A parallel algorithm for three-dimensional gravity modelling and inversion

Diplomarbeit
vorgelegt von
Dror-John Röcher
Bochum, April 2002
## Contents

List of Tables ........................................ IV

List of Figures ........................................ V

1 Introduction .......................................... 1
  1.1 Structure of the thesis .............................. 2

2 Geophysical Gravity Modelling ......................... 3
  2.1 Fundamentals of Gravimetrics ........................ 3
  2.2 Gravimetric Modelling .............................. 4
  2.3 The Nagy Algorithm ................................ 14
  2.4 Conclusion ......................................... 17

3 Nagy’s Algorithm in Action ............................. 18
  3.1 Nagy’s Algorithm .................................... 18
  3.2 Using the parallel Beowulf cluster .................. 22

4 Results .................................................. 24

5 Resumee ................................................. 28

Bibliography .............................................. 31

A Cluster Setup and Maintainance ......................... 34
  A.1 Selfmade Cheap PC Cluster ......................... 35
    A.1.1 ‘Count von Count’ Setup ........................ 35
    A.1.2 General Headnode Setup ........................ 37
# List of Tables

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Block and station layout after figure 3.1 (A)</td>
<td>19</td>
</tr>
<tr>
<td>3.2</td>
<td>Block and station layout after figure 3.1 A adapted for station coordinates</td>
<td>20</td>
</tr>
<tr>
<td>3.3</td>
<td>Results of partial calculations of the layout shown in figure 3.1 - NaN=Not a Number - representing undefined mathematical operations like e.g. ln(0)</td>
<td>20</td>
</tr>
<tr>
<td>3.4</td>
<td>Results of partial calculations of the layout shown in figure 3.1 zeros approximated by 0.001 or -0.001</td>
<td>21</td>
</tr>
<tr>
<td>3.5</td>
<td>Block and station layout after figure 3.1 B, adapted for station coordinates</td>
<td>21</td>
</tr>
<tr>
<td>3.6</td>
<td>Kernel Layout, $T_i$ according to formula 2.35</td>
<td>21</td>
</tr>
<tr>
<td>A.1</td>
<td>Hardware for Count von Count</td>
<td>36</td>
</tr>
<tr>
<td>A.2</td>
<td>Harddisk Partitions</td>
<td>37</td>
</tr>
<tr>
<td>A.3</td>
<td>Standard Diskless Concepts</td>
<td>39</td>
</tr>
<tr>
<td>A.4</td>
<td>Headnode File List</td>
<td>45</td>
</tr>
<tr>
<td>A.5</td>
<td>Node File List</td>
<td>45</td>
</tr>
<tr>
<td>B.1</td>
<td>Graf Zahl (Count von Count) Configuration</td>
<td>64</td>
</tr>
</tbody>
</table>
List of Figures

2.1 Newtonian gravity with two 'Point Masses' ............................... 3
2.2 Geometric layout for the calculation of the gravity effect of a right rectangular prism model .......................................................... 6
2.3 Geometric layout for the calculation of the gravity effect of a polygonal Prism. 8
2.4 Geometric layout of a polyhedron. ............................................. 10
2.5 Geometric layout for the line integral. ....................................... 11
2.6 Polyhedral assembly using triangulation. .................................... 13
2.7 A right rectangular prism with the volume element and its relation to the cartesian coordinate system. .............................. 14

3.1 Top views of a four block, one station right rectangular prism model. . 19
3.2 A schematic view of the PC cluster named "Count von Count" .......... 22

4.1 Cutout of the subsurface model layout and its anomaly response at the surface . 25
4.2 Calculated anomaly and difference distribution ............................ 26
4.3 Surface view of the anomaly. ...................................................... 27

B.1 Trivial communication example with MPI. ............................... 66
B.2 The files lamnodes on the left side and appschema on the right side 67
B.3 Calling mcc from within Matlab on an m-file ............................ 69
B.4 Using a Matlab function within MPI ........................................ 70
B.5 Compiling the pieces ............................................................... 71
B.6 Running the program ............................................................. 71
Abstract

The aim of this diploma thesis was to write a parallel program for three dimensional gravity modelling using an “inversion-suitable” approach. First the cluster of computers had to be reactivated and upgraded after an idle period. This cluster incorporates 17 dual-processor “off the shelf” computers running a standard Linux distribution (SuSE 7.2). The cluster is divided into one server and 16 computing nodes. The computing nodes are diskless clients booting their operating system from the server. The cluster has been designed to minimize maintenance work, to ease installation of new software and to simplify the addition of new computing nodes. The setup is documented in the appendices.

A rectangular prism based algorithm following Nagy (1966) has been chosen for the forward modelling. A parallel object oriented program drawing on MPI++ and Matlab C++ libraries for three dimensional gravimetric modelling on the cluster has been programmed. This program calculates the effect of a subsurface rectangular prism model on a set of surface stations and makes usage of the extended computing power of the cluster.

The results from this calculation have been compared to results from IGMAS. The differences in the results are neglectable and can be accounted to differences in the used algorithm. IGMAS uses a polyhedral approach for the forward calculation whereas the parallel program uses a prism based algorithm. The program can be used to calculate the anomaly of three dimensional density distributions with a size of up to 300,000 prisms (eg. 100 x 100 x 30 blocks in x, y, z). It has been designed to be used in a parallel three dimensional linear inversion program but can be used as a standalone modelling solution, too.

Programs using a polyhedral instead of a prism approach are faster and more flexible in modelling but can not be used for three dimensional geometric inversion. They are limited to density inversion. Using a rectangular prism approach enables a geometric inversion because a change in block densities is equivalent to a change in geometry. This kind of modelling needs huge computing powers and so a cluster approach is preferable.

The program has been written to be expandable. The model description and forward algorithm have been included in a single C++ class with an easy to use interface. This interface can be used in a later inversion program without detailed knowledge of the used forward algorithm or the inner workings of the program.
Chapter 1

Introduction

Gravimetric modelling has become increasingly important in applied geophysics and geophysical research. The work group "Applied Geophysics" at the "Institute of Geology, Mineralogy and Geophysics" at the Ruhr-Universität Bochum has expert-knowledge in the modelling of subduction zones and is currently investigating the structure of the Hellenic subduction zone in the Mediterranean sea. This know-how is based on a number of diploma and doctoral theses and multiple local campaigns on Crete. IGMAS, a program which has been developed at the "Freie Universität Berlin" by Prof. Dr. Götze and his staff, is used for modelling. This program uses a polyhedral approach which makes it a very efficient tool for forward modelling in three dimensions. But the polyhedral approach aggravates a geometric inversion. The geometric layout is fixed and inversion can only be calculated with respect to densities of the polyhedra in the model. To further the know-how and research work of the work group a three-dimensional geometric gravity inversion is preferable. This kind of inversion usually works with prismatic bodies instead of polyhedral bodies. Using rectangular prims to model the subsurface layout a change in geometry is simply the effect of a change in density. But compared to polyhedral approaches the effects of many prisms need to be calculated. A prismatic three dimensional model of 100 x 100 x 40 blocks yields a total of 400,000 blocks. The attraction of each block at each surface station needs to be calculated - and that is for the forward calculation only. The better the approximation of geological bodies with prisms it to be the smaller each prism has to be and the total number of prisms in the model increases fast. It is evident that the possibilities of this approach depend largely on computing power - the more the better.

During a European research project on seismic signal processing and seismic inversion a cluster of standard computers was purchased (Stoecker (2000)) with third-party funds. This cluster was used in two doctoral theses on seismic signal processing and inversion but has not been used for the last 6 months. The know-how about the cluster vanished together with the two persons who had used it before and so it seemed like the clusters’ useful life was about to end.

Tying these two loose ends together, the unused cluster and the needed computing power for three dimensional gravity inversion, seemed to be logical - the idea for the thesis at hand was born. The aim of the thesis is to reactivate the cluster and to provide a first step towards the inversion. This step is composed of a parallel program for the forward calculation of three
dimensional gravimetric models using a rectangular prism approach. Later on the program will be used in an inversion program, which is planned to be part of a doctoral thesis within a subproject of the current "Sonderforschungsbereich (SFB) 526 - Rheology of the Earth". The results of the program are to be validated with results from IGMAS using simple synthetic models. The program has to be scalable to adapt to possible changes of the cluster, it has to be reusable in the inversion and it should be optimized for speed and memory usage.

1.1 Structure of the thesis

Chapter 2 gives a short review of the fundamental formulas of gravimetrics and presents an overview of gravimetric forward modelling. Most forward algorithms fall into one of three categories and these three categories are presented. Two modelling algorithms are covered in greater detail others are described briefly.

Chapter 3 focuses on the abilities, mechanisms and handling of the developed program. The algorithm that will be used in the program is discussed in greater detail. A short introduction to the used PC cluster is given. Limitations of the program and the cluster are shown.

Chapter 4 shows some results of synthetic models and compares them to results of the same model from IGMAS, a commercial modelling software, developed at the "Free University Berlin" by Prof. Hans-Jürgen Götze and his staff.

Chapter 5 sums up the work done in this thesis and discusses future possibilities.

The appendices give necessary information for interested software developers. The setup of the PC cluster is explained in great detail. A tutorial to parallel C++ programming using the expanded numerical possibilities of the Matlab C++ library is presented and the developed program is examined and explained. This is meant as support for possible future additions to the program and for the maintainance of the cluster.
Chapter 2

Geophysical Gravity Modelling

At first a short summary of the theoretical framework of gravity is presented. Building on this foundation an introduction to gravimetric modelling is given and three different categories for forward modelling approaches are presented. These approaches are a) the right rectangular prism, b) the polygonal prism and c) the polyhedron. Finally two modelling algorithms are featured in greater detail. The first one discussed is a polyhedral approach after Goetze (1984). This approach is used in the modelling software IGMAS. The second algorithm is a right rectangular prism approach after Nagy (1966). His algorithm is used in the developed program, which will be discussed in chapter 3.

2.1 Fundamentals of Gravimetrics

Newton’s law of gravitation states that masses attract each other, the strength of the force of attraction is inverse proportional to the squared distance between the masses and is proportional to the product of the masses themselves. A constant \( f \) (sometimes denoted \( G \)) is introduced - this is the gravitational constant. Its value is \( f = (6,67259 \pm 0,00085) \times 10^{-8} \text{cm}^3\text{g}^{-1}\text{s}^{-2} \) in cgs notation. The traditional notation for the gravitational constant is cgs and not SI. Given two point masses (\( m \) and \( M \), see figure 2.1) at a distance \( R \) Newton’s law takes the following form

\[
F(R) = f \frac{Mm}{R}
\]  

(2.1)

Figure 2.1: Newtonian gravity with two 'Point Masses'
The conservative gravitational field of a point mass \( M \) at a point at \( \mathbf{R} \) is:

\[
g(\mathbf{R}) = f \frac{M}{|\mathbf{R}|} \tag{2.2}
\]

with a corresponding potential:

\[
\phi = f \frac{M}{|\mathbf{R}|} \tag{2.3}
\]

If the mass has a finite volume the potential at \( \mathbf{r} \) can be expressed via integration over the volume of the density distribution

\[
\phi(\mathbf{r}) = f \int_{V_0} \frac{\rho(\mathbf{r}')}{|\mathbf{r} - \mathbf{r}'|} dV \tag{2.4}
\]

where \( \rho(\mathbf{r}') \) is the density of the volume element \( dV \) at \( \mathbf{r}' \). Formula 2.4 is known as "Newtonian Gravitation". The relation between the gravitational field and its potential is given by

\[
g(\mathbf{r}) = -\Delta \phi(\mathbf{r}) \tag{2.5}
\]

Formulas 2.1 ... 2.4 form the fundamentals of geophysical gravitational theory, everything else is derived from these basic formulas.

### 2.2 Gravimetric Modelling

With the analytical formulas from section 2.1 it is possible to calculate the effect of a given density distribution. But what is really wanted is to infer the real density from the observed gravity, this we denote the "inverse problem" as opposed to the "forward calculation" which yields the gravity effect of a given density distribution. Due to the nature of the gravitational field many equally valid solutions to the same inverse problem exist. The same given gravity effect can be produced by different density distributions and boundary conditions are needed to restrict the model as much as possible. One obvious approach is to calculate the gravity for a given density distribution (which incorporates all known facts) and to compare it with the observed gravity. If both do not coincide, adapt the density distribution and recalculate the gravity effect until the result fits the observed gravity. Suitable programs, like IGMAS, are available for this task. This method has two major disadvantages. First the result depends on the first model chosen and on the person modelling. Two different modellers may come up with totally different and equally valid solutions to the the same problem, therefore results are not objective. The second drawback is that this manual modelling is very man-power intensive and relies strongly on the experience of the persons involved. It would be much better to have a
program taking two arguments - a) observed gravity and b) a starting model which incorporates known geological data as boundary conditions - and to adapt the starting model automatically in a repeatable way until the observed gravity fits the calculated gravity. This is the goal and this thesis tries to be a first step in this direction. Some of the questions and problems at hand for an inversion program are:

- Which modelling algorithm (now called “forward algorithm”) should be used?
- Which inversion algorithm should be used?
- How should a usable data-interface be implemented? Especially: should an existing file-format (like IGMAS) be used for compatibility or should a new file format be defined?
- Would a command-line user interface be acceptable - or would a graphical user interface be needed?
- Which are the hardware requirements for a program like this?
- Which programming-language would best suit the given problem?

Before looking into the program specific questions in chapter 3 a review of some of the more popular approaches to the forward problem will be given.

**Forward Algorithms**

Forward algorithms calculate the gravity effect of a subsurface body on a point at the surface. Two cartesian coordinate systems are introduced as helpers for the following calculations. First the body coordinates are \((\xi, \eta, \zeta)\) and the surface (field) coordinates are \((x, y, z)\). The vertical axes are positive downward, the horizontal axes are right-handed. An arbitrarily shaped three dimensional body with density \(\rho(\xi, \eta, \zeta)\) has the gravity effect on an observation point \((x, y, z)\) (see figure 2.2).

\[
g(x, y, z) = -f \int_{Vol} \rho(\xi, \eta, \zeta) \frac{z - \zeta}{r^3} \, d\xi \, d\eta \, d\zeta
\]

(2.6)

Where \(f\) is the gravitational constant and \(r\) is:

\[
r = \sqrt{(x - \xi)^2 + (y - \eta)^2 + (z - \zeta)^2}
\]

(2.7)

Formulas for sphere or cylinder can be used to approximate calculated gravity with measured gravity but geological situations with partially defined stratigraphy and drilling require more complex approaches. Most approaches fall into one of three popular categories, the stack of right rectangular prisms, the stack of polygonal prisms and the polyhedron. Some special algorithms limit themselves to work with surface gravity observations where the observed point is generally located outside the body. But when calculating the gravity within a drillhole or a tunnel the formula has to work on the inside of the body, as well.
CHAPTER 2. GEOPHYSICAL GRAVITY MODELLING

6

Figure 2.2: Geometric layout for the calculation of the gravity effect of a right rectangular prism model

The Right Rectangular Prism

A volume of mass can be approximated by a collection of rectangular prisms. Each prism is taken to have constant density and with small enough prisms any shape can be reproduced. Using the principle of superposition the gravitational anomaly of the body at any point can be derived from the summed effects of all prisms (see figure 2.2).

Adapting formula 2.6 for a rectangular parallelopiped with sides parallel to the \(x, y, z\) axes and with uniform density \(\rho\) extending from \(\xi_1 \leq \xi_2, \eta_1 \leq \eta_2, \zeta_1 \leq \zeta_2\) the vertical attraction due to this body at the point \((x, y, z)\) is

\[
g = -f \rho \int_{\xi_1}^{\xi_2} \int_{\eta_1}^{\eta_2} \int_{\zeta_1}^{\zeta_2} \frac{z - \zeta}{r^3} \, d\xi \, d\eta \, d\zeta
\]

(2.8)

Formula 2.8 has been derived by many researchers (see Li & Chouteau (1998) for a historical overview). According to them Sorokin (1951) derived the following form:

\[
g = -f \rho \sum_{i=1}^{2} \sum_{j=1}^{2} \sum_{k=1}^{2} \mu_{ijk} \left[ x_i \ln(y_j + r_{ijk}) + y_j \ln(x_i + r_{ijk}) + z_k \arctan \frac{z_k r_{ijk}}{x_i y_j} \right]
\]

(2.9)
where \( x_i = x - \xi_i, \quad y_i = y - \eta_i, \quad \) and \( z_k = z - \zeta_k, \) and
\[
r_{ijk} = \sqrt{x_i^2 + y_j^2 + z_k^2}, \quad \mu_{ijk} = (-1)^i(-1)^j(-1)^k
\]

Nagy (1966) carried out the integration in formula 2.8 in a different way then Sorokin using arcsine functions instead of arctangent functions:

\[
g = -f \rho \sum_{i=1}^{2} \sum_{j=1}^{2} \sum_{k=1}^{2} \mu_{ijk}
\]

\[
\cdots \left[ x_i \ln (y_j + r_{ijk}) + y_i \ln(x_i + r_{ijk}) - z_k \arcsin \frac{y_j^2 + z_k^2 + y_j r_{ijk}}{(y_j + r_{ijk})\sqrt{y_j^2 + z_k^2}} \right]
\] (2.10)

The Right Polygonal Prism

The right rectangular prism offers a very straightforward approach but suffers from the handicap, that geological bodies are often too difficult to be modeled with right rectangular blocks. And if two adjacent prisms have the same density there is no need to include their mutual interface in the calculation - but again the right rectangular prism approach does not account for this situation. Always the effect of each block is calculated.

Talwani & Ewing (1960) described an approach where the body is approximated by a stack of infinitely thin, horizontal laminas. The shape of each lamina is a polygon. For this to properly work it is necessary to develop a formula for the gravity effect of a thin polygon. Talwani and Ewings formulas, which heavily draw on transcendental functions, have been improved by different researchers. Noteworthy is the contribution of Plouff (1976) who showed that the gravity field within or near a lamina cannot be accurately determined by the lamina method.

Therefore Plouff altered the representation to work with layers of finite thickness, vertical sides and polygonal top and bottom surfaces. The geometric layout is shown in figure 2.3. For the \( i \)th edge subscript \( k \) in \( R_{kj} \) refers to the corner of that edge and the subscript \( j \) refers to the face of the prism. The gravity effect of this polygon is expressed as a summation of the effects of each individual edge of the \( n \)-sided prism:

\[
g = -f \rho s_m \sum_{i=1}^{n} \left[ s_p A(z_2 - z_1) + z_2 \left( \arctan \frac{z_2 d_1}{R_{12}} - \arctan \frac{z_2 d_2}{R_{22}} \right) - z_1 \left( \arctan \frac{z_1 d_1}{R_{11}} - \arctan \frac{z_1 d_2}{R_{21}} \right) - P \ln \left( \frac{R_{22} + d_2 R_{11} + d_1}{R_{11} + d_1 R_{21} + d_2} \right) \right]
\] (2.11)
Figure 2.3: Geometric layout for the calculation of the gravity effect of a polygonal Prism. Top view of a single side of the polygon (b) and front view of a complete polygon (a) (After Plouff (1976)).

$P$ is the perpendicular distance of the observation point (located at the origin) to a vertical side of the polygon:

$$P = \frac{x_1y_2 - x_2y_1}{\Delta s}$$

$$\Delta s = \sqrt{(\Delta x)^2 + (\Delta y)^2} = |d_1 - d_2|$$

$$A = \arccos \frac{x_1x_2 + y_1y_2}{r_1r_2}$$

The symbol $s_m$ is either $-1$ or $+1$ depending on whether the mass of the prism is below or above the computation point. Xiong Li and Michel Chouteau validated the above formulas and compared results for the standard cubic model and a vertical cylinder with results from formula 2.9. Very small difference in the results were found and were attributed to rounding errors in the FORTRAN single precision codes.

The Polyhedron

Both above mentioned methods approximate a geological body by a stack of prisms. But it is desireable to compute the gravity effect of any complex three-dimensional body without the need to approximate it. Therefore the analytical representation of the gravity effect of polyhedral bodies has been researched by Paul (1974), Barnett (1976), Goetze & Lahmeyer (1988) and others (an overview of these and more solutions can be found in Li & Chouteau (1998)).

Most derivations consist of the same steps but result in different analytical expressions. First the gravity effect at an observation point caused by a homogenous polyhedron is represented as a
volume integral:

\[ g = f \rho \int_{\text{Vol}} \frac{\partial}{\partial z} \left( \frac{1}{r} \right) \, dv \]  

(2.12)

with \( r \) the distance between the computation point and the volume \( dv \). This volume integral is then transformed into a surface integral by application of Gauss’ divergence theorem:

\[ g = f \rho \oint \cos(n, z) \frac{1}{r} \, ds \]  

(2.13)

For a polyhedron the surface integral can be expressed as a sum of the contributions of each planar polygonal face \( S_i \) where the cosine term in formula 2.13 represents the outward normal, \( N_i \), of the element with respect to the \( z \)-axis.

\[ g = f \rho \sum_{i=1}^{m} \left[ \cos(n_i, z) \int_{S_i} 1 \frac{1}{r} \, ds \right] \]  

(2.14)

with facets \( S_i \) from \( i = 1 \ldots m \), \( m = \text{total number of facets in the polyhedron} \). The next step is to transform into a new cartesian coordinate \((x', y', z')\) system where the outward normal of facet \( S_i \) is concordant with the new \( z' \)-direction (see figure 2.5). The final step is to convert the surface integral from formula 2.14 into a line integral of a polygon limiting the facet \( S_i \). The numerical result is conceived by inserting the limits of integration, which are the coordinates of the vertices of the polygon.

Goetze & Lahmeyer (1988) and Goetze (1984) developed a numerical solution for formula 2.12 which follows the above mentioned steps and is presented in detail because this algorithm is used in their program IGMAS which will be used to compare results from the developed program. For easier reference their notation is not changed - the geometric layout and the involved components are shown in figure 2.4. According to their notation, the attraction of a polyhedron of homogenous density \( \rho \) at a station \( P \) is based on the calculation of potential \( U(P) \):

\[ U(P) = f \int_{\text{Poly.}} \frac{1}{R} \, dm \]  

(2.15)

with \( dm = \rho \, dx \, dy \, dz \) and \( R \) the distance between the station \( P \) and \( dm \). The vertical component of the gravity at the station, \( g_z(P) \), is obtained by taking the derivative of the potential with respect to the \( z \)-component. This partial derivative can be reformulated into a surface integral:

\[ \frac{\partial U}{\partial z}(P) = g_z(P) = f \rho \oint \cos(n, z) \frac{1}{R} \, dS \]  

(2.16)
where the integral has to be calculated for the whole polyhedron surface and the cosine term determines the orientation of the surface element \( dS \) with respect to the cartesian coordinate system. For any polyhedron surface, \( S_j, \cos(m_j, z) \) is constant and therefore the gravity effect of a polyhedron can be expressed via superposition of the effects of all its surfaces:

\[
g(P) = f \rho \sum_{j=1}^{m} \cos(n_j, z) \left[ \oint_{S_j} \frac{1}{R} dS_j \right] \tag{2.17}
\]

Götze and Lahmeyer then transform the coordinate system for each surface (see figure 2.4 for the geometry), so that the new x-axis, \( x' \), is parallel to \( V_1V_2 \), the new z-axis, \( z' \), is parallel to the outward normal of the surface and the transformed y-axis, \( y' \) is chosen to be orthogonal to \( x' \) and \( z' \) (\( (x', y', z') = \mathcal{T} \mathcal{M}(x, y, z) \) and \( (x, y, z) = \mathcal{T} \mathcal{M}^T(x', y', z') \)):

\[
\mathcal{T} \mathcal{M} = \begin{bmatrix}
\alpha_1 & \alpha_2 & \alpha_3 \\
\beta_1 & \beta_2 & \beta_3 \\
\gamma_1 & \gamma_2 & \gamma_3
\end{bmatrix}
\tag{2.18}
\]

The surface integral now needs to be transformed into a line integral via polygon \( P_j \) limiting the surface \( S_j \) (see figure 2.5 for the used syntax):

\[
g(P) = f \rho \sum_{j=1}^{m} \cos(n_j, z) \left[ \oint_{P_j} \frac{h_j}{r^2} \sqrt{\left(PP_j^2 + r^2\right)} dp_j + 2\pi PP_j^\perp \delta \epsilon \right] \tag{2.19}
\]
with \( \delta = \begin{cases} 0 & \text{if } P^* \not\in S_j \\ 1 & \text{if } P^* \in S_j \end{cases} \)

Using \( k_j \) as the number of the sides in the current polygon the line integral in formula 2.19 can be evaluated to:

\[
I = \sum_{i=1}^{k_j} \left[ h_{j,i} \int_{a_{j,i}}^{b_{j,i}} \frac{\sqrt{PP^*_j + h_{j,i}^2 + s_{j,i}^2}}{h_{j,i}^2 + s_{j,i}^2} \, ds_{j,i} \right] \tag{2.20}
\]

with \( k_j \) number of sides of polygon \( P_j \), \( a_{j,i}, b_{j,i} \) limits of integration. The integral in 2.20 is of the general form:

\[
I = \int \frac{\sqrt{H^2 + s^2}}{h^2 + s^2} \, ds \tag{2.21}
\]

with \( H > |h| \) and \( H^2 = PP^*_j^2 + h_{j,i}^2 \) and \( s^2 = s_{j,i}^2 \). Substituting

\[
\dot{s} = \frac{H}{2} \left( \sqrt{u} - \frac{1}{\sqrt{u}} \right) \quad \text{and} \quad \frac{ds}{du} = \frac{du}{du} \left[ \frac{H}{2} \left( \sqrt{u} - \frac{1}{\sqrt{u}} \right) \right] = \frac{H}{4} \left( u^{-\frac{3}{2}} + u^{-\frac{1}{2}} \right)
\]
leads to:

\[ I = \int \frac{H(u+1)}{h^2 + \frac{H^2}{4u}(u-1)^2} \frac{H}{4}(u^{-\frac{1}{2}} + u^{-\frac{3}{2}})du \]  \hspace{1cm} (2.22)

which can be solved to:

\[ I = \frac{1}{2} \left[ \int \frac{u \, du}{z} + \int \frac{2 \, du}{z} + \int \frac{du}{u} \right] \]  \hspace{1cm} (2.23)

with \( z = u^2 - 2u\lambda + 1 \) and \( \lambda = 1 - \frac{2h^2}{H^2} \).

Using case differentiation the three integrals, \( A, B \) and \( C \), in formula 2.23 can be solved (see Goetze (1984) for the case differentiation). Finally the numerical formula for the attraction of a polygon can be written as:

\[ g(P) = f \rho \left[ \sum_{j=1}^{m} \cos(n_{j,z}) \left[ \sum_{i=1}^{k} h_{j,i} \left[ T_1 + \frac{\left| PP_j^* \right|}{h_{j,i}} (T_2 - T_3) \right] + 2\pi \left| PP_j^* \right| \delta_\epsilon \right] \right] \]  \hspace{1cm} (2.24)

with

\[ T_1 = \ln \frac{b_{j,i} + \overline{PV_{j,i-1}}}{a_{j,i} + \overline{PV_{j,i}}} \]
\[ T_2 = \arctan \frac{r_{j,i-1}^2 + b_{j,i} \overline{PV_{j,i-1}}}{\left| PP_j^* \right| |h_{j,i}|} \]
\[ T_3 = \arctan \frac{r_{j,i}^2 + a_{j,i} \overline{PV_{j,i}}}{\left| PP_j^* \right| |h_{j,i}|} \]

and

\[ \delta = \begin{cases} 0 & , \text{if } P^* \not\in S_j \\ 1 & , \text{if } P^* \in S_j \end{cases} \]

Li & Chouteau (1998) and Polhanka (1988) pointed out that different versions of the formulæ should yield equivalent results but that in real cases there are differences. Mathematically equivalent formulæ may produce different numerical results and they differ also in the number of intermediate computations, the ability to avoid singularities and in the efficiency with which they can be programmed.
Figure 2.6: Polyhedral assembly using triangulation as used in IGMAS (after Goetze (1984))

**Polyhedral Assembly**  A polyhedral body is made up of many polygonal planes. In order to assemble the body a system to name and position the different planes needs to be developed. The polyhedron is organized using the coordinates of the vertices and by the assembly of the vertices to form the facets of the polyhedron. Okabe (1979) suggested a way in which all vertices are numbered and the vertex coordinates are stored by number. Each facet is defined by the vertex numbers in anti-clockwise order about the outward facet normal. This is very time-consuming. And how to fill all space in the model with multiple polyhedra without gaps or overlaps is another complex question. Goetze & Lahmeyer (1988) provided a practical approach which is used in their program IGMAS. The model is divided into parallel planes. The user defines the geological structures in each planar section and the polyhedral bodies are automatically assembled between the sections using a triangulation algorithm. This is shown in figure 2.6.
2.3 The Nagy Algorithm

As the Nagy algorithm (Nagy (1966)) was chosen for the forward modelling this algorithm and its derivation will be presented in more detail - see figure 2.7 for the geometric layout used. This algorithm is a classical member of the right-rectangular prism approach and is directly derived from formula 2.8.

The magnitude of attraction of an elementary mass on a unit mass is given by:

$$\Delta F = f \rho \frac{\Delta v}{r^2}$$ \hspace{1cm} (2.25)

With $\Delta v$ the volume element at $r$, $f$ is the gravitational constant and $\rho$ the density. The vertical component $F_z$ of the attraction is of interest and this can be expressed with the angle between the vertical axis $z$ and the distance vector $r$, denoted $\gamma$, by integration of $\Delta F \cos \gamma$ over the volume:

$$F_z = f \rho \int_V \frac{d\gamma}{r^2} \cos \gamma = f \rho \int_V \frac{zd\gamma}{r^3}$$ \hspace{1cm} (2.26)

Using the geometry from figure 2.7 equation 2.26 can be written as:

$$F_z = f \rho \int_{x_1}^{x_2} dx \int_{y_1}^{y_2} dy \int_{z_1}^{z_2} \frac{zd\gamma}{\sqrt{x^2 + y^2 + z^2}}$$ \hspace{1cm} (2.27)
The next three steps are to integrate with respect to \( z \) (yielding \( I_1 \)) then to integrate \( I_1 \) with respect to \( y \) (yielding \( I_2 \)) and finally to integrate \( I_2 \) with respect to \( x \) (yielding \( I_3 \)):

\[
I_1 = \int \frac{zdz}{\sqrt{x^2 + y^2 + z^2}}
\]
\[
= -\frac{1}{\sqrt{x^2 + y^2 + z^2}}
\]

\[
I_2 = \int I_1 dy = \int \frac{dy}{\sqrt{x^2 + y^2 + z^2}}
\]
\[
= \ln(y + \sqrt{x^2 + y^2 + z^2})
\]

\[
I_3 = \int I_2 dx = \int \ln(y + \sqrt{x^2 + y^2 + z^2}) dx
\]
\[
= x \ln(y + \sqrt{x^2 + y^2 + z^2}) - \int \frac{x^2 dx}{(y + \sqrt{x^2 + y^2 + z^2})\sqrt{x^2 + y^2 + z^2}}
\]

Substituting \( u = y + \sqrt{x^2 + y^2 + z^2} \) leads to

\[
x^2 = (u - y)^2 - y^2 - z^2
\]

\[
dx = \frac{(u - y)dy}{\sqrt{(u - y)^2 - y^2 - z^2}}
\]

and thus the integral \( I = \int \frac{x^2 dx}{(y + \sqrt{x^2 + y^2 + z^2})\sqrt{x^2 + y^2 + z^2}} \) in \( I_3 \) can be expressed as:

\[
I = \int \frac{x^2 dx}{(y + \sqrt{x^2 + y^2 + z^2})\sqrt{x^2 + y^2 + z^2}} = \int \frac{\sqrt{u^2 - 2uy - z^2} - y \ln(u - y + \sqrt{u^2 - 2uy - z^2}) - z \arcsin\left(-\frac{uy - z^2}{u\sqrt{z^2 + y^2}}\right)}{u} du
\]

Transforming back to the original variables the integral \( I \) is:

\[
I = x - y \ln(x + \sqrt{x^2 + y^2 + z^2}) - z \arcsin\left(\frac{z^2 + y^2 + y\sqrt{x^2 + y^2 + z^2}}{y + \sqrt{x^2 + y^2 + z^2}\sqrt{y^2 + z^2}}\right)
\]

The last step is to substitute the limits of integration:

\[
F_z = \int \rho \left| x \ln(y + r) + y \ln(x + r) - z \arcsin\left(\frac{z^2 + y^2 + yr}{(y + r)\sqrt{y^2 + z^2}}\right) \right| _{z_1}^{z_2} dz
\]
The solution is:

\[ F_z = f \rho \sum_{i=1}^{24} T_i \]  

(2.34)

with:

\[
T_{01} = +x_2 \ln(y_2 + r_{221}) ; \quad T_{02} = -x_1 \ln(y_2 + r_{121}) \\
T_{03} = +y_2 \ln(x_2 + r_{221}) ; \quad T_{04} = -y_2 \ln(x_1 + r_{121}) \\
T_{07} = -x_2 \ln(y_1 + r_{211}) ; \quad T_{08} = +x_2 \ln(y_1 + r_{111}) \\
T_{09} = -y_1 \ln(x_2 + r_{211}) ; \quad T_{10} = +y_1 \ln(x_1 + r_{111}) \\
T_{13} = -x_2 \ln(y_2 + r_{222}) ; \quad T_{14} = +x_1 \ln(y_2 + r_{122}) \\
T_{15} = -y_2 \ln(x_2 + r_{222}) ; \quad T_{16} = +y_2 \ln(x_1 + r_{122}) \\
T_{19} = +x_2 \ln(y_1 + r_{212}) ; \quad T_{20} = -x_1 \ln(y_1 + r_{112}) \\
T_{21} = +y_1 \ln(x_2 + r_{212}) ; \quad T_{22} = -y_1 \ln(x_1 + r_{112}) \\
T_{05} = -z_1 \arcsin \left( \frac{z_2^2 + y_2^2 + y_2 r_{221}}{y_2 + r_{221} \sqrt{y_2^2 + z_1^2}} \right) \\
T_{06} = +z_1 \arcsin \left( \frac{z_2^2 + y_2^2 + y_2 r_{121}}{y_2 + r_{121} \sqrt{y_2^2 + z_1^2}} \right) \\
T_{11} = +z_1 \arcsin \left( \frac{z_1^2 + y_1^2 + y_1 r_{211}}{y_1 + r_{211} \sqrt{y_1^2 + z_1^2}} \right) \\
T_{12} = -z_1 \arcsin \left( \frac{z_1^2 + y_1^2 + y_1 r_{111}}{y_1 + r_{111} \sqrt{y_1^2 + z_1^2}} \right) \\
T_{17} = +z_2 \arcsin \left( \frac{z_2^2 + y_2^2 + y_2 r_{222}}{y_2 + r_{222} \sqrt{y_2^2 + z_2^2}} \right) \\
T_{18} = -z_2 \arcsin \left( \frac{z_2^2 + y_2^2 + y_2 r_{122}}{y_2 + r_{122} \sqrt{y_2^2 + z_2^2}} \right) \\
T_{23} = -z_2 \arcsin \left( \frac{z_2^2 + y_2^2 + y_2 r_{212}}{y_2 + r_{212} \sqrt{y_2^2 + z_2^2}} \right) \\
T_{24} = +z_2 \arcsin \left( \frac{z_2^2 + y_2^2 + y_2 r_{112}}{y_2 + r_{112} \sqrt{y_2^2 + z_2^2}} \right) \\
\]  

(2.35)

where:

\[ r_{111} = \sqrt{x_1^2 + y_1^2 + z_1^2} ; \quad r_{112} = \sqrt{x_1^2 + y_1^2 + z_2^2} \]
Please note that the original form of $T_{01} \ldots T_{24}$ as published by Nagy contains a few errors which have been corrected in the above listing. Chapter 3 contains a discussion of the calculation of the formulas 2.35.

2.4 Conclusion

The fundamentals of geophysical gravimetrics have been introduced and three different approaches for forward modelling have been presented. The right rectangular prism is the most straightforward approach but it is ponderous to approximate a geological body with prisms. The polygonal prism is closer to geological situations but is cumbersome to assemble. The closest to reality is the polyhedron, but this is also the most complex to calculate. The program IGMAS is a good example for the power of the polyhedron approach. If the forward algorithm is to be used in an inversion the right rectangular prism is usually chosen. The geometric inversion can be done by simply changing densities and restricting information is relatively easy to incorporate into the model. This will get evident in chapter 3. That is why a right rectangular prism approach after Nagy has been chosen.
Chapter 3

Nagy’s Algorithm in Action

This chapter focuses on the implementation of the Nagy algorithm introduced in chapter 2. At first the algorithm itself is discussed, some obstacles are pinpointed and the suitability for later inversion is shown. Thereafter one possible approach for parallelization is presented and evaluated.

3.1 Nagy’s Algorithm

A simple "four blocks, one station" model is used to clarify how the algorithm can be used to get correct results for the force due to the gravimetric attraction of the four blocks on the station. A top view of the four blocks is shown in figure 3.1 A. The sum in formula 2.34 is the geometric effect, $GF$, of a block on a station. It is independent of the block density:

$$GF = \sum_{i=1}^{24} T_i$$  \hspace{1cm} (3.1)

The extension of the blocks in $z$ is taken to be $z_1 = 0$ and $z_2 = 1$ for all four blocks. Each block is defined by the 7 parameters $x_1, x_2, y_1, y_2, z_1, z_2, \rho$. For the calculation of the geometric effect the density $\rho$ is not of interest and is omitted for this example. The 6 parameters left define the geometry of each block, they are listed in table 3.1 for all four blocks of this simple model.

The algorithm calculates the effect of each block at the station. The easiest way to do this is to shift the station to the origin of the coordinate system and to adapt the block coordinates respectively. This yields the block layout shown in table 3.2.

If the values from table 3.2 are entered into formula 2.35 and the results from the 24 partial calculations for each block are summed this will result in the "geometric effect" of each block at the station. Multiplication with the gravity constant and the respective density of each block yields the attraction of each block at the station in $z$-direction and finally the sum of all gravitational effects results in the final gravitational attraction at the station. This will now be shown with the
Figure 3.1: Top views of a four block, one station right rectangular prism model. Surface station is above the intersection of the blocks (A), surface station is above one block (B).

Table 3.1: Block and station layout after figure 3.1 (A)

<table>
<thead>
<tr>
<th>Block Number</th>
<th>x1</th>
<th>x2</th>
<th>y1</th>
<th>y2</th>
<th>z1</th>
<th>z2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Station Coordinates: 1, -1, -1, 0

above values starting with the calculation of formula 2.35. The results for the four blocks are shown in table 3.3. The many zero and "NaN" result from the zeros in the input - neither \( \ln(0) \) nor division by zero are defined. So the zero values have to be approximated by values close to zero. The results of this calculation are shown in table 3.4. The sum of all 24 partial calculations for each block yields the geometric effect of that block on the station at the origin. As all blocks have the same dimensions and are grouped symmetrically around the station one would expect that all blocks have the same geometric effect - but this is currently not the case. The results differ, because the algorithm works on positive numbers only. The solution is to use absolute values for the block coordinates only. Using absolute values only and calculating the geometric effect for each of the four blocks in the example results in a negative geometric effect for blocks 2 and 3. Recall that this number -the geometric effect- multiplied with the density of the block and the gravity constant is supposed to be the gravity effect of that block on the station at the origin of the coordinate system. Both the density and the gravity constant are positive numbers so that the gravity effect will be negative if the geometric effect is negative. And this can not be. But as all four blocks yield the same absolute geometric effect the solution to this is at hand. One must use the absolute value of the sum to get the correct geometric effect.

A similar but different layout is depicted in figure 3.1 B. The surface station is moved \((1, 2/0, 4)\) - shifting the station to the origin yields the coordinates for the blocks shown in table 3.5. Some
blocks extend in \( x \) and/or \( y \) from negative to positive coordinates. The partial calculations again yield incorrect results. This is because the integration in formula 2.32 has to be splitted into two parts, one from \( x_1 \) to 0 and the second from 0 to \( x_2 \), or from \( y_1 \) to 0 and 0 to \( y_2 \). In the special case that the block coordinates cross the origin in \( x \) and \( y \) the block needs to be divided into 4 subblocks. Please note that the zero values have again to be approximated by values close to zero and that the absolute values of the coordinates have to be used for the calculation like in the previous example. The total number of blocks will increase depending on the station coordinates.

For later inversion it is necessary that all stations have the same number of blocks and that the geometric effects are listed in the same order. Table 3.6 shows the so called "kernel" for this model. This is of special interest to later inversion. For each block in the model the geometric effect is calculated upon each station. In order to get the gravitational effect at the surface the only steps needed is a multiplication of each block with its corresponding density and the gravitational constant and a final summation over each block for each station. Like that different models are created by a simple change in the density of the blocks. The kernel needs to be calculated only once, for \( m \) blocks and \( n \) station it is a matrix of size \( m \times n \).
Table 3.4: Results of partial calculations of the layout shown in figure 3.1 zeros approximated by 0.001 or -0.001

<table>
<thead>
<tr>
<th>Partial Calculation</th>
<th>Result Block 2</th>
<th>Result Block 2</th>
<th>Result Block 3</th>
<th>Result Block 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>T01</td>
<td>-0.001</td>
<td>0.8814</td>
<td>0.0078</td>
<td>-0.0007</td>
</tr>
<tr>
<td>T02</td>
<td>0.0078</td>
<td>-0.0007</td>
<td>-0.001</td>
<td>0.8814</td>
</tr>
<tr>
<td>T03</td>
<td>0.0007</td>
<td>0.8814</td>
<td>0.0078</td>
<td>-0.001</td>
</tr>
<tr>
<td>T04</td>
<td>-0.006</td>
<td>-0.001</td>
<td>-0.0145</td>
<td>0.8814</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>T21</td>
<td>-1.0051</td>
<td>0.0009</td>
<td>-0.3459</td>
<td>0</td>
</tr>
<tr>
<td>T22</td>
<td>0.3473</td>
<td>0</td>
<td>-0.3119</td>
<td>0.0009</td>
</tr>
<tr>
<td>T23</td>
<td>-0.2618</td>
<td>-0.7861</td>
<td>-1.5691</td>
<td>-1.5698</td>
</tr>
<tr>
<td>T24</td>
<td>1.5691</td>
<td>1.5698</td>
<td>0.2618</td>
<td>0.7867</td>
</tr>
<tr>
<td>sum</td>
<td>0.3283</td>
<td>0.3283</td>
<td>0.5690</td>
<td>0.5690</td>
</tr>
</tbody>
</table>

Table 3.5: Block and station layout after figure 3.1 B, adapted for station coordinates

<table>
<thead>
<tr>
<th>Block Number</th>
<th>x1</th>
<th>x2</th>
<th>y1</th>
<th>y2</th>
<th>z1</th>
<th>z2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-0.2</td>
<td>1.2</td>
<td>-0.6</td>
<td>0.6</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>-0.2</td>
<td>1.2</td>
<td>0.6</td>
<td>1.6</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>-1.2</td>
<td>-0.2</td>
<td>-0.6</td>
<td>0.6</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>-1.2</td>
<td>-0.2</td>
<td>0.6</td>
<td>1.6</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Station Coordinates</td>
<td>0</td>
<td>-</td>
<td>0</td>
<td>-</td>
<td>0</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 3.6: Kernel Layout, $T_i$ according to formula 2.35

<table>
<thead>
<tr>
<th>Block 1</th>
<th>Station 1</th>
<th>Station 2</th>
<th>...</th>
<th>Station n</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\sum_{i=1}^{24} T_i$</td>
<td>$\sum_{i=1}^{24} T_i$</td>
<td>...</td>
<td>$\sum_{i=1}^{24} T_i$</td>
<td></td>
</tr>
<tr>
<td>Block 2</td>
<td>$\sum_{i=1}^{24} T_i$</td>
<td>$\sum_{i=1}^{24} T_i$</td>
<td>...</td>
<td>$\sum_{i=1}^{24} T_i$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Block m</td>
<td>$\sum_{i=1}^{24} T_i$</td>
<td>$\sum_{i=1}^{24} T_i$</td>
<td>...</td>
<td>$\sum_{i=1}^{24} T_i$</td>
</tr>
</tbody>
</table>
CHAPTER 3. NAGY’S ALGORITHM IN ACTION

3.2 Using the parallel Beowulf cluster

The calculation of the kernel for a model with many stations and blocks is very time consuming. For this reason it was chosen to calculate the kernel on a parallel cluster of "off-the-shelf" PCs. All technical details are given in appendices and will not be given here. But a schematic understanding is necessary and can be achieved without delving into computer related technogtalk.

A schematic view of the cluster is shown in figure 3.2. The computers in the cluster are networked so that they can communicate with each other. The only one having a display and keyboard is the headnode, all computing nodes are so-called "diskless" clients, which means they use the headnode's disks to load the operating system and to store data. Using a special command the same program is started on the headnode and any number of computing nodes. The program on each computing node is able to communicate with the headnode and with each computing node, although in the program used they only communicate with the headnode. Communication means that at any time during the execution of the program it is able to send and receive data, but it has to be explicitly told to send or wait for data.

The program starts at the headnode with the creation of the subsurface layout (the blocks) and the surface layout (the stations). The headnode determines how many computing nodes are available. As all nodes have two processors the program can be started twice on each computing node. The next step is to calculate how many stations need to be calculated by each node and which node will calculate the effect on which station. This is written to a file and the model layout is written to a second file. The filenames are passed to the computing nodes which load both files and parse the file with the station-to-node correlation to find out which stations have to be calculated by which computing node. Then the computing nodes loop through the relevant stations and calculate either the gravity effect of the model on that station or the kernel for that

Figure 3.2: A schematic view of the PC cluster named "Count von Count"
station, depending on whether the result is to be used in modeling or inversion. The results are saved into a file named after the station number. This parallel approach takes advantage of the fact that the gravity effect of the model on one station is independent of the gravity effect on a different station. The results can be displayed using GMT, Matlab or even IGMAS.

In a later inversion the cluster can be useful again. After the kernel for each station is calculated the headnode will analyze this kernel and create a range of different block densities. These densities can then be passed to the computing nodes to calculate the gravity effect and pass it to the headnode which will again analyze it. This loop can be continued until a reasonable result is found. The inversion has so far not been implemented.

**Limits of the Cluster** Each computing node is equipped with two processors and a total of 256 MB of memory (RAM). After the operating system is loaded the free memory is about 219 MB. The limit in model size which can be calculated is defined by these parameters, especially the RAM limits the maximum size. As the program loops through the stations and not the blocks the total number of blocks need to be restricted in a way that the used memory for the effect of the model on one station does not exceed the available memory. Otherwise the program will crash. Usually each computing node calculates the effect of two stations simultaneously using the two available processors. As these processors share the total of 219 MB free RAM each processor effectively has only about 110 MB RAM for itself. The calculation for 200000 blocks needs approximately 120 MB RAM per started job, 300000 blocks need approximately 190 MB RAM and 400000 blocks exceed the total maximum memory available and will crash the program. With up to 150000 blocks the program can be started twice on each node as less than half of the total available memory is needed. This effectively reduces the time needed for calculation by 50%.
Chapter 4

Results

In order to validate the correctness of the used algorithm and developed program the anomaly of a simple synthetic model is calculated and the result is compared to the result from IGMAS for the same model. The anomaly was calculated in two different ways with the parallel program, both calculations yield exactly the same result. The model chosen to validate the program is made up of a buried vertical shaft. The shaft dimensions are 10 m x 10 m horizontally, extending from 10 m to 100 m below the surface with an absolute density of 0 kg/m$^3$. It is covered by a 10 m thick layer of absolute density 2000 kg/m$^3$, the rest of the model has a homogenous density of 2670 kg/m$^3$. In order to calculate the anomaly, relative densities instead of absolute densities were used in both calculations with a reference density of 2670 kg/m$^3$. The two calculations differ in the setup of the model blocks, the stations at which the anomaly was calculated were the same in both cases. Stations were placed on the flat surface at a 5 m grid in $x$ and $y$ starting at the origin and extending 500 m in both directions, resulting in a total of 10,000 stations.

The first setup consists of 220,000 blocks, each block is 5 m x 5 m x 5 m large. The blocks are arranged to form a model extending from 0 m to 500 m in $x$ and $y$ and from 0 m to 110 m in $z$. The shaft and the cover are located exactly at the midpoint of the model in $x$ and $y$, that means they extend from 245 m to 255 m in both directions. Using relative densities all blocks, but the blocks in the shaft and the blocks in the cover, have a density of 0 kg/m$^3$, the cover (2 x 2 x 2 blocks) has a density of -670 kg/m$^3$ and the shaft itself (2 x 2 x 18 blocks) has a density of -2670 kg/m$^3$. A cutout of the model is shown in figure 4.1. The blocks with density 0 kg/m$^3$ do not contribute to the anomaly, but nevertheless it is justified to include these blocks even though they significantly slow down the calculation. Calculation of the anomaly on the cluster took about 6 hours. Having in mind a future inversion the blocks in the model space are needed, because there is no commonly known inversion algorithm which is able to create new blocks in the model on demand. Therefore all applicable blocks have to be included in the model right from the beginning. If one is only interested in the forward calculation without intention of a later inversion these blocks with 0 kg/m$^3$ density can be omitted, as was done in the second setup.
Figure 4.1: Cutout of the subsurface model layout and its anomaly response at the surface
The second setup reduced the model from 220,000 blocks to just 2 blocks. One block for the cover, with $x_1 = 245$ m, $x_2 = 255$ m, $y_1 = 245$ m, $y_2 = 255$ m and $z_1 = 0$ m, $z_2 = 10$ m, and one block for the shaft with the same $x$-$y$ extension as the cover and $z_1 = 10$ m, $z_2 = 100$ m. Again relative densities as in the first setup were used to calculate the anomaly at the 10,000 grid stations. Calculation on the cluster took less than 2 minutes to complete. The anomaly of both calculations is exactly the same.

The anomaly of the model was then calculated using IGMAS, the model setup consists of 2 polyhedra, one for the cover and one for the shaft itself. The same relative densities as before were used and the surface grid was chosen to match the station layout of the parallel program. In order to compare the results the parallel result from the first calculation was loaded as ”measured” gravity into IGMAS and IGMAS was used to create the surface plot of the anomaly shown in figure 4.3. The stations at the surface are not shown in this figure, obviously the anomaly is the same in both cases and therefore one can state, that the parallel program works correctly.

---

**IGMAS vs. Parallel modelling program**

---

**Figure 4.2:** Top: Cutout off the calculated anomaly of a cross-section through the shaft. Bottom: Histogram showing the difference between the result from the parallel program and IGMAS
So as to get a quantitative statement of the match between the IGMAS result and the parallel result the difference in the anomaly at each station of a cutout of a cross-section cutting through the shaft was calculated and plotted in figure 4.2. The plot shows the anomaly of the cross section for the parallel program and for IGMAS at the top and a histogram of the difference distribution between the anomaly from IGMAS and from the parallel program at the bottom.

The anomaly of the parallel program fits the anomaly calculated with IGMAS very well, a small difference exists at the minimum and is due to distinctions in the used algorithms. The difference distribution shows that of the 41 stations in this section 38 have a difference in the anomaly between 0 mGal and 0.01 mGal and 3 stations have a difference between 0.01 mGal and 0.02 mGal. The maximum anomaly is -0.26 mGal and therefore the discrepancies in the results from IGMAS and the parallel program are neglectable, especially as the resolution of gravimeters is in the same range as the divergence between the IGMAS result and the parallel result. A similar analysis of sections not cutting through the shaft show an even better consistency and have therefore not been included.
Chapter 5

Resumee

The work done for this diploma thesis can be divided into four different steps. The first step was to reactivate and setup the PC cluster. The second step was to select a forward algorithm which is suitable for a later inversion. The third step was to write a program which realizes the selected algorithm in parallel on the cluster and the last step was to validate results from this self-written forward modelling program. All fours steps were conducted successfully and the work at hand can therefore be evaluated as a complete success.

The PC Cluster  The PC cluster was reactivated and reconfigured. The reconfiguration eases maintainance and future upgrades both in software and hardware. The setup uses a combination of ”Root-over-NFS” and ”Root Ramdisk” for the diskless computing nodes. This setup allows for a single shared NFS-mounted file system for all computing nodes. The file system is located on the headnode and all software upgrades can be conducted on the headnode at runtime without the need to reboot the computing nodes. Adding new nodes is very simple, too. The MAC address of the new nodes’ network card needs to be included into the bootptab file on the headnode and the addition is complete.

The Forward Algorithm  A right rectangular prism approach following Nagy’s algorithm was chosen for forward modelling. Though the approximation of geological bodies using rectangular prisms is less close to geological reality than a polyhedral approach the right rectangular prism approach offers the possibility of a geometric inversion. Most inversion methods are based on the use of a prism approach for the forward calculation. Only inversion methods which limit themselves to a predefined geometry, eg. a sphere or a cylinder, use a polyhedral approach for the forward calculation.

The Parallel Program  The program was developed using ANSI C++ and two external libraries. The first library is the LAM/MPI C++ library which adds the means of parallel programming. The second included library is the Matlab C++ library with powerful numerical functions
and arrays. The program is object oriented, one single class was designed for the complete forward calculation. The calculation of a model made up of 220,000 blocks with 10,000 stations takes about 6 hours on the cluster.

**The Results** The results from the parallel program have been verified with IGMAS, the same model yields the same anomaly with both programs. The model used for verification is a buried vertical shaft in a homogenous environment. The anomaly has been calculated in two different ways, both yielding the same expected result. First the model was made up of two blocks only, one block with a relative density of $-470 \text{ kg/m}^3$ as the cover of the buried shaft and the second block with a relative density of $-2670 \text{ kg/m}^3$. The second anomaly was calculated with a model made up of 220,000 blocks. All blocks but the cover blocks and the shaft blocks had a relative density of $0 \text{ kg/m}^3$, the cover blocks had a relative density of $-470 \text{ kg/m}^3$ and the shaft blocks had a relative density of $-2670 \text{ kg/m}^3$. These two model setups are basically the same, the first model description is suitable for forward modelling only, whereas the second model description is also suitable for inversion, where many blocks are needed to describe the model. These blocks are needed for inversion, because the inversion is based on the change of density of the blocks and if the model contains only two blocks, as in the first model setup, then only these two blocks can be inverted. As the anomaly and not the attraction is of interest all models used relative rather than absolute densities.

**Future Perspectives**

The developed program by itself can be used as a 3-dimensional forward modelling solution, but as such it does not contain the power and comfort of other readily available programs like IGMAS. Probably the greatest impediment is the lack of a graphical user interface to easily create input models for the forward calculation. The power of the program lies in the combination of an inversion-suitable algorithm with a parallel cluster of cheap computers. This combination is, as far as I know, unique and if an inversion will be programmed based on this solution this will probably greatly enhance gravimetric research at the “Applied Geophysics” work-group. Inversion is extremely computing power intensive and it is already foreseeable that the cluster will have to be upgraded. Currently the amount of RAM in the computing nodes defines the limit in model size and an upgrade is highly recommended. The addition of new PCs to the cluster would speed up the computation but is currently not urgent. With respect to inversion the forward algorithm can probably be improved in speed and memory usage by an intensive analysis of the kernel matrix. If it would be possible to convert the kernel from a full to a sparse matrix, memory on the computing nodes could be reduced without impact on the precision of the calculation.

**Possible Inversion Approaches**

Although implementing inversion is not part of the thesis some short thoughts about inversion will be presented here to hint at some possible future extensions of the written program.
One of the most important factors in inversion is the constraint of possible models. If no constraints are used both the total number of models to calculate and the number of valid solutions are out of scope for even the largest cluster of computers. These constraints are added as a priori information into the starting model whenever geological or density information is available (Green (1975)). Some approaches constrain themselves to a given geometry, like a vertical cylinder (Murthy & Swamy (1995)) or to interfaces (Rao et al. (1999)).

As block sensitivity decreases with depth some authors suggest to counteract this effect by weighting block results with an inverse function of depth (see Li & Oldenburg (1996), Li & Oldenburg (1998), Pilkington (1997)). Last & Kubik (1983) found a compact solution using a minimum volume constraint. A global constraint called "smoothing" was developed by Li & Oldenburg (1996) to find a minimum structure.

A very promising approach is described by Boulanger & Chouteau (2001). They propose the application of a combination of multiple constraints. These constraints include minimum distance, flatness, smoothness and compactness constraints, combined by a Lagrangian formulation. In addition to using the multitude of constraints they also analyze the sensitivity coefficients with respect to symmetry which leads to a decrease of the memory needed for the calculation. Using this approach they were able to invert real gravity data for the Rouyn-Noranda (Quebec) mining camp and the solution is in agreement with known lithological contacts at the surface and known geology. This approach is based on a rectangular prism algorithm for the forward calculation and it looks like it would be possible to adapt the approach for parallel computing.
Bibliography


BIBLIOGRAPHY


Appendix A

Cluster Setup and Maintainance

From March 2001 until May 2001 we restructured our beowulf cluster setup. This was necessary because we purchased a new headnode. The old setup used a traditional 'Diskless Root over NFS' setup with one root filesystem on the headnode for every computing node and a shared userspace on the headnode. Performance monitoring showed the network to be the current bottleneck and so we looked for a cluster setup with minimal network load yet we still wanted to use our diskless clients. Node management on the headnode was rather labor intensive due to the unique root filesystem for every node and so we looked for a solution to ease maintainance.

Incorporating all aspects we developed a new approach for our cluster setup and managed to come up with a solution eliminating all disadvantages of the old setup. The general idea is to split the root filesystem across a ramdisk image and various NFS mounts and to automate node specific configuration at boottime. This paper documents our solution; Linux and networking knowledge is required to understand what we did, why we did it and how we did it.

This is document version 1.0, written by Toni St"ocker and Dror R"ocher. Axel Tillmann helped proof reading. Bochum, May 2001.
A.1 Selfmade Cheap PC Cluster

A.1.1 ’Count von Count’ Setup

We have chosen to buy 18 off-the-shelf dual SMP PCs for the cluster and to use switched fast ethernet as the network, making the whole cluster legacy free, compatible to future extensions and cutting down on expenses compared to either GigabitEthernet or Myrinet based solutions. Of the 18 PCs only one is fully equipped with harddisks, floppy disk and CD-ROM, this one is defined as the headnode and is acting as a server for the remaining 17 almost diskless nodes (the only disk installed in the nodes is a floppy disk which is only used at the beginning of the boot process). For better network traffic control and enhanced network performance we have equipped the server with a 4 port FastEthernet network information card (NIC). As all nodes and the server are connected to the same switch some subnetting needs to take place in order to use all 4 ports of the server NIC. Bonding is not an option because it requires equal NIC ports on all attached computers and multiple switches or VLAN-capable switches. The used equipment is listed in table A.1.

The cluster uses private IP addresses (Rekhter et al. (1996)) in the range 192.168.0.0 using a standard class C netmask on the server (24 Bit = 255.255.255.0) and a supernetted class B netmask (16 Bit = 255.255.0.0) on the nodes. The effect is that the nodes take each other and all 4 server NIC ports to be local ensuring communICation between the nodes without rout-ing through the server, whereas the server sees the nodes in 4 different networks connected to 4 different NIC ports, therefore giving each subnet 100 MBit/s full duplex connection to the server. The nodes are equally distributed across the 4 subnets (192.168.26.0, 192.168.27.0, 192.168.28.0 and 192.168.29.0) and have IP addresses starting from .1 (192.168.26.1, 192.168.26.2, ..., 192.168.27.1, 192.168.27.2, ...), the server has the .254 address in every subnet: 192.168.26.254, 192.168.27.254, 192.168.28.254, 192.168.29.254

IP address assignment is done using static BOOTP mappings, the server acts as a BOOTP-server for the nodes. The BOOTP daemon also tells the nodes to download and execute a file from the server. Using TFTP the clients downloads a netbootable image containing a kernel and a ramdisk. The ramdisk holds selected files and directories of the node root filesystem. This filesystem is not the root filesystem of the headnode but a special manually created filesystyem residing in a directory on the headnode. These files and directories are needed to boot the clients and mount the rest of the file system via NFS. All clients share the same netboot-image and the same file system facilitating maintenance and ensuring scalability.
Table A.1: Hardware for Count von Count

<table>
<thead>
<tr>
<th>Type</th>
<th>Model</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Server Hardware</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPU</td>
<td>2 x PentiumIII 700MHz</td>
<td></td>
</tr>
<tr>
<td>RAM</td>
<td>4 x 256 MB DIMM</td>
<td>totals 1 GB RAM</td>
</tr>
<tr>
<td>Mainboard</td>
<td>Gigabyte 6VXD7</td>
<td>Dual PentiumIII</td>
</tr>
<tr>
<td>VGA</td>
<td>Matrox Millenium AGP 8MB</td>
<td></td>
</tr>
<tr>
<td>NIC</td>
<td>1 x D-LINK DFE 570 TX</td>
<td>4 x 100BaseTX, Busmaster capable</td>
</tr>
<tr>
<td>Disks</td>
<td>1 x 3COM 3c905B TX</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 x Floppy</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 x IBM 9 GB U2W</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 x IBM 4 GB U2W</td>
<td></td>
</tr>
<tr>
<td>Other</td>
<td>Keyboard, Mouse, Monitor, CDRW</td>
<td></td>
</tr>
<tr>
<td><strong>Node Type 1 Hardware</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPU</td>
<td>2 x PentiumII 450MHz</td>
<td></td>
</tr>
<tr>
<td>RAM</td>
<td>1 x 256 MB DIMM</td>
<td></td>
</tr>
<tr>
<td>Mainboard</td>
<td>Gigabyte P2DBX</td>
<td>Dual PentiumIII</td>
</tr>
<tr>
<td>VGA</td>
<td>AGP, El Cheapo, 4MB</td>
<td></td>
</tr>
<tr>
<td>NIC</td>
<td>1 x 3COM 3c905B TX</td>
<td></td>
</tr>
<tr>
<td>Disks</td>
<td>1 x Floppy</td>
<td></td>
</tr>
<tr>
<td><strong>Node Type 2 Hardware</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPU</td>
<td>2 x PentiumIII 450MHz</td>
<td></td>
</tr>
<tr>
<td>RAM</td>
<td>1 x 256 MB DIMM</td>
<td></td>
</tr>
<tr>
<td>Mainboard</td>
<td>Gigabyte P2DBX</td>
<td>Dual PentiumIII</td>
</tr>
<tr>
<td>VGA</td>
<td>AGP, El Cheapo, 4MB</td>
<td></td>
</tr>
<tr>
<td>NIC</td>
<td>1 x 3COM 3c905B TX</td>
<td></td>
</tr>
<tr>
<td>Disks</td>
<td>1 x Floppy</td>
<td></td>
</tr>
<tr>
<td><strong>Network Hardware</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Switch</td>
<td>D-Link DES 3224, 24-Port FE-Switch</td>
<td>used for Server-Nodes networking</td>
</tr>
<tr>
<td>HUB</td>
<td>D-Link DFE 816-STX, 16-Port FE-Hub</td>
<td>used for connection to the rest of the world</td>
</tr>
<tr>
<td>Other</td>
<td>cables, power supplies, etc</td>
<td></td>
</tr>
</tbody>
</table>
### Table A.2: Harddisk Partitions

<table>
<thead>
<tr>
<th>Partition Number</th>
<th>Partition Size</th>
<th>Filesystem Mountpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>/dev/sda1</td>
<td>128 MB</td>
<td>Swap</td>
</tr>
<tr>
<td>/dev/sda2</td>
<td>3875 MB</td>
<td>/</td>
</tr>
<tr>
<td>/dev/sdb1</td>
<td>8388 MB</td>
<td>/gz_usr_data</td>
</tr>
<tr>
<td>/dev/sdc1</td>
<td>8609 MB</td>
<td>/grafzahl_home</td>
</tr>
</tbody>
</table>

#### A.1.2 General Headnode Setup

The Headnode is used as a Bootstrap Protocol (BOOTP) (Croft & Gilmore (1985)), Network File System (NFS) (Callaghan et al. (1995)) and Trivial File Transfer Protocol (tftp) (Sollins (1981)) server for the computing nodes. It has 3 built-in U2W harddisks: /dev/sda, /dev/sdb and /dev/sdc, the partitions are listed in table A.2.

On /dev/sda2 Linux (used distribution is SuSE 7.1 Professional) is installed, it contains all system directories and a directory /tftpboot which contains all node files. The most important daemons running on the headnode are the bootpd, the tftpd and the nfsd.

The Bootstrap Protocol allows a client to discover its own IP address, the address of a server host, and the name of a file to be loaded into memory and then executed. The bootstrap operation can be thought of as consisting of two phases: The first phase could be labeled 'address determination and bootfile selection'. After this address and filename information is obtained, control passes to the second phase of the bootstrap where a file transfer may occur. The file transfer will typically use the Trivial File Transfer Protocol. However BOOTP could also work with other protocols such as Simple File Transfer Protocol (SFTP) (Lottor (1984)) or File Transfer Protocol (FTP) (Postel & Reynolds (1985)). In our setup BOOTP passes the linux kernel to the nodes (see section: A.1.3). The nodes are uniquely identified by their Media Access Control (MAC) address (this is an ISO/ANSI/IEEE standard as defined in IEEE 802.3) which is burned into the ROM of the NIC. The configuration of BOOTP is stored in /etc/bootptab on the headnode (see A.2.1).

The Network Filesystem Version 3 is used to mount most of the node filesystem from the headnode. For performance reasons the kernel NFS daemon is used and for convenience all nodes are grouped in a single security entity (see A.2.1) named 'beowulf'. The files and directories reside on the headnode in the directory /tftpboot/gz/. NFS exports are handled in the file /etc/exports (see A.2.1).

The Kernel used for the headnode is a self-compiled modular 2.4 SMP Kernel, the configuration file is /usr/src/linux/conf\_15\_03\_01.
A.1.3 Node Setup and Node Booting

The main effort was to create a cluster setup which is easy to maintain and expand. The 17 computing nodes are almost diskless - that is they have floppy disks only which are exclusively used at the very beginning of the boot process. All nodes are dual Pentium III SMP machines with 256 MB RAM and a 3COM 3C905TX-B network interface card. No monitors or keyboards are connected to the nodes. Graphic adapters are built in because computers will not boot without them.

On ‘power on’ the nodes boot a bootstrap from the floppy which enables them to search for a BOOTP server. The BOOTP daemon running on the headnode receives these requests and looks into the bootptab file (see A.2.1) if it can serve the request. Part of the request is the node’s MAC address which is used to identify a node, the bootptab file has entries for every single MAC address of cluster nodes. therefore the headnode can handle the BOOTP request and assign the node a unique IP address and a subnetmask. BOOTP also notifies the nodes about the location and name of a netbootable image to load (in our case this is: /tftpboot/bootImage2.4).

The next step is that the nodes download this image using the Trivial File Transfer Protocol (TFTP). The image includes a kernel and an initial ramdisk with a less than minimal root filesystem and a few statically compiled programs which are used before the rest of the filesystem is mounted via NFS.

Diskless Approaches

There are good reasons for choosing diskless clients (see Nemkin & Dev (2000) and de Goede (1999)) for our beowulf cluster. In the first place money for 17 harddisks can be saved cutting total expenses. And in the second place moving all system files and userspace executables to a centralized point eases the burden of administration.

There are two classic approaches to diskless clients, both of which yield their own disadvantages (table: A.3).
### Table A.3: Standard Diskless Concepts

<table>
<thead>
<tr>
<th>Concept</th>
<th>Characteristics</th>
<th>Further Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Root over NFS</td>
<td>Classical ’Root over NFS’ uses one directory on the server as a root-directory for one client. For 17 clients 17 ’root-directories’ on the server are needed. This leads to an increased administrative burden and increases the needed diskspace on the server as all files exist 17 times. Installing new software on the clients needs to be done 17 times. All node-logging to /var/log is done over NFS increasing network load. On the other hand very little RAM is used for this solution.</td>
<td>Maor (1999) Kostyrka (1997)</td>
</tr>
<tr>
<td>Root Ramdisk</td>
<td>Some disadvantages of the ’Root-NFS’ concept can be overcome with the ’Root in Ramdisk’ concept. All files are put into a ramdisk which is included in a netbootable image. This image also includes all executables Every node needs its own ramdisk image with node-specific configuration files but logging is done into the ramdisk easing network load. But the memory usage is increased dramatically as all files are stored in RAM. Installing new software means installing it 17 times and creating 17 new netbootable images. The ramdisk is a lot faster than any network operation via NFS and enhances overall performance. This concepts yields the best performance at the cost of available memory.</td>
<td>Bishop (1999) Nemkin &amp; Dev (2000) de Goede (1999)</td>
</tr>
</tbody>
</table>
Our Diskless Approach  We were looking for a diskless approach with the following characteristics:

1. As our network is a bottleneck we were looking for a solution with minimal network usage, e.g. logging should not be done over the network.
2. To ease administrative tasks all node software should be installed just once.
3. It should be possible to install most node software at runtime, that is without need to recreate a netboot image.
4. One single netboot image for all identical nodes.
5. Node specific configuration should be done automatically at boot-time.
6. Memory usage for the system should be kept at a minimum.

The solution we developed satisfies all of the above mentioned conditions. We combined 'Root over NFS' and 'Root in Ramdisk' to include all their advantages and exclude all their disadvantages. The root filesystem is split into two parts. The smaller part with node specific configuration and the absolute minimum files for booting is included in a ramdisk. This ramdisk and a kernel are included in a netbootable image which is passed via BOOTP and TFTP to the node. Node specific configuration is done at boot time with a small self-written program. The rest of the (root)-filesystem and the userspace for the nodes resides in a directory on the headnode and is mounted via NFS. This setup uses approximately 3 MB of RAM on the nodes. There is only one single root-filesysytem for all nodes. New software is installed into the node-root-filesystem on the headnode and is immediately available to all nodes without further modifications or reboots. This approaches needs some major modifications to the boot process which are explained in detail in the following sections.

Node Kernel

The Kernel is a self-compiled modular 2.4 SMP with compiled in support for the NIC, the configuration file is /usr/src/linux/gz\_11\_04\_2001. The program 'mknbi-linux' is used to prepare the kernel image and the prepared ramdisk for netbooting.

Creating the Netboot Image

The 'Netboot Image' is a special file which can be passed to a bootstrap program for execution. In our case this file contains the kernel and the prepared ramdisk. In order to create a ramdisk an empty block device is needed which can be populated with the files and directories for the ramdisk. We first create an empty file of about 3 MB (3072 blocks of 1024 Bytes). After the file is created an ext2 filesystem is created within the file. This file can be mounted on a virtual loop back device.

First create an empty ramdisk:
APPENDIX A. CLUSTER SETUP AND MAINTAINANCE

root@beaker:/tftpboot/ramdisk > dd if=/dev/zero of=rd bs=1024 count=3072

Create an ext2 filesystem on the empty file:

root@beaker:/tftpboot/ramdisk > mke2fs -vm0 -N 4096 rd
(mount file)
beaker:/tftpboot/ramdisk > mount -o loop rd /mnt

Now a directory structure can be created within the mounted file, the needed directories are (see Beekmans (2000):
/bin /etc /home /proc /sbin /usr /dev /floppy
/lib /opt /root /tmp /var

Most directories are empty and will serve as mountpoints for NFS. Not empty directories are:
/dev /etc

The content of /dev is copied from the /dev of the headnode, /etc is the most important directory for our boot process and holds all configuration files and some statically compiled utilities which are needed before /usr /bin /sbin /lib and /var are mounted. Zip the ramdisk using gzip -9 rd.

Take the compiled kernel plus the ramdisk, create the netbootable image and configure it to use a different init and take the root filesystem from the supplied ramdisk image. Last thing remaining to do is to copy the image to /tftpboot where BOOTP and TFTP expect it to be.

root@beaker:/tftpboot/ramdisk > mknbi-linux -x -d RAM -i ROM -r rd.gz -k bzImage -o gzImage2.4_ramdisk -a "init=/etc/init"

root@beaker:/tftpboot/ramdisk > cp gzImage2.4_ramdisk ../bootImage2.4

The '-a' option in the 'mknbi-linux' command appends a kernel parameter (here: init=/etc/init) to the kernel which is parsed and evaluated after the kernel is loaded. This option is explained later.

The Boot Process

The generalized regular Suse Linux boot process after POST and reading the MBR is:

1. A bootloader (like lilo) is executed from the MBR. The bootloader looks for a kernel, loads the kernel and passes some arguments to the kernel.
2. The kernel loads, checks the hardware and initializes compiled-in drivers which allow the kernel to access various devices.

3. The root filesystem is mounted read-only for booting, the validity of the root filesystem is checked.

4. Init (usually /sbin/init) is executed by the kernel, behaviour of init is defined in /etc/inittab. Init controls the various run-levels of the operating system and executes various scripts located in /etc/init.d/. All scripts parse the /etc/rc.config for information about which services to start. Which service is to be started in which runlevel is defined in the /etc/rc.d/rcX.d directories. The more important startup scripts are /etc/init.d/boot and /etc/init.d/boot.local.

5. Init starts /etc/init.d/boot which unmounts the root filesystem to run a check and then remounts the root filesystem and mounts /dev/pts, /proc and other local filesystem read-writeable as defined in /etc/fstab using /bin/mount.

6. After boot has mounted the filesystems other scripts from /etc/init.d/boot.d/ are executed and finally /etc/init.d/boot.local is run.

In our client setup this boot process cannot work as /bin and /sbin and their contents are not available to /etc/inittab and /etc/init.d/boot. This means that /sbin/init and /bin/mount need to be moved to a different directory for the boot process to complete. The mount program relies on RPC calls which in turn need a working portmapper which again needs a loopback device. The program /sbin/ifconfig is used to configure a loopback device. As /bin/mount, /sbin/portmap and /sbin/ifconfig are dynamically linked executables they rely on libraries from /lib and /var/lib which again are not available until after the mount. So it is necessary to compile statically linked mount, ifconfig and portmap executables and to include them in addition to init into the ramdisk. As all directories but /etc on the ramdisk are empty mountpoints the obvious location for mount, ifconfig, init and portmap is /etc.

As init, mount, portmap and ifconfig are needed before /etc/boot is executed and as theses programs are not in the used location some changes to the /etc/inittab and /etc/init.d/boot are needed.

The nodes are distributed across 4 subnets and mount most of their filesystem via NFS from the server which is equipped with a 4-port NIC and 4 different IP addresses. All nodes use the same ramdisk image to boot but need to contact the server through different IP addresses. This means that even mounting NFS needs to be done on different IP addresses depending on the local IP address. All nodes from with a 192.168.26.xxx address are to mount via IP address 192.168.26.254, all nodes from with a 192.168.27.xxx address are to mount via IP address 192.168.27.254 etc. pp. As we have 4 subnets this would mean creating 4 different /etc/fstab files which counteracts our intention of using one single identical boot image. We therefore wrote a small programm which parses the content of /proc/cmdline
for the local IP address and adjusts the `/etc/fstab` to represent the correct IP address of the headnode. The file `/proc/cmdline` contains the command parameters for the kernel exactly as they were passed to the kernel by BOOTP. As the IP configuration is done using BOOTP and BOOTP hands this to the kernel the local IP address is part of `/proc/cmdline` of our client nodes. The executable ‘ip_file’ also needs to be statically linked and needs to be available before issuing any mount commands from `/etc/init.d/boot`. We called the program ‘ip_file’ and it is located in `/etc`. So now we can have a look at the adjustments needed in `/etc/inittab` and in `/etc/init.d/boot`.

Added lines in inittab just after defining the default run-level are:

```
gv:I:bootwait:/etc/mount -v -t proc proc /proc
gw:I:bootwait:/etc/portmap
gx:I:bootwait:/etc/ifconfig lo 127.0.0.1 netmask 255.0.0.0 up
gy:I:bootwait:/etc/ip_file
gz:I:bootwait:/etc/mount -av
```

First `/proc` is mounted. Proc is needed for everything else. After `/proc` is available the portmapper is started and the loopback device is initialized. The next step is to adjust the `fstab` an the last step is to mount all other filesystems via NFS. Usually the filesystem mounts are done in `/etc/init.d/boot` but since we mount them earlier we need to remove the mount commands from `/etc/init.d/boot`. Looking at the modified node boot process it can be divided into the following steps:

1. The bootstrap is read and executed from the floppy disk.
2. The bootstrap searches a BOOTP server and asks for a configuration.
3. The BOOTP server replies to the request with the correct IP address and with a path to a kernel image to load.
4. TFTP is used to download the kernel.
5. The kernel loads, checks the hardware and initializes compiled-in drivers which allow the kernel to access various devices.
6. The root filesystem is mounted read-only from the ramdisk, the validity of the root filesystem is checked.
7. Init (`/etc/init`) is executed by the kernel, first `inittab` mounts `/proc`, then it runs the statically linked portmap, `ifconfig`, `ip_file` and finally mounts the rest of the filesystem via NFS.
8. Init starts `/etc/init.d/boot` which does not perform any mounts.
9. boot.local is executed, it synchronizes time and date using Simple Network Time Protocol Mills (1996) and deletes the statically linked executables which are not needed anymore.

At the end of the boot process we have approximately 2.5 MB of free space on the ramdisk which can be used for logging or for temporary files. Userspace and system executables are available via NFS, logging is done locally, all nodes share the same netboot image and the same mounts from the headnode, software upgrades and additions can be done at runtime on the exported directory of the headnode. We believe this setup to be very sophisticated and elegant. Any comments or questions are welcome.
A.2 Configuration Files

This chapter documents the actual configuration files of our cluster. These files are for internal use only and should not be made available to the public.

<table>
<thead>
<tr>
<th>Filename</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>/etc/bootptab</td>
<td>BOOTP configuration</td>
</tr>
<tr>
<td>/etc/exports</td>
<td>NFS exports</td>
</tr>
<tr>
<td>/etc/netgroup</td>
<td>creating netgroups</td>
</tr>
<tr>
<td>/etc/hosts</td>
<td>host name resolution</td>
</tr>
<tr>
<td>/etc/hosts.allow</td>
<td>allowed hosts</td>
</tr>
<tr>
<td>/etc/hosts.equiv</td>
<td>trusted hosts (r-commands)</td>
</tr>
<tr>
<td>/etc/route.conf</td>
<td>routing configuration</td>
</tr>
<tr>
<td>/usr/src/linux/conf_05_03_01</td>
<td>kernel configuration</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Filename</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>ipfile.c++</td>
<td>source code of ipfile</td>
</tr>
<tr>
<td>/etc/inittab</td>
<td>controls init</td>
</tr>
<tr>
<td>/etc/init.d/boot</td>
<td>boot configuration</td>
</tr>
<tr>
<td>/etc/init.d/boot.local</td>
<td>more boot configuration</td>
</tr>
<tr>
<td>/etc/fstab</td>
<td>filesystem mounting configuration</td>
</tr>
<tr>
<td>/etc/route.conf</td>
<td>routing configuration</td>
</tr>
<tr>
<td>/usr/src/linux/gz_11_04_01</td>
<td>kernel configuration</td>
</tr>
</tbody>
</table>
A.2.1 Headnode Configuration Files

/etc/bootptab

# This file is: /etc/bootptab
# used for bootp configuration
#
# changed by T.S. and D.-J. R.
# April 23rd 2001
#
#
gz01:hd=/tftpboot:
   :ip=192.168.26.01:
   :sm=255.255.0.0:
   :ht=ethernet:
   :ha=00500409d24c:
   :bf=bootImage2.4:
   :rp=/tftpboot/gz01

gz02:hd=/tftpboot:
   :ip=192.168.27.01:
   :sm=255.255.0.0:
   :ht=ethernet:
   :ha=00500409d236:
   :bf=bootImage2.4:
   :rp=/tftpboot/gz02

gz03:hd=/tftpboot:
   :ip=192.168.28.01:
   :sm=255.255.0.0:
   :ht=ethernet:
   :ha=00500409d249:
   :bf=bootImage2.4:
   :rp=/tftpboot/gz03

gz04:hd=/tftpboot:
   :ip=192.168.29.01:
   :sm=255.255.0.0:
   :ht=ethernet:
   :ha=005004076b14:
   :bf=bootImage2.4:
   :rp=/tftpboot/gz04

gz05:hd=/tftpboot:
   :ip=192.168.26.02:
   :sm=255.255.0.0:
   :ht=ethernet:
APPENDIX A. CLUSTER SETUP AND MAINTAINANCE

:ha=00500409cf57:
:bf=bootImage2.4:
:rp=/tftpboot/gz05
gz06:hd=/tftpboot:
:i=192.168.27.02:
:sm=255.255.0.0:
:ht=ethernet:
:ha=005004505118:
:bf=bootImage2.4:
:rp=/tftpboot/gz06
gz07:hd=/tftpboot:
:i=192.168.28.02:
:sm=255.255.0.0:
:ht=ethernet:
:ha=005004505121:
:bf=bootImage2.4:
:rp=/tftpboot/gz07
gz08:hd=/tftpboot:
:i=192.168.29.02:
:sm=255.255.0.0:
:ht=ethernet:
:ha=0050045051ac:
:bf=bootImage2.4:
:rp=/tftpboot/gz08
gz09:hd=/tftpboot:
:i=192.168.26.03:
:sm=255.255.0.0:
:ht=ethernet:
:ha=0050045030f9:
:bf=bootImage2.4:
:rp=/tftpboot/gz09
gz10:hd=/tftpboot:
:i=192.168.27.03:
:sm=255.255.0.0:
:ht=ethernet:
:ha=0050045030e6:
:bf=bootImage2.4:
:rp=/tftpboot/gz10
gz11:hd=/tftpboot:
:i=192.168.28.03:
:sm=255.255.0.0:
:ht=ethernet:
:ha=005004505127:
APPENDIX A. CLUSTER SETUP AND MAINTAINANCE

:bf=bootImage2.4:\n:rp=/tftpboot/gz11

gz12:hd=/tftpboot:\n:ip=192.168.29.03:\n:sm=255.255.0.0:\n:ht=ethernet:\n:ha=0050045030fe:\n:bf=bootImage2.4:\n:rp=/tftpboot/gz12


gz14:hd=/tftpboot:\n:ip=192.168.27.04:\n:sm=255.255.0.0:\n:ht=ethernet:\n:ha=0050045030ec:\n:bf=bootImage2.4:\n:rp=/tftpboot/gz14

gz15:hd=/tftpboot:\n:ip=192.168.28.04:\n:sm=255.255.0.0:\n:ht=ethernet:\n:ha=005004502eec:\n:bf=bootImage2.4:\n:rp=/tftpboot/gz15

gz16:hd=/tftpboot:\n:ip=192.168.29.04:\n:sm=255.255.0.0:\n:ht=ethernet:\n:ha=00500450520b:\n:bf=bootImage2.4:\n:rp=/tftpboot/gz16

gz17:hd=/tftpboot:\n:ip=192.168.26.05:\n:sm=255.255.0.0:\n:ht=ethernet:\n:ha=00500450512c:\n:bf=bootImage2.4:\n
:rp=/tftpboot/gz17
# End of File

/etc/exports

# This file is: /etc/exports
# used for nfs exports
#
# changed by T.S. and D.-J. R.
# April 23rd 2001
#
# This file contains a list of all directories
# exported to other computers.
# It is used by rpc.nfsd and rpc.mountd.
/tftpboot @beowulf (no_root_squash)
/gz_usr_data @beowulf (no_root_squash)
/tftpboot scooter zoot statler beaker
/grafzahl_home/dror scooter zoot statler beaker

/etc/netgroup

# This file is: /etc/netgroup
# used to group list of hosts in a single named group
#
# changed by T.S. and D.-J. R.
# April 23rd 2001
#
beowulf (gz01,)(gz02,)(gz03,)(gz04,)(gz05,)
(gz06,)(gz07,)(gz08,)(gz09,)(gz10,)(gz11,)
(gz12,)(gz13,)(gz14,)(gz15,)(gz16,)(gz17,)

/etc/hosts

# this file is: /etc.hosts
# Syntax:
#
# IP-Address Full-Qualified-Hostname Short-Hostname
#
127.0.0.1 localhost
134.147.26.6 beaker
APPENDIX A. CLUSTER SETUP AND MAINTENANCE

134.147.26.1  hamy mailhost loghost
134.147.26.2  sammy
134.147.26.3  leemmy
134.147.26.4  piggy
134.147.26.5  kermit
134.147.26.7  scooter
134.147.26.8  zoot
134.147.26.9  statler

192.168.26.100 graf26
192.168.26.1  gz01
192.168.26.2  gz05
192.168.26.3  gz09
192.168.26.4  gz13
192.168.26.5  gz17

192.168.27.100 graf27
192.168.27.1  gz02
192.168.27.2  gz06
192.168.27.3  gz10
192.168.27.4  gz14

192.168.28.100 graf28
192.168.28.1  gz03
192.168.28.2  gz07
192.168.28.3  gz11
192.168.28.4  gz15

192.168.29.100 graf29
192.168.29.1  gz04
192.168.29.2  gz08
192.168.29.3  gz12
192.168.29.4  gz16

/etc/hosts.allow

# See tcpd(8) and hosts_access(5) for a description.

#(ALL EXCEPT in.fingerd) EXCEPT in.identd : ALL :
# (safe_finger -l @%h 2>&1 | \
# /bin/mail -s "%d-%h %u" root) &
ALL EXCEPT in.rshd in.rlogind: 127.0.0.1
APPENDIX A. CLUSTER SETUP AND MAINTAINANCE

134.147.26.1 134.147.26.2 134.147.26.3 \ 134.147.26.4 134.147.26.5 134.147.26.6 \ 134.147.26.7 134.147.26.8 134.147.26.9  

/etc/hosts.equiv

# hostname
piggy
kermit
beaker
lemmy
sammy
hamy
scooter
zoot
statler
gonzo
@beowulf

/etc/route.conf

#
# /etc/route.conf
#
192.168.29.0 192.168.29.100 255.255.255.0 eth5
192.168.28.0 192.168.28.100 255.255.255.0 eth4
192.168.27.0 192.168.27.100 255.255.255.0 eth3
192.168.26.0 192.168.26.100 255.255.255.0 eth2
134.147.26.0 0.0.0.0 255.255.255.0 eth1
default 134.147.26.15
A.2.2 Node Configuration Files

ipfile.c++

// short hack to write local gateway into /etc/fstab
//
// g++ -static -o ip_file ip_file.c++; strip ip_file
//
// DJR and TS 15.4.2001

#include <iostream>
#include <string>
#include <fstream>

int main()
{

char fileName1[]="/proc/cmdline";
char fileName2[]="/etc/fstab";
char ch;

string content1;
string content2;
string sbuffer;

// read ip number from /proc/cmdline

ifstream fin(fileName1);
while (fin.get(ch))
{ content1 = content1 + ch; }
fin.close();
int first = content1.find("192.168.");
sbuffer = content1.substr(first+8,2);

// read /etc/fstab to string content2
// and replace all XX by the ip number

ifstream fin2(fileName2);
while (fin2.get(ch))
{ content2 = content2 + ch; }
fin2.close();
first = content2.find("XX");
while (first > -1)
{
    content2.replace(first,2,sbuffer.substr(0,2));
    first = content2.find("XX");
}

// write content2 to /etc/fstab

ofstream fout(fileName2);
    fout << content2;
    fout.close();

return 0;
}

/etc/inittab

#
# /etc/inittab
#
# default runlevel
id:3:initdefault:

#################################################################
# Aenderung fuer Clients mit root-ramdisk (/etc /var),
# Rest ueber nfs (auch /sbin)
# => init muss als erstes nfs-mounten
#
# gv : proc mount
# gw : portmapper
# gx : local loopback
# gy : schreibt nfs-server ip nummer in fstab
# gz : nfs mount
#
# 11.4.2001
#
# gv:I:bootwait:/etc/mount -v -t proc proc /proc
gw:I:bootwait:/etc/portmap
gx:I:bootwait:/etc/ifconfig lo 127.0.0.1
    netmask 255.0.0.0 up
APPENDIX A. CLUSTER SETUP AND MAINTENANCE

```
gy:I:bootwait:/etc/ip_file
gz:I:bootwait:/etc/mount -av

# check system on startup
# first script to be executed if not
# booting in emergency (-b)
# mode
si:I:bootwait:/etc/init.d/boot

# /etc/init.d/rc takes care of runlevel handling
#
# runlevel 0 is System halt
# runlevel 1 is Single user mode
# runlevel 2 is Local multiuser w/o remote network
# runlevel 3 is Full multiuser w network
# runlevel 4 is Not used
# runlevel 5 is Full multiuser with network and xdm
# runlevel 6 is System reboot
#
l0:0:wait:/etc/init.d/rc 0
l1:1:wait:/etc/init.d/rc 1
l2:2:wait:/etc/init.d/rc 2
l3:3:wait:/etc/init.d/rc 3
l4:4:wait:/etc/init.d/rc 4
l5:5:wait:/etc/init.d/rc 5
l6:6:wait:/etc/init.d/rc 6

# what to do in single-user mode
ls:S:wait:/etc/init.d/rc S
~:S:respawn:/sbin/sulogin

# what to do when CTRL-ALT-DEL is pressed
ca::ctrlaltdel:/sbin/shutdown -r -t 4 now

# special keyboard request (Alt-UpArrow)
# look into the kbd-0.90 docs for this
kb::kbrequest:/bin/echo "Keyboard Request
-- edit /etc/inittab to let this work."

# what to do when power fails/returns
pf::powerwait:/etc/init.d/powerfail start
pn::powerfailnow:/etc/init.d/powerfail now
```
#pn::powerfail:/etc/init.d/powerfail now
po::powerokwait:/etc/init.d/powerfail stop

# for ARGO UPS
sh:12345:powerfail:/sbin/shutdown -h now
THE POWER IS FAILING

# getty-programs for the normal runlevels
# <id>:<runlevels>:<action>:<process>
# The "id" field MUST be the same as the last
# characters of the device (after "tty").
1:2345:respawn:/sbin/mingetty --noclear tty1
2:2345:respawn:/sbin/mingetty tty2
3:2345:respawn:/sbin/mingetty tty3
4:2345:respawn:/sbin/mingetty tty4
5:2345:respawn:/sbin/mingetty tty5
6:2345:respawn:/sbin/mingetty tty6
#S0:123:respawn:/sbin/agetty -L 9600 ttyS0

# modem getty.
# mo:235:respawn:/usr/sbin/mgetty -s 38400 modem

# fax getty (hylafax)
# mo:35:respawn:/usr/lib/fax/faxgetty /dev/modem

# vbox (voice box) getty
# I6:35:respawn:/usr/sbin/vboxgetty -d /dev/ttyI6
# I7:35:respawn:/usr/sbin/vboxgetty -d /dev/ttyI7
# end of /etc/inittab

/etc/init.d/boot

#!/bin/sh
#
# Copyright (c) 1996 SuSE GmbH Nuernberg, Germany.
# All rights reserved.
#
# Author: Ruediger Oertel <ro@suse.de>, 1996-2000
# Werner Fink <werner@suse.de>, 1996-2000
# Burchard Steinbild <bs@suse.de>, 1996-2000
# Florian La Roche <florian@suse.de>, 1996
#
# /etc/init.d/boot
# first script to be executed from init on
# system startup
#
#
# gaendert fuer gz-nodes am 12.4.2001
#
. /etc/rc.status
. /etc/rc.config
echo "Running $0"

cr_reset
# echo -n "Mounting /proc device"
# mount -n -t proc proc /proc
# proc schon von init gemountet
rc_status -v -r

echo -n "Mounting /dev/pts"
optpts="-o mode=0620,gid=5"
mount -n -t devpts $optpts devpts /dev/pts
rc_status -vl -r

#
# Start blogd, requires /proc and /dev/pts
# REDIRECT=" 'showconsole 2>/dev/null'"
test -x /sbin/blogd & & /sbin/blogd $REDIRECT

# # check if sysrq should be enabled
# if test -e /proc/sys/kernel/sysrq ; then
if test "$ENABLE_SYSRQ" = yes ; then
  echo "1" > /proc/sys/kernel/sysrq
else
  echo "0" > /proc/sys/kernel/sysrq
fi
fi

# # check if STOP-A should be enabled
# if test -e /proc/sys/kernel/stop-a ; then
if test "$ENABLE_STOP_A" = yes ; then
  echo "1" > /proc/sys/kernel/stop-a
else
  echo "0" > /proc/sys/kernel/stop-a
fi
fi

# Disable ECN if required.
if test "$DISABLE_ECN" = "yes" -a
  -f /proc/sys/net/ipv4/tcp_ecn;
then
  echo "0" >/proc/sys/net/ipv4/tcp_ecn
fi

# # maybe we use "Multiple devices". So initialize MD.
# if test -f /etc/raidtab -a -x /sbin/raid0run ; then
  echo "Initializing Multiple Devices..."
  /sbin/raid0run -a
elif test -f /etc/mdtab -a -x /sbin/mdadd ; then
  echo "Initializing Multiple Devices..."
  /sbin/mdadd -ar & & MDADD_RETURN=0 || MDADD_RETURN=1
  if test $MDADD_RETURN -ne 0 ; then
    if test -x /sbin/ckraid ; then
      echo "Initializing Multiple Devices failed."
Trying to recover it....
/sbin/mdstop -a
for i in /etc/raid?.conf ; do
   /sbin/ckraid --fix $i
done
/sbin/mdadd -ar
rc_status -v1 -r
else
   rc_status -v1 -r
fi

# s390 is 'like' serial console
# (better: no console ttys at all)
# and we don’t have access to the HW Clock
#if test "$HOSTTYPE" = "s390" ; then
  SERIAL_CONSOLE=yes
  HWCLOCK_ACCESS=no
#fi

# set and adjust the CMOS clock
if test "$HWLOCK_ACCESS" != "no" ; then
  echo -n Setting up the CMOS clock
  CLOCKCMD=hwclock
  while read line; do
     case "$line" in
        # *MacRISC*) CLOCKCMD=clock ;;
        *MTX\ Plus*)
        CLOCKCMD="hwclock --mtxplus --directisa" ;;
       *PREP\ Dual\ MTX*)
        CLOCKCMD="hwclock --mtxplus --directisa" ;;
esac
done < /proc/cpuinfo

test "$GMT" != "YAST_ASK" &&
   $CLOCKCMD --hctosys $GMT
rc_status
test -f /etc/adjtime || echo "0.0 0 0.0" >
/proc/adjtime
if test "GMT" != "YAST_ASK" -a "START_XNTPD"
   != "yes" ;
then
   $CLOCKCMD --adjust GMT
   rc_status
fi
rc_status -v -r
fi

#
# clean up
#

####################################################
# /etc/mtab wird hier nicht gelöscht, da init schon
# nfs-mounts gemacht hat
#rm -f /etc/mtab* /etc/nologin /nologin /fastboot
# 12.4.2001
####################################################
rm -f /etc/nologin /nologin /fastboot

####################################################
# keine lokalen filesystems, die nicht schon
# gemountet waren....
# Mount local filesystems in '/etc/fstab'
# (and create an entry for / and /proc).
#
#echo "Mounting local file systems..."
#mount -fv -t proc proc /proc
#rc_status
#mount -fv -t devpts $optpts devpts /dev/pts
#rc_status
#mount -av -t nonfs,noproc,nodevpts
#rc_status -v1 -r
####################################################

# Let ld.so rebuild its cache.
# But do some tests before if it is really
# needed (bs@suse.de 01/2000)
LDCONFIG_NEEDED=false
test -s /etc/ld.so.cache || LDCONFIG_NEEDED=true
test "$run ldconfig" = true && LDCONFIG_NEEDED=true
test -x /usr/bin/find -a "$LDCONFIG_NEEDED" =
false && {
  for DUMMY in '/usr/bin/find /etc/ld.so.cache -mtime +30';
do
    LDCONFIG_NE

/etc/init.d/boot.local

#! /bin/sh
#
# Copyright (c) 1996 SuSE GmbH Nuernberg, Germany.
# All rights reserved.
#
# Author: Werner Fink <werner@suse.de>, 1996
#        Burchard Steinbild <bs@suse.de>, 1996
#
# /etc/init.d.boot.local
#
# script with local commands to be executed
# from init on system startup
#
# . /etc/rc.config
#
# Here you should add things, that should happen
# directly after booting before we’re going to
# the first run level.
#
# remove static binaries for booting in
# /etc to free ramdisk memory
rm /etc/ifconfig
rm /etc/init
rm /etc/ip_file
rm /etc/mount
rm /etc/portmap

# set time and date
ntpdate -s graf29

/etc/fstab

# proc is already mounted
# all XX will be replaced by ip_file before nfs mounts
# (see /etc/inittab)

/dev/fd0 /floppy auto noauto, user 0 0

192.168.XX.100:/tftpboot/gz/usr /usr nfs
   auto, async, rw, suid, dev, exec, udp, rsize=4096,
   wsize=4096, timeo=4 0 2

192.168.XX.100:/gz_usr_data/gz/home /home nfs
   auto, async, rw, suid, dev, exec, udp, rsize=4096,
   wsize=4096, timeo=4 0 2

192.168.XX.100:/tftpboot/gz/bin /bin nfs
   auto, async, rw, suid, dev, exec, udp, rsize=4096,
   wsize=4096, timeo=4 0 2

192.168.XX.100:/tftpboot/gz/sbin /sbin nfs
   auto, async, rw, suid, dev, exec, udp, rsize=4096,
   wsize=4096, timeo=4 0 2

192.168.XX.100:/tftpboot/gz/lib /lib nfs
   auto, async, rw, suid, dev, exec, udp, rsize=4096,
   wsize=4096, timeo=4 0 2

192.168.XX.100:/tftpboot/gz/opt /opt nfs
   auto, async, rw, suid, dev, exec, udp, rsize=4096,
   wsize=4096, timeo=4 0 2

192.168.XX.100:/tftpboot/gz/root /root nfs
   auto, async, rw, suid, dev, exec, udp, rsize=4096,
   wsize=4096, timeo=4 0 2

/etc/route.conf

#
# /etc/route.conf
#
# In this file you can configure your
# static routing...
#
# This file is read by /sbin/init.d/route.
# Destination Dummy/Gateway Netmask Device

192.168.0.0 0.0.0.0 255.255.0.0 eth0
Appendix B

Matlab Parallel Programming Tutorial

This manual explains how to write parallel C-applications with the MPI library and build-in MATLAB standalone functions. MPI (Message Passing Interface) provides easy to use functions for sending and receiving information from one process (task) to another. The tasks can be located on different computers (nodes) in a networked environment - in our case the PC-Cluster GrafZahl. This is the so called ”distributed memory” concept.

On GrafZahl with its 17 dual-SMP nodes, a parallel application can, at most, give a speed-up of 34 compared to a single processor realization of the same problem. This ideal speed-up can only be realized, if the overhead caused by network traffic is negligible, which is nearly the case for many geophysical applications like signal processing or inversion, where several totally independent tasks can be performed at the same time. Of course, a speed-up of 34 is an improvement to such an extent, that it allows to go for tasks which are simply not possible to work out on a single-cpu system. (Waiting 34 weeks instead of 1 week for a result, would generally prohibit to compute the task.)

Why using MATLAB build in functions? The answer is simply, to reduce the time for application development. In MATLAB we can rapidly develop code to solve complex mathematical and geophysical problems. However, MATLAB applications does not run in parallel. Coding the equivalent routines in C would, however, take much more time for development (and the result might be unpleasing, since the MATLAB interpreter uses very powerful numerical routines, and the self-written C-code may not.) Therefore, we code the outer framework of the parallel application in C, whereas the real computing routines to be invoked on the different nodes are coded in MATLAB. Using the MATLAB compiler and the MATLAB C-math-library, the routines work independently from the MATLAB kernel and therefore, they can be called from the main C program.

In this manual we first give a very brief introduction to MPI and then explain how to compile and call MATLAB functions. Finally, a more complex example illustrates the process.
Table B.1: Graf Zahl (Count von Count) Configuration

<table>
<thead>
<tr>
<th>Operating System</th>
<th>SuSE Linux 7.2, Kernel 2.6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compiler</td>
<td>gcc 2.95.3</td>
</tr>
<tr>
<td>MPI Implementation</td>
<td>lam 6.5.1</td>
</tr>
<tr>
<td>MATLAB</td>
<td>MATLAB 6.1 (Release 12.1)</td>
</tr>
<tr>
<td></td>
<td>MATLAB Compiler 2.1 (R12.1)</td>
</tr>
<tr>
<td></td>
<td>MATLAB C-math library 2.1 (R12.1)</td>
</tr>
</tbody>
</table>

Remark: the examples in this manual work well with the current configuration of GrafZahl, i.e.

The introduction and the first example have been partly written by Dr. Tony Stöcker. He also introduced me to the concept of Beowulf clusters and MPI. A document describing the cluster-environment and setup is available. This manual assumes the reader has knowledge of Matlab, Linux and some basic c-programming and gcc experience. If the reader is totally new to these things it is recommended to start with some basic Matlab and c-programming to get a feeling for what we try to accomplish here.

B.1 The first example: Simple usage of MPI

B.1.1 Concept

This manual does not give an overview about the possibilities of MPI, neither it describes the
background, philosophy and technical aspects of MPI. It is just a ‘quick-and-dirty’ manual to
program, compile and execute Master-Slave applications with MPI on GrafZahl.
A Master-Slave application is the following: one program, typically running on the server
(beaker), is the master program. On the nodes several replicas of the slave program are run-
ning. The slave programs are all equal except that they get different id-numbers from MPI. In
this way the master (which has typically the id-number 0) can distinguish between them. Then
the normal application is that the master feed the slaves with different portions of the data, the
data gets (independently of each other) processed by the slaves, and finally all the results are
send back to the master.

B.1.2 Trivial example

Compilation

Now, compiling is done with the helper script hcc, which has the same syntax as the gcc
compiler – the only difference is, that hcc already includes all MPI specific libraries and header
files.
Assume the user whoever is logged on at beaker and he wants to compile trivial.c
(located at, e.g., /home/whoever/MPI) then,

whoever@beaker:˜/MPI> hcc -o trivial trivial.c

will produce the executable trivial.
The sourcecode for trivial.c is depicted in figure B.1.

Execution

The executable can not be directly started, instead it has to be invoked by the helper program
mpirun. However, at first a LAM daemon has to be started on all nodes, which should be used
for the parallel application. This is done by the script lamboot, which is called with one input
parameter, specifying a file, which contains the names of the nodes to be used.

whoever@beaker:˜/MPI> lamboot -v lam_nodes

LAM 6.3.2/MPI 2 C++/ROMIO – University of Notre Dame

Executing hboot on n0 (graf26)...

/*
 * Transmit a message
 * in two process system
 */

#include <stdio.h>
#include <mpi.h>

#define BUFSIZE 64

int buf[BUFSIZE];

int main(argc, argv)
    int argc;
    char *argv[];
{
    int size, rank;
    MPI_Status status;
    /*
    * Initialize MPI.
    * /
    MPI_Init(&argc, &argv);
    /*
    * Error check the number of
    * processes.
    * Determine my rank in the
    * world group.
    * The sender will be rank 0
    * and the receiver rank 1.
    * /
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    if (size < 2) {
        printf("Need at least 2 processes.\n");
        MPI_Finalize();
        return(1);
    }
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    /*
    * As rank 0, send a message to rank 1.
    * /
    if (0 == rank) {
        MPI_Send(buf, BUFSIZE, MPI_INT, 1, 11, MPI_COMM_WORLD);
        printf("rank %d sent message\n", rank);
    }
    /*
    * As rank 1, receive a message from rank 0.
    * /
    else if (1 == rank) {
        MPI_Recv(buf, BUFSIZE, MPI_INT, 0, 11, MPI_COMM_WORLD, &status);
        printf("rank %d received message\n", rank);
    }
    MPI_Finalize();
    return(0);
}

Figure B.1: Trivial communication example with MPI.
Executing hboot on n1 (gz01)...
Executing hboot on n1 (gz02)...
topology done

Here, the -v option stands for verbose output. At this point, the nodes beaker, gz01, gz02 are ready for running LAM applications.¹

The most common usage of mpirun is to use an application schema. This is a file, describing which programs should be invoked on what machines.

Of course, it is possible to pass this information directly to mpirun, but an application schema is very useful for more complex tasks, where several programs have to be invoked on different nodes.

For the trivial example, now the execution is simply

```
whoever@beaker:/MPI> mpirun -O app_schema
rank 0 sent message
rank 1 received message
```

and one recognizes, that the simple MPI communication with the program trivial executed as expected. The option -O stands for optimization in a homogenous environment (no machine specific data-type conversions), and should always be used on GrafZahl.

```
# This file is used by
# lamboot it contains a
# list of computers to
# be used for the
# parallel computations.
# The names must be
# resolvable to ip
# addresses.
# Graf 26 is the headnode.
# Mpi ranks will be
# according to the order
# in the file.
graf26
gz01
gz02
gz03
gz04

# this file is used
# with mpirun
# it tells mpirun
# which processes to
# start how many
# times and on
# which nodes.
# The program trivial
# is executed
# 2 times on the
# computers
# ranked 0 and 1.
n0-1 -np 2 -s h trivial
```

Figure B.2: The files lamnodes on the left side and appschema on the right side

¹Note that beaker itself is specified by graf26 in the nodes file, since this name is equivalent to the correct IP-number, connecting beaker to the gz-nodes. For more information, see the setup-documentation of GrafZahl.
B.2  A simple example using Matlab and MPI

Now you have seen how lamboot, mpirun and hcc can be used to create a very simple 2-process system. The next step is to take this example and expand to use a simple Matlab function. The way to accomplish this sounds easy but is rather tricky. So make sure you understand what actually happens and read through all the source code carefully. We will walk through this example step by step. First set up the environment. Then code the Matlab function. Next compile the Matlab function into a c-function. After that code your c framework to use the supplied Matlab function. Second to last Compile all together and finally run the application.

B.2.1 Setting up the environment

First a few environment variables need to be set, these include libraries etc. are needed. We work on beaker, so you will need to ssh to beaker. The most important environment is the **LD_LIBRARY_PATH**. It is quite likely, that you already have one, so you will need to add more paths without overwriting the existing paths. I use bash. In bash you can add the following to your ".bashrc":

```bash
export LD_LIBRARY_PATH=/opt/matlab6_1/bin/glnx86:/opt/matlab6_1/extern/lib/glnx86:$LD_LIBRARY_PATH
```

If you dont use bash look into the manpages for your shell to see how to set and export variables for the shell used. Now setting up some regular PATHs is a good idea, too:

```bash
export PATH=/usr/lib/lam/bin:/opt/matlab6_1:/opt/matlab6_1/bin:/opt/matlab6_1/etc:$PATH
```

That should be all for now. Again, this works on beaker only. So dont forget to log in to beaker using ssh. And use the matlab on beaker only. Once you are done, you can head on to creating the m-file with your function.

B.2.2 The m-file

The m-file is a regular m-file containing a single function. The function can be complex - or very easy. We will use an easy function as we dont want to write a matlab primer. Our function is called bla - it is saved in a file called bla.m. It takes a single input argument and returns the square root of the doubled input argument. The code is printed in figure B.3 on the right side. This function will be saved in an m-file named slave.m Please note the semi-colon. It will prevent that the result will be printed to standard output - not only when using it within Matlab - but also in the parallel c-program.

B.2.3 Running the Matlab to C compiler

This is important. The matlab-to-c compiler is called mcc and takes millions of options, type mcc -? to see a list of them all. We want to create a slave.c and a slave.h and some libraries
called libSlave.so and a Slave.o and some needed helper files. All this is accomplished with a single mcc command (see B.3 left side)

```
whoever@beaker> matlab -nojvm
< M A T L A B >
Copyright 1984-2000
The MathWorks, Inc.
Version 6.0.0.88 Release 12
Sep 21 2000
>> !ls
slave.m
>> mcc -t -W lib:libSlave
-L C -T link:lib slave
>> !ls
libSlave.c libSlave.exports
libSlave.h slave.h
libSlave.mlib libSlave.so
slave.c slave.m
>>quit
```

Figure B.3: Calling mcc from within Matlab on an m-file

If you want to know what these files are for - read the matlab and mcc documentation.
Next step is to create a slave.o from the slave.c file - accomplish this with gcc:
```
gcc -I/opt/matlab6_1/extern/include -c -o slave.o slave.c
```

### B.2.4 Integrating the Matlab-C-File into MPI

The next step is to integrate the Matlab-generated C file into our MPI construct from the first example. We want the master to pass an integer to the first slave. This slave will call the Matlab function and will print the result to standard output. Note that the Matlab-generated functions all start with an ”mlf” and that the way arguments are passed and values are returned differ significantly from regular C functions. It is also noteworthy that a new datatype is introduced by Matlab - the so called ”mxArray”. Everything is done using mxArrays. Passing arguments is done by reference or by pointers. In order to understand the example you should be familiar with these concepts. The source code is shown in figure B.4. You should be able to figure out what this does.

### B.2.5 Compiling the Code and running the program

Now we have all the needed parts. Next step is to compile all pieces together - this is a 3 step process. First create a trivial.o from trivial.c, then create a slave.o from slave.c and finally we need to compile the pieces together. The hcc script is used for two of the three steps, one step

---

2note that this mcc command creates 7 new files. In order to include them in later compilation steps it is recommended to follow the obvious naming convention in the mcc command
// Integrate Matlab slave.m into MPI
#include <stdio.h>
#include <mpi.h>
#include <unistd.h>
#include "matlab.h"
#include "matrix.h"
#include "slave.h"

int main(argc, argv) // start main
int argc;
char *argv[];
{
int size, rank, mrank;
MPI_Status status;

// Initialize Slave
libSlaveInitialize();
// Initialize MPI.
MPI_Init(&argc, &argv);
// Determine rank
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
// As rank 0, send a message to rank 1.
if (0 == rank) {
int toSend = 9;
MPI_Comm_rank(MPI_COMM_WORLD, &mrank);
printf("Im the master. I will send an int to a slave.\n");
MPI_Send(&toSend,sizeof(toSend), MPI_INT, 1, 111, MPI_COMM_WORLD);
printf("rank %d sent message\n", rank);
}
// As rank 1, receive a message from rank 0.
else if (1 == rank) {
int *a;
mxArray *b, *c;
MPI_Recv(a,4, MPI_INT, 0, 111, MPI_COMM_WORLD, &status);
sleep(5);
printf("Im one of the slaves. I am rank 1. I will do a calculation. \n");
printf("OK, I have called a Matlab-compiled function and passed an argument.\n");
mlfAssign(&b,mlfScalar(*a)); // make scalar from passed argument. assign to b
//call slave.m (or better: slave.c) on the argument and return value to c
mlfAssign(&c,mlfSlave(b));
printf("I called on slave.m to calculate the squareroot of the given argument. The result is:\n");
mlfDisp(c); //display result
printf("\nSee - it works fine. \n\n");
}
// as rank greater than 1, dont do anything
else if (1 < rank) {
sleep(rank);
printf("Im one of the slaves. My rank is greater than 1. I wont do anything.
My rank is: %d \n", rank);
}
// clean finish
MPI_Finalize();
return(0);
} // end main

Figure B.4: Using a Matlab function within MPI
can be accomplished with gcc. The needed hcc and gcc commands are shown in figure B.5. The result is shown in figure B.6.

whoever@beaker>
whoever@beaker>hcc -I/opt/matlab6_1/extern/include -L/opt/matlab6_1/bin/glnx86 -L/opt/matlab6_1/extern/lib/glnx86 -c -o trivial.o trivial.c
whoever@beaker>
whoever@beaker>gcc -I/opt/matlab6_1/extern/include -c -o slave.o slave.c
whoever@beaker>
whoever@beaker>
whoever@beaker>hcc -I/opt/matlab6_1/extern/include -I. -L -L/opt/matlab6_1/bin/glnx86 -L/opt/matlab6_1/extern/lib/glnx86 -lmatlb -lmat -lut -lmmfile -lmx -lSlave -o trivial slave.o trivial.o
whoever@beaker> ls
libSlave.c libSlave.h libSlave.so slave.h slave.o trivial.c
libSlave.exports libSlave.mlib slave.c slave.m trivial trivial.o
whoever@beaker>

Figure B.5: Compiling the pieces

whoever@beaker> mpirun -O -np 4 n0-3 ./trivial
I'm the master. I will send an int to a slave.
rank 0 sent message
I'm one of the slaves. My rank is greater than 1. I won't do anything. My rank is: 2
I'm one of the slaves. My rank is greater than 1. I won't do anything. My rank is: 3
I'm one of the slaves. I am rank 1. I will do a calculation.
I have received the following: 9
OK, I have called a Matlab-compiled function and passed an argument.
I called on slave.m to calculate the square root of the given argument.
The result is: 4.2426

See - it works fine.

whoever@beaker>

Figure B.6: Running the program
B.3 An Object oriented approach: Using the Matlab C++ Library and MPI++

The approach described before works fine for small projects, but the programmer has very little control over the output of mcc and there is no way to program in a modern object oriented approach. If the Matlab C++ library is available this library can be used together with the MPI++ library to program object oriented. This is more work, but yields complete control over the source code as no cross-language-compiling is done. The program developed for this thesis was programmed in this fashion. It also includes the possibility to throw exceptions for better error-handling. But that is not part of this short tutorial.

The source code starts with including all necessary libraries, most important are the mpi++.h and matlab.hpp:

```c
// generic header for parallel matlab c++ programs
#include "usr/include/mpi++.h"
#include<string.h>
#include<iostream.h>
#include<fstream.h>
#include"matlab.hpp"
#include<stdlib.h>
```

The mpi++ library enables us to use the parallel environment and the matlab.hpp library includes an array class, mwArray, which has the same methods as Matlab functions. Lets first have a short look at how to use the mwArray object without parallelization.

```c
// file: ex1.cpp
#include<string.h>
#include<iostream.h>
#include<fstream.h>
#include"matlab.hpp"
#include<stdlib.h>
#include"CModel.h"

int main()
{
    // declare some data
    double data1[] = {2, 4, 6, 8};
double data2[] = {1, 3, 5, 7};

    // create 1x4 array with data1
    mwArray a(1,4,data1);
```
// create 2x2 array with data2
mwArray b(2,2,data2);

// reshape a to be 2x2
a = reshape(2,2,a);

// find average
mwArray c = average(a,b);

// print result to screen
cout << a << " + " << b << " /2 = " << c;

return 0;
}

To compile use "mbuild ex1.cpp", this will result in an executable "ex1".

The next example is how to use the mpi++ library without the matlab library. Please note that
MPI uses the terms "master" and "slave" to distinguish roles in the parallel cluster. For obvious
reasons I would prefer to use "headnode" and "computing node" but have kept the original
terms in the following examples:

/* use "hcp -o outfile hello.cpp to compile" */
/* use "mpirun n0-XXX -np XYZ -lamd ./outfilename" to run*/
/* comment by dror */
#include <iostream.h>
#include "/usr/include/mpi++.h"

int main(int argc, char *argv[])
{
    int rank, size;
    int n=77;

    // initialize MPI
    MPI::Init(argc, argv);

    // Find rank and size
    rank = MPI::COMM_WORLD.Get_rank();
    size = MPI::COMM_WORLD.Get_size();

    if (rank == 0)
    {
        cout << " I am the master. I am rank :" << rank;
        cout << "\n" << n << "\n";
// send an integer to all slaves
MPI::COMM_WORLD.Send(&n,1,MPI::INT,2,99);
}

if (rank != 0)
{
    cout << "Hello world! I am one of the slaves."
    cout << "I am " << rank << " of " << size << endl;
    if (rank == 2)
    {
        int m;
        // receive integer from master.
        MPI::COMM_WORLD.Recv(&m,1,MPI::INT,0,99);
        cout << "received: " << m << endl;
    }
}

MPI::Finalize();

return 0;

The MPI Send and MPI Recv methods are used to pass information between the nodes in the cluster. They work with predefined MPI data objects like int, double, char only, but have no knowledge about the Matlab mwArray object and can not pass it between the nodes. This is a major drawback, but a work-around exists which works fine as long as the information passed is limited as much as possible because it draws on slow disk I/O operations. The workaround is to write the mwArray to a file and then pass the filename to the other nodes which can the load the mwArray from file. This is demonstrated in the next example.

// on beaker only:
// mbuild -g -v -llammpi++ -llammpio -lpmpi -lmpi -llam
// -linsl -lutil main.cpp
#include "./usr/include/mpi++.h"
#include<string.h>
#include<iostream.h>
#include<fstream.h>
#include"matlab.hpp"
#include<stdlib.h>

int main(int argc, char *argv[])
{

int rank, size;
/* initialize mpi */
MPI::Init(argc, argv);
rank = MPI::COMM_WORLD.Get_rank();
size = MPI::COMM_WORLD.Get_size();

/*
 * as master (rank==0) create subsurface
 * and station layout.
 */
if (rank == 0)
{
    char fid[8] = "model01";
    mwArray filename = fid;
    mwArray array1 = ones(1,10,1);
    save(filename,"theArray",array1);

    for (int i=1;i<size;i++)
    {
        MPI::COMM_WORLD.Send(&fid,8,MPI::CHAR,i,77);
    }
} // end of master

if ( rank != 0 ) // all slaves
{
    mwArray mwRank = rank;
    char filerecv[8]="0";
    // receive filename
    MPI::COMM_WORLD.Recv(&filerecv,8,MPI::CHAR,0,77);
    mwArray fileToOpen = filerecv;
} // end of if (rank !=0)

MPI::Finalize(); /* clean up MPI */
return 0;

Compilation is done with mbuild, which automatically includes and links all necessary matlab headers and objects. The needed mpi bindings are included in the mbuild command as arguments: mbuild -g -v -llammpi++ -llammpio -lmpi -llam -llsl -llutil main.cpp. The -v options prints verbose output while compiling and linking and the -g options turns off code optimization. This will result in faster compilation but slower executable.
Appendix C

Source Code Documentation
C.1 CModel.h

/********************
* CModel.h header file
* defining the class CModel
* which is used to describe
* the subsurface layout
* and station layout
* for a model used
* in 3-d gravimetric
* modelling using
* Nagy’s algorithm
* ******************/

#include<iostream.h>
#include"matlab.hpp"
#include<stdlib.h>

class CModel
{

/********************
* protected part, only
* usable within
* Class methods
* ******************/
protected:
/* member data */
mwArray stationLayout;
mwArray subsurfaceLayout;

/* member methods */
mwArray adaptForStation(mwArray belowLayout,
    double StationNumber) const;
// Take Below Layout and a StationNumber
// and move belowLayout so that
// Station is at (0,0,0)

mwArray resolveZeros(mwArray belowLayout) const;
// Take BelowLayout after it has been adapted
// for StationCoordinates
// and eliminate Zero-Values by signed
// "almost Zero" values

mwArray resolveZeroCrossing(mwArray belowLayout);

// Take BelowLayout after adapted for a Station
// and remove all Zero-Crossings, this will
// return a larger BelowLayout which can
// be passed to the actual calculation.
// Additionally the addedBlocks
// are saved in the model but are not passed
// back. They are needed to construct the
// original Layout

mwArray readaptGeometricEffect(mwArray belowLayout);

// After calculation the original Size has to be
// restored. This will take the belowLayout
// and the added Blocks, and return the original
// belowLayout (columns 1-6) and the geometric
// Effect of each Block on the Station (column7)

//calculation methods
mwArray geometricEffectOnStation(mwArray ... belowLayout) const;

// After the ZeroCrossings are resolved, the
// larger BelowLayout is taken and the geometric
// effect of each Block is calculated and returned

/**********************
* public part,
* the interface
**********************/
public:

/* constructor methods */
CModel();
// included for completeness only.
// Doesn't do anything
// don't use

CModel(char filename[]);
APPENDIX C. SOURCE CODE DOCUMENTATION

// if a file describing the model
// exists, than read the file
// and create a model object for
// that file

CModel(mwArray aboveLayout, mwArray belowLayout);
// passing two arrays, create object

/* file operations */
void modelToFile(mwArray filename);
// writing model to file
// format is below layout,
// above layout

/* manipulating methods */
void setDensity(double x1, double y1, double z1, ...
    double rho);
// sets the density at the block
// defined by x1,y1,z1 to rho

mwArray setDensityReturn(double x1, double y1, ...
    double z1, double rho);
// sets the density at the block
// defined by x1,y1,z1 to rho
// and returns the block
// coordinates and new density
// in an array

/* accessor methods */
mwArray getStationLayout() const;
// returns the stationlayout

mwArray getSubsurfaceLayout() const;
// returns the block layout

mwArray getStationCount() const;
// returns the number of stations in the model

mwArray getStationCoordinates(double ...
    stationNumber) const;
// returns the coordinates of the given station
/* calculating methods */
mwArray geometricBlockEffect(mwArray belowLayout, ...
   double stationNumber);
   // returns the geometric kernel
   // for the station and blocklayout given

mwArray geometricBlockEffect(double stationNumber);
   // returns the geometric kernel
   // for the station, works on the
   // created model

mwArray gravityBlockEffect(mwArray belowLayout, ...
   double stationNumber);
   // returns the gravit kernel
   // for the station and blocklayout given

mwArray gravityBlockEffect(double stationNumber);
   // returns the geometric kernel
   // for the station, works on the
   // created model

mwArray gravityEffect(mwArray belowLayout, ...
   double stationNumber);
   // returns the single value
   // gravity effect
   // for the station and blocklayout given

mwArray gravityEffect(double stationNumber);
   // returns the single value
   // gravity effect
   // for the station, works
   // on the created model

};
C.2  CModel.cpp

#include "CModel.h"

CModel::CModel()
{
}

CModel::CModel(char filename[])
{
    mwArray fname = filename;
    mwArray tempBelow;
    mwArray tempAbove;
    load(fname,"above", &tempAbove, "below", ... &tempBelow);
    CModel::stationLayout = tempAbove;
    CModel::subsurfaceLayout = tempBelow;
}

CModel::CModel(mwArray aboveLayout, ...
mwArray belowLayout)
{
    CModel::stationLayout = aboveLayout;
    CModel::subsurfaceLayout = belowLayout;
}

//**********************************************************
* CModel::CModelresolveZeros ...
* (mwArray belowLayout) const
**********************************************************/

mwArray CModel::resolveZeros ...
    (mwArray belowLayout) const
{
    mwArray x1ZeroPositions = ...
    find(0,~belowLayout(colon(),1));
    mwArray changedx1 = belowLayout(colon(),1);
    double numberOfX1Zeros = ...
    x1ZeroPositions.EltCount();
    for (int i=1;i<=numberOfX1Zeros;i++)
    {
        changedx1(x1ZeroPositions(i))=0.001;
    }

    mwArray x2ZeroPositions = ...
    find(0,~belowLayout(colon(),2));
    mwArray changedx2 = belowLayout(colon(),2);
    double numberOfX2Zeros = ...
    x2ZeroPositions.EltCount();
    for (int i=1;i<=numberOfX2Zeros;i++)
    {
        changedx2(x2ZeroPositions(i))=-0.001;
    }

    mwArray y1ZeroPositions = ...
    find(0,~belowLayout(colon(),3));
    mwArray changedy1 = belowLayout(colon(),3);
    double numberOfY1Zeros = ...
    y1ZeroPositions.EltCount();
    for (int i=1;i<=numberOfY1Zeros;i++)
    {
APPENDIX C. SOURCE CODE DOCUMENTATION

```cpp
{ 
    changedy1(y1ZeroPositions(i)) = 0.001;
}

mwArray y2ZeroPositions = ... 
    find(0,˜belowLayout(colon(),4));
mwArray changedy2 = belowLayout(colon(),4);
double numberOfY2Zeros = ... 
    y2ZeroPositions.EltCount();
for (int i=1;i<=numberOfY2Zeros;i++)
{
    changedy2(y2ZeroPositions(i)) = -0.001;
}

mwArray z1ZeroPositions = ... 
    find(0,˜belowLayout(colon(),5));
mwArray changedz1 = belowLayout(colon(),5);
double numberOfZ1Zeros = ... 
    z1ZeroPositions.EltCount();
for (int i=1;i<=numberOfZ1Zeros;i++)
{
    changedz1(z1ZeroPositions(i)) = 0.001;
}
return horzcat(changedx1,changedx2, ... 
    changedy1, changedy2, changedz1, ... 
    belowLayout(colon(),6), ... 
    belowLayout(colon(),7));
}

/**************************** 
* CModel::CModeladaptForStation(mwArray ... / belowLayout, double StationNumber) const 
*******************************
mwArray CModel::adaptForStation(mwArray ...
    belowLayout, double StationNumber) const
{
    mwArray xCoo = stationLayout(StationNumber,1);
mwArray yCoo = stationLayout(StationNumber,2);
mwArray zCoo = stationLayout(StationNumber,3);
return (horzcat(subsurfaceLayout(colon(), ... 
    colon(1,2))−xCoo, subsurfaceLayout(colon() ... 
```
CModel::resolveZeroCrossing(mwArray belowLayout)

mwArray rampx = ramp(1,belowLayout.Size(1));
belowLayout(colon(),8) = transpose(rampx);
mwArray x1x2 = times(belowLayout(colon(), ... 1),belowLayout(colon(),2));
mwArray y1y2 = times(belowLayout(colon(), ... 3),belowLayout(colon(),4));
mwArray xCrossings = find(x1x2<0);
mwArray yCrossings = find(y1y2<0);
mwArray xCrossCount = xCrossings.EltCount();
mwArray yCrossCount = yCrossings.EltCount();
double xcount = xCrossCount.ExtractScalar(1);
double ycount = yCrossCount.ExtractScalar(1);
double xycount = 0;
mwArray xRep;
mwArray yRep;
mwArray addedBlock;
mwArray loop1 = 0;
mwArray countLoop1 = 0;
if (xcount > 0 && ycount > 0)
{
    cout << "crossings in x and y!
"
    xRep = repmat(xCrossings,1,ycount);
yRep = repmat(reshape(yCrossings,1, ... 
    ycount),xcount,1);

mwArray xycross = find(0,˜(xRep-yRep));
double numberOfBlocksToAdd = xcount*2 ... 
    +ycount*2;
mwArray addedBlock=zeros ... 
    (numberOfBlocksToAdd,8);
mwArray loop1 = xRep(xycross);
mwArray countLoop1 = loop1.EltCount();

for (int i = 1; i <= countLoop1; i++)
{
    xCrossings(find(xCrossings(colon( )
    ) == loop1(i)))=empty();
    yCrossings(find(yCrossings(colon( )
    ) == loop1(i)))=empty();
}

xCrossCount = xCrossings.EltCount();
yCrossCount = yCrossings.EltCount();
xcount = xCrossCount.ExtractScalar(1);
ycount = yCrossCount.ExtractScalar(1);
xycount = countLoop1.ExtractScalar(1);

if (xycount > 0)
{ cout << "xy-crossings, 4 block resolve mode\n"; }

for (int i = 1; i <= xycount; i++)
{
    addedBlock((4*i-3),colon(5,8))= ... belowLayout(loop1(i),colon(5,8));
    addedBlock((4*i-2),colon(5,8))= ... belowLayout(loop1(i),colon(5,8));
    addedBlock((4*i-1),colon(5,8))= ... belowLayout(loop1(i),colon(5,8));
    addedBlock((4*i),colon(5,8))= ... belowLayout(loop1(i),colon(5,8));

    addedBlock((4*i-3),1)= ... belowLayout(loop1(i),1);
    addedBlock((4*i-3),2)=0.001;
    addedBlock((4*i-3),3)= ... belowLayout(loop1(i),3);
    addedBlock((4*i-3),4)=-0.001;

    addedBlock((4*i-2),1)=0.001;
    addedBlock((4*i-2),2)= ... belowLayout(loop1(i),2);
    addedBlock((4*i-2),3)= ... belowLayout(loop1(i),3);
    addedBlock((4*i-2),4)=-0.001;
APPENDIX C. SOURCE CODE DOCUMENTATION

```plaintext
addedBlock((4*i-1),1)=0.001;
addedBlock((4*i-1),2)= ...  
   belowLayout(loop1(i),2);
addedBlock((4*i-1),3)=0.001;
addedBlock((4*i-1),4)= ...  
   belowLayout(loop1(i),4);

addedBlock((4*i),1)= ...  
   belowLayout(loop1(i),1);
addedBlock((4*i),2)=-0.001;
addedBlock((4*i),3)=0.001;
addedBlock((4*i),4)= ...  
   belowLayout(loop1(i),4);

addedBlock((4*i-3),8)=loop1(i);
addedBlock((4*i-2),8)=loop1(i);
addedBlock((4*i-1),8)=loop1(i);
addedBlock((4*i),8)=loop1(i);
}
}
else // no x-ycrossings
{
    mwArray countLoop1 = 0;
}
// loop through x-only crossings
if (xcount >0)
{
    for (int i = 1; i <= xcount; i++)
    {
        addedBlock((2*i-1+(4*countLoop1. ...
            ExtractScalar(1))),colon(3,8))= ...  
            belowLayout(xCrossings(i), ...
            colon(3,8));

        addedBlock((2*i+(4*countLoop1. ...
            ExtractScalar(1))),colon(3,8))= ...  
            belowLayout(xCrossings(i), ...
            colon(3,8));

        addedBlock((2*i+(4*countLoop1. ...
            ExtractScalar(1))),1)=0.001;
```

---

The source code documentation provides a detailed explanation of the implementation, including the use of `addedBlock` and `belowLayout` functions to add blocks and layout elements, respectively. The code snippet demonstrates how to handle x-only and x-ycrossings in a specific context, likely part of a larger program or application.
APPENDIX C. SOURCE CODE DOCUMENTATION 87

addedBlock((2*i+(4*countLoop1. ExtractScalar(1))),2)=
belowLayout(xCrossings(i),2);

addedBlock((2*i-1+(4*countLoop1. ExtractScalar(1))),2)=-0.001;

addedBlock((2*i-1+(4*countLoop1. ExtractScalar(1))),1)=
belowLayout(xCrossings(i),1);

addedBlock((2*i-1+(4*countLoop1. ExtractScalar(1))),8) =
xCrossings(i);

addedBlock((2*i+(4*countLoop1. ExtractScalar(1))),8)= xCrossings(i);
}

// loop through y-cross only
if (ycount > 0)
{
for (int i = 1; i <= ycount; i++)
{
addedBlock((2*i)-1+(2*xcount)+(4*countLoop1.ExtractScalar(1)), colon(1,2))=belowLayout(yCrossings(i),colon(1,2));

addedBlock((2*i)-1+(2*xcount)+(4*countLoop1.ExtractScalar(1)), colon(5,8))=belowLayout(yCrossings(i),colon(5,8));

addedBlock((2*i)+(2*xcount)+(4*countLoop1.ExtractScalar(1)), colon(1,2))=belowLayout(yCrossings(i),colon(1,2));

addedBlock((2*i)+(2*xcount)+(4*countLoop1.ExtractScalar(1)), colon(5,8))=belowLayout(yCrossings(i),colon(5,8));
(yCrossings(i), colon(5, 8));

addedBlock(((2*i)+(2*xcount)+(4*... countLoop1.ExtractScalar(1))), 3) ...
= 0.001;

addedBlock(((2*i)+(2*xcount)+(4*... countLoop1.ExtractScalar(1))), 4) ...
= belowLayout(yCrossings(i), 4);

addedBlock(((2*i)-1+(2*xcount)+ ... (4*countLoop1.ExtractScalar(1))... ), 3) = -0.001;

addedBlock(((2*i)-1+(2*xcount)+ ... (4*countLoop1.ExtractScalar(1))... ), 4) = belowLayout(yCrossings(i), 3);

addedBlock(((2*i)-1+(2*xcount)+ ... (4*countLoop1.ExtractScalar(1))... ), 8) = yCrossings(i);

if (xcount > 0 || ycount > 0)
{
    mwArray deleteElements = ...
    vertcat(loop1, xCrossings, yCrossings);
    mwArray zeroDelete = ...
    find(0, ~deleteElements);

    deleteElements(zeroDelete) = empty();
    mwArray crosstype;
    crosstype(colon(1, xycount), 1) = 1;
    // type 1 = xycrosses
    crosstype(colon(xycount+1, xycount+ ... xcount), 1) = 2; // type 2 = xcross
    crosstype(colon(xycount+xcount+1, ... xycount+xcount+ycount), 1) = 3;
// type 3 = ycross

crosstype(colon(1,xcount),2) = loop1;
crosstype(colon(xcount+1,xcount+ycount+ycount+ycount),2) = yCrossings;
crosstype(colon(xycount+1,xycount+ycount+ycount+ycount),2) = yCrossings;

// creating the returnValue
mwArray returnLayout = belowLayout;
returnLayout(colon(),8) = zeros(belowLayout.Size(1),1);
returnLayout(colon(),8) = transpose(rampx);
returnLayout(deleteElements, colon()) = empty();

returnLayout = vertcat(returnLayout, ... addedBlock(colon(), colon(1,8)));

return returnLayout;

} else 
{
return belowLayout;
}

} /************************** 
* CModel::reapadaptGeometricEffect ... 
* (mwArray belowLayout) 
***************************/
mwArray CModel::reapadaptGeometric ... Effect(mwArray belowLayout)
{

mwArray mwlineCount = belowLayout.Size(1);
double lineCount = mwlineCount. ... ExtractScalar(1);
int i = 1;
while (i <= lineCount)
{
    if (i+1 <= lineCount)
    {
        if (toBool(belowLayout(i, 2) == ...)
            belowLayout(i+1, 2))
        {
            belowLayout(i, 1) = ...
            belowLayout(i, 1)+belowLayout(i+1, 1);
            belowLayout(i+1, colon()) = empty();
        }
        else
        {
            i++;
        }
    }
    mwlineCount = belowLayout.Size(1);
    lineCount = mwlineCount.ExtractScalar(1);
    else
    {
        break;
    }
}
belowLayout=sortrows(belowLayout, 2);
return belowLayout;

/**************************
* CModel::geometricEffectOnStation ...
* (mwArray belowLayout) const
*************************/
mwArray CModel::geometricEffectOnStation ...
    Station(mwArray belowLayout) const
{
    mwArray x1_1 = abs(belowLayout(colon(), 1));
    mwArray x2_1 = abs(belowLayout(colon(), 2));
    mwArray y1_1 = abs(belowLayout(colon(), 3));
    mwArray y2_1 = abs(belowLayout(colon(), 4));
    mwArray z1_1 = abs(belowLayout(colon(), 5));
mwArray z2_1 = abs(belowLayout(colon(), 6));

mwArray x1_2 = power(x1_1, 2);
mwArray x2_2 = power(x2_1, 2);
mwArray y1_2 = power(y1_1, 2);
mwArray y2_2 = power(y2_1, 2);
mwArray z1_2 = power(z1_1, 2);
mwArray z2_2 = power(z2_1, 2);

mwArray r111 = sqrt(x1_2 + y1_2 + z1_2);
mwArray r112 = sqrt(x1_2 + y1_2 + z2_2);
mwArray r121 = sqrt(x1_2 + y2_2 + z1_2);
mwArray r122 = sqrt(x1_2 + y2_2 + z2_2);
mwArray r222 = sqrt(x2_2 + y2_2 + z2_2);
mwArray r212 = sqrt(x2_2 + y1_2 + z2_2);
mwArray r221 = sqrt(x2_2 + y2_2 + z1_2);
mwArray r211 = sqrt(x2_2 + y1_2 + z1_2);

mwArray T01 = times(x2_1, log(y2_1 + r221));
mwArray T02 = times(x1_1, log(y2_1 + r121));
mwArray T03 = times(y2_1, log(x2_1 + r221));
mwArray T04 = times(y2_1, log(x1_1 + r121));

mwArray T05 = times(z1_1, asin(rdivide(...
(z1_2 + y2_2 + times(y2_1, r221)), (times(...
y2_1 + r221, sqrt(y2_2 + z1_2))))));

mwArray T06 = times(z1_1, asin(rdivide(...
(z1_2 + y2_2 + times(y2_1, r121)), (times(...
y2_1 + r121, sqrt(y2_2 + z1_2))))));

mwArray T07 = times(x2_1, log(y1_1 + r211));
mwArray T08 = times(x1_1, log(y1_1 + r111));
mwArray T09 = times(y1_1, log(x2_1 + r211));
mwArray T10 = times(y1_1, log(x1_1 + r111));

mwArray T11 = times(z1_1, asin(rdivide(...
(z1_2 + y1_2 + times(y1_1, r211)), (times(...
y1_1 + r211, sqrt(y1_2 + z1_2))))));

mwArray T12 = times(z1_1, asin(rdivide(...
(z1_2 + y1_2 + times(y1_1, r111)), (times(...
y1_1 + r111, sqrt(y1_2 + z1_2))))));
mwArray T13 = times(x2_1, log(y2_1 + r222));  
mwArray T14 = times(x1_1, log(y2_1 + r122));  
mwArray T15 = times(y2_1, log(x2_1 + r222));  
mwArray T16 = times(y2_1, log(x1_1 + r122));  

mwArray T17 = times(z2_1, asin(rdivide( ...  
    (z2_2 + y2_2 + times(y2_1, r222)),(times( ...  
    y2_1 + r222, sqrt(y2_2 + z2_2)))))));  

mwArray T18 = times(z2_1, asin(rdivide( ...  
    (z2_2 + y2_2 + times(y2_1, r122)),(times( ...  
    y2_1 + r122, sqrt(y2_2 + z2_2))))));  

mwArray T19 = times(x2_1, log(y1_1 + r212));  
mwArray T20 = times(x1_1, log(y1_1 + r112));  
mwArray T21 = times(y1_1, log(x2_1 + r212));  
mwArray T22 = times(y1_1, log(x1_1 + r112));  

mwArray T23 = times(z2_1, asin(rdivide( ...  
    (z2_2 + y1_2 + times(y1_1, r212)),(times( ...  
    y1_1 + r212, sqrt(y1_2 + z2_2))))));  

mwArray T24 = times(z2_1, asin(rdivide( ...  
    (z2_2 + y1_2 + times(y1_1, r112)),(times( ...  
    y1_1 + r112, sqrt(y1_2 + z2_2))))));  

mwArray stationKernel = abs(T01-T02+T03 ...  
-T04-T05+T06-T07+T08-T10+T11-T12- ...  
T13+T14-T15+T16+T17-T18+T19-T20+T21- ...  
T22-T23+T24);  

return horzcat(stationKernel,belowLayout ...  
(colon(),8)); 
}  

/****************************
* CModel::geometricBlockEffect(mwArray ...  
* belowLayout, double stationNumber)
* ***************************

mwArray CModel::geometricBlockEffect ...
(mwArray belowLayout, double stationNumber)
{
    mwArray t1 = adaptForStation(belowLayout, stationNumber);
    mwArray t2 = resolveZeros(t1);
    mwArray t3 = resolveZeroCrossing(t2);
    mwArray t4 = geometricEffectOnStation(t3);
    mwArray t5 = readaptGeometricEffect(t4);
    return t5(colon(),1);
}

/****************************
* CModel::gravityBlockEffect ...
( mwArray belowLayout, double stationNumber )
******************************/
mwArray CModel::gravityBlockEffect(mwArray ...
    belowLayout, double stationNumber)
{
    mwArray t1 = adaptForStation(belowLayout, ...
        stationNumber);
    mwArray t2 = resolveZeros(t1);
    mwArray t3 = resolveZeroCrossing(t2);
    mwArray t4 = geometricEffectOnStation(t3);
    mwArray t5 = readaptGeometricEffect(t4);
    mwArray t6 = t5(colon(),1);
    mwArray t7 = subsurfaceLayout(colon(),7);
    return times(t6,t7)* 6.673e-11;
}

/****************************
* CModel::gravityEffect(mwArray ...
* belowLayout, double stationNumber)
******************************/
mwArray CModel::gravityEffect(mwArray ...
    belowLayout, double stationNumber)
{
    mwArray t1 = adaptForStation(belowLayout, ...
        stationNumber);
    mwArray t2 = resolveZeros(t1);
    mwArray t3 = resolveZeroCrossing(t2);
mwArray t4 = geometricEffectOnStation(t3);
mwArray t5 = readaptGeometricEffect(t4);
mwArray t6 = t5(colon(),1);
mwArray t7 = subsurfaceLayout(colon(),7);
return sum(times(t6,t7)* 6.673e-11);
}

/***************************
* CModel::geometricBlockEffect(double stationNumber)
***************************/
mwArray CModel::geometricBlockEffect ...
  (double stationNumber)
{
  mwArray t1 = adaptForStation ...  
   (CModel::subsurfaceLayout, stationNumber);
cout << "after adapting for ... 
   station coordinates\n";

  mwArray t2 = resolveZeros(t1);
cout << "after resolving zeros \n";
  //cout << t2(colon(),colon(1,4)) << endl;

  mwArray t3 = resolveZeroCrossing(t2);
cout << "after resolving zero-crossings\n";
  //cout << t3(colon(),colon(1,4)) << endl;

  mwArray t4 = geometricEffectOnStation(t3);
cout << "after geo effect not readapted \n";
  //cout << t4 << endl;

  mwArray t5 = readaptGeometricEffect(t4);
cout << "after readaption\n";
  return t5(colon(),1);
}

/***************************
* CModel::gravityBlockEffect(double stationNumber)
***************************/
mwArray CModel::gravityBlockEffect(double ...
stationNumber)
{
    mwArray t1 = adaptForStation(CModel::... 
        subsurfaceLayout, stationNumber);
    mwArray t2 = resolveZeros(t1);
    mwArray t3 = resolveZeroCrossing(t2);
    mwArray t4 = geometricEffectOnStation(t3);
    mwArray t5 = readaptGeometricEffect(t4);
    mwArray t6 = t5(colon(),1);
    mwArray t7 = subsurfaceLayout(colon(),7);
    return times(t6,t7)* 6.673e-11;
}

/***************************************************************************/
/* CModel::gravityEffect(double stationNumber) */
/***************************************************************************/
mwArray CModel::gravityEffect(double stationNumber)
{
    mwArray t1 = adaptForStation(CModel::... 
        subsurfaceLayout, stationNumber);
    cout << "After adapting for Station: " ...
        << stationNumber << endl;
    mwArray t2 = resolveZeros(t1);
    cout << "After Resolving Zeros: " << ...
        << stationNumber << endl;
    mwArray t3 = resolveZeroCrossing(t2);
    cout << "After Resolving Zero Crossings: " ...
        << stationNumber << endl;
    mwArray t4 = geometricEffectOnStation(t3);
    cout << "After Calculation of G Effect: " ...
        << stationNumber << endl;
    mwArray t5 = readaptGeometricEffect(t4);
    cout << "After Readapting Geometry: " << ...
        << stationNumber << endl;
    mwArray t6 = t5(colon(),1);
    cout << "Before Multiplying whit density: " ...
        << stationNumber << endl;
    mwArray t7 = subsurfaceLayout(colon(),7);

    return sum(times(t6,t7)* 6.673e-11);
}
/******************************
* CModel::getStationLayout() const
*******************************/
mwArray CModel::getStationLayout() const
{
    return stationLayout;
}

/******************************
* CModel::getSubsurfaceLayout() const
*******************************/
mwArray CModel::getSubsurfaceLayout() const
{
    return subsurfaceLayout;
}

/******************************
* CModel::getStationCount() const
*******************************/
mwArray CModel::getStationCount() const
{
    return stationLayout.EltCount()/4;
}

/******************************
CModel::getStationCoordinates(double ... 
    stationNumber) const
*******************************/
mwArray CModel::getStationCoordinates ...
    (double stationNumber) const
{
    return stationLayout(stationNumber,colon());
}

/******************************
* CModel::modelToFile(mwArray filename)
*******************************/
void CModel::modelToFile(mwArray filename)
{
    save(filename,"above",stationLayout, ...
"below", subsurfaceLayout);
}

/******************************************
* CModel::setDensity(double x1, double y1, double ...
* z1, double rho)
*******************************************/
void CModel::setDensity(double x1, double y1, ...
    double z1, double rho)
{
    mwArray tempCount = subsurfaceLayout ...
        (colon(), 1);
    mwArray blockCount = tempCount.EltCount();
    mwArray blockIndex = ramp(1, blockCount);
    CModel::subsurfaceLayout(colon(), 8) = ...
        reshape(blockIndex, blockCount, 1);

    mwArray tx1 = find(subsurfaceLayout ...
        (colon(), 1) == x1);
    mwArray tempbelow = ...
        CModel::subsurfaceLayout(tx1, colon());

    mwArray ty1 = find(tempbelow(colon(), 3) == y1);
    tempbelow = tempbelow(ty1, colon());

    mwArray tz1 = find(tempbelow(colon(), 5) == z1);
    tempbelow = tempbelow(tz1, colon());

    mwArray whichBlock = tempbelow(1, 8);

    CModel::subsurfaceLayout(whichBlock, 7) = rho;
}

/******************************************
* CModel::setDensityReturn(double x1, double ...
* y1, double z1, double rho)
*******************************************/
mwArray CModel::setDensityReturn(double x1, ...
double y1, double z1, double rho)
{
    mwArray tempCount = subsurfaceLayout ...
        (colon(),1);
    mwArray blockCount = tempCount.EltCount();
    mwArray blockIndex = ramp(1,blockCount);
    CModel::subsurfaceLayout(colon(),8)= ...
        reshape(blockIndex,blockCount,1);

    mwArray tx1=find(subsurfaceLayout ...
        (colon(),1)==x1);
    mwArray tempbelow = ...
        CModel::subsurfaceLayout(tx1,colon());

    mwArray ty1=find(tempbelow(colon(),3)==y1);
    tempbelow = tempbelow(ty1,colon());

    mwArray tz1=find(tempbelow(colon(),5)==z1);
    tempbelow = tempbelow(tz1,colon());

    mwArray whichBlock = tempbelow(1,8);
    CModel::subsurfaceLayout(whichBlock,7)=rho;

    return whichBlock;
}
C.3 main.cpp

// on beaker only:
// mbuild -g -v -llammpi++ -llammpio -lpmpi ...
// -lmpi -llam -lnsl -lutil main.cpp CModel.cpp
#include "/usr/include/mpi++.h"
#include<string.h>
#include<iostream.h>
#include<fstream.h>
#include"matlab.hpp"
#include<stdlib.h>
#include"CModel.h"

int main(int argc, char *argv[])
{
    int rank, size;

    /* initialize mpi */
    MPI::Init(argc, argv);
    rank = MPI::COMM_WORLD.Get_rank();
    size = MPI::COMM_WORLD.Get_size();
    int slavecount = size-1;

    mwArray indexFileName = "stationindex";
    char fid[8] = "model01";

    /*
    * as master (rank==0) create
    * subsurface and station layout.
    */
    if (rank == 0)
    {
        CModel Mastermodel("model01");
        mwArray stationcount = ...
        Mastermodel.getStationCount();
    }
if (tobool(stationcount <= slavecount))
    // if more slaves than stations, n0
    // doesn't count as 1!
    {
        cout << "more slaves than stations!\n";
        mwArray stationindex = ramp(1,stationcount);
        save(indexFileName,"stationindex",stationindex);
    }
else // if more stations than slaves
    {
        cout << "more stations than slaves!\n";
        mwArray stationsperslave = ...
            floor(rdivide(stationcount,slavecount));
        mwArray slaveswithonemorestation = ...
            mod(stationcount,slavecount);
        if (0==tobool(stationsperslave))
            {
                stationsperslave=1;
            }
        mwArray index1 = ramp(1,slavecount);
        mwArray stationindex = ...
            repmat(index1,1,stationsperslave);
        if (tobool(slaveswithonemorestation != 0))
            {
                stationindex(colon(slavecount* ...
                    stationsperslave+1,slavecount* ...
                    stationsperslave+ ...
                    slaveswithonemorestation))= ...
                    ramp(1,slaveswithonemorestation);
            }
        save(indexFileName,"stationindex", ...
            stationindex);
    }

for (int i=1;i<size;i++)
    {
        MPI::COMM_WORLD.Send ...
            (&fid,8,MPI::CHAR,i,77);
    }
APPENDIX C. SOURCE CODE DOCUMENTATION

} // end of master

/* as slave receive filenames and Station-Number
* from master, calcaulte force
* from passed model on Station or the
* kernel for that Station.
*/
if ( rank > 0 ) // all slaves
{
  mwArray mwRank = rank;
  char filerecv[8]="0";
  // receive filename
  MPI::COMM_WORLD.Recv(&filerecv,8,MPI::CHAR,0,77);
  // receive string
  mwArray fileToOpen = filerecv;
  // put char in mwArray for load
  CModel Slavemodel(filerecv);
  // create model from file
  mwArray stationcount = Slavemodel.getStationCount();
  // get number of stations for this model

  if (tobool(rank <= stationcount))
  {
    // find appropriate stations for me
    mwArray stationindex;
    load(indexFileName,"stationindex",&stationindex);
    stationindex=stationindex-rank;
    mwArray myStations = find(0,~stationindex);

    //cout << "I am slave: " << rank << ..." and my Stations are: " << myStations << endl;
    mwArray mySize = myStations.Size(2);

    // loop through my Stations,
    // adapt model and calculate kernel
    for (int im=1; tobool(im<=mySize); im++)
    {
      //cout << " ************* going for a ... new station *************\n";
      mwArray ArrayCurrentStation =myStations(im);
double currentStation = ...
    ArrayCurrentStation.ExtractScalar(1);
cout << "Station is\t" << ...
    currentStation << endl;

mwArray kernel = ...
    Slavemodel.gravityEffect(currentStation);

    // saving to files
    mwArray stationString = ...
        num2str(ArrayCurrentStation);

    save(stationString,"kernel",kernel);
}

} // end of if for slaves up to stationcount

} // end of if (rank !=0)

MPI::Finalize(); /* clean up MPI */
return 0;
}
C.4 Makemodel.cpp

// on beaker only:
// mbuild -g -v main.cpp CModel.cpp

#include<string.h>
#include<iostream.h>
#include<fstream.h>
#include"matlab.hpp"
#include<stdlib.h>
#include"CModel.h"

int main(int argc, char *argv[]) {

mwArray indexFileName = "stationindex";
char fid[8] = "model01";


    /* ****************************************** */
    /* ********** SUBSURFACE LAYOUT ********** */
    /* ****************************************** */

    mwArray tempx = ramp(0,5,495);
    mwArray tempy = ramp(0,5,495);
    mwArray tempz = ramp(0,5,105);

    mwArray x1, x2, y1, y2, z1, z2, rho;

    double xcount = tempx.EltCount();
    double ycount = tempy.EltCount();
    double zcount = tempz.EltCount();

    x1 = meshgrid(&y1, &z1, tempx, tempy, tempz);
    x1 = reshape(x1, x1.EltCount(), 1);

    y1 = reshape(y1, y1.EltCount(), 1);
\textbf{APPENDIX C. SOURCE CODE DOCUMENTATION}

\begin{verbatim}
z1 = reshape(z1,z1.EltCount(),1);
x2 = x1+tempx(2)-tempx(1);
y2 = y1+tempy(2)-tempy(1);
z2 = z1+tempz(2)-tempz(1);

mwArray Blockcount = x1.EltCount();
rho=zeros(x1.EltCount(),1);
rho = rho + 2670;
mwArray belowLayout = ... 
    horzcat(x1,x2,y1,y2,z1,z2,rho); 

// now adding large surrounding blocks of
// uniform density:

// belowLayout(Blockcount+1,colon(1,7)) = ...
// horzcat(-5000,0,-5000,5500,0,110,2670);

// belowLayout(Blockcount+2,colon(1,7)) = ...
// horzcat(500,5500,-5000,5500,0,110,2670);

// belowLayout(Blockcount+3,colon(1,7)) = ...
// horzcat(0,500,-5000,0,0,110,2670);

// belowLayout(Blockcount+4,colon(1,7)) = ...
// horzcat(0,500,500,5500,0,110,2670);

//*****************************************************************************/

/* ******************* STATION LAYOUT ********** */

mwArray xStationstemp = ramp(50,5,450);
mwArray yStationstemp = ramp(50,5,450);

mwArray xStation,yStation,zStation,gStation;

xStation = ... 
    meshgrid(&yStation,xStationstemp,yStationstemp);
xStation = reshape(xStation,xStation.EltCount(),1);
yStation = reshape(yStation,yStation.EltCount(),1);
gStation = zeros(xStation.EltCount(),1);
\end{verbatim}
zStation = zeros(xStation.EltCount(),1);
mwArray stationcount = xStation.EltCount();
mwArray aboveLayout = ...
    horzcat(xStation,yStation,zStation,gStation);

/* ************************************ */
/* ********** CREATEING MODEL ********* */
/* ************************************ */
CModel Mastermodel(aboveLayout, belowLayout);

/* ************************************ */
/* ********** Setting Densities ******* */
/* ************************************ */
Mastermodel.setDensity(245,245,0,2000);
Mastermodel.setDensity(245,250,0,2000);
Mastermodel.setDensity(250,245,0,2000);
Mastermodel.setDensity(250,250,0,2000);
Mastermodel.setDensity(245,245,5,2000);
Mastermodel.setDensity(245,250,5,2000);
Mastermodel.setDensity(250,245,5,2000);
Mastermodel.setDensity(250,250,5,2000);

for (int i =10;i <100; i+=5)
{
    Mastermodel.setDensity(245,245,i,0);
    Mastermodel.setDensity(245,250,i,0);
    Mastermodel.setDensity(250,245,i,0);
    Mastermodel.setDensity(250,250,i,0);
}

/* ************************************ */
/* ****** Writing Model to file ******* */
/* ************************************ */

Mastermodel.modelToFile("model01");

return 0;