

PARALLEL COMPUTING USING LINUX CLUSTERS

PFRACT – A PARALLEL FRACTAL GENERATION PROGRAM

Srivas N. Chennu

(**Addr:** No 62, 3rd Main, SBM Colony, Anandnagar, Bangalore-24.

Ph: 3337809, **E-mail:** srivas@softhome.net)

Vishwas N.

(**Addr:** No 6, 2ND Cross, Dinnur main, Akkamma Block, R.T.Nagar, Bangalore-32.

Ph: 3433205, **E-mail:** vishwas_n@hotmail.com)

8th Semester

Department of Computer Science

R.V.College of Engineering

Bangalore-560059

PARALLEL COMPUTING USING LINUX CLUSTERS

PFRACT – A PARALLEL FRACTAL GENERATION PROGRAM

ABSTRACT

Parallel computing is a processing paradigm that involves exploiting the parallelism inherent in computer programs. The major goal is to improve the performance of computer systems by utilizing their processing capabilities so as to gain maximum throughput.

Cluster computing in Linux involves a collection of conventional Linux machines connected by a fast network interconnect, working in parallel in order to speed up the process of finding solutions to large, complex parallel computing problems. The many advantages offered by the Linux Operating System in terms of flexibility and reliability can be efficiently exploited to provide a full-fledged, off-the-shelf parallel computing platform.

This paper briefly traces the evolution of parallel computing paradigms and architectures before

delving into the issues involved in developing parallel applications for Linux Clusters. The paper then elaborates on the development of a parallel computing application designed and implemented by the authors, called PFract. PFract is a parallel fractal generation system, which speeds up the computationally intensive task of drawing fractal images. It parallelizes the fractal generation algorithm and executes it concurrently on a Linux Cluster. This parallel application is used as an example to elucidate the process of parallelization of conventional applications for execution on cluster architectures.

The authors have also developed a generic software architecture for deploying parallelized applications on clusters. The PFract system implements this architecture and has been used to test its capabilities compile performance statistics included in this paper.

Srivas N. Chennu

Vishwas N.

8th Semester CSE

R.V.C.E. Bangalore

PARALLEL COMPUTING USING LINUX CLUSTERS

1 INTRODUCTION

Parallel computing is a processing paradigm that involves exploiting the parallelism in computer programs. The major goal is to improve the performance of computer systems by utilizing their processing capabilities so as to gain maximum throughput.

Some of the important trends that have fuelled the growth of parallel computing are -

- Heavy computing demands from large-scale applications – Computing applications from the various fields listed below have driven the need for massive parallel processing.
 - Scientific problems in Astrophysics, Molecular Chemistry, Biotechnology, Material Sciences, Earth Sciences etc.
 - Engineering applications from the domains of Mechanical, Petroleum, Aeronautics, Automotive technology, Pharmaceuticals etc.

- Commercial applications involving Online Transaction Processing (OLTP) Systems.
- Emergence of low cost electronic microprocessing hardware – High speed RISC computing architectures with high price-performance ratios in the last decade have led to emphasis on utilization of distributed computing resources rather than centralized processing pools.
- Availability of high-speed network architectures – High-speed LAN architectures with bandwidths over 100 Mbps provide excellent network throughput, with very low latencies.
- Development of parallelized computational algorithms – Large amount of development work has been directed towards the development of efficient parallelization of traditional sequential algorithms.

2 PARALLEL COMPUTING ARCHITECTURES

Parallel computing architectures can be broadly classified into the following taxonomy with respect to the programming model employed.

SIMD – Single Instruction Stream - Multiple Data Stream architectures perform a single set of operations on a parallel set of data streams. Parallelized matrix algorithms deploy the SIMD model to speed up matrix calculations. SIMD are synonymous with Data Parallel architectures.

MIMD – Multiple Instruction Stream – Multiple Data Stream architectures feed separate data streams to different parallel instruction streams. Synchronization between the executions of the streams is a key issue in this model. MIMD models can be further classified as:

Shared Memory Architectures – Multiple instructions streams execute in a tightly coupled system accessing a common, globally shared memory region for implicit communication and synchronization. Parallel programs are usually implemented as a collection of cooperating threads that execute synchronously on different processors.

Message Passing Architectures – Every instruction stream operates on a private memory area. Communication and synchronization is achieved through explicit message interchanges.

NUMA Architectures – Non Uniform Memory architectures are a hybrid of message-based and shared memory systems. NUMA systems incorporate an asymmetric 2-level memory hierarchy. The memory system seen by the

programmer is a logically shared, but a physically distributed one. Latency of access to memory areas non-local to a processor are higher than the local one.

Systolic Architectures – Systolic architectures consisting of a 2-dimensional matrix pipeline of computing nodes operating in rhythmic sequence, with input and output of data occurring at the array boundaries.

3 CLUSTER COMPUTING USING LINUX

Cluster computing in Linux involves a collection of Linux boxes connected by a network, working in parallel in order to speed up the process of finding solutions to large, complex parallel computing problems. A Linux cluster typically consists of a single master node that controls all the other slave nodes. The master initializes all the slave nodes, and sends the computational tasks, which are computed at the nodes. The master then collects the results from the slave nodes and merges them into the resulting output.

Linux Clusters differ in some ways from conventional networks of workstations. First of all, in most cases client nodes in a Linux cluster do not

have keyboards, mice, video cards or monitors. All access to the slave 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 or for machines outside the cluster to access client nodes directly, it is a common practice for the client nodes to use private IP addresses. Usually the only machine that is also connected to the outside world using a second network card is the server node. 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 personal workstation. Once on the server node, users can edit and compile their code, and also spawn jobs on all nodes in the cluster.

Linux clusters offer a highly scalable parallel computing architecture. Computing power in the individual nodes can be kept at moderate levels. Parallel software architectures operate on the cluster hardware and provide a platform for a programmer to split up a program into parallelized tasks and efficiently distribute them among the machines in the cluster. Additional computing resources can be added to the cluster as and when required, with minimal changes to the existing system. These features make Linux Clusters ideal for large scale parallel computing.

4 THE PFRACT SYSTEM

PFract – short for Parallel Fractal Generator is a parallel computing initiative using Linux clusters. PFract is a parallel processing application developed by the authors and is used in this paper as a representative system to delineate the process of parallelization of sequential programs for Linux Clusters. PFract employs a generic parallelization engine to draw complex fractal images. The computations involved in generating the fractal images are executed in parallel on the nodes in the Linux cluster.

4.1 INTRODUCTION TO FRACTALS

Fractals are graphical shapes that are produced by the iterative application of a specific mathematical function to points in an n-dimensional space. This special type of equation is typically a feedback equation, in which the result of iteration is used as the input to the next.

Fractal images display properties of self-similarity at every level of scaling. This means that sections of the fractal image can be magnified repeatedly to see structures that are very similar to the parent fractal. Additionally fractal boundaries

have infinite length enclosed in a finite area of space. This means that the measurement of the exact length of a boundary of a fractal is impossible.

Fractals are called so because they have non-integral dimensions. Fractal structures have many instances in nature. Typical examples of fractal structures occurring in natural systems are fern leaves, Lorenz attractors etc.

The Mandelbrot and Julia sets are some of the commonly known synthetic fractal images. These fractals are generated by means of a simple iterative equation that identifies a set of points that belong to the fractal image. The iterations are applied to each point in a two dimensional complex plane space repeatedly. Depending on the results generated by the iterations, the point is either included or excluded from the set. The numbers of iterations that are executed for each point determine the depth and clarity of the resulting fractal image.

The computation of fractal images is a processor intensive task. Hence, it is an ideal candidate for parallel computation. Additionally, it lends itself well to parallelization, due to the concurrency inherent in the algorithm used to generate fractals. The computation of any fractal point is independent of any other point. Hence, the iterative application of the fractal equation to individual points in the image can be done in parallel at different nodes in the

cluster. This greatly increases the speed of generation of the images, even when a large number of iterations are being executed. Parallelization thus enables generation of fractals of high clarity and depth in reasonable amount of time.

The basic fractal generation algorithm for the Mandelbrot set is detailed below.

Step 1 – Initialize the fractal plane as 2-Dimensional plane of complex numbers of the form $(X + iY)$.

Step 2 – Consider a point $(X + iY)$ in the fractal plane. Initialize $C = X + iY$.

Step 3 – Iteratively compute the feedback equation $Z_{n+1} = Z_n^2 + C$ where $Z_0 = 0$.

Step 4 – If the value of Z rapidly increases towards infinity, the point $(X + iY)$ does not belong to the fractal set. In this case, the Z value moves further and further away from the center of the plane. On the other hand, if the value of Z converges to a single point or a finite set of points, it belongs to the set.

Step 5 – Mark the point in the fractal plane, using a color-coding scheme that corresponds to the iteration count for the point.

Step 6 – Repeat above steps for every point in the current fractal region.

5 PFRACT ARCHITECTURE

The diagram below illustrates the master-slave architecture of a parallel cluster system. This architecture deviates from the conventional client-server design. It involves a single master client running on a control node, which controls the operation of slave servers running on multiple compute nodes in the cluster.

The components of this 3-layer parallel architecture are listed below.

- The User Interface
- The Parallelization Engine
- The Communication Manager

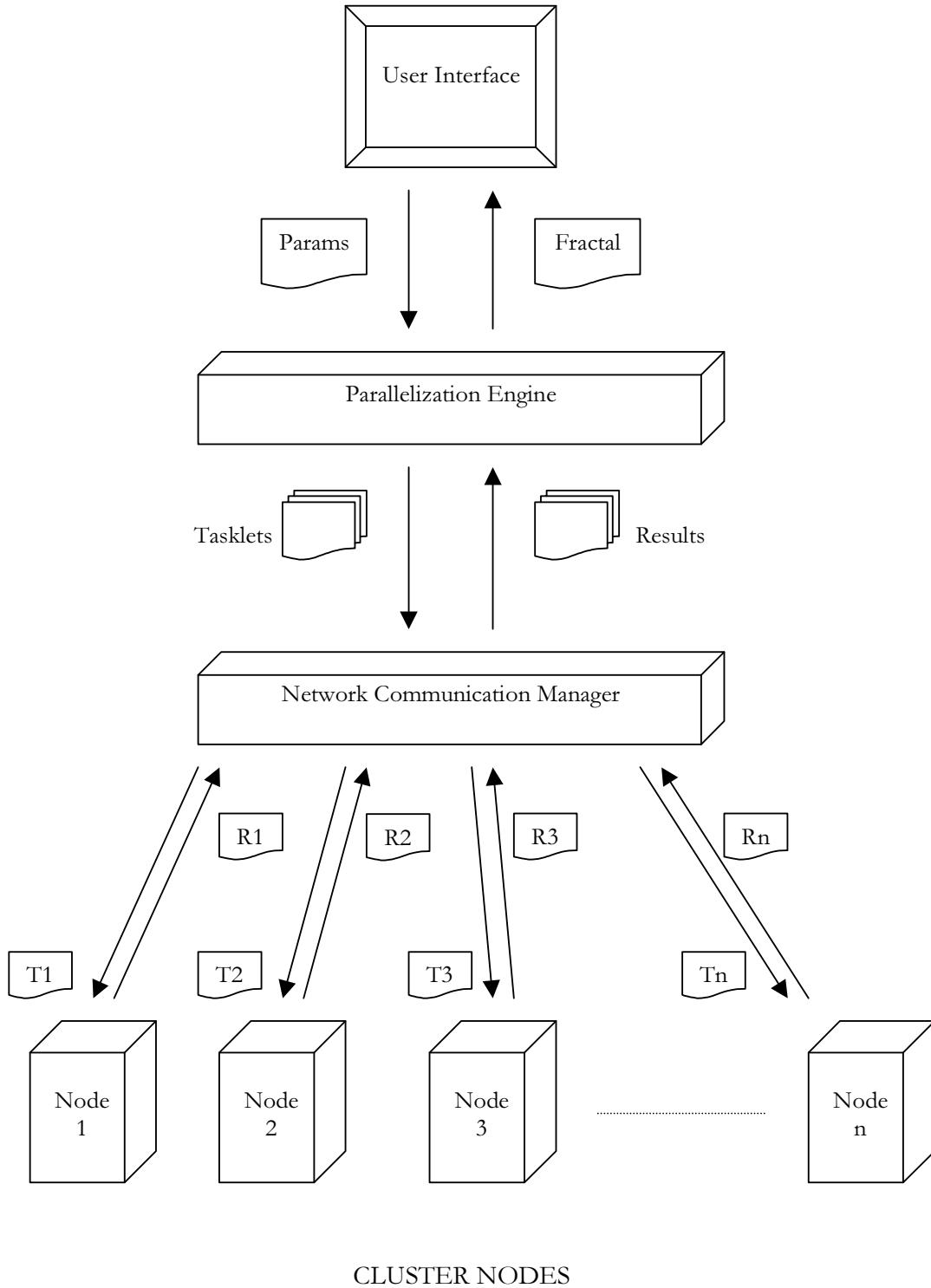
The design and function of these layers is detailed in the sections below. The parallelization of the example PFract fractal generation application is also elaborated alongside as an illustration of the process.

5.1 THE USER INTERFACE

The user interface is the topmost layer in the 3-layer parallel architecture. It provides the user with a front-end tool to control the execution parameters of the parallel program. Specifically the following variants can be manipulated, and their effects on the throughput of the system can be measured quantitatively.

- The number of active nodes - The total number of compute nodes can be varied to study the effects of machine scaling on the time for computation.
- The problem size - The fractal parameters, including the clarity, depth and screen position of the image can be altered.
- Quantitative performance estimates – The user interface provides the user with an estimate of the overall computation time for the current problem size on the current cluster configuration, including parallelization overhead and actual computing time.
- The image magnification scale.
- Cluster status and load monitoring.

PFRACT ARCHITECTURE



5.2 THE PARALLELIZATION ENGINE

Parallelization is the process of identifying and distributing the inherently parallel sections of a program to individual processing elements in a parallel architecture. Additionally, the results computed by the individual nodes have to be collected and merged to produce the final output. The 2nd layer in the 3-layer architecture, the Parallelization Engine, performs these operations. The important steps involved in parallelizing an algorithm are detailed and applied with respect to parallelizing the fractal generation algorithm given above.

5.2.1 Decomposition

In this step, the aim is to expose the concurrency inherent in the program. Such concurrent sections ideally do not have any interdependencies, and thus can be executed in parallel to achieve speedup. In this first step, the sequential algorithm is divided up into computational units called tasks.

The PFract system identifies the computation for every point on a single scanline in the fractal image as a single task. The iterations performed for all the points on a horizontal scanline to determine their inclusion the fractal set is independent of any other line. Moreover, computation for all points on a scanline requires only the y-coordinate of the line as

input. This feature makes parallel computation of scanlines at the computing nodes particularly efficient.

5.2.2 Assignment

Individual computational tasks identified during computation are assigned to compute processes that execute at the individual processing elements in the cluster. The primary goals of an ideal task-to-process assignment are:

- Workload on every processing node is balanced.
- Communication volume between processes is kept at a minimum.
- Optimum throughput is obtained.

The PFract system implements a dynamic assignment mechanism. Processing tasks are assigned to compute processes at the nodes on a first-come-first-served basis. This system ensures that tasks are sent to the first available free processor on any slave node with minimum scheduling latency. Such a dynamic assignment policy offers the strategic advantage of efficient load balancing over a static assignment policy.

5.2.3 Orchestration

Orchestration is the process of managing the parallel execution of the processes at the compute nodes. Specifically, the following goals of orchestration can be identified.

- Manage synchronization and communication mechanisms between remote processes ensuring minimal overhead of parallelization.
- Minimize artifactual communication by exploiting data and network locality.
- Manage execution to ensure a topological ordering between the processes to satisfy data and control dependencies.

The PFract communication protocol discussed below provides explicit mechanisms for inter-process communication and synchronization. Processes communicate using message-based mechanism that provides implicit support for point-to-point synchronization. Processes synchronize their execution with the control process executing at the master using blocking message primitives.

5.2.4 Mapping

Mapping is the final step where processes are scheduled for execution at the processing nodes.

This involves choosing a configuration such that related tasks are scheduled on the same processor, and locality in the network topology is exploited.

PFract implements two distinct mappings of processes to processors:

- Control Process – The control process executes at the master node in the cluster and sends commands to slave processes, instructed them to compute specified tasks on the local processor.
- Compute Processes – The compute processes execute at the slave nodes in the cluster and accept commands from the control process, consisting of parameters of the task to be scheduled for execution on the slave processor.

5.2.5 Result Collection

The results generated by the slave processes are collected by the control process for formatting and display. The slave processes send the computed results to the control process on completion of the computation. Result accumulation at the control process is done on a first-come first-served basis. As and when results arrive at the master, they are displayed to the user. High throughput is achieved by this method, since it accounts for the fact that

differential speed of computation at the slave nodes may cause varying delays.

5.2.6 Formatting and Display

In this final step, the computed iteration values for the points in the current fractal region are displayed to the user as a full-color fractal image. PFract uses the Qt and KDE user interface libraries available in Linux for displaying the fractal to the user and providing dialogs for configuration options.

5.3 THE PARALLELIZED ALGORITHM

The parallelized version of the fractal generation algorithm after application of the above steps, as employed by PFract, is given below. It parallelizes the concurrent parts of the algorithm, executes them in the processing nodes of the cluster, and collects the results for display.

Step 1 – Initialize the fractal plane as 2-Dimensional plane of complex numbers of the form $(X + iY)$.

Step 2 - Let N be the set of nodes in the cluster having a total of C computing nodes. Initialize a Boolean array $Used[C]$ of FALSE values.

Step 3 – Consider a line in the complex plane with the equation $Y = Y_i$.

Step 4 – Select a Node N_j such that $Used[j] = FALSE$. Send the Y_i coordinate of the line to a processing node N_j . Set $Used[j] = TRUE$.

Step 5 – In the node N_j , compute the iteration counts for all the points lying on the line. Return the computed values to the master node.

Step 6 – If no unused node (with $Used[j] = FALSE$) can be found, obtain computed results on a First-Come-First-Serve basis from the computing nodes.

Step 7 – If node N_j has finished computing, store the results and set $Used[j] = FALSE$.

Step 8 – Display the obtained results using a color-coding scheme that corresponds to the iteration count for the points.

Step 9 – Repeat the above steps for every line in the current fractal region.

5.4 COMMUNICATION MANAGER

The network communication manager is the lowest layer in the 3-layer parallel architecture. The communication manager provides mechanisms for communication and synchronization between the cluster nodes. The communication layer in the cluster manages point-to-point communication between the control and compute nodes in the cluster. Communication and implicit synchronization are implemented by a specially designed cluster communication protocol detailed below.

5.4.1 The Parallel Cluster Orchestration Protocol (PCOP)

PCOP is a simple yet efficient protocol implemented for coordinating of the communication between the nodes of a Linux cluster system. Some of the key services and features offered by PCOP to parallel applications are as below.

- PCOP is a reliable, connection-oriented protocol.
- The protocol implements the master - slave architecture.

- The protocol provides for synchronous (blocking) and asynchronous (non-blocking) communication.
- The protocol provides for command and reply based message interaction.
- The protocol uses TCP based sockets.
- The protocol provides for error handling and reporting.
- PCOP provides cluster-monitoring functions to view system status of the slave nodes and diagnose workload imbalance problems.

5.4.2 The PCOP API

PCOP provides two basic primitives, **send** and **receive** to the users of the service. Importantly, the PCOP API abstracts over the services provided by the underlying TCP layer to provide operations specialized for management of clusters. The list of service primitives provided by PCOP is given below.

- **INITIALIZE** – Initializes all the currently active compute processes to a default state.
- **DECOMP** – Segregates the tasks allocated to a process.

- **REDUCE** – Collects and tags results generated at the slave nodes.
- **BROADCAST** – Sends a broadcast message to every active compute node in the cluster.
- **SEND** – The basic SEND MESSAGE primitive provided for sending variable-sized messages to specified nodes.
- **RECEIVE** – The basic RECEIVE MESSAGE primitive provided for receiving variable-sized messages from specified nodes.
- **DISCONNECT** – Indicates completion of operations to slave nodes.

5.4.3 PCOP Implementation

PCOP uses a tagged message interaction system in which all message boundaries are preserved. Messages contain headers that are used to distinguish between the different types of messages.

PFract Message Formats – The generic Message format used in the PFract system is shown below.

Header	Destination Node	Source Node	Data Length	Message Data
--------	------------------	-------------	-------------	--------------

PFract Generic Message Format

The different message types recognized in the PFract parallelized fractal generation system are as follows.

MFractType – This header instructs the server to set the current fractal type, to that specified in the **Fractal Type** field of the message.

MFractType	Fractal Type
------------	--------------

MFractInit – This header instructs the server to initialize a new fractal object of the predetermined type. The message data contains the initialization parameters.

MFractInit	Fractal Parameters
------------	--------------------

MFractCalc – This header instructs the server to calculate the iteration counts for all points on a specified line ($Y = Y_i$). The message data contains the Y_i value to be used. The pre-initialized is automatically used for the computation.

MFractCalc	Fractal Type
------------	--------------

MDone – This header tells the server that the client has finished computation and is exiting.

MDone	Empty
--------------	--------------

MUnknown - This header tells the client that some error has occurred, or an implemented header type has been used.

MUnknown	Empty
-----------------	--------------

MStat - This header sends information about the slave node to the controller. The status information typically contains the current system load, free memory, swap etc.

MStat	Status Information
--------------	---------------------------

6 PFRRACT FEATURES

6.1 OBJECT ORIENTED DESIGN AND IMPLEMENTATION

PFract provides its functionality through the cooperation of a collection of interdependent class structures.

6.2 GENERIC ARCHITECTURE

The Parallelization Engine and Network Manager are designed to be independent of the application to be parallelized. These software components can be deployed in the parallelization of any other application also. The core parallel computing engine in PFract has also been used in applications involving parallelization of large prime number generation and factorization.

6.3 DYNAMIC NODE CONFIGURATION OPTIONS

Compute nodes can be added during operation to the current group of active nodes. This allows the user to observe the performance variation as nodes are added and deleted.

6.4 MULTITHREADED OPERATION

Server processes are multithreaded for efficient management of computing and communication tasks.

6.5 FAULT TOLERANCE

In the event of failure of one or more slave nodes, the master dynamically readjusts its configuration so

as to reflect the change in status of the cluster. The parallel computation process then continues using only the currently available nodes.

7 PERFORMANCE STATISTICS

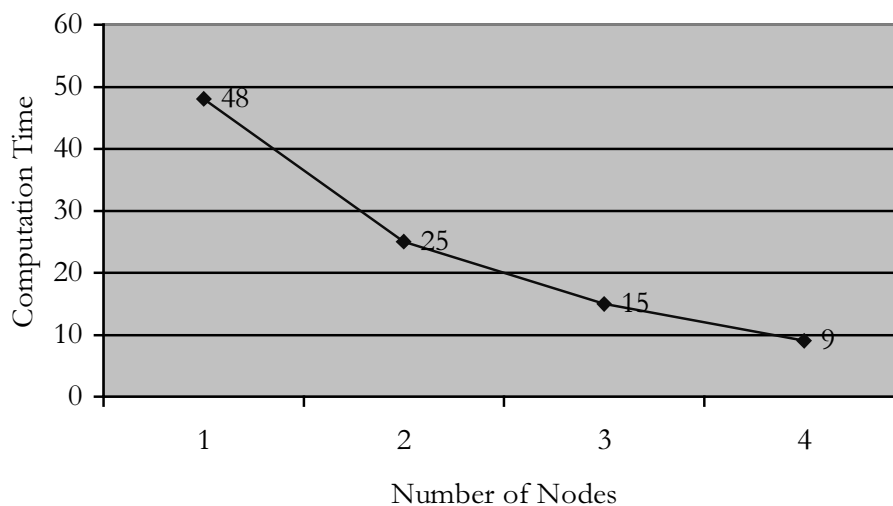
The figure below depicts a quantitative estimate of the performance variation of the PFract system. The overall fractal generation time decreases as the number of the active compute nodes in the cluster is increased, keeping the problem size i.e. the fractal parameters fixed. The problem and machine size must be simultaneously scaled to extrapolate the graph for larger cluster sizes.

8 CONCLUSION

This paper has elucidated the emerging concept of large-scale parallel computing using dedicated Linux Clusters. Their high scalability, flexibility and robustness make Linux clusters ideal solutions for large scale, off-the-shelf parallel computing.

The PFract parallel fractal generation system was successfully designed and implemented on a typical Linux Cluster. It has demonstrated the significant performance advantages parallel processing offers for solving complex computing problems. Furthermore, the PFract system has also displayed the potential offered by cluster computing using the Linux Operating System as a parallel computing platform.

PFract Performance Statistics



9 REFERENCES

- Parallel Computer Architecture – David E. Culler, Jaswinder Pal Singh and Anoop Gupta.
- Unix Network Programming – W. Richard Stevens
- Chaos Theory and Fractals – James Gleick
- www.beowulf-underground.org - Cluster computing resources and documentation
- www.pvm.org - Web resources for the Parallel Virtual Machine architecture.
- www.mpi.org - Web resources for the Message Passing Interface Architecture.
- **IP or IPv4** – Internet Protocol version 4 is Network layer protocol in the TCP/IP protocol stack. A unique 32-bit IP address identifies every host on an IP based network.
- **TCP Socket** – One endpoint of a connection between two processes. A (IP Address, TCP Port) pair uniquely identifies a TCP Socket attached to a process.
- **Complex Numbers** – Set of numbers of the form $X + iY$, where X and Y are real numbers, and
 - X is known as the real part
 - Y is known as the imaginary part.
 - $i = (-1)^{1/2}$

10 GLOSSARY

- **LAN** – Local Area Network
- **TCP** – Transmission Control Protocol is the Transport layer protocol in the TCP/IP protocol stack. A unique 16-bit TCP port identifies every TCP layer connection.
- **Synchronous Communication** – Mode of inter-process communication where sender process is blocked till the receiver process acknowledges the receipt of the message.
- **Asynchronous Communication** – Mode of inter-process communication where message transfer from sender to receiver is performed concurrent with the execution of sender and receiver processes. This method of communication offers maximum parallelism at the cost of unreliable communication.