ParCFD2022
33^{rd} International Conference on Parallel Computational Fluid Dynamics
May 25-27 2022, Alba Italy

STRATEGIES TO INCREASE THE ARITHMETIC INTENSITY OF THE LINEAR SOLVERS

Àdel Alsalti-Baldellou[†‡], Xavier Álvarez-Farré[†], Andrey Gorobets[§], Assensi Oliva[†] and F. Xavier Trias[†]

[†]Heat and Mass Transfer Technological Center, Technical University of Catalonia
Carrer de Colom 11, 08222 Terrassa (Barcelona), Spain – https://www.cttc.upc.edu
{adel.alsalti, xavier.alvarez.farre, francesc.xavier.trias}@upc.edu

[‡]Termo Fluids S.L.
Carrer de Magí Colet 8, 08204 Sabadell (Barcelona), Spain – https://www.termofluids.com
adel@termofluids.com

[§]Keldysh Institute of Applied Mathematics, Russian Academy of Sciences
Miusskaya Sq. 4A, 125047 Moscow, Russia – https://www.keldysh.ru
andrey.gorobets@gmail.com

Key words: Poisson equation, Mesh symmetries, SpMM, Arithmetic intensity, Memory footprint, GPU

Abstract. The present work presents a strategy to increase the arithmetic intensity of the solvers. Namely, we profit spatial reflection symmetries to block diagonalise Poisson's equation and split the resulting blocks into a common part plus another subsystem-dependent whose size is, in practice, negligible. By doing so, we reduce the solvers' memory footprint and replace the standard sparse matrix-vector products with the more compute-intensive sparse matrix-matrix product. Then, we present a study about the impact of applying such strategies on different architectures. More concretely, we study how the subsystems' smaller size may make CPUs benefit from larger speedups when solving the decoupled blocks sequentially (due to the resulting greater cache reuse). Conversely, GPUs' lack of cache memory makes them more suitable for implementations based on sparse matrix-matrix products.

1 INTRODUCTION

Divergence constraints are present in many disciplines and usually lead to a Poisson equation whose solution is one of the most computationally intensive parts of scientific simulation codes. Despite the relevance of theoretical aspects like the solvers' time complexity or stability, unrelated computational factors may make them impractical for large-scale simulations. For instance, their excessive memory requirements and intrinsic lack of parallelism made traditional direct solvers not applicable to large 3D cases.

On top of that, the fact that most algorithms have a low arithmetic intensity makes them enjoy a small fraction of the systems' peak performances, which becomes even more dramatic given the growing unbalance between modern hardware's bandwidth and peak

performance. Hence, in the race towards exascale, reversing this trend and developing more efficient codes is crucial. Approaches trying to remedy this are generally based on reducing memory traffic, solving multiple right-hand sides (RHSs), using mixed-precision algorithms or adapting more compute-intensive methods.

This work focuses on a strategy profiting spatial reflection symmetries to block diagonalise Poisson's equation [1, 2]. Then, by recalling the regular structure of the resulting subsystems, the solvers' arithmetic intensity can be considerably increased thanks to replacing the standard sparse matrix-vector product (`SpMV`) with the more compute-intensive sparse matrix-matrix product (`SpMM`) [3].

Without losing generality, we will focus our numerical experiments on incompressible CFD simulations, thus arising our discrete Poisson equation from a direct application of the Fractional Step Method and equalling:

$$\mathsf{L}\tilde{\mathbf{p}}^{n+1} = \mathsf{M}\mathbf{v}^{p}, \tag{1}$$

where $\tilde{\mathbf{p}}^{n+1} = \mathbf{p}^{n+1}\Delta t$ stands for the discrete pseudo-pressure, and M and $\mathbf{v}^{p}$ for the discrete divergence and predictor velocity, respectively. Further details about the discretisation, the algorithm and its implementation can be found in [4, 5].

The rest of the work is organised as follows. Section 2 presents a mathematical description of L's block diagonalisation and gives the resulting algorithm. Conversely, section 3 discusses the main computational aspects involved and the behaviours expected both from CPU and GPU.

2 BLOCK DIAGONALISATION OF LAPLACE OPERATOR

For the sake of clarity, let us consider a mesh with a single reflection symmetry. Additionally, let us impose the same grid points' ordering on each of the sides [1]. As a result, the mesh is halved into two subdomains and all the scalar fields satisfy:

$$\mathbf{x} = \begin{pmatrix} \mathbf{x_1} \\ \mathbf{x_2} \end{pmatrix} \in \mathbb{R}^{N}, \tag{2}$$

where $\mathbf{x_1}, \mathbf{x_2} \in \mathbb{R}^{N/2}$ correspond to $\mathbf{x}$'s restriction to each of the subdomains. Then, spatially symmetric points are in the same position within the subvectors, and the discrete Laplace operator satisfies:

$$\mathsf{L} = \begin{pmatrix} \mathsf{L}_{\text{inn}} & \mathsf{L}_{\text{out}} \\ \mathsf{L}_{\text{out}} & \mathsf{L}_{\text{inn}} \end{pmatrix} \in \mathbb{R}^{N\times N}, \tag{3}$$

where N stands for the mesh size and $\mathsf{L}_{\text{inn}}, \mathsf{L}_{\text{out}} \in \mathbb{R}^{N/2\times N/2}$ for the inner- and outer-subdomain couplings, respectively.

Remarkably enough, another consequence of the grid points' ordering imposed is that practically all the operators satisfy the following structure:

$$\hat{\mathsf{A}} = \begin{pmatrix} \mathsf{A} & 0 \\ 0 & \mathsf{A} \end{pmatrix}, \tag{4}$$

where $\hat{\mathsf{A}} \in \mathbb{R}^{N\times N}$ stands for the operator itself and $\mathsf{A} \in \mathbb{R}^{N/2^p\times N/2^p}$ for its restriction to any of the halves.

In such a context, we can define the following change-of-basis matrix:

$$\mathsf{S} \coloneqq \frac{1}{\sqrt{2}} \begin{pmatrix} \mathbb{I}_{N/2} & \mathbb{I}_{N/2} \\ \mathbb{I}_{N/2} & -\mathbb{I}_{N/2} \end{pmatrix} \in \mathbb{R}^{N\times N}, \tag{5}$$

which satisfies $\mathsf{S}^{-1} = \mathsf{S}$. Then, changing the basis of L by means of S leads to:

$$\hat{\mathsf{L}} \coloneqq \mathsf{S}\mathsf{L}\mathsf{S}^{-1} = \begin{pmatrix} \mathsf{L}_{\text{inn}} + \mathsf{L}_{\text{out}} & 0 \\ 0 & \mathsf{L}_{\text{inn}} - \mathsf{L}_{\text{out}} \end{pmatrix}, \tag{6}$$

decomposing eq. (1) into two fully-decoupled and half-sized subsystems, $\hat{\mathsf{L}}_1 \coloneqq \mathsf{L}_{\text{inn}} + \mathsf{L}_{\text{out}}$ and $\hat{\mathsf{L}}_2 \coloneqq \mathsf{L}_{\text{inn}} - \mathsf{L}_{\text{out}}$. Algorithm 1 summarises the resulting algorithm.

Algorithm 1 Poisson solver profiting a reflection symmetry

Require: $\mathsf{L}_{\text{inn}}, \mathsf{L}_{\text{out}} \in \mathbb{R}^{N/2\times N/2}$ and $\mathbf{b} \in \text{im}(\mathsf{L}) \subseteq \mathbb{R}^N$

1: **procedure** SOLVE($\mathbf{b}, \hat{\mathsf{L}}$)
2: Transform $\hat{\mathbf{b}} = \mathsf{S}\mathbf{b}$
3: Decoupled solution of $\hat{\mathsf{L}}_1\hat{\mathbf{x}}_1 = \hat{\mathbf{b}}_1$ and $\hat{\mathsf{L}}_2\hat{\mathbf{x}}_2 = \hat{\mathbf{b}}_2$
4: Inverse transform $\mathbf{x} = \mathsf{S}^{-1}\hat{\mathbf{x}}$
5: **end procedure**

3 COMPUTATIONAL CHALLENGES

Apart from the theoretical benefits of solving eq. (1) through algorithm 1, $\hat{\mathsf{L}}$'s regular structure leaves place for computational improvements. In the first place, it reduces the memory footprint of the solvers substantially. Indeed, recalling eqs. (3) and (4), to hold the operators $\hat{\mathsf{L}}$ and $\hat{\mathsf{A}}$ we only require L_{inn}, L_{out} and A, and, remarkably enough, L_{out}'s footprint is negligible with respect to that of L_{inn}.

On the other hand, algorithm 1 allows replacing the standard `SpMV` with the more compute-intensive `SpMM`. Indeed, given a scalar field $\mathbf{x}$, let us define $\bar{\mathbf{x}} \coloneqq (\mathbf{x_1}|\mathbf{x_2}) \in \mathbb{R}^{N/2\times 2}$. Then, $\hat{\mathsf{L}}$'s matrix multiplications can be computed as:

$$\bar{\mathbf{y}} = \mathsf{L}_{\text{inn}}\bar{\mathbf{x}} + \mathsf{L}_{\text{out}}\bar{\mathbf{x}}, \tag{7}$$

where $\mathbf{y} \coloneqq \hat{\mathsf{L}}\mathbf{x}$. Similarly, defining $\mathbf{w} \coloneqq \hat{\mathsf{A}}\mathbf{x}$, we have: $\bar{\mathbf{w}} = \mathsf{A}\bar{\mathbf{x}}$. Hence, in both cases computing matrix multiplications through `SpMM` allows reducing the amount of times the matrices are read.

Recalling that `SpMV` and `SpMM` are both memory-bound, the maximum speedup achievable by replacing an `SpMV` with an `SpMM` equals the ratio $\text{AI}_{\texttt{SpMM}}/\text{AI}_{\texttt{SpMV}}$, which, considering double-precision floating-point operations and the standard CSR matrix format, reads:

$$\frac{\text{AI}_{\texttt{SpMM}}}{\text{AI}_{\texttt{SpMV}}} = \frac{[8\text{nnz}(A) + 4\text{nnz}(A) + 4(m+1) + 8m + 8n + 8]\cdot[(2\text{nnz}(A)+1)K]}{[2\text{nnz}(A)+1]\cdot[8\text{nnz}(A) + 4\text{nnz}(A) + 4(m+1) + (8m+8n+8)K]}, \tag{8}$$

where nnz(A), m, n and K are the number of non-zeros, rows, columns and subvectors, respectively. It is clear, then, that the larger K is, the higher the speedup becomes and, on its limit for square matrices ($n = m$):

$$\lim_{K\to\infty} \frac{\mathrm{AI}_{\texttt{SpMM}}}{\mathrm{AI}_{\texttt{SpMV}}} \underset{m\gg 1}{\simeq} \frac{12\mathrm{nnz}(A)/m + 20}{16}.$$

In the conference, we will present the strategy with more detail and discuss the consequences of solving either sequentially or concurrently the subsystems of line 3, i.e., of computing the matrix multiplications either through two $N/2$-sized `SpMV`s or a single `SpMM`. Special attention will be paid to the greater CPU cache reuse offered by the `SpMV` approach in opposition to the `SpMM`'s suitability for GPUs.

ACKNOWLEDGEMENTS

Àdel Alsalti-Baldellou, Xavier Álvarez-Farré, Assensi Oliva and F. Xavier Trias have been financially supported by two competitive R+D projects: RETOtwin (PDC2021-120970-I00), given by MCIN/AEI/10.13039/501100011033 and European Union Next GenerationEU/PRTR, and FusionCAT (001-P-001722), given by *Generalitat de Catalunya* RIS3CAT-FEDER. Àdel Alsalti-Baldellou has also been supported by the predoctoral grants DIN2018-010061 and 2019-DI-90, given by MCIN/AEI/10.13039/501100011033 and the Catalan Agency for Management of University and Research Grants (AGAUR), respectively.

REFERENCES

[1] A. Gorobets, F. X. Trias, R. Borrell, O. Lehmkuhl, and A. Oliva, "Hybrid MPI+OpenMP parallelization of an FFT-based 3D Poisson solver with one periodic direction," *Comput. Fluids*, vol. 49, no. 1, pp. 101–109, 2011.

[2] O. Shishkina, A. Shishkin, and C. Wagner, "Simulation of turbulent thermal convection in complicated domains," *J. Comput. Appl. Math.*, vol. 226, pp. 336–344, apr 2009.

[3] H. Anzt, S. Tomov, and J. Dongarra, "On the performance and energy efficiency of sparse linear algebra on GPUs," *Int. J. High Perform. Comput. Appl.*, vol. 31, pp. 375–390, sep 2017.

[4] F. X. Trias, O. Lehmkuhl, A. Oliva, C.-D. Pérez-Segarra, and R. W. C. P. Verstappen, "Symmetry-preserving discretization of Navier-Stokes equations on collocated unstructured grids," *J. Comput. Phys.*, vol. 258, pp. 246–267, feb 2014.

[5] X. Álvarez-Farré, A. Gorobets, and F. X. Trias, "A hierarchical parallel implementation for heterogeneous computing. Application to algebra-based CFD simulations on hybrid supercomputers," *Comput. Fluids*, vol. 214, p. 104768, jan 2021.