1. Abstract
We have successfully ported discontinuous Galerkin (ADER-DG) method for solving the three-dimensional elastic seismic wave equation on unstructured tetrahedral meshes with Nvidia Tesla GPUs using the Nvidia CUDA programming model. On average our implementation obtained a speedup factor of about 24.3 for the single-precision version of our single GPU code and about 28.2 for the single-precision version of our multiple GPU code.

2. Motivation
- Seismologists need accurate and fast forward calculation result to solve inverse problems.
- Accurate simulation of elastic wave propagation in heterogeneous media with complicated geometry was difficult.
- New Discontinuous Galerkin method not only adapt unstructured mesh but also solve the elastic wave equations with high accuracy in both space and time.
- Implemented this Discontinuous Galerkin method on GPU system can greatly enhance the performance, which could make this method even more competitive.

3. CUDA implementation
- Our single GPU implementation obtained a speedup factor of about 24.3x compared with single CPU implementation. (Figure 1)
- This performance improvement is based on our own matrix-matrix multiplication approach, which 4.1x faster than dense matrix-matrix multiplication and 3.0x faster than the sparse matrix-matrix multiplication. (Figure 2)
- Generically, dense matrix-matrix multiplication has better memory access pattern, but can not handle sparse matrix efficiently. Meanwhile, sparse matrix-matrix multiplication can get rid of zero elements calculation, but the memory accesses pattern are normally uncoalesced.
- Our own matrix-matrix multiplication approach considering that the locations of the non-zero elements in the matrices can be determined beforehand and are fixed throughout the program, so the results of the matrix-matrix multiplications can be evaluated analytically beforehand.
- The expressions of the multiplication results can be hardwired into the CUDA kernels.
- This implementation eliminates all redundant calculations involving zero elements and by carefully arranging the order of the calculations in accordance with the thread layout, we can also minimize the number of uncoalesced memory accesses to the time-integrated DOF array.

4. Scalability
- Because we successfully hide those non-blocking MPI communication time with GPU computing kernels, our multiple GPUs code has satisfying weak scalability. (Figure 5)
- As for the strong scalability, we need to satisfy two requirements to achieve the best result. First, we need to feed the GPU global memory with enough elements. Take Tesla C2070 as example, the minimum memory load should be 3Gb. Secondly, we need to keep a very small outer/inner elements ratio. For Fermi series GPU, we suggest that the ratio should be smaller than 2%. (Figure 6)

5. Practical example
- We have built a 3D marmousi2 model with the dimension of 3500m in depth, 17000m in length and 1500m in width. There are 38554 tetrahedral element, and each element has its own material property. Our CUDA code spend 3052.57(s) calculate 5s seisogram, while the SeisSol need 96798.10(s), it's a speedup of 22.87x. (Figure 7)

6. Conclusion
We have investigated the implementation of Discontinuous Galerkin method on GPUs system. By developing our own matrix-vector multiplication scheme and combine GPU CUDA multiple stream and non-blocking MPI, our high performance GPUs code could provide another solution for accurate and efficient forward simulation.

Figure 1. The single GPU speedup factors for problems with 7 different mesh sizes. The number of tetrahedral elements used in our experiments are 3799, 6899, 12547, 15764, 21121, 24606 and 29335. The speedup factors were computed for our single-precision GPU code with respect to the CPU code running on 1/2/4/8 cores in different mesh sizes. The number of tetrahedral elements used in Figure 1 are 75000, 250000, 500000, 1000000.

Figure 2. Result comparison among dense matrix-vector multiplication scheme, sparse matrix-vector multiplication scheme and our own matrix-vector multiplication scheme. Our optimized matrix-vector multiplication approach obtained a speedup of 4.1x to dense matrix-vector scheme and 3.0x to sparse matrix-vector scheme.

Figure 3. The flowcharts of the major steps in the reference parallel CPU codes (left) and those in our CPU-GPU hybrid implementation (right). The whole calculation can be separated into 3 sections: The pre-processing section reads and calculates all the data that the time-stepping section will use. The time-stepping section updates the DOFs of each tetrahedral element which has been ported to the GPU. The post-processing section is in charge of writing out the DOFs and/or the seisograms at the pre-specified locations.

Figure 4. The multiple GPUs implementation’s speedup factors for problems with 2 different sizes. The number of tetrahedral elements used in our experiments are 327866, 935870. The speed factors were computed for our single-precision multiple GPU code with respect to the GPU code running on 8/16/32/64 cores on different nodes.

Figure 5. Weak scalability of our multiple GPU code performed by 2 – 8 GPUs, the black line shows the average wall time per 100 time steps for these size- varied problems. The average number of elements per GPU is around 53000 with about 6% fluctuation.

Figure 6. Strong scalability of our multiple GPU code with 1.92 million elements, the black line shows the average wall time per 100 time steps for this size-fixed problem performed by 32 to 64 GPUs.

Figure 7. The plot of marmousi2 P-wave velocity and the marmousi2 model shot gather plot, computed by our single GPU code. There are 341 receivers located at 5m beneath the surface and the horizontal interval is 50m, the explosive source located at 10.0m (depth), 8500.0m(length), 750.0m(width).