TOWARDS A MODEL FOR ESTIMATING GRID RESOURCE FOR OIL AND GAS EXPLORATION

By

SIREESHA LANKA

Master of Science

Oklahoma State University

Stillwater, Oklahoma

2006

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
July, 2006

TOWARDS A MODEL FOR ESTIMATING GRID

RESOURCE FOR OIL AND

GAS EXPLORATION

Thesis Approved:

Dr. Venkatesh Sarangan

Dr. Xiaolin Li

Dr. Surinder K. Sahai

Dr. A. Gordon Emslie

Dean of the Graduate College

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

The seismic exploration is primary tool in search for oil and gas exploration. The results from these explorations are used to produce images of the earth's subsurface, that geologists analyze for understanding the earth subsurface picture. There are several processing techniques involved in processing data from seismic explorations. Seismic migration process is one of the most computationally intensive steps of all the seismic data processing sequence. Migration techniques are highly compute and I/O intensive and therefore require high performance systems to carry out the operations efficiently. It is well evident that rather than sequential machines parallel provide cost-effective solutions as migration algorithms show lot of potential for parallelism. One such algorithm is Kirchhoff migration. The processing of Kirchhoff migration algorithm involves handling large volumes of data which needs high computational power. Most of the exploration companies cannot afford to have computational power or the estimate of the power they need on their own. Therefore there should be a model or technique by which oil and gas exploration companies can find out an estimate for the computational power they need. This thesis gives a model for estimating the computer power needed to run Kirchhoff migration algorithm for a given volume of data.

CHAPTER I

INTRODUCTION

1.1 The Goal

In oil and gas exploration, seismic data is gathered by sending sound waves into the earth and then by listening for the responses that come back in the form of reflections from rock layers. The reflected waves are detected by receivers called geophones. These data are recorded and processed to generate 2-D seismic sections or a 3-D volume for geological interpretation. The time it takes for sound pulse to return back to the earth's surface tells us how far the pulse has traveled. Thus we can determine from where the wave had reflected off rock layers and determine the position of those layers. Then several seismic data processing techniques are applied to the raw seismic data to produce seismic sections. This data processing involves processing large volumes of data in which high computationally intensive algorithms and applications are employed. To do this the oil and gas exploration companies require high compute power. The goal of this thesis is to develop a theoretical model by which small or mid-sized oil and gas exploration companies, which do not have their own computational resources, can estimate the computational power they need to do the processing.

1.2 Significance of the goal

Seismic Data Processing has great importance in imaging earth’s geological sub-structures for exploration of oil and gas deposits. The need for more detailed imaging of underground structures has increased, in order to make well informed drilling decisions during seismic exploration. These demands for more detailed imaging with high resolution have increased the amount of data being acquired, thus leading to an increase in the computational effort, often requiring parallel computers which can handle the large volumes of data with high intensive I/O operations and computationally complex seismic algorithms. A theoretical model that can estimate the computational power required for seismic data processing will be of great help to any oil and gas exploration company in the initial resource planning and budget applications. It will also help grid vendors who sell ‘computational power’ to approximately allocate resources and develop a matching pricing structure.

1.3 The Approach:

The parallel implementations of seismic algorithms are specific to a single parallel architecture. The parallel implementation of an algorithm differs from architecture to architecture. In this attempt we try to map the actual run time values of an algorithm implemented on a specific parallel architecture with the theoretically computed complexity of algorithm.

To achieve this goal, we started with inspecting various seismic data processing techniques. The seismic migration process is identified as one of the most computationally intensive processes. Then we studied the functionality of the Kirchhoff migration algorithm which is one of the most effective algorithm in terms of cost and time. Then, the sequential implementation of the algorithm is studied and the running time complexity is computed using the Random Access Model (RAM). As the Kirchhoff migration algorithm has high potential to be parallelized we studied its parallel implementation. For Kirchhoff algorithm's parallel implementation we computed the asymptotic running time complexities, on n-processor parallel system. The actual running time values of the algorithm are taken from reference [1] which has a Kirchhoff's parallel implementation on the PARAM 10000 system. By comparing the actual values and the theoretical complexities of the algorithm a relation between these two is established. A theoretical model which takes data volume as input and gives a close approximation of computational power needed for the application in terms of processors and running time is developed.

CHAPTER II

BACKGROUND AND METHODOLOGY

2.1 Basics of Seismology

Seismic Exploration:

Seismic explorations are done to locate oil and gas deposits. Towards this goal geophysicists perform seismic experiments in which sound waves are artificially generated. These waves and passed into the earth and the time required for the waves to travel from the source to a series of geophones is measured.

Seismic sources:

Seismic sources are the sources of energy generators which are sent into the ground for seismic explorations. The most often used sources are vibrator trucks, hammer blows or an explosions for land surveys, and air guns are used for marine surveys [2]. In land surveys the vibrator trucks continuously shake the ground starting from low frequency then progressively moving towards higher frequencies. In marine surveys, air-guns compress air to produce explosive blast of air into the water surrounding the air.

Fig 2.1 Vibrator Truck

Recording Equipment:

The typical recording equipments are the geophones. These geophones are the spring-mounted electric coils moving with magnetic field, which generate electric currents in response to ground motion [3]. Typically geophones are connected in series to one another so that the summed signal is linked to one channel in multi-channel recording cable.

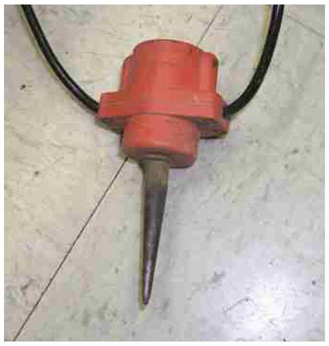


Fig 2.2 Geophone [4]

Seismic data acquisition:

In a typical seismic survey, a cable with geophones attached at regular intervals is laid out along a line or in an area. The seismic sources generate the seismic waves at regular intervals. As the source shot goes off, signals are recorded from each geophone for certain amount of time, producing series of seismic traces. Then the seismic traces are recorded on magnetic tapes in the recording truck. [5]

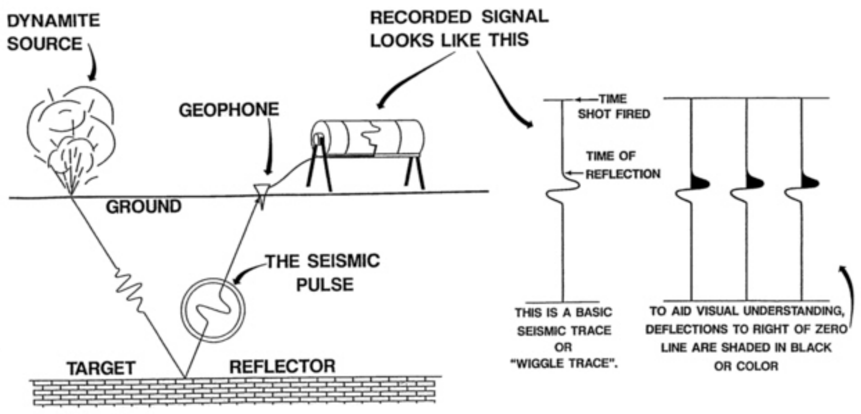

DYNAMITE SOURCE
RECORDED SIGNAL LOOKS LIKE THIS
GEOPHONE
TIME SHOT FIRED
TIME OF REFLECTION
GROUND
THE SEISMIC PULSE
TARGET
REFLECTOR
THIS IS A BASIC SEISMIC TRACE OR "WIGGLE TRACE".
TO AID VISUAL UNDERSTANDING, DEFLECTIONS TO RIGHT OF ZERO LINE ARE SHADED IN BLACK OR COLOR

Fig 2.3 Seismic data acquisition [6]

Seismic Reflections:

Reflections of sound waves from the subsurface arrive at geophones at some measurable time after the source pulse. If we know the speed of sound in the earth and the geometry of the wave path, we can convert that seismic travel time to depth. By measuring the arrival time at successive surface locations we can produce a profile, or cross-section, of seismic travel times. [7]

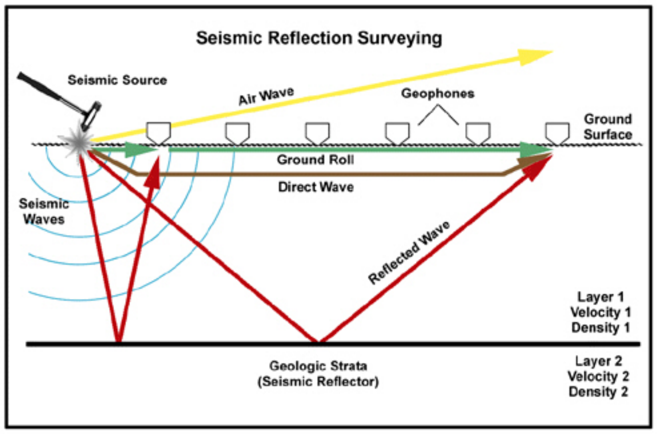


Fig 2.4 Seismic reflection surveying [8]

Seismic Refractions:

Seismic refraction involves measuring the travel time of the wave which travels down to rock surface, and refracted along the surface and return to the surface as a head wave. This happens when the wave crosses the interface between layers of two different velocities. Depending on the relative velocity of the medium, angle the wave leaving one medium is altered from the angle of incidence. Seismic refractions come into picture when the seismic velocities of layers increase with depth. So, when a wave is traveling from low-velocity layer to high-velocity layer with a particular incident angle it will be refracted along the upper surface of the lower layer. Therefore at some point of time refracted wave will overtake the direct wave. Then this refracted wave will be recorded as first arrivals to all the geophones. As a result this will yield in incorrect results. [7]

Common shot gather:

The recorded traces for a single shot can be grouped together to form a common shot gather. To cover a greater subsurface area with reflection events, the shot and geophone locations are translated by the same distance and the common short gather experiment is repeated to give another shot gather. These experiments are repeated along a line until a sufficient subsurface coverage has been achieved. [9]

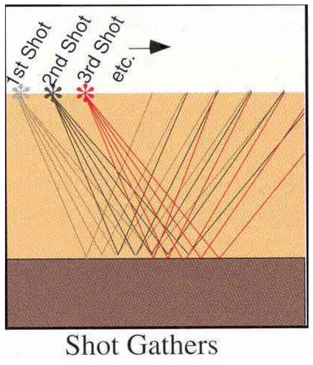


Fig 2.5 Common shot gather [9]

Common mid point gathers (CMG):

The data collected from common shot gathers are re-sorted so that they are in form of common midpoint gather. In common midpoint gathers any trace has a same source-receiver midpoint. In CMG each trace is the reflection energy that is sampled at the same

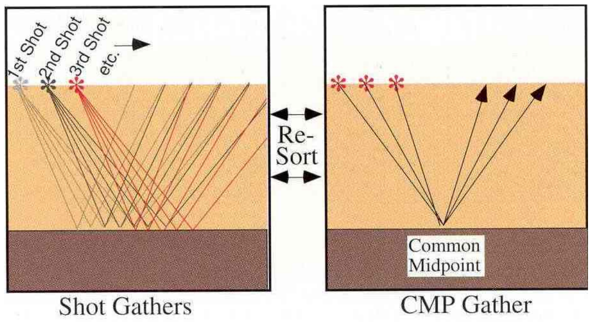


Fig 2.6 Common shot gather to Common mid point gather [9]

place of the reflector as the other traces. Common mid point (CMP) gather represent only a subset of the information in the shot gathers. The number of traces in a CMP gather defines the fold of the data. Larger the fold, larger is the redundantly sampled subsurface reflection points so after the stacking process, the stacked traces will have good signal to noise ratio. [9]

Moveout Correction:

In general, a reflection typically arrives first at the receiver nearest to the source. The offset between the source and other receivers induces a delay in the arrival time of

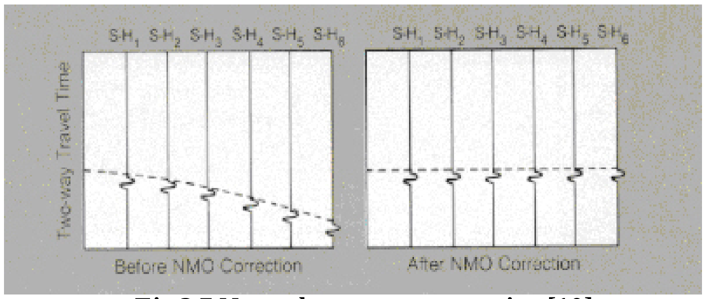


Fig 2.7 Normal move out correction [10]

reflections from horizontal subsurface [11]. A plot of arrival times versus offset has a hyperbolic shape. This is called move out and when reflectors are flat it is called Normal moveout and if reflectors are dipping then we have dip moveout in addition to the normal moveout. For normal moveout correction a function of time and offset will be used to compensate for the effects of normal moveout.

Stacking:

This is a process where traces are added together from different records to reduce noise and improve overall data quality. The number of traces that have been added together during stacking is called the fold. The traces are summed together to improve the signal-to-noise ratio, reduce noise and improve seismic data quality.

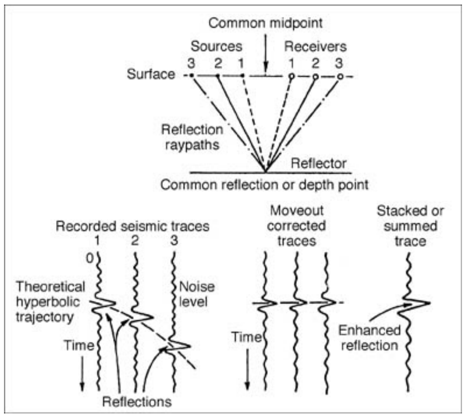


Fig 2.8 Stacking process [12]

Traces from different shot records with a common reflection point, such as common midpoint (CMP) data, are stacked to form a single trace during seismic processing. Stacking reduces the amount of data by a factor called the fold. After this process most of the coherent noise in the traces are eliminated because of the time shifts applied which will align the reflections with one another in such a way that only the primary reflections are coherently added together. [9]

2.2 Basics of Seismic data processing techniques:

Seismic data processing:

Processing of the seismic data basically involves noise suppression, signal enhancement and migration of the seismic traces to their true location in space. Processing steps typically include analysis of velocities and frequencies, static corrections, deconvolution,

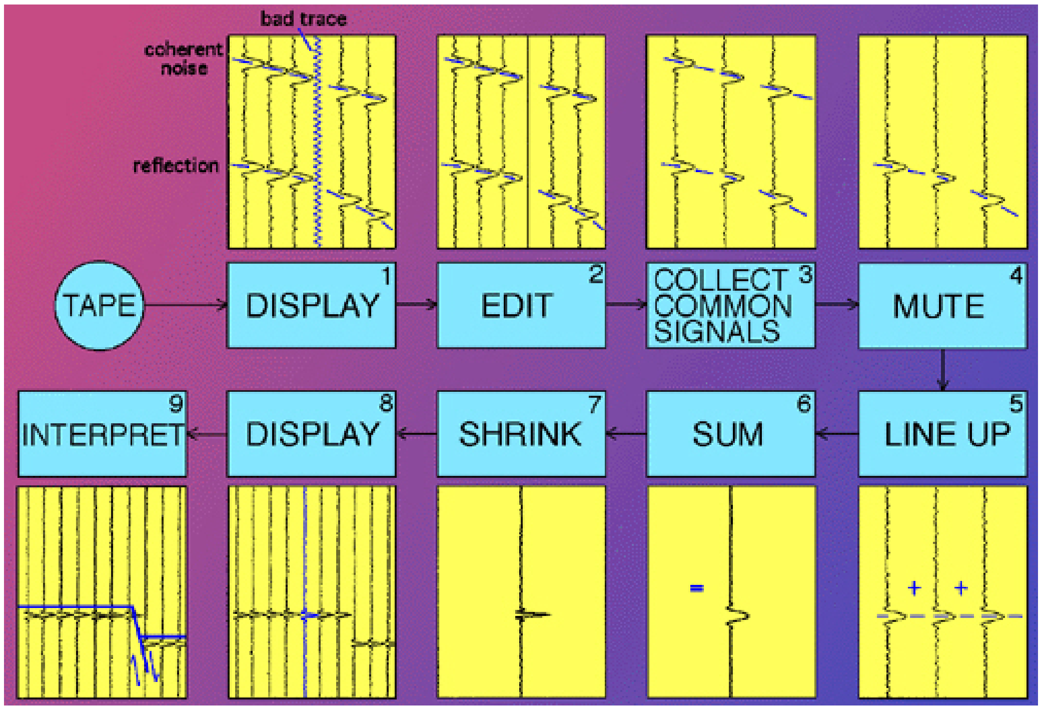


Fig 2.9 Seismic data processing [5]

normal moveout, dip moveout, stacking, and migration, which can be performed before or after stacking. Seismic processing facilitates better interpretation because subsurface structures and reflection geometries are more apparent. The typical seismic processing sequence is very well explained in detail in an online source from [5] which is explained as follows.

The typical seismic data processing starts with reading the data recoded from the tape, all the traces recorded for a given short called short records are displayed (as in fig 2.9)(1) as in fig. The next step in the process is editing (2) out the bad seismic traces, due to short circuits in recording equipment or due to noise interference. Then the common mid-point gathers (3) are collected in such a way that each gather of the trace belong to a common reflection surface point. In the next step direct arrivals and surface waves are removed by digital filtering called muting (4). The next step is to perform move out corrections (5) for the time the reflected ray spends traveling laterally, so the reflected arrivals are lined up. These traces are then stacked which cancels out the noise and reinforces the reflected waves. The traces are added here to produce a single output trace as in (6). Then traces are shrunk by deconvolution method or by frequency filtering to improve the resolution (7). Thus steps (4) through (7) are repeated for each common reflection point and resulting seismic traces are displayed as seismic sections which can be interpreted.

2.3 Basics of Migration techniques:

Migration:

Migration of the seismic data involves repositioning the measured seismic data to determine accurately the topology of the subsurface reflections. This process is best useful in areas when rocks are more complex and not uniform with dipping reflectors. In general migration is done as another routine process on all the seismic sections, but the effect of the same might not be seen in all the scenarios. It moves the dipping reflections to their true subsurface positions and collapses diffractions thus increasing spatial resolution and yielding a seismic image of the subsurface.

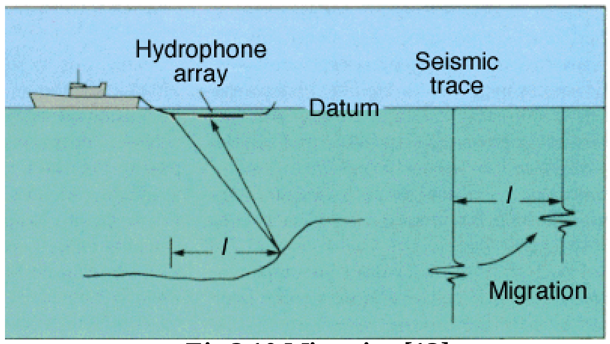


Fig 2.10 Migration [13]

Complex rocks scatter the echoing seismic waves in unexpected directions because of diffractions. Then stacked sections may show such subsurface features in wrong location, if imaged. Using migration an accurate picture of underground layers can be obtained. In this process the traces will be geometrically repositioned to their true subsurface position using migration algorithms. Migration can be performed in two domains time or depth. There are certain imaging problems that can be solved using time migration, but most complex problems need depth migration.

There are two important migration methods:

- *Pre-stack migration:* Pre-stack migration is essentially done when seismic data is adjusted before the stacking sequence occurs. This method can handle the most complex structures and velocity fields.

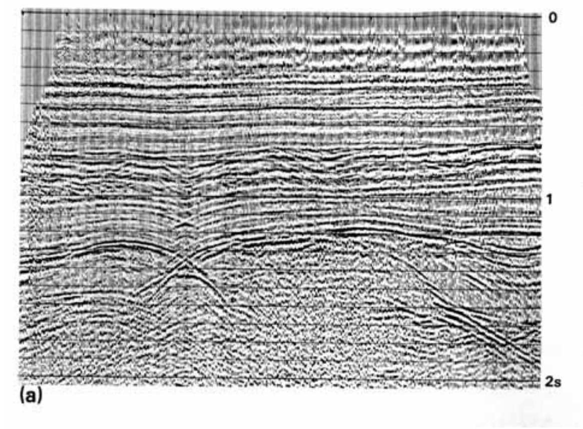
- *Post-stack migration:* Post stack migration is the process of migration in which the data is stacked after it has been migrated. This method is usually much faster than pre-stack migration, because stacking reduces by an order of magnitude the number of traces that must be processed. For the post-stack migration to be successful, the assumptions made in stacking must be well-founded. The amplitude of the stacked trace must represent [11] that of the normal incidence trace and reflected arrivals must be approximately hyperbolic.

Migration process is usually performed on stacked data but as computational power is growing with time pre-stack migration is also becoming more common. Seismic migration algorithms are computationally complex, very intensive and require processing large amounts of data. Stacking reduces the amount of data to be migrated and improves the signal to noise ratio, yet the tasks remain computationally complex. In addition to this we also have models for processing three-dimensional data, which calls for dramatic increase in the computational requirements. Therefore migration is considered as computationally expensive operation.

2.4 Kirchhoff Migration

Kirchhoff migration:

Kirchhoff migration is one of the approximate and affordable approaches. Kirchhoff migration is based on diffraction summation approach, in this method the amplitudes are summed along the diffraction hyperbola and the result is placed as its apex [1]. The aperture width used for the amplitude summation is an important parameter that affects the quality and performance of the Kirchhoff's migration. This migration method uses integral form of wave equation. All the methods of seismic migration involve the back propagation of seismic wave field from the region where it was



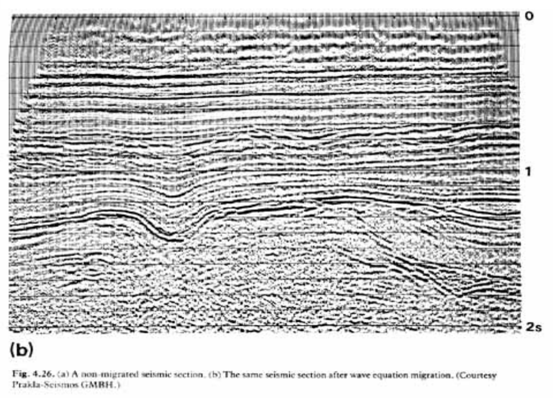


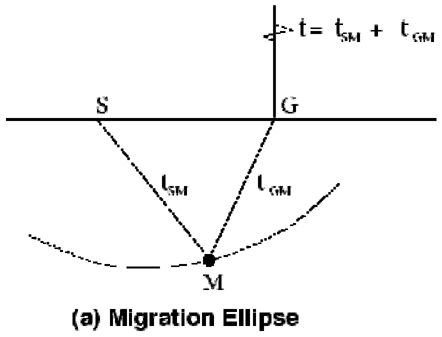
Fig. 4.26. (a) A non-migrated seismic section. (b) The same seismic section after wave equation migration. (Courtesy Prakla-Seismos GMBH.)

Fig 2.11 (a) Seismic section before migration **Fig 2.11 (b) Seismic section after migration[14]**

measured into the region to be imaged. In Kirchhoff's migration this is done by using the Kirchhoff integral representation of a field at a given point as a super position of waves propagating from adjacent points [15]. Continuation of wave filed requires a background model of seismic velocity, which is usually a model of constant or smoothly varying velocity. Because of the integral form of Kirchhoff migration its implementation reduces to stacking the data along the curves that trace the arrival time of energy scattered by important points in the earth.

Kirchhoff migration principle:

The principle of Kirchhoff migration is that a wave field at a given point in space and time can be considered as the superposition of waves propagating from adjacent points due to diffractions. Using wave equation a point is represented as the sum (integral) of contributions from a surface enclosing (hyperbola) the given point.



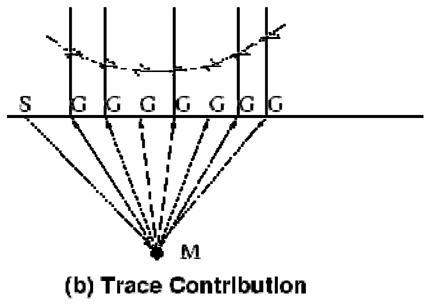


Fig 2.12 (a) Migration ellipse [16] **Fig 2.12 (b) Trace contribution[16]**

The wave equation is the basis for Kirchhoff migration. This equation is applied recursively to all the data points on space time plane to get each point on the depth plane. By summing along the hyperbolas when the equation is applied we get the imaging of subsurface. The point M in the fig 2.12(b) is a point scatterer. Wave incident on point M from any direction will reflect a wave in all directions. Model is superposition of such points.

Sequential Kirchhoff migration algorithm:

The Kirchhoff migration is used to model each point in the depth (x, z) plane by a hyperbola in the data (x, t) plane. In the pseudo code below the parameters ih refer to the separation of a point on a hyperbola from its top at ix. This algorithm computes corresponding z point by summing each t on a given x point to get the z point in (x, z) plane [17] and then plot back the (x,z) migrated point in the (x,t) plane. The terms nx refers total number of 'ix' (trace) points, nt refers to total number of 'it' (time) levels, h refers to maximum trace number in aperture width and nz refers to total number of depth points in (x,z) plane. Pseudo code of the 2D Kirchhoff migration algorithm is as follows

```
KIRCHHOFF MIGRATION (nx, nt, h, velocity, nz)
Do ix = 1, nx
    Do it =1, nt
        Do ih = -h to +h
            iz = $\sqrt{(\text{it} * \text{velocity})^2 - (\text{ih})^2}$
            ig = ix + ih
            zz(iz, ix) = zz(iz, ix) +tt(it, ig)
        od
    od

    Do iz = 1, nz
        Do ih = -h to +h
            it = $\sqrt{(\text{iz}^2 + \text{ih}^2)/\text{velocity}}$
            ig = ix + ih
            tt(it, ig) = tt(it, ig) +zz(iz, ix)
        od
    od
od
```

2.5 Time Complexity Analysis

Time complexity analysis Basics:

To analyze an algorithm is to determine the amount of resources algorithm needs to execute it. The efficiency and complexity of an algorithm is stated as a function of input size to number of steps which is measured as time complexity or output locations which is measured as space or memory complexity. Algorithm analysis is one of the important aspects in computational complexity theory, which provides theoretical estimations for the resources needed by an algorithm for solving a given computational problem [18]. Such estimations can provide us with insight in identifying efficient algorithms. By theoretically analyzing algorithms we estimate their complexity in asymptotic sense. Asymptotically means we estimate the complexity function for reasonably large inputs. This is represented as Big O notation, omega notation and theta notation. Different implementations of the same algorithm may differ in efficiency, but in most cases efficiencies of two reasonable implementations of a given algorithm are related by a hidden constant.

Complexity of 2D Kirchhoff migration Algorithm:

Finding asymptotic running time of Kirchhoff migration algorithm

```
KIRCHHOFF MIGRATION (nx, nt, h, velocity)                          Cost Time

Do ix = 1, nx                                                      C1    (X+1)
    Do it =1, nt                                                   C2    X(T+1)
        Do ih = -h to +h                                           C3    XT(H+1)
            iz = $\sqrt{(\text{it} * \text{velocity})^2 - (\text{ih})^2}$        C4    XTH*1
            ig = ix + ih                                           C5    XTH*1
            zz(iz, ix) = zz(iz, ix) +tt(it, ig)                    C6    XTH*1
        od
    od

    Do iz = 1, nz                                                  C7    X(Z+1)
        Do ih = -h to +h                                           C8    XZ(H+1)
            it = $\sqrt{(\text{iz}^2 + \text{ih}^2)/\text{velocity}}$            C9    XZH*1
            ig = ix + ih                                           C10   XZH*1
            tt(it, ig) = tt(it, ig) +zz(iz, ix)                    C11   XZH*1
        od
    od
od
```

The running time of the algorithm is the sum of running times for each statement executed; a statement takes c_i steps to execute and is executed n times will contribute $c_i\, n$ to the total running time. To compute T(n) the running time of Kirchhoff algorithm we sum the products of the cost and times columns obtaining

$$T(n) = (X+1)C_1 + [X(T+1)]C_2 + [XT(H+1]C_3 + (XTH)C_4 + (XTH)C_5 + (XTH)C_6$$
$$+ [X(Z+1)]C_7 + [XZ(H+1)]C_8 + (XZH)C_9 + (XZH)C_{10} + (XZH)C_{11}$$

$$T(n) = XC_1 + C_1 + (XT)C_2 + XC_2 + TC_2 + (TXH)C_3 + (XT)C_3 + (XTH)C_4 + (XTH)C_5$$
$$+ (XTH)C_6 + (XZ)C_7 + XC_7 + (XZH)C_8 + (XZ)C_8 + (XZH)C_9$$
$$+ (XZH)C_{10} + (XZH)C_{11}$$

$$T(n) = XTH(C_3 + C_4 + C_5 + C_6) + XZH(C_8 + C_9 + C_{10} + C_{11}) + XT(C_2 + C_3)$$
$$+ XZ(C_7 + C_8) + X(C_1 + C_2) + TC_2 + C_1$$

$$T(n) = O(XTH + XZH)$$

This running time can be expressed as an^3 + bn^2 + cn + d for constants a, b, c and d that depend on statement costs C$_{i.}$

The worst case running time can be expressed as 3rd degree polynomial.

Asymptotic Notation for the same is as follows

$$T(n) = O(XTH + XZH)$$

Worst Case analysis

This characterizes the asymptotic behavior of function by providing an upper bound on the rate at which the function grows as *XTH+XZH* gets large. (Worst case analysis)

2.6 PRAM Model Introduction

Parallel Random Access Model (PRAM):

The theoretical models are termed as abstract models for developing algorithms. In a uniprocessor computing environment Random Access Model (RAM) is widely used in developing algorithms. This model had made it possible to develop uniprocessor algorithms and establish the algorithm performance relatively independent of the specific uniprocessor system on which program is to be run. This model had the property that algorithm that worked well on the model worked well on real serial computers regardless of architectures.

When we talk about parallel computers, we can find many abstract models of computation. However there is no model that is as simple and straightforward as RAM. The reason for this might be because of difficulties involved in formulating a simple model generalizing for all parallel computers as there are several variations among the parallel architectures themselves. Parallel Random Access (PRAM) model is a straightforward and natural generalization of RAM. It is a simple synchronous shared memory model. It consists of a collection of *p* sequential processors, each with its own private local memory and communicating with each other through a shared global memory. The input to the parallel algorithm consists of *n* items stored in *n* shared memory cells and output is *n'* items stored in *n'* shared memory cells. [19]

A PRAM computation is a sequence of steps alternating between three types of instructions:

- read
- compute
- write

As in RAM model all the three steps are assumed to take unit time in the model. This model is relatively easy and simple to use as its shared memory abstraction hides the details of the interprocessor communication and synchronization.

As discussed in [20] mapping PRAM algorithms onto actual shared memory MIMD (Multiple instruction multiple data) machines is not straightforward, because many of the realistic machines have limited communication capabilities than the PRAM.

The PRAM model assumes each processor can access any shared memory cell in unit time, but in reality the realistic machines are more limited. To model shared memory synchronous parallel computers [21] we need to consider variations such as EREW-PRAM (exclusive read, exclusive write-PRAM), CREW-PARAM (concurrent read, exclusive write-PRAM) and CRCW-PRAM (concurrent read, concurrent write-PRAM) that account for differences in memory access conflict resolution schemes. Another aspect of PRAM model is asynchronous shared memory parallel computers. Since many of the existing MIMD parallel machines are asynchronous PRAM assumes the processors execute in lock-step. In order to effectively support a large number of processors, and multiple instruction streams realistic machines must permit each of the processor to execute its instruction independently of timing of the other processors. These aspects are well described in [20] where it focuses on asynchronous PRAM model.

Asynchronous PRAM [20]:

We now present the discussion on asynchronous PRAM from reference [20] in this section. Similar to basic PRAM model the Asynchronous PRAM model too [20] consists of p sequential processors, each with its own private local memory, communicating one another through a shared global memory. In this case each local processor has its own local program. In Asynchronous PRAM each processor executes its instruction independently of the timing of the other processors without any global clock. A processor can issue up to one instruction per local clock tick and an instruction is said to be complete after a finite delay. In this case there are four types of instructions:

- *Global reads:* reading the contains of shared memory locations into a local memory location
- *Local operations:* typical operations performed on data present in local memory and the result is also stored in local memory.
- *Global writes:* writing the contents of local memory cell into global memory cell.
- Synchronization steps.

A synchronization step among set of S processors is one in which each processor in S waits for all the processors in S to arrive before continuing in its local program. All the instructions for processors in S prior to synchronization step should complete before any processor in S issues an instruction from its next phase. However, all the processors can read and write to the shared memory asynchronously, but no processor can read the same memory location that another one writes unless there is a synchronization step involving both the processors between the two accesses.

In general PRAM a processor at step *i* can use the fact that all the processors have completed step *i-1*. Whereas, in asynchronous PRAM, because of absence of global clock and possible arbitrary delays a processor at instruction *i* does not automatically know the progress of the processors. Therefore in order to be sure that all the accesses to a global location at a point in the computation have completed, the processor must synchronize with all other processors that might be accessing the location.

2.7 Parallel Kirchhoff migration:

Parallel Kirchhoff migration details:

The parallel version of Kirchhoff migration algorithm adheres to the entire properties PRAM model. This algorithm is more like Asynchronous PRAM because each migration location process is independent of one another. From the reference [1] we can see that Kirchhoff time migration is implemented using MPI I/O parallel programming environments by Centre of Development of Advanced computing, (C-DAC) located at Pune, India. The Kirchhoff time migration algorithm for both 2D and 3D data volumes are developed and implemented on a 100 GF distributed memory parallel computer, also known as PARAM 10000. For this effort synthetic data set for an overthrust model, provided by SEG/EAGE is used for testing of the codes.

Pseudo-code for Parallel Kirchhoff migration algorithm from reference [1]:

```
INPUT DATA:
P_in (x, y, t) → stacked data in SEGY Format
V_rms (x, y, t) → rms velocity data
PHASE CONVERSION OF INPUT TRACES
P_phase (x, y, t) → Apply the 45/90 degree phase-shift to 2D/3D input traces
Calculate the aperture function
MIGRATION COMPUTATION
For I = 1, 2, ….. No. of seismic lines. (For 3D data only)
{
    For J = 1, 2, ......., No. of output locations (for 2D and 3D data both)
    {
        Calculate the no of cdp required for this migration location
        Using the maximum half-aperture width from the aperture function.
        Get the First and the last input cdp number
```

```
                For K = First input cdp, Last input cdp
                {
                1. Calculate the hyperbolic trajectories for each input trace at each
                time level
                2. Interpolate the input trace and phase-shifted trace using the sinc interpolation
                3. Calculate the obliquity and spherical spreading factors for input
                trace for each time level
                4. Multiply the interpolated input trace with obliquity factor and multiply
                the interpolated phase shifted input trace with spherical spreading factor.
                Add both the traces.
                5. Carry out summation of this input trace according to the aperture
                function of that location for each time level.
                }
                write the migrated traces in the output file according to the output locations.
        }
}
```

OUTPUT

Pout (x, y, and t) → migrated seismic section

Explanation of paralleling the above code:

The above parallel algorithm is inline with the asynchronous PRAM model of computation. The output locations loop is parallelized. In the parallel implementation of the algorithm migration location and velocity will be computed by the master processor and send to the slave processors. Based on the migration location master can calculate the first and last common depth points (Cdp) (amount of input data for migrating their locations) and sends it to the worker processors. Once the master sends the required information for division of migration locations the workers read their potion of input data directly from data volume. This is like slave processor reading from global memory location as in asynchronous PRAM model. Then the slave processors do all the required computations and write the migrated output directly to the disk. This is writing back the results back to the global location.

Time Complexity using PRAM:

Finding complexity of parallel algorithm is no different from complexity of sequential algorithm.

MIGRATION COMPUTATION

```
                                                                         Cost   time
For I = 1, 2, ….. No. of seismic lines. (For 3D data only)   ----------------→ C1      (I+1)
{
  For J = 1, 2, ......., No. of output locations (for 2D and 3D data both) -→ C2    I(J+1)
  {
    Calculate the no of cdp required for this migration location-----------------→ C3    IJ * 1
    Using the maximum half-aperture width from the aperture function. -------→ C4    IJ * 1
    Get the First and the last input cdp number
    For K = First input cdp, Last input cdp ---------------------------------→ C5   IJ (K+1)
    {
        1. Calculate the hyperbolic trajectories for each input trace at each ---→ C6    IJK * 1
           time level
        2. Interpolate the input trace and phase-shifted trace using the sinc -----→ C7    IJK * 1
           interpolation
        3. Calculate the obliquity and spherical spreading factors for in---------→ C8    IJK * 1
           trace for each time level
        4. Multiply the interpolated input trace with obliquity factor -----------→ C9    IJK * 1
           and multiply the interpolated phase shifted input trace with
           spherical spreading factor.
           Add both the traces.
        5. Carry out summation of this input trace according to ----------------→C10    IJK * 1
           the aperture function of that location for each time level.
    }
    write the migrated traces in the output file according ----------------------→C11    IJK * IJK
    to the output locations.
  }
```

}

The running time of the algorithm is the sum of running times for each statement executed; a statement takes c_i steps to execute and is executed n times will contribute $c_i\, n$ to the total running time. To compute T(n) the running time of Kirchhoff algorithm we sum the products of the cost and times columns obtaining

$$T(n) = C_1(I+1) + [I(J+1)]C_2 + (IJ)C_3 + (IJ)C_4 + [IJ(K+1)]C_5 + (IJK)C_6 + (IJK)C_7 \\ + (IJK)C_8 + (IJK)C_9 + (IJK)C_{10} + (IJK)^2 C_{11}$$

$$T(n) = (IJK)^2 C_{11} + IJK(C_5 + C_6 + C_7 + C_8 + C_9 + C_{10}) + IJ(C_2 + C_3 + C_4 + C_5) \\ + I(C_1 + C_2) + C_1$$

When parallelized over output locations. First part of T(n) is parallelized.

This running time can be expressed as $an^3+bn^2+cn$for constants a, b, c that depend on statements cost $C_{i.}$ It is this a quadratic function of $(IJK)^2+ (I+1)$

$$T(n) = O(IJK)^2 + O(I)$$

This runtime is for 1 processor system.

For p processor system it will be

$$T\left[\frac{n}{p}\right] = O\left(\frac{(IJK)^2}{p}\right) + O(I)$$

2.8 Methodology

Proposed Plan for solution:

The methodology for the solution of the problem is explained in this chapter. To achieve the goal firstly, after identifying seismic migration as one of the most compute intensive process we studied the function of Kirchhoff migration algorithm which as most effective algorithm in terms of cost and time and understood the sequential implementation of the same and computed the running time complexity of the same using the RAM model. As the Kirchhoff migration algorithm has high potential to be parallelized we studied the parallel model of the same. The theoretical complexity of the algorithm is computed in form of the asymptotic running time complexities of the algorithm for a single processor system as well as for *p*-processor parallel system. From reference [1] we use the results of the Kirchhoff's parallel implementation on the PARAM system.

In our proposed solution we are mapping the algorithmic theoretical complexity with the actual values obtained by the running the Kirchhoff algorithm on the actual PARAM System. From the reference [1] we have the results of running the Kirchhoff time migration parallel implementation code on PARAM 10000 computer with 40 nodes having 4 CPUs and 513 MB RAM per node. From their results we have execution times in seconds when the data set of volume 61 MB of input size segy seismic data and 52 MB velocity data run on 32 processors as we get the total execution time as 48 min. In similar way the data has been collected for same input size and for different no. of processors like 8, 16 and 24. Now using the same input size of the data we substitute the input values into the theoretically computed worst case running time asymptotic functions and get the values in terms of flops of a factor K_p. We also convert the actual run time values from the reference [1] in terms of flops of processors. Once we have both the values obtained theoretically and actual values from the PARAM system, we relate the functional behavior of both by proposing a function in terms of input size and number of processors which gives compute power in terms of execution times. By this we attempt to established a theoretical model which takes data volume as input and can give a near to close approximation of compute power in terms of running times.

Theoretical function to be used will be

For *p* processor system it will be $T(n/p) = O((IJK)^2/p) + O(I)$

Where I = no of seismic lines to be taken in case of 3D data

J = no of output locations (X points to consider)

K = Cdp values.

Using these three parameters running time of the algorithm on *p* processors can be found out.

The function derived will take the input data size. Defining a function or mapping between theoretical and actual is not only the goal of this thesis but we will also examine the sources of inaccuracies which need to be considered while using the compute power function. We will also explain the ideal case if we take into consideration all the unstated assumptions and approximations and what are the effects of those inaccuracies on the solution proposed.

2.9 Review of Literature

Reference [1] presents the parallel implementation of 2D and 3D Kirchhoff migration algorithm. The reference describes a new approach in aperture width selection which is a crucial factor in obtaining a high resolution image of the subsurface structures. It also discusses about the implementation of the parallel algorithm for Kirchhoff time migration for 2D and 3D data sets. The parallel algorithm is developed on PARAM 10000, a distributed memory parallel system consisting of 40 nodes. Each node has 4 CPUs and 512 MB of RAM. Nodes are interconnected through high speed network for communication and the clusters use NFS (Network file system) and MPI (Message passing Interface) calls to communicate and synchronize between the processors. The implementation of the algorithm is based on Master-Slave model system. In Master-Slave system, the job of the master is to provide the required data and parameters to the slave processors and also distribute the work among those processors in such a way that it minimizes the idle time of the workers and also it is the master's responsibility to collect the result of finished work from the slave processors. This is the general idea about master slave system. In actual MPI I/O parallel implementation master sends only the required parameters to workers and identifies the portion of the data to be processed by each worker. Then it is the worker's job to read the needed input directly using MPI I/O and processes that data and writing back the migrated results directly on the disk using MPI I/O. The parallelization of Kirchhoff migration algorithm uses the data-parallelism approach and the parallelization is done over the output locations (for X locations). Each migration point and its corresponding velocity which can be a constant velocity for the whole model or can be varying one is send to the slave processor after computing the

Common depth points (Cdp) for that location and also identifies how much data has to be each worker requires for migrating the locations. In a 2D case output locations are Cdp locations of a line, and incase of 3D they are seismic lines. In the parallel implementation with MPI I/O, master is said to send only parameters required for migrating location and velocity to the slave. It is the job of the slave to read the locations from the global data volume and process the calculations and write the migrated output directly to the output location.

The parallel algorithm for the 2D and 3D Kirchhoff migration is applied to the data set of SEG/EAGE (1997) overthrust model with 101x25 CDP traces with inline spacing of 100m and crossline spacing of 100m and interpolated data is 401x97 CDP traces with both inline and crossline spacing of 25m. Each CDP trace has 350 time samples with sampling rate of 8ms.

The scalability of the algorithm is studied by running the code on several CPUs. Some data with execution times as function of the no. of processors has been recorded and found from their results that the graph is scalable. The proposed solution to the problem makes use of the parallel implementation results from the above approach and attempts to map a function between theoretical running time complexities computed from the algorithm and the actual run times of algorithm. And finding out the estimated run times by substituting the real data set used in the reference [1] into the theoretical asymptotic function and the model derived.

CHAPTER III

RESULTS

3.1 Solution Approach

The solution for the problem is explained in this chapter. In Process of finding this solution firstly Kirchhoff migration implementation on sequential machines is studied and the running time complexity is computed following the RAM model. Then we studied the parallel model of the same algorithm and complexity of the parallel version of algorithm is computed. Kirchhoff migration can be separated into 2 steps firstly each point in the depth (x, z) plane is imaged from the hyperbola summation in the data (x, t) plane then the next step is to model the migrated point back into the data (x,t) plane to get back the effect of the migration process. The first part is time to depth imaging and the second part is depth to time modeling.

These asymptotic complexities are used in computing the parallel complexity of the parallel Kirchhoff migration algorithm. This is explained in greater detail later in this chapter. The actual running time values of the Kirchhoff migration are taken from reference [1] which has parallel implementation code on PARAM 10000 computer with 40 nodes having 4 CPUs and 513 MB RAM per node. From these results we have execution times in seconds when the original data set of SEG/EAGE (1997) Overthrust model which had 101x25 CDP Traces with inline and crossline spacing of 100m was used. The original Execution time Vs. No. of processors graph is below.

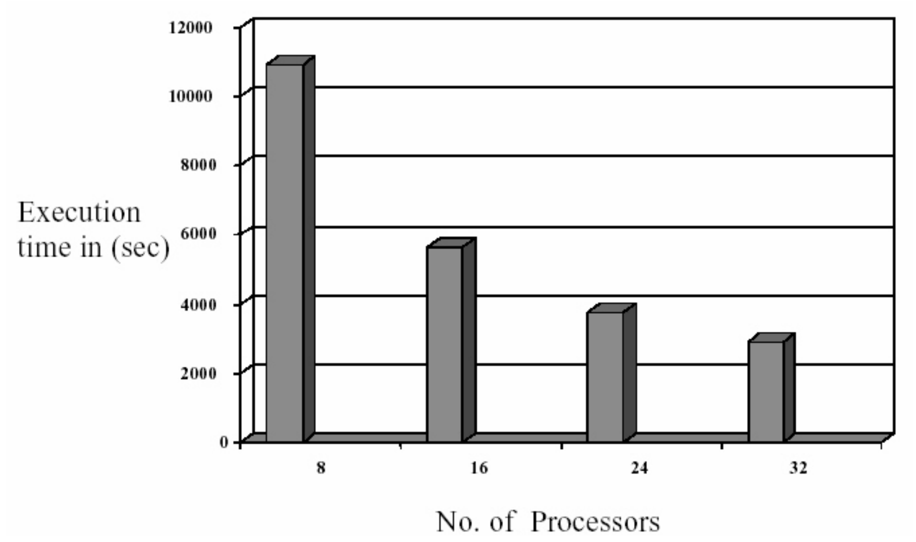


Fig 3.1 Actual running time graph of Kirchhoff migration algorithm on PARAM 10000 [1]

The table of actual values of execution times in seconds taken from the above graph is given below:

No. of Processors	Exec. Time in Sec
8	11000
16	5800
24	3900
36	3000

Table 1: Actual running times of Kirchhoff migration on PARAM 10000 computer.

The same input size of the data is used to compute the theoretical running time values of the algorithm from asymptotic functions. The theoretical asymptotic running time for the parallel Kirchhoff migration algorithm is expressed as

$T(n) = O((I^2J^2K^2)/p) + O(I)$

This is the running time for the algorithm executed on *p* processors

Where the asymptotic function parameters are

Where I = no of seismic lines to be taken

J = no of output locations (Z points to consider)

K = Cdp values.

From the reference [6] findings we have the input data set values which are 101x25 CDP traces. From this we can infer that

- no of seismic lines to be considered are 101
- no of output locations will be 101 times 25 (101*25) which is 2525
- Cdp values can be taken as 104 which are obtained from maximum aperture function.

This gives us I = 101, J = 2525 and K = 104.

Now plugging in these values into the asymptotic function gives us the approximate running times for *p* = 8, 16, 24 and 32. The table below shows the values.

No. of Processors	**Theoretical Flops per processor**
8	8.7925×10^{13}
16	4.3962×10^{13}
24	2.930×10^{13}
36	2.1981×10^{13}

Table 2: Theoretical flops calculated from asymptotic function of Kirchhoff migration algorithm

The graph showing the execution time computed using the asymptotic function is shown below:

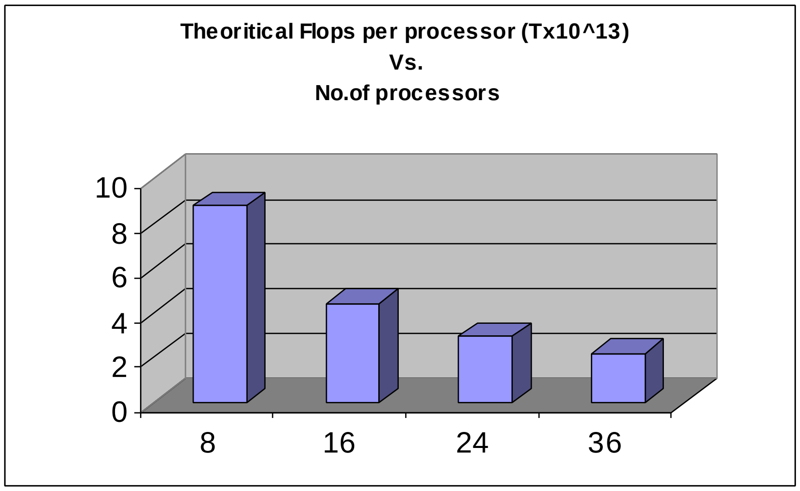


Fig 3.2 Graph of theoretical flops calculated from asymptotic function of Kirchhoff migration

The execution time ratios of the actual values and the theoretical values when compared it is found that theoretical and actual values approximately are having the similar increase. The parallel implementation of the Kirchhoff algorithm is implemented on PARAM 10000 computer whose system parameters specify the per processor flops of the computer as 800 Mflop/s. with processor speed of 400MHz [22]. The actual running time values obtained from reference [1] are also expressed in flops by multiplying with the PARAM 10000 computer flops i.e., 800 MFlops/s and are equated to the theoretical flops the theoretical values obtained are expressed as factor of K_p in terms of flops. Similarly the actual values obtained from reference [1] are also expressed in terms of flops, since the theoretical values should match the actual values these two results are equated as below and corresponding K_p values (units of K_p are flops) are calculated:

$$Actual\_Values = K_p \times Theortical\_values$$

3.2 Results

Thus we get the values of K_p for p= 8,16,24,36 processors. The table and its corresponding graph below shows the values and behavior of K_p values.

No. of Processors	**K_p Values**
8	0.100
16	0.105
24	0.106
32	0.109

Table 3: K_p values

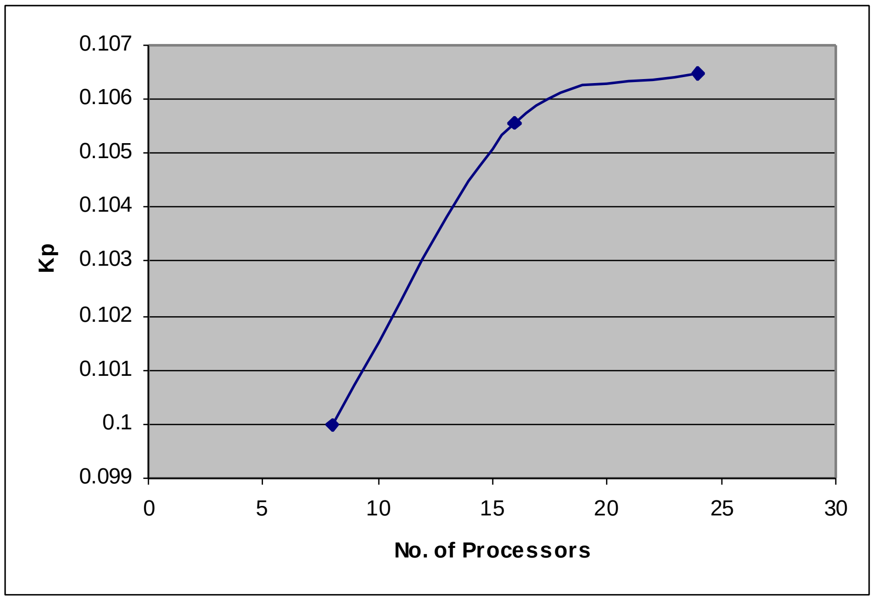


Fig 3.3 Graph of No. of processors Vs. K_p values obtained by equating actual and theoretical flops.

A graph is plotted using the K_p values from the table 4. We have to note that the graph is plotted only using the first three values i.e., 8, 16 and 24 K_p values. The reason for doing so is, we define a model using the first three values of K_p and verify its correctness by obtaining the K_p value for 32 processors using the model, similarly we can extended this to get values for any no of processors *p*.

Using the K_p values from the graph, we develop a linear model for the K_p using simple linear regression method in form of equation y = m(x) + b where values of m, and b are calculated using implicit linear regression formulae [23].

$$m = \frac{n\sum(xy) - \sum x \sum y}{n\sum(n^2) - (\sum x)^2}$$

$$b = \frac{\sum y - m\sum x}{n}$$

$$r = \frac{n\sum(xy) - \sum x \sum y}{\sqrt{\left([n\sum(x^2) - (\sum x)^2][n\sum(y^2) - (\sum y)^2]\right)}}$$

In our case the linear model will be

$K_p = a \times number\_of\_processors + b$, where a is to be determined by regression.

a is slope of the equation determined using linear regression slope formula.

The values obtained for a, b intercept and r are in table 5. below:

Parameters	**Values**
A	0.000405
B (y intercept)	0.097537
R	0.9253

Table 4: **Table of values for slope, y-intercept and correlation coefficient for linear model of K_p and no. of processors**

The *r* value which is correlation coefficient is obtained as 0.9253. The value of *r* closer to 1 indicates that there is an excellent linear reliability between

K_p and no. of Processors.

So, $K_p = 0.000405 \times number\_of\_processors + 0.097537$

Therefore K_p is found by linear regression method.

We now estimate the running time of the algorithm using the Kp values from the above model, which gives us estimated flops for 8, 16, 24 and 32 processors. The values are shown below

No. of Processors	**K_p Values**
8	0.1007
16	0.1040
24	0.1072
32	0.1104

Table 5: Table of K_p values calculated using the linear model of Kp and no. of processors

We multiply the Kp values with the Theoretical flops from table 2 to get the estimated flops for the algorithm on 8, 16, 24 and 32 processors. These values are divided by the actual system's processor flop capacity to get the running times in seconds. In this case it will be PARAM 10000 computer per processor flop capacity which is 800 Mflop/s. to get the estimated running time. The table 7 shows the estimated running time of the algorithm.

No. of Processors	**Estimated running time in sec**
8	11062
16	5712
24	3925
32	3025

Table 6: Table of estimated running times of the Kirchhoff migration algorithm

The graph below shows the trend of estimated running time and actual running time of the Kirchhoff migration algorithm.

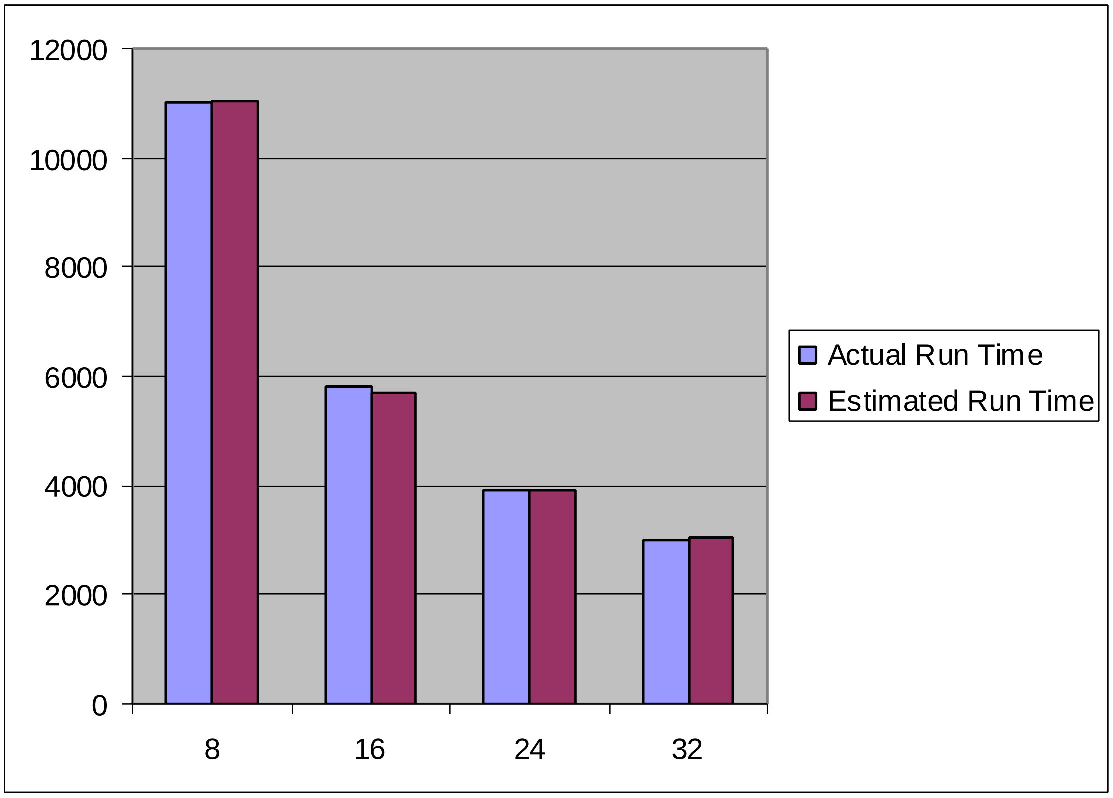


Fig 3.4 Comparative graph of actual and estimated running times.

Therefore this model can be defined as

Actual running time for the application = K_p (theoretical running time of the algorithm)

3.3 Explanation of the results

Given a compute intensive algorithm we can always compute the time complexity of the algorithm and based on the processor peak performance flops of the network/system architecture we can compute the K_p values from the simple linear model

$$K_p = a \times number\_of\_processors + b$$

that is proposed. Using the factor of these two will approximately give us the actual running time of the algorithm on specific architecture.

3.4 Assumptions and Inaccuracies in the model

The model proposed is not perfect as there are some of assumptions and inaccuracies. Few such things are discussed here.

1. The aperture width function used in Kirchhoff migration algorithm plays a good part in getting the more accurate migration results. In our study this has been made a simple and a more general function which applies the same equation for every point of migration is used. Using a more optimal method of aperture function might increase or decrease the complexity of the algorithm which might improve the results.
2. The model is just tested on single data set due to the restrictions on resources to get more data sets. Testing on more data sets might result in better understanding of the model this identifying the errors and approximations more easily.
3. In terms of computation aspects the parallel algorithm used in creating this model is implemented on a specific architecture. In theoretically computing the time complexity of algorithm we ignore the actual time required for the processor synchronization and the communication time between master and slave processors of the architecture. This time might be of great importance in real life.
4. The model is proposed based on single application of the seismic data processing. Taking all other data processing algorithm will results in more accurate model for resource estimation.
5. The study is more based on contact velocity model, a varying velocity model will have a better results.

CHAPTER IV

CONCULSION

This thesis describes creation of a new theoretical model using which, an estimate of computational power needed can be found for a seismic migration. In process of working towards this goal we map the theoretical complexity of one compute intensive migration algorithm, the Kirchhoff migration with the actual results of running time of the algorithm. In the course of this thesis we identified migration process as most compute intensive process and had examined the Kirchhoff migration algorithm in detail and computed the running time complexity of the algorithm on a serial computer architecture and parallel architecture. The results and the input data sets from parallel implementation of Kirchhoff migration algorithm on PARAM 10000 computer by researchers at C-DAC, India are used in process of defining the model proposed.

The importance of this thesis is by proposing such a theoretical model we will be helping oil and gas exploration companies in initial resource planning and estimating the compute power they require for compute intensive processes. The model is completely based on theoretical study of the algorithm rather than empherically obtaining the results by actually running the application on system architecture. All the techniques used in this thesis towards achieving the goal are done in an analytical fashion either deriving the values of processor flops using regression or running time asymptotic notations.

In this study we use the theoretical results and the actual running time values of an algorithm and try to map a linear relation between them. In this process we convert the values obtained theoretically and actual values in terms of Flops that particular computer architecture can handle and equated them to get a factor $K_{p.}$ The behavior of K_p being linear we derived the K_p value using regression and proposed a model using this K_p value for the problem.

In future the Kirchhoff migration implementation should be done and tested on different parallel architectures and the actual and theoretical results should be compared and by doing so, this model can be made more universal. Apart from this the same procedure can be extended to many other seismic processes and algorithms and a model can be created involving all the processes complexities thus giving a complete solution for oil and gas companies.

REFERENCES

[1] R. Rastogi and S. Phadke. *Optimal aperture width selection and parallel implementation of Kirchhoff migration algorithm*. SPG 4th Conference & Exposition on Petroleum Geophysics. India. January 2002.

[2] G. Schuster. *Stanford Mathematical Geophysics Summer School Lectures: Basics of Exploration Seismology and Tomography*. January 1999. 13 July. 2005. http://utam.geophys.utah.edu/stanford/node7.html.

[3] G. Schuster. *Stanford Mathematical Geophysics Summer School Lectures: Basics of Exploration Seismology and Tomography*. January 1999. 13 July 2005. http://utam.geophys.utah.edu/stanford/node8.html.

[4] *Equipment—Receivers, Geophones*. Kansas Geological Survey, Exploration Services Section. March 2000. April 2005. http://www.kgs.ku.edu/Geophysics/Equip/phones.html.

[5] *About the seismic reflection profiling technique: The seismic reflection method*. 05 Nov. 1998 U of Calgary. Alberta http://www.litho.ucalgary.ca/atlas/seismic.html. 02 March 2005.

[6] Walter Kessinger. *Overview of Seismic Exploration: Seismic data acquisition*. http://walter.kessinger.com/work/images/seisx_fig3_big.gif. 3rd February 2005.

[7] Glaccum, Robert. *Seismic Methods: Seismic Reflection*. Geosphere Inc. April, 2001. http://www.geosphereinc.com/seis_reflection.html, 29 January 2001.

[8] Resolution Resources: 3D Acoustic Imaging for the environment, *Seismic surveys*, http://www.rr-inc.com/Frame%20Pages/Tech%20Pages/Seismic/seismic.htm, 2001, 10th October 2005.

[9] Baker S. Gregory. Young A. Roger. *Processing Near-Surface Seismic-Reflection Data: A Primer*. Society of Exploration Geophysicists, ISBN 1560800909. Chapter 2. pp.45-63, December 1999.

[10] Schlumberger. *Oilfield Glossary: Diagram of stacking process common midpoint and moveout*. 2006. http://www.glossary.oilfield.slb.com/DisplayImage.cfm?ID=272, April 2006.

[11] Albertin U, Kapoor J, Randall R, Smith M, Brown G, Soufleris C, Whitfield P, Dewey F, Farnsworth J, Grubitz G and Kemme M: "*The Time for Depth Imaging*," Oilfield Review, Spring 2002.

[12] Schlumberger. *Oilfield Glossary: Diagram of stacking process*, 2006. April 2006. http://www.glossary.oilfield.slb.com/DisplayImage.cfm?ID=272.

[13] Schlumberger. *Oilfield Glossary: Diagram of datum and hydrophone array*, 2006. April 2006. http://www.glossary.oilfield.slb.com/DisplayImage.cfm?ID=272,.

[14] O. Yilmaz. *Seismic Data Processing: Investigations in Geophysics*, Vol. 2, Society of Exploration Geophysicists, Tulsa, Oklahoma, 1990.

[15] Schlumberger, *Oilfield Glossary: Kirchhoff migration*, 2006. April 2006 www.glossary.oilfield.slb.com/ Display.cfm?Term=Kirchhoff%20migration.

[16] Jing Chen. *Kirchhoff-Type Prestack Migration Operator*. October 2000. April 2006. http://utam.gg.utah.edu/UTAMtheses/JingChen/latex_2_html/node20.html.

[17] Claerbout F. Jon. *Earth Soundings Analysis: Processing versus Inversion (PVI) 1992* http://sepwww.stanford.edu/ftp/prof/toc_html/toc_html/toc_html/toc_html/toc_html/pvi.pdf, 21 October 1998. Retrived on: 15 July 2005.

[18] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, Second Edition. MIT Press and McGraw-Hill, ISBN 0262032937. Chapter 1: Foundations, pp.3–122, 2001.

[19] Tvrdik Pavel. CS838: Topics in parallel computing, PRAM models, Jan 21, 1999, http://www.cs.wisc.edu/~tvrdik/2/html/Section2.html , Department of Computer science , University of Wisconsin, Madison, 7 April 2006.

[20] P. B. Gibbons. "*A More Practical PRAM Model"*. In 1st Symposium on Parallel Algorithms and Architectures, 1989.

[21] S. Sahni and V. Thanvantri. "*Parallel computing: Performance metrics and models*." Research Report, Computer Science Department, University of Florida, May 1995.

[22] Aad van der Steen. *In Focus – The C-DAC Param 10000 Open Frame*, Top 500 supercomputer sites, http://www.top500.org/orsc/2001/cdac.htm, l July 2001.

[23] *Clemson University: Linear Regression Tutorial*, 2006 http://phoenix.phys.clemson.edu/tutorials/regression/index.html, January2006.

VITA

Sireesha Lanka

Candidate for the Degree of

Master of Science

Thesis: TOWARDS A MODEL FOR ESTIMATING GRID RESOURCE FOR OIL AND GAS EXPLORATION

Major Field: Computer Science

Biographical:

Education: Bachelor's of Technology in Computer Science, AIU, India.

Experience: Graduate Assistant, OSU CHES Development & External Affairs.
Software Developer, Netripples Software Ltd. India.

Name: Sireesha Lanka Date of Degree: July, 2006

Institution: Oklahoma State University Location: Stillwater, Oklahoma

Title of Study: TOWARDS A MODEL FOR ESTIMATING GRID RESOURCE FOR OIL AND GAS EXPLORATION

Pages in Study: 42 Candidate for the Degree of Master of Science

Major Field: Computer Science

Scope and Method of Study:

The seismic exploration is primary tool in search for oil and gas exploration. The results from these explorations are used to produce images of the earth's subsurface, that geologists analyze for understanding the earth subsurface picture. Seismic migration process is one of the most computationally intensive steps of all the seismic data processing sequence. Migration techniques are highly compute and I/O intensive and therefore require high performance systems to carry out the operations efficiently. It is well evident that rather than sequential machines parallel computers provide cost-effective solutions as migration algorithms show lot of potential for parallelism. One such algorithm is Kirchhoff migration. The processing of Kirchhoff migration algorithm involves handling large volumes of data which needs high computational power. Most of the exploration companies cannot afford to have computational power or the estimate of the power they need on their own. Therefore there should be a model or technique by which oil and gas exploration companies can find out an estimate for the computational power they need. This thesis gives a model for estimating the computer power needed to run Kirchhoff migration algorithm for a given volume of data.

Finding and Conclusion:

In this thesis we propose a new technique by which this calculation of resources can be found using a model. Towards this goal we are trying to map the theoretical complexity of one compute intensive migration algorithm, the Kirchhoff migration with the actual results of the same algorithm whose parallel implementation is done on PARAM 10000 computer by some researchers at C-DAC, India. The theoretical results and the actual running time values of an algorithm are used and we map a linear relation between them. In this process we convert the values obtained theoretically and actual values in terms of Flops that particular computer architecture can handle and equated them to get a factor $K_{p.}$ The behavior of K_p being linear we derived the K_p value using regression and proposed a model using K_p factor. Based on migration algorithm results we generalized this model. Using this mapping we derive a function which gives the amount of resources in terms of processors and execution times. Using this we can suggest a cluster or grid confirmation. This attempt will be an initial step towards the goal of finding complete grid confirmation accurately.

ADVISER'S APPROVAL: ______________________________

Dr. Venkatesh Sarangan