Accelerating Laue Depth Reconstruction Algorithm With CUDA

Ke Yue, Schwarz Nicholas, and Tischler Jonathan Z.
Department of APS
Argonne National Lab
Argonne, Illinois 60439
Email: {kyue, nschwarz, tischler}@aps.anl.gov

Abstract—The Laue diffraction microscopy [1] experiment uses the polychromatic Laue micro-diffraction technique to examine the structure of materials with sub-micron spatial resolution in all three dimensions. During this experiment, local crystallographic orientations, orientation gradients and strains are measured as properties which will be recorded in HDF5 image format [2]. The recorded images will be processed with a depth reconstruction algorithm for future data analysis. But the current depth reconstruction algorithm consumes considerable processing time and might take up to 2 weeks for reconstructing data collected from one single experiment. To improve the depth reconstruction computation speed, we propose a scalable GPU program solution on the depth reconstruction problem in this paper. The test result shows that the running time would be 10 to 20 times faster than the prior CPU design for various size of input data.

I. INTRODUCTION

The Laue diffraction microscopy examine the structure of materials with sub-micron spatial resolution in all three dimensions. The materials which are investigated include inter-granular and intra-granular orientation distributions in polycrystals, elastic strain tensors in elastically deformed materials, and plastic deformation microstructures under microindents in Cu single crystals. This structural microscopy techniques is very powerful for detailed investigation of the microstructure and evolution in materials, especially local crystallographic orientations, orientation gradients and strains [1] [3]. The data set collected from the structural microscopy technique used in sector 34ID at Advanced Photon Source of Argonne National Laboratory is recorded in HDF5 format and processed by the depth reconstruction program running on CPU. This poster proposes a new CUDA implementation for depth reconstruction algorithm of HDF5 images.

II. CUDA SOLUTION FOR RECONSTRUCTION PROGRAM

To speed up the total time for reconstructing the image, we proposed a cuda program design to handle this problem. For designing an efficient cuda program, several challenges need to be considered in advance. The challenges include designing a data structure which could minimize communication time spent on transferring data and minimizing the computation time for each kernel thread, and utilizing the limited video memory for GPU.

For handling the first challenge, the special programming characteristic of cuda program needs to be taken into consideration. The speciality of cuda programming characteristic is that it will launch multiple kernel thread on GPU side at same time. Each kernel thread is usually doing computation on each data element in the dataset. The data input is p 2D images and each image has m rows and n columns pixels. Each pixel in the image must be mapped to a corresponding kernel thread. In order to map each kernel thread to a corresponding pixels in the image, we could either dynamically making a 3D array or a 1D array. Then map the pixel’s subscription in the 3D or 1D array to kernel thread’s id (x,y,z).

To choose the more efficient data structure from 3D array or 1D array, the challenge of minimizing communication time between CPU and GPU and the computation time spent on both side needs to be considered. In terms of communication and computation time, these two data structures can incur big performance difference, under the assumption that all the arrays are created dynamically. From the characteristic of CUDA program design, the communication time is usually spent on transferring data between CPU and GPU. The computation time is spent on the depth reconstruction of each data element and index mapping which is used to map each data index to each thread index.

The first method is to dynamically create a 3D array, with a 1D array of pointers pointing to a 2D arrays. The advantage of this method is that the pixel can be accessed directly based on the array subscript (x,y,z). The disadvantage is that extra pointers need to be passed from CPU side to GPU side which incurs extra transferring time. The second method is to dynamically create a 1D array, with just one pointer pointing to the first pixels of the array. The advantage of this methods is that no extra pointers are created and passed to CPU, which saves communication time. However, the disadvantage part is that the array index needs to be changed back and forth from 3D index. Extra computation time is incurred on both the CPU side and GPU side.

The difference between these two methods is that the first one needs more communication time to copy extra array pointers from CPU to GPU, while the second method needs more computation time for changing the index back and forth between 3D index and 1D index. Communication time between CPU and GPU is usually a threshold for CUDA program. Experiment is done on 5G data set for two designs and the result is shown in Fig. 1. It could seen from the result that 1D array design performs better than 3D array. Thus, we
implement the program using the first method for a better performance.

After determining the data structures, the limited memory on the video card needs to take into consideration. Since most GPU has limited memory and might not be able to handle all the data at one time. For example, the video card Tesla M2070 we used on our machine has maximum 6G memory. Other than the input data, the temporary data structures generated during the program has also to be count in. Thus, we can only divide the data input into several pieces and pass one piece each time to GPU.

In order to make the GPU to process the data, cudaMemcpypc is used to copy the data structures designed from CPU to GPU. And after the GPU processing, the result is copied back to CPU side via cudaMemcpy again. The parameter cudaMemcpypcHostToDevice and cudaMemcpypcDeviceToHost is used to specify which direction the data should be copied to. Other than the computation part, the rest program, such as reading data from HDF5 files and writing result back to text files are still running on CPU. The start of the CUDA kernel function for computation is setTwo() function. After passing the data from CPU to GPU, the kernel function started to do depth reconstruction on the data input. The detailed program flow for each kernel function is illustrate in Fig. 2.

III. EVALUATION

In this section, experiments are performed on one node of the cluster to compare the performance for four different data sets and GPU model is Nvidia Tesla M2070. To compare the performance between CPU and GPU code, we run two experiments. The first is to change data set from small size to big size. As showed in the Fig. 3a, we have four data sets with size 2.1G, 2.7G, 3.7G, 5.2G. The total memory used for CPU and GPU code are both 4G. The final running time for CUDA design is 10 to 20 times faster than the original CPU version. The GPU design of the image depth reconstruction outperforms CPU version in terms of performance. The second experiment is to change the percentage of the pixels in the data set. We sort all the pixel values and only compute certain percentage of all the pixels. We change the pixel percentage to 25%, 50%, 100%, and run two programs on the data set. The result is shown in Fig. 3b and we can conclude that the more pixels we handle, the better performance we can get. When the pixel percentage is increasing, the more data is also transferred to GPU side which incurs more data communication time. While the time saved on computation still makes the total running time for GPU code less than CPU code.

Furthermore when the data set increases from 2.1G to 5.2G as showed in Fig. 3a, we could see from the figure that the total running time for GPU did not change as much as CPU version. The conclusion can be got from the figure that our CUDA design did not just outperforms the CPU design in terms of performance, but also in scalability.

IV. CONCLUSION

In this poster, we propose a GPU design for the image depth reconstruction problem using CUDA. The test result shows that GPU design runs 10 to 20 times faster than the prior CPU design and thus gains a great performance improvement.

REFERENCES

[1] B. Larson, W. Yang, G. Ice, J. Budai, and J. Tischler, “Three-dimensional x-ray structural microscopy with submicrometre resolution,” Nature, vol. 415, no. 6874, pp. 887–890, 2002.
[2] M. Folk, A. Cheng, and K. Yates, “Hdf5: A file format and i/o library for high performance computing applications,” in Proceedings of Supercomputing, vol. 99, 1999.
[3] W. Liu, P. Zschack, J. Tischler, G. Ice, and B. Larson, “X-ray laue diffraction microscopy in 3d at the advanced photon source,” in THE 10TH INTERNATIONAL CONFERENCE ON X-RAY MICROSCOPY, vol. 1365, no. 1. AIP Publishing, 2011, pp. 108–111.