Stanford Exploration Project, Report 124, April 4, 2006, pages 1–10

Parallel implementation of image segmentation for tracking 3D salt boundaries

Jesse Lomask and Robert G. Clapp

ABSTRACT

We distribute the modified normalized cuts image segmentation with random boundaries algorithm on a parallel network to track 3D salt boundaries. We identify two key steps of this algorithm for parallelization. Firstly, we parallelize the calculation of the weight matrix. Secondly, we parallelize the matrix-vector product of the eigenvector calculation. This method is demonstrated to be effective on a 3D seismic cube.

INTRODUCTION

Normalized cuts image segmentation (Shi and Malik, 2000) is a pattern analysis technique developed to extract global impressions from images . Hale and Emanuel (2003, 2002) adapted this technique to painting 3D atomic meshes. Lomask and Biondi (2003); Lomask et al. (2004) have since applied this technique to tracking salt boundaries. In addition to sparse matrix storage and random sampling (Shi and Malik, 2000), other memory saving measures such as random bounds (Lomask and Biondi, 2005) have been implemented in order to reduce the size of problem thus reducing the exorbitant cost of applying this technique. However, to tackle realistic 3D data sets, the computational time of this algorithm still needs to be reduced significantly.

There are two significant computational time bottlenecks in the normalized cuts image segmentation with random bounds algorithm. This image segmentation technique creates a matrix containing weights relating each pixel to every other pixel in a local neighborhood. The weights are dependent on the negative absolute value of the complex trace (instantaneous amplitude) of the seismic. The matrix is then used to cut the image where the normalized sum of weights cut is minimized. This normalized cut is minimized by solving an eigenvector problem. The first bottleneck is the creation of the weight matrix as it can become quite large requiring a lot of computation time to build. The second bottleneck is the estimation of the eigenvector which requires numerous matrix-vector products involving the large sparse weight matrix.

In this paper, we present a parallel implementation of the normalized cuts image segmentation with random bounds technique for tracking 3D salt boundaries. We first review the algorithm. We then describe how we have distributed the calculation of the weight matrix on a parallel network. We then describe how we have parallelized the matrix-vector products of the eigenvector calculation. Lastly, we test this technique on a 3D field seismic cube.

METHODOLOGY

Normalized cuts image segmentation partitions images into two groups. To do this, it first creates weights relating each sample to randomly selected samples along paths within local neighborhoods. These weights are stored in a sparse matrix **W**. It then finds the cut that partitions the image into two groups, A and B, by minimizing the normalized cut:

$$N_{cut} = \frac{cut}{total_A} + \frac{cut}{total_B} \tag{1}$$

where *cut* is the sum of the weights cut by the partition. $total_A$ is the sum of all weights in Group A, and $total_B$ is the sum of all weights in Group B. Normalizing the cut by the sum of all the weights in each group prevents the partition from selecting overly-small groups of nodes.

The minimum of N_{cut} can be found by solving the generalized eigensystem:

$$(\mathbf{D} - \mathbf{W})\mathbf{y} = \lambda \mathbf{D}\mathbf{y}, \tag{2}$$

created from the weight matrix (**W**) and a diagonal matrix (**D**), with each value on the diagonal being the sum of each column of **W**. The eigenvector (**y**) with the second smallest eigenvalue (λ) is used to partition the image by taking all values greater than zero to be in one group, and its complement to be in the other. For a more detailed description, see Shi and Malik (2000).

Application to seismic

To apply this segmentation method to seismic data, the weight calculation needs to be modified. Rather than looking for clusters of pixels with similar intensity, we are now looking for groups of pixels on each side of the bright amplitude salt boundary. Therefore, we want the weights connecting pixels on either side of the salt boundary to be low and the weights connecting pixels on the same side of the salt boundary to be relatively high. Taking the negative of the maximum amplitude along the shortest path between two nodes as the weight would insure that the weights connecting pixels on either side of the salt boundary will be low. However weights on the same side would be alternating from low to high as they go from peak to trough on the seismic data. This could make the grouping more uncertain. To correct this problem, we take the negative of the maximum of the absolute value of the complex trace (instantaneous amplitude) along the shortest path between two nodes.

$\mathbf{v}_{ij}$ is a vector representing the shortest path between two nodes i and j but excluding the nodes themselves. The weight connecting two nodes i and j is determined from the minimum of the negative instantaneous amplitude A sampled along $\mathbf{v}_{ij}$ and a user specified tolerance μ as:

$$W_{ij} = \begin{cases} 0 & (\min \mathrm{A}(\mathbf{v}_{ij}) < \mathrm{A}(\mathbf{v}_i))\ \&\ (\min \mathrm{A}(\mathbf{v}_{ij}) < \mathrm{A}(\mathbf{v}_j))\ \&\ (\min \mathrm{A}(\mathbf{v}_{ij}) < \mu\ ) \\ 1 & \text{otherwise.} \end{cases} \tag{3}$$

The algorithm as designed by Shi and Malik (2000) is capable of using weights that are real-valued instead of binary as we are using here. We have found thus far that binary weights give the best results, but we still wish to experiment with real-valued weights. If we determine that binary weights are the best way to go, then we can take advantage of the cost savings of using logical arrays instead of real.

Random bounds

By applying bounds we greatly reduce the size of the problem. Even though the weight matrix (**W**) is stored as a sparse matrix (only non-zero values are stored), it can still be very large. Typically, the size of the sparse weight matrix is the size of the input image multiplied by the number of samples taken from each pixel's search neighborhood. In 2D, this can be twenty to thirty times the size of the input image. In 3D, this problem may be even worse. Therefore, any reduction in the size on the input image is helpful. These bounds can acquired from several sources. For instance, these initial rough bounds can be found by first running the algorithm with small search neighborhoods and coarse sampling.

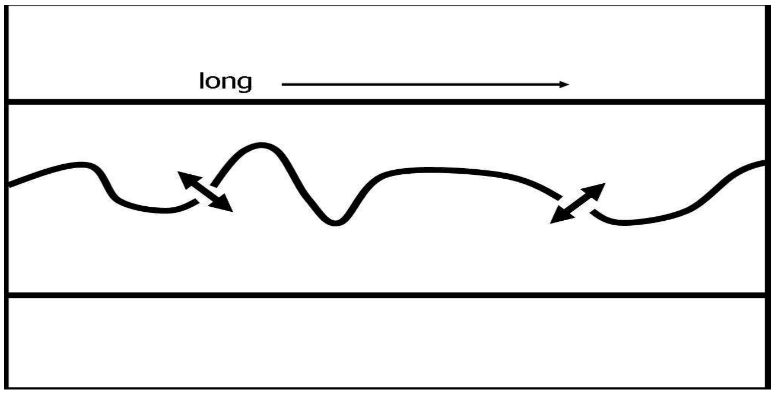

Figure 1: A cartoon of a masked salt boundary. It is long and thin with a discontinuous salt boundary snaking across it. jesse1-pic2 [NR]

Unfortunately, the normalized cut segmentation method tends to partition elongated images along their shortest dimension. For instance, Figure 1 is a cartoon of an elongated image with a salt boundary snaking across it. If the segmentation algorithm were to function as hoped, the minimum cut would be found along the salt boundary. However, because the salt boundary is discontinuous, it is likely that the minimum of the normalized cut in equation (1) will be found by cutting the image vertically where the image is thin.

To correct this problem, we exploit the fact that the upper boundary will necessarily be in Group A and the lower boundary will be in Group B. In other words, we want to force the segmentation algorithm to put the coarsely picked bounds in different groups.

We can enforce this constraint during the creation of the weight matrix (**W**). Recall that this matrix contains weights relating each sample to other samples along paths within a neighborhood. For any given sample, if its search neighborhood happens to cross a coarse boundary, it becomes weighted to another sample at a random distance along the boundary. This can be imagined by wrapping the image on a globe so that both the upper and lower bounds collapse to points at the poles. When estimating the weight matrix, every time a path crosses the north or south pole, it continues down the other side. By implementing these “Random” boundaries, we are effectively removing the upper and lower boundaries.

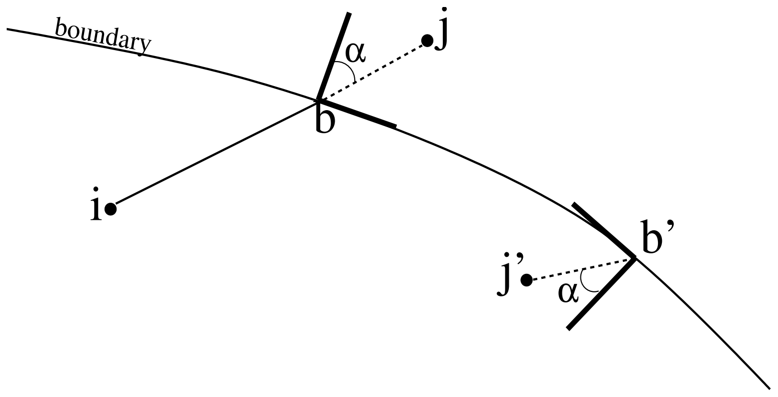

Figure 2: A cartoon illustrating random bounds. $\boxed{\text{jesse1-fig1}}$ [NR]

In Figure 2, two nodes, i and j, of a 2D image are plotted separated by a boundary indicating that node j is outside of the bounds. Vector $\mathbf{v}_{ij}$ represents the shortest path connecting them but excluding the nodes themselves. $\mathbf{n}_b$ and $\mathbf{t}_b$ are the unit normal and tangent vectors where $\mathbf{v}_{ij}$ crosses the boundary at location b. Together, $\mathbf{n}_b$ and $\mathbf{t}_b$, make a basis on to which vector $\mathbf{v}_{bj}$ is projected and then mapped on to another basis at $\mathbf{b}'$ at a random distance along the boundary. In summary, if j is outside the boundary, then $\mathbf{v}_{ij'}$ defines the path instead of $\mathbf{v}_{ij}$ as:

$$\begin{aligned}
\mathbf{v}_{ij} &= \mathbf{v}_{ib} + \mathbf{v}_{bj} && (4)\\
\mathbf{v}_{b'j'} &= \mathbf{b}' - (\mathbf{v}_{bj} \cdot \mathbf{n}_b)\mathbf{n}_{b'} - (\mathbf{v}_{bj} \cdot \mathbf{t}_b)\mathbf{t}_{b'} && (5)\\
\mathbf{v}_{ij'} &= \mathbf{v}_{ib} + \mathbf{v}_{b'j'}. && (6)
\end{aligned}$$

3D implementation

The most significant difference between 3D image segmentation and 2D image segmentation is during the generation of the weight matrix, the rest of the algorithm is almost identical. When creating the weight matrix, instead of randomly sampling from a circular neighborhood, we sample from a sphere. Of course this means that more points are sampled per node. This, in turn, means, that the sparse matrix is considerably less sparse and the entire algorithm more expensive. Therefore, even with sparse matrices and tight boundaries, we still need to look for ways of reducing the computional-time cost of this algorithm for 3D problems.

PARALLEL IMPLEMENTATION

Parallel calculation of the weight matrix

We have distributed the calculation of the weight matrix on a beowulf cluster using the parallel infrastructure described in Clapp (2005). The complete image and masks are distributed to each node. This is necessary because the random bounds requires random jumps around the image. Different jobs are assigned different rows of the weight matrix, no communication beween nodes is necessary while calculating the weight matrix. Upon completion, the weights are collected into a single sparse matrix on the master node

Parallel calculation of the eigenvector

To calculate the eigenvector with the second smallest eigenvalue, we use ARPACK Fortran77 software (Lehoucq and Scott, 1996) as recommended by Shi and Malik (2000) . This is a package of routines designed specifically for computing a few eigenvectors and eigenvalues for large sparse matricies.

The ARPACK interface requires the user to supply the subroutine that does the matrix-vector multiplication, The matrix-vector multiplication is the most expensive portion of calculating the eignvector, and therefore the obvious target for parallelzation. We implement the eignvector calculation in a modified master-slave scheme.

The slave nodes are intialized with a portion of the off-diagonal elements of the matrix. The master node is given a vector by the ARPACK library. It sends that vector to the first slave node and then begins to create the output vector by multiplying the diagonal terms of the matrix. Upon receiving the input vector the first slave node, it passes the vector the second slave node, and then begins multiplying the input vector by its portion of the off-diagonal terms creating its own output vector. This process is repeated by all of the slave nodes. The master node upon finishing multiplying the diagonal terms passes the output vector to the first slave node. It adds it to its output vector, and passes it to the second slave node. The process is repeated until the last slave node, which passes the completed matrix multiplication to the master node.

By implementing the matrix multiplication in this form, a good level of load balancing, and minimal communication wait time is acheived. We ran the approach on a Infinband network and were able to do 200 iteration of 2 billion non-zero matrix elements in 55 minutes.

Parallel Issues

Our current bottleneck is a software design issue. Much of SEPlib, assumes axes no larger than 2^{31} by using integers. If we exceed this number we get semi-random errors. Another potential problem is the way we implemented the eignvector calculation. If we scale to many nodes, the communication time will dominate, and we will significantly degraded performance. We have so far avoided this problem by running on a low-latency, high-bandwidth network.

FIELD TEST CASES

We tested this method on a Gulf of Mexico 3D data set.

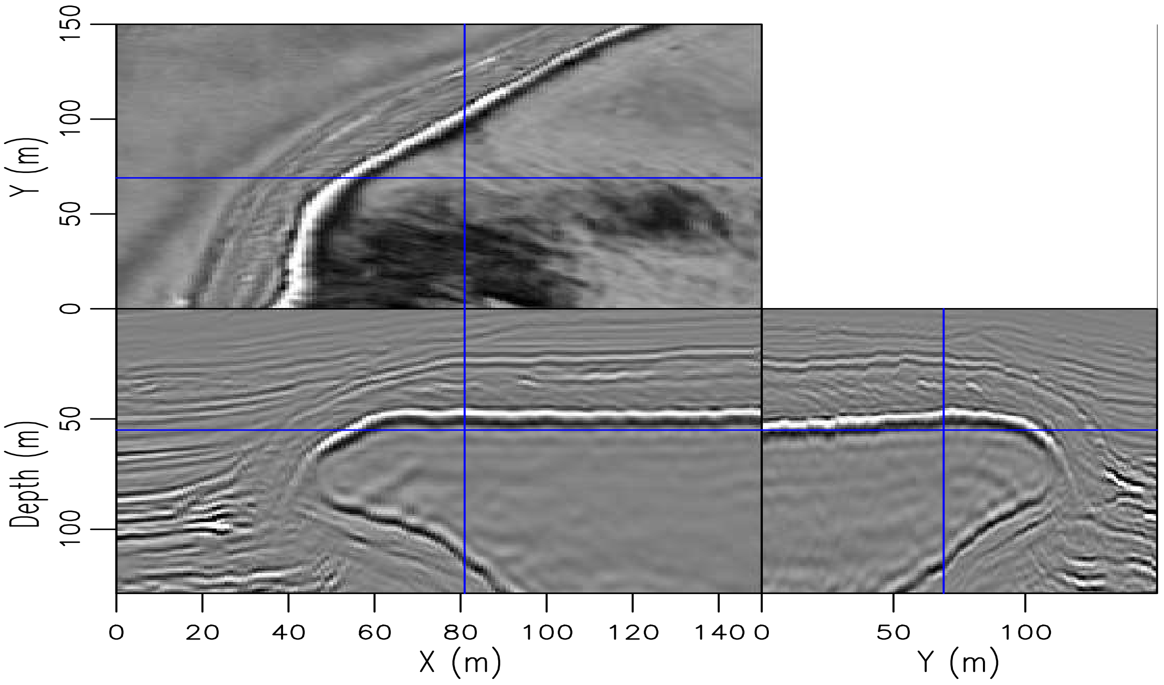

Figure 3: A Gulf of Mexico 3D data set with a prominent salt boundary. jesse1-gom3d.dat [ER]

In Figure 3 is a 3D seismic cube from the offshore with a prominent salt boundary. Figure 4 shows the mask used to define the bounds. In this case, we used the velocity model to define the upper and lower bounds. Alternatively, we could have garnered the bounds from a first pass on a sub-sampled cube.

The second smallest eigenvector from the method is displayed in Figure 5. The eigenvector is split along the salt boundary rather than along its shorted dimension as would have resulted without random bounds. Figure 6 displays the envelope of the data with the resulting salt boundary pick overlaying it. Notice it does a good job of tracking the peak of the amplitude. This illustrates that our method can be applied to 3D cubes. This is not surprising as the method has already been applied successfully to 3D atomic meshes (Hale and Emanuel, 2003).

Some errors can be seen in Figure 7 where the picks are now overlaying the seismic data. The errors occur in two places. One place is where the instantaneous amplitude of the salt boundary is weak and the other area is near the boundary. The weak amplitude error can also be seen in Figure 5, where the eigenvector is smoother. This illustrates the need for this method to utilize more than one attribute to delineate the salt boundary.

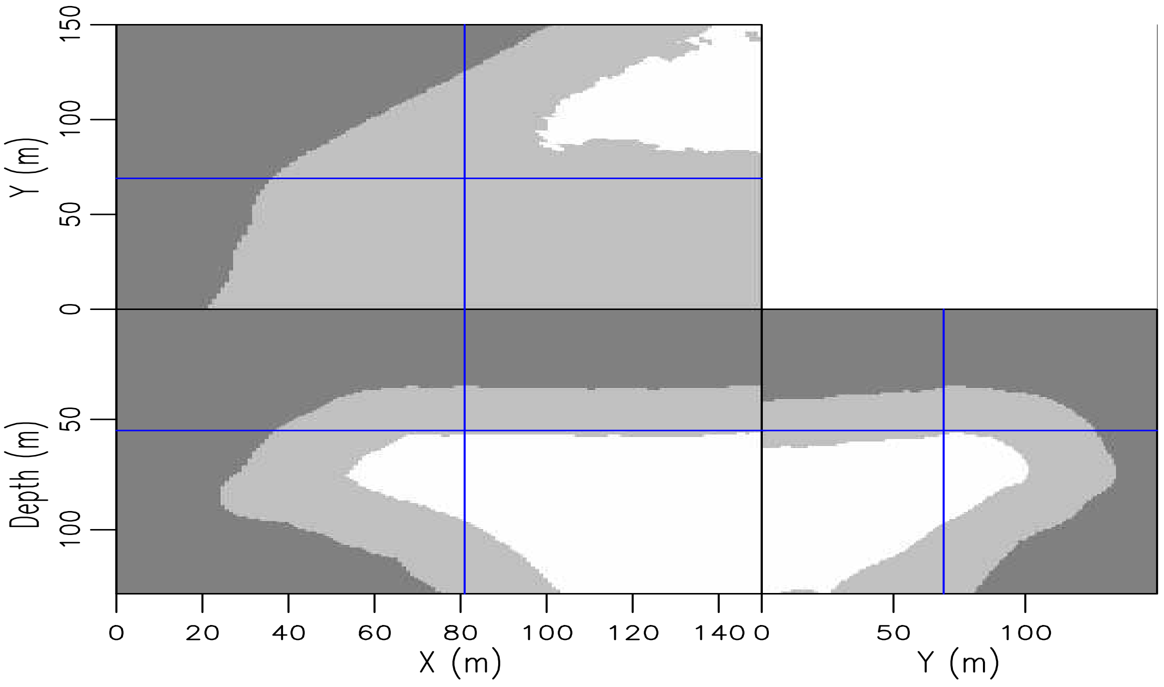

Figure 4: A mask used to define the upper and lower bounds of the salt interface. This was created from the velocity model. In cases, where a velocity model is not available, a similar mask can be created from segmenting a sub-sampled cube or from a rough manual interpretation. jesse1-gom3d.maskfig [ER]

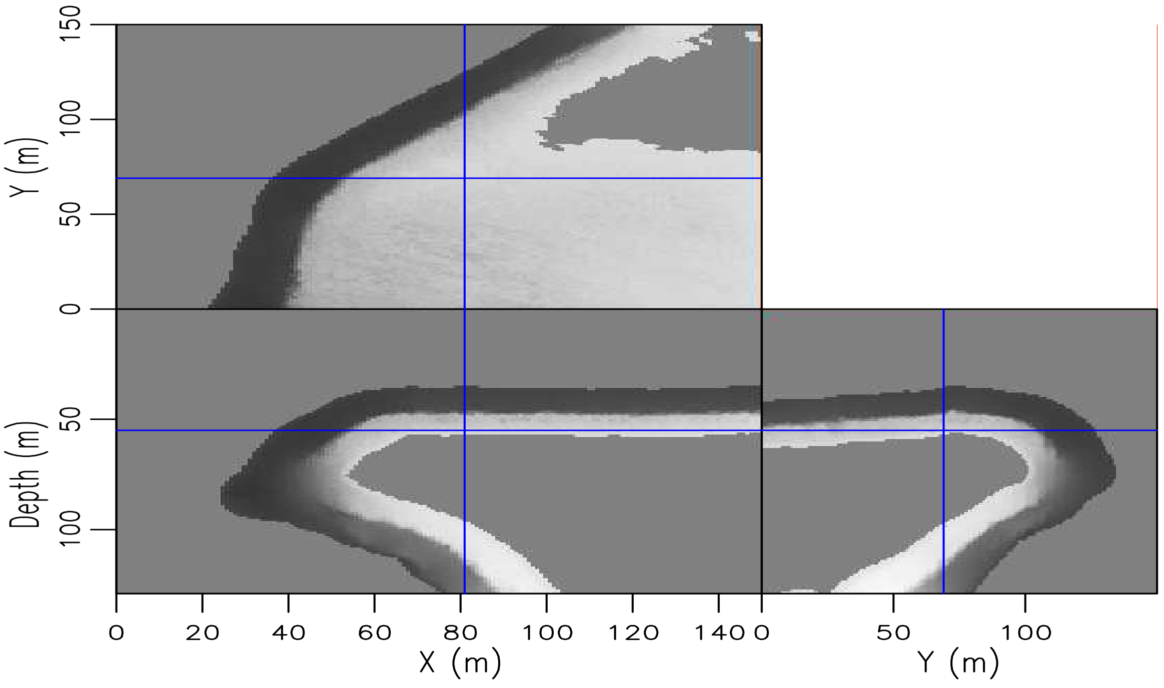

Figure 5: The second smallest eigenvector that is used to partition the image. Notice it is smooth where the amplitude does not delineate the boundary well. jesse1-gom3d.peig [CR]

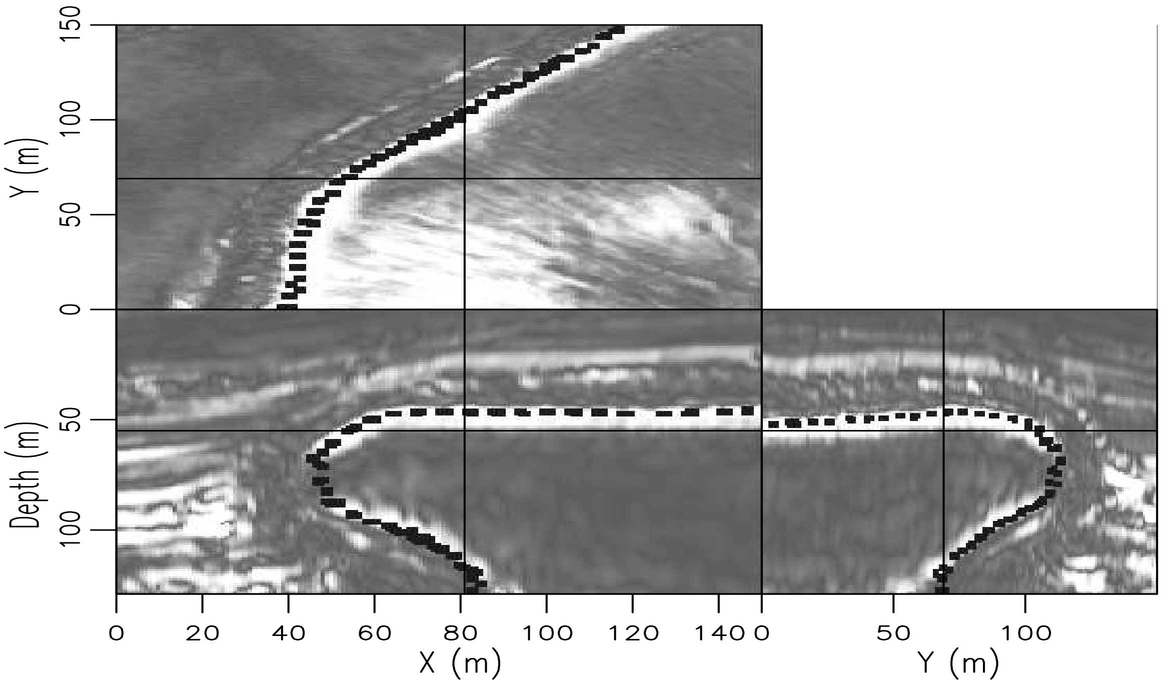

Figure 6: The boundary is extracted from the image in Figure 5 and overlain on the instantaneous amplitude of data. It accurately tracks the boundary. jesse1-gom3d.horizon_overlay1 [CR]

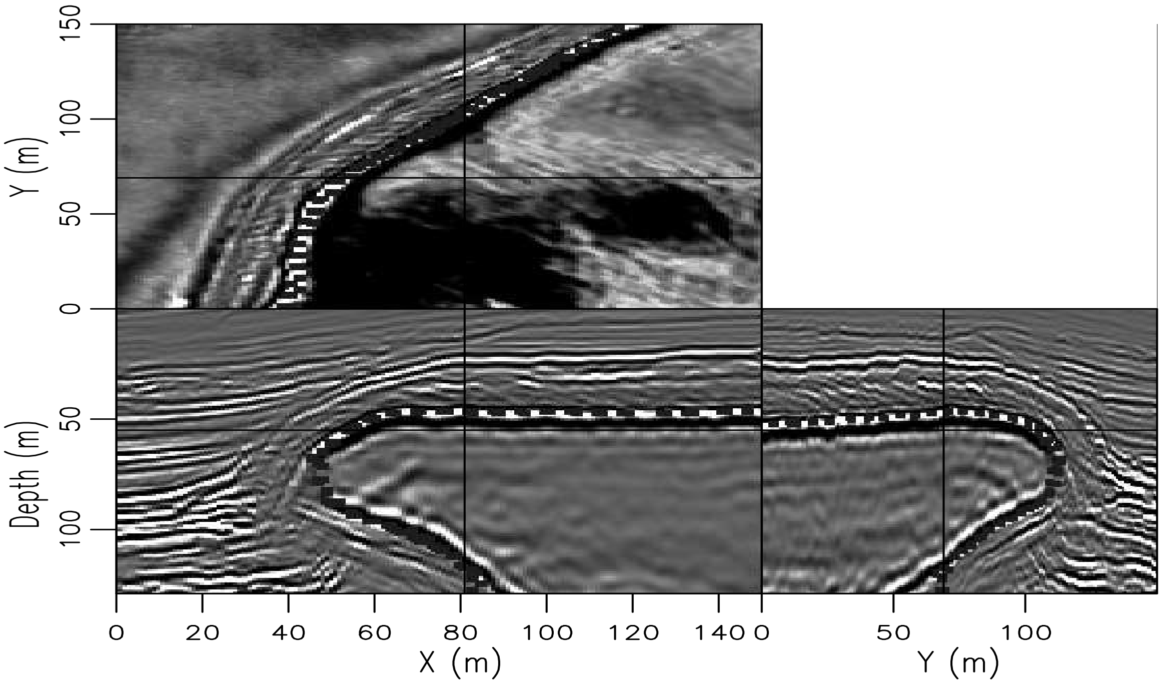

Figure 7: As Figure 6 except the boundary is overlain on the data itself. Notice it accurately tracks the boundary except where the amplitude is weak. It also has difficulty at the boundary of the image itself. jesse1-gom3d.horizon_overlay2 [CR]

CONCLUSIONS AND FUTURE WORK

We have successfully parallelized two key steps of this segmentation method: the calculation of the weigtht matrix and the estimation of the eigenvectors. Now, with a large enough cluster, almost any sized post-stack 3D data set can be globally segmented. This is an exciting development in that this algorithm is becoming significantly more practical.

In many places on salt boundaries the amplitude can become weak and the boundary can more easily be tracked using another attribute such as instantaneous frequency. On first glance it seems straight forward to estimate the weights of the normalized cuts image segmentation method using multiple attributes but to balance the weights in an optimal way may be somewhat challengeing. We hope to address this problem in the near future.

ACKNOWLEDGMENT

We would like to thank BP for allocating resources used to write this paper.

REFERENCES

Clapp, R. G., 2005, Inversion and fault tolerant parallelization using Python: SEP–**120**, 41–62.

Hale, D. and J. U. Emanuel, 2002, Atomic meshing of seismic images: Soc. of Expl. Geophys., Expanded Abstracts, 2126–2129.

Hale, D. and J. U. Emanuel, 2003, Seismic interpretation using global image segmentation: Soc. of Expl. Geophys., Expanded Abstracts, 2410–2413.

Lehoucq, R. B. and J. A. Scott, 1996, An evaluation of software for computing eigenvalues of sparse nonsymmetric matricies: Preprint MCS-P547-1195, Argonne National Laboratory, Argone III.

Lomask, J. and B. Biondi, 2003, Image segmentation for tracking salt boundaries: SEP–**114**, 193–200.

Lomask, J. and B. Biondi, 2005, Image segmentation with bounds: SEP–**120**, 179–186.

Lomask, J., B. Biondi, and J. Shragge, 2004, Improved image segmentation for tracking salt boundaries: SEP–**115**, 357–366.

Shi, J. and J. Malik, 2000, Normalized cuts and image segmentation: IEEE Trans on Pattern Analysis and Machine Intelligence, **22**, no. 8, 838–905.