Reverse Time Migration Algorithm

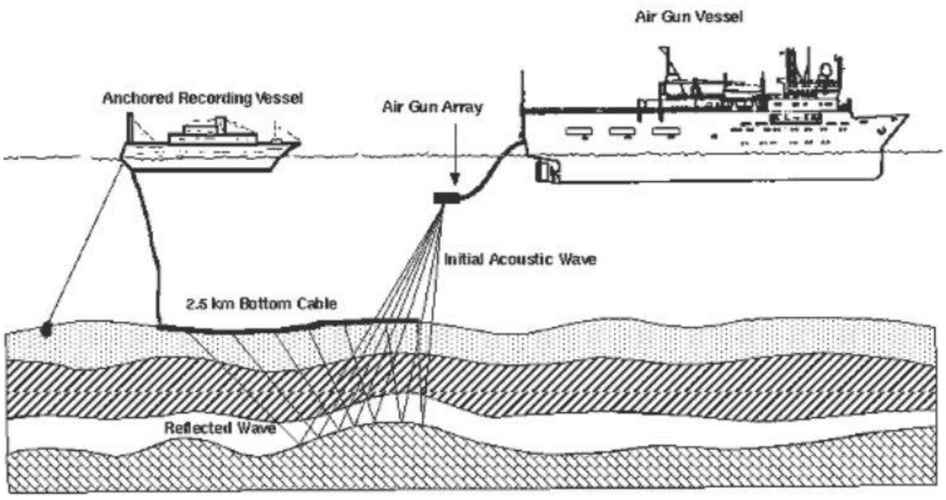
What is seismic exploration:

The purpose of the seismic exploration is to understand the constitution of the interior earth as much as possible. As we know, it is impractical or even impossible to penetrate the earth and put your camera there to capture the image, other indirect approaches are used to attain the similar or same result. These approaches include seismological measurement, electromagnetic measurement and gravity measurement. As these approaches are indirect, they usually give an analyze or indication of the measured result. Some tools, such as computers, are required as part of the analyze and the capability of the tools may be the bottleneck of the analyze. For example, in the past, with the poor computational performance of the computer, the data in only a small region and in a low resolution could the seismologists analyzed due to the limited computing power. As the Moore's Law indicates that the number of transistors double every 18 months, the computer gains much more computing power than before. Some seismological methods, which are computationally-demanding, seems not practical in the past, can be implemented by utilizing the advanced technology today. One of the most useful but computationally-demanding algorithm, **Reverse Time Migration algorithm**, also comes into reality. Reverse Time Migration is an algorithm of seismic migration. And seismic migration is one of the most important part of seismic exploration, and more specifically, seismic imaging. By seismic migration, the constitution of the interior earth could be imaged, sketching, for example, the water, rocks, gas, oil, faults etc in the subsurface.

General Process of Seismic Exploration:

Seismic exploration, as one of the background knowledge of this project which is explained in this section briefly. Figure 1 shows a typical scenario of a marine based seismic exploration. The vessel will inject waves periodically by the air gun. The waves just injected will spread in all direction quickly until it penetrate different media, such as rocks, ands, gas, or oil beneath the water, where the waves will reflect back or refract through another medium. The reflected waves, or those refracts first, then reflected waves would be recorded by a large array of Geo phones. In figure sea_floor_seismic, another smaller ship shows up, dragging a cable which contains an array of sensors to record the reflected signals. Geophysics data processing then follows the process of recording the signals. First of all, multiple steps should be applied to perform the preprocessing. For example, use the bandpass filter to remove the noise. And last of all, is the migration step to reconstruct the image of subsurface. The detail of how the data are processed is not the topic of this project, so I won't explain it.

Figure 1



Reverse Time Migration:

The **Reverse Time Migration** is a complicated algorithm, this section will provide a simple version of this algorithm. As the inputs the algorithm takes the results of the waves which come from the earth's surface. The waves have three dimension (X and Y) and time of propagation. Using some calculation which includes integral, the algorithm could be implemented by some for loops which has backward and forward part.

The algorithm for two dimensions (X-Y):

Forward propagation: send the waves

Backward propagation: resend the waves

Receiver propagation: get the waves

All the steps require integrals and derivations which can be implemented by using time as a time step in the for loops. The input is 3 dimensional arrays which is the point on the earth surface and the time propagation for it. I just implement the forward propagation as the other parts are somehow the same and use the same algorithm for parallelism. The algorithm contains 4 for loops.

Algorithm for the forward part:

NOTE: We need some weights in order to have more realistic results. The weights are from a real calculation which is calculated in geographical articles.

```
for ( num = 0 ; num < 5 ; num++ ){
for ( t = 5 ; t < NUM-5 ; t++ ){
for ( x = 0 ; x < NUM ; x++ ){
for ( y = 0 ; y < NUM ; y++ ){
output[t*(NUM)+x*NUM+y] = weight[0]*input[t*(NUM)+x*NUM+y]+ weight[1]*(input[(t-1)*NUM+x*NUM+y] + input[(t+1)*NUM+x*NUM+y])
+ weight[2]*(input[(t-2)*NUM+x*NUM+y] + input[(t+2)*NUM+x*NUM+y])+ weight[3]*(input[(t-3)*NUM+x*NUM+y] + input[(t+3)*NUM+x*NUM+y])
+ weight[4]*(input[(t-4)*NUM+x*NUM+y] + input[(t+4)*NUM+x*NUM+y])+weight[5]*(input[(t-5)]*NUM+x*NUM+y + input[(t+5)*NUM+x*NUM+y]);
}
}
}
}
}
```

I implemented the above algorithm by **<u>pthreads, MPI and OpenMP</u>** in order to use threads for doing the loops.

In my implementation the parallelism take place for the X loop, since the X loop has more iteration and according to the dependencies in the algorithm the order of loops can be changed so that X loop contain y loop and T loop.

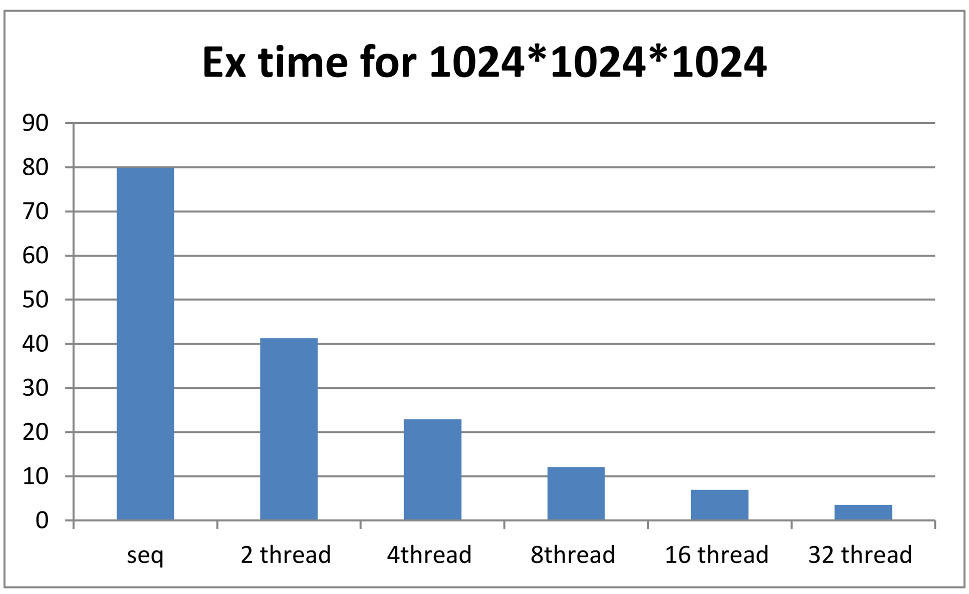
In pthread implementation, each thread has a chunk of X iterations the chunks are the number of surface points divided by number of threads. MPI implementation is the same, but each processor in MPI works on a chunk of X iterations.

The difference between pthreads and MPI is their barriers, though the results of them is expected to be close but not the same.

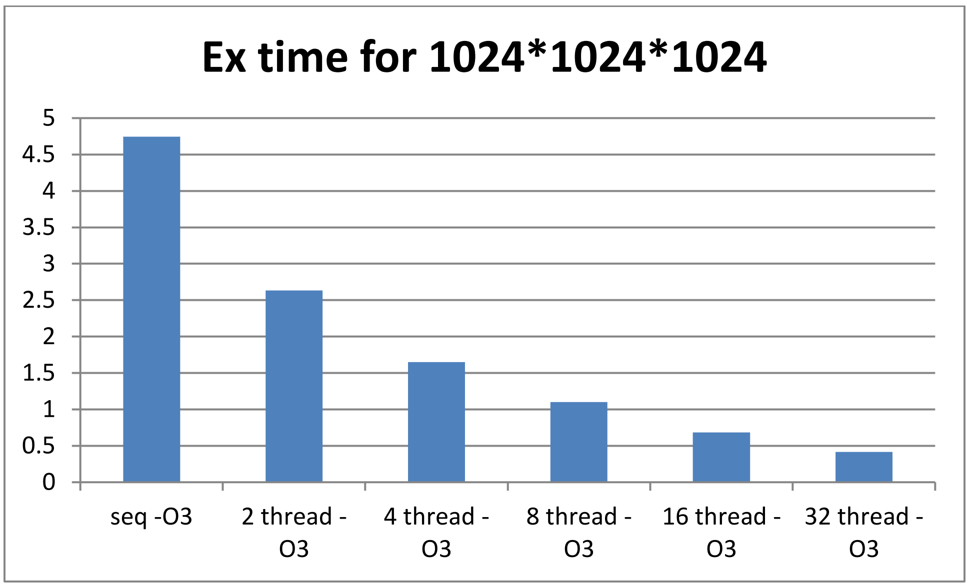
OpenMP is a way to program on shared memory devices and parallelism occurs where every parallel thread has access to all the data, whereas **MPI** is a way to program on distributed memory devices which parallelism occurs where every parallel process is working in its own memory space. Since my implementation is just a simulation and is not loading actual data from memory, these two algorithms will not have a major performance difference.

The results are calculated for 1024 inputs. Input is 1024*1024*1024 dimension. Also the Excel file is in uploaded files.

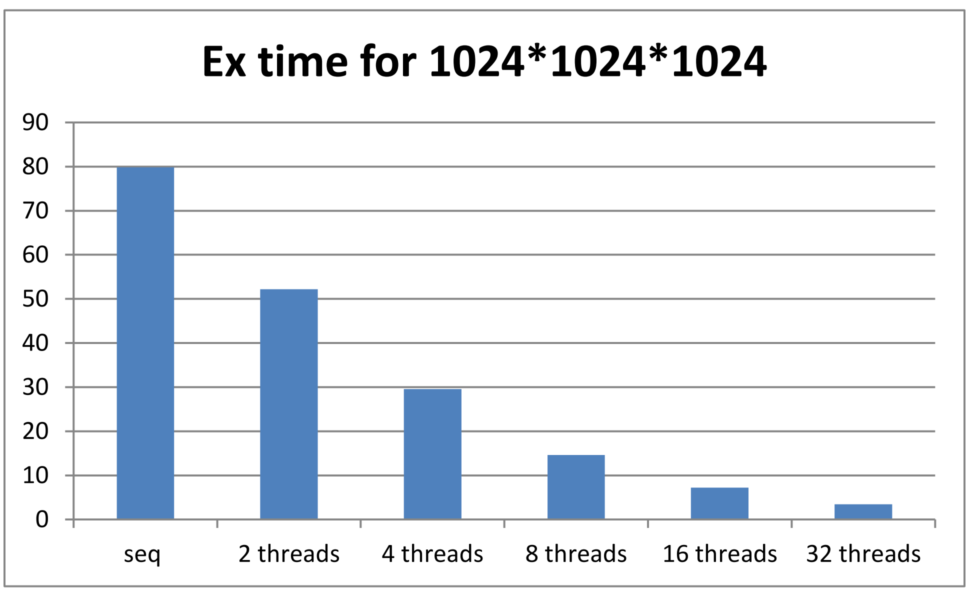
PTHREAD without any optimization tag:



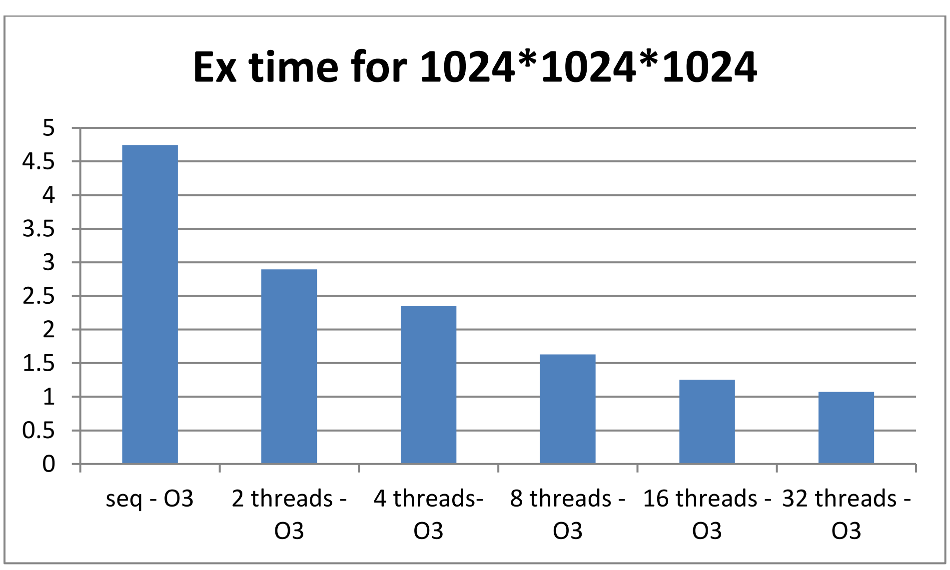
PTHRAED with O3 optimization:



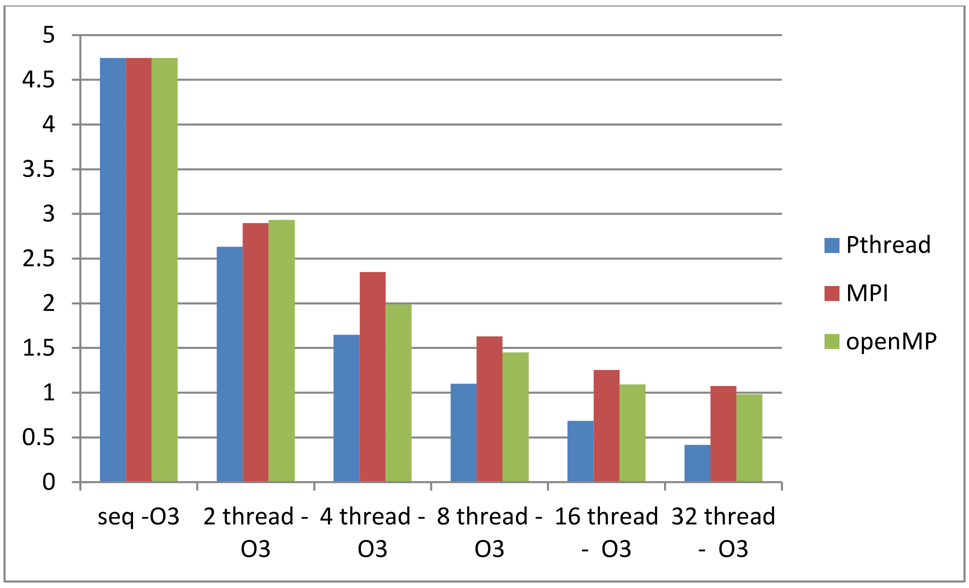
MPI without optimization flags:



MPI with O3 flag:



The results for all three methods in comparison:



5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
seq -O3
2 thread - O3
4 thread - O3
8 thread - O3
16 thread - O3
32 thread - O3
Pthread
MPI
openMP