Oral presentation | Multi-phase flow
High performance computing-I
Tue. Jul 16, 2024 4:30 PM - 6:00 PM Room D

[6-D-03] Task-decomposed Overlapped Pressure Preconditioner for Sustained Strong Scalability on Accelerated Pre-exascale Systems

*Niclas Jansson[1], Martin Karp[1], Szilárd Páll[1], Stefano Markidis[1], Philipp Schlatter[2,1] (1. KTH Royal Institute of Technology, 2. Friedrich-Alexander-Universität (FAU) Erlangen–Nürnberg)

Keywords: Accelerators, Spectral element method, Direct numerical simulation

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

Task-decomposed Overlapped Pressure Preconditioner for Sustained Strong Scalability on Accelerated Pre-exascale Systems

N. Jansson*, M. Karp*, S. Páll*, S, Markidis* and P. Schlatter**,*
Corresponding author: njansson@kth.se
* KTH Royal Institute of Technology, Sweden.
** Friedrich-Alexander-Universität (FAU) Erlangen–Nürnberg, Germany.

Abstract: We detail our work on improving the performance and scalability of key numerical methods in the high-fidelity spectral element code Neko on accelerated pre-exascale machines. Efficient preconditioners are essential in incompressible fluid dynamics; however, the most efficient method (with respect to convergence) might be challenging to implement with good performance on an accelerator. We present our development of a GPU-optimised preconditioner with task overlapping for the pressure-Poisson equation, reducing the execution time of the preconditioner by almost 20%. The new preconditioner is explained in detail, together with a performance study on the pre-exascale supercomputer LUMI.

Keywords: Accelerators, Spectral element method, Direct numerical simulation.

1 Introduction

Computational Fluid Dynamics (CFD) is a natural driver for exascale computing with a virtually unbounded need for computational resources for accurate simulation of turbulent fluid flow, both for academic and engineering usage. However, with exascale computing capabilities on the horizon, we have seen a transition to more heterogeneous computer architectures. Traditional homogeneous scalar processing machines are replaced with heterogeneous machines that combine scalar processors with various accelerators, such as GPUs. While offering high theoretical peak performance and high memory bandwidth, complex programming models and significant programming investments are necessary to efficiently exploit these systems. Furthermore, most known pre and exascale systems currently planned or installed, e.g., Frontier, LUMI, and JUPITER, contain a large fraction of accelerators. Thus, the challenge of porting and tuning scientific codes for these new platforms can no longer be ignored.

One of the challenges in porting CFD codes to accelerators is to achieve good performance while retaining the efficiency of a given numerical method, e.g., convergence rates. Efficient numerical methods, that perform well on CPUs, might have algorithmic features that renders them very difficult to implement on a GPU in a performant way. This is especially true, if disruptive changes to the method, aimed at improving performance, could have a negative impact on the numerical properties. Therefore, the challenge of porting established CFD codes to accelerators involves not only GPU porting and hardware-specific tuning, but also formulation of suitable numerical methods that could efficiently utilise a GPU without sacrificing good numerical properties.

In this paper, we describe our work on developing an efficient preconditioner for the pressure-Poisson equation in incompressible flow. The new preconditioner is designed to work well on modern accelerators, and utilises a novel parallelisation based on a task-decomposed formulation, allowing for increased GPU utilisation and improved strong scaling characteristics. Using the new formulation, managed to near-perfect strong scaling of our spectral element code Neko on up to 80% of LUMI, currently number 5 on the June 2024 Top500 [1].

The outline of this paper is the following; In Section 2 a description of the spectral element flow solver Neko is given, following by a description of the new preconditioner formulation and its parallelization in Section 3. A performance evaluation and conclusions outlining future work are given in Sections 4 - 5.

[1] `https://top500.org/lists/top500/2024/06/list`

2 Neko

To address the exascale computing challenge and to enable performant high-fidelity fluid simulations across various heterogeneous platforms, we have developed Neko, a portable framework for high-order spectral element-based simulations, mainly focusing on incompressible flow [1]. The spectral element method combines the geometrical flexibility of the finite element method with the accuracy of spectral methods, keeping the exponential decay of the error as the polynomial order increases, making it an ideal method for direct numerical simulation of turbulence with complex geometries. Neko is written in modern Fortran 2008 and parallelized using MPI, and adopts an object-oriented approach allowing for multi-tier abstractions of the solver stack, facilitating various hardware backends ranging from general-purpose processors, accelerators and vector processors [2, 3]. For accelerators, Neko uses a device abstraction layer to manage device memory, data transfer, and kernel launches directly from Fortran. Behind this interface, Neko calls the native accelerator implementation written in, e.g., CUDA, HIP or OpenCL.

2.1 Numerical Method

Neko advances the incompressible Navier–Stokes equations in time,

$$\begin{aligned} \frac{\partial u}{\partial t} + (u \cdot \nabla)u &= -\nabla p + \frac{1}{Re}\nabla^2 u + f, \\ \nabla \cdot u &= 0, \end{aligned}$$

where u is the velocity, p the pressure, f a volume force and the Reynolds number $Re = UL/\nu$, with the reference velocity and length U and L and kinematic viscosity ν. The solver is based on conformal function spaces for both the pressure and momentum equation based on schemes developed by Orszag, Israeli, Deville, Karniadakis and Tomboulides [4, 5, 6]. Time integration is performed using an implicit-explicit scheme based on backward differentiation and k-th order extrapolation,

$$\sum_{j=0}^{k} \frac{b_j}{dt} u^{n-j} = -\nabla p^n + \frac{1}{Re}\nabla^2 u^n + \sum_{j=1}^{k} a_j \left(u^{n-j} \cdot \nabla u^{n-j} + f^n \right),$$

with b_k and a_k coefficients of the implicit-explicit scheme. Thus the discrete system to solve for a time-step n becomes,

$$\begin{aligned} \Delta p^n &= \nabla \cdot \left(\sum_{j=1}^{k} a_j \left(u^{n-j} \cdot \nabla u^{n-j} + f^n \right) \right), \\ \frac{1}{Re}\Delta u^n - \frac{b_0}{dt} u^n &= \nabla p^n + \sum_{j=1}^{k} \left(\frac{b_j}{dt} u^{n-j} + a_j \left(u^{n-j} \cdot \nabla u^{n-j} + f^n \right) \right). \end{aligned}$$

In each time step, we solve one Poisson equation to obtain the pressure using extrapolated velocities on the boundaries, followed by a Helmholtz equation for each velocity component. All systems are solved using iterative Krylov subspace methods. A preconditioned Conjugate Gradient (CG) method is used for velocity with a block Jacobi preconditioner. For the Poisson equation, we use a preconditioned Generalised Minimal Residual Method (GMRES).

3 Scalable Pressure Preconditioner

An efficient precondition for the pressure is essential in incompressible fluid dynamics, as the Poisson equation that arises as a consequence of incompressibility is the main source of stiffness when computing a solution. Neko uses a two-level additive overlapping Schwarz multigrid method [7, 8] combined with a coarse grid solver to precondition the linear system,

$$M_0^{-1} = R_0^T A_0^{-1} R_0 + \sum_{k=1}^{K} R_k^T \tilde{A}_k^{-1} R_k. \tag{1}$$

For a general k-level formulation, R_k and R_k^T are the restriction and prolongation operations to move between different grid levels. The coarse grid problem A_0, on linear elements, is solved for using an approximate Krylov solver, a preconditioned CG method, with a fixed number of iterations (≈ 10) and an element-wise block Jacobi preconditioner. This has proven to be both an effective and scalable preconditioner on both CPUs and GPUs [2]. However, in terms of computational efficiency, the coarse grid's smaller problem size will reduce the preconditioners ability to fully utilize the GPU. The reduced computational efficiency is illustrated in Figure 1, showing a trace timeline from a GPU execution, with the coarse grid solver marked by the light yellow box (`HSMG coarse grid`) in the NVTX region, which overlaps with a region of low GPU utilization, as indicated by the top part of Figure 1 showing GPU HW activity.

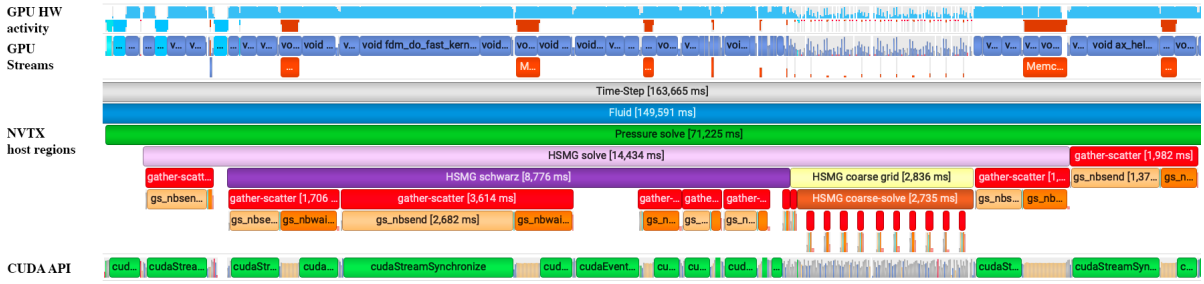


Figure 1: A trace timeline view of the additive Schwarz preconditioner executing on an Nvidia A100. The top part shows the GPU HW streams activity and the lower half of the figure shows the NVTX regions and CUDA API calls.

3.1 Task-decomposed Formulation

To increase the computational efficiency, we need to change the numerical method, either using a different coarse grid solver or changing the entire preconditioner to one that can better utilize accelerators (while retaining a similar convergence rate). Another approach is to solve the coarse grid problem on the CPU. However, since the coarse grid solver performs communication inside each Krylov iteration (e.g. inner productions and reductions), data transfer from GPU memory to CPU memory quickly becomes a bottleneck, particularly for architectures without a high bandwidth link between CPU and GPU memory, or shared global memory.

Since Neko's pressure preconditioner (Eq. 1) uses an additive formulation, it is possible to decouple the coarse grid solve (first part of the right-hand side in Eq. 1) from the rest of the multigrid solver, and compute both parts concurrently. To increase GPU utilization, amortize communication costs, and thus improve strong scalability, we have developed a new parallelization method of the additive Schwarz preconditioner. The new formulation aims at exploiting all available task parallelism by launching the left and right parts of the right-hand side of Eq. 1 in parallel on the accelerator. Tasks are launched in separate streams to allow overlapping work and increase GPU utilization. Since the coarse grid's work is dominated by kernel launch latency due to its many small device kernels, launching GPU work in parallel allows for hiding this launch latency. Furthermore, it also exposes more concurrency on the GPU, allowing kernels and data movement to be overlapped on different streams, particularly the typically short coarse-solve kernels (too small to keep the GPU utilization high) with other larger kernels. To maximize kernel overlap and ensure progress on both streams, we use stream priorities and assign higher priority to the stream where the coarse solve is launched. Synchronization between dependencies in different streams and between host and streams is performed using CUDA Events, allowing for fine-grained synchronisation without waiting for all work on the GPU to finish before, e.g., initiating MPI communication.

Figure 2 shows the effect of the new task-parallel formulation for a small test case running on two Nvidia A100 GPUs connected via PCIE. The coarse grid solver (yellow box in the NVTX region) is now entirely overlapped by the right part of the right-hand side in (1), the purple box in the NVTX region (`HSMG schwarz`). Comparing the trace timelines in Figure 1 and 2 we see that the execution time for the coarse grid part has increased significantly (almost 138%) since it has to compete for resources with the Schwarz part (right side of the right-hand side of (1)), which has also slightly increase its execution time. However, since both coarse grid and Schwarz overlap and execute concurrently, we see that the

Figure 2: A trace timeline view of the new task-parallel additive Schwarz preconditioner executing on an Nvidia A100. The top part shows the GPU HW streams activity and the lower half of the figure shows the NVTX regions and CUDA API calls.

total time for the entire pressure preconditioner has reduced by almost 20% (`HSMG solve` in the NVTX region) using the new task-parallel formulation.

4 Performance Evaluation

To assess the performance of the new task-decomposed overlapped pressure preconditioner, we carry out a performance evaluation against the current preconditioner implemented in Neko. The performance measurements were performed on the EuroHPC JU pre-exascale supercomputer LUMI, hosted by CSC in Finland. LUMI is a 380 PFlop/s HPE CRAY EX equipped with AMD MI250X GPUs. A detailed list of the experimental platform is given in Table 1.

System	**LUMI**
Computing device	AMD MI250X
Peak TFlop FP64/s	47.9 (95.7 Matrix)
Peak BW/s	3300
No. devices	11912
Interconnect	HPE Slingshot 11
	200 GbE NICs (4x200 Gb/s)
MPI	Cray MPICH 8.1.27
Compiler	CCE 16.0.1
GPU Driver	5.16.9.22.20
CUDA/ROCm	ROCm 5.2.3

Table 1: Hardware and software details of our experimental platform LUMI.

The new preconditioner formulation has previously been proven to significantly improve Neko's strong scalability characteristics. In [9], Neko was used to compute Rayleigh–Bénard convection (RBC) at extreme-scale on, among other systems, LUMI. The results in Figure 3, for a mesh with 108M elements and 7th-order polynomials for simulations at a Rayleigh number $Ra = 10^{15}$, show that Neko, using the new preconditioner formulation, achieves close to perfect parallel efficiency on up to 80% of the then, April 2023, full machine capacity [9]. Furthermore, it also demonstrated sustained strong scalability with as few as 7000, 7th order elements per GPU ($\approx$ 3.6M degrees of freedom (DoF)) compared to previously reported results [10, 2] where close to 15000, 7th order elements ($\approx$ 7.7M DoFs) were needed to sustain ideal strong scalability.

In this paper, we focus our performance measurements on the pressure solver to assess the impact of the new preconditioner formulation, in relation to the observed improved strong scaling results. For the experiments, we use the Taylor-Green Vortex (TGV) test case with Reynolds number $Re = 1600$, in a

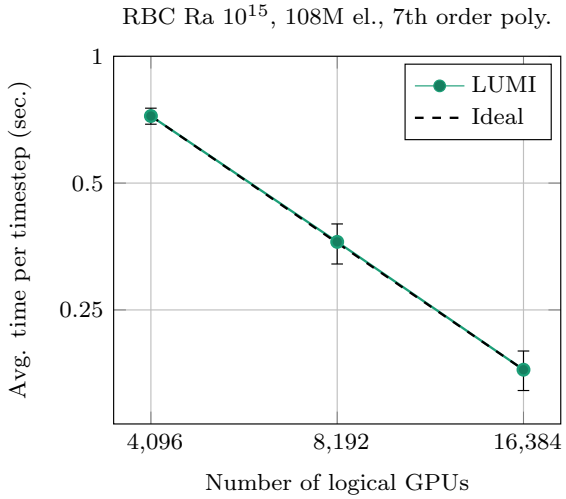


Figure 3: Strong scaling of Neko for the RBC case from [9], with performance measured in average time per time-step. A logical GPU refers to one MI250X Graphics Compute Die (GCD). Perfect strong scaling is illustrated as a black dashed line, and the 99% confidence intervals is illustrated as error bars.

box with side length 2π and periodic boundaries, with the initial conditions:

$$\begin{aligned} u_x(x,y,z) &= \sin(x)\cos(y)\cos(z), \\ u_y(x,y,z) &= -\cos(x)\sin(y)\cos(z), \\ u_z(x,y,z) &= 0. \end{aligned}$$

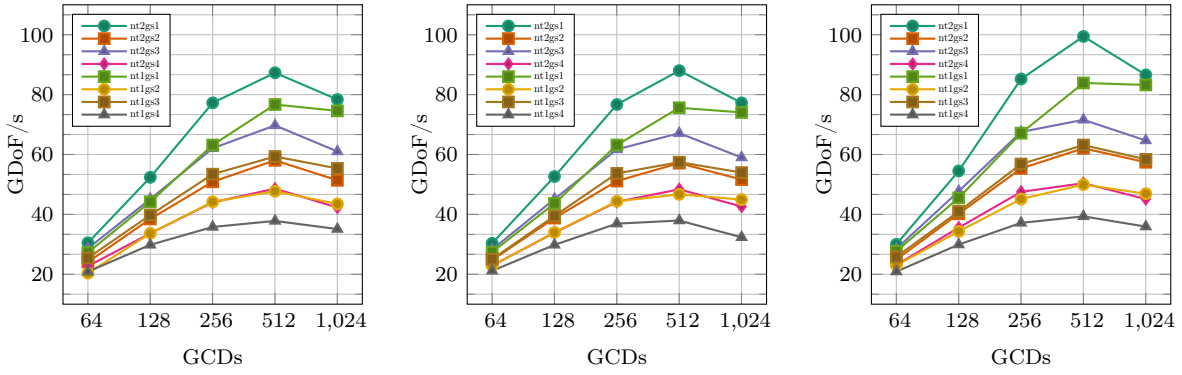
We ran the experiments using two different meshes, one with 128^3 elements and one with 256^3 elements, both with a polynomial degree of 7, leading to roughly 1G and 8.5G DoFs respectively. Performance was measured for three different coarse grid solvers: a standard preconditioned Conjugate Gradient method (standard PCG), a preconditioned pipelined Conjugate Gradient method (pipelined PCG) and a standard PCG optimized for accelerators using kernel fusion to reduce the number of small kernel launches during the coarse grid solve (Fused kernels PCG). For each mesh, and solver combination we tested the standard non-overlapped and the new overlapped preconditioner formulation, and four different strategies for gather-scatter, a key component for an efficient matrix-free based spectral element implementation.

Neko's gather-scatter kernel implements four different communication strategies for systems with accelerators and support for device-aware MPI. Each pack and unpack operation has been designed to support two modes of operation, either working on all data to be sent or received or only operating on parts of the data related to one neighbouring rank. This allows Neko to either launch pack and/or unpack operations concurrently in different GPU streams (one for each neighbouring MPI rank) or, if GPU API overheads (such as launch latencies and event handling) are too costly, wait for all communication to finish before unpacking all data. We refer to [1, 10] for more details on the gather-scatter implementation.

The smaller mesh was tested on eight up to 128 nodes, corresponding to 64 to 1024 AMD MI250X Graphics Compute Dies (GCD), and the larger mesh on 64 up to 1024 nodes, corresponding to 512 to 8192 GCDs. For each run, we measured the time to compute one GMRES iteration in the pressure solver, averaged the results over 200 timesteps after the initial transient, and computed the throughput as GDoF/s per GMRES iteration. In the reported results, *ntX* refers to the preconditioner formulation with *nt1* being the none-overlapped and *nt2* the new overlapped formulation, and *gsX* indicates the four different gather-scatter strategies discussed above, with *gs1* being single stream pack and unpack, *gs2* multi-stream, concurrent pack operations, *gs3* multi-stream, concurrent unpack operations and *gs4* multi-stream, concurrent pack and unpack operations.

The result for the smaller case, Figure 4, shows that the highest throughput is achieved using the new overlapped formulation and single stream gather-scatter strategy (*nt2gs1*). Furthermore, there was no apparent difference (in throughput) between the standard CG formulation (see Figure 4(a)) and the fused kernel version (see Figure 4(b)). In contrast, the pipelined CG's overlapped communication

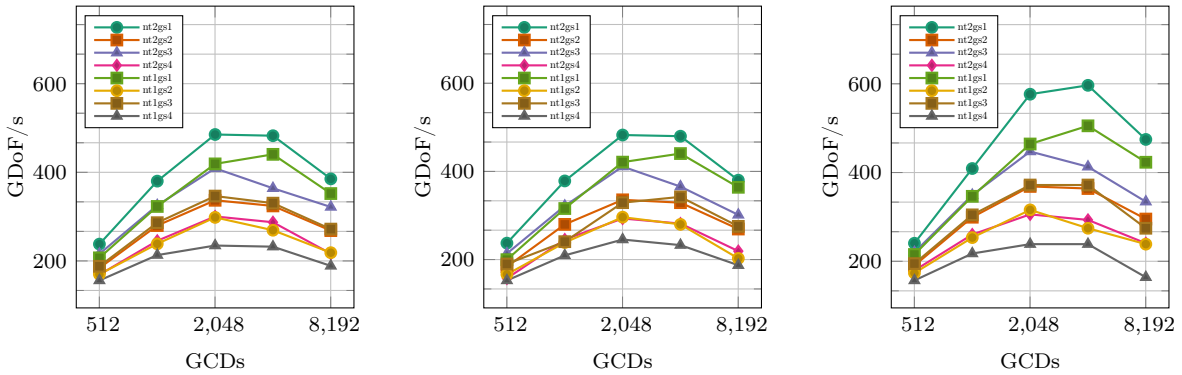
and computation show a clear benefit in throughput compared to the other two conjugate gradient formulations (see Figure 4(c))). Furthermore, for all tested CG methods (Figures 4(a) - 4(c)), scalability is lost around 7000 elements, which agrees with the observations made for the large RBC case in [9].



(a) Standard PCG. **(b) Fused kernels PCG.** **(c) Pipelined PCG.**

Figure 4: Strong scaling of Neko on LUMI for the TGV case with a 128^3 element mesh, with performance measured in throughput (GDoF/s) of one GMRES iteration when solving the pressure equation. For each experiments, number of threads is indicated by *ntX*, and the gather-scatter strategy by *gsY*.

The results for the larger mesh, shown in Figure 5, show a similar trend as observed for the smaller problem, with the new overlapped formulation using a single stream for gather-scatter as the fastest (highest throughput) version for across all number of nodes and CG formulations. Standard and fused CG also showed similar performance (Figures 5(a) - 5(b)), while the pipelined formulation achieves significantly higher throughput (Figure 5(c)). Neither of the tested CG methods scale past 7,000 elements per GCD.

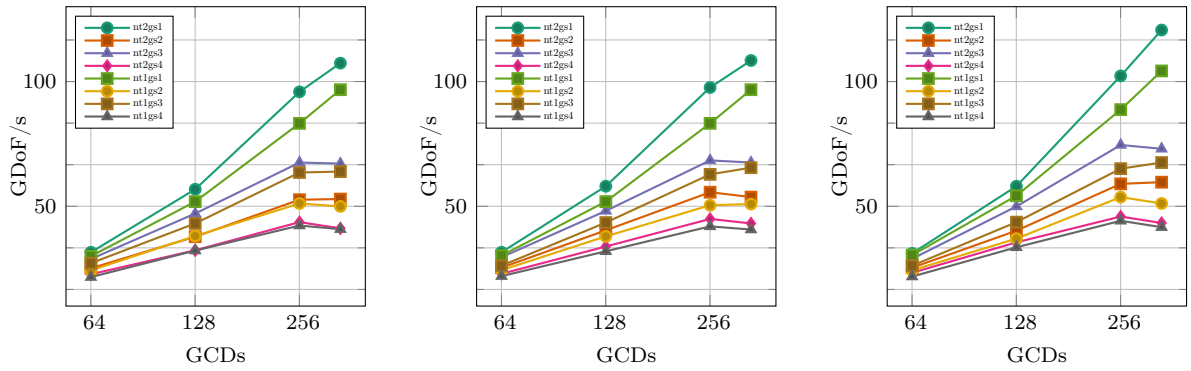


(a) Standard PCG. **(b) Fused kernels PCG.** **(c) Pipelined PCG.**

Figure 5: Strong scaling of Neko on LUMI for the TGV case with a 256^3 element mesh, with performance measured in throughput (GDoF/s) of one GMRES iteration when solving the pressure equation. For each experiments, number of threads is indicated by *ntX*, and the gather-scatter strategy by *gsY*.

For both tested problem sizes, it was unexpected that multi-stream concurrent gather-scatter would not improve performance, particularly for the unpacking part (*gs3*), which, in theory, should allow mitigating some of the overheads caused by the late arrival of MPI messages which would otherwise delay the entire unpack operation. This could be due to costly API / event synchronization calls. Furthermore, since early results on NVIDIA GPU platforms indicate that multi-stream/parallel unpack is beneficial, detailed analysis is necessary to better understand these results, requiring better profiling tools than were available (and supported) on LUMI. However, since LUMI's ROCm version 5.2.3 is rather old, released in August 2022, and the scheduling of work in the GPU stream could have been improved in later releases, we decided to rerun the experiments on another HPE Cray EX. Dardel is a smaller, 62 node HPE Cray EX hosted by PDC at KTH in Stockholm. Besides its smaller size, it has the same specifications (hardware, compilers etc) as LUMI (see Table 1) but uses ROCm version 5.7.0 from

September 2023. Figure 6 shows the results from running the smaller test case on Dardel using eight up to 42 nodes. Compared to LUMI (Figure 4), we see similar results for the different gather-scatter strategies on Dardel, but the achieved throughput is up to 20% higher than LUMI. Furthermore, in contrast to the LUMI results, all three CG formulations perform more similarly on Dardel (Figure 6(a) - 6(c)). It should be noted that some of the performance improvement is due to the smaller size of Dardel; nevertheless, the results indicate how important the runtime/driver is for performance on accelerated systems. Also, the AMD software stack is still maturing, which brings new benefits but also poses challenges as it can change performance/scaling behavior, e.g., with the reduction of API overheads.



(a) Standard PCG. **(b) Fused kernels PCG.** **(c) Pipelined PCG.**

Figure 6: Strong scaling of Neko on Dardel for the TGV case with a 128^3 element mesh, with performance measured in throughput (GDoF/s) of one GMRES iteration when solving the pressure equation. For each experiment, the number of threads is indicated by ntX, and the gather-scatter strategy by gsY.

5 Conclusion and Future Work

Our work on the high-fidelity spectral element code Neko has resulted in the formulation of a new, efficient, and scalable preconditioner for the pressure-Poisson equation. This task-decomposed overlapped preconditioner has not only reduced the execution time of the preconditioner by almost 20% but has also shown improved strong scaling compared to the old formulation for all tested configurations. This improved scalability is a significant advancement and allowed Neko to sustain near-perfect parallel efficiency up to 80% of the pre-exascale supercomputer LUMI.

However, the results show that some algorithmic developments, e.g., concurrent gather-scatter operations, gave negligible or even worse performance. A more detailed analysis is necessary to better understand these results. A more detailed analysis could also explain the improved performance of changing to a newer ROCm version.

Furthermore, since AMD and NVIDIA GPUs use different scheduling strategies for concurrent kernels, we are in the process of extending the performance evaluation to NVIDIA GPUs and performing an in-depth performance analysis, comparing performance on both GPU architectures.

Acknowledgments

This work was supported by the Swedish Research Council under grant reference 2019-04723 “Efficient Algorithms for Exascale Computational Fluid Dynamics”. Financial support was also provided by the Swedish e-Science Research Centre Exascale Simulation Software Initiative (SESSI) and the European High Performance Computing Joint Undertaking (EuroHPC-JU) and Sweden, Denmark, Greece, Germany, Spain under grant agreement no. 101093393. We also acknowledge the EuroHPC Joint Undertaking for awarding this project access to the EuroHPC supercomputer LUMI, hosted by CSC (Finland) and the LUMI consortium through a EuroHPC Development Access call. Some of the computations were enabled by resources provided by the National Academic Infrastructure for Supercomputing in Sweden (NAISS), partially funded by the Swedish Research Council through grant agreement no. 2022-06725.

References

[1] Niclas Jansson, Martin Karp, Artur Podobas, Stefano Markidis, and Philipp Schlatter. Neko: A modern, portable, and scalable framework for high-fidelity computational fluid dynamics. *Computers & Fluids*, 275:106243, 2024.

[2] Martin Karp, Daniele Massaro, Niclas Jansson, Alistair Hart, Jacob Wahlgren, Philipp Schlatter, and Stefano Markidis. Large-scale direct numerical simulations of turbulence using GPUs and modern Fortran. *The International Journal of High Performance Computing Applications*, 37(5):487–502, 2023.

[3] Niclas Jansson. Spectral element simulations on the NEC SX-Aurora TSUBASA. In *The International Conference on High Performance Computing in Asia-Pacific Region*, HPC Asia 2021, pages 32–39, Virtual Event, Republic of Korea, 2021. ACM.

[4] George Em Karniadakis, Moshe Israeli, and Steven A Orszag. High-order splitting methods for the incompressible navier-stokes equations. *Journal of computational physics*, 97(2):414–443, 1991.

[5] Steven A Orszag, Moshe Israeli, and Michel O Deville. Boundary conditions for incompressible flows. *Journal of Scientific Computing*, 1(1):75–111, 1986.

[6] AG Tomboulides, JCY Lee, and SA Orszag. Numerical simulation of low mach number reactive flows. *Journal of Scientific Computing*, 12(2):139–167, 1997.

[7] Paul F Fischer. An overlapping Schwarz method for spectral element solution of the incompressible Navier–Stokes equations. *Journal of Computational Physics*, 133(1):84–101, 1997.

[8] Paul F. Fischer and James W. Lottes. Hybrid Schwarz-multigrid methods for the spectral element method: Extensions to Navier-Stokes. In *Domain Decomposition Methods in Science and Engineering*, pages 35–49. Springer Berlin Heidelberg, 2005.

[9] Niclas Jansson, Martin Karp, Adalberto Perez, Timofey Mukha, Yi Ju, Jiahui Liu, Szilárd Páll, Erwin Laure, Tino Weinkauf, Jörg Schumacher, Philipp Schlatter, and Stefano Markidis. Exploring the ultimate regime of turbulent rayleigh–bénard convection through unprecedented spectral-element simulations. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '23, New York, NY, USA, 2023. Association for Computing Machinery.

[10] Niclas Jansson, Martin Karp, Jacob Wahlgren, Stefano Markidis, and Philipp Schlatter. Scalable High-Fidelity Simulation of Turbulence With Neko Using Accelerators. In *Proceedings of the Cray User Group (CUG)*, Helsinki, Finland, 2023.