside a non-hollow object.
Fog and participating media may not work as expected.
 supersampled 0 times.
  -:--:-- Rendering line    1 of  480 supersampled 0 times.
  -:--:-- Rendering line    2 of  480 supersampled 0 times.
  -:--:-- Rendering line    3 of  480 supersampled 0 times.
  -:--:-- Rendering line    4 of  480 supersampled 0 times.
  0:00:01 Rendering line    5 of  480 supersampled 0 times.
.
.
.

  0:02:08 Rendering line  479 of  480 supersampled 10 times.
  0:02:09 Rendering line  480 of  480 supersampled 10 times.        Done Tracing
skyvase.pov Statistics, Resolution 640 x 480
```

```
----------------------------------------------------------------------------
Pixels:          307840   Samples:          396160   Smpls/Pxl: 1.29
Rays:           1519523   Saved:                 9   Max Level: 5/5
----------------------------------------------------------------------------
Ray->Shape Intersection          Tests       Succeeded  Percentage
----------------------------------------------------------------------------
Cone/Cylinder                  2382542        1281629     53.79
CSG Intersection               3664171         396353     10.82
CSG Union                      2563258         738033     28.79
Plane                         26388678       14489601     54.91
Quadric                        2563258        1571892     61.32
Sphere                         2563258         652132     25.44
Bounding Object                2382542        1281629     53.79
----------------------------------------------------------------------------
Calls to Noise:            1445139   Calls to DNoise:        2816194
----------------------------------------------------------------------------
Shadow Ray Tests:          3726692   Succeeded:                90399
Reflected Rays:            1123363
----------------------------------------------------------------------------
Smallest Alloc:                 12 bytes   Largest:          1024008
Peak memory used:          1107606 bytes
----------------------------------------------------------------------------
Time For Trace:    0 hours  2 minutes   9.0 seconds (129 seconds)
    Total Time:    0 hours  2 minutes   9.0 seconds (129 seconds)
[beowulf@node8 povray31]$ exit
Script done on Fri Nov 23 23:08:30 2001
```

Skyvase on 2 nodes

```
Script started on Sat Nov 24 00:15:17 2001
[beowulf@node8 povray31]$ _mpirun n0 n0-1 ./mpi-x-povray -i skyvase.pov +v1 -d +f _t -x
+a0.300 +r3 -q9 -w640 -h480 -mv2.0 +b1000 > results.txt Persistence of Vision(tm) Ray Tracer
Version 3.1g
  This is an unofficial version compiled by:
  Leon Verrall (leon@sgi.com)
  The POV-Ray Team(tm) is not responsible for supporting this version.
Copyright 1999 POV-Ray Team(tm)
Initializing MPI-POVRAY
Slave PE 2 successfully started.

STARTING FRAME 0...


Parsing (Slave PE 2)Slave PE 1 successfully started.

Parsing (Slave PE 1)............skyvase.pov:83: warning: CSG union unnecessarily bounded.


Creating bounding slabs.
Scene contains 4 frame level objects; 3 infinite.

 Rendering...(Slave PE 2)
Camera is inside a non-hollow object.
Fog and participating media may not work as expected.
............skyvase.pov:83: warning: CSG union unnecessarily bounded.
.
..
Creating bounding slabs..
Scene contains 4 frame level objects; 3 infinite.
.
 Rendering...(Slave PE 1)
.Camera is inside a non-hollow object.
Fog and participating media may not work as expected.
........................
 0.33 of blocks complete.
Camera is inside a non-hollow object.
Fog and participating media may not work as expected.
.............................
 0.67 of blocks complete.
..........................Camera is inside a non-hollow object.
Fog and participating media may not work as expected.
..
 1.00 of blocks complete.
Camera is inside a non-hollow object.
Fog and participating media may not work as expected.
......................................
 1.33 of blocks complete.
............Camera is inside a non-hollow object.
.
.
.
99.67 of blocks complete.
All blocks are assigned. Stopping PE 2.

Done Tracing
Slave PE 2 has exited. 1 minions left...
..........
99.75 of blocks complete... 456 of  480 lines finished (in frame 0)....
99.83 of blocks complete.. 464 of  480 lines finished (in frame 0)........
99.92 of blocks complete... 472 of  480 lines finished (in frame 0)......
100.00 of blocks complete.

100.00 of blocks complete. 480 of  480 lines finished (in frame 0).

Finishing Frame 0...


All blocks are assigned. Stopping PE 1.

Done Tracing
```

Copyright © 2003

```
Slave PE 1 has exited. 0 minions left...
All slave tasks have exited!


PE Distribution Statistics:
            Slave PE  [ done ]            Slave PE  [ done ]
                   1  [42.00%]
                   2  [58.00%]

POV-Ray statistics for finished frames:
skyvase.pov Statistics (Partial Image Rendered), Resolution 640 x 480
----------------------------------------------------------------------------
Pixels:          133056   Samples:          186768   Smpls/Pxl: 1.40
Rays:            738432   Saved:                 0   Max Level: 0/5
----------------------------------------------------------------------------
Ray->Shape Intersection          Tests       Succeeded  Percentage
----------------------------------------------------------------------------
Cone/Cylinder                  1155707         633920     54.85
CSG Intersection               1789627         202500     11.32
CSG Union                      1267840         383159     30.22
Plane                         12824910        7000758     54.59
Quadric                        1267840         796920     62.86
Sphere                         1267840         336564     26.55
Bounding Object                1155707         633920     54.85
----------------------------------------------------------------------------
Calls to Noise:             729155   Calls to DNoise:        1446800
----------------------------------------------------------------------------
Shadow Ray Tests:          1812532   Succeeded:                45367
Reflected Rays:             551664
----------------------------------------------------------------------------
Smallest Alloc:                 32 bytes   Largest:          1024008
Peak memory used:          1720450 bytes
----------------------------------------------------------------------------
Time For Trace:    0 hours  1 minutes  12.0 seconds (72 seconds)
    Total Time:    0 hours  1 minutes  12.0 seconds (72 seconds)
[beowulf@node8 povray31]$ exit
Script done on Sat Nov 24 00:17:56 2001
```

Skyvase on 3 nodes

```
Script started on Sat Nov 24 00:22:09 2001
[beowulf@node8 povray31]$ mpirun no__[K0 n0-2 ./mpi-x-povray -i skyvase.pov +v1 -d +f _t -x
___[K+a0.300 +r3 -q9 -w640 -h480 -mv2.0 +b1000 > results.txt Persistence of Vision(tm) Ray
Tracer Version 3.1g
  This is an unofficial version compiled by:
  Leon Verrall (leon@sgi.com)
  The POV-Ray Team(tm) is not responsible for supporting this version.
Copyright 1999 POV-Ray Team(tm)
Initializing MPI-POVRAY
Slave PE 2 successfully started.
Slave PE 3 successfully started.

STARTING FRAME 0...

Slave PE 1 successfully started.

Parsing (Slave PE 3).
Parsing (Slave PE 2)..
Parsing (Slave PE 1).................skyvase.pov:83: warning: CSG union unnecessarily bounded.
skyvase.pov:83: warning: CSG union unnecessarily bounded.

Creating bounding slabs.
Scene contains 4 frame level objects; 3 infinite.

Creating bounding slabs.
Scene contains 4 frame level objects; 3 infinite.

 Rendering...(Slave PE 2)

 Rendering...(Slave PE 3)
Camera is inside a non-hollow object.
Fog and participating media may not work as expected.
Camera is inside a non-hollow object.
Fog and participating media may not work as expected.
.....................................
 0.33 of blocks complete.
....skyvase.pov:83: warning: CSG union unnecessarily bounded.
....
....
Creating bounding slabs...
Scene contains 4 frame level objects; 3 infinite.
..
 Rendering...(Slave PE 1)
..Camera is inside a non-hollow object.
Fog and participating media may not work as expected.
...........Camera is inside a non-hollow object.
Fog and participating media may not work as expected.
..
 0.67 of blocks complete.
Camera is inside a non-hollow object.
Fog and participating media may not work as expected.
......................................
 1.00 of blocks complete.
......................................
 1.33 of blocks complete.
.
.
.
99.33 of blocks complete.
.Camera is inside a non-hollow object.
Fog and participating media may not work as expected.
....All blocks are assigned. Stopping PE 2.

Done Tracing
Slave PE 2 has exited. 2 minions left...
..........
99.50 of blocks complete. 456 of  480 lines finished (in frame 0)..................
99.67 of blocks complete.
.
99.67 of blocks complete.. 464 of  480 lines finished (in frame 0).............All blocks are
assigned. Stopping PE 3.
```

Copyright © 2003

```
Done Tracing
Slave PE 3 has exited. 1 minions left...
.......
99.92 of blocks complete.... 472 of  480 lines finished (in frame 0).
100.00 of blocks complete.

100.00 of blocks complete. 480 of  480 lines finished (in frame 0).

Finishing Frame 0...


All blocks are assigned. Stopping PE 1.

Done Tracing
Slave PE 1 has exited. 0 minions left...
All slave tasks have exited!


PE Distribution Statistics:
            Slave PE  [ done ]            Slave PE  [ done ]
                   1  [24.00%]
                   2  [37.33%]                   3  [38.67%]


POV-Ray statistics for finished frames:
skyvase.pov Statistics (Partial Image Rendered), Resolution 640 x 480
----------------------------------------------------------------------------
Pixels:          76032   Samples:          106488   Smpls/Pxl: 1.40
Rays:           432403   Saved:                 0   Max Level: 0/5
----------------------------------------------------------------------------
Ray->Shape Intersection          Tests       Succeeded  Percentage
----------------------------------------------------------------------------
Cone/Cylinder                   670749         377745     56.32
CSG Intersection               1048494         123796     11.81
CSG Union                       755490         227670     30.14
Plane                          7462980        4090889     54.82
Quadric                         755490         459300     60.79
Sphere                          755490         199338     26.39
Bounding Object                 670749         377745     56.32
----------------------------------------------------------------------------
Calls to Noise:             459305   Calls to DNoise:         889040
----------------------------------------------------------------------------
Shadow Ray Tests:          1048064   Succeeded:                28543
Reflected Rays:             325915
----------------------------------------------------------------------------
Smallest Alloc:                 32 bytes   Largest:          1024008
Peak memory used:          1822934 bytes
----------------------------------------------------------------------------
Time For Trace:    0 hours  0 minutes  48.0 seconds (48 seconds)
    Total Time:    0 hours  0 minutes  48.0 seconds (48 seconds)
[beowulf@node8 povray31]$ exit
Script done on Sat Nov 24 00:24:35 2001
```

Skyvase on 4 nodes

```
Script started on Sun Nov 25 18:11:01 2001
[beowulf@node8 povray31]$ mpirun n0 n0-3 ./p__[Kmpi-x-povray -i sj__[Kkyvase.pov +v1 -d +f _t
-x +a0.300 +r3 -q9 -w640 -h48-__[K0 -mv2.0 +b1000 > results.txt Persistence of Vision(tm) Ray
Tracer Version 3.1g
  This is an unofficial version compiled by:
  Leon Verrall (leon@sgi.com)
  The POV-Ray Team(tm) is not responsible for supporting this version.
Copyright 1999 POV-Ray Team(tm)
Initializing MPI-POVRAY
Slave PE 2 successfully started.
Slave PE 4 successfully started.

STARTING FRAME 0...

Slave PE 1 successfully started.
Slave PE 3 successfully started.

Parsing (Slave PE 4).
Parsing (Slave PE 1)....
Parsing (Slave PE 2).....
Parsing (Slave PE 3).................skyvase.pov:83: warning: CSG union unnecessarily bounded.
.skyvase.pov:83: warning: CSG union unnecessarily bounded.



Creating bounding slabs.
Scene contains 4 frame level objects; 3 infinite.
skyvase.pov:83: warning: CSG union unnecessarily bounded.

Creating bounding slabs.

Scene contains 4 frame level objects; 3 infinite.

Creating bounding slabs.
 Rendering...(Slave PE 2)

 Rendering...(Slave PE 4)
Camera is inside a non-hollow object.
Fog and participating media may not work as expected.

Scene contains 4 frame level objects; 3 infinite.
Camera is inside a non-hollow object.
Fog and participating media may not work as expected.

 Rendering...(Slave PE 3)
Camera is inside a non-hollow object.
Fog and participating media may not work as expected.
..............................................
 0.33 of blocks complete.
.............................
 0.67 of blocks complete.
.............skyvase.pov:83: warning: CSG union unnecessarily bounded.
......Camera is inside a non-hollow object.
Fog and participating media may not work as expected.
....Camera is inside a non-hollow object.
Fog and participating media may not work as expected.
..
 1.00 of blocks complete.
Camera is inside a non-hollow object.
Fog and participating media may not work as expected.
.
....
Creating bounding slabs.......
Scene contains 4 frame level objects; 3 infinite.
......
 Rendering...(Slave PE 1)
......Camera is inside a non-hollow object.
Fog and participating media may not work as expected.
.....................
 1.33 of blocks complete.
```

```
99.33 of blocks complete.
......Camera is inside a non-hollow object.
Fog and participating media may not work as expected.
........All blocks are assigned. Stopping PE 2.

Done Tracing..
Slave PE 2 has exited. 3 minions left...
.........All blocks are assigned. Stopping PE 4.
.
Done Tracing
Slave PE 4 has exited. 2 minions left...
.
99.67 of blocks complete.
..
99.67 of blocks complete. 456 of  480 lines finished (in frame 0)...........
99.83 of blocks complete. 464 of  480 lines finished (in frame 0).All blocks are assigned.
Stopping PE 1.
.
Done Tracing.
Slave PE 1 has exited. 1 minions left...
......
99.92 of blocks complete. 472 of  480 lines finished (in frame 0).......
100.00 of blocks complete.

100.00 of blocks complete. 480 of  480 lines finished (in frame 0).

Finishing Frame 0...


All blocks are assigned. Stopping PE 3.

Done Tracing
Slave PE 3 has exited. 0 minions left...
All slave tasks have exited!


PE Distribution Statistics:
            Slave PE  [ done ]            Slave PE  [ done ]
                   1  [15.00%]
                   2  [29.33%]                   3  [27.67%]
                   4  [28.00%]

POV-Ray statistics for finished frames:
skyvase.pov Statistics (Partial Image Rendered), Resolution 640 x 480
----------------------------------------------------------------------------
Pixels:           87648   Samples:          116560   Smpls/Pxl: 1.33
Rays:            454282   Saved:                 3   Max Level: 0/5
----------------------------------------------------------------------------
Ray->Shape Intersection          Tests       Succeeded  Percentage
----------------------------------------------------------------------------
Cone/Cylinder                   713545         379049     53.12
CSG Intersection               1092594         119175     10.91
CSG Union                       758098         217557     28.70
Plane                          7893548        4307876     54.57
Quadric                         758098         480966     63.44
Sphere                          758098         190100     25.08
Bounding Object                 713545         379049     53.12
----------------------------------------------------------------------------
Calls to Noise:             430048   Calls to DNoise:         840608
----------------------------------------------------------------------------
Shadow Ray Tests:          1118580   Succeeded:                24834
Reflected Rays:             337722
----------------------------------------------------------------------------
Smallest Alloc:                 32 bytes   Largest:          1024008
Peak memory used:          1925418 bytes
----------------------------------------------------------------------------
Time For Trace:    0 hours  0 minutes  36.0 seconds (36 seconds)
    Total Time:    0 hours  0 minutes  36.0 seconds (36 seconds)
[beowulf@node8 povray31]$ exit
```

Script done on Sun Nov 25 18:13:10 2001

Skyvase on 5 nodes

Script started on Sat Nov 24 00:29:40 2001

```
[beowulf@node8 povray31]$ mpirun n0 n0-4 mpirun__[K__[K__[K__[K__[Kpi-x-povray -i skyvase.pov
+v1 -d +ft  _-x +a0.300 +r3 -q9 -w640 -h480 -mv2.0 +b1000 > results.txt Persistence of
Vision(tm) Ray Tracer Version 3.1g
  This is an unofficial version compiled by:
  Leon Verrall (leon@sgi.com)
  The POV-Ray Team(tm) is not responsible for supporting this version.
Copyright 1999 POV-Ray Team(tm)
Initializing MPI-POVRAY
Slave PE 2 successfully started.
Slave PE 4 successfully started.

STARTING FRAME 0...

Parsing (Slave PE 4)Slave PE 3 successfully started.
Slave PE 5 successfully started.
.
Parsing (Slave PE 5).
Parsing (Slave PE 2)....
Parsing (Slave PE 3)...Slave PE 1 successfully started.
.
Parsing (Slave PE 1)....................skyvase.pov:83: warning: CSG union unnecessarily
bounded.

.skyvase.pov:83: warning: CSG union unnecessarily bounded.

Creating bounding slabs.
.
Scene contains 4 frame level objects; 3 infinite.
.
 Rendering...(Slave PE 2)

Creating bounding slabs.
Scene contains 4 frame level objects; 3 infinite.
..skyvase.pov:83: warning: CSG union unnecessarily bounded.
skyvase.pov:83: warning: CSG union unnecessarily bounded.
Camera is inside a non-hollow object.
Fog and participating media may not work as expected.

 Rendering...(Slave PE 4)

.
.Camera is inside a non-hollow object.
Fog and participating media may not work as expected.
.
Creating bounding slabs.
Creating bounding slabs.
Scene contains 4 frame level objects; 3 infinite.

Scene contains 4 frame level objects; 3 infinite.
.
 Rendering...(Slave PE 3)

 Rendering...(Slave PE 5)
.Camera is inside a non-hollow object.
Fog and participating media may not work as expected.
Camera is inside a non-hollow object.
Fog and participating media may not work as expected.
...................................
 0.33 of blocks complete.
................................
 Rendering...(Slave PE 1)

98.67 of blocks complete.
.Camera is inside a non-hollow object.
Fog and participating media may not work as expected.
............Camera is inside a non-hollow object.
Fog and participating media may not work as expected.
...........All blocks are assigned. Stopping PE 1.
....
Done Tracing..
99.00 of blocks complete.
```

Copyright © 2003

```
..
Slave PE 1 has exited. 4 minions left...
................................
99.25 of blocks complete. 456 of  480 lines finished (in frame 0)..
99.33 of blocks complete.
............................
99.50 of blocks complete. 464 of  480 lines finished (in frame 0)...
99.67 of blocks complete.
All blocks are assigned. Stopping PE 4.
..All blocks are assigned. Stopping PE 5.
..
Done Tracing..
Done Tracing..
Slave PE 4 has exited. 3 minions left...
......
99.83 of blocks complete. 472 of  480 lines finished (in frame 0).
Slave PE 5 has exited. 2 minions left...
.....All blocks are assigned. Stopping PE 2.
.
Done Tracing
Slave PE 2 has exited. 1 minions left...
...
100.00 of blocks complete.
100.00 of blocks complete. 480 of  480 lines finished (in frame 0).
Finishing Frame 0...

All blocks are assigned. Stopping PE 3.

Done Tracing
Slave PE 3 has exited. 0 minions left...
All slave tasks have exited!

PE Distribution Statistics:
            Slave PE  [ done ]            Slave PE  [ done ]
                   1  [11.00%]
                   2  [23.67%]                   3  [22.67%]
                   4  [21.00%]                   5  [21.67%]


POV-Ray statistics for finished frames:
skyvase.pov Statistics (Partial Image Rendered), Resolution 640 x 480
----------------------------------------------------------------------------
Pixels:           71808   Samples:           95928   Smpls/Pxl: 1.34
Rays:            366381   Saved:                 0   Max Level: 0/5
----------------------------------------------------------------------------
Ray->Shape Intersection          Tests       Succeeded  Percentage
----------------------------------------------------------------------------
Cone/Cylinder                   571431          297967     52.14
CSG Intersection                869398           99074     11.40
CSG Union                       595934          176956     29.69
Plane                          6310244         3463621     54.89
Quadric                         595934          370114     62.11
Sphere                          595934          155124     26.03
Bounding Object                 571431          297967     52.14
----------------------------------------------------------------------------
Calls to Noise:             355777   Calls to DNoise:         678932
----------------------------------------------------------------------------
Shadow Ray Tests:           894800   Succeeded:                24286
Reflected Rays:             270453
----------------------------------------------------------------------------
Smallest Alloc:                 32 bytes   Largest:          1024008
Peak memory used:          1822974 bytes
----------------------------------------------------------------------------
Time For Trace:    0 hours  0 minutes  29.0 seconds (29 seconds)
    Total Time:    0 hours  0 minutes  29.0 seconds (29 seconds)
[beowulf@node8 povray31]$ exit
```

Script done on Sat Nov 24 00:31:43 2001

Skyvase on 6 nodes

```
Script started on Sat Nov 24 00:32:31 2001
[beowulf@node8 povray31]$ mpirun n0 n0-5 ./mpi-x-povray -i skyvase.pov +v1 -d +f _t -x +a0.300
+r3 -q9 -w640 -h480 -mv2.0 +b1000 > results.txt Persistence of Vision(tm) Ray Tracer Version
3.1g
  This is an unofficial version compiled by:
  Leon Verrall (leon@sgi.com)
  The POV-Ray Team(tm) is not responsible for supporting this version.
Copyright 1999 POV-Ray Team(tm)
Initializing MPI-POVRAY
Slave PE 2 successfully started.
Slave PE 4 successfully started.
Slave PE 6 successfully started.

STARTING FRAME 0...

Slave PE 1 successfully started.
Slave PE 3 successfully started.
Slave PE 5 successfully started.

Parsing (Slave PE 1).
Parsing (Slave PE 6)
Parsing (Slave PE 2)
Parsing (Slave PE 3)
Parsing (Slave PE 5)....
Parsing (Slave PE 4)...................................skyvase.pov:83: warning: CSG union
unnecessarily bounded.
.
skyvase.pov:83: warning: CSG union unnecessarily bounded.

Creating bounding slabs.
Scene contains 4 frame level objects; 3 infinite.

 Rendering...(Slave PE 2)


Creating bounding slabs.
Scene contains 4 frame level objects; 3 infinite.
..Camera is inside a non-hollow object.
Fog and participating media may not work as expected.
skyvase.pov:83: warning: CSG union unnecessarily bounded.
skyvase.pov:83: warning: CSG union unnecessarily bounded.


.
 Rendering...(Slave PE 4)
.Camera is inside a non-hollow object.
Fog and participating media may not work as expected.
.
Creating bounding slabs.
Creating bounding slabs..
Scene contains 4 frame level objects; 3 infinite.

Scene contains 4 frame level objects; 3 infinite.

 Rendering...(Slave PE 3)

 Rendering...(Slave PE 5)
Camera is inside a non-hollow object.
Fog and participating media may not work as expected.
skyvase.pov:83: warning: CSG union unnecessarily bounded.
.Camera is inside a non-hollow object.
Fog and participating media may not work as expected.

..
Creating bounding slabs..
Scene contains 4 frame level objects; 3 infinite.

 Rendering...(Slave PE 6)
.......Camera is inside a non-hollow object.
Fog and participating media may not work as expected.
....................
```

```
 0.33 of blocks complete.
........................................
 0.67 of blocks complete.
.....................................
 1.00 of blocks complete.
.....................
 1.33 of blocks complete.
...
 Rendering...(Slave PE 1)
Camera is inside a non-hollow object.
Fog and participating media may not work as expected.

...Camera is inside a non-hollow object.
Fog and participating media may not work as expected.
Camera is inside a non-hollow object.
Fog and participating media may not work as expected.
.........Camera is inside a non-hollow object.
99.67 of blocks complete.
..All blocks are assigned. Stopping PE 2.
.All blocks are assigned. Stopping PE 5.

Done TracingAll blocks are assigned. Stopping PE 4.

Slave PE 2 has exited. 3 minions left...

Done Tracing
Done Tracing
Slave PE 5 has exited. 2 minions left...

Slave PE 4 has exited. 1 minions left...
........
99.83 of blocks complete... 464 of  480 lines finished (in frame 0).....
99.92 of blocks complete.. 472 of  480 lines finished (in frame 0).......
100.00 of blocks complete.

100.00 of blocks complete. 480 of  480 lines finished (in frame 0).

Finishing Frame 0...


All blocks are assigned. Stopping PE 1.

Done Tracing
Slave PE 1 has exited. 0 minions left...
All slave tasks have exited!


PE Distribution Statistics:
            Slave PE  [ done ]            Slave PE  [ done ]
                   1  [ 8.67%]
                   2  [18.00%]                   3  [18.33%]
                   4  [19.00%]                   5  [18.33%]
                   6  [17.67%]

POV-Ray statistics for finished frames:
skyvase.pov Statistics (Partial Image Rendered), Resolution 640 x 480
----------------------------------------------------------------------------
Pixels:           27456   Samples:           40240   Smpls/Pxl: 1.47
Rays:            155069   Saved:                 3   Max Level: 0/5
----------------------------------------------------------------------------
Ray->Shape Intersection          Tests       Succeeded  Percentage
----------------------------------------------------------------------------
Cone/Cylinder                   245864          127244     51.75
CSG Intersection                373108           36704      9.84
CSG Union                       254488           73191     28.76
Plane                          2713128         1480394     54.56
Quadric                         254488          162635     63.91
Sphere                          254488           75631     29.72
Bounding Object                 245864          127244     51.75
----------------------------------------------------------------------------
Calls to Noise:             150696   Calls to DNoise:         291116
----------------------------------------------------------------------------
Shadow Ray Tests:           387520   Succeeded:                 6545
```

```
Reflected Rays:            114829
----------------------------------------------------------------------------
Smallest Alloc:                32 bytes   Largest:          1024008
Peak memory used:          1925458 bytes
----------------------------------------------------------------------------
Time For Trace:    0 hours  0 minutes  25.0 seconds (25 seconds)
    Total Time:    0 hours  0 minutes  25.0 seconds (25 seconds)
[beowulf@node8 povray31]$ exit
```
Script done on Sat Nov 24 00:34:26 2001

Skyvase on 7 nodes

Script started on Fri Nov 23 23:13:37 2001

```
[beowulf@node8 povray31]$ _mpirun n0 N ./mpi-x-povray -i skyvase.pov +v1 -d +ft - _x +a0.300
+r3 -q9 -w640 -h480 -mv2.0 ___[K+b1000 > results.txt Persistence of Vision(tm) Ray Tracer
Version 3.1g
  This is an unofficial version compiled by:
  Leon Verrall (leon@sgi.com)
  The POV-Ray Team(tm) is not responsible for supporting this version.
Copyright 1999 POV-Ray Team(tm)
Initializing MPI-POVRAY
Slave PE 2 successfully started.
Slave PE 4 successfully started.
Slave PE 6 successfully started.

STARTING FRAME 0...


Parsing (Slave PE 2)
Parsing (Slave PE 4)..Slave PE 1 successfully started.
Slave PE 3 successfully started.
Slave PE 5 successfully started.
Slave PE 7 successfully started.
..
Parsing (Slave PE 7)..
Parsing (Slave PE 1)....
Parsing (Slave PE 6).........
Parsing (Slave PE 3)..........
Parsing (Slave PE 5)............skyvase.pov:83: warning: CSG union unnecessarily bounded.
..
skyvase.pov:83: warning: CSG union unnecessarily bounded.
.
Creating bounding slabs.
Scene contains 4 frame level objects; 3 infinite.

 Rendering...(Slave PE 2)
Camera is inside a non-hollow object.
Fog and participating media may not work as expected.

Creating bounding slabs.
Scene contains 4 frame level objects; 3 infinite.

Creating bounding slabs.
 Rendering...(Slave PE 6)

Scene contains 4 frame level objects; 3 infinite.

 Rendering...(Slave PE 4)
Camera is inside a non-hollow object.
Fog and participating media may not work as expected.
Camera is inside a non-hollow object.
Fog and participating media may not work as expected.
..........skyvase.pov:83: warning: CSG union unnecessarily bounded.
skyvase.pov:83: warning: CSG union unnecessarily bounded.
.

skyvase.pov:83: warning: CSG union unnecessarily bounded.
.
Creating bounding slabs..
Creating bounding slabs.
Scene contains 4 frame level objects; 3 infinite.

Scene contains 4 frame level objects; 3 infinite.
```

```
 Rendering...(Slave PE 3)

 Rendering...(Slave PE 5)
.
Creating bounding slabs.
Scene contains 4 frame level objects; 3 infinite.
Camera is inside a non-hollow object.
Fog and participating media may not work as expected.
Camera is inside a non-hollow object.
Fog and participating media may not work as expected.
..
 Rendering...(Slave PE 7)
Camera is inside a non-hollow object.
Fog and participating media may not work as expected.
..................................................
 0.33 of blocks complete.
..................
 0.67 of blocks complete.
...................................
 1.00 of blocks complete.
..........................................
 1.33 of blocks complete.
98.67 of blocks complete.
...............All blocks are assigned. Stopping PE 6.

Done Tracing
Slave PE 6 has exited. 6 minions left...
........................
99.00 of blocks complete.
.....................
99.17 of blocks complete. 456 of  480 lines finished (in frame 0).......
99.33 of blocks complete.
....All blocks are assigned. Stopping PE 5.
....
Done Tracing...All blocks are assigned. Stopping PE 7.
...........
99.50 of blocks complete. 464 of  480 lines finished (in frame 0).
Slave PE 5 has exited. 5 minions left...
...
99.67 of blocks complete.

Done TracingAll blocks are assigned. Stopping PE 2.
.All blocks are assigned. Stopping PE 4.

Slave PE 7 has exited. 4 minions left...

Done Tracing
Done Tracing
Slave PE 2 has exited. 3 minions left...

Slave PE 4 has exited. 2 minions left...
.........All blocks are assigned. Stopping PE 3.

Done Tracing
Slave PE 3 has exited. 1 minions left...
...........
99.92 of blocks complete... 472 of  480 lines finished (in frame 0).
100.00 of blocks complete.

100.00 of blocks complete. 480 of  480 lines finished (in frame 0).

Finishing Frame 0...


All blocks are assigned. Stopping PE 1.

Done Tracing
Slave PE 1 has exited. 0 minions left...
All slave tasks have exited!
```

```
PE Distribution Statistics:
            Slave PE  [ done ]            Slave PE  [ done ]
                   1  [ 6.33%]
                   2  [15.33%]                   3  [16.00%]
                   4  [15.33%]                   5  [15.67%]
                   6  [15.00%]                   7  [16.33%]


POV-Ray statistics for finished frames:
skyvase.pov Statistics (Partial Image Rendered), Resolution 640 x 480
----------------------------------------------------------------------------
Pixels:           20064   Samples:           28008   Smpls/Pxl: 1.40
Rays:            104286   Saved:                 0   Max Level: 0/5
----------------------------------------------------------------------------
Ray->Shape Intersection          Tests       Succeeded  Percentage
----------------------------------------------------------------------------
Cone/Cylinder                   161648          97365     60.23
CSG Intersection                259013          35699     13.78
CSG Union                       194730          64334     33.04
Plane                          1811210         993641     54.86
Quadric                         194730         158522     81.41
Sphere                          194730          65474     33.62
Bounding Object                 161648          97365     60.23
----------------------------------------------------------------------------
Calls to Noise:            122814   Calls to DNoise:         240434
----------------------------------------------------------------------------
Shadow Ray Tests:          246972   Succeeded:                 7013
Reflected Rays:             76278
----------------------------------------------------------------------------
Smallest Alloc:                 32 bytes   Largest:          1024008
Peak memory used:         1925478 bytes
----------------------------------------------------------------------------
Time For Trace:    0 hours  0 minutes  22.0 seconds (22 seconds)
    Total Time:    0 hours  0 minutes  22.0 seconds (22 seconds)
[beowulf@node8 povray31]$ exit
Script done on Fri Nov 23 23:15:47 2001
```

Skyvase on 8 nodes

```
Script started on Sun Nov 25 18:14:22 2001
[beowulf@node8 povray31]$ mpirun n0 n0-7 ./pm__[Kmpi__[K__[K__[K__[Kmpi-x-povray -i
skyvase.pov +v1 -d +f _t -x +a0.300 +r3 -q9 -w640 =h480__[K__[K__[K__[K__[K-h480 -mv2.0 +b1000
> results.txt Persistence of Vision(tm) Ray Tracer Version 3.1g
  This is an unofficial version compiled by:
  Leon Verrall (leon@sgi.com)
  The POV-Ray Team(tm) is not responsible for supporting this version.
Copyright 1999 POV-Ray Team(tm)
Initializing MPI-POVRAY
Slave PE 1 successfully started.
Slave PE 2 successfully started.
Slave PE 3 successfully started.
Slave PE 4 successfully started.
Slave PE 5 successfully started.
Slave PE 6 successfully started.
Slave PE 7 successfully started.
Slave PE 8 successfully started.

STARTING FRAME 0...


Parsing (Slave PE 4)
Parsing (Slave PE 6)
Parsing (Slave PE 8)
Parsing (Slave PE 1)
Parsing (Slave PE 2)
Parsing (Slave PE 5)........
Parsing (Slave PE 3)..........
Parsing (Slave PE 7)....................................
skyvase.pov:83: warning: CSG union unnecessarily bounded.

Creating bounding slabs.
Scene contains 4 frame level objects; 3 infinite.
```

```
 Rendering...(Slave PE 3)
 Rendering...(Slave PE 2)
 Rendering...(Slave PE 4)
 Rendering...(Slave PE 8)
 Rendering...(Slave PE 6)
 Rendering...(Slave PE 7)
 Rendering...(Slave PE 5)
..........................................................
 0.33 of blocks complete.
.......................
 0.67 of blocks complete.

 1.00 of blocks complete.
..........................
 1.33 of blocks complete.
 Rendering...(Slave PE 1)
..........................
98.58 of blocks complete........... 456 of  480 lines finished (in frame 0)....
98.67 of blocks complete.
...All blocks are assigned. Stopping PE 7.

Done Tracing
Slave PE 7 has exited. 7 minions left...
.......................All blocks are assigned. Stopping PE 5.

Done Tracing..
Slave PE 5 has exited. 6 minions left...
...
99.00 of blocks complete.
..................................
99.33 of blocks complete.
....All blocks are assigned. Stopping PE 8.
...All blocks are assigned. Stopping PE 2.
...
Done Tracing..
Done Tracing...All blocks are assigned. Stopping PE 3.
.
Slave PE 8 has exited. 5 minions left...
.
Slave PE 2 has exited. 4 minions left...

Done Tracing
99.67 of blocks complete.

Slave PE 3 has exited. 3 minions left...
All blocks are assigned. Stopping PE 4.

Done Tracing
Slave PE 4 has exited. 2 minions left...
..All blocks are assigned. Stopping PE 6.

Done Tracing
Slave PE 6 has exited. 1 minions left...
....
99.83 of blocks complete. 464 of  480 lines finished (in frame 0)........
99.92 of blocks complete. 472 of  480 lines finished (in frame 0)........
100.00 of blocks complete.

100.00 of blocks complete. 480 of  480 lines finished (in frame 0).

Finishing Frame 0...


All blocks are assigned. Stopping PE 1.

Done Tracing
Slave PE 1 has exited. 0 minions left...
All slave tasks have exited!


PE Distribution Statistics:
            Slave PE  [ done ]            Slave PE  [ done ]
                   1  [ 4.67%]
```

```
                   2  [13.33%]                     3  [13.67%]
                   4  [13.67%]                     5  [14.33%]
                   6  [13.33%]                     7  [13.00%]
                   8  [14.00%]

POV-Ray statistics for finished frames:
skyvase.pov Statistics (Partial Image Rendered), Resolution 640 x 480
----------------------------------------------------------------------------
Pixels:           14784   Samples:           23888   Smpls/Pxl: 1.62
Rays:             81064   Saved:                 0   Max Level: 0/5
----------------------------------------------------------------------------
Ray->Shape Intersection          Tests       Succeeded  Percentage
----------------------------------------------------------------------------
Cone/Cylinder                   123967           82687     66.70
CSG Intersection                206654           27632     13.37
CSG Union                       165374           47952     29.00
Plane                          1405044          779425     55.47
Quadric                         165374          128278     77.57
Sphere                          165374           43523     26.32
Bounding Object                 123967           82687     66.70
----------------------------------------------------------------------------
Calls to Noise:              91622   Calls to DNoise:         186122
----------------------------------------------------------------------------
Shadow Ray Tests:           186268   Succeeded:                 5234
Reflected Rays:              57176
----------------------------------------------------------------------------
Smallest Alloc:                 32 bytes   Largest:          1024008
Peak memory used:          2130426 bytes
----------------------------------------------------------------------------
Time For Trace:    0 hours  0 minutes  20.0 seconds (20 seconds)
    Total Time:    0 hours  0 minutes  20.0 seconds (20 seconds)
[beowulf@node8 povray31]$ exit
Script done on Sun Nov 25 18:16:23 2001
```

Poolballs on 1 node – Sequential POV-Ray

Script started on Sat Nov 24 03:03:00 2001

```
[beowulf@node8 povray31]$ x-povray -i demo.pov -w1024 -768___h768___ Persistence of Vision(tm)
Ray Tracer Version 3.1g.Linux.gcc
  This is an official version prepared by the POV-Ray Team(tm). See the
    documentation on how to contact the authors or visit us on the
    internet at http://www.povray.org.
Copyright 1999 POV-Ray Team(tm)
Parsing Options
  Input file: demo.pov (compatible to version 3.1)
  Remove bounds........On  Split unions........Off
  Library paths: /usr/local/lib/povray31 /usr/local/lib/povray31/include
Output Options
  Image resolution 1024 by 768 (rows 1 to 768, columns 1 to 1024).
  Output file: demo.png, 24 bpp PNG
  Graphic display.....Off
  Mosaic preview......Off
  CPU usage histogram.Off
  Continued trace.....Off  Allow interruption...On  Pause when done.....Off
  Verbose messages....Off
Tracing Options
  Quality:  9
  Bounding boxes.......On  Bounding threshold: 25
  Light Buffer.........On  Vista Buffer.........On
  Antialiasing........Off
  Radiosity...........Off
Animation Options
  Clock value....   0.000  (Animation off)
Redirecting Options
  All Streams to console..........On
  Debug Stream to console.........On
  Fatal Stream to console.........On
  Render Stream to console........On
  Statistics Stream to console....On
  Warning Stream to console.......On
```

```
Parsing........demo.pov:26: warning: All #version and #declares of float, vector, and color
require semi-colon ';' at end.
demo.pov:50: warning: All #version and #declares of float, vector, and color require semi-
colon ';' at end.
13_Pball.inc:100: warning: All #version and #declares of float, vector, and color require
semi-colon ';' at end.

No pigment type given.
Creating bounding slabs.
Scene contains 13 frame level objects; 0 infinite.

Rendering... Done Tracing
demo.pov Statistics, Resolution 1024 x 768
----------------------------------------------------------------------------
Pixels:          786432   Samples:          786432   Smpls/Pxl: 1.00
Rays:           1343159   Saved:              3401   Max Level: 4/5
----------------------------------------------------------------------------
Ray->Shape Intersection          Tests       Succeeded  Percentage
----------------------------------------------------------------------------
Box                           19451664        13021308     66.94
Cone/Cylinder                 25935552         5203596     20.06
CSG Intersection              95097024         4222255      4.44
CSG Union                     34580736         1595749      4.61
Sphere                        43225920         6257460     14.48
True Type Font                51871104         1392782      2.69
----------------------------------------------------------------------------
Calls to Noise:                      0   Calls to DNoise:         356028
----------------------------------------------------------------------------
Shadow Ray Tests:         10679994   Succeeded:                92606
Reflected Rays:             556727
----------------------------------------------------------------------------
Smallest Alloc:                 10 bytes   Largest:            20508
Peak memory used:           179004 bytes
----------------------------------------------------------------------------
Time For Trace:    0 hours 42 minutes  21.0 seconds (2541 seconds)
    Total Time:    0 hours 42 minutes  21.0 seconds (2541 seconds)
[beowulf@node8 povray31]$ exit
Script done on Sat Nov 24 03:47:30 2001
```

Poolballs on 1 node – Parallel POV-Ray

```
Script started on Sun Nov 25 18:32:46 2001
[beowulf@node8 povray31]$ mpirun ./povray_________ ./po___[1@mp_p_[1@po___[1@ip__[1@0p__[1@-
p___[Pp___[Pp__[1@-p__[1@xp__[1@-p__________[1@n.__[1@0.__[1@ .__[1@n.__[1@0.__[1@ ._./mpi-x-
povra y__y_[K_y -i demo.pov -h__[Kw1024 -h768 Persistence of Vision(tm) Ray Tracer Version
3.1g
  This is an unofficial version compiled by:
  Leon Verrall (leon@sgi.com)
  The POV-Ray Team(tm) is not responsible for supporting this version.
Copyright 1999 POV-Ray Team(tm)
Initializing MPI-POVRAY

Parsing (Slave PE 1).....demo.pov:26: warning: All #version and #declares of float, vector,
and color require semi-colon ';' at end.

demo.pov:28: warning: All #version and #declares of float, vector, and color require semi-
colon ';' at end.
13_Pball.inc:98: warning: All #version and #declares of float, vector, and color require semi-
colon ';' at end.
No pigment type given.
Creating bounding slabs.
Scene contains 13 frame level objects; 0 infinite.
 Rendering...(Slave PE 1)
...All blocks are assigned. Stopping PE 1.

Done Tracing
Slave PE 1 has exited. 0 minions left...
All slave tasks have exited!

PE Distribution Statistics:
            Slave PE  [ done ]            Slave PE  [ done ]
```

```
                   1  [100.00%]

POV-Ray statistics for finished frames:
demo.pov Statistics, Resolution 1024 x 768
----------------------------------------------------------------------------
Pixels:          786432   Samples:          786432   Smpls/Pxl: 1.00
Rays:           1343159   Saved:              3401   Max Level: 0/5
----------------------------------------------------------------------------
Ray->Shape Intersection          Tests       Succeeded  Percentage
----------------------------------------------------------------------------
Box                           19451664        13021308     66.94
Cone/Cylinder                 25935552         5203596     20.06
CSG Intersection              95097024         4222255      4.44
CSG Union                     34580736         1595749      4.61
Sphere                        43225920         6256890     14.47
True Type Font                51871104         1392782      2.69
----------------------------------------------------------------------------
Calls to Noise:                  0   Calls to DNoise:         356038
----------------------------------------------------------------------------
Shadow Ray Tests:         10679994   Succeeded:                92606
Reflected Rays:             556727
----------------------------------------------------------------------------
Smallest Alloc:                32 bytes   Largest:            20508
Peak memory used:           761161 bytes
----------------------------------------------------------------------------
Time For Trace:    0 hours 29 minutes  46.0 seconds (1786 seconds)
    Total Time:    0 hours 29 minutes  46.0 seconds (1786 seconds)
[beowulf@node8 povray31]$ exit
Script done on Sun Nov 25 19:04:38 2001
```

Poolballs on 2 nodes

Script started on Sat Nov 24 02:34:04 2001

```
[beowulf@node8 povray31]$ mpirun n0 n0-1 ./mpi-x-povray -i demo.pov -w6__[K1024 -h768 _
Persistence of Vision(tm) Ray Tracer Version 3.1g
  This is an unofficial version compiled by:
  Leon Verrall (leon@sgi.com)
  The POV-Ray Team(tm) is not responsible for supporting this version.
Copyright 1999 POV-Ray Team(tm)
Initializing MPI-POVRAY

Parsing (Slave PE 1)...
Parsing (Slave PE 2).....demo.pov:26: warning: All #version and #declares of float, vector,
and color require semi-colon ';' at end.

demo.pov:28: warning: All #version and #declares of float, vector, and color require semi-
colon ';' at end.
13_Pball.inc:101: warning: All #version and #declares of float, vector, and color require
semi-colon ';' at end.
No pigment type given.

Creating bounding slabs.
Scene contains 13 frame level objects; 0 infinite.

 Rendering...(Slave PE 2)

Creating bounding slabs.
Scene contains 13 frame level objects; 0 infinite.

 Rendering...(Slave PE 1)
...All blocks are assigned. Stopping PE 2.

Done Tracing
Slave PE 2 has exited. 1 minions left...
......................All blocks are assigned. Stopping PE 1.

Done Tracing
Slave PE 1 has exited. 0 minions left...
All slave tasks have exited!
```

```
PE Distribution Statistics:
            Slave PE  [ done ]            Slave PE  [ done ]
                   1  [48.96%]
                   2  [51.04%]

POV-Ray statistics for finished frames:
demo.pov Statistics (Partial Image Rendered), Resolution 1024 x 768
----------------------------------------------------------------------------
Pixels:          385024   Samples:          385024   Smpls/Pxl: 1.00
Rays:            660462   Saved:              1298   Max Level: 0/5
----------------------------------------------------------------------------
Ray->Shape Intersection          Tests       Succeeded  Percentage
----------------------------------------------------------------------------
Box                            9556641         6395308     66.92
Cone/Cylinder                 12742188         2552058     20.03
CSG Intersection              46721356         2094686      4.48
CSG Union                     16989584          790196      4.65
Sphere                        21236980         3092195     14.56
True Type Font                25484376          690290      2.71
----------------------------------------------------------------------------
Calls to Noise:                      0   Calls to DNoise:         173563
----------------------------------------------------------------------------
Shadow Ray Tests:          5234905   Succeeded:                44974
Reflected Rays:             275438
----------------------------------------------------------------------------
Smallest Alloc:                 32 bytes   Largest:            20508
Peak memory used:           925085 bytes
----------------------------------------------------------------------------
Time For Trace:    0 hours 15 minutes   6.0 seconds (906 seconds)
    Total Time:    0 hours 15 minutes   6.0 seconds (906 seconds)
[beowulf@node8 povray31]$ exit
Script done on Sat Nov 24 03:02:49 2001
```

Poolballs on 3 nodes

```
Script started on Sat Nov 24 02:21:11 2001
[beowulf@node8 povray31]$ mpirun n0 n0-2 ./mpi-x-povray -i demo.pov -w1024 -h768 _ Persistence
of Vision(tm) Ray Tracer Version 3.1g
  This is an unofficial version compiled by:
  Leon Verrall (leon@sgi.com)
  The POV-Ray Team(tm) is not responsible for supporting this version.
Copyright 1999 POV-Ray Team(tm)
Initializing MPI-POVRAY

Parsing (Slave PE 1).
Parsing (Slave PE 2)......
Parsing (Slave PE 3)......
demo.pov:26: warning: All #version and #declares of float, vector, and color require semi-
colon ';' at end.
: All #version and #declares of float, vector, and color require semi-colon ';' at end.
13_Pball.inc:98: warning: All #version and #declares of float, vector, and color require semi-
colon ';' at end.
No pigment type given.
Creating bounding slabs.No pigment type given.
Scene contains 13 frame level objects; 0 infinite.
 Rendering...(Slave PE 3)
Creating bounding slabs..
Scene contains 13 frame level objects; 0 infinite.
 Rendering...(Slave PE 2)
Creating bounding slabs.
Scene contains 13 frame level objects; 0 infinite.
 Rendering...(Slave PE 1)
.....All blocks are assigned. Stopping PE 2.

Done Tracing
Slave PE 2 has exited. 2 minions left...
.........All blocks are assigned. Stopping PE 3.

Done Tracing
Slave PE 3 has exited. 1 minions left...
```

```
....................All blocks are assigned. Stopping PE 1.
Done Tracing
Slave PE 1 has exited. 0 minions left...
All slave tasks have exited!

PE Distribution Statistics:
            Slave PE  [ done ]            Slave PE  [ done ]
                   1  [31.38%]
                   2  [34.24%]                   3  [34.38%]


POV-Ray statistics for finished frames:
demo.pov Statistics (Partial Image Rendered), Resolution 1024 x 768
----------------------------------------------------------------------------
Pixels:          246784   Samples:          246784   Smpls/Pxl: 1.00
Rays:            430194   Saved:              2213   Max Level: 0/5
----------------------------------------------------------------------------
Ray->Shape Intersection          Tests       Succeeded  Percentage
----------------------------------------------------------------------------
Box                            6235065         4159613     66.71
Cone/Cylinder                  8313420         1677537     20.18
CSG Intersection              30482540         1396883      4.58
CSG Union                     11084560          532510      4.80
Sphere                        13855700         2036300     14.70
True Type Font                16626840          478098      2.88
----------------------------------------------------------------------------
Calls to Noise:                  0   Calls to DNoise:         113900
----------------------------------------------------------------------------
Shadow Ray Tests:          3442452   Succeeded:                32137
Reflected Rays:             183410
----------------------------------------------------------------------------
Smallest Alloc:                 32 bytes   Largest:            20508
Peak memory used:          1089009 bytes
----------------------------------------------------------------------------
Time For Trace:    0 hours 10 minutes   5.0 seconds (605 seconds)
    Total Time:    0 hours 10 minutes   5.0 seconds (605 seconds)
[beowulf@node8 povray31]$ exit
Script done on Sat Nov 24 02:33:58 2001
```

Poolballs on 4 nodes

```
Script started on Sat Nov 24 02:12:05 2001
[beowulf@node8 povray31]$ mpirun n0 n0-3 ./mpi-x-povray -i demo.pov -w1024 -h768 _ Persistence
of Vision(tm) Ray Tracer Version 3.1g
  This is an unofficial version compiled by:
  Leon Verrall (leon@sgi.com)
  The POV-Ray Team(tm) is not responsible for supporting this version.
Copyright 1999 POV-Ray Team(tm)
Initializing MPI-POVRAY

Parsing (Slave PE 1)..
Parsing (Slave PE 2)
Parsing (Slave PE 3).......
Parsing (Slave PE 4).........
demo.pov:26: warning: All #version and #declares of float, vector, and color require semi-
colon ';' at end.
13_Pball.inc:100: warning: All #version and #declares of float, vector, and color require
semi-colon ';' at end.
No pigment type given.
 Rendering...(Slave PE 1)

Creating bounding slabs.
Scene contains 13 frame level objects; 0 infinite.
No pigment type given.

Creating bounding slabs.
 Rendering...(Slave PE 2)
..
Scene contains 13 frame level objects; 0 infinite.

 Rendering...(Slave PE 4)
```

```
......13_Pball.inc:100: warning: All #version and #declares of float, vector, and color
require semi-colon ';' at end.

13_Pball.inc:101: warning: All #version and #declares of float, vector, and color require
semi-colon ';' at end.

No pigment type given.
......No pigment type given.
No pigment type given.
No pigment type given.
......No pigment type given.


Creating bounding slabs.......
Scene contains 13 frame level objects; 0 infinite.

 Rendering...(Slave PE 3)
....All blocks are assigned. Stopping PE 3.

Done Tracing
Slave PE 3 has exited. 3 minions left...
....All blocks are assigned. Stopping PE 4.

Done Tracing
Slave PE 4 has exited. 2 minions left...
..........All blocks are assigned. Stopping PE 2.

Done Tracing
Slave PE 2 has exited. 1 minions left...
.................All blocks are assigned. Stopping PE 1.

Done Tracing
Slave PE 1 has exited. 0 minions left...
All slave tasks have exited!


PE Distribution Statistics:
            Slave PE  [ done ]            Slave PE  [ done ]
                   1  [22.92%]
                   2  [25.78%]                   3  [25.52%]
                   4  [25.78%]

POV-Ray statistics for finished frames:
demo.pov Statistics (Partial Image Rendered), Resolution 1024 x 768
----------------------------------------------------------------------------
Pixels:          180224   Samples:          180224   Smpls/Pxl: 1.00
Rays:            315611   Saved:               966   Max Level: 0/5
----------------------------------------------------------------------------
Ray->Shape Intersection          Tests       Succeeded  Percentage
----------------------------------------------------------------------------
Box                            4586931         3050974     66.51
Cone/Cylinder                  6115908         1210434     19.79
CSG Intersection              22424996         1030496      4.60
CSG Union                      8154544          391033      4.80
Sphere                        10193180         1511995     14.83
True Type Font                12231816          339630      2.78
----------------------------------------------------------------------------
Calls to Noise:                      0   Calls to DNoise:          82219
----------------------------------------------------------------------------
Shadow Ray Tests:          2535182   Succeeded:                23202
Reflected Rays:             135387
----------------------------------------------------------------------------
Smallest Alloc:                 32 bytes   Largest:            20508
Peak memory used:          1089029 bytes
----------------------------------------------------------------------------
Time For Trace:    0 hours  7 minutes  37.0 seconds (457 seconds)
    Total Time:    0 hours  7 minutes  37.0 seconds (457 seconds)
[beowulf@node8 povray31]$ exit
Script done on Sat Nov 24 02:21:03 2001
```

Poolballs on 5 nodes

```
Script started on Sat Nov 24 02:04:32 2001
[beowulf@node8 povray31]$ mpirui__[Kn n0 n0-4 ./mpi-x-run__[K__[K__[Kpovray -i demo.pov -w1024
-h768 _ Persistence of Vision(tm) Ray Tracer Version 3.1g
  This is an unofficial version compiled by:
  Leon Verrall (leon@sgi.com)
  The POV-Ray Team(tm) is not responsible for supporting this version.
Copyright 1999 POV-Ray Team(tm)
Initializing MPI-POVRAY

Parsing (Slave PE 1).
Parsing (Slave PE 4)..
Parsing (Slave PE 3)..
Parsing (Slave PE 2)........
Parsing (Slave PE 5).......
demo.pov:26: warning: All #version and #declares of float, vector, and color require semi-
colon ';' at end.
13_Pball.inc:98: warning: All #version and #declares of float, vector, and color require semi-
colon ';' at end.
No pigment type given.
 Rendering...(Slave PE 2)

Creating bounding slabs..No pigment type given.

Scene contains 13 frame level objects; 0 infinite.
No pigment type given.
.13_Pball.inc:100: warning: All #version and #declares of float, vector, and color require
semi-colon ';' at end.
No pigment type given.
.
 Rendering...(Slave PE 4)
Creating bounding slabs..
Scene contains 13 frame level objects; 0 infinite.
.No pigment type given.


 Rendering...(Slave PE 5)

Creating bounding slabs.........
Scene contains 13 frame level objects; 0 infinite.

 Rendering...(Slave PE 3)
.........................No pigment type given.
Creating bounding slabs.
Scene contains 13 frame level objects; 0 infinite.

 Rendering...(Slave PE 1)
...All blocks are assigned. Stopping PE 4.

Done Tracing
Slave PE 4 has exited. 4 minions left...
...................All blocks are assigned. Stopping PE 5.

Done Tracing
Slave PE 5 has exited. 3 minions left...
.......................All blocks are assigned. Stopping PE 1.

Done Tracing
Slave PE 1 has exited. 2 minions left...
..All blocks are assigned. Stopping PE 3.
.
Done Tracing
Slave PE 3 has exited. 1 minions left...
...............All blocks are assigned. Stopping PE 2.

Done Tracing
Slave PE 2 has exited. 0 minions left...
All slave tasks have exited!


PE Distribution Statistics:
            Slave PE  [ done ]            Slave PE  [ done ]
                   1  [17.71%]
```

```
                   2  [20.44%]                      3  [20.70%]
                   4  [20.44%]                      5  [20.70%]


POV-Ray statistics for finished frames:
demo.pov Statistics (Partial Image Rendered), Resolution 1024 x 768
----------------------------------------------------------------------------
Pixels:          160768   Samples:          160768   Smpls/Pxl: 1.00
Rays:            274492   Saved:               707   Max Level: 0/5
----------------------------------------------------------------------------
Ray->Shape Intersection          Tests       Succeeded  Percentage
----------------------------------------------------------------------------
Box                            3977136         2654137     66.73
Cone/Cylinder                  5302848         1078482     20.34
CSG Intersection              19443776          884746      4.55
CSG Union                      7070464          339164      4.80
Sphere                         8838080         1267375     14.34
True Type Font                10605696          305770      2.88
----------------------------------------------------------------------------
Calls to Noise:                  0   Calls to DNoise:          73461
----------------------------------------------------------------------------
Shadow Ray Tests:          2185547   Succeeded:                18978
Reflected Rays:             113724
----------------------------------------------------------------------------
Smallest Alloc:                 32 bytes   Largest:            20508
Peak memory used:          1089049 bytes
----------------------------------------------------------------------------
Time For Trace:    0 hours  6 minutes   4.0 seconds (364 seconds)
    Total Time:    0 hours  6 minutes   4.0 seconds (364 seconds)
[beowulf@node8 povray31]$ exit
Script done on Sat Nov 24 02:11:57 2001
```

Poolballs on 6 nodes

```
Script started on Sat Nov 24 01:57:28 2001
[beowulf@node8 povray31]$ mpirun n0 n0-5 ./mpi-x-povray -i demo.pov -w1024 -h 76 _8
_[A_[79C__7688
_[K_[A_[77C768
 Persistence of Vision(tm) Ray Tracer Version 3.1g
  This is an unofficial version compiled by:
  Leon Verrall (leon@sgi.com)
  The POV-Ray Team(tm) is not responsible for supporting this version.
Copyright 1999 POV-Ray Team(tm)
Initializing MPI-POVRAY

Parsing (Slave PE 1).
Parsing (Slave PE 4)
Parsing (Slave PE 5)....
Parsing (Slave PE 6)....
Parsing (Slave PE 3)......
Parsing (Slave PE 2)............
demo.pov:26: warning: All #version and #declares of float, vector, and color require semi-
colon ';' at end.
13_Pball.inc:98: warning: All #version and #declares of float, vector, and color require semi-
colon ';' at end.
No pigment type given.
Creating bounding slabs.
Scene contains 13 frame level objects; 0 infinite.

 Rendering...(Slave PE 2)
Creating bounding slabs.
Scene contains 13 frame level objects; 0 infinite.

Creating bounding slabs.
Scene contains 13 frame level objects; 0 infinite.
 Rendering...(Slave PE 6)
 Rendering...(Slave PE 4)
Creating bounding slabs........
Scene contains 13 frame level objects; 0 infinite.

 Rendering...(Slave PE 5)
```

```
...All blocks are assigned. Stopping PE 6.

Done Tracing
Slave PE 6 has exited. 5 minions left...
.............................All blocks are assigned. Stopping PE 3.
All blocks are assigned. Stopping PE 2.

Done Tracing
Done Tracing
Slave PE 2 has exited. 4 minions left...
.
Slave PE 3 has exited. 3 minions left...
.....................All blocks are assigned. Stopping PE 4.

Done Tracing
Slave PE 4 has exited. 2 minions left...
....................All blocks are assigned. Stopping PE 5.

Done TracingAll blocks are assigned. Stopping PE 1.

Slave PE 5 has exited. 1 minions left...

Done Tracing
Slave PE 1 has exited. 0 minions left...
All slave tasks have exited!


PE Distribution Statistics:
            Slave PE  [ done ]            Slave PE  [ done ]
                   1  [12.89%]
                   2  [16.93%]                   3  [17.32%]
                   4  [17.84%]                   5  [17.58%]
                   6  [17.45%]

POV-Ray statistics for finished frames:
demo.pov Statistics (Partial Image Rendered), Resolution 1024 x 768
----------------------------------------------------------------------------
Pixels:          101376   Samples:          101376   Smpls/Pxl: 1.00
Rays:            177122   Saved:               474   Max Level: 0/5
----------------------------------------------------------------------------
Ray->Shape Intersection          Tests       Succeeded  Percentage
----------------------------------------------------------------------------
Box                            2611044         1726022     66.10
Cone/Cylinder                  3481392          731658     21.02
CSG Intersection              12765104          598613      4.69
CSG Union                      4641856          232867      5.02
Sphere                         5802320          837760     14.44
True Type Font                 6962784          212564      3.05
----------------------------------------------------------------------------
Calls to Noise:                      0   Calls to DNoise:          50314
----------------------------------------------------------------------------
Shadow Ray Tests:          1475084   Succeeded:                12732
Reflected Rays:              75746
----------------------------------------------------------------------------
Smallest Alloc:                 32 bytes   Largest:            20508
Peak memory used:          1252973 bytes
----------------------------------------------------------------------------
Time For Trace:    0 hours  5 minutes   9.0 seconds (309 seconds)
    Total Time:    0 hours  5 minutes   9.0 seconds (309 seconds)
[beowulf@node8 povray31]$ exit
```

Script done on Sat Nov 24 02:03:56 2001

Poolballs on 7 nodes

```
Script started on Sat Nov 24 01:51:13 2001
[beowulf@node8 povray31]$ mpirun n0 n0-6 ./mpi-x-povray -i demo.pov -w1024 -h768 _ Persistence
of Vision(tm) Ray Tracer Version 3.1g
  This is an unofficial version compiled by:
  Leon Verrall (leon@sgi.com)
  The POV-Ray Team(tm) is not responsible for supporting this version.
Copyright 1999 POV-Ray Team(tm)
Initializing MPI-POVRAY

Parsing (Slave PE 2)
Parsing (Slave PE 7).
Parsing (Slave PE 1)...
Parsing (Slave PE 4)..
Parsing (Slave PE 3)...
Parsing (Slave PE 6)......
Parsing (Slave PE 5)...............
demo.pov:26: warning: All #version and #declares of float, vector, and color require semi-
colon ';' at end.
13_Pball.inc:98: warning: All #version and #declares of float, vector, and color require semi-
colon ';' at end.
No pigment type given.
Creating bounding slabs.
Scene contains 13 frame level objects; 0 infinite.
 Rendering...(Slave PE 7)
 Rendering...(Slave PE 3)
 Rendering...(Slave PE 4)
 Rendering...(Slave PE 6)
 Rendering...(Slave PE 2)
 Rendering...(Slave PE 5)
 Rendering...(Slave PE 1)
...All blocks are assigned. Stopping PE 4.

Done Tracing
Slave PE 4 has exited. 6 minions left...
..All blocks are assigned. Stopping PE 6.
..
Done Tracing.
Slave PE 6 has exited. 5 minions left...
..............All blocks are assigned. Stopping PE 3.
All blocks are assigned. Stopping PE 7.

Done Tracing
Done Tracing
Slave PE 3 has exited. 4 minions left...

Slave PE 7 has exited. 3 minions left...
....All blocks are assigned. Stopping PE 5.

Done Tracing..
Slave PE 5 has exited. 2 minions left...
All blocks are assigned. Stopping PE 2.

Done Tracing
Slave PE 2 has exited. 1 minions left...
........All blocks are assigned. Stopping PE 1.

Done Tracing
Slave PE 1 has exited. 0 minions left...
All slave tasks have exited!


PE Distribution Statistics:
            Slave PE  [ done ]            Slave PE  [ done ]
                   1  [11.98%]
                   2  [14.71%]                   3  [14.45%]
                   4  [14.84%]                   5  [14.58%]
                   6  [14.84%]                   7  [14.58%]


POV-Ray statistics for finished frames:
demo.pov Statistics (Partial Image Rendered), Resolution 1024 x 768
```

```
----------------------------------------------------------------------------
Pixels:          94208   Samples:          94208   Smpls/Pxl: 1.00
Rays:           164181   Saved:              418   Max Level: 0/5
----------------------------------------------------------------------------
Ray->Shape Intersection          Tests       Succeeded  Percentage
----------------------------------------------------------------------------
Box                            2395890        1608816     67.15
Cone/Cylinder                  3194520         656058     20.54
CSG Intersection              11713240         558802      4.77
CSG Union                      4259360         217846      5.11
Sphere                         5324200         810185     15.22
True Type Font                 6389040         207192      3.24
----------------------------------------------------------------------------
Calls to Noise:                      0   Calls to DNoise:          44546
----------------------------------------------------------------------------
Shadow Ray Tests:          1331811   Succeeded:                11470
Reflected Rays:              69973
----------------------------------------------------------------------------
Smallest Alloc:                 32 bytes   Largest:            20508
Peak memory used:          1089089 bytes
----------------------------------------------------------------------------
Time For Trace:    0 hours  4 minutes  22.0 seconds (262 seconds)
    Total Time:    0 hours  4 minutes  22.0 seconds (262 seconds)
[beowulf@node8 povray31]$ exit
Script done on Sat Nov 24 01:56:59 2001
```

Poolballs on 8 nodes

```
Script started on Sun Nov 25 18:19:13 2001
[beowulf@node8 povray31]$ mpirun ./mpi-x-povray -i demo.pov -h1024 -
h768____________[P1__[1@w1_1024 -h768 mpirun: cannot parse: Exec format error
[beowulf@node8 povray31]$ mpirun ./mpi-x-povray -i demo.pov -w1024 -
h768________________________________________[1@n.__[1@0.__[1@ .__[1@n.__[1@0.__[1@-
.__[1@7.__[1@ .__[38C8  Persistence of Vision(tm) Ray Tracer Version 3.1g
  This is an unofficial version compiled by:
  Leon Verrall (leon@sgi.com)
  The POV-Ray Team(tm) is not responsible for supporting this version.
Copyright 1999 POV-Ray Team(tm)
Initializing MPI-POVRAY

Parsing (Slave PE 1)
Parsing (Slave PE 4)
Parsing (Slave PE 2)
Parsing (Slave PE 3)
Parsing (Slave PE 6)
Parsing (Slave PE 8)
Parsing (Slave PE 7)
Parsing (Slave PE 5)....................

13_Pball.inc:101: warning: All #version and #declares of float, vector, and color require
semi-colon ';' at end.

Creating bounding slabs.No pigment type given.
No pigment type given.
No pigment type given.

Scene contains 13 frame level objects; 0 infinite.

 Rendering...(Slave PE 8)
 Rendering...(Slave PE 3)
 Rendering...(Slave PE 7)
 Rendering...(Slave PE 1)
 Rendering...(Slave PE 2)
 Rendering...(Slave PE 4)
 Rendering...(Slave PE 6)
 Rendering...(Slave PE 5)
..All blocks are assigned. Stopping PE 7.

Done Tracing
Slave PE 7 has exited. 7 minions left...
...All blocks are assigned. Stopping PE 5.
```

```
Done Tracing
Slave PE 5 has exited. 6 minions left...
.........................All blocks are assigned. Stopping PE 2.

Done Tracing..
Slave PE 2 has exited. 5 minions left...
All blocks are assigned. Stopping PE 6.
.
Done Tracing
Slave PE 6 has exited. 4 minions left...
..............................All blocks are assigned. Stopping PE 4.

Done Tracing
Slave PE 4 has exited. 3 minions left...
........All blocks are assigned. Stopping PE 3.

Done Tracing
Slave PE 3 has exited. 2 minions left...
....All blocks are assigned. Stopping PE 8.

Done Tracing
Slave PE 8 has exited. 1 minions left...
..............All blocks are assigned. Stopping PE 1.

Done Tracing
Slave PE 1 has exited. 0 minions left...
All slave tasks have exited!


PE Distribution Statistics:
            Slave PE  [ done ]            Slave PE  [ done ]
                   1  [10.55%]
                   2  [12.89%]                   3  [12.89%]
                   4  [12.63%]                   5  [12.63%]
                   6  [12.37%]                   7  [12.89%]
                   8  [13.15%]

POV-Ray statistics for finished frames:
demo.pov Statistics (Partial Image Rendered), Resolution 1024 x 768
----------------------------------------------------------------------------
Pixels:           82944   Samples:           82944   Smpls/Pxl: 1.00
Rays:            147318   Saved:               257   Max Level: 0/5
----------------------------------------------------------------------------
Ray->Shape Intersection          Tests       Succeeded  Percentage
----------------------------------------------------------------------------
Box                            2134350         1424997     66.76
Cone/Cylinder                  2845800          559332     19.65
CSG Intersection              10434600          481141      4.61
CSG Union                      3794400          183495      4.84
Sphere                         4743000          711005     14.99
True Type Font                 5691600          164144      2.88
----------------------------------------------------------------------------
Calls to Noise:                  0   Calls to DNoise:          37254
----------------------------------------------------------------------------
Shadow Ray Tests:          1171157   Succeeded:                 9252
Reflected Rays:              64374
----------------------------------------------------------------------------
Smallest Alloc:                 32 bytes   Largest:            20508
Peak memory used:          1089109 bytes
----------------------------------------------------------------------------
Time For Trace:    0 hours  3 minutes  49.0 seconds (229 seconds)
    Total Time:    0 hours  3 minutes  49.0 seconds (229 seconds)
[beowulf@node8 povray31]$ exit
Script done on Sun Nov 25 18:26:36 2001
```

Tulips on 1 node

```
Script started on Wed Nov 21 01:48:42 2001
[beowulf@node1 povray31]$ x-povray -i tulips.pov Persistence of Vision(tm) Ray Tracer Version
3.1g.Linux.gcc
  This is an official version prepared by the POV-Ray Team(tm). See the
    documentation on how to contact the authors or visit us on the
    internet at http://www.povray.org.
Copyright 1999 POV-Ray Team(tm)
Parsing Options
  Input file: tulips.pov (compatible to version 3.1)
  Remove bounds........On  Split unions........Off
  Library paths: /usr/local/lib/povray31 /usr/local/lib/povray31/include
Output Options
  Image resolution 320 by 240 (rows 1 to 240, columns 1 to 320).
  Output file: tulips.png, 24 bpp PNG
  Graphic display.....Off
  Mosaic preview......Off
  CPU usage histogram.Off
  Continued trace.....Off  Allow interruption...On  Pause when done.....Off
  Verbose messages....Off
Tracing Options
  Quality:  9
  Bounding boxes.......On  Bounding threshold: 25
  Light Buffer.........On  Vista Buffer.........On
  Antialiasing........Off
  Radiosity...........Off
Animation Options
  Clock value....   0.000  (Animation off)
Redirecting Options
  All Streams to console..........On
  Debug Stream to console.........On
  Fatal Stream to console.........On
  Render Stream to console........On
  Statistics Stream to console....On
  Warning Stream to console.......On


Parsing.......
Focal blur is used. Standard antialiasing is switched off.

Creating bounding slabs.
Scene contains 22011 frame level objects; 1 infinite.

Creating light buffers.......
Rendering... Done Tracing
tulips.pov Statistics, Resolution 320 x 240
----------------------------------------------------------------------------
Pixels:           76800   Samples:        19660800   Smpls/Pxl: 256.00
Rays:          19660800   Saved:                 0   Max Level: 1/5
----------------------------------------------------------------------------
Ray->Shape Intersection          Tests       Succeeded  Percentage
----------------------------------------------------------------------------
Blob                        1275777783       298321419     23.38
Blob Component              2381289467      2259449386     94.88
Blob Bound                  2551555566      2381289467     93.33
Box                        10251883237      2972673609     29.00
Cone/Cylinder                 53735942         2517948      4.69
CSG Intersection            8516484938       285092797      3.35
Plane                       8445288949      3616358569     42.82
Torus                       2432060743       789401935     32.46
Torus Bound                 2432060743       899809497     37.00
Bounding Object             8909077334      1863108116     20.91
Bounding Box               58535570831     11813741448     20.18
Light Buffer              206031501066     87812364509     42.62
----------------------------------------------------------------------------
Roots tested:          1684034445   eliminated:          448286134
Calls to Noise:          25936097   Calls to DNoise:      94313101
----------------------------------------------------------------------------
Shadow Ray Tests:      1288804564   Succeeded:           195215845
----------------------------------------------------------------------------
Smallest Alloc:                11 bytes   Largest:           176088
Peak memory used:       125446732 bytes
```

Copyright © 2003

```
-------------------------------------------------------------------------
Time For Parse:    0 hours  1 minutes   5.0 seconds (65 seconds)
Time For Trace:   77 hours 33 minutes  19.0 seconds (279199 seconds)
    Total Time:   77 hours 34 minutes  24.0 seconds (279264 seconds)
[beowulf@node1 povray31]$ exit
Script done on Sun Nov 25 17:40:58 2001
```

End of Document