Rapid calculation of gravity anomalies based on residual node densities and its GPU implementation

Zhang Sheng\textsuperscript{a,b}, Meng Xiaohong\textsuperscript{a,b,*}, Chen Zhaoxi\textsuperscript{a,b}, Zhou Junjie\textsuperscript{a,b}

\textsuperscript{a} Key Laboratory of Geo-Detection (China University of Geosciences, Beijing), Ministry of Education, Beijing 100083, China
\textsuperscript{b} School of Geophysics and Information Technology, China University of Geosciences, Beijing 100083, China

Abstract

Gravity forward modeling is an important part of three-dimensional physical property inversion; however, its time-consuming nature restricts its practical application. Several authors have made efforts to develop this procedure and have achieved some success. This paper proposes a new simple and fast approach for computing gravity anomalies at arbitrary points. In our method, the subsurface is divided into a 3D array of rectangular prismatic blocks, each with a constant density. Then the block density model is transformed into a residual node density model. Finally, gravity anomalies at survey points are calculated using the new residual node density-based forward formula. This method is implemented on a CPU–GPU heterogeneous architecture with the CUDA FORTRAN language and achieves more than four times acceleration compared to the classical method in the same computational environment and two orders of magnitude acceleration compared to one CPU processor on synthetic random density model tests.

© 2015 Elsevier Ltd. All rights reserved.

1. Introduction

The gravity method has been used by geophysicists in a wide range of investigations including tectonic studies, mineral explorations, engineering studies, and environmental problems (Blakely, 1995; Wolfgang and Peter, 2009). Over the last decade, the gravity method has made significant progress in instrumentation, data acquisition, data processing and interpretation methods, especially in the gravity quantitative inversion method (Nabighian et al., 2005; Li and Oldenburg, 1998; Yao et al., 2003; Pilkington, 2009; Chen et al., 2012; Sun and Li, 2014; Zhang and Meng, 2015).

The commonly used gravity quantitative inversion method can be separated into two approaches: one is the so-called geometry inversion, which assumes that gravity anomalies are caused by a specific, but geometrically simple, block or surface with specific density contrast. The aim of this inversion method is to determine the shape of the source or the boundary of the density interface (Parker, 1972; Oldenburg, 1974; Nabighian et al., 2005; Feng et al., 2014). The other approach is the physical property inversion or generalized inversion, which assumes that the subsurface is represented by the summation of rectangular prismatic blocks, each with a constant density. The aim of this inversion method is therefore to recover the distribution of the density of each prismatic block. This type of inversion has become increasingly popular over the last two decades (Li and Oldenburg, 1998; Yao et al., 2003; Pilkington, 2009; Chen et al., 2012a; Sun and Li, 2014; Zhou et al., 2015).

However, solving a physical property inversion requires a large amount of calculation when used in practical applications. When the inversion extends from 2D to 3D, it is in general very computationally demanding, and parallel computational resources or even supercomputers are often used to obtain a result (Yao et al., 2003; Moorkamp et al., 2010; Čuma and Zhdanov, 2014; Chen et al., 2012a, 2012b). Within the inversion process, one of the computational bottlenecks is the forward calculation of the 3D density model (Chen et al., 2012a).

In recent years, several authors attempt to improve the forward modeling process. One way is to optimize the forward algorithm, for example by the information reduction approach (Portniaguine and Zhdanov, 2002; Li and Oldenburg, 2003; Yao et al., 2003; Čuma and Zhdanov, 2014), which has achieved a speedup ratio of several or dozens of times compared to the original method. However, this type of method may result in lower accuracy during the reduction approach. Another efficient way to accelerate the forward procedure is to use parallel computing with specific hardware (Moorkamp et al., 2010; Chen et al., 2012a, 2012b; Liu et al., 2012; Čuma and Zhdanov, 2014), such as multi-core CPUs.
and general-purpose graphics processing units (GP-GPUs), which have been used as co-processors of CPU to achieve high-performance computing (NVIDIA Corporation, 2010; Greg and Massimiliano, 2011; PGI Corporation, 2011) and has been used successfully in many geophysical parallel computing applications (Zhang et al., 2009; Moorkamp et al., 2010; Chen et al., 2012). In a potential field application, Moorkamp et al. (2010), Chen et al. (2012a, 2012b), and Čuma and Zhdanov (2014) use a GPU accelerator in their forward/inversion program and achieve greatly improved efficiency compared to the serial code on CPU.

In this paper, we simplify the classical forward approach by introducing the node density model. The proposed method can not only reduce the redundant computations of the classical forward method, but is also suitable for massively parallel computing. We implement the new method on a CPU–GPU heterogeneous architecture with the CUDA FORTRAN language, and then evaluate its efficiency and accuracy with random density model test.

This paper is organized as follows. In the next section, we will briefly describe the existing gravity forward method and point out its redundant computations. We will then introduce the residual node density model and the new residual node density-based gravity forward formula. Then we will detail the implementation of the new method using the CUDA FORTRAN language. Finally, the use of the random density model to validate the speed and accuracy of the proposed method will be described.

2. Method

2.1. Forward modeling of the 3D density model

In a rectangular reference coordinate system with the horizontal plane and the z-axis positive in the downwards direction, the 3D density model is divided by rectangular blocks, with each block having a constant density. A gravity anomaly at an arbitrary point \( P(x, y, z) \) can be seen as the summation of gravity anomalies caused by each individual block (Fig. 1):

\[
\Delta g(x, y, z) = \sum_{i=1}^{nx} \sum_{j=1}^{ny} \sum_{k=1}^{nz} \Delta g(i, j, k),
\]

(1)

where \( \Delta g(i, j, k) \) is the gravity anomaly caused by block \((i, j, k)\).

For each block bounded by planes \( x=x_1, x=x_2, y=y_1, y=y_2, z=z_1, z=z_2 \), the gravity anomaly can be calculated by the following equation (Blakely, 1995):

\[
\Delta g(x, y, z) = \rho \sum_{i=1}^{nx} \sum_{j=1}^{ny} \sum_{k=1}^{nz} (-1)^{j+i+k} S(x, y, z, z_k) + \gamma \rho S(x, y, z, z_k) + r
\]

\[
= \sqrt{(x-x_1)^2 + (y-y_1)^2 + (z-z_1)^2} S(x, y, z, z_1 - z) + \gamma \rho S(x, y, z, z_1 - z)
\]

\[
+ (y-y_1)ln((z-z_1) + r) + (x-x_1)ln((y-y_1) + r)
\]

\[
- (z-z_1)tan^{-1}((x-x_1)(y-y_1))
\]

(2)

where \( \gamma \) is the universal constant of gravititation, \( \rho \) is the block density, and \( S(x, y, z, z_1 - z) \) is the position-related function.

Presently, the above approach is the most commonly used gravity forward method. However, when the number of model cells or the number of observation points increase, computation of the forward approach becomes increasingly time-consuming (Yao et al., 2003; Moorkamp et al., 2010; Chen et al., 2012).

From Eq. (2), the gravity anomalies of a rectangular prism cell can be seen as the linear combination of its eight corners’ position-related function \( S(x, y, y_1, z_1 - z) \). In the case of a 3D mesh, eight adjacent blocks share a common node, and the shared node’s position-related function \( S(x, y, y_1, z_1 - z) \) must be repetitively calculated eight times in the forward approach, leading to seriously redundant computations. To reduce these repeated calculations and improve computational efficiency, we introduced the node density \( \sigma \).

2.2. Definition of residual node density and the modified forward formula

Assuming that the calculation node is the origin, the residual node density \( \sigma \) is defined as

\[
\sigma = (\rho_1 + \rho_3 + \rho_5 + \rho_7) - (\rho_2 + \rho_4 + \rho_6 + \rho_8),
\]

(3)

where \( \rho_i (i = 1, 2, ..., 8) \) is the eight octants’ block density surrounding the computing node (Fig. 2).

For a 3D density model, the residual node density at node \((i, j, k)\) can be calculated as follows:

Fig. 1. Nomenclature for gravity forward modeling. The position of the measurement is described by the coordinate triple \((x, y, z)\). The model is divided into rectangular prisms of constant density \( \rho \). The coordinates of each block can be described by the opposing corners of the block.

Fig. 2. Definition of residual node density \( \sigma \). \( \rho_i (i = 1, 2, ..., 8) \) is the eight octants’ block density.
\[ \sigma(\mathbf{x}_i, \mathbf{y}_j, \mathbf{z}_k) = (\rho_1 + \rho_2 + \rho_6 + \rho_7) - (\rho_2 + \rho_3 + \rho_4 + \rho_5) \]

\[ \rho_1 = \rho(\mathbf{x}_i, \mathbf{y}_j, \mathbf{z}_k - 1); \]
\[ \rho_2 = \rho(\mathbf{x}_i, \mathbf{y}_j - 1, \mathbf{z}_k - 1); \]
\[ \rho_3 = \rho(\mathbf{x}_i - 1, \mathbf{y}_j - 1, \mathbf{z}_k - 1); \]
\[ \rho_4 = \rho(\mathbf{x}_i - 1, \mathbf{y}_j, \mathbf{z}_k - 1); \]
\[ \rho_5 = \rho(\mathbf{x}_i, \mathbf{y}_j, \mathbf{z}_k); \]
\[ \rho_6 = \rho(\mathbf{x}_i, \mathbf{y}_j - 1, \mathbf{z}_k); \]
\[ \rho_7 = \rho(\mathbf{x}_i - 1, \mathbf{y}_j - 1, \mathbf{z}_k); \]
\[ \rho_8 = \rho(\mathbf{x}_i - 1, \mathbf{y}_j, \mathbf{z}_k); \]

\[ \Delta g(\mathbf{x}, \mathbf{y}, \mathbf{z}) = \gamma \sum_{\mathbf{x}_i=1}^{\mathbf{nx}+1} \sum_{\mathbf{y}_j=1}^{\mathbf{ny}+1} \sum_{\mathbf{z}_k=1}^{\mathbf{nz}+1} \sigma(\mathbf{x}_i, \mathbf{y}_j, \mathbf{z}_k) \mathbf{S}(\mathbf{x}_i - \mathbf{x}, \mathbf{y}_j - \mathbf{y}, \mathbf{z}_k - \mathbf{z}), \]  \(\text{(5)}\)

where \(\sigma(\mathbf{x}_i, \mathbf{y}_j, \mathbf{z}_k)\) is the residual node density at node \((\mathbf{x}_i, \mathbf{y}_j, \mathbf{z}_k)\), and \(\mathbf{x}_i, \mathbf{y}_j, \mathbf{z}_k\) is the node number in the \(x\)-, \(y\)-, and \(z\)-direction, respectively. 

**Fig. 3.** 3D block density model and its relative node density model. Red: 1.0 g/cm\(^3\), blue: –1.0 g/cm\(^3\). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

3. Implementation

Forward calculation in matrix form provides a good parallel architecture for parallel computing on a large scale. In this paper, we intended to use CPU–GPU heterogeneous parallel architecture to achieve a fast computing speed and an advantage in massive computations.

In CPU–GPU parallel computing, the CPU is responsible for the serial part and the GPU for the parallel part. When using a GPU for parallel computing, we need to design a parallel strategy. Many GPU computing strategies for the classical gravity forward method have been covered by Moorkamp et al. (2010) and Chen et al. (2012b). This section will mainly focus on the different parts of the new method.

According to the matrix form of the node density-based forward formula, three major steps need to be followed (Fig. 4): Step 1. Calculate the residual node density vector from the cell density model; Step 2. Calculate the kernel matrix \(G\) of the forward approach from the model grid and observations; and Step 3. Calculate the gravity anomalies by matrix multiplication.

**Fig. 4.** Flowchart of the proposed forward method for GPU parallel computing.
3.1. Calculate the residual node density vector

This is a major difference from the classical forward method. It is apparent that every node density is computed independently and can be obtained very efficiently by parallel computing.

Assuming that there are $nx+1$, $ny+1$, and $nz+1$ nodes in the x-, y-, and z-directions in the subsurface respectively, a 2D thread structure can be defined. For each block, a 2D thread number defined as $\text{dimblock}$ (e.g., 16, 16, 1) indicates that in each block there are 16 threads in each of the x- and y-directions. Then,

```fortran
attributes(global) subroutine den2mden(den_d,nx,ny,nz,density_d)
  implicit none
  integer, value :: nx,ny,nz
  real*8, device :: den_d(nx+1,ny+1,nz+1), density_d(nx+1)*(ny+1)*(nz+1)
  integer i,j,k

  i=threadidx%x + (blockidx%x-1)*blockdim%x
  j=threadidx%y + (blockidx%y-1)*blockdim%y

  if((i.ge.(nx+1)).and.(j.ge.(ny+1))) then
    do k=1,nz+1
      ijk=(i-1)*(ny+1)*(nz+1)+(j-1)*(nz+1)+k
      density_d(ijk)= (den_d(i,j,k-1)+den_d(i-1,j,k-1))
      &+den_d(i-1,j,k)+den_d(i,j-1,k)
      &+(den_d(i-1,j-1,k)+den_d(i,j,k-1))
      &+(den_d(i,j,k)+den_d(i-1,j-1,k))
    enddo
  endif
  call syncthreads()
end subroutine
```

**Fig. 5.** CUDA FORTRAN code for calculating the node density model.

```fortran
attributes(global) subroutine calc_kernel_4d(xo,yo,zo,xmin,xmax,ymin,ymax,zmin,zmax)
  implicit none
  integer, value :: nx,ny,nz
  real*8, value :: xmin,xmax,ymin,ymax,zmin,zmax
  real*8, device :: a_d(nx+1)*(ny+1)*(nz+1), xo,yo,zo
  integer :: i,m,j,n,ijk
  real*8 :: gama=6.67d-3
  real*8 :: dxm=(xmax-xmin_mod)/(nx)
  real*8 :: dym=(ymax-ymin_mod)/(ny)
  real*8 :: dzm=(zmax-zmin_mod)/(nz)

  im=threadidx%x + blockdim%x * (blockidx%x-1)
  jm=threadidx%y + blockdim%y * (blockidx%y-1)

  if ((im.le.(nx+1)) .and. (jm.le.(ny+1))) then
    xmin_mod=xo+dxm*(im-1)
    ymax_mod=yo+dym*(jm-1)
    do km=1,nz+1
      x=xm-xo ; y=ym-yo ; z=zm-zo
      rijk=sqrt(x*x+y*y+z*z)
      a_d(ijk)=gama*(z.atan2((x*y),(z+rijk)))*x*log(rijk+y)-y*log(rijk+x))
    enddo
  endif
  call syncthreads()
end subroutine
```

**Fig. 6.** CUDA FORTRAN code for calculating the forward kernel matrix.
according to \( nx+1 \) and \( ny+1 \), the grid size is defined as \((\text{ceiling}((nx+1)/\text{dimblock}x), \text{ceiling}(ny+1)/\text{dimblock}y), 1)\). The GPU code written in CUDA FORTRAN for calculating the node density vector is presented in Fig. 5.

3.2. Calculate the sensitivity matrix \( G \)

Because each element of the sensitivity matrix can be calculated independently with relatively few input parameters, this part can be performed very efficiently by parallel computing. For the size of forward problems that we aim to solve, the sensitivity matrix, with dimensions of the number of data points \( no \) times the number of 3D volume cells \((nx+1)(ny+1)(nz+1)\), is too large to keep in memory. The solution is to parallelize over the number of grid cells \((nx+1)(ny+1)(nz+1)\), i.e., a single row of the matrix \( G \). This is especially important for the accelerators, which have limited memory capacity, e.g., 1023 MB onboard memory in a GTX 650Ti GPU, and high cost of data transfer between the host and the accelerator (Moorkamp et al., 2010; Čuma and Zhdanov, 2014). Fig. 6 shows the core algorithm of this step using CUDA FORTRAN.

3.3. Calculate the gravity anomalies

In this step, the gravity anomalies are calculated in two steps. The first step involves multiplication of the corresponding data of the node density vector and the calculated row of the sensitivity matrix \( G \). The second step involves summation of the multiplication result with the reduction algorithm (Greg and Massimiliano, 2011; Chen et al., 2012a, 2012b).

4. Experiment and performance

This section shows the performance gain that we can achieve with our new forward method. A synthetic random density model with 256 observation points, where each cell of the model is randomly assigned a density between 0.1 and 3.0 g/cm\(^3\), was used to validate the efficiency and accuracy of our method on both a CPU and a GPU.

To compare the performance of our method with the classical method, six forward approaches were used in our experiment: (1) the classical method with one CPU processor; (2) the proposed method with one CPU processor; (3) the classical method with four CPU processors; (4) the proposed method with four CPU processors; (5) the classical method with GPU accelerator (Moorkamp et al., 2010; Chen et al., 2012b); and (6) the proposed method with GPU accelerator.

![elapsed_time.png](image-url)

**Table 1**

<table>
<thead>
<tr>
<th>Name</th>
<th>Classical GPU implementation</th>
<th>Proposed GPU implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Calculate kernel matrix</td>
<td>2.26639 s</td>
<td>466.62 ms</td>
</tr>
<tr>
<td>[CUDA memcpy DtO]</td>
<td>81.356 ms</td>
<td>96.329 ms</td>
</tr>
<tr>
<td>Sum_cuda</td>
<td>32.048 ms</td>
<td>29.581 ms</td>
</tr>
<tr>
<td>Vecmul_cuda</td>
<td>21.928 ms</td>
<td>22.244 ms</td>
</tr>
<tr>
<td>Density vector</td>
<td>512.48 μs</td>
<td>370.48 μs</td>
</tr>
<tr>
<td>Node density vector</td>
<td>[CUDA memcpy HtoD]</td>
<td>371.91 μs</td>
</tr>
<tr>
<td>Total</td>
<td>2.78 s</td>
<td>0.67 s</td>
</tr>
</tbody>
</table>

![plot.png](image-url)

Fig. 7. Elapsed time of the six approaches: (1) classical method with one CPU processor; (2) proposed method with one CPU processor; (3) classical method with four CPU processors; (4) proposed method with four CPU processors; (5) classical method with GPU accelerator (Moorkamp et al., 2010; Chen et al., 2012b); and (6) proposed method with GPU accelerator.
We run all the tests on an Intel i3-2100 CPU with 3.1 GHz, 8 GB of main memory, and an NVIDIA GTX 650Ti graphics card, which has 768 processor cores and 1023 MB of onboard memory with a bandwidth of 144.2 G/s. The FORTRAN compiler and GPU development toolkit is PGI Visual FORTRAN 12.10. All the code is compiled with PGFORTRAN version 12.10 under Centos 6.4 using the “-O3” optimization flag, the “-Mcuda” flag for the CUDA FORTRAN implementation, and the “-mp” flag for the OpenMP implementation. Because the performance with different execution block sizes varies little, we therefore choose a block size of 256 for all subsequent experiments and do not attempt to optimize this value (see also Moorkamp et al. (2010)).

The elapsed times of the six approaches for varying model sizes between 8 and 32 million model cells and 256 stations are shown in Fig. 7. We can see that the residual node density-based method run more than four times faster than the classical method in the same computational environment. When extended to GPU parallel computing, the proposed method achieve more than two orders of magnitude acceleration relative to the classical method on one GPU processor when the model size is greater than 1 million. Table 1 shows the elapsed time for each step in the GPU-based approaches. We can see that the node-based method greatly reduced the elapsed time for calculating the kernel matrix, as well as the total elapsed time.

The computing precision of the results is of significant importance in these types of studies and is carefully considered here as well. Because single precision showed low accuracy in Moorkamp’s study (2010), all the tests described below are performed with double precision. Varying sizes of random density model are used for the computational accuracy test. Fig. 8 shows the histograms of the absolute and relative differences between the results from the proposed GPU code and the results from the classical CPU code. The relative difference histograms show that most values are concentrated between $-1 \times 10^{-12}$ and $+1 \times 10^{-12}$. The minimum and maximum relative differences are $-2.5 \times 10^{-12}$ and $+2.5 \times 10^{-12}$. Meanwhile, the minimum and maximum absolute differences are $-4 \times 10^{-4}$ mGal and $+4 \times 10^{-4}$ mGal. This shows that the results are in agreement with theoretical values in numerical precision and are identical for practical purposes.

5. Conclusion

We have developed a simple and fast approach for computing gravity anomalies at arbitrary points from a 3D density model. In our approach, the density model was first transformed into a residual node density model, and then the gravity anomalies at arbitrary points can be calculated with the simplified forward formula. The method was implemented on a CPU–GPU heterogeneous parallel architecture using the CUDA FORTRAN language. When compared with the classical method, the proposed method achieved high speed-up ratios, especially for large volumes of data. This approach is easily extendable to forward calculations of other quantities such as gravity gradients, gravity gradient tensors, and their relevant properties. The proposed method provides us an efficient way to obtain the gravity anomalies of large volumes density distributions as required in an iterative gravity inversion without storing the sensitivity matrix.

Acknowledgments

We would like to thank the two anonymous reviewers for constructive comments on this paper. This work was funded by the Fundamental Research Funds for the Central Universities (No. 35732015013), the National High Technology Research and Development program (“863” Program) of China (Nos. 2013AA063901-4 and 2013AA063905-4), the R&D of Key Instruments and Technologies for Deep Resources Prospecting (The National R&D Projects for Key Scientific Instruments) (No. ZDYZZ2012-1-02-04), and the Constrained Multi-Parameter Inversion of Geophysical Technology and Software Systems (No. 2014AA06A613).

References

Greg, R., Massimiliano, F., 2011. CUDA FORTRAN for Scientists and Engineers. Morgan Kaufmann.

Fig. 8. Difference in calculated gravity anomalies between the proposed GPU approach and the classical CPU-based method: (Left) histogram of the absolute differences (units: mGal); (Right) histogram of the relative differences.


