Accelerating the Initial Value Problem of Ordinary Differential Equations using GPUs for following Field lines in a Toroidal Domain

Anikait Singh

asap7772@berkeley.edu

Contents

*Mentored by by Dr. Stephane Ethier, Garret Wright, and Dr. Samuel Lazerson of the Princeton Plasma Physics Laboratory

1 Introduction

The Stellarator Optimizer Library is a package of various equilibrium magnetic confinement fusion software. A tool that is heavily used throughout the package is an Ordinary Differential Equation solver. This is done through common numerical methds such as the Runge-Kutta-Fehlberg 45 algorithm, Backwards Differentiation Formula, Adams-Moulton Method and the Predictor Corrector set of algorithms.

Previously, the ODE Solver, LSODE, the Livermore Solver for Ordinary Differential Equations from Lawrence Livermore National Lab was utilized. We have now transitioned to using the solver, CVODE from the SUNDIALS suite of ODE Solvers from Lawrtence Livermore National Lab, because of its support of CUDA, a parallel computing platform that allows one to interact with a GPU, a coprocessor that allows one to accelerate their task by running parallel tasks concurrently. In order to do this an interface was created between the FORTRAN code of the STELLOPT library and the C code used to invoke CUDA.

Also, the function that was used to calculate the Right Hand Side of the ODE and the Jacobian of the differential equation was accelerated through a kernel. A kernel is a GPU function that is meant to be called from CPU code and can be executed on a grid of blocks which each has a certain amount of threads. This allows the function to be called in a parallel manner by multiple threads.

2 Core Tools Used:

There are multiple tools that were utilized during this summer that were used to for the core functionality and also allowed for more efficient code development

2.1 CUDA

Cuda is a parallel computing platform that can be used for general computing on GPU's. This was the primary interface that allowed for the management of memery on both the device (the GPU) and host (the CPU) along with the parallel function structure (the kernel). This is important because each processor has an independent set of memory and thus memory transfer between each device is essential to this style of programming. Not all programs are compatible with CUDA because each thread on the GPU has to run in an independent manner. In our code, CUDA was utilized to do parallel lookups in a multidimensional array and to perform an aggregate sum within the spline interpolation function.

2.2 Fortran C interface

This was used to enable invoking a C function from a Fortran subroutine and utilize the value returned from the function within Fortran. This was done using Fortran and C Interoperability.

2.3 Matlab

Matlab was primarily used to produce Poincare plots from the HDF5 files that were outputted from the FIELDLINES package of the STELLOPT Library. It was also used for testing purposes for the different along with experimenting with RKF-45

2.4 Makefiles

Makefiles are a useful method for compiling large amounts of program files in a consistent manner. This is done by declaring each compiler used, linking the libraries used in the porject, and specifiying a rule for how each type of file is compiled to create Object Files. This allowed for the compilation of the code to be automated and clearly defined.

2.5 Unix Commands

ENACS and other Linux tools were utilized for a more efficient programming style. This was necessary since the machine that was used was accessed through SSH so there wes no visual interactive environment.

3 Numerical Methods to Solve ODE's

3.1 Background

Ordinary Differential Equations is an equality involving a function and its derivatives of the form:

$$f(x, y, dy/dx, d^2y/dx^2, ...) = 0$$

ODE's can also be expressed as a system of first order differential equations, which is used to numerically compute the solution to the initial value problem. This is done by breaking up the large

3.2 Runge Kutta Algorithm

The Runge Kutta method is a family of explicit and implicit numerical methods to solve ODEs. They are iterative methods that include the commonly taught Euler's Method. The most commonly used algorithm is the RK4 algorithm, where the value at the next step is determined by the weighted average of 4 equidistant increments.

Another algorithm in the family is the Runge-Kutta-Fehlberg 4-5 algorithm. It is an embedded method to solve nonstiff Ordinary Differential Equations. This method allows for an adaptive step size by generating a value for error as the difference between the value computed by taking intermediate steps and one larger step thus increasing the order of the method.

3.3 Linear Multistep Methods

Multistep methods are a form of numerical methods that are efficient because they use n previous steps to determine the value of the dependent variable at a certain step. There are two common forms of linear multistep methods that are heavily used: the Adams-Moulton and Backward Differentiation algorithms. The Adams-Moulton method is an algorithm that works better on nonstiff equations whereas the Backward Differentiation method is better for stiff equations

3.4 Predictor Corrector Method

Predictor-Corrector algorithms are a class of algorithms that initially extrapolates the value for the next point and then refines the initial approximation using the predicted value and another method to interpolate the function at the same point. The predictor method is usually an embedded, explicit method to provide since this method is quick and only an initial approximation is necessary.

4 Method Chosen for the STELLOPT Library

Since the equations that are used to solve a system of two stiff first-order differential equations, the backward differentiation linear multistep method was chosen with a Newton-Raphson predictor-corrector, a variable-order, variable-step multistep method. Thus the libraries LSODE was chosen in the prior implementation of the FEILDLINES Package of the Stellarator Optimizer Library created by Dr. Samuel Lazerson.

Since LSODE was not compatible with CUDA, the parallel computing platform for NVIDIA GPUs, the CVODE library was used to replace it (from the sundials suite of Lawrence Livermore National Labs). To interface with solver, the Spline Interpolation, Jacobian and Right Hand Side functions and kernels were written in C. CUDA was used for the transfer of memory between the CPU and GPU and the acceleration of the spline interpolation routine in the form of a kernel.

5 Serial vs GPU Programming

Traditional programming is done in a serial manner where one statement is executed one after the other. This can be accelerated by having multiple threads that have distributed/independent memory which allows for multiple serial tasks to execute in a parallel manner

GPU computing utilizes the GPU (Graphics Processing Unit) as a coprocessor. This is advantageous because of the hundreds of cores that the GPU which is much larger than 8-16 cores that a CPU has.

The tradeoff for using a coprocessor is the copying of memory between host and device which is quite slow and can result in errors if not done properly. However, by providing read-only data, the transfer is much faster since it is unidirectional.

6 Results

6.1 Computational Accuracy

There computational difference between CVODE and LSODE was negligibly different. This is seen in the Poincare Trajectory plots below of R vs Z (in cylindrical coordinates. These points are evenly space and linear which demonstrate that the trajectory was correctly created.

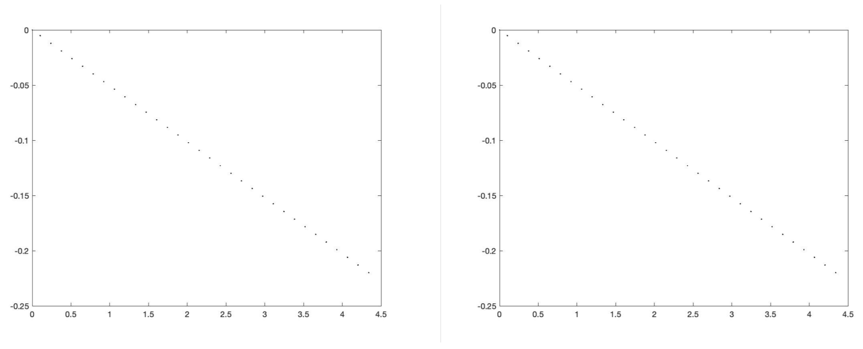

Figure 1: Poincare Plots of Cvode vs LSODE

The overall trajectory was also consistent and shown below.

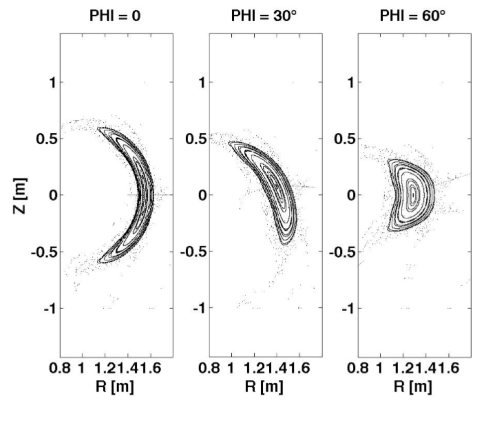

Figure 2: Stellarator Magnetic Fieldline Trajectories

6.2 Speed Difference

There was a negligible increase in speed between the multicore LSODE implementation and GPU CVODE implementation. This is because some of the time that was saved from parallelizing the spline interpolation was spent on initially moving data between the device and host memory (from the CPU to GPU). A larger amount of difference in speed could be seen if used for a larger set of parallel calculations or a more recent graphics architechture was used. Below is the result from Nvidia's Visual Profiler, which shows where time was spent in the program between the CPU and GPU:

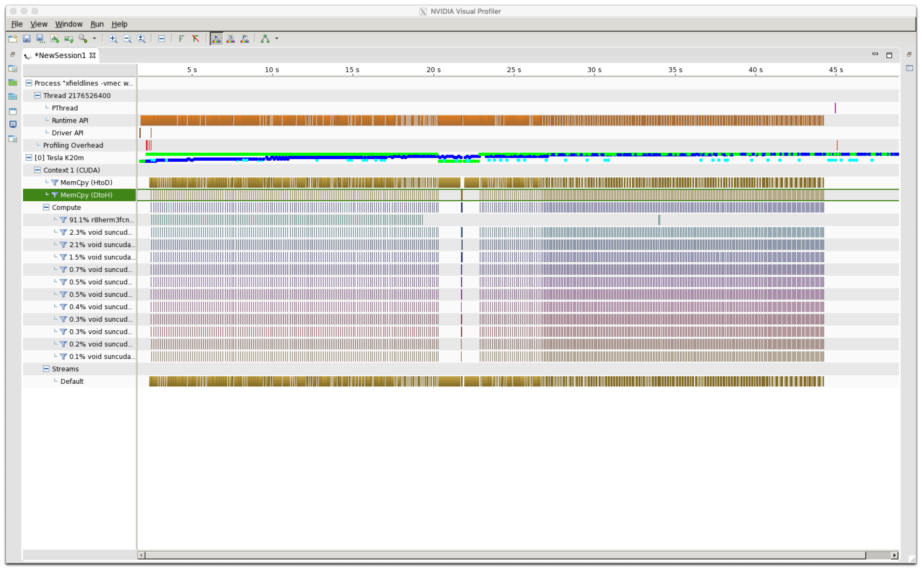

Figure 3: Nvidia Visual Profile Output

7 References

[1] "CVODE." Computing, Lawrence Livermore National Laboratory, 28 June 2019, https://computing.llnl.gov/projects/sundials/cvode

[2] Lazerson, S. A., Chapman, I. T. (2013). STELLOPT modeling of the 3D diagnostic response in ITER.Plasma Physics and Controlled Fusion, 55(8), 084004. doi: 10.1088/0741-3335/55/8/084004

[3] Press, W. H., Teukolsky, S. A., Vetterling, W. T. (1992).Numerical Recipes in C The Art of Scientific Computing Second Edition. Cambridge: Cambridge University Press.