A high-throughput hybrid task and data parallel Poisson solver for large-scale simulations of incompressible turbulent flows on distributed GPUs

Hadi Zolfaghari[a,*], Dominik Obrist[b]

[a] *Centre for Mathematical Sciences, Department of Applied Mathematics and Theoretical Physics, University of Cambridge, Wilberforce Rd, Cambridge CB3 0WA, U.K.*
[b] *ARTORG Center for Biomedical Engineering Research, University of Bern, 3010 Bern, Switzerland*

Abstract

The solution of the pressure Poisson equation arising in the numerical solution of incompressible Navier–Stokes equations (INSE) is by far the most expensive part of the computational procedure, and often the major restricting factor for parallel implementations. Improvements in iterative linear solvers, e.g. deploying Krylov-based techniques and multigrid preconditioners, have been successfully applied for solving the INSE on CPU-based parallel computers. These numerical schemes, however, do not necessarily perform well on GPUs, mainly due to differences in the hardware architecture. Our previous work using many P100 GPUs of a flagship supercomputer showed that porting a highly optimized MPI-parallel CPU-based INSE solver to GPUs, accelerates significantly the underlying numerical algorithms, while the overall acceleration remains limited (Zolfaghari et al., *Comput. Phys. Commun.*, 244, 132-142, 2019). The performance loss was mainly due to the Poisson solver, particularly the V-cycle geometric multigrid preconditioner. We also observed that the pure compute time for the GPU kernels remained nearly constant as grid size was increased. Motivated by these observations, we present herein an algebraically simpler, yet more advanced parallel implementation for the solution of the Poisson problem on large numbers of distributed GPUs. Data parallelism is achieved by using the classical Jacobi method with successive over-relaxation and an optimized iterative driver routine. Task parallelism is enhanced via minimizing GPU-GPU data exchanges as iterations proceed to reduce the communication overhead. The hybrid parallelism results in nearly 300 times less time-to-solution and thus computational cost (measured in node-hours) for the Poisson problem, compared to our best-case scenario CPU-based parallel implementation which

*Corresponding Author
Email address: hz382@damtp.cam.ac.uk (Hadi Zolfaghari)

uses a preconditioned BiCGstab method. The Poisson solver is then embedded in a flow solver with explicit third-order Runge-Kutta scheme for time-integration, which has been previously ported to GPUs. The flow solver is validated and computationally benchmarked for the transition and decay of the Taylor-Green Vortex at $Re = 1600$ and the flow around a solid sphere at $Re_D = 3700$. Good strong scaling is demonstrated for both benchmarks. Further, nearly 70% lower electrical energy consumption than the CPU implementation is reported for Taylor-Green vortex case. We finally deploy the solver for DNS of systolic flow in a bileaflet mechanical heart valve, and present new insight into the complex laminar-turbulent transition process in this prosthesis.
Keywords: GPU computing, incompressible Navier–Stokes equations, pressure Poisson equation, high-order accurate methods, transitional and turbulent flows, hybrid supercomputing

1. Introduction

The performance of the numerical solution of the incompressible Navier−Stokes equations (INSE) is often restricted by the performance of the pressure Poisson solver. In particular, for direct numerical simulations (DNS), high grid resolution results in a large linear system of equations to be solved at least once per time step. Over the last three decades, there has been notable progress in optimizing the INSE solvers on massive numbers of distributed CPU cores, mainly using the Message Passing Interface (MPI). For CPU-based supercomputing systems, Krylov subspace methods (such BiCGstab or GMRES) together with a preconditioning scheme (such as geometric multigrid with Gauss-Seidel or Jacobi relaxation) result in good CPU-based parallel performance [1, 2]. More recently, general purpose graphics processing units (GPGPUs) have been introduced extensively in flagship supercomputing facilities such as $PizDaint$ (Swiss National Supercomputing Centre), $Titan$ and $Summit$ (Oak Ridge National Laboratory). GPUs operate differently from CPUs on a hardware level: instead of sequential (task parallel) instruction of floating point arithmetic operations, they perform the numerical operations in a data parallel manner. Using the so-called single instruction multiple data (SIMD) standard, GPUs resort to a shared-memory and dedicated streaming processors to perform thousands of threads concurrently. Because of such differences, the numerical methods that have performed favourably on CPU-based parallel computers may not necessarily perform faster on GPUs. This was reflected in our former work [3], where a highly optimized MPI-parallel legacy flow solver was ported to GPUs with only minor changes to the underlying numerical methods. This resulted in performance speedups of factor 20 for isolated numerical routines, for instance, relaxation operators within the multigrid method, or operations outside of the Poisson

solver. However, it resulted in a limited overall speedup for the larger routines, such as the V-cycle multigrid preconditioner. Further, it was shown that for the majority of the GPU routines used in the INSE solver, the pure compute time did not increase significantly by increasing the number of grid points. These observations motivated the development of the present solver, which aims at replacing the existing Poisson solver by numerical routines that are better suited for GPUs.

In [3], we presented the workflow of porting a CPU-based high-order three-dimensional incompressible Navier−Stokes solver to large numbers of NVIDIA P100 GPUs of a Cray XC40/XC50 supercomputer (*Piz Daint*, Swiss National Supercomputing Centre). The CPU-based legacy solver was originally developed for geometric domain decomposition, to run optimally on massively parallel CPU cores within a torus network, using the Message Passing Interface (MPI) [1]. We reprogrammed the computationally intensive parts of the code using CUDA C for GPUs, while keeping the least intensive operations of the code running on CPUs. By benchmarking the individual kernels on NVIDIA P100 GPUs, it was shown that the worst-case scenario double-precision performance (i.e., including all memory operations) was on average 20 times faster than their exact CPU counterparts. The pure compute performance (i.e., without host-device data copies and related host-side memory operations) of the GPU kernels remained on the order of ten microseconds, even when the grid resolution was enhanced by up to 512 times (resolutions from $32 \times 32 \times 32$ to $256 \times 256 \times 256$ per device were examined).

In this work, we replace the preconditioned Krylov-based solver with an iterative solver based on the Jacobi method with successive-over-relaxation. Although this already results in slightly lower time to solution than the preconditioned Krylov-based solver, the kernel implementations (in C language) and MPI communication routines (in FORTRAN90 language) are further enhanced to create an iterative scheme with only minimal data copies as iterations proceed. Further, we use the Fourier analysis of the Jacobi iterative scheme to formulate a grid-size-adaptive termination criterion for the proposed solver, which ensures that total residuals at the termination remain close to that for the preconditioned Krylov-based solver. After integrating the Poisson solver within an incompressible Navier-Stokes solver using an explicit third-order Runge-Kutta scheme which was ported to GPUs as described in [3], the resulting flow solver is validated for two flow problems: the Taylor-Green vortex transition and decay at $Re = 1600$, and turbulent flow around a solid sphere at $Re_D = 3700$. Good agreement with literature data is found for both cases.

Excellent performance speedups are reported for both benchmark cases (up to three orders of magnitude less time to solution compared to the legacy MPI-parallel CPU-based solver running on

the same number of compute nodes). When only one CPU core per node is used for seeding the GPU, it is shown that at least linear strong scaling is achieved, whereas the directly ported GPU-based flow solver [3] and the legacy CPU-based solver [1] both failed to offer linear strong scaling.

Finally, a highly resolved DNS of systolic laminar-turbulent transition past a bileaflet mechanical heart valve is performed. Using roughly 340M grid points resulted in, to our knowledge, by far the best-resolved simulation of heart valve hemodynamics. This grid resolution was required to resolve flow structures near the leaflets which are linked to the impinging leading edge vortex instability [4]. Pilot simulations of this case showed that roughly 1.5 years would be required with the legacy CPU-based solver to simulate 0.2s of the cardiac cycle on 20 compute nodes of the Cray XC40/50 (*Piz Daint*). This prohibitively high cost is probably the reason why earlier simulations of the bileaflet mechanical heart valve flow only used up to 10M grid points [5]. With the present GPU-based solver, this extensive simulation is completed in three days on twenty P100 GPUs (resulting in an approximate speedup of 150 with respect to the legacy CPU-based solver).

The remainder of this paper is organized as follows. In the Section 2, we briefly introduce the governing equations and the numerical procedure leading to the pressure Poisson equation. Section 3 describes the novel Poisson solver routine where the hybrid GPU implementation concerning domain decomposition and data parallelism is emphasized. In Section 4 we validate and assess the parallel performance the GPU-based solver via the problem of transition to turbulence and decay of the Taylor-Green vortex at $Re = 1600$. In Section 5, the proposed solver is validated for the problem of turbulent flow around a solid sphere at $Re_D = 3700$, where parallel performance is also examined. In Section 6, we deploy the solver for direct numerical simulation of systolic laminar-turbulent transition in a bileaflet mechanical heart valve. Finally, we conclude and summarize our major findings and outlook in Section 7.

2. Numerical procedure

2.1. Governing equations

We solve the dimensionless incompressible Navier–Stokes equations

$$\frac{\partial}{\partial t}\begin{bmatrix}\underline{\mathbf{u}}\\0\end{bmatrix}+\begin{bmatrix}-\mathscr{L} & \mathscr{G}\\ \mathscr{D} & 0\end{bmatrix}\begin{bmatrix}\underline{\mathbf{u}}\\p\end{bmatrix}=\begin{bmatrix}\mathscr{N}\,\underline{\mathbf{u}}\\0\end{bmatrix}, \tag{1}$$

where $\underline{\mathbf{u}}$=[u_x,u_y,u_z] and p denote dimensionless velocity and pressure, respectively. Operators $\mathscr{G}$, $\mathscr{D}$, $\mathscr{L}$, and $\mathscr{N}$, represent the differential operators ∇, $\nabla\cdot$, $Re^{-1}\nabla^2$ and $-\underline{\mathbf{u}}\cdot\nabla$, respectively. The

Reynolds number (Re) is defined as:

$$Re = \frac{\mathscr{U}_0 \mathscr{L}_0}{\nu}, \tag{2}$$

where $\mathscr{U}_0$, $\mathscr{L}_0$ and ν are the reference velocity, length scale and kinematic viscosity.

2.2. Spatial and temporal discretizations

Explicit sixth-order finite-differences are used for the discretization of the spatial operators on a rectilinear staggered grid. For the time discretization, an explicit low-storage third-order Runge-Kutta scheme (RK3) [6] is used, which takes the form:

$$\begin{bmatrix} \mathscr{J} & (c_m)\Delta t \mathscr{G} \\ \mathscr{D} & 0 \end{bmatrix} \begin{bmatrix} u^m \\ p^m \end{bmatrix} = \begin{bmatrix} f(u^{m-1}, u^{m-2}) \\ 0 \end{bmatrix}, \qquad m = 1, 2, 3, \tag{3}$$

where $\mathscr{J}$ is the identity operator with integrated boundary conditions, f is given as

$$f(u^{m-1}, u^{m-2}) = u^{m-1} + \Delta t[a_m \mathscr{N} u^{m-1} + b_m \mathscr{N} u^{m-2} + a_m \mathscr{L} u^{m-1} + b_m \mathscr{L} u^{m-2}], \tag{4}$$

where a_m, b_m and c_m (m=1, 2, 3) are coefficients of the Runge-Kutta scheme. The discretized form of this equation reads

$$\begin{bmatrix} \mathbf{J} & \mathbf{G} \\ \mathbf{D} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{u} \\ \mathbf{p} \end{bmatrix} = \begin{bmatrix} \mathbf{f} \\ \mathbf{0} \end{bmatrix}. \tag{5}$$

It is solved via a Schur complement [7, 8], which leads to a discretized Poisson equation for the pressure:

$$\underbrace{(\mathbf{D}\mathbf{J}^{-1}\mathbf{G})}_{\mathbf{A}}\mathbf{p} = \underbrace{\mathbf{D}\mathbf{J}^{-1}\mathbf{f}}_{\mathbf{b}}. \tag{6}$$

This equation is solved iteratively three times in each time step. The Schur complement matrix $\mathbf{A}$ has exactly one zero eigenvalue corresponding to the undefined pressure constant, but several iterative solvers can handle this rank-deficiency and produce a converged solution to this Poisson equation. More details on addressing the rank deficiency can be found in [1].

2.3. Solving the pressure Poisson problem

2.3.1. Overview of the CPU-based legacy solver and its GPU-enabled implementations

In the CPU-based MPI-parallel legacy solver [1] and its directly-ported GPU version [3], Eq. (6) is solved using the Krylov subspace method BiCGstab [9]. Low-frequency errors are removed by a V-cycle geometric multigrid preconditioner. This choice has been shown to substantially improve the convergence rate of the BiCGstab solver. Yet, we showed through Cray Performance Measurement and Analysis Toolset (Cray-PAT)[1] profiling reports that the relaxation operations of the preconditioner routine turned out to be the most expensive part of the entire scheme [3].

For accelerating the multigrid preconsitioner, the relaxation operators in this routine were replaced in [3] by data parallel types. This resulted in a promising speedup for the relaxation kernel itself, but the overall speedup for the multigrid routine, comprising up to 15 levels of coarsening, remained limited. This apparent performance offset is likely to be caused by the special memory hierarchy of the multigrid type operations. The problem was two-fold: first, the MPI communications between two grid levels required extra data copies to the GPU and back to the CPU, and second, the operations on the coarser grid levels proved to be memory-bound, and were in fact slower on GPU than on the CPU. In an ideal scenario, it would be favourable to perform the computationally intensive operations at once, and to communicate to the other GPUs as rarely as possible. This is indeed an obstacle for developing efficient multigrid schemes for GPUs, where in general no sensible speedup has been reported, even using very advanced implementations [2]. At the same time, one has to keep in mind that increasing the performance of the multigrid relaxation routines on GPUs (e.g., by lowering the levels of the coarsening) must not come at the cost of deteriorating the overall algebraic efficiency of the preconditioner. Furthermore, the iterative Krylov-based BiCGstab is not a suitable choice for GPUs either, probably due to reduction-based operations such as inner products and norm calculations [10].

3. An alternative hybrid task and data parallel Poisson solver

3.1. Overview

We propose a hybrid task and data parallel solution based on Jacobi method with over-relaxation to replace our best CPU-based scenario, i.e. BiCGstab with multigrid preconditioning. The key idea

[1] https://pubs.cray.com/content/S-2376/6.4.0/cray-performance-measurement-and-analysis-tools-user-guide-640/craypat-lite

[2] https://devblogs.nvidia.com/high-performance-geometric-multi-grid-gpu-acceleration/

is, that the higher numbers of required solver iterations, due to lower convergence rate of Jacobi, can be compensated by much faster computations per iteration on GPU. For instance, if the GPU can perform the computations two orders of magnitude faster than the CPU, then increasing the number of iterations by one order of magnitude still results in an order of magnitude speedup. As we will show in the following, this indeed turns out to be the case for solving the pressure Poisson equation.

3.2. The iterative scheme

The Jacobi method to solve the Eq. (6) can be written as

$$\mathbf{p}^{(r+1)} = \underbrace{(\mathbf{M}^{-1}\mathbf{b})}_{\mathbf{d}} + \underbrace{(-\mathbf{M}^{-1}\mathbf{R})}_{\mathbf{G}}\mathbf{p}^{(r)} \tag{7}$$

where $\mathbf{M}$ holds only the diagonal components of $\mathbf{A}$ and $\mathbf{R} = \mathbf{A} - \mathbf{M}$. $\mathbf{G}$ is the *point-iteration* matrix for the Jacobi method [11]. With a single-level relaxation using parameter ω, the Jacobi iteration with successive over-relaxation (SOR) is obtained

$$\mathbf{p}^{(r+1)} = \omega\mathbf{M}^{-1}(\mathbf{b} - \mathbf{R}\mathbf{p}^{(r)}) + (1-\omega)\mathbf{p}^{(r)}. \tag{8}$$

Equation (8) can be rearranged as

$$\mathbf{p}^{(r+1)} = \underbrace{\omega\mathbf{M}^{-1}\mathbf{b}}_{\mathbf{d}_\omega} + \underbrace{((1-\omega)\mathbf{I} - \omega\mathbf{M}^{-1}\mathbf{R})}_{\mathbf{G}_\omega}\mathbf{p}^{(r)}. \tag{9}$$

where $\mathbf{I}$ is the identity matrix with the same size as $\mathbf{A}$, and $\mathbf{G}_\omega$ is the *point-iteration* matrix for the iterative Jacobi method with SOR. The error at iteration r with respect to an exact solution $\mathbf{p}_*$ is

$$\mathbf{e}^{(r)} = \mathbf{p}_* - \mathbf{p}^{(r)}. \tag{10}$$

It follows from Eq. (9) that

$$\mathbf{e}^{(r)} = \mathbf{G}_\omega\mathbf{e}^{(r-1)}, \tag{11}$$

and convergence is achieved if

$$\lim_{r\to\infty}\mathbf{e}^{(r)} = \lim_{r\to\infty}\mathbf{G}_\omega^r = \mathbf{0}. \tag{12}$$

Hence, the convergence of the Jacobi iterative scheme with SOR is guaranteed if the spectral radius of $\mathbf{G}_\omega$ is less than unity, i.e., $\rho(\mathbf{G}_\omega) < 1$.

The point-wise format for a 6s+1 point three-dimensional finite-difference stencil of width 2s+1 takes the form

$$\begin{aligned} p_{i,j,k}^{(r+1)} = \frac{\omega}{\sum_{l=1}^{3} C_{l,0,i,j,k}} \Big(b_{i,j,k} - \sum_{q=-s}^{-1} C_{1,q,i,j,k} p_{i+q,j,k}^{(r)} - \sum_{q=1}^{s} C_{1,q,i,j,k} p_{i+q,j,k}^{(r)} \\ - \sum_{q=-s}^{-1} C_{2,q,i,j,k} p_{i,j+q,k}^{(r)} - \sum_{q=1}^{s} C_{2,q,i,j,k} p_{i,j+q,k}^{(r)} - \sum_{q=-s}^{-1} C_{3,q,i,j,k} p_{i,j,k+q}^{(r)} \\ - \sum_{q=1}^{s} C_{3,q,i,j,k} p_{i,j,k+q}^{(r)} \Big) + (1-\omega) p_{i,j,k}^{(r)} \end{aligned} \tag{13}$$

where $C_{l,-s:s,i,j,k}$ contains the stencil coefficients in direction l, for the rth approximation $p_{i,j,k}^{(r)}$. It can be directly seen from the right-hand side of Eq. (13) that the Jacobi method can be executed efficiently in a data-parallel fashion on the GPU, as opposed to the Gauss-Seidel method, which in its original form without relaxation reads

$$\begin{aligned} p_{i,j,k}^{(r+1)} = \frac{1}{\sum_{l=1}^{3} C_{l,0,i,j,k}} (b_{i,j,k} - \sum_{q=-s}^{-1} C_{1,q,i,j,k} p_{i+q,j,k}^{(r+1)} - \sum_{q=1}^{s} C_{1,q,i,j,k} p_{i+q,j,k}^{(r)} \\ - \sum_{q=-s}^{-1} C_{2,q,i,j,k} p_{i,j+q,k}^{(r+1)} - \sum_{q=1}^{s} C_{2,q,i,j,k} p_{i,j+q,k}^{(r)} - \sum_{q=-s}^{-1} C_{3,q,i,j,k} p_{i,j,k+q}^{(r+1)} \\ - \sum_{q=1}^{s} C_{3,q,i,j,k} p_{i,j,k+q}^{(r)}). \end{aligned} \tag{14}$$

Even though the Gauss-Seidel method has been preferred over the Jacobi method due to faster convergence and lower memory requirements, it is not as suited for a SIMD implementation. In the Gauss-Seidel method, for the computation of $p_{i_0,j_0,k_0}^{(r+1)}$, all $p_{i,j,k}^{(r+1)}$ with $i < i_0, j < j_0$ and $k < k_0$ should be already computed. This provides a favourable memory layout, as the matrix entries can be progressively updated without the need for allocating memory for both (r) and $(r+1)$ approximations. However, it does not allow a fully synchronous CUDA kernel to perform all operations of an iteration concurrently.

Usually, an absolute residual norm $\epsilon_\infty = ||(\mathbf{A}p - \mathbf{b})||_\infty$ or a relative residual norm $\epsilon_\infty^* = ||(\mathbf{A}p - \mathbf{b})||_\infty / ||\mathbf{b}||_\infty$ are considered for assessing the convergence of the pressure problem. Typically, a tolerance value ϵ_p is set and the iterations are terminated when $\epsilon_\infty \leq \epsilon_p$. The absolute residual

norm in discrete form is

$$\epsilon_\infty^{(r)} = \max_{i,j,k} \Big| b_{i,j,k} - \sum_{q=-s}^{s} C_{1,q,i,j,k} p_{i+q,j,k}^{(r)} - \sum_{q=-s}^{s} C_{2,q,i,j,k} p_{i,j+q,k}^{(r)} - \sum_{q=-s}^{s} C_{3,q,i,j,k} p_{i,j,k+q}^{(r)} \Big| \tag{15}$$

involving a reduction operation, which is not well suited for the shared-memory operations on the GPU. Therefore, we compute this norm on the CPU. This requires a full host-device data copy, which could be more costly than a few extra iterations on the device. Hence, we compute this norm only every n_{norm} iterations, where n_{norm} is typically set to 100. The relation between the error and the residual depends on the grid resolution and the truncation error $\mathbf{G}_\omega$, because the error for the r-th iteration $\mathbf{e}^{(r)}$ is related to the residual $\epsilon_\infty^{(r)}$ by

$$\epsilon_\infty^{(r)} = ||\mathbf{G}_\omega \mathbf{e}^{(r)}||_\infty. \tag{16}$$

Therefore, care must be taken when choosing the appropriate termination criterion, such that it does not need to be adjusted *ad hoc* when the grid resolution is changed. In what follows, we introduce a practical termination criterion which is suited for larger grid resolutions, in that it is less expensive than computing the residual or relative residual, and is adaptively adjusted based on the grid size.

3.3. A grid-size-adaptive termination criterion

In the following, we present a grid-size-adaptive termination criterion for the Jacobi iteration based on the second-order finite-difference discretization that is used for the pressure equation. We assume a uniform Cartesian grid and periodic boundary conditions in three dimensions. Extension to non-uniform Cartesian grids is straightforward and given at the end of this section.

For the Jacobi method with SOR and assuming second-order finite-difference discretization, the eigenvalue of $\mathbf{G}_\omega$ associated to a certain error mode $\mathbf{e}^{(r)} = \hat{\mathbf{e}}(\underline{\mathbf{k}}) e^{\mathrm{i}\underline{\mathbf{k}}\cdot\mathbf{x}}$ with the wave number $\underline{\mathbf{k}} = (k_x, k_y, k_z)$ (also known as amplification factor, denoted here by G_ω) can be obtained as

$$G_\omega(\underline{\mathbf{k}}) = \frac{||\mathbf{e}^{(r+1)}||}{||\mathbf{e}^{(r)}||} = 1 - \frac{2\omega}{3}(\sin^2(k_x h_x/2) + \sin^2(k_y h_y/2) + \sin^2(k_z h_z/2)), \tag{17}$$

where h_x, h_y and h_z denote the grid spacing in the x, y and z directions. It follows that convergence (regardless of its rate) is guaranteed when $0 < \omega \leq 1$. Furthermore, it follows from Eq. (17) that

largest wave numbers $\underline{\mathbf{k}}$ present the highest damping rates which explains the faster total convergence rates for the first iterations. The small wave number errors are the slowest to be damped because

$$\lim_{k_x,k_y,k_z\to 0} G_\omega = 1. \tag{18}$$

Note that this is the case for periodic boundary conditions, which allow exact zero wave numbers. However, it is possible to terminate the iterations when G_ω is below this limit by a small offset. This is inspired by the multigrid method, which may produce valid flow field data even though the coarsest grid level is often not of size $1 \times 1 \times 1$ (i.e., $N_{c,\min} = 1$ is the grid dimension in all directions). For instance, the multigrid routine in the CPU-based solver allows the coarsest grid level size of $N_{c,\min} = 4$ in all directions, due to MPI ghost cell layers for the sixth-order discretization (this follows the global discretization of the flow solver rather than the local second-order discretization in the Poisson solver). Given this offset, the CPU-based multigrid preconditioner still produces valid DNS data [1]. We proceed to find a suitable offset limit related to $N_{c,\min} = 4$ for the Jacobi iteration. It is expected that at large enough numbers of iterations (denoted by ∞_c), G_ω reaches the low wave numbers (denoted by 0_c) associated to $N_{c,\min} = 4$ in all directions. Additionally, the lowest modal convergence rate corresponding to this offset, which comes into effect at advanced iteration numbers, decreases for higher grid resolutions. If we assume a uniform mesh of size $M_x \times M_y \times M_z$ on a domain of size $[0, L_x] \times [0, L_y] \times [0, L_z]$, it can be written that

$$\lim_{k_x,k_y,k_z\to 0_c} G_\omega \simeq 1 - \underbrace{\frac{2\omega}{3}\left(\frac{4\pi^2}{M_x^2} + \frac{4\pi^2}{M_y^2} + \frac{4\pi^2}{M_z^2}\right)}_{\epsilon(M)} \leq 1 - \frac{8\omega\pi^2}{\max\{M_x, M_y, M_z\}^2} \simeq 1 - \mathcal{O}(M^{-2}), \tag{19}$$

where $M = \max\{M_x, M_y, M_z\}$. Therefore the error ratio between two subsequent iterations approaches unity when iterations progress, but depending on grid resolution, it remains below unity by an offset of order $\mathcal{O}(M^{-2})$ for all the wave numbers above 0_c limit. This ratio could also be obtained through power iteration in terms of the eigenvalues and eigenvectors of $\mathbf{G}_\omega$, where it approaches the largest eigenvalue of $\mathbf{G}_\omega$ (for the 0_c wave number limit) indicated by λ_1. The above analysis follows the concept of Von Neumann stability analysis [12].

We proceed to use the equivalence of the error ratio and pressure increment ratio between two subsequent iterations, where the pressure increment is $\mathbf{E}^{(r)} = \mathbf{p}^{(r)} - \mathbf{p}^{(r-1)}$. Following [11], this ratio could also be realized as

$$\frac{||\mathbf{E}^{(r+1)}||}{||\mathbf{E}^{(r)}||} = \frac{||\mathbf{p}^{(r+1)} - \mathbf{p}^{(r)}||}{||\mathbf{p}^{(r)} - \mathbf{p}^{(r-1)}||} = \frac{||\mathbf{e}^{(r+1)}||}{||\mathbf{e}^{(r)}||} \xrightarrow{r \to \infty_c} \lambda_1, \tag{20}$$

and we refer to it as κ_ω, hereinafter. Meanwhile, λ_1 can be estimated from Eq. (19), and thus it can be written that

$$\lim_{r \to \infty_c} \kappa_\omega = 1 - \epsilon(M). \tag{21}$$

where $\epsilon(M) = 2\omega/3(4\pi^2/M_x^2 + 4\pi^2/M_y^2 + 4\pi^2/M_z^2)$. We use this result, besides the residual ϵ_∞, as a termination criterion for the presented solver. We terminate the iterative procedure as soon as κ_ω has approached its maximum value, which is close to unity.

It is more convenient to formulate this termination criterion in terms of a new "error" and "tolerance". It can be written that

$$\ln \kappa_\omega \leq \ln(1 - \epsilon(M)), \tag{22}$$

and given that $|\epsilon(M)| \ll 1$, Taylor expansion of the right-hand side of Eq. (22) gives

$$\ln \kappa_\omega \leq -\sum_{n=1}^{\infty} \frac{\epsilon(M)^n}{n} = -\epsilon(M) + \mathcal{O}(\epsilon(M)^2), \tag{23}$$

neglecting the higher order terms and taking the absolute value of the inequality, we have

$$|\ln \kappa_\omega| \geq \epsilon(M), \tag{24}$$

that is, to terminate the iteration when the "error" $|\ln \kappa_\omega|$ reduces to its "tolerance" value $\epsilon(M)$. We will show in Section 5 that, even though $\epsilon(M)$ is the lower limit for $|\ln \kappa_\omega|$ (for the 0_c wave number limit), in practice, $\epsilon(M)$ can fall below $|\ln \kappa_\omega|$ for a few iterations and then return back above it, exhibiting oscillations. This is due to limited arithmetic precision: $\mathbf{E}$ may drop below machine precision and therefore is not evaluated correctly. However, these oscillations do not lead to numerical divergence, as a growing $|\kappa_\omega|$ will be fully representable with machine's arithmetic, and therefore, it will be damped again.

The convergence criterion given in Eq. (21) is only valid when $\mathbf{E}$ is computed for two subsequent iterations. The convergence criterion can be generalized for $n_{\text{norm}} > 1$. Supposing that $n_{\text{norm}} =$

$m_0 > 1$, a corresponding parameter κ_{ω,m_0} can be defined, which approaches to $\lambda_1^{m_0}$ as iterations progress to infinity:

$$\kappa_{\omega,m_0} = \frac{||\mathbf{E}_{m_0}^{(r+m_0)}||}{||\mathbf{E}_{m_0}^{(r)}||} = \frac{||\mathbf{p}^{(r+m_0)} - \mathbf{p}^{(r)}||}{||\mathbf{p}^{(r)} - \mathbf{p}^{(r-m_0)}||} = \frac{||\mathbf{e}^{(r+m_0)} - \mathbf{e}^{(r)}||}{||\mathbf{e}^{(r)} - \mathbf{e}^{(r-m_0)}||} = \frac{||\mathbf{G}_\omega^{m_0}(\mathbf{e}^{(r)} - \mathbf{e}^{(r-m_0)})||}{||\mathbf{e}^{(r)} - \mathbf{e}^{(r-m_0)}||} \xrightarrow{r \to \infty_c} \lambda_1^{m_0}. \tag{25}$$

Considering this limit value and resorting to the Eqs. (19), (20) and (21), it can be written that

$$\kappa_{\omega,m_0} \leq (1 - \epsilon(M))^{m_0} = \sum_{q=0}^{m_0} \binom{m_0}{q} (-\epsilon(M))^q = 1 - m_0\epsilon(M) + \mathcal{O}(m_0^2\epsilon(M)^2). \tag{26}$$

For a practical value of $n_{\text{norm}} = 100$, and for calculations on up to $\mathcal{O}(10^{10})$ degrees of freedom, $|m_0\epsilon(M)| \ll 1$. Thus, cancelling the second order terms of Eq. (26) and taking the logarithm gives

$$\ln \kappa_{\omega,m_0} \leq -\sum_{n=1}^{\infty} \frac{m_0^n \epsilon(M)^n}{n} = -m_0\epsilon(M) + \mathcal{O}(m_0^2\epsilon(M)^2), \tag{27}$$

or

$$|\ln \kappa_{\omega,m_0}| \geq m_0\epsilon(M). \tag{28}$$

Thus it is sufficient to terminate the iterations when $|\ln \kappa_{\omega,m_0}|$ reduces to $m_0\epsilon(M)$.

We finally show that the κ_ω termination criterion is also valid for stretched Cartesian grids. Because $\epsilon(M)$ is derived in the low wave number limit (this is equivalent to coarsest grid level allowed for the multigrid method), it is helpful to obtain an effective wave number for this level for a stretched grid. This wave number can be obtained based on a mean value for error wave lengths on this level, as the resulting wave here may not be monochromatic. For example, for the case of symmetric stretching (i.e., where grid spacings are symmetric with respect to the planes of symmetry $x = L_x/2$, $y = L_y/2$, $z = L_z/2$) the mean wave number can be calculated as $\bar{\lambda} = (\lambda_{c,1} + \lambda_{c,2})/2$, where $\lambda_{c,1} = \alpha L$ and $\lambda_{c,2} = (1 - \alpha)L$ and α is a stretching parameter (Fig. 1). It follows that $\bar{\lambda} = L/2$ which is independent of the stretching parameter α. It can be shown that this is valid for asymmetric stretched grids as well (up to four wave lengths maybe be involved, but the mean wave length remains the same), which shows that Eq. (19) holds also for these grids. We will verify this result by showing that using κ_ω termination criterion reduces ϵ_∞ by the same order of magnitude

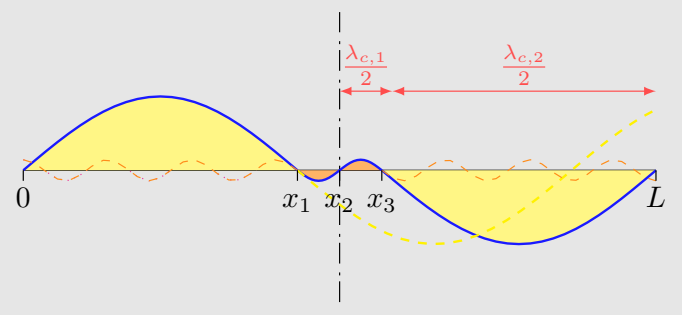


Figure 1: Schematic showing the two wave lengths involved in representing the error mode (solid blue line) at the low wave number limit ($N_{c,\min} = 4$, $\underline{\mathbf{k}} \to 0_c$) for a symmetric stretched grid. The dash-dotted line shows the centerline. L is the domain length in the given direction $x_1 = (1-\alpha)L/2$, $x_2 = L/2$ and $x_3 = (1+\alpha)L/2$ are the grid points used for discretization with the stretching parameter α. Two wave lengths $\lambda_{c,1}$ (sections filled with orange) and $\lambda_{c,2}$ (sections filled with yellow) are involved. The dashed lines show the extension of each section to the full interval $[0,L]$, which shows neither wave length could be considered to construct $\epsilon(M)$ due to periodic boundary conditions.

as the CPU-based preconditioned BiCGstab method.

3.4. Data decomposition and parallelism

Before introducing our parallel methodology for the solution of pressure Poisson equation (cf. Eq. (6)), we briefly describe the overall parallel organization of the entire flow solver. The global computational domain is decomposed into blocks which are distributed to different MPI threads (see Level I in [3]). The number of MPI threads is normally equal to the number of GPUs (n_{GPU}). If the CUDA multiprocess service (MPS) is used, the number of MPI processes n_{process} will be

$$n_{\text{process}} = n_{\text{GPU}} \times n_{\text{seed}} \tag{29}$$

where n_{seed} is the number of CPU cores on each node which communicate with the existing GPU on that node.

The computational operations arising from the iterative solution of pressure Poisson equation and other intensive operations of the explicit scheme, such as those for computing the nonlinear term $\mathcal{N}\mathbf{u}$, are offloaded to GPUs using CUDA kernels (see Level II in [3]). The GPU kernel structures for the parts of the flow solver other than the Poisson solver can be found in [3]. The host or driver routines, however, have been optimized for all GPU operations by moving memory instructions outside of the time-integration loop as much as possible.

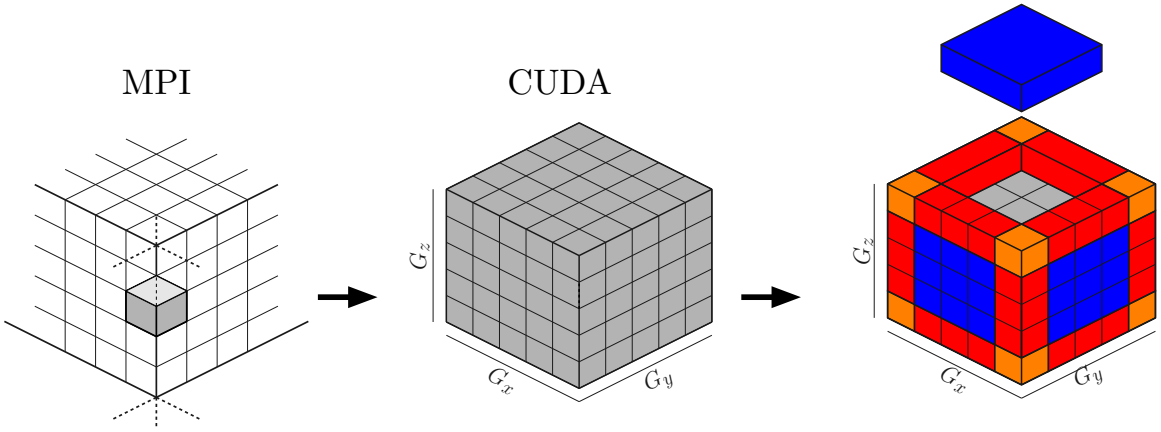


Figure 2: The MPI domain decomposition and the specific layout of boundary cells for minimal data exchanges.

The iterative part of the scheme (cf. Eq. (13)), which is also the computationally most expensive part, can be implemented on a massive number of CPU cores in a relatively straightforward manner: i) update the pressure based on Eq. (13); ii) compute the residual norm; iii) exchange the ghost cell data via MPI; iv) terminate if a predefined tolerance is satisfied; repeat. On GPUs, more instructions are needed in the iteration loop due to separation of the GPU memory accessible to the threads and the node memory (see Algorithm 1). To achieve a higher degree of data parallelism within the iterative solver, we split the computational block on one GPU into a central part and the boundary parts (Fig. 2). To minimize the memory overhead, we copy the entire pressure grid data to the CPU only every n_{norm} iterations to calculate the residual norm. At any other iteration, only the boundary data are copied to the CPU for the MPI communication. Because all three-dimensional arrays are copied to GPU memory as unfolded one-dimensional arrays, it is not straightforward to fetch specific strides of data from these arrays in the C environment embedded in the FORTRAN code. Therefore, we used an extra kernel (indicated as `GetBoundaryCells` in Algorithm 1) to fetch boundary data from GPU and copy them to the CPU for MPI communications. On the GPU side, this results in modifying the iterative kernel `cuJORx` to incorporate the boundary data for threads which are operating on cells next to the GPU grid boundaries.

Double-precision data parallel routines are implemented as external CUDA C kernels. The iso_C_binding library is used for memory allocation on the graphics card, for data transfer between host and device, for defining CUDA C data and event types in FORTRAN, for passing FORTRAN host variables by C pointers to the device memory, and for launching the GPU kernels. Further details on the C-FORTRAN interoperability standards used in our implementation can be found in

Algorithm 1: Driver for GPU-based Poisson Solver

Data: $\mathbf{DJ^{-1}Gp} = \mathbf{DJ^{-1}}(-\mathbf{N} + \mathbf{f} + Re^{-1}\mathbf{L})\mathbf{u}$
Result: $\mathbf{p}$
build the C-FORTRAN interface;
initialize FORTRAN and CUDA C pointers;
if *first time-step* **then**
  allocate GPU memory for $\mathbf{p}$;
  allocate GPU memory for ghost and boundary cells;
  allocate GPU memory for diagonals (band) of $\mathbf{A} = \mathbf{DJ^{-1}G}$;
  allocate GPU memory for $\mathbf{b} = \mathbf{DJ^{-1}}(-\mathbf{N} + \mathbf{f} + Re^{-1}\mathbf{L})\mathbf{u}$;
  copy non-zero entries of $\mathbf{A}$ to GPU;
end
while $\epsilon_\infty \geq \epsilon_p$ **do**
  iteration=iteration+1;
  if *iteration=1* **then**
    copy $\mathbf{b}$ to GPU;
    copy $\mathbf{p}$ to GPU;
  end
  exchange ghost-cell data with other GPUs via MPI and CUDA;
  if mod (*iteration,* n_{norm})*=1* **then**
    copy $\mathbf{p}$ from GPU to the host CPU;
    compute the residual norm (e.g. $\epsilon_\infty = ||(\mathbf{Ap} - \mathbf{b})||_\infty$);
    get the maximum of residual from all the GPUs;
  end
  configure the GPU grid;
  setup kernel arguments for `cuJORx` kernel;
  launch the `cuJORx` kernel;
  setup kernel arguments for `GetBoundaryCells` kernel;
  launch `GetBoundaryCells` kernel;
  copy the boundary cells to host for MPI exchange;
end
if *last time step* **then**
  free the locally allocated GPU memory;
end

[3].

For performance monitoring, we compare the elapsed MPI wall clock time of our GPU implementation with that of the original MPI-parallel implementation. For the CPU case, we deploy our best-case scenario Poisson solver which yields the lowest time to solution for the MPI implementation (a geometric multigrid preconditioner and BiCGstab). The timings are measured by placing MPI barriers within the time integration loop, so the MPI wall clock times are measured. The elapsed time within the legacy CPU-based solver was reported as $\overline{T_c^P}$, and the averaged elapsed time for the

GPU-based solver was reported as $\overline{T_p^P}$. It follows that an effective acceleration factor can be defined as

$$S_{eff} = \left\lceil \frac{\overline{T_c^P}}{\overline{T_p^P}} \right\rceil, \tag{30}$$

where $\lceil \cdot \rceil$ denotes a ceiling function. This effective acceleration factor is labeled as effective because it reflects the overall reduction in time-to-solution for the pressure problem, when using the GPU-based solver instead of the CPU-based legacy solver. It is important to note that our best-case scenario implementation for CPUs has been used to collect the CPU-based data. If a CPU-based solver with the exact same numerical scheme (e.g. Jacobi iteration) was used, the resulting acceleration factor would be significantly larger than reported here, although an optimized CPU-based solver would probably not use such a numerical method.

4. Benchmark problem I: Taylor-Green vortex transition to turbulence and decay at $Re = 1600$

4.1. Overview

The transition to turbulence and decay of the Taylor-Green vortex (TGV) is used to validate the GPU-based flow solver. Initial conditions for the vortices in a cube with an edge length of 2π are given as

$$\begin{aligned} u_x(x,y,x,t=0) &= +\frac{2}{\sqrt{3}}\sin(\beta+\frac{2\pi}{3})\sin(x)\cos(y)\cos(z) \\ u_y(x,y,z,t=0) &= -\frac{2}{\sqrt{3}}\sin(\beta-\frac{2\pi}{3})\cos(x)\sin(y)\cos(z) \\ u_z(x,y,z,t=0) &= +\frac{2}{\sqrt{3}}\sin(\beta)\cos(x)\cos(y)\sin(z), \end{aligned} \tag{31}$$

with periodic boundary conditions on the domain $-\pi \leq x,y,z \leq \pi$. β is a free parameter and is set to zero here. The Reynolds number based on the vortex length is set to $Re = 1600$. For the GPU-based solver, the relaxation parameter ω is set to 0.8 and the number of iterations for calculating the norm n_{norm} is set to 30.

4.2. Results

Figure 3 shows the transition and decay of the TGV from t=0 to 20. Temporal evolution of the total kinetic energy $E_{k,\Omega}$ and its dissipation rate defined as

$$\frac{-dE_{k,\Omega}}{dt} = \frac{-d}{dt}\int_{\Omega}\frac{1}{2}\underline{\mathbf{u}}\cdot\underline{\mathbf{u}}d\Omega \tag{32}$$

are computed and analysed in the following. A series of hybrid simulations on up to 64 GPUs have been performed (Tab. 1).

The solution to Poisson equation for the TGV simulation usually converges with few iterations due to the lack of external forcing. Therefore, we do not use the termination criterion developed in section 3.3, instead we examine the absolute residual ϵ_∞ for termination. First, we investigate the sensitivity of the $E_{k,\Omega}(t)$ and $-dE_{k,\Omega}/dt$ to the termination tolerance ϵ_p (Fig. 4), for different tolerance values $\epsilon_p = 10^{-2}$ (case T1), $\epsilon_p = 10^{-4}$(case T2) and $\epsilon_p = 10^{-6}$ (case T3). It is clearly seen that the sensitivity to ϵ_p is small at the given resolution. On the left panel of Fig. 4, a small overestimation for the dissipation rate of $E_{k,\Omega}$ can be observed for T1 at $t \approx 13$, whereas curves for T2 and T3 are nearly indistinguishable. Therefore, we proceed to use the termination tolerance $\epsilon_p = 10^{-4}$ for all other cases.

Next, we examined the spatial order of convergence of our implementation. To this end, we performed the cases G2, G3b and G4 (cf. Tab. 1), where the resolution was increased in all three directions by factors of 2, 4 and 8, respectively. A total of 64 GPUs were used for the cases G3b and G4, while only 8 GPUs were deployed for G2. The dissipation rate of $E_{k,\Omega}$ for T2, G2, G3b and G4 up to $t = 10$ is shown on the left panel of Fig. 5. Note that, we acquired data for case G4 only until $t \approx 8.6$ because of the high computational costs of this case with approximately 1.7 billion grid points. Additionally, we compared the results to the spectral direct numerical simulations of Van Rees et al. [13], who used 512^3 grid points. For $t < 8$, the case T2 underestimates the dissipation rate, while for $t > 8$, it provides a good agreement with the spectral DNS. The cases G2, G3b and G4 are in good agreement with the spectral DNS data, although G2 and G3b slightly over-estimated the dissipation rate for $t > 9$. The absolute error $|\epsilon_t|$ is plotted against the normalized grid spacing $\tilde{h}$ on the right panel of Fig. 5. This error at time t was calculated as

$$|\epsilon_t| = \left| -\frac{d}{dt}E_{k,\Omega}(t) + \frac{d}{dt}E^*_{k,\Omega}(t)\right| \tag{33}$$

where $E^*_{k,\Omega}(t)$ was the total kinetic energy at time t for the reference case G4, which used the

highest resolution. It is observed that for the smaller grid spacings $\tilde{h}$, an approximate fourth order of convergence was recovered, which falls between the sixth order of convergence which is used for all computations outside the pressure solver, and the second order stencil which is used in the GPU-based pressure solver.

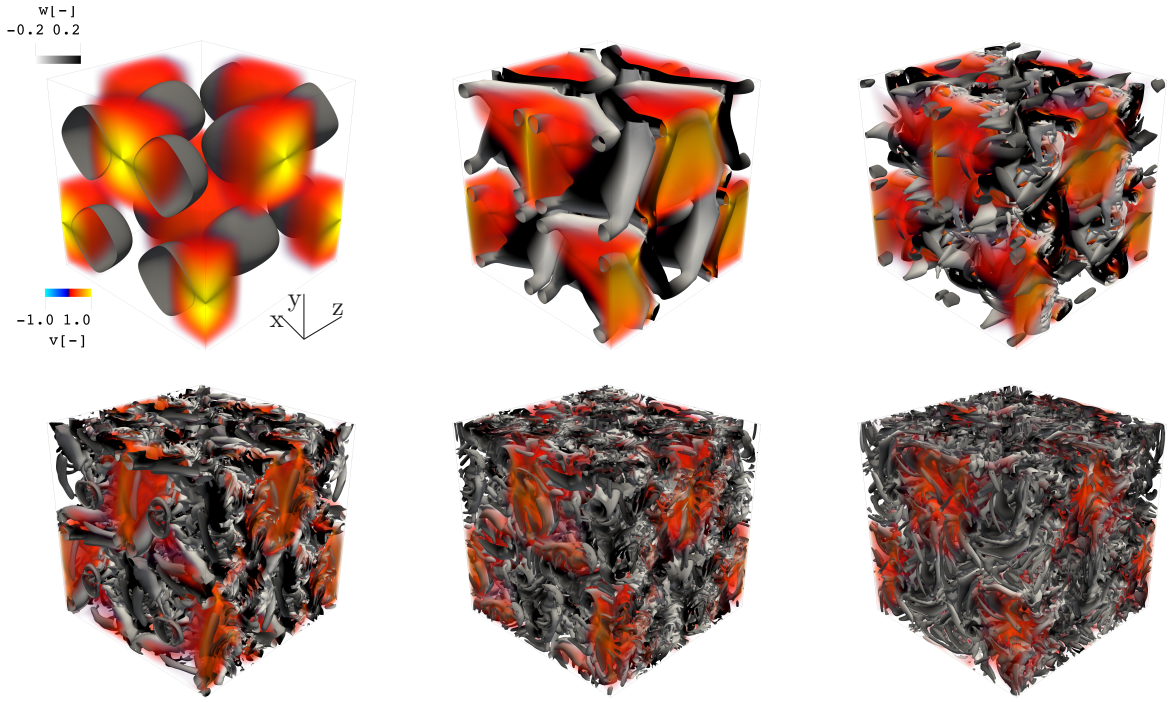


Figure 3: Transition to turbulence and decay of the Taylor-Green Vortex at $Re = 1600$. Iso-surfaces of $\lambda_2 = -0.2$ [14] are coloured by the z-component of the velocity field (denoted by w) are shown together with volume renderings of positive part of the y-component of velocity (denoted by v). The snapshots are taken at times $t = 0, 4, 8, 12, 16, 20$ from top left to the bottom right.

We also provide the computational performance data for the data parallel Poission solver for all the cases listed in Tab. 1, where timings for the Poisson solver are compared to the original CPU-based solver for corresponding compute node configurations and problem sizes. We enabled the NVIDIA multiprocess service (MPS) for all GPU runs, such that four MPI processes shared the work on the available GPU on each node (instead of only one driver MPI process per node, which is the single process standard). Simulation run C0 was performed using only the CPU-based solver for a short physical time $t = 0.015$ with 16 MPI processes allocated per node, in order to acquire the electrical energy consumption E_t of the simulation runs when the CPU code utilized all

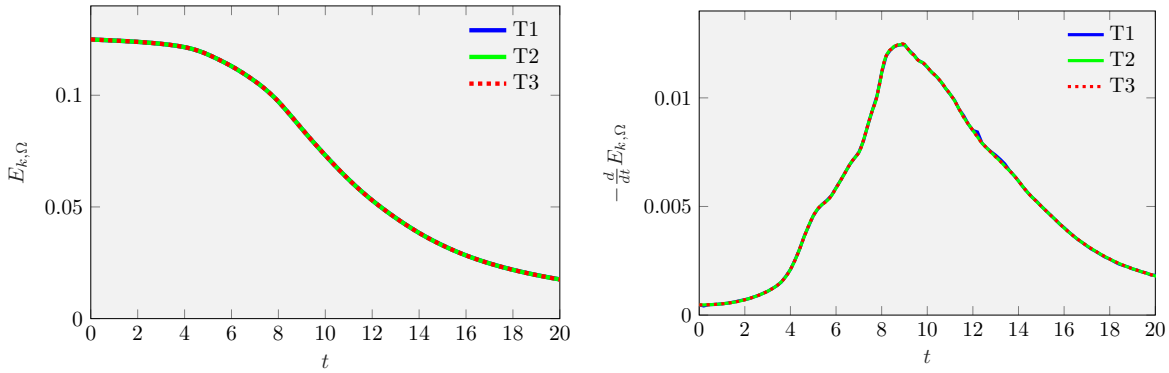


Figure 4: Temporal evolution of the total kinetic energy $E_{k,\Omega}$ (left) and its dissipation rate (right) using three different values for cases T1, T2 and T3. Simulations where performed on 128^3 grid points.

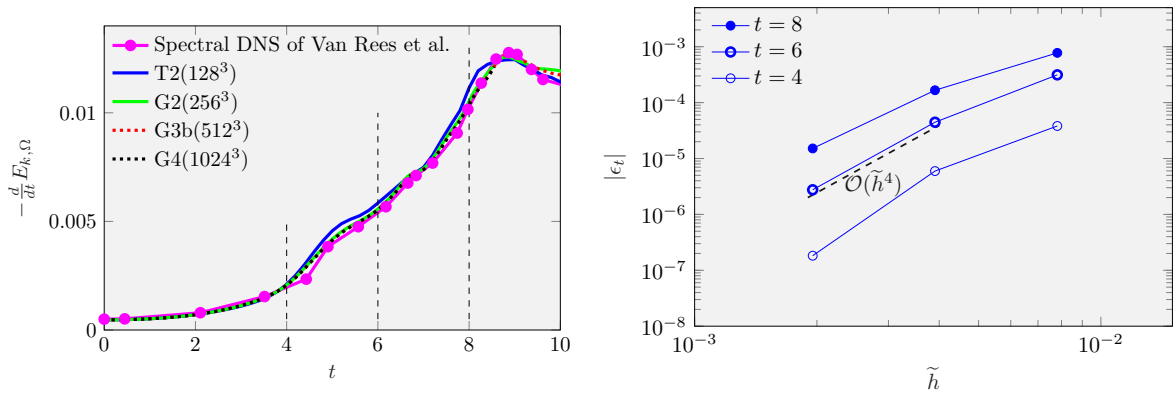


Figure 5: (Left) effect of spatial resolution on the temporal evolution of the dissipation rate of the total kinetic energy $E_{k,\Omega}$ for the Taylor-Green Vortex simulation. Simulations using four different grid resolutions 128^3 (T2), 256^3 (G2), 512^3 (G3b) and 1024^3 (G4) are performed and compared against the spectral DNS results of Van Rees et al. [13]. The termination tolerance in the Poisson solver was set to $\epsilon_p = 10^{-4}$. (Right) The absolute error $|\epsilon_t|$ is plotted against the normalized grid spacing $\widetilde{h}$ ($= h/2\pi$, where 2π is the domain's length). ϵ_t is presented for T2, G2, and G3b at three different times $t = 4, 6, 8$ (dashed lines on the left panel), with respect to the reference case G4. Dashed black line indicates fourth ($\mathcal{O}(\widetilde{h}^4)$) order of convergence.

12 cores available on each node with an extra 4 processes using hyperthreading (number of MPI processes are set to result the lowest elapsed time for each case). The energy data are collected from dedicated monitoring tools CrayPAT. The total consumed energy is reported using this toolset, thus the energy consumption for the GPU cases includes the consumption by the CPU cores as well.

Case	n_{DOF}	GPUs($\mathbf{N}$)	MPS(n)	ϵ_p	$\overline{T_p^P}$	$\overline{T_c^P}$	S_{eff}	t_e	E_t(MJ)
T1	128^3	yes(8)	yes(4)	10^{-2}	0.005	0.16	32	22	14.04
T2	128^3	yes(8)	yes(4)	10^{-4}	0.005	0.16	32	22	14.57
T3	128^3	yes(8)	yes(4)	10^{-6}	0.006	0.16	27	22	24.04
G2	256^3	yes(8)	yes(4)	10^{-4}	0.04	0.65	16	22	73.41
G3a	512^3	yes(8)	yes(4)	10^{-4}	0.32	4.35	14	9.11	198.43
G3b	512^3	yes(64)	yes(4)	10^{-4}	0.038	1.93	51	13.9	409.31
G4	1024^3	yes(64)	yes(4)	10^{-4}	0.32	14.45	48	8.59	1562.22
C0	512^3	no(64)	no(16)	10^{-4}	-	1.47	-	0.015	1.85

Table 1: The first set of Taylor-Green vortex simulations performed on Cray XC40/50 GPU nodes (Intel® Xeon® E5-2690 v3 @ 2.60GHz (12 cores, 64GB RAM) and NVIDIA® Tesla® P100 16GB). This set includes cases performed for validation and assessing the energy consumption of the solver. The total number of grid points is given as n_{DOF}. $\mathbf{N}$ and n are the number of GPUs (or nodes) and the number of MPI processes per node, respectively. All runs (except C0) deployed GPUs and used the multiprocess service (MPS) for streaming the computations from more than one CPU core to the existing GPU on a node. ϵ_p is the termination tolerance for the GPU-based pressure solver. The elapsed time for the GPU-based and CPU-based pressure solvers as well as the effective acceleration factor are given as $\overline{T_p^P}$, $\overline{T_c^P}$ and S_{eff}, respectively. t_e denotes the physical time at which the simulation run was terminated and E_t(MJ) is the total energy required for each case.

The effective acceleration factor S_{eff} for T1, T2 and T3 in Tab. 1 show that changing the termination tolerance ϵ_p does not significantly influence the GPU acceleration. Comparing S_{eff} for T2, G2 and G3a shows that the acceleration factor decreases when the number of grid points per GPU is increased. This is in contrast to our findings in [3], where it was shown that the GPU performance generally improves for larger grid resolutions. This discrepancy may be caused by the cost of calculating the residual norm or by communicating the ghost cell data communication between the GPUs, which both become significant when the number of degrees of freedom per node is increased. This overhead is also due to elevated number of iterations in the Jacobi solver for larger grid sizes.This can also be observed by comparing cases G3a and G3b, where increasing the number of GPUs resulted in a three times higher acceleration factor. Apart from speeding up the calculations, which lowers the cost in terms of node hours on a supercomputer, GPUs are also reputed for lowering the energy consumption and thus reducing electric power cost. To compare the

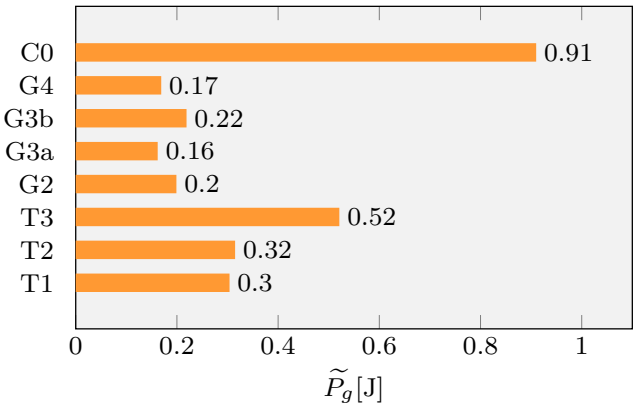


Figure 6: Energy consumption per grid point per unit of physical time for all cases listed in Tab. 1 for the Taylor-Green vortex simulation.

energy consumption for the runs in Tab. 1, we define a normalized measure as

$$\widetilde{P_g} = \frac{E_t}{t_e \times n_{\mathrm{DOF}}}, \tag{34}$$

which is the total energy per unit of simulated flow-related time per grid point. Since we use flow-related time t_e here for normalization, not the elapsed wall time, the presented measure is not illustrating power, and $\widetilde{P_g}$ has the unit of Joule, because t_e is dimensionless. It can be seen from Fig. 6 that the normalized energy consumption $\widetilde{P_g}$ remains around 0.3J for all GPU-enabled cases with $\epsilon_p = 10^{-4}$, whereas it reaches 0.91 J for the CPU-only case C0. This indicates that the GPU-enabled solver cuts the energy consumption by approximately 70%.

In the remainder of this section, we present a second set of simulation runs for which we put more focus on the parallel performance of the proposed solver, in particular, on the strong scaling capabilities. To this end, we performed a total of 87 simulation runs on the Cray XC40/50 (*Piz Daint*), where the total number of degrees of freedom was kept constant at 512^3 (cf. Tab. 2). Starting from one node, we doubled the number of nodes (or GPUs, because there is one GPU per node) up to to 512 nodes (each row in Tab. 2 corresponds to a fixed number of nodes), and for each node configuration, we utilized $n = 1, 2, 4, 8, 16$ processes per node. We performed simulations with the CPU-based Poisson solver ($g = 0$) and the GPU-based Poisson solver ($g = 1$ in Tab. 2). Accordingly, the timings in columns of Tab. 2 with $g = 0$ correspond to $\overline{T_c^P}$, and those in columns with $g = 1$ correspond to $\overline{T_p^P}$ for constant n and number of nodes.

Case	$\frac{n=1}{g=0}$	$\frac{n=1}{g=1}$	$\frac{n=2}{g=0}$	$\frac{n=2}{g=1}$	$\frac{n=4}{g=0}$	$\frac{n=4}{g=1}$	$\frac{n=8}{g=0}$	$\frac{n=8}{g=1}$	$\frac{n=16}{g=0}$	$\frac{n=16}{g=1}$
S1-n-g	80.5	-	46.31	-	30.1	-	24.8	-	23.64	-
S2-n-g	40.34	4.09	23.17	2.14	15.47	1.24	12.52	-	12.29	-
S4-n-g	20.52	2.05	11.96	1.09	8.07	0.64	6.29	0.37	6.20	0.31
S8-n-g	10.57	1.03	6.12	0.54	4.30	0.32	3.29	0.16	6.07	0.17
S16-n-g	5.39	0.52	3.22	0.28	2.28	0.14	4.14	0.085	3.55	0.082
S32-n-g	2.78	0.26	1.73	0.12	3.65	0.07	2.26	0.043	2.05	0.042
S64-n-g	1.42	0.12	3.21	0.065	1.93	0.036	1.22	0.022	1.51	0.024
S128-n-g	3.03	0.064	1.66	0.032	1.03	0.020	0.80	0.013	0.92	0.015
S256-n-g	1.60	0.035	0.90	0.021	0.61	0.010	0.44	0.009	0.25	0.022
S512-n-g	0.88	0.018	0.51	0.012	0.36	0.009	0.17	0.011	0.11	0.012

Table 2: Second set of the Taylor-Green vortex simulations, performed on GPU nodes of a Cray XC40/50 (Intel® Xeon® E5-2690 v3 @ 2.60GHz (12 cores, 64GB RAM) and NVIDIA® Tesla® P100 16GB). The cases in this set are performed to examine the computational performance. Cases are tagged as "S**N**-n-g", where **N** is the number of nodes (or GPUs), n denotes the number of MPI processes per node, and g indicates if the GPU solver ($g = 1$) or the CPU-based MPI-parallel solver ($g = 0$) is used. Timings are given in seconds [s].

Figure 7 shows the effective acceleration factor S_{eff} for different node configurations. It can be seen that acceleration factors up to 60 are achieved, and the effective acceleration factor never drops below $S_{eff} \approx 10$. For lower numbers of nodes (or GPUs), increasing the number of MPI processes on a node results usually in a higher GPU-acceleration, while for large numbers of nodes ($\mathbf{N} \geq 16$), this is not always the case.

4.3. Strong scaling

The data in Tab. 2 is then used to examine the strong scalability of the proposed Poisson solver. For every fixed number of MPI threads per node (or GPU), we plotted in Fig. 8. the elapsed time for 2, 4,$\cdots$, 512 nodes (or GPUs). The problem size was fixed at 512^3 grid points. Because the data associated with such a problem size does not fit into the memory of a single P100 GPU, the elapsed times for one GPU ($\mathbf{N} = 1$) are not presented in Tab. 2. Also missing are the data for $\mathbf{N} = 2$ when $n \geq 8$ due to thread memory overhead in the CUDA multi-process service. For the tested grid size, the CPU-based MPI-parallel solver diverges from an ideal scaling when the number of threads exceeds 64. This result is independent of number of MPI processes per node. For instance, cases S**N**-1-0 diverge from ideal scaling when $\mathbf{N} = 64$, while cases S**N**-4-0, diverge from ideal scaling when $\mathbf{N} = 16$. The deviation from ideal strong scaling in the CPU-based solver is likely to be caused by an interplay between the MPI regrouping operations and the memory layout in the multigrid preconditioner.

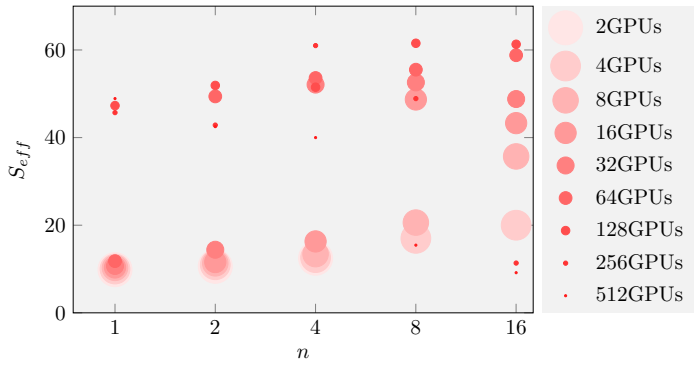


Figure 7: Effective acceleration factor (S_{eff}) obtained using the GPU-based solver, compared to the legacy CPU-based solver, are reported for the Taylor-Green Vortex simulation at $Re = 1600$. n shows the number of MPI processes on each node. At a fixed number of GPUs, n is varied, and the acceleration is then measured with respect to the elapsed time in the Poisson solver for the CPU-based and GPU-based solvers.

The GPU-based solver exhibits ideal strong scaling when one MPI thread per node is used (cases S**N**-1-1, when **N**=2, 4,$\cdots$, 512). When the number of MPI processes per node is increased, we observed that the GPU-solver achieved slightly better performance than it would be expected for the linear scaling at certain numbers of nodes **N** (e.g., cases S32-2-1, S16-4-1 and S8-8-1). Such an improvement in scaling was not observed when 16 MPI threads were used per node. Nonetheless, this configuration also presented ideal scaling up to $\mathbf{N} = 64$. Increasing the number of MPI processes beyond 12 does not seem to aid the GPU solver with better data streaming between host and device. This observation could be related to the fact that there are only 12 cores per node. The lowest time-to-solution is achieved for case S256-8-1 ($\overline{T_p^P} = 0.009$s), which results in a total speedup factor of 8978 (compared to the serial case S1-1-0 which converged in 80.8s). The results for the multigrid preconditioned GPU-based solver [3] are also shown in Fig. 8 without multi-process service ($n = 1$). It is observed that this implementation is only slightly advantageous to CPU-based solution for large **N**.

4.4. Efficacy of the κ_ω termination criterion at larger grid sizes

The efficacy of κ_ω criterion in terms of dropping the residual ϵ_∞ is investigated for increasing grid sizes. To this end, the pressure problem is solved for four grid sizes 128^3, 256^3, 512^3 and 1024^3 using $\mathbf{N} = 1, 8, 64$ and 512 nodes (or GPUs), respectively. The ratio of residual ϵ_∞ after termination to the residual when iterations start is defined as

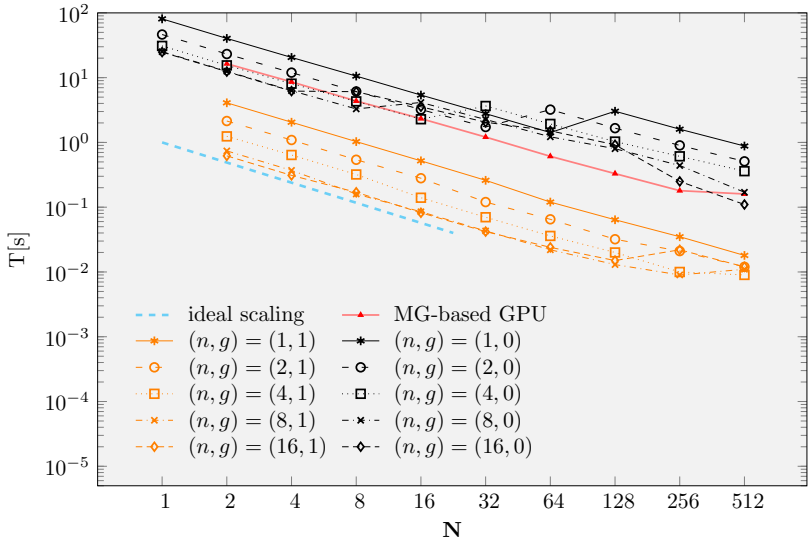


Figure 8: Time-to-solution obtained with the GPU solver ($g = 1$, orange), compared to the CPU-based solver ($g = 0$, black) for the Taylor-Green Vortex simulation at $Re = 1600$. n shows the number of MPI-processes on each node and **N** denotes the number of GPUs (or nodes). Dashed blue line shows ideal scaling. The red line shows the results for the multigrid preconditioned GPU-based solver [3] with $n = 1$.

$$\Gamma^{+}_{c,\infty} = \frac{\epsilon^{+(K)}_{\infty}}{\epsilon^{+(0)}_{\infty}}, \tag{35}$$

where K is the number of iterations when the termination is signalled. The "+" sign indicates that the residual is obtained by summation over the three Runge-Kutta substeps (because the pressure problem is solved once per substep). This four problem sizes are tested once using the GPU-based solver with κ_{ω,m_0} criterion with $m_0 = 100$. and once using the CPU-based preconditioned BiCGstab solver. The CPU-based solver uses a termination criterion based on the BiCGstab method [1]. Figure 9 shows that the $\kappa_{\omega,100}$ termination criterion drops the residual ϵ_∞ by similar orders of magnitudes as the preconditioned BiCGstab method.

5. Benchmark problem II: turbulent flow in the wake of a solid sphere at $Re_D = 3700$

5.1. Overview

In this section, we present an implicit large eddy simulation (ILES) of the flow around a solid sphere at $Re_D = 3700$, where the Reynolds number is defined as

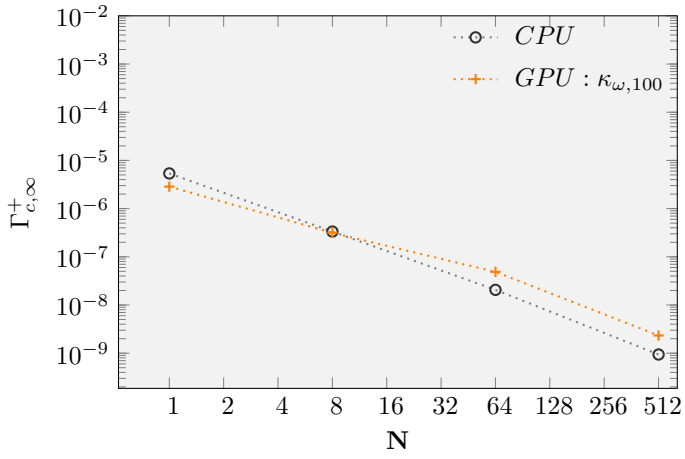


Figure 9: Effect of grid size on $\Gamma^{+}_{c,\infty}$ when κ_{ω,m_0} termination criterion is used in the GPU-based solver. The CPU-based solution is used as reference which deploys the preconditioned BiCGstab method.

$$Re_D = \frac{U_\infty D}{\nu}, \tag{36}$$

where U_∞, D and ν are the free stream velocity, sphere diameter and kinematic viscosity, respectively. This flow configuration has been studied using numerical simulations [15, 16] and experiments [17]. Here, we briefly report turbulent flow statistics in the wake of the sphere to validate the present solver. The periodic computational domain is shown in Fig. 10. The origin of the coordinate system is located on the downstream end of the sphere with respect to the bulk flow direction z. In contrast to the Taylor-Green vortex transition and decay, the current case uses localized forcing to represent the sphere with an Immersed Boundary technique. Such discrete forcing usually results in a considerable increase in the number of iterations in the Poisson solver. This effect has to be evaluated for the present solver, as the speedup achieved by data parallelism may be offset through a larger number of iterations.

According to *a posteriori* calculations of [15], the near wake flow structures ($z/D < 3$) have been associated to a Kolmogorov length scale of $\eta_{kol}/D = 0.0134$. Their simulation used an average grid spacing of $\bar{h}/D = 0.008$ in the wake region, with a second-order body-fitted mesh flow solver. In our former work [3], we used $N_x \times N_y \times N_z = 256 \times 256 \times 768 \approx 50.3M$ grid points together with appropriate grid stretching to accommodate a similar grid spacing in the near wake region, which ensured sufficient resolution for scale-resolving simulation of near-wake flow ($h/D \approx 0.008 < \eta_{kol}$).

We chose here to reduce the grid resolution by a factor of two in all the three directions, hoping to recover the time-averaged flow fields by taking advantage of higher-order discretization schemes than [15]. Accordingly, we compute the turbulent flow past the sphere using $N_x \times N_y \times N_z = 128 \times 128 \times 384 \approx 25.2M$ grid points, which results in the near-wake grid spacing of $h/D \approx 0.016$. Because the grid resolution used here is slightly higher than η_{kol}, the simulations are considered ILES or under-resolved DNS. We use 24 GPUs (24 nodes). Boundary conditions, fringe forcing for

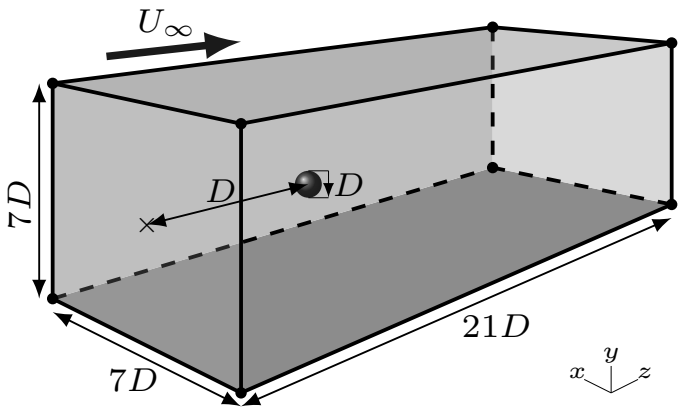


Figure 10: Periodic computational box dimensions for the simulation of the turbulent wake behind the sphere at $Re_D = 3700$ illustrated together with size and position of the immersed sphere.

driving the bulk flow, and the immersed boundary forcing on the sphere surface are imposed as described in [3, 18, 19].

5.2. Results

The simulation is performed for a total time of $t \approx 80D/U_\infty$ on 24 GPUs. The transition to turbulence in the wake was triggered by placing the sphere off the channel center by $0.035D$ in the x direction. Transient structures were washed out of the domain's outlet at approximately $t = 30D/U_\infty$. From this time onward, the simulation was continued for approximately 11 shedding cycles ($St = f_s D/U_\infty = 0.215$ [15], where f_s is the frequency of the principal vortex shedding).

The instantaneous streamwise velocity field behind the sphere at time $t = 49D/U_\infty$ (Fig. 11) shows complex flow structures due to interaction of boundary layer and wake instabilities up to approximately $z = 3D$. Reverse flow is observed roughly until $z = 2D$, which is in agreement with

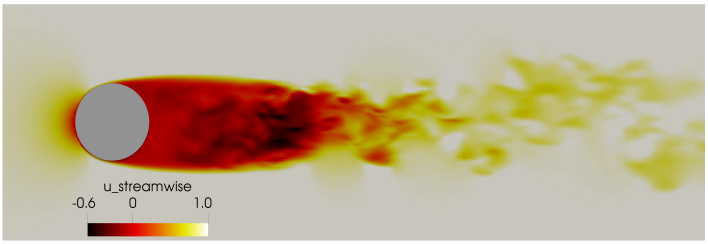


Figure 11: Streamwise velocity for the ILES of flow around sphere at $Re = 3700$ in the x-z plane at $t = 49D/U_\infty$. The snapshot shows strong reverse flow up to $z = 2D$.

[15].

Vortical structures in the wake, indicated by iso-surfaces of $\lambda_2 = -0.2$ [14], together with volume renderings of the pressure field are illustrated in Fig. 12. Adverse pressure gradient downstream of the sphere is apparent roughly up to $z = 2D$.

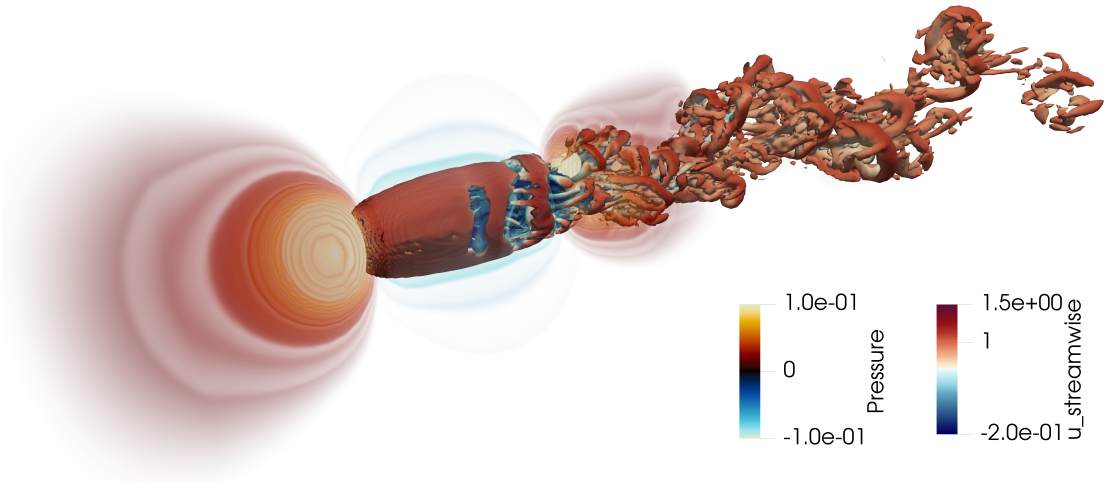


Figure 12: Iso-surfaces of $\lambda_2 = -0.2$ to illustrate the vortex structures in the wake of the sphere at $t = 63D/U_\infty$. Iso-surfaces are coloured by the non-dimensional streamwise velocity. The volume renderings show the pressure field behind, around, and past the sphere.

We compare the time-averaged streamwise velocity profiles $\overline{w}$ for different cross-sections down-

stream of the sphere and in the near-wake region. For calculating the mean quantities, we statistically average the simulation data over ten principal shedding cycles after the initial transient effects were washed out of the simulation domain. Fig. 13 shows the mean streamwise flow profiles in the y direction up to $y/D = 2$ from the centerline for two locations $z = 1.6D$ and $z = 2D$. Results agree reasonably well to those of [15] who averaged over much longer times.

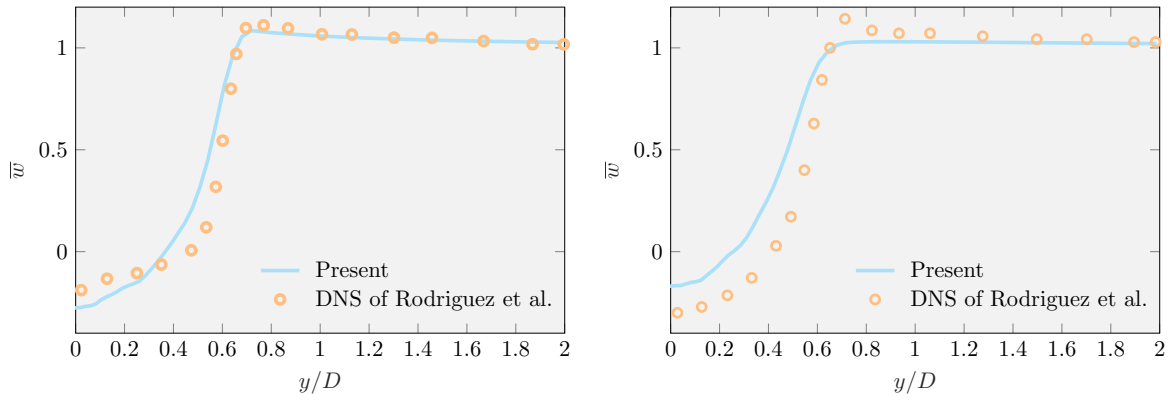


Figure 13: Time-averaged streamwise velocity $\overline{u}_z$ at (left) $z = 1.6D$ and (right) $z = 2D$. Solid lines indicate present numerical simulations, and symbols are obtained from DNS data of Rodriguez et al. [15].

Further, we briefly investigate the frequency spectra for the oscillations in the streamwise velocity w at a fixed location within the recirculation zone. To this end, we record the time history of the streamwise velocity fluctuations w' for 8 shedding cycles, between the times $t_1 = 42D/U_\infty$ and $t_2 = t_1 + 8/St$:

$$w'(x_0, y_0, z_0, t) = w(x_0, y_0, z_0, t) - \overline{w}(x_0, y_0, z_0), \tag{37}$$

where $\overline{w}$ is the time-averaged streamwise velocity over the interval $[t_1,t_2]$. w' is collected at a probe located $3D$ downstream of the sphere's leading point. The power spectrum (Fig. 14) indicates the -5/3 scaling law. It is also observed that the maximum frequency peak (marked by a solid red circle) corresponds to $St = 0.24$ which is close to reference DNS data of Rodriguez et al. [15] ($St = 0.215$) and experimental measurements of Kim and Durbin [17]($St = 0.225$).

5.3. Strong scaling

Next, we compare the strong scaling performance of the present implementation to the scaling of the legacy CPU-based solver. Similar to the data in Tab. 2 for the Taylor-Green vortex simulation,

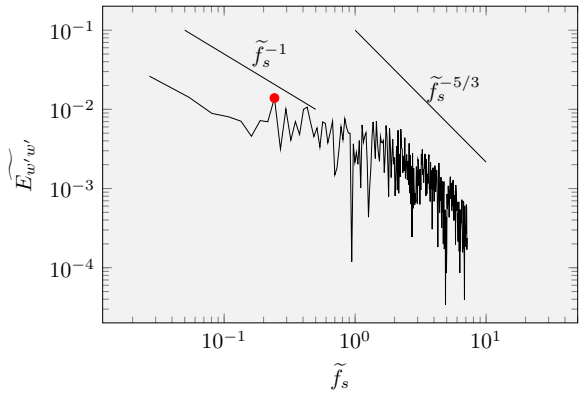


Figure 14: Power spectrum at $z = 3.5D$. The frequency peak associated with the highest fluctuation energy at the onset of inertial range is marked by the red symbol, and corresponds to $St = 0.24$. Reference scaling slopes of -1 (injection range) and -5/3 (inertial range) are shown.

we report the pressure solver's performance data for different number of nodes in Tab. 3, where for every number of nodes, we vary the number of processes that have access to the GPU. Simulations are performed on a grid size of $512 \times 512 \times 1536 = 402{,}653{,}184$, which is eight times higher than in our previous DNS reported in [3]. We deliberately set higher number of grid points to study the impact of excessive number of iterations which may occur at higher grid resolutions. The number of nodes (or GPUs) is varied from 8 to 256. We use the termination criterion given in Eq. (24) were we evaluate κ_ω to terminate the iterations. n_{norm} is set to 1 (that is, κ_ω is calculated in every iteration). Fig. 15, shows that for the present case, the GPU-based solver performs nearly two orders of magnitudes faster than the legacy CPU-based solver. Increasing the number of processes per node in the multiprocess mode increases the effective acceleration for the GPU-based solver for $\mathbf{N} \leq 32$. For a constant number of GPUs, S_{eff} might decrease slightly for lower amounts of data per MPI thread. It can be also observed that the lowest elapsed time for the CPU-based implementation was 10.63s for $n_{\text{process}} = 256 \times 12 = 3072$ MPI threads. Note that the smallest GPU-based case, using only 8 GPUs which are seeded by 3 CPU cores on each node (case S8-3-1), converges in 5.7s and thus still outperforms the largest CPU-based configuration S256-12-0 by a factor of two.

In Fig. 16, we plot the timings for fixed numbers of MPI threads per node, when the number of nodes (or GPUs) is varied, to obtain a clearer view of the strong scaling efficiency of the presented GPU-based solver, and to compare it to the legacy CPU-based solver. It is observed, that even though the elapsed time for the GPU-solver is on average two orders of magnitude lower, the strong

Case	$\frac{n=3}{g=0}$	$\frac{n=3}{g=1}$	$\frac{n=6}{g=0}$	$\frac{n=6}{g=1}$	$\frac{n=12}{g=0}$	$\frac{n=12}{g=1}$
S8-n-g	206.1	5.7	135.8	2.11	126.2	1.04
S16-n-g	98.4	1.6	80.9	0.68	119.37	0.69
S32-n-g	48.7	0.5	111	0.45	71	0.25
S64-n-g	98.1	0.35	52.6	0.16	30.1	0.11
S128-n-g	45.7	0.25	30.2	0.11	20.1	0.08
S256n-g	23.7	0.1	14.0	0.06	10.63	0.065

Table 3: Summary of a series of performance simulations for the benchmark case of flow around sphere at $Re_D = 3700$. All the cases are performed on the GPU nodes of a Cray XC40/50 (Intel® Xeon® E5-2690 v3 @ 2.60GHz (12 cores, 64GB RAM) and NVIDIA® Tesla® P100 16GB). Cases are tagged as "S**N**-n-g", where **N** is the number of nodes (or GPUs), n denotes the number of MPI processes per node, and g shows if the the GPU solver ($g = 1$) or the CPU-based MPI-parallel solver ($g = 0$) is used. Timings are given in seconds [s].

scalability is not preserved for the full range of number of nodes, as it was already observed for the Taylor-Green vortex simulation (cf. Fig. 8). Despite the lack of full linear scaling, the solver still presents a larger reduction in time to solution than the CPU-based legacy solver for $\mathbf{N} < 32$ and $n < 12$ (i.e., without hyperthreading). The results for the multigrid preconditioned GPU-based solver [3] are also shown in Fig. 16 for $n = 3$. It is observed that this implementation is approximately twice as fast as the CPU-based solver. The strong scaling efficiency is linear but not ideal for this case.

5.4. Effect of grid size on iteration behaviour using κ_ω termination criterion

Figure 17 shows the evolution of $||\mathbf{E}||$, i.e. the norm of the difference between two subsequent pressure approximations, and κ_ω for the iterative Jacobi solver with SOR. The left panel of this figure shows that $||\mathbf{E}||$ decreases rapidly towards values below 10^{-8}, and then it plateaus for all of the grid resolutions that have been analysed here. We expected this from the Fourier analysis (cf. Section 3.3), as G_ω takes values far below unity during initial iterations, which correspond to larger wave number modes. As iterations progress, G_ω approaches unity, which indicates slower decay of the error $||\mathbf{e}||$ and increment $||\mathbf{E}||$. The effect of grid resolution on the asymptotic behaviour of G_ω and $||\mathbf{E}||$, that is given in Eq. (19), is also shown: $||\mathbf{E}||$ approaches to zero faster for the coarsest grid resolution $\langle G_0 \rangle$ ($32 \times 32 \times 96$ grid points) than for the larger grid resolutions. Note that the grid resolution is increased uniformly in all three dimensions. The behaviour of κ_ω throughout the iterations is plotted on the right panel of Fig. 17. κ_ω approaches unity for the initial iterations and

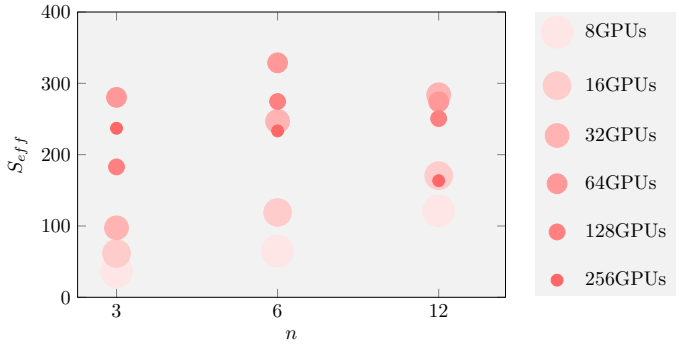


Figure 15: Effective accelerations obtained with the GPU solver, compared to the legacy CPU-based solver, are reported for the DNS of turbulent flow around a sphere at $Re_D = 3700$. n shows the number of MPI-processes on each node. At a fixed number of GPUs (or equivalently nodes), the number of active MPI threads on each node (n) is varied, and the acceleration is then measured in terms of the elapsed time in the Poisson solver for the CPU-based and GPU-based solvers.

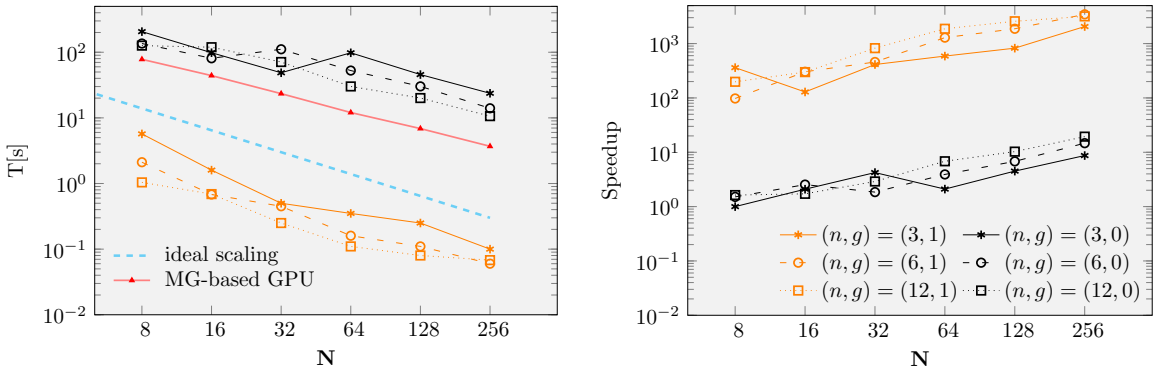


Figure 16: (Left) elapsed times for various node configurations (**N** denotes the number of nodes, or GPUs) are plotted where the number of MPS processes seeding the GPU per node (n) is varied ($g = 1$: the GPU-based solver, $g = 0$: the CPU-based legacy solver). Dashed blue line shows ideal scaling. The red line shows the results for the multigrid preconditioned GPU-based solver [3] with $n = 3$. (Right) relative speedups are shown for all configurations. Speedups are computed as T_0/T, where T is the elapsed time for a given configuration and T_0 is the elapsed time for the case S-8-3-0, that corresponds to the smallest configuration used for the CPU-based solver.

then eventually plateaus. Based on the Fourier analysis, we expect a smaller $\epsilon(M)$ for κ_ω at larger grid sizes, that is, a smaller distance from unity at the low wave number limit ($\underline{\mathbf{k}} \to 0_c$). A smaller distance is also observed before this limit is reached, which is verified by the results shown in the inset of the right panel of Fig. 17. An oscillatory behaviour is observed for the grid of size $64\langle G_0 \rangle$: κ_ω grows beyond unity instead of remaining below it. We conjectured in Section 3.3 that this is due to misrepresentation of κ_ω due to limited the arithmetic precision, and will not lead to divergence.

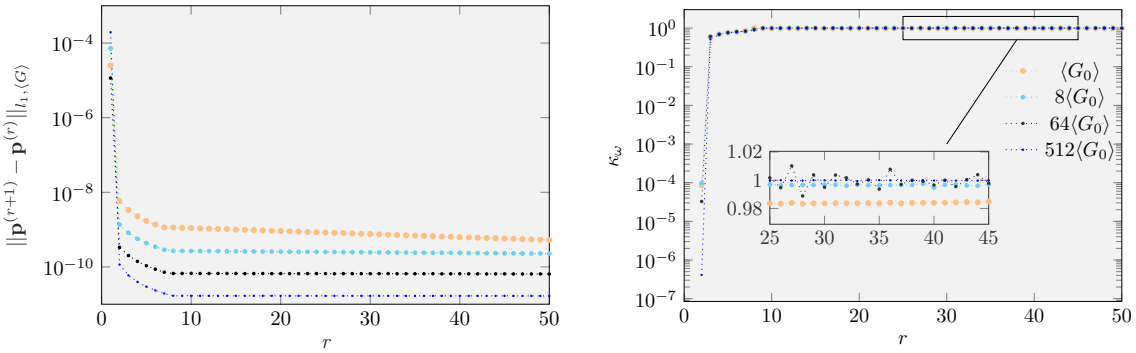
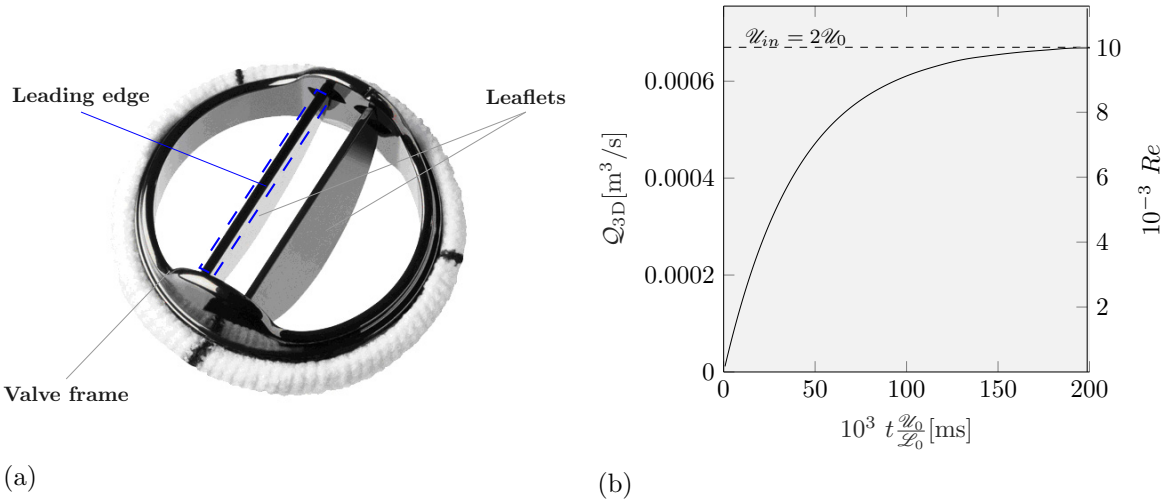


Figure 17: (Left) Progression of the grid-averaged l_1 norm of the increment between two subsequent iterations r and $r+1$ is plotted for 50 iterations. $\langle G_0 \rangle$ denotes the smallest grid size ($32 \times 32 \times 96$), and grid resolution in increased by the same factor in all three directions. (Right) Progression of κ_ω towards unity as iterations advance is illustrated for four grid sizes. The coarsest grid $\langle G_0 \rangle$ contains $32 \times 32 \times 96$ grid points, which corresponds to an average grid spacing of $\bar{h} = 1/32$. The inset shows that coarser grids are associated with lower values of κ_ω (larger distance from unity) for larger numbers of iterations. The visual flattening of κ_ω values is not in general a sign of convergence and this has to be checked using the $\epsilon(M)$ value.

6. Direct numerical simulation of systolic transition in flow past a bileaflet mechanical heart valve

6.1. Overview

In this section, we deploy the GPU-based solver for DNS of systolic flow past a model of a bileaflet mechanical heart valve (BMHV). It was recently shown using fully resolved 2D simulations that the laminar-turbulent transition process and the intensity of turbulent blood flow past the valve is largely influenced by the impinging leading-edge vortex instability near the leading edge of the leaflets [4]. This mechanism has slipped previous 3D simulations of this flow probably due to



(a) (b)

Figure 18: (a) Leading edge view of a Regent mechanical valve (`https://www.structuralheartsolutions.com`). The valve leaflets are hinged in a metal frame housing, which is sutured to the aortic root. (b) the flow rate waveform that has been used for systolic acceleration and the associated Reynolds number.

insufficient grid resolution. Thanks to the enhanced computational performance of the GPU-based solver, we investigate here the existence of this instability in a simplified 3D model of a BMHV.

We use the reference quantities $\mathscr{U}_0 = 0.75\mathrm{m/s}$, $\mathscr{L}_0 = 3 \times r_r = 36\mathrm{mm}$ and $\nu = 2.7 \times 10^{-6}\mathrm{m}^2/\mathrm{s}$. The velocity scale is one half of the inflow velocity at peak flow rate.

6.2. 3D model and boundary conditions

6.2.1. 3D aortic root and BMHV model

The 3D model is an extension of the 2D model used in [4]. Similar to [20] we fix the two leaflets of the heart valve in the fully open position (black in Fig. 19a). The angle of attack at this position is fixed at $90 - \theta = 5°$. The sinuses of Valsalva are modelled as three spherical cavities with a radius $r_s = r_r\sqrt{3}/2$ (Fig. 19b, bottom). The centers of these spheres are located on the $x = 0$ plane, and on the sides of the equilateral triangle, to which the aortic root's cross section is circumscribed. Leaflets are modelled as blunt plates with triangular leading, and trailing edges(Fig. 19a). The spanwise profile of the leaflets is given on the top panel of Fig. 19b.

6.2.2. Flow forcing

The flow is smoothly accelerated from zero to the mean velocity $\mathscr{U}_{in} = 2\mathscr{U}_0$ at $t = 200\mathrm{ms}$ (Fig. 18b). No-slip boundary conditions are imposed on the rigid valve leaflets and aortic root boundaries.

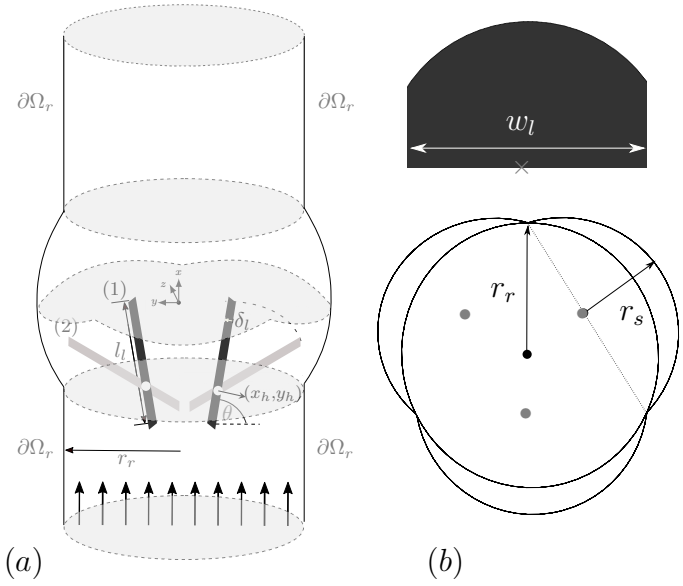


Figure 19: (a) 3D model of the BHMV in the aortic root. The 2D cross-sections of leaflets in their plane of symmetry are shown in (1) open (dark gray) and (2) closed (light gray) positions. All geometrical parameters are given in Tab. 4. (b, bottom) Cross-section of the geometry at $x = 0$. The centers of the spherical cavities representing the sinuses of the Valsalva (three gray solid points) are placed on the middle of the sides of the equilateral triangle, to which the aortic root cross section is circumscribed ($r_s = r_r\sqrt{3}/2$). (b, top) geometry of the leaflet in the spanwise direction: the elliptical leading edge of the leaflet is modelled as a semicircle whose center is located on the axis of symmetry and on the leading edge (gray cross mark) and its radius is equal to l_l.

Parameter	**Notation**	**Value**
root radius	r_r	12mm
sinus radius	r_s	$0.86r_r$
leaflet length	l_l	$1.15r_r$
leaflet leading edge width	w_l	$1.963r_r$
leaflet thickness	δ_l	$0.09r_r$
hinge longitudinal position	x_h	$-1.062r_r$
hinge radial position	y_h	$-0.19r_r$

Table 4: Geometrical parameters of the three-dimensional model.

Periodic boundary conditions are used in the flow direction, and the systolic waveform is forced using a fringe region technique [21] upstream of the valve:

$$\widetilde{\mathbf{f}} = \lambda(x)(\widetilde{\mathbf{u}} - \widetilde{\mathbf{u}}_{\mathbf{0}}(t)), \tag{38}$$

where $\lambda(x)$ is the fringe function which vanishes beyond the fringe region and $\widetilde{\mathbf{u}}_{\mathbf{0}}(t)$ is a uniform flow profile. The amplitude of $\widetilde{\mathbf{u}}_{\mathbf{0}}(t)$ and the amplitude of the fringe function $\lambda(x)$ are tuned *ad hoc* to yield the desired systolic acceleration (Fig. 18b). The fringe forcing enforces the given inflow profile, while it simultaneously damps out the outflow disturbances reentering the domain at the inflow, due to the periodic boundary conditions.

6.2.3. Configuration of the numerical experiment

We present a highly resolved numerical simulation using 337,644,801 grid points in a cuboid domain of size $3r_r \times 3r_r \times 15r_r$, which allows accurate representation of the non-conforming geometry of the leaflets, and also resolves the spatio-temporal instability waves and their interactions. This resolution is roughly 30 times higher than the resolution used for the fine mesh BMHV simulations of [5]. Grid stretching is applied in all three directions to place more grid points near the leaflets, so that a similar grid resolution to that reported in [3] is achieved near the leaflets. Accordingly, 31 grid points are placed along the leaflet thickness δ_l in the y direction, providing a grid spacing of $\tilde{h}_y = 35\mu$m. Same resolution is utilized for the streamwise direction x, while half this resolution is used in the z direction (that is, a grid spacing of $\tilde{h}_z = 70\mu$m). Less resolution is set in the spanwise direction z because the leaflet leading-edge geometry is uniform in this direction. To the best of our knowledge, this is the highest resolution that has ever been used for three-dimensional numerical simulation of heart valve hemodynamics.

The simulations were completed in only three days using the novel GPU-based solver on 20 GPUs. A small pilot run revealed that the GPU implementation is approximately 150 times faster than the legacy CPU-based flow solver on the same node configuration (20 nodes of Cray XC40/50, Piz Daint), therefore, it can be said that the equivalent simulation on equivalent number of CPU cores ($20 \times 16 = 320$ cores with hyper-threading) would have taken approximately 1.5 years to complete.

As a result of the large grid resolution, the size of each data output of the velocity field amounted in size to 8.1GBs. Nearly 750 instantaneous flow field datasets (velocity field, pressure, and λ_2 values), was generated resulting in a total of 9.2TB of data for the physical time of 0.2s (approximately

20% of heartbeat). The large amount of generated data required the use of parallel visualization techniques, for which up to 4 nodes of the Cray XC40/50 supercomputer were used.

6.3. 3D ILEV structures in the valve model

Figure 20 shows an snapshot of the streamwise velocity of the turbulent flow field past the 3D BMHV model after the turbulent breakdown in the wake had taken place. The instantaneous streamwise velocity is shown for the centre-plane parallel to the valve leaflets ($y = 0$). Moreover, cross-sectional views are given for four different streamwise locations denoted by $x_{1,2,3,4}$. Cross-section x_1 provides a view of the 3D instability of the ILEV structures between the leaflets. Cross-section x_2 which cuts through the elliptical part of the trailing edge of the leaflets, indicates a larger area of oscillatory flow compared to cross-section x_1, which is in agreement with findings of 2D simulations [4]. Signatures of the small-scale flow structures in the bulk flow and close to the centerline indicate the influence of the vorticity waves (generated by the ILEV instability mechanism on the wake flow). This is also reflected in Fig. 21 where the evolution of the vortex structures around the valve leaflets shows the significant production of small-scale structures in the wake upon interaction with ILEV-induced flow oscillations. This figure also shows that the wake flow consists of relatively large-scale laminar vortex structures prior to interaction with disturbances coming from the ILEV zone, e.g at time $t = 0.5$ (24ms in dimensional form).

6.4. Effect of the grid resolution on resolving the ILEV zone

It was shown in [4] that high resolutions are required to resolve the ILEV velocity profiles in 2D. This is shown here for the 3D model, although no finer grid than that used for the presented DNS (referred to as case M1) is analysed due to prohibitively high computational cost. We performed three coarser simulations where we coarsened the grid by a factor 2 in all directions (case M2), by a factor 4 in x direction and a factor 2 in y and z directions (case M3), and by a factor 4 in x and y directions and a factor 2 in z direction (case M4). Velocity profiles in the y direction and within the central orifice area (i.e., the area between the leaflets) for these cases are shown in Fig. 22. Profiles were taken 2.08mm downstream of the leading edge at two spanwise locations $z = 0$ (center of the leaflet span) and $z = -9.1$mm (closer to the hinge area). Location $z = 9.1$mm (not shown here) is equivalent to $z = -9.1$mm due to symmetry. The streamwise location was selected within the laminar part of the ILEV. All profiles were taken at $t\mathscr{L}_0/\mathscr{U}_0 = 72$ms, after ILEV zone was sufficiently developed. Little difference can be seen between case M1 and M2. Case M4 which has the lowest resolution in the y direction shows almost no reverse flow, which emphasizes the

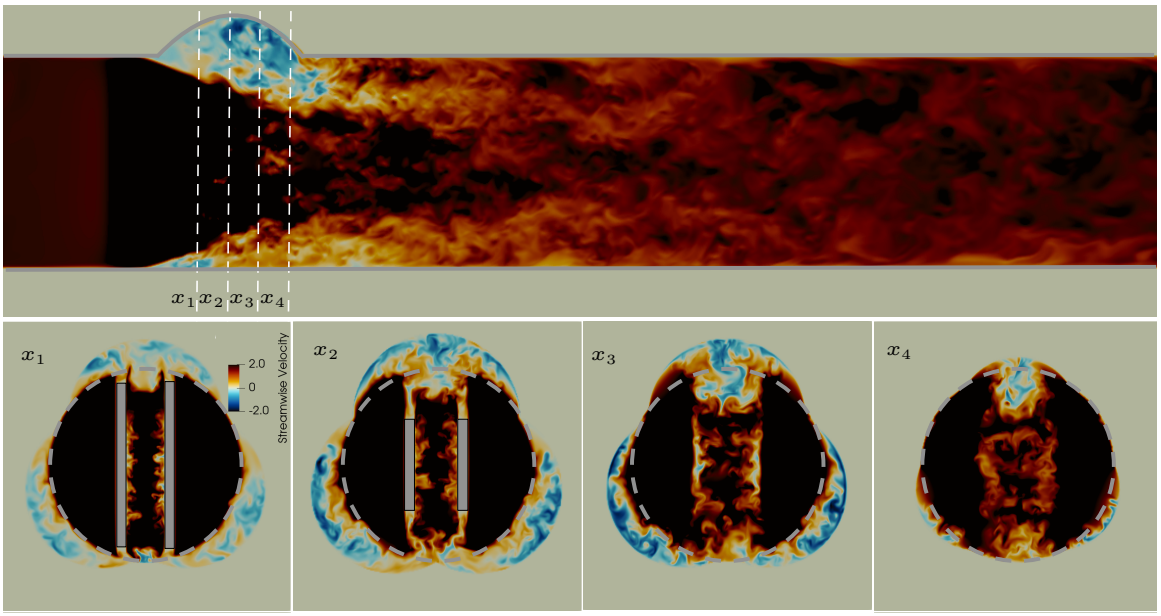


Figure 20: Fully resolved turbulent flow past an open BMHV. (Top) streamwise velocity field on a plane passing through the central orifice on the valve model on the axis of symmetry without cutting through the leaflets. (Bottom) cross-sections of the streamwise velocity are shown at $x_1 = -0.75r_r$, $x_2 = -0.45r_r$, $x_3 = -0.15r_r$ and $x_4 = 0.15r_r$. Signature of ILEV instability is seen between the valve leaflets for the cross-sections x_1 and x_2. Influence of ILEV instabilities on the wake flow can be observed in cross-sections x_3 and x_4.

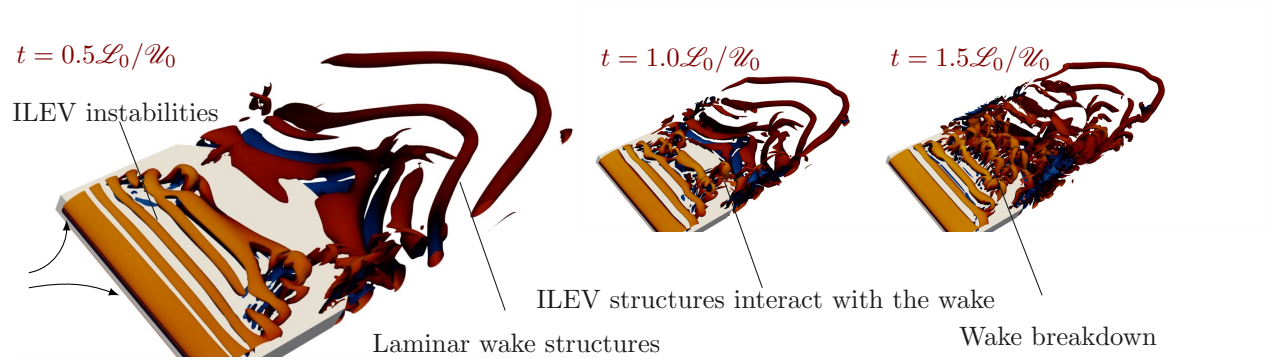


Figure 21: ILEV instabilities trigger the transition to turbulence in the wake of the BMHV. Evolution of Lagrangian coherent structures (iso-surfaces of $\lambda_2 = -0.1$) are represented around one valve leaflet (gray solid structure) for times $t = 0.5\mathscr{L}_0/\mathscr{U}_0$ (before wake breakdown), $t = 1.0\mathscr{L}_0/\mathscr{U}_0$ (onset of wake breakdown) and $t = 1.5\mathscr{L}_0/\mathscr{U}_0$ (after the wake breakdown). Light red and blue show maximum positive and minimum negative streamwise velocities, respectively.

significance of the resolution in this direction. Comparing the cases M2 and M3 shows that the streamwise resolution is also important in resolving the reverse flow profiles in the ILEV zone.

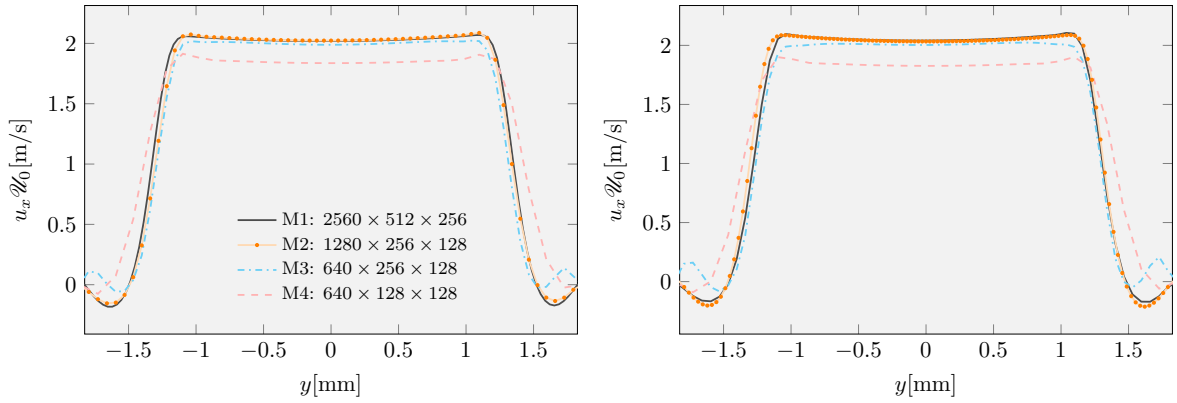


Figure 22: Effect of grid resolution on streamwise velocity profiles in the y direction within the laminar part of the ILEV zone at time $t\mathscr{L}_0/\mathscr{U}_0 = 72$ms. Four grid resolutions, including the reported DNS case M1 are shown 2.08mm downstream of the leading edge at (left) $z = 0$ and (right) $z = -9.1$mm spanwise locations.

6.5. Effect of κ_ω and κ_{ω,m_0} termination criteria on weak scaling

We use the current case to study the effect of the termination criterion using κ_ω (Eq. (24)) and κ_{ω,m_0} (Eq. (28)) at elevated resolutions. This check is motivated by the suspicion that larger grid resolutions may result in significantly smaller $\epsilon(M)$, thus render the convergence more difficult to achieve and reduce the computational performance. To this end, we increase the grid size and simultaneously the compute node resources (weak scaling). The largest problem that we address here comprises 21.5B grid points ($10240 \times 2048 \times 1024$), which was benchmarked on 1280 P100 GPUs seeded by 8 CPU cores per node. We conducted performance benchmarking simulations for both CPU-based legacy and GPU-based solvers. We collected the residual data for both cases and reported the ratio of the sum of the residuals that are obtained from solving the pressure equation in each substep of the RK3 time integration scheme. The ratio of the residual obtained for the GPU-based solver ($\epsilon^+_{\infty,1}$) over that for the CPU-based preconditioned Krylov-based solver ($\epsilon^+_{\infty,0}$) is plotted on the left panel of Fig. 23. The ratio remains near unity, and for largest grid sizes, a lower termination residual is obtained for the GPU-based solver than for CPU-solver. For the largest grid adopted here, the GPU-based pressure solver converged on average in 350 iterations which is nearly four times that of the lowest grid resolution. The elevated number of iterations reveals that the κ_ω

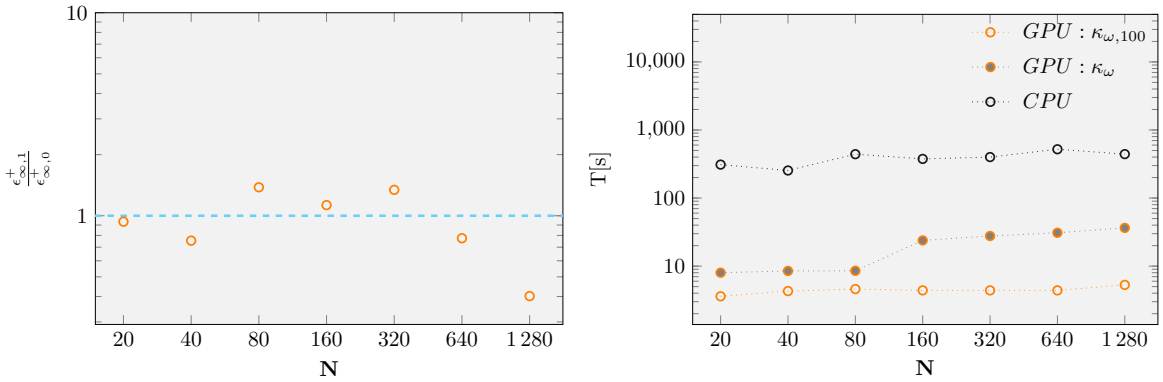


Figure 23: (Left) ratio of the total residuals ϵ_∞ when the GPU solver ($\epsilon^+_{\infty,1}$) is terminated with the κ_ω criterion versus that for the CPU-based solver ($\epsilon^+_{\infty,0}$, which terminates the iterations based on the error within the preconditioned BiCGstab method). **N** denotes the number of nodes or GPUs. The + superscript shows that the residual is sum of the three substeps of the Runge-Kutta scheme. (Right) weak scalability of the present solver when $n_{\text{norm}} = 100$, i.e. when the termination criterion is checked every 100 iterations ($\kappa_{\omega,100}$), compared against the scalability when the norm is checked every iteration (κ_ω).

termination criterion can harm the performance for larger grid sizes. The timings are then measured when $\kappa_{\omega,100}$ termination criterion was used. Right panel of Fig. 23 shows the elapsed times for κ_ω and $\kappa_{\omega,100}$ termination criteria, together with the timings for the legacy CPU-based solver. It shows that $n_{\text{norm}} = 100$ exhibits better effective acceleration and overall weak scaling than $n_{\text{norm}} = 1$. This observation emphasizes the fact that, as the iteration count increases, weak scaling efficiency deteriorates due to more intensive host to device data copies for larger grid sizes. However, if the termination criterion is checked once in every 100 iterations, the weak scaling is preserved. Even at the smallest grid sizes studied here, for which the number of iterations in the pressure solver is usually less than 100, using $n_{\text{norm}} = 100$ nearly doubles the performance, even though, the number of iterations may increase two-fold. As a result the final residual after termination drops to values even lower than those reported on the left panel of Fig. 23, which is favourable for the accuracy of the flow solver.

7. Conclusion

A novel high-throughput Poisson solver is developed through combining task and data parallelism for scale-resolving solutions of the incompressible Navier-Stokes equations. The solver resorts

to geometric domain decomposition techniques for task parallelism, while it uses CUDA for data parallelism. The classical Jacobi method with successive over-relaxation is used in combination with a scheme to minimize data copies arising from task parallelism. The hybrid implementation yields excellent computational performance and was shown to outperform a legacy CPU-based MPI-parallel solver using preconditioned Krylov-based methods by factors exceeding 300. Moreover, the electrical energy consumption of hardware infrastructure was decreased by approximately 70%. The Poisson solver was integrated in a high-order time-integration scheme for incompressible Navier-Stokes equations, which was formerly ported to GPUs.

The flow solver was tested for physical validity and computational performance using two flow benchmarks: Taylor-Green vortex transition to turbulence and decay at $Re = 1600$ and turbulent flow past a rigid sphere at $Re_D = 3700$. The flow statistics obtained from both benchmarks agreed well with corresponding data in the literature. For the first benchmark, the solver showed linear strong scaling when each GPU was seeded by only one CPU core. The second benchmark showed larger reductions in time to solution compared to the first, but the strong scaling was not linear.

Finally, the solver was deployed for a scale-resolving simulation of systolic laminar-turbulent transition in a 3D model of of a bileaflet mechanical heart valve. The results illustrated the systolic laminar-turbulent transition process involving impinging leading edge vortex instabilities [4]. This simulation, using 337,644,801 grid points, was completed in three days using 20 GPUs of Cray XC40/50 system. The same simulation using an optimized CPU-based solver would have taken an estimated 1.5 years to complete on 20 nodes of the same system.

The present study shows the suitability of GPUs for significantly faster simulations of fluid flows including elliptic governing equations, which has been shown previously for fluid flows governed by hyperbolic equations.

Acknowledgements

HZ and DO would like to thank the Platform for Advanced Scientific Computing (PASC) for funding this work through the AV-FLOW and HPC-PREDICT projects. Both authors would also like to thank the Swiss National Supercomputing Centre (CSCS) for providing technical support and GPU-node resources on the Cray XC40/50 supercomputer *Piz Daint*. HZ would like to additionally acknowledge the financial support from Swiss National Science Foundation (SNSF) through the Early PostDoc Mobility Fellowship P2BEP2_191786.

References

[1] R. Henniger, D. Obrist, L. Kleiser, High-order accurate solution of the incompressible Navier–Stokes equations on massively parallel computers, Journal of Computational Physics 229 (10) (2010) 3543–3572.

[2] A. Brüger, B. Gustafsson, P. Lötstedt, J. Nilsson, High order accurate solution of the incompressible navier–stokes equations, Journal of Computational Physics 203 (1) (2005) 49–71.

[3] H. Zolfaghari, B. Becsek, M. G. Nestola, W. B. Sawyer, R. Krause, D. Obrist, High-order accurate simulation of incompressible turbulent flows on many parallel gpus of a hybrid-node supercomputer, Computer Physics Communications 244 (2019) 132–142.

[4] H. Zolfaghari, D. Obrist, Absolute instability of impinging leading edge vortices in a submodel of a bileaflet mechanical heart valve, Physical Review Fluids 4 (12) (2019) 123901.

[5] I. Borazjani, L. Ge, F. Sotiropoulos, Curvilinear immersed boundary method for simulating fluid structure interaction with complex 3d rigid bodies, Journal of Computational Physics 227 (16) (2008) 7587–7620.

[6] A. A. Wray, Very low storage time-advancement schemes, Internal Rep. NASA-Ames Research Center, Moffett Field, CA (1986).

[7] S. Le Borne, Hierarchical matrix preconditioners for the oseen equations, Computing and Visualization in Science 11 (3) (2008) 147–157.

[8] S. Le Borne, Block computation and representation of a sparse nullspace basis of a rectangular matrix, Linear Algebra and its Applications 428 (11-12) (2008) 2455–2467.

[9] H. A. Van der Vorst, Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems, SIAM Journal on Scientific and Statistical Computing 13 (2) (1992) 631–644.

[10] E. Cormie-Bowins, A comparison of sequential and gpu implementations of iterative methods to compute reachability probabilities, arXiv preprint arXiv:1210.6412 (2012).

[11] G. D. Smith, Numerical solution of partial differential equations: finite difference methods, Oxford University Press, 1985.

[12] C. Hirsch, Numerical computation of internal and external flows, Volume 1: Fundamentals of numerical discretization, Vol. 9, John Wiley and Sons, 1988.

[13] W. M. Van Rees, A. Leonard, D. Pullin, P. Koumoutsakos, A comparison of vortex and pseudo-spectral methods for the simulation of periodic vortical flows at high reynolds numbers, Journal of Computational Physics 230 (8) (2011) 2794–2805.

[14] J. Jeong, F. Hussain, On the identification of a vortex, Journal of Fluid Mechanics 285 (1995) 69–94.

[15] I. Rodriguez, R. Borell, O. Lehmkuhl, C. D. P. Segarra, A. Oliva, Direct numerical simulation of the flow over a sphere at Re = 3700, Journal of Fluid Mechanics 679 (2011) 263–287.

[16] S. Liska, T. Colonius, A fast immersed boundary method for external incompressible viscous flows using lattice green's functions, Journal of Computational Physics 331 (2017) 257–279.

[17] H. Kim, P. Durbin, Observations of the frequencies in a sphere wake and of drag increase by acoustic excitation, The Physics of Fluids 31 (11) (1988) 3260–3265.

[18] H. Zolfaghari, D. Izbassarov, M. Muradoglu, Simulations of viscoelastic two-phase flows in complex geometries, Computers & Fluids 156 (2017) 548–561.

[19] R. Mittal, H. Dong, M. Bozkurttas, F. Najjar, A. Vargas, A. Von Loebbecke, A versatile sharp interface immersed boundary method for incompressible flows with complex boundaries, Journal of Computational Physics 227 (10) (2008) 4825–4852.

[20] A. Bellofiore, E. M. Donohue, N. J. Quinlan, Scale-up of an unsteady flow field for enhanced spatial and temporal resolution of piv measurements: application to leaflet wake flow in a mechanical heart valve, Experiments in Fluids 51 (1) (2011) 161–176.

[21] J. Nordström, N. Nordin, D. Henningson, The fringe region technique and the fourier method used in the direct numerical simulation of spatially evolving viscous flows, SIAM Journal on Scientific Computing 20 (4) (1999) 1365–1393.