GPU Accelerated Mesh free Solvers for Compressible Flows

Nischay Ram mamidi[1], Kumar Prasun[2], Srikanth C.S.[3]and Anil Nemili[4]
Department of Mathematics, BITS-Pilani, Hyderabad Campus, Hyderabad, 500078
SM Deshpande[5]
Engineering Mechanism Unit, JNCASR, Bengaluru, 560064

Keywords: Python, Julia, FORTRAN, GPUs, CUDA, LSKUM, Mesh free methods.

INTRODUCTION

The Least Squares Kinetic Upwind Method (LSKUM) [1] is a mesh free solver for the numerical solution of Euler and Navier-Stokes equations of the compressible fluid flows. Over the years, LSKUM based mesh free solvers developed at Defense Research and Development Laboratory (DRDL) and National Aerospace Laboratories (NAL) have been continuously used to compute flows around a wide variety of configurations.

The mesh free solvers in these Laboratories are written in traditional programming languages like C, C++ and FORTRAN. Porting these legacy codes to HPC platforms with rapidly evolving hardware is challenging as the developers need to tune their codes frequently. One way to circumvent this problem is to employ modern languages such as Python or Julia. These languages are known to be architecture independent with an added advantage of easy code maintenance and readability. Furthermore, Julia is designed for high performance computing of numerically intensive algorithms. Recently, a hybrid parallel CFD solver called PyFR [2] has been developed in Python. This code has been successfully tested on massively parallel modern hardware platforms scaling to Petaflops. In other works, a parallel code, Celeste [3], written in Julia is used to perform Petascale operations for astronomical applications.

The long term aim is to develop LSKUM based Python and Julia mesh free solvers that can be run on hybrid platforms such as GPGPUs and CPUs+GPUs. As a first step towards this objective, in the present work, an attempt has been made to develop both serial and GPU accelerated mesh free solvers for two-dimensional flows. The computational efficiency of these solvers is compared with equivalent FORTRAN based serial and GPU solvers. Furthermore, analysis of several performance metrics such as Streaming Multiprocessor (SM) utilization and achieved occupancy gives very useful information for future development.

To measure the performance of mesh free solvers, we adopt a cost metric called the Rate of Data Processing (RDP). The RDP of a mesh free code can be defined as the total wall clock time in seconds per iteration per point. The RDP values based on a single core CPU calculations are listed in Table 2. Compared to the FORTRAN code, the performance of the python code is observed to be very poor as its RDP values are larger

NUMERICAL RESULTS

In this section, we present numerical results to assess the computational efficiency of the Python, Julia and FORTRAN based GPU accelerated mesh free solvers. The test case under investigation is the in viscid fluid flow simulation around NACA 0012 airfoil at Mach number M = 0.85 and angle of attack AoA= 1°. For the benchmarks, six levels of uniformly refined point distributions are used. The coarsest distribution consists of 9, 600 points, while the finest distribution consists of 98, 30, 400 points. Note that further point refinement is not pursued as the desired memory requirements exceed the available GPU resources. All computations are performed with double precision on a Linux workstation, whose configuration details are presented in Table 1.

	CPU	GPU
Model Intel Xeon	$E5 - 2698$ v4	Nvidia Quadro $M5000$
Cores	40 (20×2)	2048
Core Frequency	2.20 GHz	1.038 GHz
Global Memory	128 GB	8 GB
$L2$ Cache	5 MB	2 MB

TABLE 1: CONFIGURATION OF THE WORKSTATION USED TO PERFORMSIMULATIONS.

To measure the performance of mesh free solvers, we adopt a cost metric called the Rate of Data Processing (RDP). The RDP of a mesh free code can be defined as the total wall clock time in seconds per iteration per point. The RDP values based on a single core CPU calculations are listed in Table 2. Compared to the FORTRAN code, the performance of the Python code is observed to be very poor as its RDP values are larger by an order of O 10^2. This behavior is anticipated as pure Python is an interpreted and dynamically typed language while FORTRAN is a compiled and statically typed language. On the other hand, while Julia is also a dynamically typed language, its compiler can statically type functions through multiple dispatch. Due to this, the RDP values based on Julia are better than Python but its performance is still slower than FORTRAN.

Number of points	Python serial code	Julia serial code	Fortran serial code
160 × 60 = 9600	10.3562×10^{-4}	5.4479×10^{-5}	8.5423×10^{-6}
320 × 120 = 38400	9.8451×10^{-4}	6.0964×10^{-5}	8.0099×10^{-6}
640 × 240 = 153600	9.7971×10^{-4}	7.9173×10^{-5}	7.2401×10^{-6}
1280 × 480 = 614400	9.4958×10^{-4}	14.3439×10^{-5}	5.9362×10^{-6}
2560 × 960=2457600	8.3596×10^{-4}	27.8621×10^{-5}	4.8165×10^{-6}
5120 × 1920 = 9830400	6.1861×10^{-4}	31.8860×10^{-5}	2.7674×10^{-6}

TABLE 2: COMPARISON OF THE RDP VALUES FOR A SINGLE CORE CPU COMPUTATION.

It is well-known that the overall performance of a GPU accelerated code depends on the number of threads employed per block. To find the optimal number of threads for the present algorithm and the hardware, we perform numerical experiments with 8, 16, 32, 64, 128 and 256 threads per block. Note that with 512 and 1024 threads, the simulations crashed due to lack of sufficient thread memory. For the Python and Fortran GPU codes, the optimal choice for the number of threads on all levels of point distribution is foundtobe32. On the other hand, for the Julia GPU code, the optimal number of threads is observed to be 128.

Table 3 shows a comparison of RDP values based on the GPU accelerated mesh free codes. From these values, we can observe a very significant improvement in Python GPU code performance over its serial version. In fact, on medium levels of point distribution, the Python GPU code exhibits superior performance over the FORTRAN version. Perhaps, this could be due to the efficient utilization of streaming multiprocessors (SM) by the Python GPU code. On the other hand, the performance of Julia GPU code is observed to be slower than both Python and FORTRAN. Perhaps, this slowdown can be due to the current implementation which requires a larger data structure. Furthermore, Julia CUDA libraries are relatively newer and lack many features.

Number of points	Python GPU code	Julia GPU code	Fortran GPU code
9600	10.9781×10^{-7}	12.1875×10^{-7}	8.9708×10^{-7}
38400	4.4550×10^{-7}	11.1458×10^{-7}	6.7234×10^{-7}
153600	2.9135×10^{-7}	10.7617×10^{-7}	5.0638×10^{-7}
614400	2.3671×10^{-7}	10.7617×10^{-7}	3.6609×10^{-7}
2457600	2.1370×10^{-7}	11.1108×10^{-7}	2.5656×10^{-7}
9830400	2.0654×10^{-7}	9.0841×10^{-7}	1.5623×10^{-7}

TABLE 3: COMPARISON OF THE RDP VALUES BASED ON THE GPU ACCELERATED MESHFREE CODES.

In order to assess the overall performance enhancement due to GPU computing, we define the speedup as the ratio of the RDP based on a single core CPU simulation to the RDP obtained using the GPU accelerated code with optimal number of threads per block. Table 4 shows the relative speedup achieved by the GPU codes. It can be observed that continuous point refinement enhanced the utilization of GPU computing power and thus increased the speedup. In the case of Python, the speedup has reached saturation on level 4 point distribution, while for FORTRAN the speedup got saturated on level 5 point distribution. Beyond this point, the speedup decreases as the computational effort on the finest distribution exceeds the available GPU resources. Note that a reasonable explanation for massive speedup achieved by the Python GPU code is due to a very poor performance of its serial code.

Number of points	Python GPU code	Julia GPU code	Fortran GPU code
9600	943	44.7	9.5
38400	2210	54.7	11.9
153600	3363	73.6	14.3
614400	4012	133.3	16.2
2457600	3912	250.8	18.8
9830400	2995	351.0	17.7

TABLE 4: RELATIVE SPEEDUP OF GPU ACCELERATED CODES OVER A SINGLE CORE CPU COMPUTATION.

OUTLOOK OF THE FULL PAPER

In the full paper, the following aspects will be improve, added

- Details pertaining to the development of Mesh free GPU solvers.
- Performance metrics of the kernels invoked in GPU codes.
- RDP values based on optimized GPU solvers.
- References will be updated.

REFERENCES

[1] A.K. Ghosh and S.M. Deshpande. Least squares kinetic upwind method for in viscid compressible flows. AIAA paper 1995-1735, 1995.

[2] F.D.Witherden, A.M.Farrington, and P.E.Vincent. PyFR: Anopensource frame work for solving advection–diffusion type problems on streaming architectures using the flux reconstruction approach. Computer Physics. Communication on s, 185(11):3028–3040, Nov 2014.

[3] J. Regier, K. Pamnany, K. Fischer, A. Noack, M. Lam, J. Revels, S. Howard, R. Giordano, D. Schlegel, J. McAuliffe, R. Thomas, and Prabhat. Cataloging the visible universe through bayesian inference at petascale. In 2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pages 44–53, May 2018.