Oral presentation | Numerical methods
Numerical methods-III
Wed. Jul 17, 2024 10:45 AM - 12:45 PM Room A

[7-A-03] Parallel Multigrid for Reynolds Averaged Navier Stokes on Block Structured Cartesian Grids

*Ankith Anil Das[1], Nicholas Williamson[1], Steven Armfield[1] (1. University of Sydney)

Keywords: Parallel Multigrid, Full Approximation Scheme, AMReX

**Twelfth International Conference on
Computational Fluid Dynamics (ICCFD12),
Kobe, Japan, July 14-19, 2024**

Parallel Multigrid for Reynolds-Averaged Navier-Stokes Equations on Block Structured Cartesian Grids

Ankith A. Das*, N. Williamson*, S.W. Armfield*
Corresponding author: ankith.anildas@sydney.edu.au
*School of Aerospace, Mechanical and Mechatronic Engineering, University of Sydney, Australia

Abstract:
This extended abstract presents preliminary results for a parallel, multigrid-accelerated solver developed for solving the Reynolds-Averaged Navier-Stokes (RANS) equations on block-structured Cartesian grids. Utilizing the cut-cell method to handle complex geometries, the solver is optimized for efficient execution on modern heterogeneous computing architectures with the help of AMReX framework. Validation tests for lid-driven cavity flow, with and without an embedded sphere, demonstrated second-order accuracy. The solver shows excellent multigrid performance and parallel scaling on CPUs, with ongoing optimization efforts to improve GPU utilization. Future work will focus on dynamic load balancing, integration of turbulence models with wall functions, and implementation of adaptive mesh refinement.

Keywords: Multigrid, Cut-Cell, Parallel Computing, Computational Fluid Dynamics.

1 Introduction

Urban areas currently house approximately 55% of the global population, a figure projected to rise to 68% by 2050 [1]. As this shift continues, understanding the intricate processes governing air quality and local weather becomes increasingly important. High population density and limited evacuation options in urban areas heighten the risks of human exposure to airborne contaminants. Consequently, ensuring security and rapid response capabilities for accidental or intentional releases of chemical, biological, or radiological substances is a pressing challenge in these densely populated areas. In the event of such an emergency, the application of an urban dispersion model becomes crucial for emergency management protocols and decision-making frameworks [2]. Such a system must possess short latency times to enable swift actions.

Computational Fluid Dynamics (CFD) models can offer detailed insights into flow and dispersion processes in urban environments [3]. However, traditional CFD models are often computationally expensive for dispersion modeling in urban areas, where rapid turnaround times are essential. Fast models are essential for applications like emergency response, vulnerability assessments, and optimal sensor placement [4].

Achieving fast physics-based modeling requires leveraging both fast numerical methods and efficient parallel computing. Consequently, the next generation of urban dispersion modeling must integrate both computational efficiency and high-performance computing capabilities.

Multigrid schemes are a widely recognised approach used for accelerating convergence of iterative methods. It capitalizes on the fact that high-frequency components in the solution-error vector decay much faster than low-frequency components. By treating low-frequency error components on a fine mesh as high-frequency errors on a coarse mesh, multigrid can efficiently remove these errors, leading to faster convergence. Recent work by Ryan et al. [5] and Lien et al. [6] have demonstrated the effectiveness of multigrid in the context of urban dispersion problems.

Our study focuses on leveraging multigrid algorithms on modern heterogeneous hardware to tackle large-scale urban dispersion problems. Developed in C++ using the AMReX framework [7], our solver aims to combine the multigrid method with automatic mesh generation and refinement, embedded geometry via the cut-cell method, and integrate turbulence and thermal models with wall treatment. Our goal is to accelerate convergence using parallel heterogeneous hardware. This extended abstract presents our initial findings on incorporating the cut-cell method and utilizing parallel computing while using multigrid.

2 Numerical Method

The steady-state, incompressible Reynolds-averaged Navier-Stokes equations are

$$\frac{\partial(\rho\bar{u}_j)}{\partial x_j} = 0, \tag{1}$$

$$\frac{\partial(\rho\bar{u}_i\bar{u}_j)}{\partial x_j} = -\frac{\partial\bar{p}}{\partial x_j} + \frac{\partial}{\partial x_j}\left[\mu\left(\frac{\partial\bar{u}_i}{\partial x_j} + \frac{\partial\bar{u}_j}{\partial x_i}\right) - \rho\overline{u_i'u_j'}\right], \tag{2}$$

where $\bar{u}_i$ are the Cartesian components of the mean velocity vector, $\bar{p}$ is the mean pressure, ρ is density, μ is dynamic viscosity, and $\overline{u_i'u_j'}$ are the mean turbulent Reynolds stresses. The equations are discretized in space using a finite volume formulation with a collocated arrangement of variables. Pressure-velocity coupling is achieved through the SIMPLE scheme. Second-order central discretization scheme is used for diffusion terms, pressure gradient source term in the momentum equations, and the pressure correction equations. Convective fluxes through the faces are evaluated using a first-order upwind scheme, with higher-order of accuracy achieved using deferred correction. More detailed information regarding the discretization process and the SIMPLE algorithm can be found in the book by Ferziger and Perić [8]. For simplifying notation, the over-bar on the variables will be omitted henceforth.

2.1 Cut-Cell Discretization

Special numerical treatment is required for grid cells next to an embedded boundary. These cells are truncated to fit the boundary, as shown in Figure 1. The truncated cells require modifications to the finite volume discretization to account for the presence of a rigid boundary. The new discretization is based on the geometry of the irregular control volumes, which are characterized by integral quantities, also referred to as embedded fractions. The discretization on these partial cells are based on these embedded fractions. AMReX provides helpful data structures for accessing these embedded fractions, which are precomputed for each multigrid level. This information includes cell volume fraction, face area fractions, cell centroid, face centroid, and connectivity information between neighboring cells. Additional information, such as the surface normal and area of the cut boundary, can be easily computed from these quantities [9, 10].

The current implementation only supports a single cut boundary feature per cell. While this limitation can be addressed using multi-cut cell methods, as outlined in [11], this is currently outside the scope of this study. More details regarding the computation of integral fractions can be found in [9, 10].

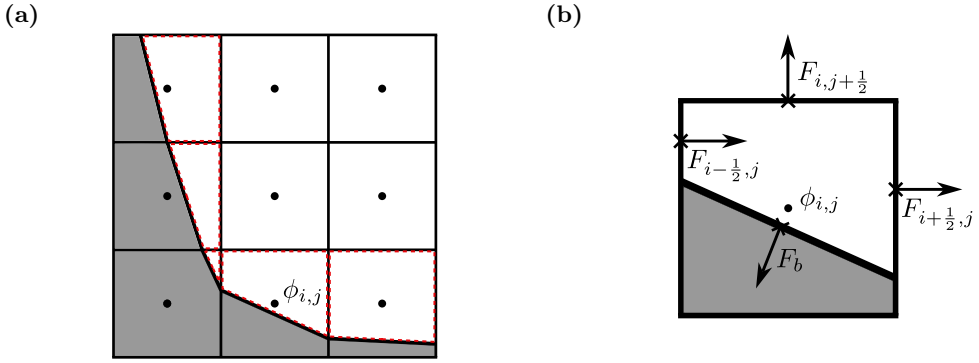

Figure 1: **(a)** Diagram shows truncated cells next to an embedded boundary, highlighted with red borders. **(b)** illustrates the control volume and fluxes which are centered on the face-centroid. Here F_b denote the centered flux at the cut-boundary.

2.1.1 Divergence operator

The finite volume approximation of the divergence of a flux F in a cut-cell is given by

$$\nabla \cdot F \approx \frac{1}{\Delta^D\mathcal{V}} \sum_{f=1}^{N_f} (F_f \cdot n_f)\, A_f, \tag{3}$$

where Δ is the grid spacing, D is the number of space dimensions (2 or 3), $\mathcal{V}$ is the volume fraction of the cut-cell, n_f is the outward pointing unit normal, and F_f and A_f denote the fluxes and effective areas of the faces as indicated in Figure 1 respectively. The discrete fluxes are located at the centroid of the faces. In the context of incompressible Navier-Stokes, these discrete fluxes F_f correspond to the non-linear advection flux and viscous diffusive flux. The following sections outlines the computation of these fluxes.

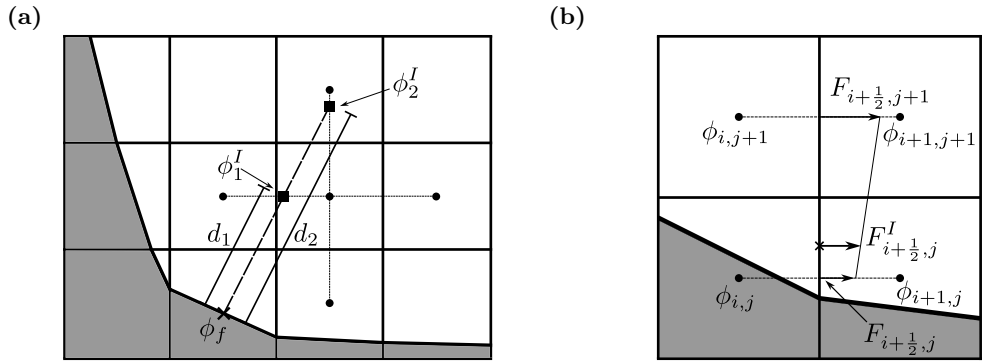

Figure 2: Graphical representation of the methodology for computing **(a)** boundary gradient and **(b)** face fluxes in a cut-cell. Here ■ indicates the interpolation points used for constructing the quadratic profile, ● are the active cell points used for the linear interpolation, and × is the location of the face-centroid.

2.1.2 Computation of cut-face gradients

At cut-faces, gradients are computed at the cell centroid using the method outlined in Johansen and Colella [12]. In this approach, the solutions is thought to be located at the cell-center of the original uncut cell, not at the cell-centroid. The diffusive and convective gradients are normally computed at the midpoint of the uncut face. Then the face gradient is computed using linear interpolation (bilinear in 3D), at the face centroid, using the face gradients of the neighbouring faces located in the direction orthogonal to the cut face normal. The figure illustrates this interpolation process for a 2D cut-cell. It should be noted that deferred correction is not applied on truncated cells.

2.1.3 Computation of cut-boundary gradients

For computing the cut-boundary gradients, two cases arises depending on the boundary conditions. Neumann boundary conditions are trivially imposed. For Dirichlet boundary condition, a quadratic polynomial is fitted in the normal direction and the flux is computed by taking the derivative at the face [12]. This scheme in 2D outlined as follows and also illustrated in the Figure 2

$$F_b = \frac{1}{d_2 - d_1}\left((\phi_f - \phi_1^I)\frac{d_2}{d_1} - (\phi_f - \phi_2^I)\frac{d_1}{d_2}\right). \tag{4}$$

Here, ϕ_f is the Dirichlet boundary value on the cut boundary, ϕ_1^I and ϕ_2^I are computed using bilinear interpolation at distances d_1 and d_2 from the boundary respectively. These interpolation points are located on lines joining the cell-centers.

2.1.4 Computation of cell-centroid gradient

The current implementation only supports cell-centroid gradients for irregular cells with Neumann boundary conditions. These gradients are used to compute pressure source terms in the momentum equations. Using the standard Green-Gauss gradient, given by

$$\nabla\phi = \frac{1}{\Delta^D \mathcal{V}} \sum_{f=1}^{N_f} \phi_f n_f A_f, \tag{5}$$

where ϕ_f is the average face-centroid value, leads to numerical instabilities as the gradient can become singular as $\mathcal{V} \to 0$. To avoid this issue, the gradients are instead computed using the following methods.

Re-centering: In this approach, the approximate gradient is calculated at the cell-center of the original uncut cell. The cell-center gradient is computed by taking the average of the face-center gradients. Then, the cell-centroid value is obtained by linear interpolation (bilinear in 3D) using the cell-center gradients of the neighboring cells. Figure 3 illustrates the scheme for a cut-cell in 2D.

Least squares: In this approach, the gradients at the cut-cell centroid are obtained by a least squares formulation, which results in a system of linear equations at each cut-cell centroid:

$$A_{LS}X = b_{LS}(\phi_{nb}), \tag{6}$$

where A_{LS} is a 3×3 matrix dependent only on the distances between cell-centroids, and X is the solution vector that contains the cell-centroid gradient. The neighborhood cell-center values used in the least squares minimization are indicated in Figure 3. Here, the fluid region for minimization is selected by centering a 3^D cell region at the cut-cell and then shifting it to the corner based on the direction of the cut-boundary surface normal. Note that the values of ϕ_{nb} are assumed to be located at the cell-center of the uncut cell. The above linear equation is solved using QR decomposition outlined in [13]. Lastly, the stencils for gradients are precomputed at the start of the simulation.

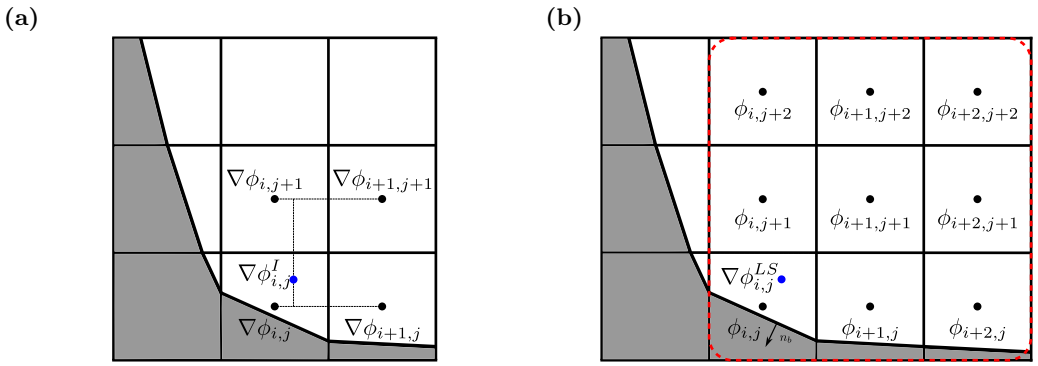

Figure 3: Graphical representation of the methodology used for computing cell-centroid gradient. Here **(a)** illustrates the re-centering approach, while **(b)** highlights the cell values used for least-squares gradient. The ● indicates the location of the cell-centroid gradient.

2.2 Multigrid

The steady state solver uses Full Approximation Scheme (FAS) multigrid, which accelerates convergence by efficiently eliminating errors at various scales. The SIMPLE algorithm is used for smoothing at each coarsened multigrid level.

Our implementation of FAS multigrid is standard, whose details are available in [14]. On partial cells, the coarse cell volume is the sum of its corresponding 4 (8 in 3D) fine-cell volumes. Grid coarsening can introduce cells with multiple cuts, a scenario that is currently not supported. Furthermore, the quadratic stencil used for interpolation imposes additional constraints, as it must not extend into cells with zero volume. These restriction can be violated on very coarse grids, thereby determining the number of allowable coarse grid levels.

The residuals and variables are restricted to the coarse grid using volume-weighted averaging, ensuring that source terms are weighted appropriately. The coarse grid equations are discretized using the same approach as the fine grid. The solution on the coarsest grid is solved to a relative residual of 10^{-1}. Testing indicated that using smaller values did not result in improved convergence. The coarse grid corrections are moved up the grid hierarchy using linear interpolation, with piece-wise constant injection used for partial cells. If the coarse grid solver fails to converge or diverges, the solution in the coarse grid is discarded, and the cycle continues to the next fine level. Additionally, if a coarse level diverges in multiple multigrid cycles, that level is eliminated.

3 Parallelism

The solver leverages AMReX's design architecture to achieve parallelism. The computational domain is divided into rectangular boxes, which are distributed to processors using an algorithm based on space-

filling curves. Stencils are implemented for cell-wise, face-wise, or node-wise operations, and can be compiled to run on either CPUs or GPUs [10, 7].

AMReX employs an MPI+X strategy for parallelism, where X represents additional parallelization techniques such as tiling. On CPUs, thread-level parallelism is achieved using OpenMP. For GPUs, AMReX utilizes either CUDA, HIP, or SYCL libraries to perform stencil operations using GPU threads. This versatile approach ensures efficient use of computational resources across different hardware architectures. More details regarding the performance portability of AMReX can be found here [10, 7].

Currently, our multigrid implementation operates with the constraint that a coarse grid level is handled by the same processors responsible for the corresponding fine grid level. This approach simplifies restriction and prolongation operations, avoiding the need for inter-processor communication. However, we aim to relax this constraint in the future. It is important to note that our domain decomposition strategy, which divides the grid into rectangular boxes assigned to processors, cannot guarantee perfectly even load distribution for all processor counts. This inherent limitation places some restrictions on the number of MPI processes that can be used efficiently.

4 Results & Performance

4.1 Order of accuracy

4.1.1 3D Lid driven cavity

The spatial order of accuracy of the scheme was tested for a cubic lid-driven cavity (LDC) flow. The cubic cavity measures 1 unit length, with the lid moving at a velocity of 1 unit per time in the x direction. The kinematic viscosity is defined as $1/\mathrm{Re}$, where Re is the Reynolds number based on the cavity length and the lid velocity. The flow was solved on uniform grids starting from 16^3 to 256^3, where the number of cells was successively doubled in each coordinate direction, for Reynolds numbers 10 and 1000. The problems were solved to a residual of 10^{-11} using the multigrid method.

The convergence of the solution was evaluated by calculating the L_2 and L_∞ norms of the relative differences between solutions on successive grids. The norms for velocity components and pressure are plotted against the cell size as shown in Figure 4. Convergence in the asymptotic region of the L_2 and L_∞ norm curves for the velocity is second-order, showing that the scheme is globally second-order accurate.

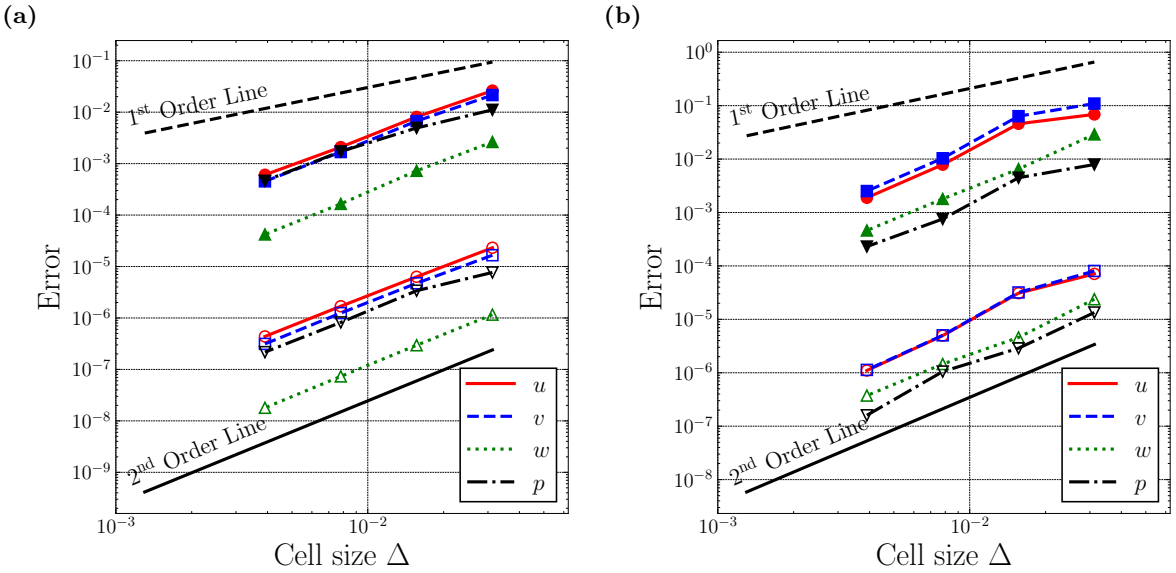

Figure 4: Convergence test for a cubic lid-driven cavity at different Reynolds numbers. **(a)** corresponds to Re = 10 and **(b)** to Re = 1000. The L_2 and L_∞ norms are represented by hollow (○, □, △, ▽) and filled markers (●, ■, ▲, ▼), respectively.

4.1.2 3D Lid driven cavity embedded with a sphere

For testing the order of accuracy of the cut-cell discretization, the lid-driven cavity case was modified to include a sphere of diameter 0.4 centered in the domain. To ensure that the norms represent the

errors of the boundary cell discretization, the errors were computed only within a cubic region of size 0.6 surrounding the sphere. Velocity and pressure were sampled on 50^3 uniformly distributed points using cubic spline interpolation. When computing the relative error between successive grids, the interpolating points lying on covered and partial cells of the finer grid were removed from both grids. These points were excluded because fine grid cut-cells can extend into coarse grid covered cells, leading to inconsistent results for L_∞ norm. Evaluating the errors at partial cells would require an an exact solution.

Results for Reynolds numbers 10 and 400, where the Reynolds number (Re_D) is based on the lid velocity and sphere diameter, are shown in Figure 5. Convergence in the asymptotic region of the L_2 and L_∞ norm curves for the velocity is second-order, showing that the scheme is globally second-order accurate.

To further validate the results, the solution obtained from our solver is compared with OpenFOAM, a widely used open-source CFD software. The problem was meshed using a block-structured Cartesian grid with inflation layers around the sphere and domain walls to accurately capture boundary layer effects. The final mesh consisted of $\approx 16 \times 10^6$ cells, providing sufficient resolution to ensure accurate results. The convergence criterion for the simulation was set to a residual of 10^{-11}.

Figure 6 shows the difference in velocity between our implementation and OpenFOAM along three lines that touch and intercept the embedded sphere for $\mathrm{Re}_D = 400$. It is clear that the near wall characteristics are well captured by the cut-cell discretization. Our scheme solved the problem on $256^3 \approx 16.7 \times 10^6$ cells.

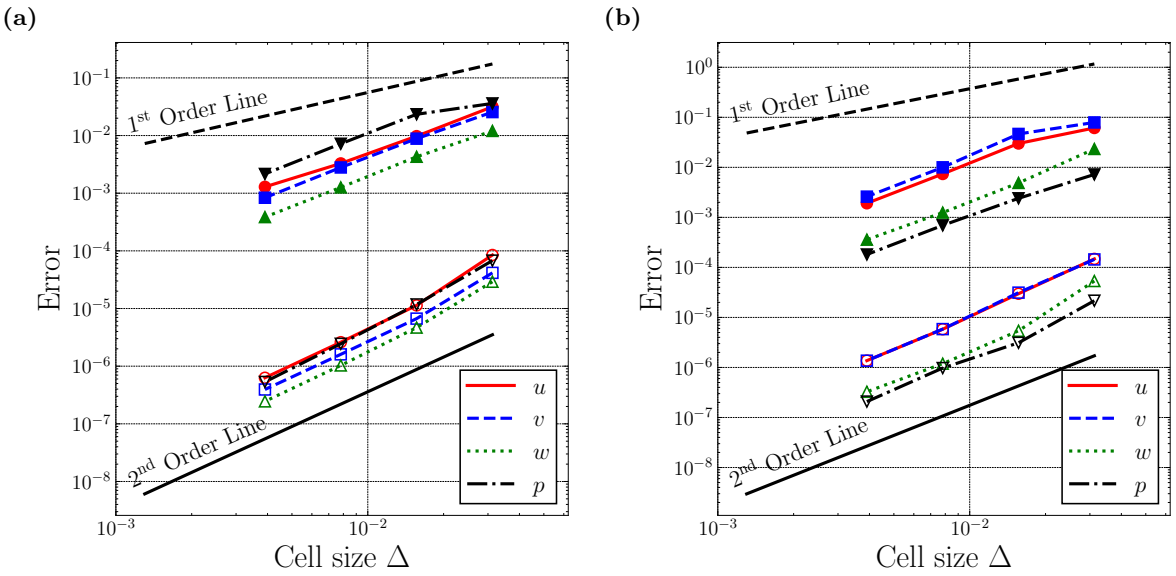

Figure 5: Convergence test for a cubic lid-driven cavity embedded with a sphere at different Reynolds numbers. **(a)** corresponds to $\mathrm{Re}_D = 10$ and **(b)** to $\mathrm{Re}_D = 400$. The L_2 and L_∞ norms are represented by hollow (○, □, △, ▽) and filled markers (●, ■, ▲, ▼), respectively.

4.2 Parallel and Multigrid performance

According to Brandt's golden rule for multigrid [14], an efficient multigrid solver shows linear dependence in computational work, i.e., it exhibits $\mathcal{O}(N)$ scaling in computational work where N is the number of cells. The solver was tested on the LDC (with and without embedded sphere) flow across various Reynolds numbers. The results, illustrated in Figure 7, clearly indicate optimal scaling behavior. The cases were solved to a residual of 10^{-8} using the QUICK scheme with 32 CPU cores. Slight deviations from the optimal trend in solution time were observed, which can be attributed to the variations to the number of multigrid cycles to achieve a converged solution. In cases where additional multigrid cycles were required, the final residual was well below the target residual.

Parallel scaling tests were carried out for the LDC scenario with Reynolds number 500. As depicted in Figure 8, the multigrid implementation showed excellent weak and strong scaling performance. The strong scaling study was conducted with a grid resolution of $512^3 \approx 134 \times 10^6$ cells. The tests were done for 2^n processors, where $n \in [3, 9]$. A slight slowdown was was observed for 2^{10} processors. This drop-off in efficiency was expected since the compute load per processor was too small. It should be noted that these cases result in an uneven distribution of compute load, as the domain is decomposed

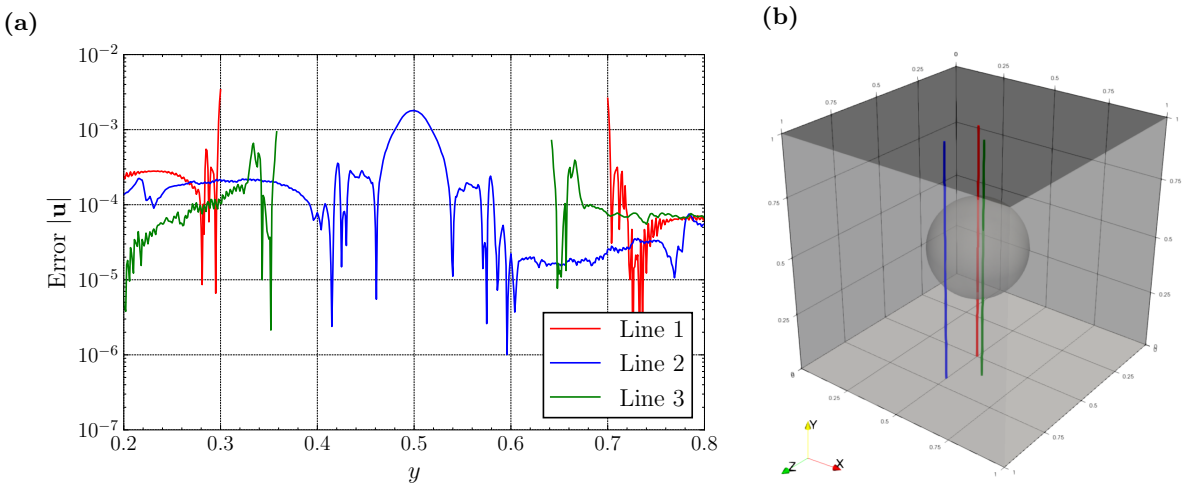

Figure 6: Absolute error in velocity magnitude between our and OpenFOAM solution. The lines span the range $y \in [0.2, 0.8]$. Line 1 is centered at $x = 0.5, z = 0.5$, Line 2 at $x = 0.5, z = 0.7$, and Line 3 at $x = 0.6, z = 0.6$. **(b)** illustrates the locations of these lines within the domain. The errors are computed for $\mathrm{Re}_D = 400$.

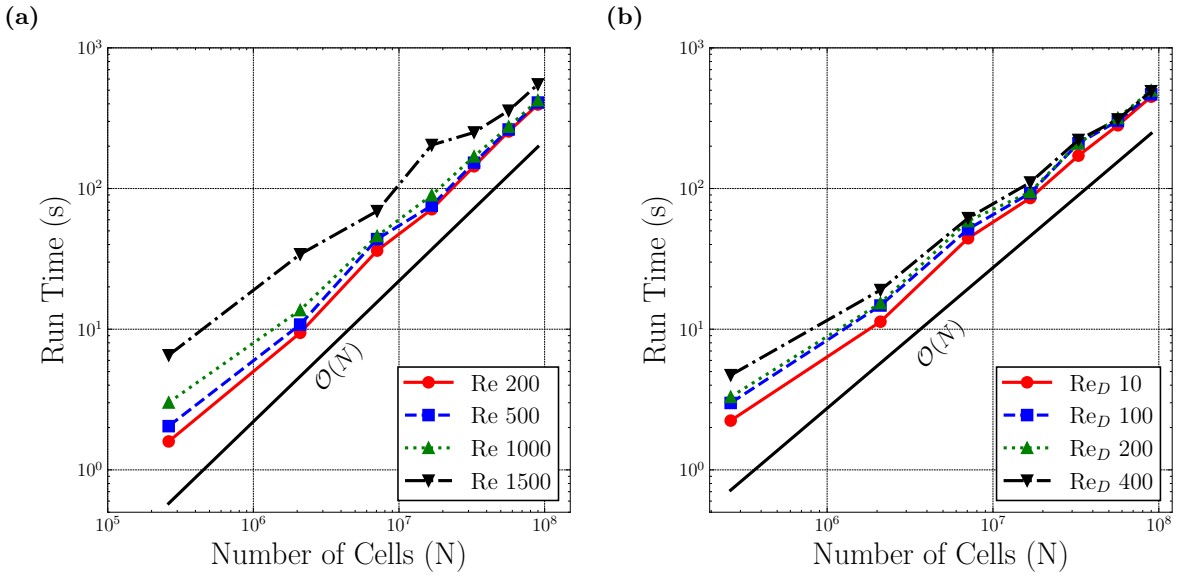

Figure 7: Multigrid scaling with respect to cell count for LDC flow at various Reynolds numbers. **(a)** depicts the standard LDC flow, while **(b)** shows the LDC flow with an embedded sphere of diameter 0.4.

into rectangular boxes which are distributed to processors.

For weak scaling tests, a configuration of $128^3 \approx 2 \times 10^6$ cells per core was employed, with the largest case comprising approximately 2×10^9 cells. The tests were done for n^3 processors, where $n \in [3, 10]$. The base case for the weak scaling was selected such that the subsequent cases involved more that one compute node. These scaling tests were performed on dual 24-core Intel Xeon Platinum 8274 (Cascade Lake) processors per node on the NCI Gadi cluster located at the Australian National University.

Parallel GPU performance was also assessed for the LDC case, although with less promising scaling results. The execution time on multiple GPUs is summarized in Table 1. Although some speedup is observed when using multiple GPUs, the speedup is far from ideal due to two main reasons. First, GPU memory bandwidth is far greater than inter-socket or inter-node communication bandwidth. When using multiple GPUs, it is essential that the problem size is large enough so that the computational workload on each GPU significantly outweighs the communication overhead between GPUs. This is indicated in the execution times, where larger problems generally show better scaling when solved with more GPUs. Second, grid coarsening can lead to problems that are too small to fully utilize the GPU threads. When the grid is coarsened, the number of cells decreases (by a factor of 8 in 3D), resulting in fewer computational tasks. GPUs are designed to handle a large number of parallel tasks efficiently, and when the grid size becomes too small, the overhead of managing GPU threads can outweigh the

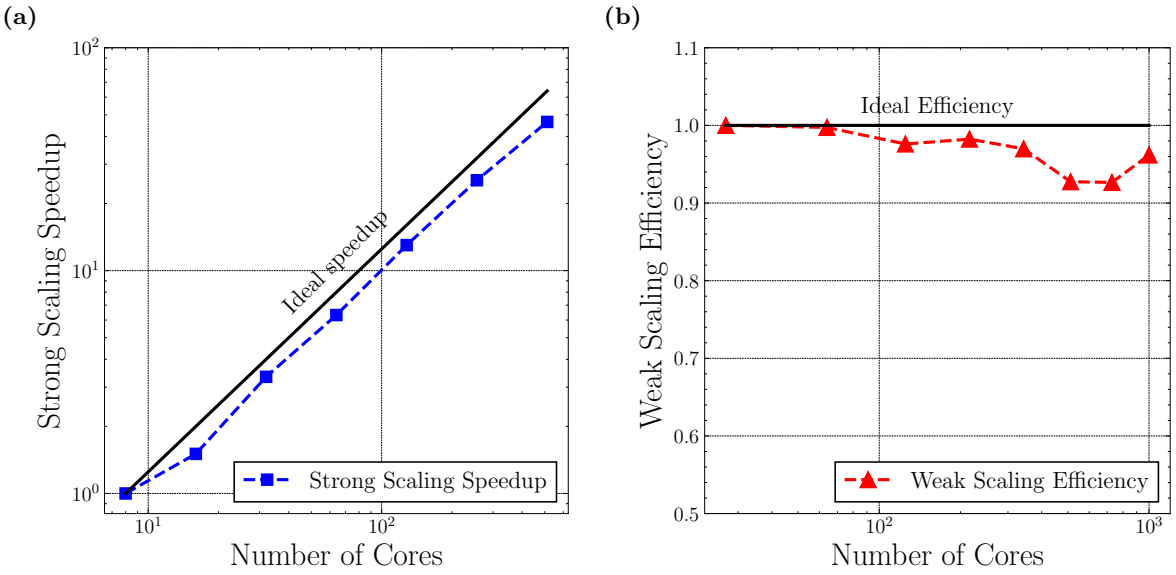

Figure 8: Parallel Multigrid scaling for LDC flow at Re = 500. **(a)** depicts strong scaling, while **(b)** shows weak scaling.

benefits of parallelism. This issue is further exacerbated when the problem is divided across multiple GPUs. Currently, two strategies are being explored to tackle these issues. The first is to agglomerate the coarse grid problem to a single GPU to reduce inter-GPU communication. The second strategy is to solve the coarse grid problem on CPUs instead of GPUs.

The execution time when run on CPUs is also provided in Table 1. While GPUs demonstrate significantly faster execution time when compared to CPUs, drawing meaningful conclusions between the two is challenging due to differences in architecture. This architectural difference means that a direct comparison can be misleading.

These tests were conducted at the Pawsey Setonix super-computing cluster, located in Perth, Western Australia [15]. The GPU tests were conducted on compute nodes equipped with eight AMD Instinct MI250X GPUs, while the CPU test were performed on nodes with dual 2.45 GHz AMD EPYC 7763 Milan 64-Core CPUs.

Table 1: Execution time for the LDC problem for Re = 500 with varying grid sizes and number of GPUs. Speedup values, shown in parentheses, are calculated relative to the preceding column, indicating the performance gains achieved by increasing the number of GPUs. Entries marked – indicate cases where the problem size exceeded GPU memory.

Case	Execution Time (seconds)					
	1 GPU	2 GPUs	4 GPUs	8 GPUs	16 GPUs	256 CPUs
320^3	37.1(1.00×)	33.5(1.10×)	29.5(1.14×)	26.4(1.12×)	32.6(0.81×)	48.83
384^3	42.7(1.00×)	32.6(1.31×)	27.3(1.19×)	23.5(1.16×)	22.3(1.05×)	89.25
512^3	–	58.6(1.00×)	42.2(1.39×)	32.5(1.30×)	27.7(1.17×)	128.8
640^3	–	–	64.8(1.00×)	45.6(1.42×)	36.0(1.26×)	259.4
768^3	–	–	–	69.3(1.00×)	51.2(1.35×)	590.2

5 Conclusions and Future Work

This work represents an ongoing effort to develop a high-performance solver for simulating fluid flow and dispersion in complex urban environments. A parallel, multigrid-accelerated steady-state solver using the cut-cell method has been introduced, demonstrating second-order accuracy and showcasing excellent multigrid and parallel scaling performance on CPUs. While parallel GPU performance shows promise, further optimization is required to fully utilise modern GPUs.

These promising results lay the groundwork for the further development of the solver. Future work focuses on incorporating turbulence models with wall functions, adaptive mesh refinement, thermal model, and enhancing load balancing.

Acknowledgements

This work was supported and funded by the Department of Defence and was supported by DMTC Limited (Australia). The authors have prepared this document in accordance with the intellectual property rights granted by a DMTC Project Agreement.

This research was also supported by the Australian Government's National Collaborative Research Infrastructure Strategy (NCRIS), with the access to computational resources provided by National Computational Infrastructure (NCI) and Pawsey Supercomputing Research Centre through the National Computational Merit Allocation Scheme and by the University of Sydney Informatics Hub.

References

[1] 68% of the world population projected to live in urban areas by 2050, May 2018. Department of Economic and Social Affairs, United Nation.

[2] Denise Hertwig, Lionel Soulhac, Vladimír Fuka, Torsten Auerswald, Matteo Carpentieri, Paul Hayden, Alan Robins, Zheng-Tong Xie, and Omduth Coceal. Evaluation of fast atmospheric dispersion models in a regular street network. *Environmental Fluid Mechanics*, 18(4):1007–1044, August 2018.

[3] Yoshihide Tominaga and Ted Stathopoulos. CFD simulation of near-field pollutant dispersion in the urban environment: A review of current modeling techniques. *Atmospheric Environment*, 79:716–730, November 2013.

[4] Akshay A. Gowardhan, Eric R. Pardyjak, Inanc Senocak, and Michael J. Brown. A CFD-based wind solver for an urban fast response transport and dispersion model. *Environmental Fluid Mechanics*, 11(5):439–464, October 2011.

[5] Sydney D. Ryan, Robert C. Ripley, Fue-Sang Lien, and Fan Zhang. Accelerated convergence for city-scale flow fields using immersed boundaries and coupled multigrid. *Journal of Wind Engineering and Industrial Aerodynamics*, 241:105541, October 2023.

[6] Fue-Sang Lien, Hua Ji, Sydney D. Ryan, Robert C. Ripley, and Fan Zhang. A coupled multigrid solver with wall functions for three-dimensional turbulent flows over urban-like obstacles. *International Journal for Numerical Methods in Fluids*, 95(9):1349–1371, 2023.

[7] Weiqun Zhang, Ann Almgren, Vince Beckner, John Bell, Johannes Blaschke, Cy Chan, Marcus Day, Brian Friesen, Kevin Gott, Daniel Graves, Max Katz, Andrew Myers, Tan Nguyen, Andrew Nonaka, Michele Rosso, Samuel Williams, and Michael Zingale. AMReX: a framework for block-structured adaptive mesh refinement. *Journal of Open Source Software*, 4(37):1370, May 2019.

[8] Joel H. Ferziger and Milovan Perić. *Computational Methods for Fluid Dynamics*. Springer, Berlin, 2nd edition, 1999.

[9] Richard B. Pember, John B. Bell, Phillip Colella, William Y. Curtchfield, and Michael L. Welcome. An Adaptive Cartesian Grid Method for Unsteady Compressible Flow in Irregular Regions. *Journal of Computational Physics*, 120(2):278–304, September 1995.

[10] Weiqun Zhang, Andrew Myers, Kevin Gott, Ann Almgren, and John Bell. AMReX: Block-structured adaptive mesh refinement for multiphysics applications. *The International Journal of High Performance Computing Applications*, 35(6):508–526, November 2021. Publisher: SAGE Publications Ltd STM.

[11] Lennart Schneiders, Claudia Guenther, Jerry H. Grimmen, Matthias H. Meinke, and Wolfgang Schroeder. Sharp resolution of complex moving geometries using a multi-cut-cell viscous flow solver. In *22nd AIAA Computational Fluid Dynamics Conference*, AIAA AVIATION Forum. American Institute of Aeronautics and Astronautics, June 2015.

[12] Hans Johansen and Phillip Colella. A Cartesian Grid Embedded Boundary Method for Poisson's Equation on Irregular Domains. *Journal of Computational Physics*, 147(1):60–85, November 1998.

[13] Jiri Blazek. Chapter 5: Unstructured finite-volume schemes. In Jiri Blazek, editor, *Computational Fluid Dynamics: Principles and Applications (Third Edition)*, pages 121–166. Butterworth-Heinemann, Oxford, third edition edition, 2015.

[14] Achi Brandt. Multi-Level Adaptive Solutions to Boundary-Value Problems. *Mathematics of Computation*, 31(138):333–390, 1977. Publisher: American Mathematical Society.

[15] Pawsey supercomputing research centre, 2023. Setonix Supercomputer. Perth, Western Australia. https://doi.org/10.48569/18sb-8s43.