GICUDA: A parallel program for 3D correlation imaging of large scale gravity and gravity gradiometry data on graphics processing units with CUDA

Zhaoxi Chen \textsuperscript{a,b}, Xiaohong Meng \textsuperscript{a,b,*}, Lianghui Guo \textsuperscript{a,b}, Guofeng Liu \textsuperscript{a,b}

\textsuperscript{a} State Key laboratory of Geological Process and Mineral Resources, Beijing 100083, China
\textsuperscript{b} School of Geophysics and Information Technology, China University of Geosciences, Beijing 100083, China

\textbf{A R T I C L E  I N F O}

Article history:
Received 20 December 2011
Received in revised form 18 April 2012
Accepted 20 April 2012
Available online 10 May 2012

Keywords:
Gravity
Gravity gradiometry
GPU
CUDA
Speedup ratio

\textbf{A B S T R A C T}

The 3D correlation imaging for gravity and gravity gradiometry data provides a rapid approach to the equivalent estimation of objective bodies with different density contrasts in the subsurface. The subsurface is divided into a 3D regular grid, and then a cross correlation between the observed data and the theoretical gravity anomaly due to a point mass source is calculated at each grid node. The resultant correlation coefficients are adopted to describe the equivalent mass distribution in a quantitative probability sense. However, when the size of the survey data is large, it is still computationally expensive. With the advent of the CUDA, GPUs lead to a new path for parallel computing, which have been widely applied in seismic processing, astronomy, molecular dynamics simulation, fluid mechanics and some other fields. We transfer the main time-consuming program of 3D correlation imaging into GPU device, where the program can be executed in a parallel way. The synthetic and real tests have been performed to validate the correctness of our code on NVIDIA GTX 550. The precision evaluation and performance speedup comparison of the CPU and GPU implementations are illustrated with different sizes of gravity data. When the size of grid nodes and observed data sets is 1024 \times 1024 and 1024 \times 1024, the speed up can reach to 81.5 for gravity data and 90.7 for gravity vertical gradient data respectively, thus providing the basis for the rapid interpretation of gravity and gravity gradiometry data.

\section{1. Introduction}

As one of the oldest geophysical methods, gravity exploration is still used in a wide range of fields including oil and gas exploration, tectonics studies, geological mapping, archaeo-geophysical surveying, as well as engineering investigations (Nettleton, 1976; Clark, 1986; Blakely, 1995; Jones et al., 2002; Kearey et al., 2002; Zeng, 2005). With dramatic improvements of measuring equipment, the airborne gravity gradiometry has made great progress and is preferred because of higher resolution (Bell et al., 1997). In the meanwhile, the interpretation of gravity and gradiometry data has also been and continues to be one of the most attractive fields of geophysical research worldwide. There mainly involves two important issues. On the one hand, we should consider good interpretation methods that can help solve important exploratory problems. One of the notable interpretation methods for gravity and gradiometry data is constrained inversion (Li and Oldenburg, 1996, 1998, 2003; Portniaguine and Zhdanov, 1999, 2002; Pilkington, 1997, 2009). It is well known that a gravity inversion without constraints can be of little significance, because observed data can reconstruct many different models (Tikhonov and Arsenin, 1977; Yang, 1997; Scales et al., 2001). This nonuniqueness is mainly derived from the following two reasons. Firstly, this is a combination of physics and the fact that measurements are only taken along one axis at one depth. The second is the inaccuracy of the observed data due to noise. To deal with this nonuniqueness, many different kinds of a priori information known as geological, geophysical and drilling data and some mathematical constrained factors are added to seek a reasonable geo-physical solution. On the other hand, we need to consider the efficiency of the interpretation method such as constrained inversion. When the inversion extends from 2D to 3D, it is in general computationally expensive and parallel computational resources or even supercomputers are often utilized to obtain a result (Yao et al., 2003, 2007; Moorkamp et al., 2010).

3D correlation imaging approach involves directly calculating the cross correlation between the observed data and the theoretical response due to a point mass source at each node of a subsurface divided by 3D regular grids (Guo et al., 2009, 2010, 2011). The results of correlation coefficients are used to describe the equivalent mass distribution. It was derived from 3D gravity probability tomography approach proposed by Mauriello and
Patella (2001a, 2001b), and Iuliano et al. (2002). These approaches have the advantage of requiring no priori information and only demanding low amount of computer memory. However, when both the observed data and grid nodes of subsurface are of very large scale, it still consumes large amount of computing time.

Over the last few years, there has been a dramatic increase in the number of publications on the applications of GPU parallel computing (Mark et al., 2003a, 2003b; Che et al., 2008; Schenk et al., 2008; NVIDIA Corporation, 2007, 2009, 2010a, 2010b). At present, the GPUs are most economical and powerful computational hardware with the advantage of more excellent performance, and have stepped into a new research field that explores the performance of GPUs for general purpose computation. The general purpose computation on GPUs is attracting more and more attention of many researcher and developers, especially who are urgent to solve time-consuming problems. Although GPUs have become a compelling platform for general-purpose computation, there are some limitations and difficulties to program them. Early GPU programming languages were based on shaders in various off-line rendering systems, which make the implementation complicated because it needs additional code to bypass the nature of GPUs. CUDA (Compute Unified Device Architecture), released by NVIDIA, is an extended C language programming environment that avoids working with a shading language. It includes the CUDA Instruction Set Architecture and the parallel compute engine. It also provides a compiler with extension to C programming language and two libraries, basic linear algebra subroutines (CUBLAS) and fast Fourier transform (CUFFT). With CUDA, GPU parallel computing has been widely applied in seismic processing, astronomy, molecular dynamics simulation, fluid mechanics and some other fields (Hegeman et al., 2006; Stone et al., 2007; Tolke, 2008; Bellemann et al., 2008; Li et al., 2009; Gaburov et al., 2009; Liu et al., 2009; Wong et al., 2011). We can find detailed program guide and SDKs on GPUs with CUDA in the literature of Sanders and Edward Kandrot (2010) and NVIDIA Corporation (2010a, 2010b).

In this paper, we present a parallel program for 3D correlation imaging for large scale gravity and gravity gradiometry data to accelerate the computing process on GPU with CUDA, called GICUDA. First of all, we give a brief description of the CUDA in Section 2. Secondly, we review the principle of 3D correlation imaging in Section 3. Next, we present the GPU implementation in detail in Section 4 and numerical tests are given in Section 5. We present our conclusions and indicate some directions for possible future work in Section 6.

2. A brief introduction to the CUDA

As a general purpose parallel computing architecture, CUDA includes both hardware architecture and software components (CUDA compiler and the system drivers and libraries). It can be used for issuing and managing computations on the GPU with the need of mapping them to a graphics API, which is available for the GeForce 8800 series and beyond. The CUDA programming model consists of functions, called kernels, which can be executed simultaneously by a large number of lightweight threads on the GPU. These threads are grouped into one-, two-, or three-dimensional block blocks, which are further organized into one- or two-dimensional grids. Threads are grouped into batches of 32 called warps which are executed in single instruction multiple data (SIMD) fashion independently. Threads within a warp execute a common instruction at a time. All the threads in the same block can not only be executed in parallel, but also achieve inter-thread communication through shared memory and barrier synchronization. In this way, different blocks in the same grid involve coarse-grained parallel computing that does not need communication, while the threads in one block involve fine-grained parallel computing that allows communication. Therefore, the key feature of CUDA is that threads are organized by two levels and can communicate with each other by shared memory and fence synchronization in lower level.

For memory access and usage, there are four types of memory, namely, global memory, constant memory, texture memory as well as shared memory. Global memory has its own separate address space for mapping data from the host CPU's main memory through the PCIe bus, which is about 8 GB/s in the GT200 GPU. Any values stored in the global memory can be accessed by all SMs via load and store instructions. Constant memory and texture memory are cached, read-only and shared between SPs. Constants that are kept unchanged during kernel execution may be stored in constant memory. Although shared memory provides much faster access rate than global memory, its size is very limited. So it is of significant importance that the memory arrangement should be carefully considered. For detailed information concerning memory optimizations, we refer the reader to the literature of NVIDIA Corporation (2010a, 2010b).

One important issue that should be carefully considered in GPU computing is the optimization of CUDA program. In general, we should explore two important aspects in the characteristics of GPU. To begin with, the program should be highly parallelized and the number of single block threads should satisfy the optimal allocation of resources, thus performing sustained high-density GPU computing. The second is that we should consider the access and storage of different kinds of memory introduced above. Memory bandwidth is one important aspect of the bottleneck of computer performance. Typically, the processor computing power exceeds far more than memory access bandwidth. The access to global memory should satisfy coalescing. If the data is well organized in the global memory with the form that a load statement in all threads in a warp accesses data in the same aligned 128-byte block, then the threads can efficiently access data from the global memory. In addition, the rational use of register memory, obtaining data in advance to reduce GPU waiting time, the thread-memory data access sequentially and avoiding branching statements are also important factors that affect the efficiency of GPU computing.

3. Theory of 3D correlation imaging

Large parts of the 3D correlation imaging methodology and associated mathematical concepts are described by Guo et al. (2009). They also compared their method with the 3D gravity probability tomography approach and pointed out that their correlation coefficient function is similar to the so-called Λ-mass occurrence probability function of Mauriello and Patella(2001a, 2001b), except that no topography surface regularization function (Mauriello and Patella 2001a) is involved in the formula. Here we simply review the theory of the correlation imaging method. In a 3D case, a reference rectangular coordinate system with horizontal (x,y)-plane at sea level and z-axis positive downwards is chosen at a survey area. Assume that an arbitrary point mass \( q(x_0,y_0,z_0) \) is present in the subsurface, its volume is \( v_0 \), and its density contrast is \( \Delta \rho \) (Fig. 1). We can calculate the theoretical gravity anomaly at an arbitrary surface station \((x_i, y_i, z_i)\) for the point mass \( q(x_0,y_0,z_0) \) by

\[
\Delta g_i(x_i,y_i,z_i) = G\Delta \rho q v_0 B_i(x_i,y_i,z_i),
\]

(1)

where \( G \) is the universal gravitation constant, and \( B_i(x,y,z) \) is the geometrical function of the point mass \( q \) for gravity at the
the correlation coefficient function is

\[ C_q = \frac{\sum_{i=1}^{N} \Delta g_{\text{obs}}(x_i,y_i,z_i)B_q(x_i,y_i,z_i)}{\sqrt{\sum_{i=1}^{N} \Delta g_{\text{obs}}^2(x_i,y_i,z_i) \sum_{i=1}^{N} B_q^2(x_i,y_i,z_i)}} \]

Calculating the derivative of formula (1) in different directions, we obtain the theoretical gradiometry anomaly at the surface station \((x_i,y_i,z_i)\) for the point mass \(q(x_i,y_i,z_q)\) by

\[ \Delta g_{x,y,q}(x_i,y_i,z_i) = G \Delta \rho \psi_q B_{x,y,q}(x_i,y_i,z_i), \]

where \(\alpha\) and \(\beta\) represent the gradiometry tensor with different derivative directions. The geometrical functions of the point mass \(q\) for gradiometry anomaly at the station \((x_i,y_i,z_i)\) are listed below:

\[ B_{x,y,q}(x_i,y_i,z_i) = \frac{2(x_q-x_i)^2(y_q-y_i)^2-(z_q-z_i)^2}{[(x_q-x_i)^2+(y_q-y_i)^2+(z_q-z_i)^2]^{3/2}}, \]

\[ B_{y,z,q}(x_i,y_i,z_i) = \frac{2(y_q-y_i)^2(x_q-x_i)^2-(z_q-z_i)^2}{[(x_q-x_i)^2+(y_q-y_i)^2+(z_q-z_i)^2]^{3/2}}, \]

\[ B_{z,x,q}(x_i,y_i,z_i) = \frac{2(z_q-z_i)^2(x_q-x_i)^2-(y_q-y_i)^2}{[(x_q-x_i)^2+(y_q-y_i)^2+(z_q-z_i)^2]^{3/2}}, \]

\[ B_{x,y,q}(x_i,y_i,z_i) = B_{y,z,q}(x_i,y_i,z_i) = \frac{3(x_q-x_i)(y_q-y_i)}{[(x_q-x_i)^2+(y_q-y_i)^2+(z_q-z_i)^2]^{3/2}}, \]

\[ B_{y,z,q}(x_i,y_i,z_i) = B_{z,x,q}(x_i,y_i,z_i) = \frac{3(y_q-y_i)(z_q-z_i)}{[(x_q-x_i)^2+(y_q-y_i)^2+(z_q-z_i)^2]^{3/2}}, \]

\[ B_{z,x,q}(x_i,y_i,z_i) = B_{x,y,q}(x_i,y_i,z_i) = \frac{3(z_q-z_i)(x_q-x_i)}{[(x_q-x_i)^2+(y_q-y_i)^2+(z_q-z_i)^2]^{3/2}}. \]

Assuming \(\Delta \rho_q > 0\), we can substitute Eqs. (2) and (4) into Eq. (6) yields

\[ C_q = \frac{\sum_{i=1}^{N} \Delta g_{\text{obs}}(x_i,y_i,z_i)B_q(x_i,y_i,z_i)}{\sqrt{\sum_{i=1}^{N} \Delta g_{\text{obs}}^2(x_i,y_i,z_i) \sum_{i=1}^{N} B_q^2(x_i,y_i,z_i)}}, \]

\[ C_{z,y,q} = \frac{\sum_{i=1}^{N} \Delta g_{\text{obs}}(x_i,y_i,z_i)B_{z,y,q}(x_i,y_i,z_i)}{\sqrt{\sum_{i=1}^{N} \Delta g_{\text{obs}}^2(x_i,y_i,z_i) \sum_{i=1}^{N} B_{z,y,q}^2(x_i,y_i,z_i)}} \]

According to the Schwarz’s inequality property, we know that the correlation coefficient \(C_q\) in Eq. (7) satisfies the condition \(-1 \leq C_q \leq 1\). The property of \(C_q\) is the same as that of \(C_q\) (similarly hereinafter).

The values of \(C_q\) describe the cross correlation degree between the observed gravity anomaly and the corresponding theoretical gravity anomaly due to different point mass \(q\). It reflects a qualitative probability that an anomalous mass at the point \(q\) is responsible for the observed data. A positive value of \(C_q\) indicates the influence of mass excess concentrated at the point \(q\), while negative values are the results of a mass deficiency at the same point. The closer the absolute value of \(C_q\) is to 1, the higher the qualitative probability of an excess or deficient mass at the point \(q\).

4. GPU parallelization

In this section, we firstly analyze the algorithm of correlation imaging and propose the GICUDA design which includes the idea of parallel algorithm of correlation imaging with CUDA. Then we provide the program flow of GICUDA with implementation details, with which the main time-assuming computations are performed on GPUs and the temporary and final results are stored in the GRAM of the graphics card. Eventually, we further illustrate the optimization of GICUDA in the parallel programming.

4.1. GICUDA design

In CUDA program model, CPU and GPU work together and carry out their respective duties. Once the parallel parts of the program are determined, it is advisable to consider the computing work performing in GPU device. The function running on GPU is called kernel. In fact, the kernel function is not a complete program. It is only a part of the program that executes in GPU device for entire CUDA program. So we should analyze the algorithm and find out which part can be transformed into parallel GPU program. Specifically, for the realization of 3D correlation imaging, the main part of computing needs five loops. The serial CPU pseudo codes are as follows:

for \(k=1:NZ\) do
  for \(i=1:NX\) do
    for \(j=1:NY\) do
      \(\text{compute } \text{the } (i,j,k)\)th mass point
      \(\text{end}\)
  \(\text{end}\)
\(\text{end}\)

for \(i=1:NY\) do
  for \(j=1:NX\) do
    \(\text{compute } \text{the } (i,j)\)th mass point
    \(\text{end}\)
  \(\text{end}\)

The corresponding pseudo codes on CUDA are as follows:

for \(i=1:NY\) do
  for \(j=1:NX\) do
    \(\text{compute } \text{the } (i,j)\)th mass point
    \(\text{end}\)
  \(\text{end}\)

for \(k=1:NZ\) do
  \(\text{compute } \text{the } (i,j,k)\)th mass point
  \(\text{end}\)
\(\text{end}\)

We define the correlation coefficient function between the observed gravity or the gravity gradiometry anomaly and the theoretical anomaly caused by the point mass \(q\) based on Pearson’s linear correlation formula. For the gravity anomaly data the correlation coefficient function is

\[ C_q = \frac{\sum_{i=1}^{N} \Delta g_{\text{obs}}(x_i,y_i,z_i)B_q(x_i,y_i,z_i)}{\sqrt{\sum_{i=1}^{N} \Delta g_{\text{obs}}^2(x_i,y_i,z_i) \sum_{i=1}^{N} B_q^2(x_i,y_i,z_i)}}, \]

Likewise, for the gravity gradiometry tensor data the correlation coefficient function is

\[ C_{z,y,q} = \frac{\sum_{i=1}^{N} \Delta g_{\text{obs}}(x_i,y_i,z_i)B_{z,y,q}(x_i,y_i,z_i)}{\sqrt{\sum_{i=1}^{N} \Delta g_{\text{obs}}^2(x_i,y_i,z_i) \sum_{i=1}^{N} B_{z,y,q}^2(x_i,y_i,z_i)}}, \]

where \(\Delta g_{\text{obs}}(x_i,y_i,z_i)\) is the observed gravity or gravity gradiometry anomaly and \(N\) is the total number of observed stations.
\[ C_q[i][j][k] = \frac{\text{tmp1}}{\text{pow}(\text{tmp2} \times \text{tmp3}, 0.5)}; \]

//Cq-the correlation coefficient value;
//tmp3-the square sum of the observed data.
end
end
end

Noted that the CUDA kernel will be exploited to parallelly process all grid nodes using threads, a for instruction is not needed for going through all grid nodes. We can transfer them to GPU device to realize parallel computing. The parallel pseudo codes are as follows:

for \( k = 1 : NZ \) //loop1 NZ-the number of grid node in \( z \) direction
for \( i = 1 : NY \) //loop2 NY-the number of grid node in \( y \) direction
for \( j = 1 : NX \) //loop3 NX-the number of grid node in \( x \) direction
    InnerLoopinGPU(tmp1,tmp2);//calculate tmp1 and tmp2 in GPU device
    //Cq-the correlation coefficient value;
    //tmp3-the square sum of the observed gridding data.
    \[ C_q[i][j][k] = \frac{\text{tmp1}}{\text{pow}(\text{tmp2} \times \text{tmp3}, 0.5)}; \]
end
end
end

However, the CUDA design above is ideal, because we do not care about the capacity of computing device when the number of observed data and the grid nodes of the subsurface are increasingly large. We consider the allocation of threads and blocks for different scales of data sets in GPU device. According to the CUDA execution model, each block of the grid will be assigned to each stream multiprocessor in the implementation. To effectively hide the pipeline delay, we should firstly consider the block size and not care about the grid size. The straightforward way to write a parallel program for 3D correlation imaging is to use two dimensional arrays for data storage and two dimensional threads for the computation. We define the dimension of each block a multiple of 16 threads. The main reason is that this block size works well for most GPUs and we then adjust the grid size to fit complete blocks into calculations. Here we define the size of 2D block as 16 \( \times \) 16. Correspondingly, the minimum number of threads is estimated for computing one grid node. Then the maximum number of grid nodes computing one time in parallel can be calculated according to the dimension of blocks. Fig. 2 illustrates the index of both blocks and threads in parallel computing in GPU device. Green grid cells represent blocks, in which include 16 \( \times \) 16 threads. Orange grid cells represent the number of threads that are used to calculate one grid node. Depending on the block size, the block number and the model size, we might have some extra threads in the last blocks for which we do not need to perform any calculations. We therefore have to check whether the index in \( x \) and \( y \) direction

![Fig. 2](image-url)
is smaller than the $nx$ and $ny$ respectively. The effective index representations are listed as follows.

$$\begin{align*}
\text{tx} &= \text{blockIdx.x} \times \text{blockDim.x} + \text{threadIdx.x}; \\
\text{ty} &= \text{blockIdx.y} \times \text{blockDim.y} + \text{threadIdx.y}; \\
\text{xsiz} &= \text{int}(nx/\text{blockDim.x}); \\
\text{ysiz} &= \text{int}(ny/\text{blockDim.y}); \\
\text{i} &= \text{blockIdx.x}/\text{xsiz}; \\
\text{i1} &= \left(\text{blockIdx.x}\%\text{xsiz}\right) \times \text{blockDim.x} + \text{threadIdx.x}; \\
\text{j} &= \text{blockIdx.y}/\text{ysiz}; \\
\text{j1} &= \left(\text{blockIdx.y}\%\text{ysiz}\right) \times \text{blockDim.y} + \text{threadIdx.y}; \\
\text{tid} &= \text{threadIdx.y} \times \text{blockDim.x} + \text{threadIdx.x}; \\
\text{bid} &= \text{blockIdx.y} \times \text{gridDim.x} + \text{blockIdx.x};
\end{align*}$$

To sum up, GICUDA implementation includes the following steps:

1. Initialize CUDA.
2. Input observed gravity or gravity gradiometry data into RAM, obtain the parameter $nx$ (the number of grid nodes in $x$ direction), $ny$ (the number of grid nodes in $y$ direction), $dx$ (the grid interval in $x$ direction), $dy$ (the grid interval in $y$ direction), and set parameters such as $nz$ (the number of grid nodes in $z$ direction), $z_0$ (the height of the observed data), $dz$ (the grid interval in $z$ direction), etc.
3. Copy the observed data to the global memory and parameters to registers.
4. Based on the maximum of the blocks and defined threads, estimate the calculating number of grid nodes.
5. Perform parallel optimized reduction in each block, and store the temporary variables in the shared memory of each block.
6. Execute parallel optimize reduction for the specific blocks to calculate correlation coefficients of the specific grid nodes and stored the results into global memory.
7. Transfer back the results to host memory.
8. Output the results.

### 4.2. Parallel program optimization

As mentioned in Section 2, we need to use different kinds of memory as much as possible. Parameters and temporary results of each kernel are stored with registers. Although the size of shared memory is very limited, we adopt it to store the reduction results within one block because it can provide much faster access.
rate than global memory. Besides, the program itself should also be considered and revised to obtain further optimization. While making the best of each thread and block, we adopt the optimized parallel reduction (Fig. 3). The optimized parallel reduction avoids divergent branches and bank conflict and conducts sequential addressing and loop unrolling, which make the implementation obtain the best performance. We can refer to CUDA programming guide.

5. Results

We give two numerical examples from synthetic and field data sets to show effectiveness of the GICUDA implementation performed in the NVIDIA GTX 550 card. We also evaluate precision of the GPU based program and present a comparison in speedup performance between the CPU and GPU implementations using different sizes of gravity data.

5.1. Numerical tests

The synthetic model consists of 7 rectangular bodies with different sizes and density contrasts at different depths, including A1 and A2, the larger cuboids in the deepest layers, B1, B2 and B3 located in the middle layer, and C1 and C2, the smallest cuboids in the shallowest layers (Fig. 4). The length of the bodies in X direction, width in the Y direction, thickness in the Z direction, depth to the top and density contrasts are listed in Table 1. The corresponding vertical gravity gradient response of the synthetic model contaminated by a Gaussian noise of 1 Eotvos and 2% the datum magnitude is shown in Fig. 5. The correlation imaging is performed for the vertical gravity gradient response within depths ranging from 0 to 5000 m and with a depth interval of 120 m. The results using GPU implementation are in agreement with those with CPU implementation.

Table 1 The parameters and density contrasts of each cell in the synthetic model.

<table>
<thead>
<tr>
<th>Cuboid no.</th>
<th>Length along x (m)</th>
<th>Length along y (m)</th>
<th>Thickness along z (m)</th>
<th>Depth to top (m)</th>
<th>Density contrast ( (\times 10^{-3} \text{ kg/m}^3) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>4000</td>
<td>5000</td>
<td>1500</td>
<td>1500</td>
<td>0.15</td>
</tr>
<tr>
<td>A2</td>
<td>3000</td>
<td>7000</td>
<td>1500</td>
<td>1500</td>
<td>-0.1</td>
</tr>
<tr>
<td>B1</td>
<td>400</td>
<td>1600</td>
<td>400</td>
<td>400</td>
<td>-0.25</td>
</tr>
<tr>
<td>B2</td>
<td>400</td>
<td>1000</td>
<td>400</td>
<td>400</td>
<td>0.25</td>
</tr>
<tr>
<td>B3</td>
<td>600</td>
<td>800</td>
<td>400</td>
<td>400</td>
<td>0.25</td>
</tr>
<tr>
<td>C1</td>
<td>100</td>
<td>200</td>
<td>100</td>
<td>100</td>
<td>0.5</td>
</tr>
<tr>
<td>C2</td>
<td>100</td>
<td>200</td>
<td>100</td>
<td>100</td>
<td>-0.5</td>
</tr>
</tbody>
</table>

We now consider an application of 3D correlation imaging to a field gravity data set in the Dabie–Sulu ultrahigh-pressure metamorphic (UHPM) belt in China, which is the largest recognized UHPM belt in the world (Fig. 7). It was formed by the Triassic collision of the Sino-Korean and Yangtze Cratons with the peak metamorphic age of 220–245 Ma (Hacker et al., 1998; Rowley et al., 1997; Ames et al., 1996). The UHPM rocks of the Dabie–Sulu metamorphic belt have been subducted to at least 100 km depth and experienced UHPM before being rapidly exhumed from the mantle back to the surface. This subduction resulted in part of the continental crust subsiding into the upper mantle. From the Late Jurassic, large-scale granitoid intrusions developed around the Dabie–Sulu areas and other parts of eastern China, possibly resulting from lithospheric thinning and heat flow upwelling from the asthenosphere. The granitic intrusions were followed by rifting and eruption of basalts until Pliocene when the mountain root of the orogenic belt almost erodes out (Yang, 2003). In order to understand the present lithosphere structure of Dabie–Sulu orogeny, geophysicists carried out lots of field work, collected abundant data to study the density, electrical and velocity structure of the lithosphere of Dabie and Sulu Terrance and proposed some constructive interpretations (Luo et al., 2011; Yang, 2003).

Fig. 8 shows the maps of residual gravity anomaly and the gravity vertical gradient anomaly respectively in the red dashed region in Fig. 7. The gravity survey covers an area with 4500 m in...
results using GPU implementation are in agreement with those with CPU implementation. We can conclude from the Fig. 9 that the result based on the gravity vertical gradient data has better resolution than that from the gravity data. This can provide the basis for the preliminary interpretation.

5.2. Precision and speedup comparison of GPU and CPU

Considering the rapid development of graphics hardware, our GPU implementation was designed in general for the GT200 architecture and the Fermi architecture. We run all the tests on Xeon CPU with 3.07 GHz, 4 G of main memory and the graphic cards NVIDIA GTX 550. The C compiler environment and GPU development toolkit used are Visual Studio 2008 and CUDA 4.0 respectively.

Computing results' precision evaluation is of significant importance and should be carefully considered. The program of correlation imaging involves only simple mathematical algorithms, so the computing results have very little difference using between single precision and double precision. Also, the results of GPU are in agreement with those obtained by CPU. So we examine the speedup ratio of performing the GICUDA implementation in single precision on GPU to obtain higher accelerating performance.

Fig. 10 shows a computing time comparison of the GPU with the single thread CPU implementation of 3D correlation imaging for gravity data. The figure shows some interesting characteristics. When the number of grid nodes and corresponding observed data sets is very small, the GPU computation time is longer than CPU. This demonstrates the overhead associated with initializing the GPU and transferring data between main memory and the memory of the graphics card (Moorkamp et al., 2010). For more than \(32 \times 32 \times 1\) grid nodes and \(32 \times 32\) observed gravity data sets, the computation time of the GPU based implementation exceeds that of the single thread CPU implementation, and the speedup ratio increases as the number of grid nodes and observed gravity data becomes larger. Specifically, there is about 81.5 times

Fig. 7. Simplified geological map of the Dabie–Sulu terrane. (Modified after Luo et al., 2011; Yang, 2003; the red dashed region represents the study area).

Fig. 8. The gravity anomaly map and the vertical gradient anomaly in UHPM belt, China.
speedup when the size of grid nodes is $1024 \times 1024 \times 1$ and observed data sets $1024 \times 1024$ respectively.

Fig. 11 gives the comparison of the GPU and single thread CPU implementation of 3D correlation imaging for gravity vertical gradient data, which has the same general behavior as for gravity data. The computing speed up can reach to 90.7 when the size of grid nodes is $1024 \times 1024 \times 1$ and observed data sets $1024 \times 1024$ respectively. The reason for the greater speed up is that the correlation imaging for vertical gradient data involves a complex interaction between memory transfer, calculations etc.

6. Conclusions and further work

A parallel program named GICUDA for correlation imaging of gravity and gravity gradiometry data is developed on GPU on CUDA platform. This paper is dedicated to illustrate the GICUDA implementation design in detailed information. The synthetic and real tests are performed to validate the correctness of our codes on NVIDIA GTX 550. The precision is evaluated and performance speedup comparison of the CPU and GPU implementations are compared using different sizes of gravity data. The computing results are reliable and the computation time is reduced, thus providing the basis for the rapid interpretation of large scale gravity and gravity gradiometry data.

It is notable that here we implicitly assume a regular grid of sites, where the number of sites in each horizontal direction is equal to the number of grid cells in that direction. However, this restriction is not necessary at all. The method works equally on completely arbitrary measurement geometry without any structure in the spacing of the sites and no relationship to the number of horizontal grid cells to be imaged. This can be achieved by replacing the two inner loops by a single loop over all sites and computing the geometric term (sf parameters) within that loop.

In addition, there are two possible directions in our further work we should take. Firstly, because the results of correlation imaging only represent probability distribution, we need to transform them to the real density contracts. After that, there is a need to incorporate some priori information in the process in order to obtain better results.

Fig. 9. The equivalent mass distribution from the 3D correlation imaging of the gravity anomaly (a) and the vertical gradient anomaly (b).

Fig. 10. Computation time comparison for gravity data between sing thread CPU implementation (Xeon CPU with 3.07 GHz, 4 G of main memory) and GPU implementation (NVIDIA GTX 550).
Acknowledgments

This work was supported by Sinoprobe Project (Grant Nos. 201011049, 201011039), the National Natural Science Foundation of China (41074095, 40904033), the Fundamental Research Funds for the Central Universities (Grant Nos. 2010Y226 and 2011PY1083) and the Open Project of State Key Laboratory of Geological Processes and Mineral Resources (Grant No.GPMR0945). The authors wish to thank the anonymous reviewers for their critical review and many helpful comments.

References

NVIDIA, 2007. NVIDIA CUDA compute unified device architecture programming guide. Santa Clara, CA.

Fig. 11. Computation time comparison for gravity vertical gradient data between single thread CPU implementation (Xeon CPU with 3.07 GHz, 4 G of main memory) and GPU implementation (NVidia GTX 550).

The number of grid nodes of subsurface (nx x ny x nz)
The number of observed data(nx x ny, the same as the number of grid nodes in x- and y direction)


