Scandic Journal of Advanced Research and Reviews
ISSN: 2703-965X CODEN (USA): SJARCA
Cross Ref DOI: dx.doi.org/10.55966/sjarr
Journal homepage: www.sjarr.com

Implementing a Multigrid Tecnique For Stokes Equations Using Matlab

Muhammad Suleman Sial¹, Wajid Ahmed Siyal² and Muhammad Afzal Sahito³

1, MS Scholar, Department of Basic Sciences and Related Studies, Mehran University of Engineering and Technology, Jamshoro, Sindh, Pakistan

sulemansial02@gmail.com

2, MS Scholar, Sukkur IBA Community College, Naushahroferoze

wajidsiyal.ccnf@iba-suk.edu.pk

3, MS Scholar, Department of Basic Sciences and Related Studies, Faculty of Science Technology and Humanities, Mehran University of Engineering &Technology, Jamshoro.

afzalsahito87@gmail.com

Scandic Journal of Advanced Research and Reviews, 2022, 3(03), 033–040
Article DOI: https://doi.org/10.55966/sjarr.2022.3.3.0055

Abstract—**In computational models of geomorphologic processes Stokes equations have been employed. Multigrid method is added to our solver so that we may construct a solution for these problems. When trying to solve the ellipse differential equation with ill-conditioned matrix, the interpolation method is frequently used to start reducing the iterative and incremental stages due to the point sets inside the matrix merging this same momentum as well as mass formulae and the firmly customisable viscosity due to rheology. Utilizing advantages of the Matlab and the capabilities of the available Graphic Processing Units (GPUs), we accelerate the original Matlab routines using the Compute Unified Device Model (CUDA).**

Keywords*— *matlab, stroke equations, GPU, multigrid, matrix variables.

I. Introduction

Since NVIDIA initially developed the Cpu Architecture (CUDA) in 2007, graphics rendering units (GPUs) are being employed more frequently to tackle numerical issues. Thanks to the special design of single data many thread (SIMT) it employs, modern GPUs excel in parallel computing. The GPU, which has a different design philosophy from the central processing unit (CPU), is focused on throughput and has multiple cores, although it lacks branch prediction, a powerful or appropriate cache, and data forwarding capabilities. GPUs compare favourably with computer clusters in terms of energy the same level of computing power. A parallel programming model called CUDA was created to control thousands of threads running on GPU hardware components called streaming multiprocessors (SMs) and streaming processors (SPs). At the virtual level, CUDA threads are arranged in grids and blocks, where a grid contains several blocks and a block has many threads. This architectural structure enables shared memory communication across threads in the same block utilizing SMs and barrier synchronisation. Each block's number of threads and each grid's number must be specified explicitly in the CPU code (host code), which is typically written in C or another common language like Python or Fortran.

Creating CUDA apps is still difficult, particularly given that the majority of scripts are not written in C. On the one hand, the algorithms are frequently implemented and optimised using scripting languages like Matlab and Python. Furthermore, script language is frequently used in scientific computation as an interface to call C and Fortran codes since it lessens the programming labour without sacrificing the fundamental performance. Fortunately, certain script programming tools have already provided support for porting CUDA kernels, such as Jacket Matlab toolbox1 and PyCUDA. The most recent version of Matlab includes a toolbox for parallel computing that supports GPU computation. Parallel thread execution (PTX) can be called directly in Matlab 2010b as opposed to the Jacket Matlab toolbox. For CUDA programming, PTX offers a low-level set of instructions that is comparable to the assembly language used in the x86 architecture. With the -ptx flag at the command line during compilation, the handwritten CUDA kernel can be converted into PTX codes, which can then be invoked as Matlab functions. In this article, we go into great depth on how to use PTX kernels, which are called by the Matlab 2010b, to analyze the lobed Stokes flow problem.

Implementing Graphics Processing Units (GPU) for analyzing numerical problems has grown in popularity ever since NVIDIA published the Computer Unified Device Infrastructure (CUDA) in 2007[1]. Modern graphics processing units (GPUs) provide excellent parallel computing capability due to their distinctive architecture using the Single Instruction Multiple Thread approach (SIMT). Additionally, compared to a PC cluster, the energy consumption of a GPU is quite low[2]. To manage the hundreds of threads running on the GPU's Streaming Multiprocessor systems (SM) and Scalar Processors (SP) at the hardware level, CUDA leverages the Grid and Block paradigm at the software level. At this point, creating CUDA programs is still challenging, especially if you already possess a large body of completed code that is not implemented in C[1]. It is inefficient to translate all the scripts into C and CUDA. Since Mathworks published Matlab 2010b in September 2010[3], we can simply test the real CUDA codes using the Matlab interfaces by executing Simultaneous Threads Implementation instructions with Matlab's parallel computing toolbox. For GPU, we already have PyCUDA (Python call CUDA kernels). In this research, we concentrate on calling PTX kernels built from CUDA routines utilising the Matlab interface to solve the Stokes flow issue.

In contrast to the one operating on the finest grids, iterative information spreads more quickly, and residuals with a wave length decay more quickly. Through three operations—restriction, smoothing, and prolongation—MG techniques may solve problems on many grids with various resolutions. While prolongation merely interpolates the coefficients in the opposite direction, restriction project the equations from finer grid to the coarser one. Limited iterations of smoothing (smoother) are performed at each point of various grids. Geometric multigrid (GMG) and algebraic multigrid are the two MG techniques that are widely employed (AMG). AMG is less effective in performance than GMG but more applicable than GMG, especially for the finite element approach which requires an expressly built-up linear matrix system. Although a GPU-based GMG solution has already been published, it was entirely redone in C, preventing us from using the preexisting scripts. Along with a pretty basic three-dimensional version using MATLAB and CUDA, we also created a the double GMG solver that was entirely rewritten in CUDA. By using a cubic sinking model (SINKER) to solve the Stokes flow problem under the Cartesian coordinate system where it is assumed that a rising and high-density block sinks in the fluid intermediate part with a low viscosity, we introduce an optimised three-dimensional GMG implementation in detail in this study. The block part and the medium section are contrasted by the viscosity structure at 106. Free slip condition conditions are assigned to each and every velocity boundary.

II. BACKGROUND

Because our earth acts like an incompressible creeping flow over very long time scales, solving the Stokes flow issue is crucial for computational geodynamics, which can explain many geodynamic phenomena[4]. For millions of years, for instance, the basement and subsidence can be thought of as flows with extremely high viscosities. Because the Renold Number for Stokes flow is so close to zero, the advective transport factor Stokes equations can be ignored. Typically, we refer to the steady state Stokes flow problem as being represented by the Stokes equations, which are shown below[5]:

$$\frac{\partial ui}{\partial xj} = 0 \qquad (1)$$

$$\frac{\partial \sigma ij}{\partial xj} - \frac{\partial P}{\partial xi} + \rho gi = 0 \qquad (2)$$

where ui is velocity, σij is pressure, ρ is density. Although there are occasions when we can investigate Stokes flow issues analytically, in the majority of cases, employing current high-speed computers to obtain numerical answers is the only option. In reality, the saddle point problem that results from the composite parts of (1) and (2) makes the equations challenging to solve [5]. The constitutive relationship between stress and strain rate and the partial derivative of velocity in relation to directions is represented by equation (3). is a measure of viscosity that can be used to characterize the rheological characteristics of the actual earth [5].

$$\sigma ij = \mu * \frac{\partial ui}{\partial xj} * \frac{\partial uj}{\partial xi} \qquad (3)$$

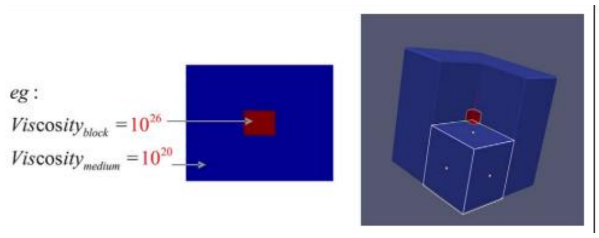


Fig 1: Testing model with strongly variable viscosity.

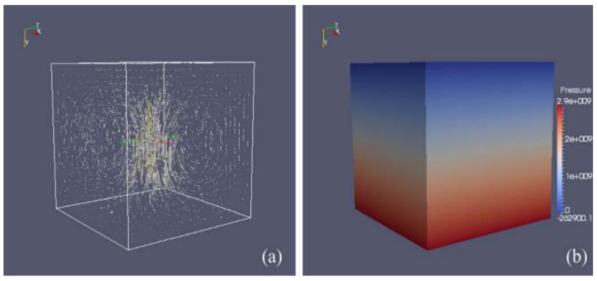


Fig 2: Computed results for viscosity and pressure.

The iterative approach is thought to work well for solving large-scale sparse linear systems. However, the classic iterative approach frequently struggles to obtain convergence solutions because of the poorly-conditioned matrix system [6]. It is desirable to use the multigrid technique to hasten iterative convergence for large classes of mathematical problems with O(log n) time complexity that can answer N unknown questions [7]. In 1964, Bachvalov developed the first multigrid approach for the conventional five-point finite difference approach to the Possion equation [8]. Although it is not difficult to comprehend what multigrid means, because it lacked a solid theoretical foundation, it had not been frequently employed in practise. A growing number of supercomputing scientists have been conducting in-depth study on the multigrid approach. Longer wavelength residuals can decay more quickly utilising the multigrid method than they can using the standard method [7]. In other words, iterative functions operate on grids of varying resolutions to allow iterative information to spread more quickly than if they exclusively operated on grids of the highest resolution. In order to solve the variable-viscosity problem, the multigrid approach is also utilised to simulate mangle convection. In this study, we employ a sinking model that simulates the sinking of a block with high viscosity (and higher density) into a medium section with low viscosity. The viscosity differential between the block component and the medium part is set at 106 in this instance. Every velocity boundary is configured with a free slip boundary [6].

III. IMPLEMENTATION

1. CPU VERSION:

The Matlab routines on the single CPU version have already been built using an appropriate V-cycle MG based finite - element approach to solve the Stokes system. It is also the foundation for our work on the GPU version. The simplest MG method, is called the V-cycle (named for the letter "V"). It consists of three operations: a constraint operation that runs straight from the finest level to the coarsest level, an extension of time operation that runs in the opposite direction, and a smoother that runs at various levels. The original Matlab routines make use of GMG and benefit from the cubic model's regularity. To put it another way, GMG intentionally performs some iterations on coarser grids to obtain the adjustments for the unknown factors on finer grids. On various grids, projection is used to acquire the residuals and corrections, with limitation serving as the basis for computing the residuals and prolongation serving as the basis for computing the corrections. The smoother must be applicable for the Stokes system to handle pressure as it is not included in the continuous equations. A computer compressibility technique is used to explicitly calculate the pressure, updating the pressure using the estimated residuals of the conservation equations at each iteration. Additionally, we used staggered schemes to prevent the decoupling of the odd-even problem, which, as we have already mentioned, also satisfies the LBB condition. For the variable viscosity situation, the conservative finite difference method is used, which satisfies the requirement of stress conservation between the nodal sites on the three-dimensional staggered grids.

The large contrast of coefficients hinders the convergence rate, notably for the significantly changing viscosity problem
for the first round of iterations. We used the strategy to succeed the initial estimation of the unknown factors with an increasing. It can be overcome via computational viscosity contrast, which is designated as a method continuation. At the start of the viscosity is scaled to create a low-viscosity contrast in the V-cycle. The computational viscosity increased after a few repetitions. Once the problem's original viscosity values are restored, contrast steadily rises.

The cautious finite differences are used for the discrete equations to prevent unusual decoupling here between pressure and velocity. For three-dimensional staggered grids, this approach can meet the maintenance of stresses across nodal locations. We use an example to illustrate the specifics of a discrete continuous equation and a discrete x-direction Stokes equation using a stencil of staggered grids, and to solve this coupled system using Taras's[1] original Matlab-based Multigrid approach codes. The smoother, limitation, and prolongation routines almost entirely consume the time when we look

[1] http://www.cambridge.org/gb/knowledge/isbn/item2709959

at how long each component of Taras' original Matlab code takes.

There are two interpolation processes in the V-cycle multigrid method: restriction and prolongation. In reality, we used a method called the multi-multigrid approach to hasten the convergence for significantly changing viscosity. In order to execute the iterations with varying viscosities on each level for both the restriction and extension procedures, the viscosity is rescaled with a progressive rise. We combine the results with the calculated adjustments at the finest grid at the conclusion of the V-cycle (figure 2) and repeat the iterations on the finest grid for each cycle. We will employ Matlab's parallel computing toolbox, which supports GPU computation with script language, as we previously said. The regular CUDA and our solution are exactly the same for developing CUDA kernel routines. Functions must be rewritten using CUDA codes and compiled into PTX files rather than binary files. Using the Kernel Object, which we can set the size of as usual, Matlab may call the PTX codes.

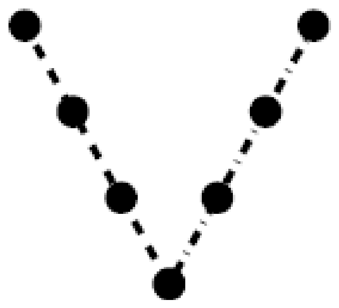
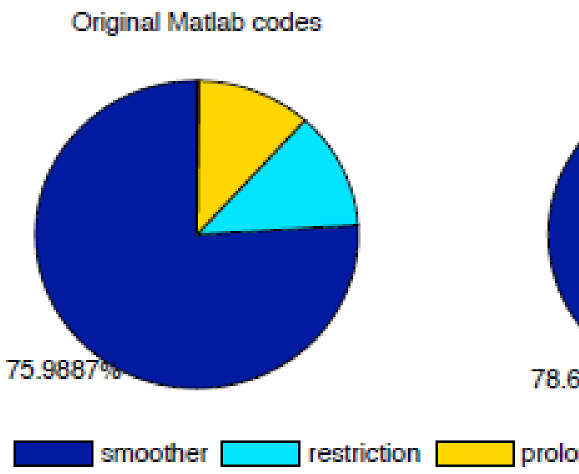


Fig 3: V-cycle multigrid

We will use the Red-Black Gauss-Seidel (RBGS) method for GPU codes in order to prevent the fragmented processes running on GPU (figure 3). The RBGS uses the red and black colours to divide the Gauss-Seidel iteration split two halves so that each point can always use the most recent information surrounding it. Boundary points should be established before the CUDA kernels in order to reduce logic functions (there is In order to condense boolean logic (there is only single logic unit on SM, therefore the logic processes can execute 16 times (half wrap) while producing any results), boundary points should be formed before the CUDA kernels. The entire workflow can be characterised as an algorithm. We built a series of macro to transfer value as follows since the index systems in Matlab and C differ in that the former uses column-major indexing and the latter uses row-major indexing.



2. GI

We followi restrict share of the time spent at each individual stage of the original Matlab scripts, according to an examination of the time spent at each stage. Most of the reusable functions were initially translated into C codes so that Matlab could call them using mexfuntion in order to simulate what may occur with a true compiled language. Using an Intel i7 CPU, Table 1 compares the amount of time that Matlab and its calling C routines take to run.

Table 1 shows that the performance of original Matlab routines can be accelerated by simply rewriting the majority of reusable functions in the C language, including smoother, limitation, and prolongation components, however this is not yet fast enough. The smoother consumes the majority of the time in both the original Matlab routines and their calling C codes, indicating that it needs to be optimised. Under the SIMT model, the sequential inner cycle on a GPU can be transformed into parallel threads across all grid points.

This SIMT model can be used in our implementation with ease for boundary conditions, restriction, and prolongation. The original CPU-based Matlab programmes use Gauss-Seidel iterations for the smoother, which rely on recently updated neighbourhood nodes for computation.

Since all of the threads are operating concurrently and haphazardly, it is challenging to multi-thread these highly interdependent algorithms on a GPU. In our application, the RBGS technique is used to get around this obstacle during GPU implementation. The RBGS divides the Gauss-Seidel iteration into two sections by the colours red and black, enabling the computing point to continue using the most recent data connected with it.

Fig 4: Time

IV. PERFORMANCE ANALYSIS

We shall then discuss our implementation's workflow. As previously indicated, the residuals of the continuity equation are used to update the pressure, and in order to do so, an initial smoothing loop must be used to calculate the residuals at the lowest level. Setting boundaries and executing the kernels of the red and black hues are two of the three main processes that take place throughout the smoother's inner iteration. They are all powered by a GPU. Activating reduce the number of logic operations, the values on the border points must be enforced prior to CUDA kernels. The main V-cycle loop (outer iteration), which includes smoother along with the restriction and extension operations, is then executed until the tolerance requirement is met. The Matlab codes using CUDA run significantly quicker than the Matlab routines already updated with C. On a simple system, using GPU at the same time can significantly boost resolution while still running Matlab routines in a manageable amount of time. We used a tolerance of 105 for the average residual in this calculation. Here, double precision is employed. When employing single precision, residual appears to be tough to converge.

On a single Nvidia Geforce gtx, an Intel i7 3.04 GHz processor, and 12GB of RAM, our simulation was executed. Since the underlying Matlab code is so sluggish, simulating complicated issues is challenging. However, the Gpus is only suitable for tasks that need many threads and high time efficiency. For large-scale applications, the efficiency of original Matlab methods and the Fourier with CUDA algorithms cannot be compared. Therefore, simply comparing the Matlab invoking C functions in the codes. For the V-cycle multigrid approach, we created grids with 6 levels and set the iterations for each level to 10, 20, 40, 80, 160, and 320, respectively. For GPU scripts, the smoother still consumes the majority of the running time, so even a little increase in smoother performance can have a significant positive impact on the entire code. We compiled the smoother function's time requirements for one step across several levels. The use of RBGS smoothers results in fewer iterations, which suggests that the cycle as a whole benefits from the GPU-based RBGS smoothers as much as the smoothers themselves. The 256*128*128 variant recorded a speedup of increase to 13.5 times. It becomes difficult to measure time consumption without GPU computing if the resolution exceeds 256*256*128, indicating that GPU can be utilised as an enhancer to increase the resolution of Matlab scripts on a simple device at a manageable level of time consumption.

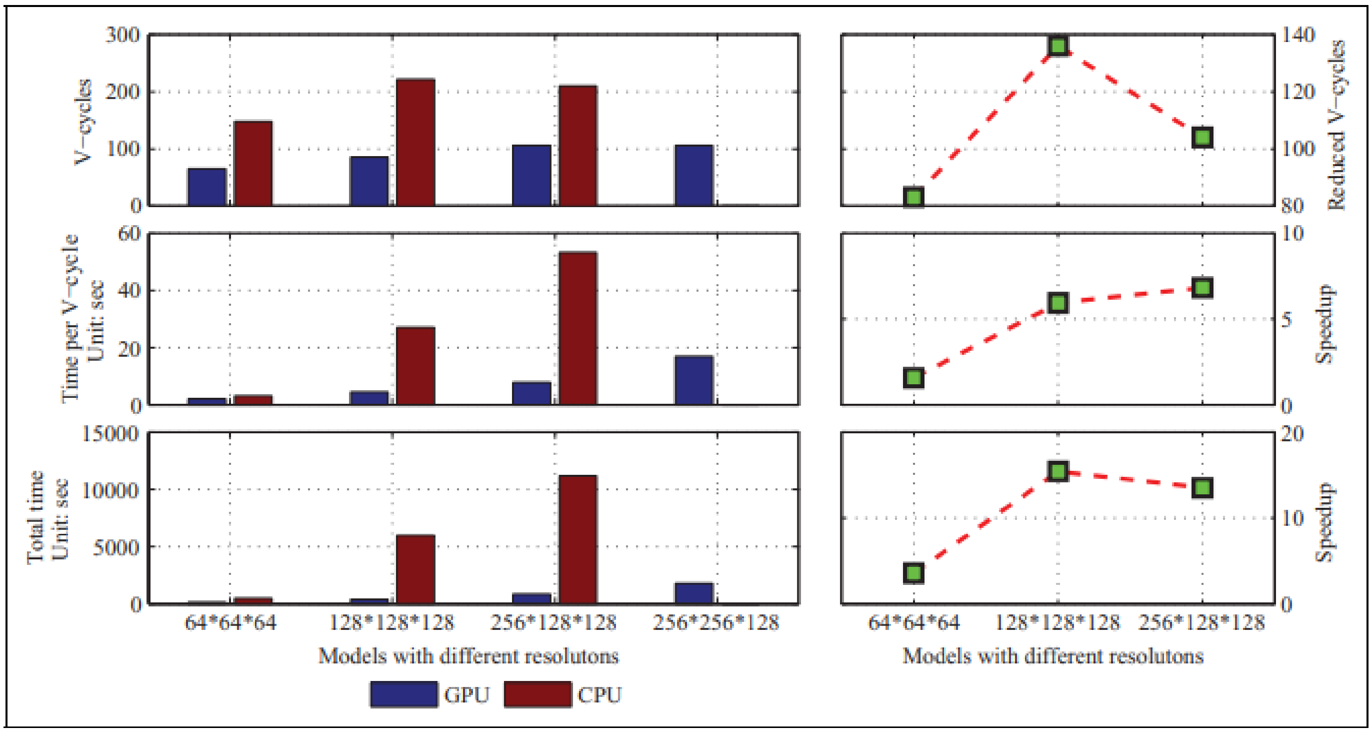


Fig 5: Iteration and time for different models.

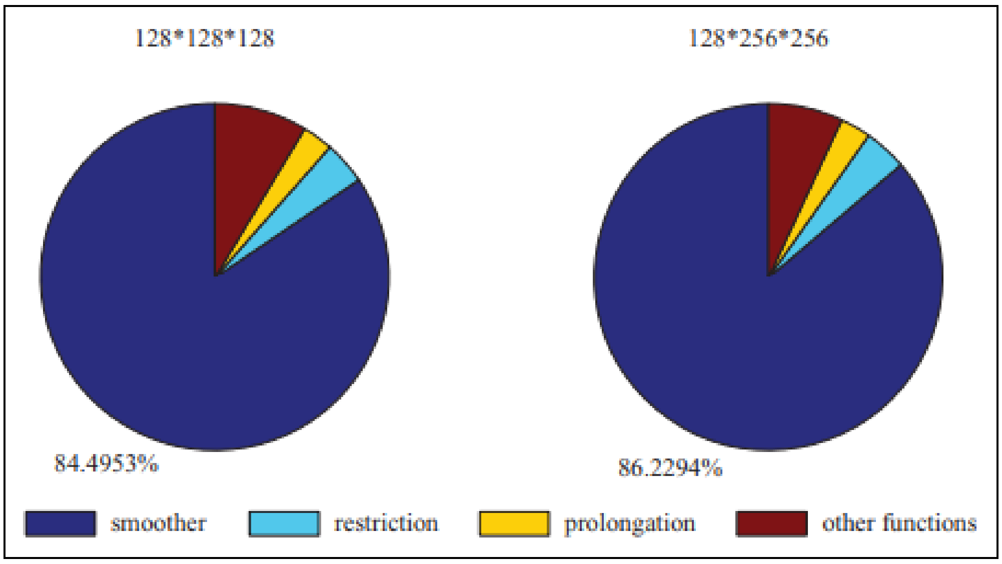


Fig 6: Time consumption analysis for GPU version

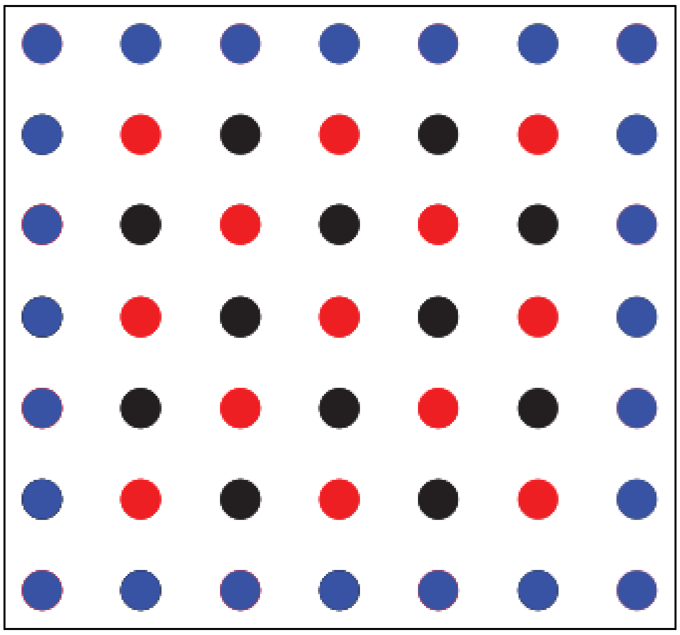


Fig 6: RED BLACK model

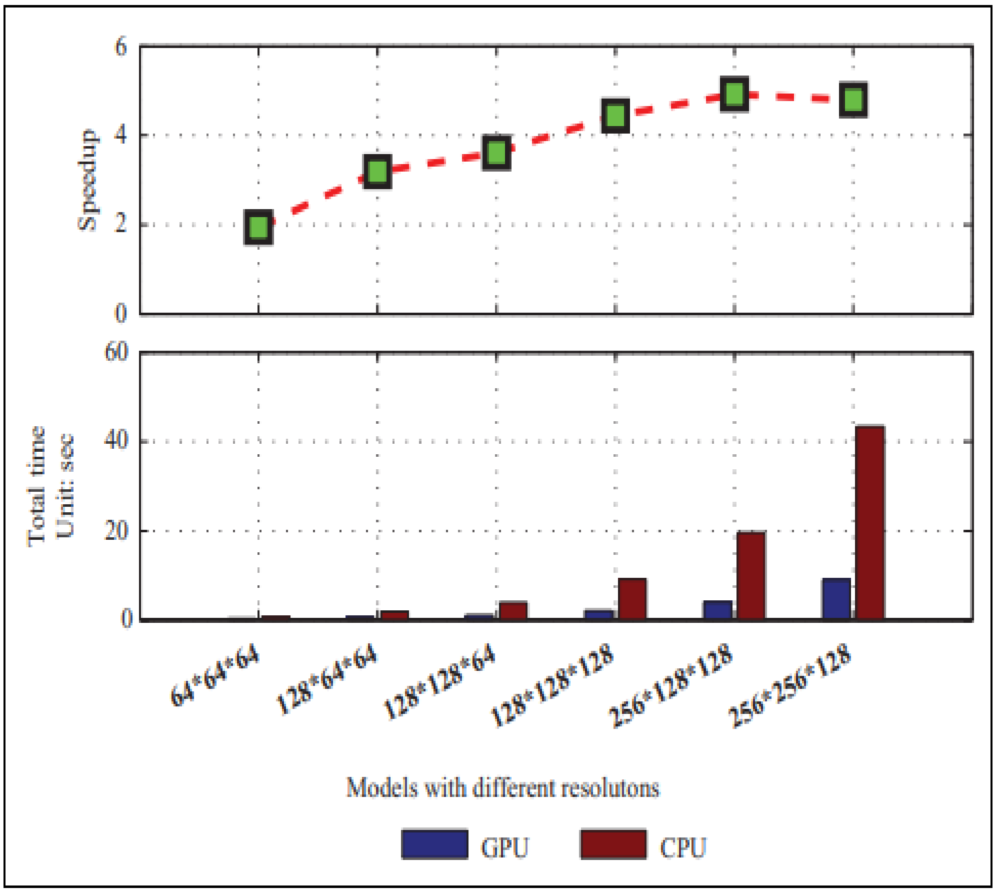


Fig 7: Time and speedup for different smoothers.

From a time-consuming perspective, it appears that performance progress does not follow the scaling law. First, when the model resolution exceeds 128*128*128 the lower V-cycles have no effect on speedup. The algorithm itself could be at blame for this. The most likely explanation is that the long-wavelength iterative information propagates more quickly on coarse grids than on small grids, making it simpler to influence the convergence rate given the low model resolution. Furthermore, the 128*128*128 model is where the speedup increases stops. The performance of the smoother is still being improved after the 128*128*128 model, though. We compiled the smoother's time usage in order to assess how well the GPU-based smoother performed at various resolutions. Up until a resolution of 256*256*128, it appeared that the speedup increased with increased resolution, presumably because to the GPU catch size restriction. To put it another way, a huge problem requires an even more powerful multi-GPU system to solve because the complexity of the problem limits the possible performance of a single GPU card. Evidently, the smoother continues to use the majority of GPU processing time, while limitation and prolonging only take up a little portion of that. The fact that the GPU codes for other portions of the GPU are more efficient than the smoother codes may be the explanation for why GPU's smoother takes a higher percentage. We are unable to optimise the smoother because its implementation is more difficult.

Table 1: Iterations and time of different models.

Bits	MATLAB (V cycle)	MATLAB (Time)	MATLAB with C (V cycle)	MATLAB with C (Time)
64x64x64	65	138s	149	155s
128x128x128	81	378s	234	582s
256x128x128	100	876s	205	1100s
256x256x128	104	1793s	-	-

V. CONCLUSION AND RECOMMENDATIONS

The GPU architecture is appropriate for the multigrid Stokes flow problem numerical simulation method. On a GPU, a simultaneous Red-Black Gauss-Seidel smoother can speed up computation. We can quickly develop the combination computing of script languages and CUDA C by using the new version of Matlab's parallel computing toolbox. In addition to multigrid, the Krylov subspace approach is frequently employed to accelerate iteration. There are already some Krylov subspace based iterative solutions like GMRES, CG, and GPU Powered preconditioned iterative linear solvers. However, because multigrid may quickly decay the residual of a long wavelength, it is frequently used as a preconditioner of Krylov subspace. Above all, adopting some hybrid GPUCPU approaches, including combining MPI with CUDA for greater resolution improvement

ACKNOWLEMENTS

REFERENCES

[1] C. Auth and H. Harder, "Multigrid solution of convection problems with strongly variable viscosity," *Geophys. J. Int.*, vol. 137, no. 3, pp. 793–804, 1999.*)*

[2] N. S. Bakhvalov, "On the convergence of a relaxation method with natural constraints on the elliptic operator," *U.S.S.R. comput. math. math. phys.*, vol. 6, no. 5, pp. 101–135, 1966.

[3] Y. Deubelbeiss and B. J. P. Kaus, "Comparison of Eulerian and Lagrangian numerical techniques for the Stokes equations in the presence of strongly varying viscosity," *Phys. Earth Planet. Inter.*, vol. 171, no. 1–4, pp. 92–111, 2008

[4] Design of 2D numerical geodynamic models," in *Introduction to Numerical Geodynamic Modelling*, Cambridge University Press, 2019, pp. 369–405.R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press.

[5] W. Hackbusch, "On the multi-grid method applied to difference equations," *Computing*, vol. 20, no. 4, pp. 291–306, 1978.

[6] L. Moresi, S. Zhong, and M. Gurnis, "The accuracy of finite element solutions of Stokes's flow with strongly varying viscosity," *Phys. Earth Planet. Inter.*, vol. 97, no. 1–4, pp. 83–94, 1996.

[7] S. C. Brenner, H. Li, and L.-Y. Sung, "Multigrid methods for saddle point problems: Stokes and Lamé systems," *Numer. Math. (Heidelb.)*, vol. 128, no. 2, pp. 193–216, 2014.