Accelerating MATLAB with CUDA

Massimiliano Fatica (mfatica@nvidia.com), NVIDIA
Won-Ki Jeong (wkjeong@cs.utah.edu), University of Utah

Introduction

MATLAB®‡ is a powerful tool for prototyping and analysis. MATLAB could be easily extended via MEX files to take advantage of the computational power offered by the latest NVIDIA graphics processor unit (GPU). The graphic processor can be considered as a compute device that is capable of efficiently executing data parallel computations, and using CUDA, (Compute Unified Device Architecture [1]) can be programmed using a C-like language. MEX-files are a way to call custom C or FORTRAN code directly from MATLAB as if they were MATLAB built-in functions.

This presentation will show the feasibility and benefits of using this approach: with the achieved speed-up, there is no need to recode the application in C or FORTRAN.

CUDA

CUDA technology gives computationally intensive applications access to the tremendous processing power of the latest GPUs through a C-like programming interface. The GeForce 8 series GPUs have up to 128 processors running at 1.5 GHz and up to 1.5GB of on-board memory. CUDA is designed from the ground-up for efficient general purpose computation on GPUs. It uses a C-like programming language and does not require remapping algorithms to graphics concepts.

CUDA exposes several hardware features that are not available via the graphics API. The most significant of these is shared memory, which is a small (currently 16KB per multiprocessor) area of on-chip memory which can be accessed in parallel by blocks of threads. This allows caching of frequently used data and can provide large speedups over using textures to access data. Combined with a thread synchronization primitive, this allows cooperative parallel processing of on-chip data, greatly reducing the expensive off-chip bandwidth requirements of many parallel algorithms. This benefits a number of common applications such as linear algebra, Fast Fourier Transforms, and image processing filters.

The current generation of GPU from NVIDIA has support for IEEE single precision. Double precision support will be available in the next generation towards the end of the year. There are however several fields in which significant results can be obtained with single precision: image and signal processing and some numerical methods like pseudo-spectral approximation are just few examples.

‡ MATLAB is a registered trademark of The MathWorks, Inc.

An application to vortex dynamics.

The examples used in this abstract are from a class on atmospheric research at the University of Washington [2]. The MATLAB scripts solve the Euler equation in vorticity-stream function using a pseudo-spectral method. Pseudo-spectral methods are very well conditioned and some operations can be safely performed in single precision without affecting the overall quality of the solution.

The MATLAB code can be easily modified to solve problems with different initial conditions or forcing: for example to study the evolution of an elliptic vortex or 2D isotropic turbulence. The code is heavily FFT based. The system used in the benchmark has an Opteron 250 processor (running at 2.4Ghz) and a Quadro FX5600 GPU. All the results were obtained with MATLAB R2006B, both under Windows and Linux. In order to interface CUDA and MATLAB, we had to slightly modify the MEX infrastructure. CUDA files have a .cu suffix and needs to be compiled with a specific compiler (nvcc). Once we had a working infrastructure under Windows and Linux, the coding of the MEX functions was very simple. The non-linear term function, for example, was ported to CUDA in less than an hour.

We performed the work in two steps: the first one was to write MEX files for theFFT2 and IFFT2 functions in MATLAB calling the CUFFT library. The second one was to port the function computing the non-linear term of the Euler equation to CUDA. The MATLAB code is running in double precision and the data is transformed to single precision before it is transferred to the GPU. The computation on the GPU is performed in single precision and the result is transformed back to double precision before it is returned to MATLAB.

Table 1 shows the speed-up for the 2D isotropic turbulence case.

Table 1: Timing details for 400 Runge-Kutta steps on a 1024x1024 mesh for 2D isotropic turbulence simulation on Windows.

	Runtime	Speed-up
Standard MATLAB	8098 s	
Overload FFT2 and IFFT2	4425 s	1.8 x
Overload Non-linear term	735 s	11. x
Overload Non-linear term, FFT2 and IFFT2	577 s	14. x

The CUDA code is not yet taking advantage of the symmetries of real transforms (the original code was written when CUFFT was only supporting complex to complex transforms). The speed-up will increase even further by using real to complex and complex to real transforms (only half the data needs to be transferred and computed). Even with this limitation, the speed-up obtained is quite significant.

Figures 1a and 1b compare the final results of the original MATLAB implementation with the one accelerated with CUDA for the advection of an elliptic vortex. At the end of the computed time interval, the stream function and vorticity fields are virtually the same.

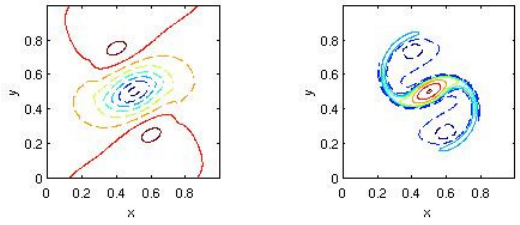

Figure 1a: Advection of an elliptic vortex on a 256x256 mesh, stream function (left), vorticity (right): MATLAB on Linux, 168 sec

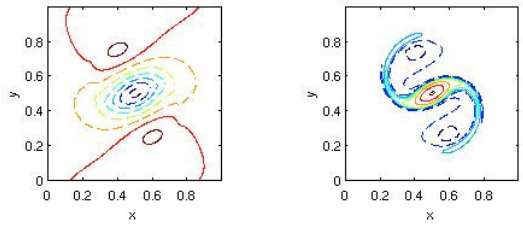

Figure 1b: Advection of an elliptic vortex on a 256x256 mesh, stream function (left), vorticity (right) :MATLAB with CUDA on Linux, 14.9 sec

To do a better comparison of the quality of the results, we have also run a 2D isotropic turbulence simulation at different resolutions and compared the vorticity power spectra of the 2 runs, one with the original MATLAB (Fig 2a) and the other one with MATLAB accelerated with CUDA (Fig 2b). Even for this quantity, that is very sensible to fine scales, the results are in excellent agreement.

The scripts created to compile CUDA MEX files are very easy to use. From the command prompt in MATLAB, the user needs to invoke nvmex instead of the regular mex script:

```
nvmex -f nvmexopts.bat filename.cu -IC:\cuda\include
-LC:\cuda\lib  -lcudart
```

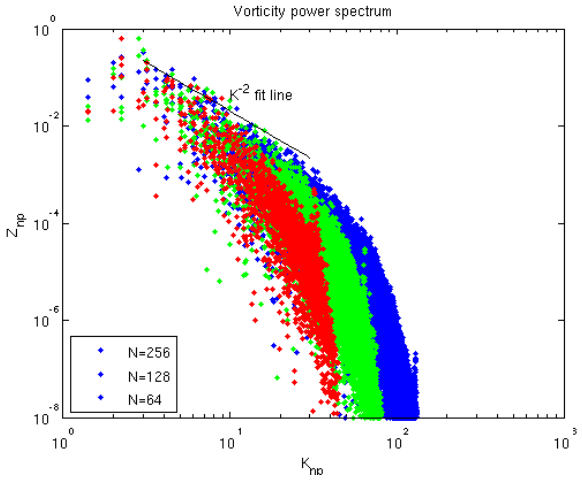

Figure 2a: Vorticity power spectrum for 2D isotropic turbulence, MATLAB .

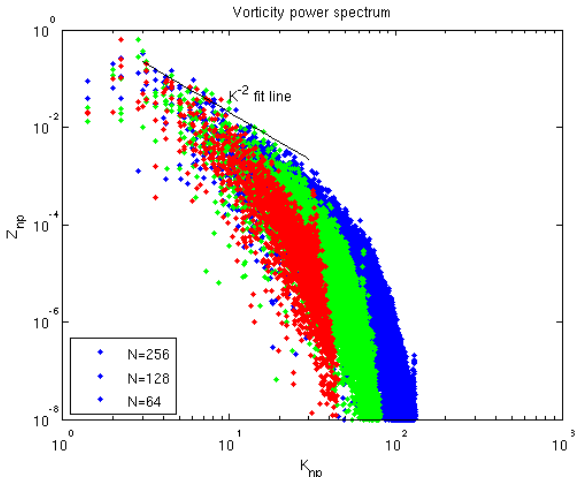

Figure 2b: Vorticity power spectrum for 2D isotropic turbulence, MATLAB with CUDA.

Conclusion

The combination of MATLAB and CUDA enables high-productivity and high-performance solutions and with GeForce 8 series GPUs now available even in laptops, will be a very effective tool for engineers and scientists.

The scripts to compile CUDA MEX files and some examples are now available for download [3].

References

[1] http://developer.nvidia.com/cuda

[2] http://www.amath.washington.edu/courses/571-winter-2006/matlab

[3] http://developer.nvidia.com/object/matlab_cuda.html