NUMERICAL COMPUTATION ON GPU

Zdeněk Konfršt

Faculty of Information Technology, Czech Technical University

Abstract

The accelerated computing on GPUs has been underway for more than 10 years. GPUs evolved in computer gaming industry and penetrated quickly into high performance computing due to the evident computing acceleration. This technical report is devoted to the description of several experiments that were accomplished with Nvidia's GPU Tesla C2050 and GPU supported applications.

1 Background

A graphics processing unit (GPU)[1] is a specialized circuit designed to rapidly manipulate and alter memory in such a way so as to accelerate the building of images in a frame buffer intended for output to a display. Modern GPUs are very efficient at manipulating computer graphics, and their highly parallel structure makes them more effective than general-purpose CPUs for algorithms where processing of large blocks of data is done in parallel. The term GPU was popularized by Nvidia[2] in 1999, that marketed the GeForce 256 as "the world's first Graphics Processing Unit", a single-chip processor with integrated transform, that is capable of processing a minimum of 10 million polygons per second.

These days GPUs are used in personal computers, workstations, and game consoles. GPU provides many advantages in demanding numerical computation. Matlab has been designed for numerical [1, 2] as well as technical [3] computation and directly supports Nvidia's GPU. Matlab supports Nvidia CUDA-enabled GPUs with compute capability version 1.x or higher, such as Tesla 10-series and 20-series GPUs.

2 Methods

Matlab supports the GPU computation within Parallel Computing Toolbox, which is a dedicated to parallel computation and supports GPU in Matlab R2010a. There are several demos, but the demos do not run on any Nvidia graphic card. The GPU has to support the compute capability 1.3 and higher. Therefore there was preliminary failure to run the demos on the Lenovo notebook equipped with Nvidia GeForce 9300M GS. Due to the previous reason, a new CPU-GPU workstation was built. Parameters of the workstation were CPU Intel Core2 6300 1.86GHz processor, 2GB RAM, Nvidia Tesla C2050, CUDA Driver v4.0.1 and OS Linux CentOS (v2.6.18-238.12.1.el5).

Several GPU experiments included: (i) Data manipulation on Nvidia Tesla GPU, (ii) GPU-accelerated Matlab operations and (iii) CPU/GPU benchmarking. Implemented GPU features in Matlab were investigated and challenged with other competitive software products and solutions. These were mainly Jacket/libJacket from AccelerEyes[3]. The first one installs as a plug-in into Matlab and the second one is a stand alone library to be used with high-level languages such as C/C++ or Fortran. Both libraries are to accelerate computation on a GPU.

3 Results

First, series of experiments were tests if the CUDA had been correctly installed. There are many examples in C/C++/OpenGL at the CUDA SDK to confirm the correct installation. Second, we

[1] www.wikipedia.org

[2] www.nvidia.com

[3] Sprinx Systems has been a re-seller of Tesla GPUs, CPU-GPU workstations as well as Jacket software.

have experimented with all GPU applications which are a part of Parallel Computing Toolbox in Matlab such as `paralleldemo_gpu_fft`, `paralleldemo_gpu_fft2`, `paralleldemo_gpu_arrayfun`, `paralleldemo_gpu_backslash` and `paralleldemo_gpu_devices`. These `paralleldemo_gpu_fft`, and `paralleldemo_gpu_fft2` are applications of a Fast Fourier Transform (FFT) on a GPU. The first one uses FFT to find the frequency components of a buried in a noisy time-domain. The second one performs a two-dimensional FFT to calculate far-field diffraction patterns. These diffraction patterns are usually observed when a monochromatic light passes through a small aperture. That is an example of Young's double-slit experiment.

The most of the time we spent with `paralleldemo_gpu_backslash`. This demo is a benchmark that solves a linear system on the GPU ($x = b/\mathbf{A}$). The program calculated matrices[4] from the size of 32^2 to $8,192^2$. We also experimented with the maxMemory parameter. The parameter allocates the amount of the system memory in GB available to the CPU and the GPU. The values of the parameter were set from 0.125 to 1.95. When the parameter increased over 0.125, matrices up to $8,192^2$ were calculated. Otherwise, it was only to the the $4,096^2$ sizes. The final graphs depicted speedups S_p of CPU-GPU over CPU. The `paralleldemo_gpu_devices` demo showed us CUDA description, parametrization and configuration of the CPU-GPU workstation. Third, we also experimented with Jacket and libJacket on the workstation. Due to time constrains, we did not reach enough clear, sufficient and convincing results to be reported. Anyway, the both libraries offer acceleration in the computation on a GPU.

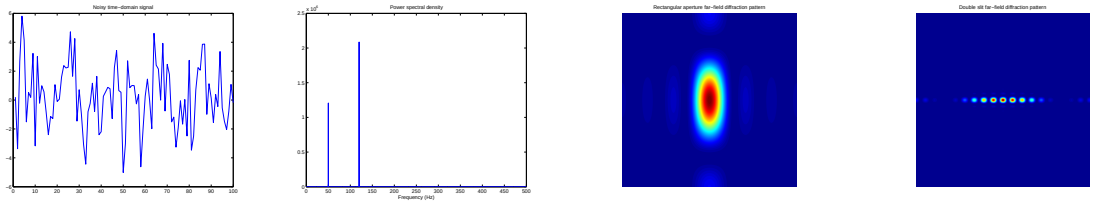

Figure 1: FFT. FFT finds the frequency components of a buried in a noisy time-domain done by `paralleldemo_gpu_fft` (left). The `paralleldemo_gpu_fft2` application performs a two-dimensional FFT to calculate far-field diffraction patterns (right).

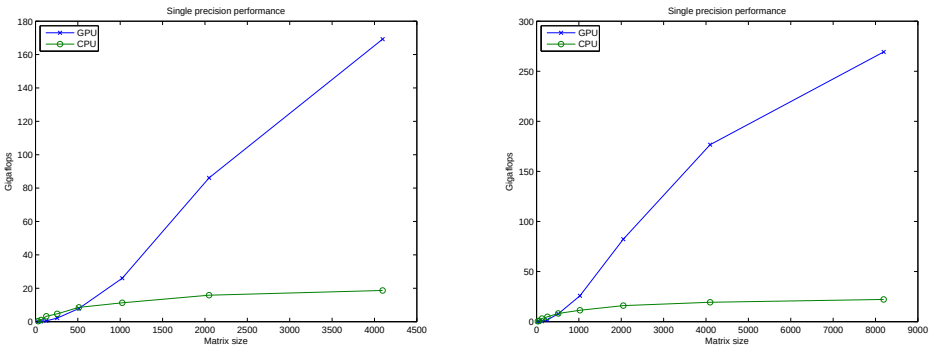

Figure 2: Single precision. The figures depict the functional dependency of computing power (Gigaflops) on the size of a matrix up to $4,096^2$ (left) and up to $8,192^2$ (right). These are results from the single precision computation of `parallel_gpu_backslash`.

4 Conclusion

This was our first serious experimental attempt with GPUs. We built the CPU-GPU workstation to support our experiments. Then, we run several tests in C/C++ and OpenGL to confirm the

[4]The size of a square matrix $\mathbf{A}_{m,m}$ is simply denoted as m^2 in the report.

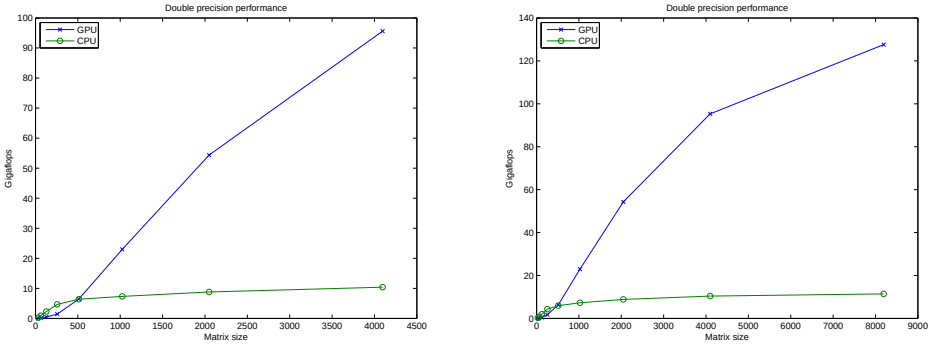

Figure 3: Double precision. The figures depict the functional dependency of computing power (Gigaflops) on the size of a matrix up to $4,096^2$ (left) and up to $8,192^2$ (right). These are results from the double precision computation of `parallel_gpu_backslash`.

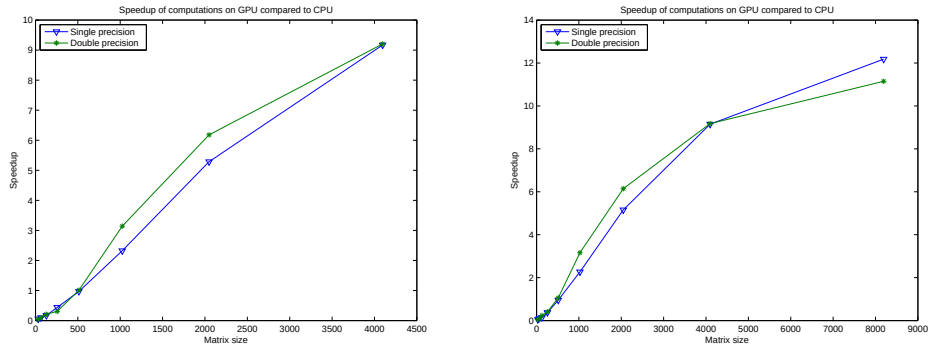

Figure 4: Speedup comparisons on precisions. The figures depict the functional dependency of Speedup S_p on the size of a matrix up to $4,096^2$ (left) and up to $8,192^2$ (right). These are results from the single and double precision computations of `parallel_gpu_backslash`.

CUDA installation and the capability of Tesla GPU. Later, we focused on GPU applications in Matlab and experiments with Jacket/libJacket.

Acknowledgements
The technical report was supported by Sprinx Systems a.s.

References

[1] R. Westermann J. Kruger. Linear algebra operators for GPU implementation of numerical algorithms. *Int. Conf. on Computer Graphics and Interactive Techniques*, 2005.

[2] M. Baboulin S. Tomova, J. Dongarra. Towards dense linear algebra for hybrid GPU accelerated manycore systems. *Parallel Computing 36 (5-6)*, pages 232–240, June 2010.

[3] J. Owens Y. Zhang. A quantitative performance analysis model for GPU architectures. *Int. Symp. on High Performance Computer Architecture*, pages 382–393, February 2011.

Ing. Zdeněk Konfršt, Ph.D.:
Faculty of Information Technology, Czech Technical University, E-mail: konfrst@fit.cvut.cz