Performance of FORTRAN and C GPU Extensions for a Benchmark Suite of Fourier Pseudospectral Algorithms

B. Cloutier[1], B.K. Muite[2], P. Rigge[3]
Depts. of Physics[1], Mathematics[2] and Electrical Engineering[3]
University of Michigan
Ann Arbor, Michigan, USA
{cloutbra[1], muite[2], riggep[3]}@umich.edu

***Abstract*—A comparison of PGI OpenACC, FORTRAN CUDA, and Nvidia CUDA pseudospectral methods on a single GPU and GCC FORTRAN on single and multiple CPU cores is reported. The GPU implementations use CuFFT and the CPU implementations use FFTW. Porting pre-existing FORTRAN codes to utilize a GPUs is efficient and easy to implement with OpenACC and CUDA FORTRAN. Example programs are provided.**

***Keywords*-OPENACC; BENCHMARK; GPGPU; FORTRAN; FAST FOURIER TRANSFORM;**

I. Introduction

Graphics processing units (GPUs) can have better performance for mathematical operations on large arrays when compared to traditional central processing units (CPUs). The fast Fourier transform (FFT) is one application for which GPUs have a significant performance advantage over CPUs. The performance advantage can be significant for simulations which fit within the memory constraints of a single GPU.

GPU acceleration has largely been accomplished in variants of C with variants of FORTRAN being a recent addition. A comparison of performance of the lesser known OpenACC and CUDA FORTRAN on GPUs is of interest because a large number of legacy codes use FORTRAN, [8] [20] and because [10, Chp. 6] indicates that on CPUs, FORTRAN compilers typically generate more efficient scientific computing codes than C compilers. Previous works which have examined speedup offered by single GPUs in solving differential equations using Fourier transforms are [1], [4] and [21].

In this paper, extensions of FORTRAN for GPUs when solving nonlinear PDEs with pseudospectral methods are compared.The user friendliness of these different GPU programming models is described. A benchmark suite of algorithms for three different nonlinear PDEs with Fourier pseudospectral methods is also provided.

II. Programming Models

At present it is unclear what programming model will be widely adopted for accelerators. Some current options include OpenCL, OpenACC, OpenHMPP, F2C-ACC, CUDA, and CUDA FORTRAN. This note will compare CUDA C, CUDA FORTRAN, and OpenACC GPU programs to serial and OpenMP[13] FORTRAN CPU programs.

A. FORTRAN and OpenACC

OpenACC is a standard currently under development to allow for parallel programming of CPUs and GPUs[12]. It primarily uses directives to move data between a host CPU and a GPU and parallelize operations on the GPU. FORTRAN, C, and C++ are supported and there is a hope that it will be merged with OpenMP in the future to allow for a unified interface to GPUs and other accelerators.

B. CUDA C

CUDA C is a set of extensions to C that allow special functions, called kernels, to be executed on supported NVIDIA GPUs[11]. CUDA C provides a lower level interface than many of the other application interfaces discussed here.

C. CUDA FORTRAN

CUDA FORTRAN, is a set of extensions to FORTRAN that allows kernels to execute on NVIDIA GPUs[14]. The main constructs are similar to CUDA C; however, CUDA FORTRAN has several directives that can automatically generate kernels for common cases. CUDA FORTRAN provides both a high level interface similar to OpenACC and a low level interface like CUDA C.

III. An Overview of The Equations

The three choosen equations, which are of mathematical and physical interest, can be solved entirely using Fourier pseudospectral methods and simple representative timestepping schemes. The resulting programs are short, can be understood in their entirety by a single person and only require an FFT routine.

A. The Cubic Nonlinear Schrödinger Equation

The focusing two dimensional cubic nonlinear Schrödinger equation is

$$i\psi_t + \psi_{xx} + \psi_{yy} = |\psi|^2\psi, \tag{1}$$

where $\psi(x,y,t)$ is a complex valued function of time, t, and two spatial variables, x and y. This equation arises in a variety of contexts including quantum mechanics, in simplified models for lasers, and water waves. When ψ has periodic boundary conditions, the cubic Schrödinger equation has two conserved quantities,

$$\iint |\psi|^2 \mathrm{d}x\mathrm{d}y \quad \text{and} \quad \iint \frac{|\nabla\psi|^2}{2} - \frac{|\psi|^4}{4} \mathrm{d}x\mathrm{d}y, \qquad (2)$$

known as the mass and energy respectively; they can be used to assess the accuracy of a numerical solution. For more background on this equation, see [16] and [19].

B. The Sine-Gordon Equation

The $2D$ sine-Gordon equation,

$$u_{tt} - \Delta u = -\sin u, \qquad (3)$$

arises in many different applications, including propagation of magnetic flux on Josephson junctions, sound propagation in a crystal lattice, and several others discussed in [15]. It has a conserved Hamiltonian

$$H = \iint \frac{1}{2}\left(u_t^2 + |\nabla u|^2\right) + (1 - \cos u)\,\mathrm{d}x\mathrm{d}y, \qquad (4)$$

which is useful in evaluating the accuracy of numerical solutions.

C. The 2D Navier-Stokes Equation

The 2D incompressible Navier-Stokes equations in stream function (ψ)-vorticity (ω) form are

$$\omega_t + \psi_y\omega_x - \psi_x\omega_y = \Delta\omega \quad \text{and} \quad \Delta\psi = -\omega. \qquad (5)$$

These equations model fluid flow, and further background on them can be found in [18] among other references. The Taylor-Green vortex solution of these equations is

$$\omega(x,y,t) = 4\pi\sin(2\pi x)\sin(2\pi y)\exp(-8\pi^2 t). \qquad (6)$$

IV. Fourier Pseudospectral Methods

Fourier pseudospectral methods are a class of numerical methods to solve partial differential equations that utilize the Fourier transform. These methods became popular after the publication of [7]; other expositions are in [2], [3], [5], [9], [16], [17] and [19]. These methods utilize the fact that differentiation of a function is a simple and fast multiplication by the wave number in Fourier space. The nonlinear terms in are computed in real space. This section gives a brief description of the second order in time algorithms used. The programs are available at http://arxiv.org/abs/1206.3215.

Table I
A list of the compiler options used to build the codes. Codes in parentheses use all flags in row. PGI 12.4 requires the flag -Mlarge_arrays when more than 2Gb are allocated.

	Compiler	Flags	Libraries
GPU: Cuf and (OpenACC)	PGI 12.4	-O3 -Mcuda -Minfo (-acc) (-ta=nvidia)	CUFFT 4.1.28
GPU: C	CUDA 4.0 V0.2.1221	-O3 –arch=sm_20	CUFFT 4.1.28
CPU: 1 core and (16 core)	GFORTRAN GCC: 4.6.2	-O3 (-fopenmp)	FFTW 3.3

A. A Numerical Method for the Nonlinear Schrödinger Equation

The nonlinear Schrödinger equation is approximated by splitting it into two equations which can be solved exactly, [16] and [19]. The Fourier transform of ψ will be denoted by $\hat{\psi}$. In Fourier space one first solves

$$i\psi_t + \psi_{xx} + \psi_{yy} = 0 \qquad (7)$$

for half a time step, $0.5\delta t$. Letting k_x and k_y denote the wave numbers in the x and y directions, eq. (7) is solved by $\hat{\psi}(t = 0.5\delta t) = \exp[-i(k_x^2 + k_y^2)\delta t]\hat{\psi}(t = 0)$. The solution of

$$i\psi_t = |\psi|^2\psi \qquad (8)$$

for a full time step is $\psi(t = \delta t) = \exp(i|\psi(t = 0)|^2\delta t)\psi(t = 0)$, since $|\psi|^2$ is conserved. Finally eq. (7) is solved for another half time step to get the solution a full time step later.

B. A Numerical Method for the Sine-Gordon Equation

Following [6], the second derivative in time is approximated by a central difference. The resulting numerical method is

$$\frac{u^{n+1} - 2u^n + u^{n-1}}{\delta t^2} + \Delta\left(\frac{u^{n+1} + 2u^n + u^{n-1}}{4}\right) = -\sin u^n \qquad (9)$$

C. A Numerical Method for the 2D Navier-Stokes Equation

Time is discretized using the Crank-Nicolson method, where the nonlinear terms are solved for using fixed point iteration

$$\frac{\omega^{n+1,k+1} - \omega^n}{\delta t} - \frac{1}{2}\Delta\left(\omega^{n+1,k+1} + \omega^n\right) + \frac{1}{2}\left(\psi_y^{n+1,k}\omega_x^{n+1,k} - \psi_x^{n+1,k}\omega_y^{n+1,k} + \psi_y^n\omega_x^n - \psi_x^n\omega_y^n\right) = 0. \qquad (10)$$

The superscript n denotes the time step and the superscript k denotes the iterate. The fixed point iterations stop when

$$\text{tolerance} > \iint (\omega^{n+1,k} - \omega^{n+1,k+1})^2 \mathrm{d}x\mathrm{d}y. \qquad (11)$$

Table II
Computation times in seconds for 20 time steps of 10^{-5} for a Fourier split step method for the cubic nonlinear Schrödinger equation on $[-5\pi, 5\pi]^2$.

Grid Size	GPU (Cuf)	GPU (C)	GPU (OpenACC)	CPU (16 cores)	CPU (1 core)
64^2	0.00292	0.00618	0.00272	0.0180	0.0180
128^2	0.00366	0.007189	0.00459	0.0130	0.086
256^2	0.00802	0.0116	0.0130	0.148	0.442
512^2	0.0234	0.0315	0.0369	0.562	1.94
1024^2	0.0851	0.105	0.132	2.27	12.7
2048^2	0.334	0.415	0.527	9.67	57.2
4096^2	1.49	2.02	2.30	37.7	329
8192^2	6.30	N/A	N/A	292.4	1454

Table III
Computation times in seconds for 500 time steps of 10^{-3} the sine-Gordon equation on $[-5\pi, 5\pi]^2$. N/A implies that the code could not be run due to memory constraints.

Grid Size	GPU (Cuf)	GPU (C)	GPU (OpenACC)	CPU (16 cores)	CPU (1 core)
64^2	0.028	0.025	0.028	0.050	0.098
128^2	0.040	0.031	0.041	0.107	0.401
256^2	0.099	0.065	0.095	1.960	1.899
512^2	0.343	0.260	0.344	2.925	9.301
1024^2	1.148	0.976	1.218	19.07	38.34
2048^2	4.485	4.007	4.869	67.84	165.9
4096^2	18.09	16.53	19.95	481.2	785.5
8192^2	85.45	N/A	93.51	934.2	4556

Table IV
Computation times in seconds for 20 time steps of 1.25×10^{-3} for the incompressible Navier-Stokes equation on $[0, 1]^2$. Each simulation generated the same max error of 1.65×10^{-5} when compared to the exact Taylor-Green solution. The tolerance for fixed point iterations was set to 10^{-10}.

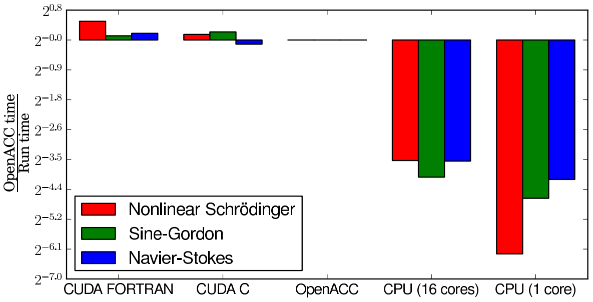
Grid Size	GPU (Cuf)	GPU (C)	GPU (OpenACC)	CPU (16 Cores)	CPU (1 core)
64^2	0.0151	0.0164	0.0171	.0018	0.014
128^2	0.0201	0.0266	0.0204	0.024	0.065
256^2	0.044	0.0435	0.0418	0.417	0.37
512^2	0.119	0.141	0.1205	0.939	1.734
1024^2	0.357	0.520	0.398	4.89	7.65
2048^2	1.386	1.88	1.59	17.04	33.56
4096^2	5.814	7.48	6.79	112.83	172.67

V. Results

All three equations are solved using different numerical methods so actual computation time comparisons between methods are of less interest than common scaling trends. For this comparison all of the codes were compiled with the compilers and flags shown in Table I.

The CPU simulations were run on AMD Opteron Magny-Cours 6136 2.4 GHz dual-socket eight-core processors with 48GB of memory and the GPU simulations on Nvidia Fermi M2070 with 6GB of memory per GPU. Double precision floating point arithmetic was used. Computation times are only measured for advancing the numerical solution forward in time, they do not include setup time for creating FFT plans, allocations, calculating exact solutions, etc. because in production simulations where many time steps are taken, these will be negligible. Performance differences between FORTRAN compilers on CPUs do not change the conclusions.

Figure 1. Comparison of OpenACC benchmark time to CuF, CUDA C, and CPU times for a grid size of 4096^2 (bigger is better).



A. The Cubic Nonlinear Schrödinger Equation

The programs used 4 double complex arrays, 2 arrays for the actual computation and 2 further arrays, to calculate the mass and energy. In all tests, the mass was conserved up to machine precision and the energy was constant to within 6 significant figures.

Table II shows that the GPU gives a speed up of up to a factor of 40 compared to a multicore OpenMP CPU implementation. An initial naive CUDA FORTRAN implementation for which kernel parallelization options are left to the compiler was typically a factor of 2 slower than the CUDA C implementation for which block sizes were specified. GPUs had speed ups of order 100 times compared to a single CPU core. Figure 1 shows that for this code, CUDA Fortran had the best performance.

B. The Sine-Gordon Equation

The programs used real-to-complex Fourier transforms. They required 2 double precision arrays of size $N_x \times N_y$ and 3 double complex arrays of size $N_x \times (N_y/2+1)$. In all tests, the final Hamiltonian was within 6 significant figures.

Table III shows that CUDA C performs the best, followed closely by CUDA FORTRAN and OpenACC. The differences between the GPU implementations were relatively small compared to the difference between GPU and CPU implementations. The GPU implementations performed on the order of 50 times better than a single core CPU. Figure 1 shows that for this code, CUDA C had the best performance.

C. The 2D Navier-Stokes Equation

The programs use the same real-to-complex Fourier transforms as the sine-Gordon equation, where the real-valued input array has a size of $N_x \times N_y$ and the complex-valued output array is of size $N_x \times (N_y/2+1)$. 10 arrays are used for the GPU codes and 11 arrays are used for CPU codes, where the extra array is required because FFTW does not preserve its input. Five complex-real transforms and 1 real-complex transform are used in the timestepping loop.

Table IV shows that the GPU gives a speed up of up to a factor of 30 compared to a single CPU core. Figure 1 shows that for this code, OpenACC does better than CUDA C, and CUDA FORTRAN has the best performance.

VI. User Comparison of the different GPU programming environments

Porting existing FORTRAN codes to CUDA FORTRAN was simple and intuitive; the combination of FORTRAN and Open ACC was a little less intuitive, but still relatively straightforward. The differences between CPU and GPU versions with CUDA FORTRAN or OpenACC are small, so it is feasible to maintain CPU and GPU versions of the same code. In contrast, porting a FORTRAN code to CUDA C is error prone, requires significant reprogramming and a good understanding of the GPU architecture.

Finally, [8] observed that the F2C-ACC directive based FORTRAN to CUDA compiler results in a runtime code with better performance than regular CUDA FORTRAN. The performance of CUDA FORTRAN codes may experience very different speedups with minor changes to the parallelization or compiler options. Choosing these options optimally requires some knowledge of the underlying architecture – autotuning compilers would be useful for doing this.

Acknowledgment

This work used the computers Forge and Kollman within the Extreme Science and Engineering Discovery Environment, which is supported by National Science Foundation grant number OCI-1053575. B. Cloutier and P. Rigge were supported by the University of Michigan Undergraduate Research Opportunity Program and Blue Waters Undergraduate Petascale Education Program respectively. The authors acknowledge support from a faculty grant for innovations in teaching with technology from the University of Michigan and a Blue Waters Undergraduate Petascale Module Development grant from the Shodor Foundation. They thank Bennet Fauber, Robert Krasny, Carl Ponder, Sarah Tariq, Divakar Viswanath, Jared Whitehead and Brian Wylie for help and suggestions, and the reviewers for a careful reading.

References

[1] H. Bauke, C. H. Keitel, *Accelerating the Fourier Split Operator Method via Graphics Processing Units*, Computer Physics Communications 182, 2454-2463, 2011.

[2] J. P. Boyd, *Chebyshev and Fourier Spectral Methods*, Dover, 2001

[3] C. Canuto, M. Y. Hussaini, A. Quarteroni and T. A. Zang, *Spectral Methods: Fundamentals in Single Domains*, Springer, 2006.

[4] F. Chen, *Efficient Spectral Methods for Coupled Elliptic Systems with Applications in Phase-field Modeling*, Midwest PDE seminar, Notre Dame, May 2012.

[5] G. Chen, B. Cloutier, N. Li, B.K. Muite, P. Rigge and S. Balakrishnan, A. Souza, J. West, *Parallel Spectral Numerical Methods*, Submitted, 2012.

[6] R. Donninger and W. Schlag, *Numerical Study of the Blowup/Global Existence Dichotomy for the Focusing Cubic Nonlinear Klein-Gordon Equation*, Nonlinearity 24, 2547-2562, 2011.

[7] D. Gottlieb and S. A. Orszag, *Numerical Analysis of Spectral Methods: Theory and Applications*, SIAM, 1999.

[8] T. Henderson, J. Middlecoff, J. Rosinski, M. Govett, P. Madden, *Experience Applying Fortran GPU Compilers to Numerical Weather Prediction*, pp. 34-41, Proc. Symposium on Application Accelerators in High Performance Computing (SAAHPC), 2011.

[9] J. S. Hesthaven, S. Gottlieb, D. Gottlieb, *Spectral Methods for Time-Dependent Problems*, Cambridge University Press, 2007.

[10] J. Levesque and G. Wagenbreth, *High Performance Computing: Programming and Applications*, CRC Press, 2011.

[11] NVIDIA. *NVIDIA CUDA C Programming Guide, Version 4.2.* NVIDIA, April 2012.

[12] OpenACC-standard.org, *The OpenACC^{TM} Application Programming Interface*, 2011

[13] OpenMP Architecture Review Board, *OpenMP Application Program Interface, Version 3.1*, OpenMP Architecture Review Board, 2011.

[14] The Portland Group. *CUDA FORTRAN Programming Guide and Reference.* The Portland Group, 2012.

[15] A. Scott, F. Chu, and D. McLaughlin, *The Soliton: A New Concept in Applied Science*, Proceedings of the IEEE, Vol. 61, No. 10, 1973.

[16] J. Shen, T. Tang and L. -L. Wang, *Spectral Methods: Algorithms, Analysis and Applications*, Springer, 2011.

[17] L. N. Trefethen, *Spectral Methods in Matlab*, SIAM, 2000.

[18] D. J. Tritton, *Physical Fluid Dynamics*, Clarendon Press, 1988.

[19] J. Yang, *Nonlinear Waves in Integrable and Nonintegrable Systems*, SIAM, 2010.

[20] F. Zafar, D. Ghosh, L. Sebald and S. Zhou, *Accelerating a climate physics model with OpenCL*, pp. 24-33, Proc. Symposium on Application Accelerators in High Performance Computing (SAAHPC), 2011.

[21] J.H. Zhang, S.-Q. Wang, Z.-X. Yao, *Accelerating 3D Fourier Migration with Graphics Processing Units*, GEOPHYSICS Vol. 74, no. 6, 129-139, 2009.