Matrix-free kernel for the anisotropic Laplacian operator for x86 architectures

1 Motivation

The development of robust solvers for partial differential equations (PDEs) using high-order spectral/hp element methods has been an area gaining momentum among research communities over recent years [1, 2]. The spectral/hp element method utilises high-order polynomial expansions which leads to a number of advantages over low-order numerical methods, including a significantly lower level of numerical dispersion and dissipation. In addition, memory bandwidth, rather than clock speed, has become the bottleneck in achieving high throughput on modern processes in recent years. This favours high-order numerical methods which have higher arithmetic intensity, performing more floating-point operations for each byte transferred from memory.

To maximise this arithmetic intensity when using the high-order spectral/hp element methods to solve the anisotropic thermal conduction formulation under the Nektar++ framework [3], an efficient matrix-free operator together with the single-instruction multiple-data (SIMD) vectorization methodology is implemented. The matrix-free operator avoids the explicit construction of either large assembled global matrices or many dense local matrices. It utilizes the tensor product construction of the basis functions in the spectral/hp element to apply a sum-factorization technique [4].

2 Implementation in Nektar++

Nektar++ [3] is a cross-platform spectral/hp element framework for solving a range of scientific and engineering problems where high-fidelity solutions are required. To implement the matrix-free operator and SIMD technique for the anisotropic thermal conduction formulation in the Nektar++ framework, the following changes are made in its architecture:

- a matrix-free implementation of Helmholtz operator supporting anisotropic diffusion is created;
- single-instruction multiple-data (SIMD) vectorisation is implemented for this Helmholtz operator.

2.1 Helmholtz equation and its variational form

The anisotropic steady diffusion equation can be formulated as the inhomogeneous Helmholtz equation with the scalar constant $\lambda = 0$, given by

$$\nabla \cdot (\boldsymbol{\kappa}_c \cdot \nabla T) - \lambda T = Q \tag{1}$$

where $\boldsymbol{\kappa}_c$, λ and Q are the anisotropic diffusion tensor, reaction coefficient and a forcing term in the field, respectively. Applying the Galerkin method of weighted residuals to Equation (1) and the divergence theorem on the Laplacian operator, its variational form satisfies

$$\int_\Omega [\nabla v^h (\boldsymbol{\kappa}_c \cdot \nabla T^h)] d\Omega + \lambda \int_\Omega [v^h T^h] d\Omega = -\int_\Omega [v^h Q] d\Omega + \int_\Gamma [v^h (\boldsymbol{\kappa}_c \cdot \nabla T^h) \cdot \boldsymbol{n}] d\Gamma \quad (2)$$

for all test functions v^h. Now dropping the h superscript for clarify, we can expand T^h and v^h in terms of the basis functions on each element,

$$T^e = \sum_i \hat{T}^e \psi_i^e,$$

where the superscript ("e") refers to elemental quantities. Expressing as a matrix system, we arrive at

$$\begin{aligned} & \boldsymbol{L}^e \hat{\boldsymbol{T}}^e + \lambda \boldsymbol{M}^e \hat{\boldsymbol{T}}^e = -\boldsymbol{B}^e \boldsymbol{W}^e \boldsymbol{Q}^e + \boldsymbol{\Gamma} \\ \implies \quad & \boldsymbol{H}^e = \boldsymbol{L}^e + \lambda \boldsymbol{M}^e \end{aligned} \quad (3)$$

where the matrices $\boldsymbol{L}^e$, $\boldsymbol{M}^e$ and $\boldsymbol{B}^e$ respectively are the discrete representations of Laplacian, mass and basis evaluation operators. $\boldsymbol{W}^e$ is a diagonal matrix of quadrature weights and $\hat{\boldsymbol{T}}^e$ is the matrix of elemental basis functions ϕ_i evaluated at the quadrature points, ξ_j. For the non-zero surface flux, the vector $\boldsymbol{\Gamma}$ is introduced to represent the boundary integral term in equation (2). Therefore, the Helmholtz operator for each element can be defined as $\boldsymbol{H}^e$ in equation (3). The detailed formulation of $\boldsymbol{L}^e$ and $\boldsymbol{M}^e$ can be written as

$$\boldsymbol{L}^e = (\boldsymbol{D}^e \boldsymbol{B}^e) \boldsymbol{g}^e (\boldsymbol{D}^e \boldsymbol{B}^e)^T \boldsymbol{W}^e \quad (4a)$$

$$\boldsymbol{M}^e = \boldsymbol{B}^e \boldsymbol{W}^e (\boldsymbol{B}^e)^T \quad (4b)$$

where $\boldsymbol{D}$, $\boldsymbol{B}$ and $\boldsymbol{W}$ respectively are the one-dimensional derivative matrix, basis matrix and diagonal weight of numerical integration in the standard region element $\Xi \subset [-1, 1]^d$ (uniquely defined for each element type). The metric, $\boldsymbol{g}^e$, incorporates the diffusion tensor $\boldsymbol{\kappa}_c$ and the inverse of Jacobian matrix $\boldsymbol{J}$, as

$$\begin{aligned} & \boldsymbol{g}^e = (\boldsymbol{J}^e)^{-1} \boldsymbol{\kappa}_c (\boldsymbol{J}^e)^{-T} = \begin{bmatrix} g_{00} & g_{01} \\ g_{10} & g_{11} \end{bmatrix} \\ & \boldsymbol{\kappa}_c = \begin{bmatrix} \kappa_{00} & \kappa_{01} \\ \kappa_{10} & \kappa_{11} \end{bmatrix}; \quad (\boldsymbol{J}^e)^{-1} = \begin{bmatrix} j_{00} & j_{01} \\ j_{10} & j_{11} \end{bmatrix} \\ & g_{00} = \kappa_{00} j_{00}^2 + 2\kappa_{01} j_{00} j_{01} + \kappa_{11} j_{01}^2 \\ & g_{01} = g_{10} = \kappa_{00} j_{00} j_{10} + \kappa_{01} (j_{01} j_{10} + j_{00} j_{11}) + \kappa_{11} j_{01} j_{11} \\ & g_{11} = \kappa_{00} j_{10}^2 + 2\kappa_{01} j_{10} j_{11} + \kappa_{11} j_{11}^2 \end{aligned} \quad (5)$$

2.2 Matrix-free operator

By noting that the expansions are tensor products of one-dimensional functions, an efficient matrix-free operator can be constructed with the technique of sum-factorisation.

For instance, considering the backward transformation that maps spectral/hp element coefficients to their physical representation, which in two dimensions is expressed as

$$T^e(\xi_i, \xi_j) = \sum_{i,j}^{pq} \hat{T}^e_{ij} \psi^e_{ij}(\xi_i, \xi_j) = \sum_{i}^{p} \sum_{j}^{q} \hat{T}^e_{ij} \phi^e_i(\xi_i) \phi^e_j(\xi_j), \tag{6}$$

where $\hat{T}$ represents the coefficients of basis expansion. Therefore, the matrix-free operation can be performed by recasting Equation (6) as

$$T^e(\xi_i, \xi_j) = \sum_{i}^{p} \phi^e_i(\xi_i) \Big[\sum_{j}^{q} \hat{T}^e_{ij} \phi^e_j(\xi_j) \Big] \tag{7}$$

The summation term in the square brackets can be evaluated and stored first. Subsequently, the physical representation, $T^e(\xi_i, \xi_j)$, can be computed by iteratively summing the terms outside the square bracket. In such a way, the original $\mathcal{O}(P^4)$ operation has been replaced with two $\mathcal{O}(P^3)$ operations during backward transformation inside the matrix-free operator. The complete algorithm of matrix-free evaluation of Helmholtz operator is described in Section 2.4

2.3 SIMD Vectorisation

Traditional processor registers only have sufficient space to hold one value at a time. However, the modern CPU registers are capable of holding vectors of multiple values. With the use of instruction set extensions, it has become possible to run one operation on multiple pieces of data at the same time. These types of operations are known as single-instruction multiple-data (SIMD) operations and lead to a type of parallelism which is known as vectorisation.

2.4 Implementation

To leverage vectorisation in the matrix-free operator in the anisotropic thermal conduction formulation, the Helmholtz operator in Nektar++ is re-formulated as shown in Algorithm 1.

In Algorithm 1, *deformed* refers to a non-linear reference-to-world mapping, for which we require the calculation of the metric $\boldsymbol{g}$ for each quadrature point. *BwdTrans*, *PhysDeriv* and *IProduct* are the backward transformation, partial derivative of the physical solution and inner product operations, respectively. Therefore, the Helmholtz operator can be re-written as,

$$\begin{aligned} \boldsymbol{H}^e = \sum_{n=0}^{qp} [\nabla \psi^h(\eta^n) \;\; (\boldsymbol{J}^e(\eta^n))^{-1} [\boldsymbol{\kappa}_c(\eta^n) \cdot (\boldsymbol{J}^e(\eta^n))^{-T} \nabla \psi^h(\eta^n)] \\ + \lambda \psi^h(\eta^n) \psi^h(\eta^n)] |\boldsymbol{J}^e(\eta^n)| \omega^h(\eta^n) \end{aligned} \tag{8}$$

Algorithm 1 Overview of matrix-free evaluation of Helmholtz operator

1: Helmholtz($\hat{T}, \omega, \boldsymbol{B}, \nabla\boldsymbol{B}, \boldsymbol{J}, \boldsymbol{\kappa}_c$)
2: **for** each element group **do**
3: $T^h \leftarrow$ BwdTrans($\hat{T}, \boldsymbol{B}$)
4: $\boldsymbol{out} \leftarrow \lambda \cdot$ IProduct($\hat{T}, \boldsymbol{B}, \omega, |\boldsymbol{J}|$)
5: $\boldsymbol{D}T^h \leftarrow$ PhysDeriv($T^h, \boldsymbol{D}, (\boldsymbol{J})^{-1}$)
6: **if** element is deformed **then**
7: **for** each quadrature point η **do**
8: $\boldsymbol{g} \leftarrow (\boldsymbol{J}(\eta))^{-1}\boldsymbol{\kappa}_c(\eta)(\boldsymbol{J}(\eta))^{-T}$
9: $\boldsymbol{D}T^h(\eta) \leftarrow \boldsymbol{g}\boldsymbol{D}T^h(\eta)$
10: **end for**
11: **else**
12: $\boldsymbol{g} \leftarrow (\boldsymbol{J})^{-1}\boldsymbol{\kappa}_c(\boldsymbol{J})^{-T}$
13: **for** each quadrature point η **do**
14: $\boldsymbol{D}T^h(\eta) \leftarrow \boldsymbol{g}\boldsymbol{D}T^h(\eta)$
15: **end for**
16: **end if**
17: **end for**
18: **for** each dimension d **do**
19: $\boldsymbol{out} \leftarrow \boldsymbol{out}$ + IProduct($\boldsymbol{D}T^h, \partial_d\boldsymbol{B}, \omega, |\boldsymbol{J}|$)
20: **end for**
21: return ***out***

Here, the symbol η refers to the location of Gauss points in the quadrature rule and qp is the total number of Gauss quadrature points. Line 9 and 14 in Algorithm 1 refer to the sum-factorisation operation in the matrix-free operator, similar to the term in the square bracket in Equation (7). After it has been computed for all elements in one dimension, the matrix-free operator is constructed by iteratively summing the tensor products in all direction, as shown in line 19 in Algorithm 1. The resultant algorithm is an optimization matrix-free operator with SIMD vectorisation.

3 Problem setup

In Nektar++, simulations are configured using XML session files. In order to employ the matrix-free operator and SIMD vectorisation in simulations, the following elements should be included in the .xml files:

```
<COLLECTIONS DEFAULT="MatrixFree" />
```

and

```
<SOLVERINFO>
    <I PROPERTY="EQTYPE" VALUE="SteadyDiffusion"/>
    <I PROPERTY="Projection" VALUE="Continuous"/>
    <I PROPERTY="GlobalSysSoln" VALUE="IterativeFull"/>
```

```
    <I PROPERTY="Preconditioner" VALUE="Diagonal"/>
</SOLVERINFO>

<GLOBALSYSSOLNINFO>
    <V VAR="u">
        <I PROPERTY="GlobalSysSoln" VALUE="IterativeFull"/>
        <I PROPERTY="LinSysIterSolver" VALUE="ConjugateGradient"/>
        <I PROPERTY="Preconditioner" VALUE="Diagonal"/>
        <I PROPERTY="MaxIterations" VALUE="5000"/>
        <I PROPERTY="IterativeSolverTolerance" VALUE="1e-6"/>
    </V>
</GLOBALSYSSOLNINFO>
```

The <COLLECTIONS DEFAULT="MatrixFree"/> element should be added directly under the root element, <NEKTAR>, which forces the Nektar++ library to use the matrix-free operator implementation for the computations. In the .xml file, the <GLOBALSYSSOLNINFO> is added as a sub-element into the <CONDITIONS> element. It defines the parameters to control the convergence of the chosen iterative solver. The Conjugate Gradient (ConjugateGradient) or Generalized Minimal Residual method (GMRES) iterative solver can be used together with the matrix-free operators by specifying the `LinSysIterSovler` tag in .xml session file. For instance, the values of `IterativeSolverTolerance` and `MaxIterations` tags control the convergence tolerance, the maximum number of iteration in the employed iterative solver. The value of `GMRESMaxStorage` tag defines the dimension of the Krylov subspace of the GMRES solver, and the maximum restart of GMRES algorithm is controlled by the value of `IterativeSolverTolerance` divided by `GMRESMaxStorage`.

The performance of the optimized matrix-free operator with SIMD vectorisation is compared with the basic matrix-free operator, `IterativeFull`, and multilevel static condensation method in this study. In the basic matrix-free operator, each arithmetic operation is performed individually without the vectorisation or optimisation for multiple elements. To switch off the optimized matrix-free operator, the element <COLLECTIONS DEFAULT="MatrixFree"/>should be removed in the .xml session file. However for both cases of basic and optimized matrix-free operators, the element <I PROPERTY="GlobalSysSoln" VALUE="IterativeFull"/> must be present in the .xml session file. In addition to the aforementioned tags, the value of the `GlobalSysSoln` tag can be set as `IterativeMultiLevelStaticCond` or `XxtMultiLevelStaticCond` to activate the multilevel static condensation method. Furthermore, to investigate the unsteady cases, the value of `EQTYPE` should be changed to `UnsteadyDiffusion` and the .xml session file will incorporates the following parameters, `TimeStep`, `FinalTime` and `NumSteps`. For the detailed configurations of the cases of basic and optimized matrix-free operators and multilevel static condensation methods, please refer to the .xml session files in example folder in the repository.

4 Performance evaluation

In this section, the performance of the matrix-free with/without SIMD vectorisation is compared with the basic matrix-free operator (using GMRES iterative solver) and Multi-Level Static Condensation method in the steady and unsteady cases. Those testing cases can be found in the directory, `example`, in the repository. In all the testing cases, the CPU used is an Intel i7-8550U with a standard clock speed of 1.8Hz and support for AVX2 instructions. The numerical examples are run in parallel using 1-4 processes. The parameters of the full iterative solver are shown in the above <`GLOBALSYSSOLNINFO`> element. The size of the computational domain for the steady and unsteady cases is 5×5 and the employed mesh resolutions are 40×40 and 80×80 elements. The order of the expansion basis ranges from 2 to 8. The same iterative solver parameters are used for all numerical examples, e.g., the tolerance, maximum iteration number and so on. The obtained results are presented in the subsequent sections.

4.1 Steady cases

The execution speeds of the basic and optimized matrix-free operators are first compared in serial in Figure 1. The orange and blue curves show the speed-up of the optimized matrix-free operator (ratios of the execution time of the basic matrix-free operator to the optimized one). It can be seen that the optimized matrix-free operator is always more efficient than the basic one for all expansion orders in this study. Figure 1 shows that the optimised operator is significantly faster for low polynomial orders. The performance gain reduces as the expansion order increases. Furthermore, as the mesh resolution increases, e.g., the blue line in Figure 1 for 80×80, the performance difference increases further.

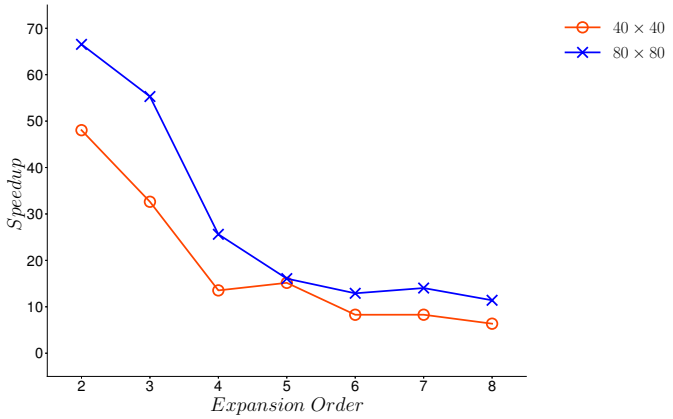
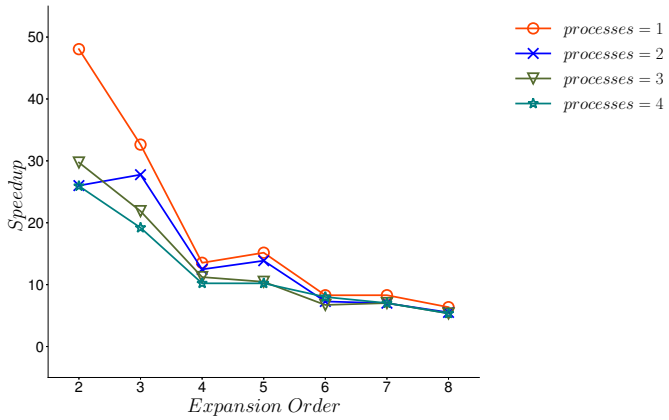
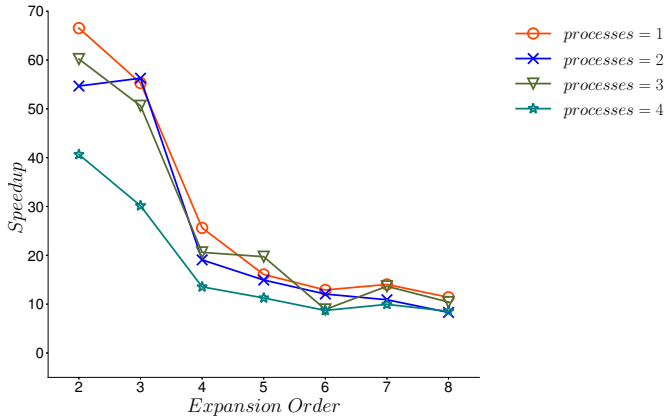


Fig. 1: Speedup (ratio of execution time of basic, relative to optimized matrix-free operators) with SIMD for steady cases in serial.

Similarly, the speedup is compared for the steady cases running in parallel on different numbers of processes. It is found that the highest ratio is observed for the cases using the low expansion orders and dense mesh resolution, as shown in Figure 2a and Figure 2b. Even for high expansion orders, the speedup of the optimized matrix-free operator is still approximately one order of magnitude higher than the basic matrix-free operator. Furthermore, except for low expansion orders, the speedup does not change significantly as the number of processes increases.



(a)



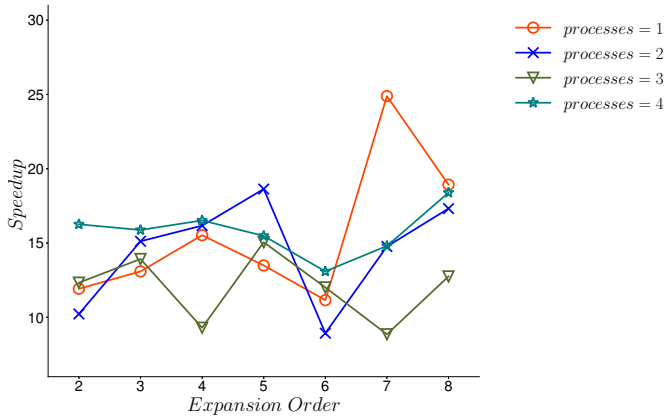
(b)

Fig. 2: Speedup (ratio of execution time of basic and optimized matrix-free operators) with SIMD for steady cases in parallel: (a) 40×40 and (b) 80×80 element meshes.

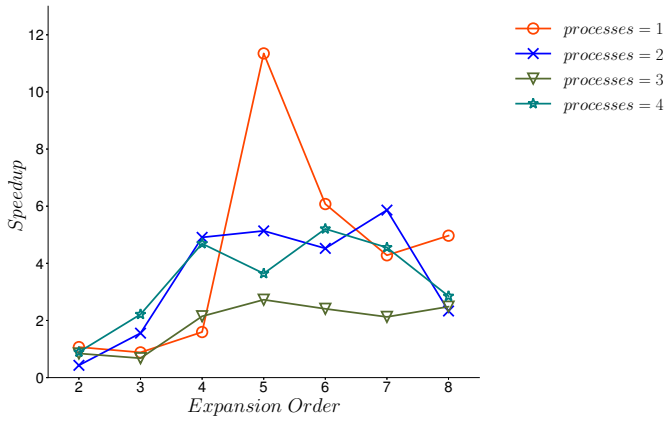
The superior performance of the optimized matrix-free operator manifests in the construction of operators and backward transformation operations. The detailed analyses of the execution time of the basic and optimized matrix-free operators on the aforementioned operations can be found in Figure 3. Figure 3a presents the ratio of execution time of the basic matrix operator to the optimized one to construct the system operators. This operation is accomplished by calling the *GeneralMatrixOp* function in Nektar++ library. It shows that the time taken by the basic matrix-free operator to assemble the matrices is at least an order of magnitude higher than the optimized matrix-free operator across all expansion orders. On the other hand, the performance gained by the optimized matrix-free backward-transform operator is less pronounced. Figure 3b shows that the speedup of the backward transformation increases significantly when the expansion order exceeds 4. In contrast, the time taken by the optimized matrix-free operator does not exceed the factor of 2 for all expansion orders in this study. Overall, the efficiency of the optimized matrix-free operator is approximately one order of magnitude higher than the basic one to construct the system operator and perform the backward transformation.

The SIMD vectorisation is another important feature to enhance the efficiency of parallelism, in which an arithmetic operation is actioned simultaneously on several data using one instruction in code. It can be seen in Figure 4 that the overall speedup of the code when using optimized matrix-free operators with SIMD vectorisation typically increases by 0.1 to 0.4 compared with the optimized matrix-free operator without SIMD, since the arithmetic intensity increases proportionally with the expansion order. This characteristic of SIMD vectorisation can be used to maximize the potential of high-order numerical methods for large-scale problems. If more processes and high expansion order are used to run the simulation in parallel, the ratio of execution time is stabilized around 1.3. Note that these timings are for the entire execution of the code and that the vectorised operators only account for a portion of the runtime.

The static condensation technique is frequently applied in high-order numerical methods, because it makes use of the natural boundary/interior decomposition of the spectral/hp element expansion to decouple the interior modes from the boundary degrees of freedom in order to efficiently compute the inversion of the global matrix. The multilevel static condensation methods [5] apply the static condensation technique recursively to further reduce the system size and improve overall performance. Although using fewer operations, this approach does lead to memory access incoherence and consequently an increase in cache misses. The performance of the optimized matrix-free operator with SIMD is compared with the multilevel static condensation method. In Figure 5, the curves speedup of the optimized matrix-free operator with SIMD relative to the multilevel static condensation method. For the steady case, it shows that the optimized matrix-free operator with SIMD is more efficient than the multilevel static condensation method in serial execution (process=1). It becomes more significant as the expansion order increases, i.e., the orange line in Figure 5. If multiple processes are used in parallel, the efficiency of the optimized matrix-free operator with



(a)



(b)

Fig. 3: Speedup of the optimized matrix-free operators with SIMD for constructing the system operator and performing the backward transformation: (a) construction of operator and (b) backward transformation

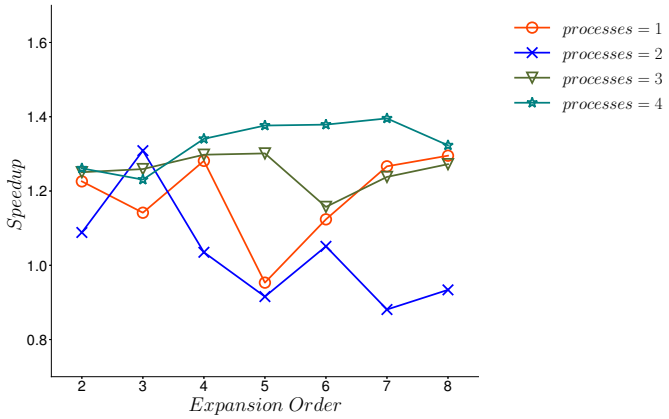


Fig. 4: Speedup of the optimized matrix-free operators with SIMD relative to the version without SIMD, using different numbers of processes (proc=1-4), for a mesh resolution of 80×80.

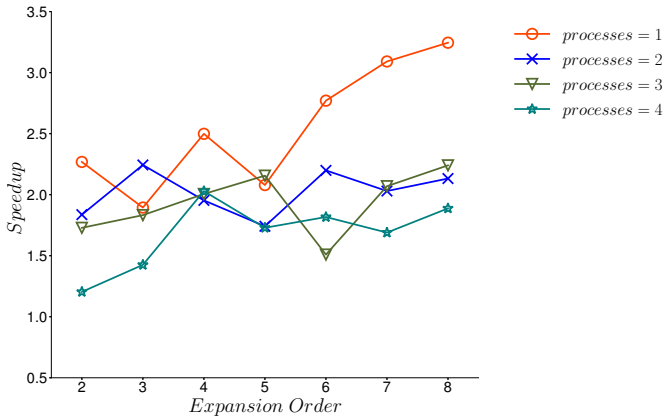


Fig. 5: Overall speedup of the optimized matrix-free operator with SIMD over the multilevel static condensation approach for steady cases (mesh resolution 80×80)

SIMD is still higher than the multilevel static condensation method by a factor of 1.5 to 2.0, e.g., the blue and green lines. However, it is noteworthy that the overall execution time taken by an optimized matrix-free operator and multilevel static condensation method is influenced by many factors, including the process of constructing the matrix operator, solving the global linear system and its implementation. In some cases, the speed of an optimized matrix-free operator can be slower than the multilevel static condensation method, for example, the transient cases discussed in Section 4.2. If the time taken to construct the local dense matrices is significant, a matrix-free operator approach can be advantageous, such as in the steady cases presented here.

4.2 Unsteady cases

In this section, the performance of the basic and optimized matrix-free operators with SIMD and Multilevel Static Condensation method is analysed for the unsteady cases of anisotropic thermal conduction. The following parameters are set for the unsteady case in the .xml file.

```
<PARAMETERS>
    <P> TimeStep = 0.001</P>
    <P> FinalTime = 0.005</P>
    <P> NumSteps = FinalTime/TimeStep</P>
    <P> IO_CheckSteps = 5</P>
    <P> IO_InfoSteps = 5</P>
</PARAMETERS>

<SOLVERINFO>
    <I PROPERTY="EQTYPE" VALUE="UnsteadyDiffusion"/>
    <I PROPERTY="Projection" VALUE="Continuous"/>
    <I PROPERTY="GlobalSysSoln" VALUE="IterativeFull"/>
    <I PROPERTY="TimeIntegrationMethod" VALUE="BDFImplicitOrder1"/>
</SOLVERINFO>

<GLOBALSYSSOLNINFO>
    <V VAR="u">
        <I PROPERTY="GlobalSysSoln" VALUE="IterativeFull"/>
        <I PROPERTY="LinSysIterSolver" VALUE="ConjugateGradient"/>
        <I PROPERTY="Preconditioner" VALUE="Diagonal"/>
        <I PROPERTY="MaxIterations" VALUE="5000"/>
        <I PROPERTY="IterativeSolverTolerance" VALUE="1e-6"/>
    </V>
</GLOBALSYSSOLNINFO>
```

Similar to the steady cases, the results in Figure 6 are the speedup of the optimized matrix-free operators with SIMD relative to the basic operators. It shows that the optimized matrix-free operator with SIMD is much more efficient

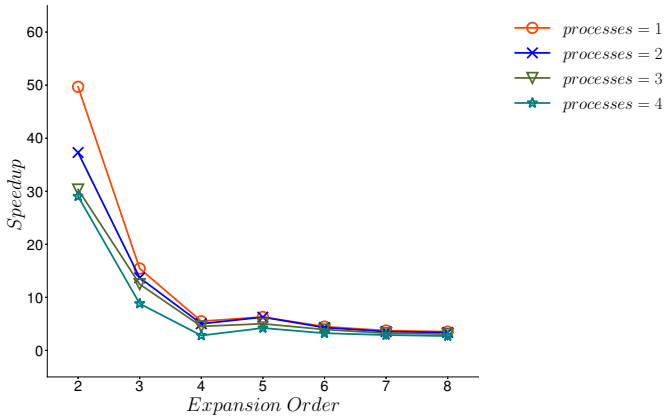


Fig. 6: Speedup of the optimized matrix-free operators with SIMD over the basic operator for unsteady cases (mesh resolution 40×40)

than the unoptimized version in unsteady cases. For the expansion order lower than 4 using multiple processes, the speed-up of the optimized matrix-free operator is beyond one order of magnitude of the basic matrix-free operator. As the expansion order increases further, the efficiency of the optimized matrix-free operator with SIMD is still about 5 times faster. Overall, for the unsteady cases, the ratio of performance between the basic and optimized matrix-free operators does not change significantly using the different number of processes in this study.

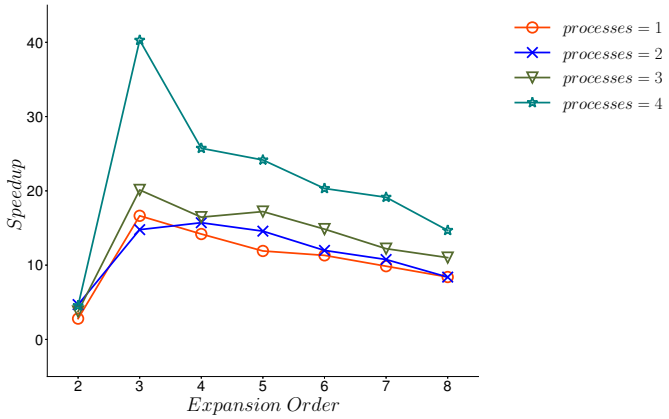


Fig. 7: Ratio of execution time of optimized matrix-free operator with SIMD to Multilevel Static Condensation method for unsteady cases (mesh resolution 40×40)

In addition, the performance of the Multilevel Static Condensation method is also compared with the matrix-free approach for the unsteady cases. As shown in Figure 7, the plotted results are the ratio of execution time of the optimized matrix-free operator with SIMD to that of Multilevel Static Condensation method. For the unsteady cases in this study, it can be seen that the Multilevel Static Condensation method is much more efficient than the matrix-free approach. Overall, the speed of the matrix-free approach is 10-30 times slower. As the number of processes increases in parallel execution, the speed up of multilevel static condensation method becomes more overt. This is opposite to the observation in the steady cases in previous sections, because the system matrix is significantly condensed in the static condensation method and the resultant system matrix must be solved repetitively at each time step. Therefore, its efficiency surpasses the matrix-free approach. In addition, the matrix operator is only constructed once at the first time step in current implementation of the multilevel static condensation method, which further increases its overall efficiency over the matrix-free approach in those unsteady cases. If either the scale of the problem and the dimensions of the global sparse matrices are large, or the global system must be solved multiple times, the performance of the multilevel static condensation method may be more efficient. This is because the dimensions of the matrix is reduced significantly by the condensation technique and the initial setup cost of this is absorbed by the many applications of the operator. However, the static condensation approach comes at a higher memory cost due to the need to construct all local matrix operators, as well as an increased number of communications, which may make the method less efficient in large parallel simulations.

5 Concluding remarks

The performance of basic and optimized matrix-free operators and multilevel static condensation method were compared in this study. The execution time of either the overall simulation or the time taken by the operator construction and backward transformation were employed as the metrics to assess their efficiencies. Overall, the optimized matrix-free operator with SIMD is always much more efficient, about one order of magnitude higher, than the basic matrix-free operator. The SIMD vectorization technique accounts for about 0.2 to 0.4% speed up of current implementation of the optimized matrix-free operator; however, the performance of SIMD drops if the computing resource is limited and the CPU cores are all utilised during a parallel execution. In some cases, e.g., the steady cases in this study, the matrix-free approach is slightly more efficient than the multilevel static condensation method. For transient cases, the implementation of multilevel static condensation can significantly outperform the matrix-free approach. However, it should be noted that the conclusion drawn above about the multilevel static condensation method and matrix-free operator depends on many factors, e.g., the number of time steps in transient cases, the scale of

the problem or the expansion order and this may not carry over to larger-scale calculations.

Bibliography

[1] Hui Xu, Chris D Cantwell, Carlos Monteserin, Claes Eskilsson, Allan P Engsig-Karup, and Spencer J Sherwin. Spectral/hp element methods: Recent developments, applications, and perspectives. *Journal of Hydrodynamics*, 30(1):1–22, 2018.

[2] David Moxey, Roman Amici, and Mike Kirby. Efficient matrix-free high-order finite element evaluation for simplicial elements. *SIAM Journal on Scientific Computing*, 42(3):C97–C123, 2020.

[3] Chris D Cantwell, David Moxey, Andrew Comerford, Alessandro Bolis, Gabriele Rocco, Gianmarco Mengaldo, Daniele De Grazia, Sergey Yakovlev, J-E Lombard, Dirk Ekelschot, et al. Nektar++: An open-source spectral/hp element framework. *Computer physics communications*, 192:205–219, 2015.

[4] George Karniadakis and Spencer Sherwin. *Spectral/hp element methods for computational fluid dynamics*. Oxford University Press, 2013.

[5] Barry Smith, Petter E. Bjrstad, and William Gropp. *Domain decomposition: parallel multilevel methods for elliptic partial differential equations*. Cambridge University Press, New York, 1996.