The 9th European Congress on Computational Methods in Applied Sciences and Engineering
ECCOMAS Congress 2024
3–7 June 2024, Lisboa, Portugal

PERFORMANCE ANALYSIS OF PARALLEL-IN-TIME TECHNIQUES IN MODERN SUPERCOMPUTERS

JOSEP PLANA-RIU[1], F.XAVIER TRIAS[1], ÀDEL ALSALTI-BALDELLOU[2], GUILLEM COLOMER[1] AND ASSENSI OLIVA[1]

[1] Heat and Mass Transfer Technological Center
Technical University of Catalonia, ESEIAAT
Carrer de Colom 11, 08222 Terrassa (Barcelona), Spain
email: josep.plana.riu@upc.edu

[2] Department of Civil, Environmental and Architectural Engineering (ICEA)
University of Padova
Via Francesco Marzolo 9, 35131 Padova, Italy

Key words: Parallel-in-time, Sparse matrix-vector product, Sparse matrix-matrix product, Arithmetic intensity, High-performance computing, Computational Fluid Dynamics

Summary. This paper aims to improve the efficiency of large-scale turbulent simulations by improving the arithmetic intensity of the operations. This is done by applying a parallel-in-time ensemble averaging technique so that multiple flow states are run simultaneouslly in the same device. This transforms sparse matrix-vector products into sparse matrix-matrix products, improving the arithmetic intensity.

The performance of these operations as well as the speed-ups generated in the operation itself, in the whole iteration and an estimation in the whole simulation is presented, so that for cases in which the averaging interval is significantlly longer than the transition interval, remarkable speed-ups in the whole iteration are obtained.

1 INTRODUCTION

Finite-volume solutions of the Navier-Stokes equations are usually characterized by a few algebraic (or stencil-based equivalent) operations, i.e. the dot product (`dot`), the linear combination of vectors (`axty`), the elementwise product of vectors (`axty`), and the sparse matrix-vector product (`SpMV`). Among all these operations, for typical applications of the simulation of the Navier-Stokes equations, the `SpMV` is a bottleneck: even though the available computational power to do the operations is big enough, data cannot be provided fast enough to use all the resources. This is defined as a memory-bound code, as Williams et al. [1] defined in their roofline model: performance is bounded by data rate instead of flops. Therefore, addressing the memory-boundness of CFD codes is of big importance to reach the full potential of current high-performance computing (HPC) systems.

An increase of the arithmetic intensity (AI) will move the status of the computation closer to the compute-bound: the calculations will be less dependant on how much data can be transferred. In order to deal with this, previous studies play with the composition of the `SpMV` operation so that the amount of data to transfer is reduced.

In his study, Krasnopolsky [2] suggests that by running a set of m flow states of the same simulation, simultaneously, to compress all the `SpMV` operations into the product of a sparse and a dense matrix consisting on all the vectors of the different flow states (Generalized sparse matrix-vector product `GSpMV`) within the solution of the Poisson equation. According to Makarashvili et al. [3], if these flow states are statistically independent, then the ensemble average of these flow states will lead to the same flow statistics that would be obtained from a long averaging simulation. By doing so, the total simulated time (averaged) could be reduced by a factor of m which given the proper flow conditions might lead to an eventual speed-up of the whole simulation.

In this case, for a turbulent channel flow the study shows a maximum speed-up of 1.58 with a times ratio, i.e. the ratio between the averaging time and the transition time, of 20. For a more complex case such as a channel with wall mounted cubes, the maximum speed-up obtained for the same times ratio of 20 is 2.35, while preserving in all cases the computed turbulent statistics.

On the other hand, Alsalti-Baldellou et al. [4] uses repetitions in the geomtry, i.e. symmetries and repeated patterns, in order to exploit the block structure of the matrices in the domain and thus applying a sparse matrix-matrix product, `SpMM` in the totality of the `SpMV` operations of the simulation. This way, the benefits of the increased AI are not only applied in the solution of the Poisson equation but instead in the whole simulation. By exploiting this, the study showed speed-ups up to 5 with the product with the Laplacian operator in CPU cores, and up to 3.25 when performed in GPU nodes by exploiting 3 symmetries, i.e. 8 columns in the dense matrix. A similar behaviour was observed for the gradient and divergence operator as well. By exploiting this, a reduced memory footprint of the whole simulation was obtained as well as the improvements in performance.

Hence, in this work, the `SpMM` operations have been applied in the framework of a parallel-in-time simulation with ensemble averaging of the results in order to test and analyze the performance of this framework given the additional benefits of applying the improved arithmetic intensity operations to all sparse matrix-vector products instead of in just the solution of the Poisson equation.

2 PARALLEL-IN-TIME ENSEMBLE AVERAGING SIMULATIONS

As firstly introduced by Krasnopolsky[2], a parallel-in-time ensemble averaging simulation is based in running m simultaneous shorter simulations in the same device. Afterwards, according to Makarashvili et al. [3], if the averaging has started when all simulations are statistically independent, i.e. the correlation between each simultaneous flow state is 0, the ensemble averaged results will be the same as the ones obtained from a classical time averaging simulation.

In order to do so, given that the simulated time for the classical setup is defined as $T_T + T_A$, being T_T the destinated transition time, and T_A the averaging simulated time, for the studied framework, given m flow states are simulated simultaneouslly, the total simulated time will be

$$T(m) = T_T + \frac{T_A}{m}, \tag{1}$$

as depicted in Fig. 1.

Note that in this case the transition time, T_T should be such that it allows the different flow states to be statistically independent so that the results from Makarashvili et al. [3] can be

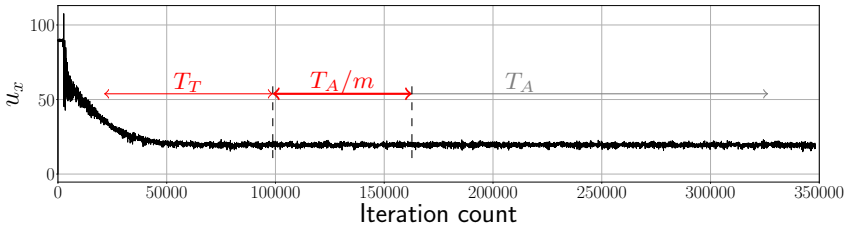

Figure 1: Transition time, T_T, and averaging time, T_A, of a numerical simulation with 1 rhs (gray) and m rhs (red).

applied. Note that the different flow states will be determined as statistically independent when the cross-correlation function, $\rho_{u_k u_l}$, is zero [2],

$$\rho_{u_k u_l} = \frac{cov(u_k, u_l)}{var(u_k) var(u_l)}, \tag{2}$$

where k, l are two indices indicating a different flow state. Note that the transition time is defined *a priori*, i.e. the user should know before approximately when the simulations will be statistically independent.

In order to further characterize the simulation, the ratio of the averaging time, T_A and the transition time, T_T, will be defined as the times ratio, β,

$$\beta = \frac{T_A}{T_T}, \tag{3}$$

so that the bigger the times ratio, the lesser the importance that the transition will time will have in the overall simulated time and thus the potential to exploit the benefits of the method will be increased.

In his study, Krasnopolsky [2] estimates the speed-up in the whole simulation to be characterized by the number of rhs, m; and the weight of the Poisson solution within the iteration, θ; and the times ratio, β, such that

$$P_m = \frac{1+\beta}{m+\beta} \frac{5m}{5m - 3\theta(m-1)}, \tag{4}$$

where the second term estimates the speed-up in the iteration, $P_{m,ite}$ and the first term extends the contribution of the iteration speed-up to the whole simulation. By doing so, by computing the speed-up in a single iteration the speed-up of the whole simulation given a times ratio can be estimated, being this the strategy followed by [2]. Moreover, given Eq. (4), an optimal number of rhs, m^* is estimated by the study, which would maximize the overall simulation speed-up,

$$m^* = \sqrt{\frac{3\beta\theta}{5 - 3\theta}}. \tag{5}$$

3 IMPROVEMENTS IN ARITHMETIC INTENSITY WITH `SpMM`

Given the definition of AI as the ratio between the number of required floating point operations and data to be transferred from memory, and given that the amount of operations to

perform cannot be modified much, it looks straightforward that the improvmements on AI in the case of any kernel operation could be due to reductions in the amount of data transferred. Thus, in the case of a sparse matrix-vector operation, if the sparse matrix A presents a repeated block m times, e.g. a $\log_2 m$ symmetries [4] or m flow states [2], these `SpMV` can be transformed into a sparse matrix-matrix operation in which the dense matrix has m columns.

By doing so, the amount of floating point operations to perform remains constant, $n \times nnz(A)$, where n is the number of matrix rows, and $nnz(A)$ is the number of non-zeros per row of the matrix A; whereas the amount of data to communicate is reduced, as the sparse matrix has to be transferred only once, instead of m times. By the definition of AI, this procedure will thus increase it, leading to speed-up in the process.

$$\begin{pmatrix} A & \\ & A \end{pmatrix} \begin{pmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \end{pmatrix} \qquad\qquad A(\mathbf{u}_1 \;\; \mathbf{u}_2)$$

Figure 2: Left: two `SpMV` are performed simultaneouslly as a bigger product with a block structured matrix. Right: structure in which the two products are performed with a single call of A.

The difference between the two cases is illustrated in Fig. 2, in which it is straigtforward to observe that in the left structure, the sparse matrix A is called twice, whereas for the right structure, there is only one call of A. Moreover, the calls to $\mathbf{u}_1, \mathbf{u}_2$ and retrieving the results of both products will imply a similar cost and memory communication time.

As previously stated, A is characterized by $nnz(A)$ and n, while $\mathbf{u}_i$ has as well n entries. Hence, the amount of data to communicate for a setup of m `SpMV` would be $m(2n + nnz(A)n)$, while for the `SpMM` with m rhs, the amount of data to communicate becomes $2mn + nnz(A)n$, with m only scaling the vectors but omitting the sparse matrix as it is only transferred once. By doing so, the arithmetic intensity of these operations can be estimated, assuming perfect temporal locality, as

$$AI_{\texttt{SpMM},m \text{ rhs}} = \frac{nnz(A)mn}{2mn + nnz(A)n}, \tag{6a}$$

$$AI_{m \texttt{ SpMV}} = \frac{nnz(A)mn}{m(2n + nnz(A)n)}, \tag{6b}$$

and thus the speed-up obtained with the inclusion of the `SpMM` kernel compared to using the classical `SpMV` is upper bounded by the ratio of the two arithmetic intensities,

$$P_{m,ub} = \frac{m(2n + nnz(A)n)}{2mn + nnz(A)n}, \tag{7}$$

which depends on the density of the matrix, the number of rhs and the size of the problem. The lower bound of the speed-up obtained can be computed assuming zero temporal locality that can be interpreted as substituting $nnz(A)$ by n, i.e. assuming the sparse matrix is actually dense.

4 TEST CASES

The method was tested in a differentially heated cavity filled with air at Ra=10^{10} using the in-house code TermoFluids Algebraic (TFA) in which the algebraic kernels are implemented within HPC^2 framework[6]. This flow will thus be governed by the non-dimensional semi-discrete incompressible Navier-Stokes equations together with the energy equation with the Boussinesq approximation, which following the notation of [5] reads

$$M\mathbf{u}_s = \mathbf{0}_c, \tag{8a}$$

$$\Omega\frac{d\mathbf{u}_c}{dt} + C(\mathbf{u}_s)\mathbf{u}_c - \frac{\mathrm{Pr}}{\mathrm{Ra}^{1/2}}D\mathbf{u}_c + \Omega G_c\mathbf{p}_c + \Omega\mathbf{f}_c = \mathbf{0}_c, \tag{8b}$$

$$\Omega\frac{d\boldsymbol{\theta}_c}{dt} + C(\mathbf{u}_s)\boldsymbol{\theta}_c - \frac{1}{\mathrm{Ra}^{1/2}}D\boldsymbol{\theta}_c = \mathbf{0}_c, \tag{8c}$$

in which $\mathbf{f}_c = (0, \mathrm{Pr}\boldsymbol{\theta}, 0)^T$ is the Boussinesq approximation term. These equations were advanced in time by use of a third-order Heun Runge-Kutta scheme implemented into the Navier-Stokes equations using the methodology developed by Sanderse and Koren [7]. The timestep was selected by use of a self-adaptive time-step in which the value was selected according to the position within the stability region of the eigenbounds of the convective and diffusive operator [8]. The Poisson equation is solved using a Jacobi preconditioned Conjugate Gradient as a proof of concept, being solved for a fixed number of iterations in order to ensure that the obtained results just depend on the parameters tuned and not in the complexity of converging the Poisson equation.

As these test cases were run in a full high memory node of MareNostrum 5 GCC supercomputer, which consist of 2x Intel Xeon Platinum 8480+ 56C, 1024 GB of RAM memory; 2 MPI processes with 54 OpenMP threads were launched so that two remaining CPUs were left for threads.

To test how the parallel-in-time approach as well as the `SpMM` implementation would behave under different conditions, three different test cases were implemented in which the mesh used corresponded to a load of 400k cells/CPU (`400k`), 300k cells/CPU (`300k`), and 200k cells/CPU (`200k`),.

Table 1: Meshes used for the different test cases of the differentially heated cavity.

Test case	$\mathbf{N_x}$	$\mathbf{N_y}$	$\mathbf{N_z}$	Total number of cells (M)
`200k`	176	704	176	21.82
`300k`	201	804	201	32.40
`400k`	220	880	220	42.55

Moreover, in order to test the dependency of the density of the matrices, the Laplacian operator has been discretized in such a way that it provides 7 non-zeros per row (`7p`), 13 non-zeros per row (`13p`), and 27 non-zeros per row (`27p`) as depicted in a two-dimensional representation in Fig. 3.

Hence, the analysis can be split into two parts. First of all, the study of the influence of the number of non-zeros per row to the behaviour of the `SpMM` compared to the `SpMV`. On the other

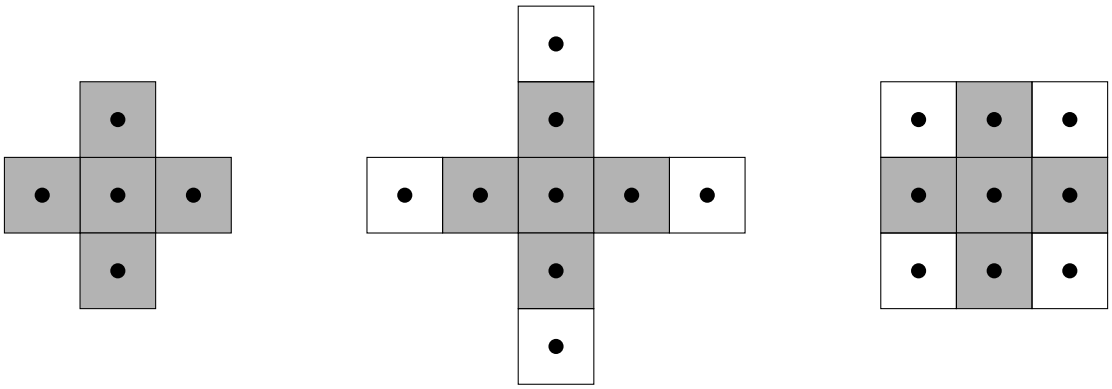

Figure 3: Two-dimensional representations of the stencil for `7p` (left), `13p` (center), `27p` (right). The shaded cells represent the first neighbors in all representations.

hand, the influence of the load of the node to this behaviour.

In order to study the influence of the discretization of the Laplacian operator, the three different CPU loads (`200k,300k,400k`) will be studied with the three different discretizations (`7p,13p,27p`), and the speed-up in the `SpMM`, iteration and the whole simulation for β values of 2, 20 and ∞ will be studied. In order to attain for different possibilities, the cases will be tested for 1, 2, 4, 8 and 16 rhs for `200k,300k`, and 1, 2, 4, and 8 rhs for `400k`, as the 16 rhs case did not fit in memory.

With regards to the speed-up of the performance of the implementation of `SpMM`, in Fig. 4 it can be observed that in all cases the speed-ups obtained are monotonically increasing with n. Moreover, a clear trend indicating that $P_{m,\texttt{SpMM}}$ grows with $nnz(A)$, as for all three cases studied this behaviour is observed.

Concerning the `200k`, the speed-up increases in a 5% for the `13p` and a 23% for the `27p`, compared to the original first neighbour case with 16 rhs. In addition, for the `300k` case with 16 rhs, these improvements are of 8% and 38%, respectively. In comparison, for 8 rhs, the `400k` case provides a 7% and 51% improvement, respectively. Note that observing Fig. 4 the trend shows that the differences between `7p,13p,27p` grow with m, so that the possible differences observed for 16 rhs in the more loaded case could even be bigger.

Moreover, it can be observed that the results are still not saturated for any of the cases, as $P_{m,\texttt{SpMM}}$ is still growing for all cases. This implies that with the current implementation the saturation point of the `SpMM` surpasses the memory of a single MareNostrum5 GPP high memory node.

Another important figure to discuss is the speed-up within the whole iteration, as it is the value that would replace the term $5m/(5m-3\theta(m-1)$ in Eq. (4); which is shown in Fig. 5. As a general trend, it can be observed that for all studied cases, the displayed values are smaller than the speed-ups for `SpMM`, as the operations are now diluded in a set of other operations (`axpy, axty, dot`) as well as lighter `SpMM` operations. Note that the speed-ups obtained now for `400k` with 8 rhs is similar to the speed-ups with 16 rhs obtained with the lower loads of the CPUs. Regarding the behaviour of the different Laplacian discretizations (`7p,13p,27p`), the improvements in all three cases for the biggest cases fitting in the memory of the node compared to `7p` are of approximately 17% for `13p` and 50% for `27p`. Nonetheless, as the differences grow with a bigger number of rhs, for `400k` it would be expected to obtain even better conditions in

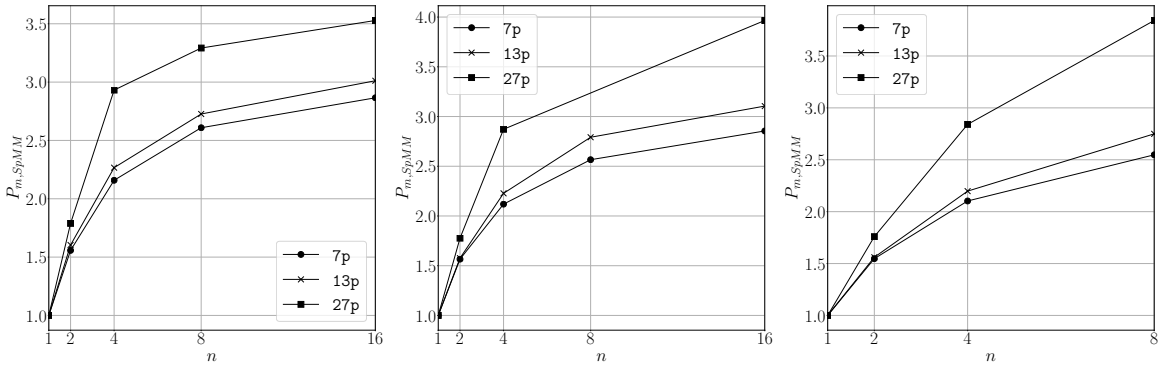

Figure 4: Speed-ups in `SpMM` for `200k` (left), `300k` (center), and `400k` (right) for the test case studied.

the iteration speed-up with 16 rhs.

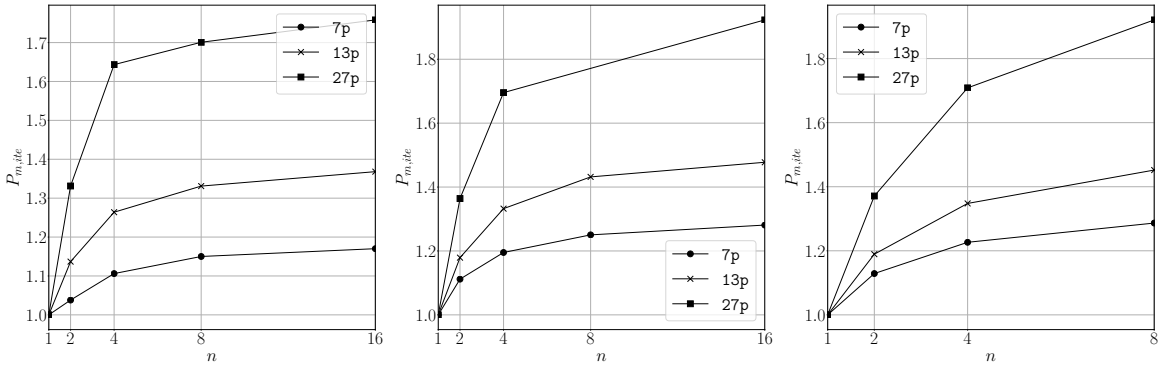

Figure 5: Speed-ups in the whole iteration for `200k` (left), `300k` (center), and `400k` (right) for the test case studied.

Hence, it can be clearly seen that the better performance of the method will be obtained with the `27p` discretization. With the iteration speed-ups calculated, the estimated simulation speed-up can be easily obtained for different β values so that the influence of the times ratio can also be studied, as it is one of the most relevant parameters in a parallel-in-time simulation. As in the previous results it has been obtained that the better discretization, for performance purposes, is `27p`, the presented results will just be with this case, analyzed for all `200k,300k` and `400k`.

As it can be seen in Fig. 6, for a bigger number of rhs the simulation speed-up decays for finite β values, as the $\beta \to \infty$ resembles the iteration speed-up figures, and it asymptotically tends to a maximum value but it does not decay for any number of rhs. Following the trends obtained for the `SpMM` speed-up (Fig. 4) and the iteration speed-up (Fig. 5), the simulation speed-up increases with the load of the CPU. As the number of Poisson solver iterations is quite high (550 iterations) given the problem size, the weight of the Poisson solver is expected to be quite high, in order to estimate the simulation speed-ups as well as the optimal number of rhs. For the `200k`, the 1 rhs-weight of the Poisson solver (1wgt) is of 0.9346, while the 1wgt for the `300k` and `400k` are of 0.9371 and 0.9346, respectively. This means that according to Krasnopolsky [2] the optimal number of rhs should be placed in a really similar spot. For the

sake of convenience, $\theta_{calc} = 0.935$ for cases, as the values differ in a 0.2% between them. By applying Eq. (5), the estimated speed-ups and optimal locations for all cases are displayed in Tab. 2.

Table 2: Estimated speed-ups and optimal number of rhs for the `27p` discretization ($\theta = 0.935$).

$\boldsymbol{\beta}$	$\boldsymbol{m^*}$	$m = 2$	$m = 4$	$m = 8$	$m = 16$
2	2	1.04	0.86	0.59	0.35
20	5	1.33	1.51	1.47	1.23
∞	∞	1.39	1.73	1.96	2.11

As it can be observed in Fig. 6 the speed-up values that most resemble the estimated optimal values are for `400k`, with negligible differences for the $\beta = 2$ and $\beta = 20$ cases and slight differences for $m = 8$ and $\beta \to \infty$. The estimated results generally overpredict the numerical results obtained for `200k` and `300k`, for greater values of m, yet for the smaller values of m the differences between estimated and calculated simulation speed-ups is also negligible. By doing so, the estimation from Krasnopolsky [2] provides the best results for large number of non-zeros per row, yet it can be considered as accurate for all sparse matrix densities as its difference is almost none.

With regards to the optimal value, it is difficult to determine in the tested cases which would be the optimal number of right hand sides to maximize the simulation speed-up. However, it is clear that in all cases the optimal for $\beta = 2$ lies in the interval $2m^* < 4$, while for $\beta = 20$, $4 \leq m^* 8$, in which the theoretical m^* can be found in the predicted intervals. For the $\beta \to \infty$ case, as the speed-up value is growing constantly, the value of $m^* \to \infty$ is recovered.

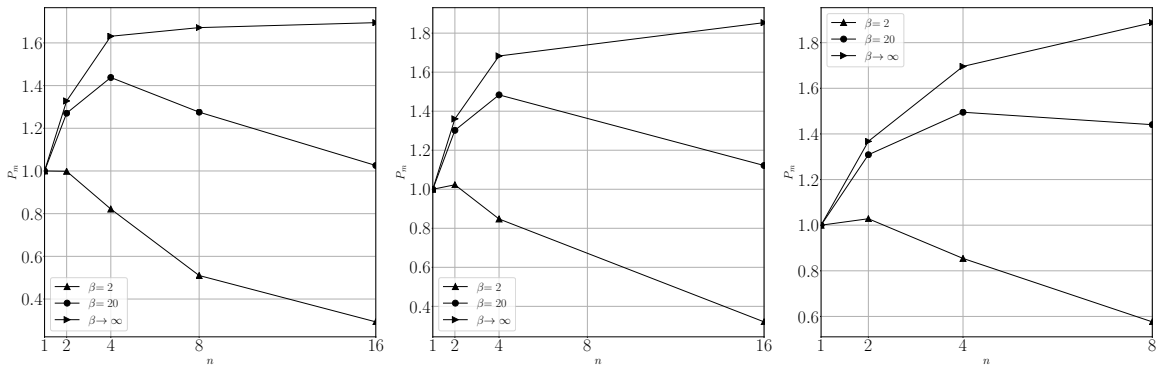

Figure 6: Estimated speed-ups in the whole simulation for `200k` (left), `300k` (center), and `400k` (right) for the test case studied with the `27p` discretization.

Furthermore, the results of the same discretization can be compared for different loads of the node as in this way the position of the `SpMM` speed-up compared to the theoretical upper and lower bounds can be analyzed. As it can be observed in Fig. 7, the dependency of the operation speed-up compared to the CPU load is almost non-existent for `7p` and `13p`. On the other hand, it is noticeable that for `27p`, there is an improvement of the speed-up of the `400k` compared to the other two loads. Moreover, it is noticeable that for `7p, 27p` the results are much closer to

the upper-bound compared to the `13p` discretization.

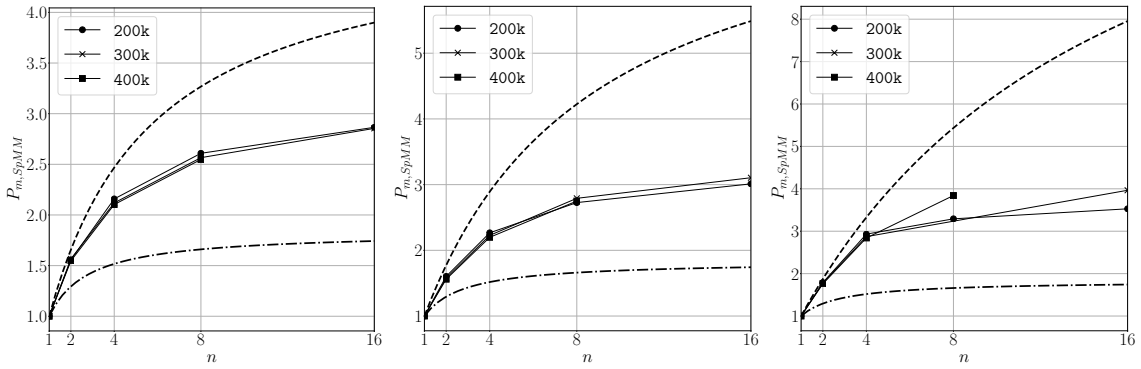

Figure 7: Speed-ups in `SpMM` for `7p` (left), `13p` (center), and `27p` (right) for the test case studied. The dashed line indicates the theoretical upper-bound, while the dot-dash line indicates the lower bound.

5 CONCLUSIONS

In this study, a generalization of the parallel-in-time approach of Krasnopolsky [2] in the totality of the `SpMV` operations in the algorithm is presented. By changing all `SpMV` for `SpMM` operations, as done by Alsalti-Baldellou et al. [4], the outcome of the method should be improved. Hence, in this approach, multiple flow states are run simultaneouslly in the same device for shorter simulated times so that the transition time is run until the different flow states are statistically independent, and then the averaging time is divided by a factor m, being m the number of flow states. By doing this, the arithmetic intensity of the operations will increase as the sparse matrix is only read once instead of m times, which leads to speed-ups [2, 4].

In order to test the new approach of the method, a differentially heated cavity of aspect ratio 4 filled with air, at Ra=10^{10} has been simulated with three different meshes (`200k,300k,400k`, Tab. 1), in a single MareNostrum 5 CPP HighMem node so that different CPU loads have been tested. Moreover, as the theory indicates that denser matrices will lead to bigger improvements in the arithmetic intensity when applying the method, three different Laplacian operator discretizations have been tested (Fig. 3), in which the number of non-zeros per row was of 7, 13, and 27. These runs have been performed for 1, 2, 4, 8 and 16 rhs, if fitting in memory. Thus, for the finer mesh (`400k`) the simulation was run up to 8 rhs.

It has been observed that in all three discretizations and meshes, there is speed-up in both `SpMM` and iterations (Figs. 4,5), with the `SpMM` values, and according to Fig. 7, the results are in between the lower and upper bounds and do not strongly depend on the CPU load. Moreover, after extending these results to the whole simulation (Fig. 6) for different times ratios ($\beta = \{2, 20, \infty\}$), the theoretical speed-ups and optimal number of rhs from Eqs. (4),(5), respectively, are recovered with a tolerable margin of error for `400k,27p`. Hence, the obtained results with the in-house code TermoFluids Algebraic match the results from Krasnopolsky [2].

In this line, future research lines include the application of this method to modern accelerators with GPU's which given their increased peak performance and memory bandwidths, which makes them more suitable for sparse algebra and, eventually, for CFD. Moreover, as the biggest downside of the method is the generation of the independent seeds. In order to treat this,

different methods will be studied to improve the generation of uncorrelated initial conditions.

ACKNOWLEDGEMENTS

The work presented in this abstract is supported by the *Ministerio de Economía y Competitividad*, Spain, SIMEX project (PID2022-142174OB-I00). J.P-R. is also supported by the Catalan Agency for Management of University and Research Grants (AGAUR), 2022 FI_B 00810. Calculations were performed on the MareNostrum 5 GCC supercomputer at the BSC. The authors thankfully acknowledge these institutions.

REFERENCES

[1] Williams, S., Waterman, A., and Patterson, D. 2009. "Roofline: an insightful visual performance for multicore architectures." Communication ACM 52.

[2] Krasnopolsky, B.I. 2018. "An approach for accelerating incompressible turbulent flow simulations based on simultaneous modelling of multiple ensembles." Computer Physics Communications 229

[3] Makarashvili, V., Merzari, E., Obabko, A., Siegel, A., and Fischer, P. 2017. "A performance analysis of ensemble averaging for high fidelity turbulence simulations at the strong scaling limit." Computer Physics Communications 219, 236-245

[4] Alsalti-Baldellou, A., Álvarez-Farré, X., Colomer, G., Gorobets, A. Pérez-Segarra, C.D., Oliva, A., Trias, F.X. 2024. "Lighter and faster simulations on domains with symmetries." Computers & Fluids 275

[5] Trias, F.X., Gorobets, A., Soria, M., Oliva, A. 2010. "Direct numerical simulation of a differentially heated cavity of aspect ratio 4 with Rayleigh numbers up to 10^{11} - Part I: Numerical methods and time-averaged flow." International Journal of Heat and Mass Transfer 53, 665-673

[6] Álvarez-Farré, X., Gorobets, A., Trias, F.X., Borrell, R., Oyarzun, G. 2018. "HPC^2-A fully-portable, algebra-based framework for heterogeneous computing. Application to CFD." Computers & Fluids 173, 285-292

[7] Sanderse, B., Koren, B. 2012. "Accuracy analysis of explicit Runge-Kutta methods applied to the incompressible Navier-Stokes equations." Journal of Computational Physics 231

[8] Plana-Riu, J., Trias, F.X., Pérez-Segarra, C.D., Oliva, A. 2023. "Cost-vs-accuracy analysis of self-adaptive time-integration methods." Proceedings of the 10th International Symposium on Turbulence, Heat and Mass Transfer