NEISP^2 Report

June 2013

Team Lead: Dr. Liqiang Wang (Computer Science) and Dr. Po Chen (Geophysics)

Other Personnel: He Huang (Ph.D. student in Computer Science) and Dawei Mu (Ph.D. student in Geophysics)

Affiliation: University of Wyoming

1. Summary

We designed and implemented two scalable algorithms for heterogeneous CPU-GPU supercomputer. One is a widely-used Sparse Equations and Least Squares (LSQR) solver, called SPLSQR, which is a numerical method for solving large-scale linear equation problems in an iterative way. The other is a community earthquake simulation software "cuDGSeis" for solving the 3D viscoelastic seismic wave equation using the discontinuous Galerkin method. Both are widely-used algorithms in geophysics.

In SPLSQR, we focus on optimizing the communication issues suffered by our previous implementation (PLSQR). We redesign a novel data decomposition strategy that treats kernel component and damping component of the coefficient matrix separately. Specially, we partition the kernel component matrix by column, partition the original damping component by row, and partition the transposed damping component by column. We design a low overhead approach to reorder the damping component such that the nonzeros are moved to diagonal area in order to minimize the potential communication after decomposition. These optimizations result in an algorithm with scalable communication volume between a fixed and modest number of communication neighbors. In addition to redesign the algorithm, we apply several other optimization techniques in our implementation: (1) use MPI-I/O to read and write data in a parallel and scalable way; (2) accelerate compute-intensive loops by multithreading using OpenMP; (3) port the entire iterative steps to GPU; (4) design a sophisticated load balance strategy that trades off between computation load and communication load; (5) propose a vector compression technique to reduce redundant communication volume. Compared with PLSQR, our SPLSQR reduces communication cost significantly, and enables scalability to O(10,000) cores. The fast SPLSQR algorithm enables geoscientists to find optimal damping coefficients in a reasonable time, which is almost impossible before.

In "cuDGSeis", we have successfully ported an arbitrary high-order discontinuous Galerkin (ADER-DG) method for solving the three-dimensional elastic seismic wave equation on unstructured tetrahedral meshes (i.e., the original "SeisSol" code

developed by the “SeisSol” Working Group at the Ludwig-Maximilians University in Germany) to a single Nvidia Tesla C2075 GPU using the Nvida CUDA programming framework and obtained a speedup factor of about 24.4 for the single-precision version of our GPU code and a speedup factor of about 12.8 for the double-precision version of our GPU code, as shown in Figure 3.

2. SPLSQR: A Scalable Parallel Algorithm of LSQR for Seismic Tomography

The row-wise partitioning based parallel implementations of LSQR such as PLSQR have the potential, depending on the non-zero structure of the matrix, to have significant communication cost. The communication cost can dramatically limit the scalability of the algorithm at large core counts.

We design a more scalable parallel LSQR algorithm that utilizes the particular nonzero structure of matrices that occurs in structural seismology. In particular, we specially treat the kernel component of the matrix, which is relative dense with a random structure, and the damping component, which is very sparse and highly structured separately. In algorithm level, our contribution includes:

- We store one copy of kernel submatrix and two copies of damping submatrix, i.e., one in original form and the other one in transpose form in memory. In particular, the kernel component is partitioned by column, the original damping component is partitioned by row, and the transposed damping component is partitioned by column. Vector x and y are partitioned accordingly.
- A low overhead reordering method is applied on damping component. The reordered damping component has multiple thin bands near the diagonal area. This feature is used later in data decomposition, where partitioned data in individual task only has very small overhead with nearby neighboring tasks. This significant reduces the algorithm's communication volume.
- A sophisticated parallel commutation and communication methods are designed based on data decomposition. Instead of having a large communication with all MPI tasks involved in PLSQR, SPLSQR involves a small size communication with all MPI tasks and local communicate with a small set of neighboring tasks with a number of small size messages. A message compression technique is designed to detect and remove the sparsity of sending or receiving message, which further reduces communication volume.

In addition, several implementation level optimization approaches are applied to accelerate I/O and computation performance at individual task.

- I/O is scalable by using MPI-I/O to read and write data in parallel. We store data in a single binary file. All MPI tasks opening the same file compute their

individual starting offsets and data lengths, and then call a collective read to finish loading data. Parallel writing is operated in the same way.

- A trade off load balance strategy is designed. The seismic data is not evenly distributed. So evenly partitioning row or column makes unbalanced computational load but with balanced communication load. However, evenly partitioning data by approximately equal number of nonzeros makes perfectly balanced computational load but with highly unbalanced communication load. Our strategy trade off computational load balance and communication load balance by setting appropriate ratios to achieve decent load balance.
- OpenMP is applied to parallelize compute intensive loops. Increasing tasks parallelism also introduces communication overhead. However, increasing threads within a compute node does not involve inter process communication. We use OpenMP threads to parallelize most compute intensive loops.
- MPI-CUDA version is also implemented for the new SPLSQR algorithm. The entire iterative steps are ported to GPU to accelerate computation. Inter GPU communication is implemented via MPI.

The resulting algorithm has a scalable communication volume with a bounded number of communication neighbors regardless of core count. We present scaling studies from a real seismic data set that illustrate good scalability up to O(10,000) cores on a Cray XT cluster. SPLSQR is significantly faster than PLSQR and PETSc. We demonstrate that on a modest seismic tomography dataset the SPLSQR algorithm is 9.9 times faster than the PETSc algorithm on 2,400 cores of a Cray XT5. The current implementation of the SPLSQR algorithm on 19,200 cores of a Cray XT5 is 33 times faster than the fastest PETSc configuration on the large size ANGF dataset. We plan to conduct more experiments on the Blue Waters supercomputer.

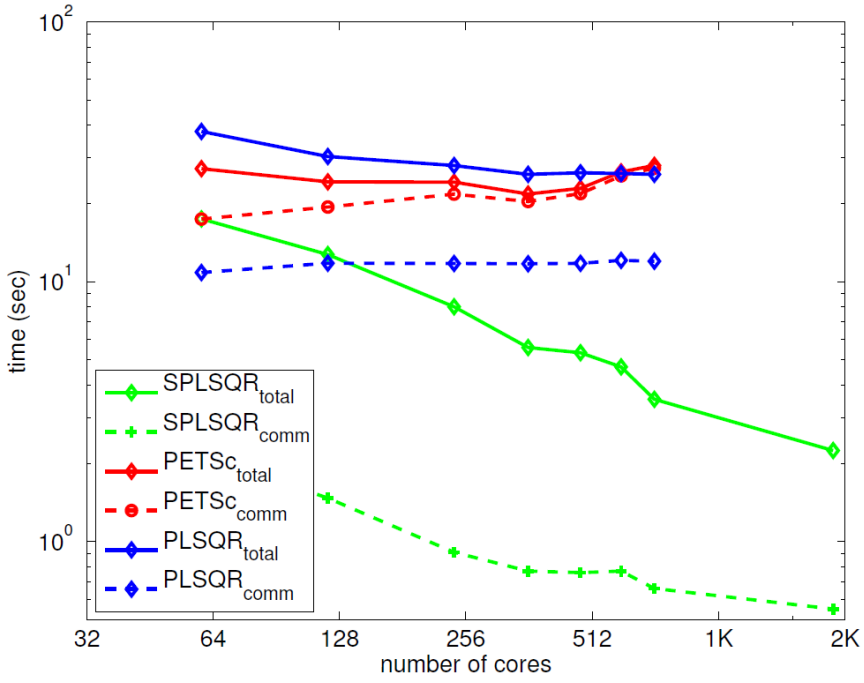

Figure 1. The total and communication time for 100 iterations of the SPLSQR, PLSQR, and PETSc implementations for the DEC3 data set from 60 to 1920 cores on the supercomputer of Kraken. The execution time of the SPLSQR algorithm is 1.7x less than PETSc at 60 cores, and is 7.8x less at 720 cores. The communication cost for SPLSQR is over 50x less than either PLSQR or PETSc.

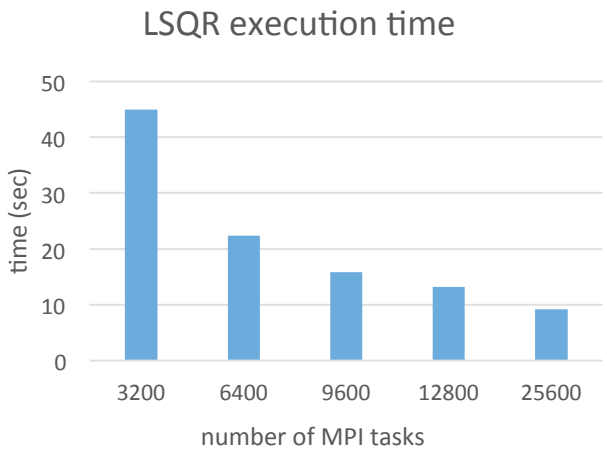

Figure 2 shows a preliminary result on EQANGF62K using BW XE nodes. It is scalable from 3200 cores to 25,600 cores. We plan to do more experiment using XE or XK nodes.

3. Accelerating Seismic Wave-Propagation Simulations Using GPUs (cuDGSeis)

Accurate and efficient computer simulations of seismic wave propagation in realistic three-dimensional geological media are becoming increasingly important in seismology for improving our understanding of the earthquake rupture process that generates seismic waves and the geological medium through which seismic waves propagate.

Within this project, we had extended our ADER-DG implementation with multiple GPUs capability. Our latest implementation, which code named “cuDGseis”, now can run on multiple nodes and each node can have multiple GPU accelerators.

To analyze the performance of the parallel version of our CUDA codes, we use a simplified version of the SEG/EAGE salt model as the benchmark. This model is geometrically complex. The ADER-DG method can accurately handle seismic wave propagation in water simply by setting the shear modulus of the elements in the water region to zero.

This salt model is discretized into tetrahedral meshes with different number of elements. In Figure 4, we show the speedup factors obtained for two different mesh sizes, one with 327886 elements and the other with 935870 elements. The simulations were run on 8, 16, 32 and 48 CPU cores. And the speedup factors were obtained by running the same simulations on the same number of GPUs. On average, the speedup factor for our parallel GPU codes is around 28, which is slightly higher than the speedup factor obtained in the single-GPU-single-CPU comparison. This may due to the fact that in the parallel CPU code the outer elements of a subdomain are not treated separately from the inner elements, which does not allow the parallel CPU code to overlap the computation on the inner elements with the communication of the time-integrated DOFs of the outer elements.

To investigate the strong scalability (i.e., the decrease in wall-time with increasing GPU number while holding the total workload, that is the number of elements and time steps, constant), we discretized the salt model using a mesh with about 1.92 million elements and ran the simulation for 100 times steps. The number of GPUs used in the simulations ranges from 32 to 64. As seen on Figure 5, the strong scaling of our parallel GPU codes is close to the ideal case with some fluctuations. Our codes start to slightly underperform the ideal case when the number of GPUs used in the simulation is larger than 48. To effectively overlap computation with communication, the ratio between the number of outer elements and the number of inner elements of a subdomain cannot exceed a certain threshold, which is determined by the processing capability of the GPU and the speed of the inter-connections. In our case, when the number of GPUs used in the simulation starts to exceed 48, this ratio becomes larger than 2%, which we believe is the threshold for our hardware configuration. The performance of our parallel GPU codes depends upon a number of factors, such as load balancing, but we think the extra

communication overhead that was not effectively hidden by the computation was the dominant factor for causing our codes to underperform the ideal case. In Figure 6, we show the results of our weak scaling test (i.e., the workload per GPU is kept about constant while increasing the number of GPUs). By definition, the total number of elements in the weak scaling test increases approximately linearly with the number of GPUs. If the communication cost is effectively overlapped by computation, the weak scaling test should be approximately flat. In our tests, the average number of elements per GPU was kept around 53000 with about 6% fluctuation across different simulations. The ratio between the number of outer and inner elements was kept around 1%. The weak scaling is approximately flat, with some fluctuation mostly caused by the variation in the number elements per GPU used in each simulation.

We must point out that out code does run slower on Bluewaters supercomputer. The Kepler K20x GPUs' performance is about 20%-25% lower than the Fermi m2075 and very close to the performance of Fermi C2050. Based on our understanding and suggestion given by engineer from NVidia, this difference may be caused by the different architectures of Fermi and Kepler GPUs. Compared with Fermi GPUs, the Kepler GPUs doubled the peak float point performance by increased the CUDA cores from 448 to 2688, expand the GPU memory bandwidth from 150 GB/s to 250 GB/s, however, the Kepler GPU's clock rate had dropped from 1.15GHz to 706 MHz. Thus, for the same performance, the Kepler GPU tends to have higher occupancy rate than Fermi GPU. In our code, all the optimization were designed and tested on the Fermi GPU, and the occupancy rate on Kepler GPU is limited by the resource (register and shared memory) within each SM, which caused the compute power of Kepler GPU is not fully utilized. In order to improve the performance of our code on Bluewaters, we need to further optimize our code aim at the features of Kepler GPUs.

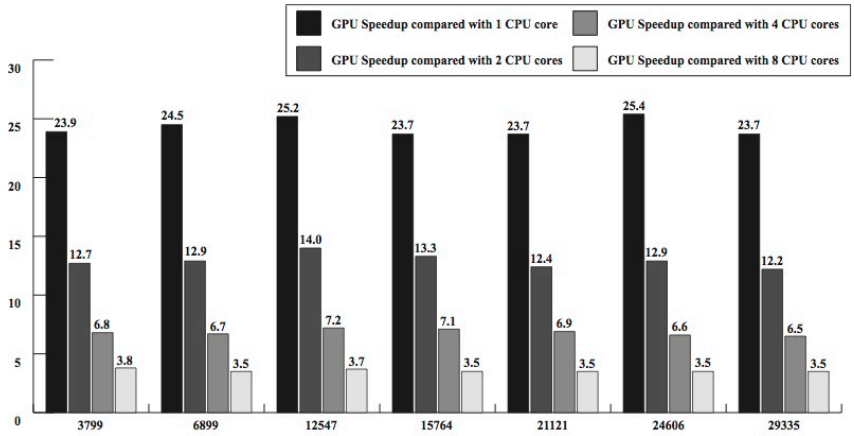

Figure 3. Single-GPU speedup factors obtained using 7 different meshes and 4 different CPU core numbers. The total number of tetrahedral elements in the 7 meshes is 3799, 6899, 12547, 15764, 21121, 24606 and 29335, respectively. The speedup factors were obtained by running the same calculation using our CPU-GPU hybrid code with 1 GPU and using the serial/parallel "SeisSol" CPU code on 1/2/4/8 CPU cores on the same compute node. The black columns represent the speedup of the CPU-GPU hybrid code relative to 1 CPU core, the dark grey columns represent the speedup relative to 2 CPU cores, the light grey column represents the speedup relative to 4 CPU cores and the lightest grey columns represent the speedup relative to 8 CPU cores.

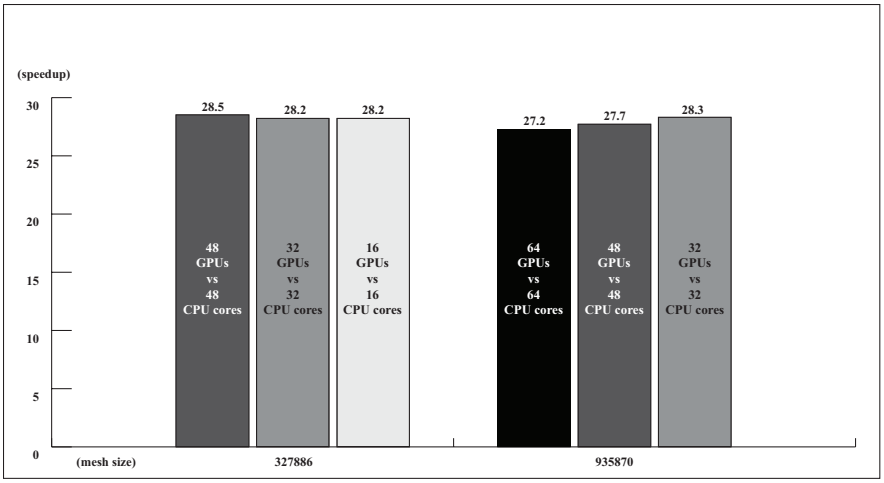

Figure 4. Speedup factors of our parallel GPU codes obtained using 2 different mesh sizes. The number of tetrahedral elements used in our experiments are 327866, 935870. The speed factors were computed for our single-precision multiple GPUs code with respect to the CPU code running on 16/32/48/64 cores runs on different nodes.

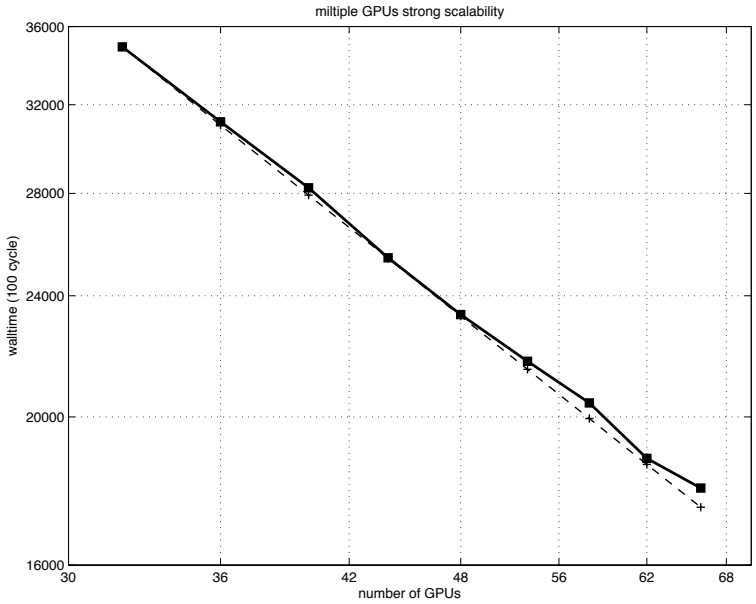

Figure 5. Strong scalability of our multiple GPUs codes with 1.92 million elements, the black line shows the average wall time per 100 time steps for this size-fixed problem performed by 32 to 64 GPUs.

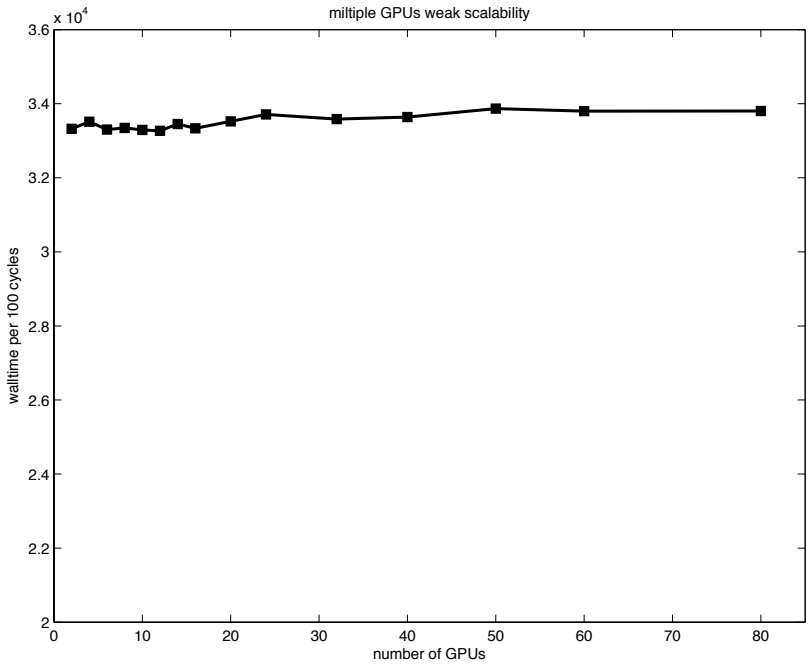

Figure 6. Weak scalability of our multiple GPUs code performed by 2 – 80 GPUs, the black line shows the average wall time per 100 time steps for these size-varied problems. The average number of elements per GPU is around 53000 with about 6% fluctuation.