# SUPPLEMENTARY MATERIALS: MURPHY - A SCALABLE MULTIRESOLUTION FRAMEWORK FOR SCIENTIFIC COMPUTING ON 3D BLOCK-STRUCTURED COLLOCATED GRIDS

THOMAS GILLIS [*] AND WIM M. VAN REES [*†]

**SM1. Implementation details - filters bank.** In this section we detail the exact construction of the filter banks, as used in the software. From an implementation perspective, two approaches have been considered in the literature: the step-by-step implementation of the lifting scheme, as in [4], or the filter-based approach as in [7]. Although the operation count is smaller in the first approach and a factor 2 has been reported in the complexity [2], we follow the second approach which provides significant simplifications especially in multiple dimensions. Further, the memory layout in the implementation is achieved more naturally when interlacing the scaling and detail coefficients:

$$u^L = \left[ \dots \lambda^L_{k-1} \, , \, \gamma^L_{k-1} \, , \, \lambda^L_k \, , \, \gamma^L_k \, , \, \lambda^L_{k+1} \, , \, \gamma^L_{k+1} \, , \, \dots \right] \; . \tag{SM1.1}$$

This does not affect the $G^a$ and $H^a$ filters used in the analysis, but requires the definition of two new filters $J^s$ and $K^s$ to replace $H^s$ and $G^s$ during the synthesis operation, such that

$$\lambda^{L+1}_{2k} = J^s \, u^L \qquad \lambda^{L+1}_{2k+1} = K^s \, u^L \; , \tag{SM1.2}$$

where the array $u^L$ in the left equation is implicitly assumed to be centered on $\lambda^L_k$, and in the right equation on $\gamma^L_k$. This change of perspective is strictly equivalent to the classical $H^s/G^s$ filters and has been done to simplify the implementation. Considering this new approach, the filter coefficients for wavelet `2.0`, wavelet `4.0`, and wavelet `6.0` are given in Table SM1.1, and the coefficients for wavelet `2.2`, wavelet `4.2`, and wavelet `6.2` are given in Table SM1.2.

| | | | |
|---|---|---|---|
| $H^a$ | | $1$ | |
| $G^a$ | $-\frac{1}{2}$ | $1$ | $-\frac{1}{2}$ |
| $J^s$ | | $1$ | |
| $K^s$ | $\frac{1}{2}$ | $1$ | $\frac{1}{2}$ |

(a) wavelet `2.0`

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| $H^a$ | | | | $1$ | | | |
| $G^a$ | $\frac{1}{16}$ | $0$ | $-\frac{9}{16}$ | $1$ | $-\frac{9}{16}$ | $0$ | $\frac{1}{16}$ |
| $J^s$ | | | | $1$ | | | |
| $K^s$ | $-\frac{1}{16}$ | $0$ | $\frac{9}{16}$ | $1$ | $\frac{9}{16}$ | $0$ | $-\frac{1}{16}$ |

(b) wavelet `4.0`

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $H^a$ | | | | | | $1$ | | | | | |
| $G^a$ | $-\frac{3}{256}$ | $0$ | $\frac{25}{256}$ | $0$ | $-\frac{75}{128}$ | $1$ | $-\frac{75}{128}$ | $0$ | $\frac{25}{256}$ | $0$ | $-\frac{3}{256}$ |
| $J^s$ | | | | | | $1$ | | | | | |
| $K^s$ | $\frac{3}{256}$ | $0$ | $-\frac{25}{256}$ | $0$ | $\frac{75}{128}$ | $1$ | $\frac{75}{128}$ | $0$ | $-\frac{25}{256}$ | $0$ | $\frac{3}{256}$ |

(c) wavelet `6.0`

Table SM1.1: Non-lifted interpolating wavelet filter coefficients as operated on interleaved scaling and detail coefficients. The filters $H^a$ and $J^s$ are centered on (even) scaling coefficients, and the filters $G^a$ and $K^s$ are centered on odd scaling coefficients and detail coefficients, respectively.

**SM2. Algorithms and implementation.** In this section we describe in detail the different algorithms as implemented in our presented software framework. We refer to section 3 for additional explanations and context regarding the different steps. In Algorithm 1 and Algorithm 2 we detail the implementation of the ghosting procedure, while in Algorithm 3 we expose the implementation of the adaptation of the grid.

**SM2.1. Ghost computation.** We describe here the technicalities of the ghost computation implementation. To facilitate the discussion, we introduce some notation inspired by the `p4est` naming convention. Any block $b$ is part of the total set of blocks $\mathbb{B}$ within the grid, which are distributed among multiple MPI ranks. In this section we understand $b$ to contain the cartesian grid data of size $N_b^3$, as well as a ghost region on the same level as $b$ extending $N_g$ points in each dimension on each side of the block,

[*]Department of Mechanical Engineering, Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, MA 02139, USA
[†]Corresponding author: wvanrees@mit.edu

| | | | | | |
|---|---|---|---|---|---|
| $H^a$ | $-\frac{1}{8}$ | $\frac{1}{4}$ | $\frac{3}{4}$ | $\frac{1}{4}$ | $-\frac{1}{8}$ |
| $G^a$ | | $-\frac{1}{2}$ | $1$ | $-\frac{1}{2}$ | |
| $J^s$ | | $-\frac{1}{4}$ | $1$ | $-\frac{1}{4}$ | |
| $K^s$ | $-\frac{1}{8}$ | $\frac{1}{2}$ | $\frac{3}{4}$ | $\frac{1}{2}$ | $-\frac{1}{8}$ |

(a) wavelet `2.2`

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $H^a$ | $\frac{1}{64}$ | $0$ | $-\frac{1}{8}$ | $\frac{1}{4}$ | $\frac{23}{32}$ | $\frac{1}{4}$ | $-\frac{1}{8}$ | $0$ | $\frac{1}{64}$ |
| $G^a$ | | $\frac{1}{16}$ | $0$ | $-\frac{9}{16}$ | $1$ | $-\frac{9}{16}$ | $0$ | $\frac{1}{16}$ | |
| $J^s$ | | | | $-\frac{1}{4}$ | $1$ | $-\frac{1}{4}$ | | | |
| $K^s$ | $\frac{1}{64}$ | $-\frac{1}{16}$ | $-\frac{1}{8}$ | $\frac{9}{16}$ | $\frac{23}{32}$ | $\frac{9}{16}$ | $-\frac{1}{8}$ | $-\frac{1}{16}$ | $\frac{1}{64}$ |

(b) wavelet `4.2`

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $H^a$ | $-\frac{3}{1024}$ | $0$ | $\frac{11}{512}$ | $0$ | $-\frac{125}{1024}$ | $\frac{1}{4}$ | $\frac{181}{256}$ | $\frac{1}{4}$ | $-\frac{125}{1024}$ | $0$ | $\frac{11}{512}$ | $0$ | $-\frac{3}{1024}$ |
| $G^a$ | | $-\frac{3}{256}$ | $0$ | $\frac{25}{256}$ | $0$ | $-\frac{75}{128}$ | $1$ | $-\frac{75}{128}$ | $0$ | $\frac{25}{256}$ | $0$ | $-\frac{3}{256}$ | |
| $J^s$ | | | | | | $-\frac{1}{4}$ | $1$ | $-\frac{1}{4}$ | | | | | |
| $K^s$ | $-\frac{3}{1024}$ | $\frac{3}{256}$ | $\frac{11}{512}$ | $-\frac{25}{256}$ | $-\frac{125}{1024}$ | $\frac{75}{128}$ | $\frac{181}{256}$ | $\frac{75}{128}$ | $-\frac{125}{1024}$ | $-\frac{25}{256}$ | $\frac{11}{512}$ | $\frac{3}{256}$ | $-\frac{3}{1024}$ |

(c) wavelet `6.2`

Table SM1.2: Lifted interpolating wavelet filter coefficients as operated on interleaved scaling and detail coefficients. The filters $H^a$ and $J^s$ are centered on (even) scaling coefficients, and the filters $G^a$ and $K^s$ are centered on odd scaling coefficients and detail coefficients, respectively.

where $N_g$ is determined from the PDE requirements or wavelet support, depending on the operation. A subset of $\mathbb{B}$ is the mirror group $\mathbb{M}$, defined as the set of blocks that are the neighbor of at least one other block on another rank, and thus have to be accessed through MPI communications. For each block $b$, we further define 6 different neighbor sets. All neighbors that exist on the same level as $b$ fall in the group $\mathbb{G}^0$, all neighbors that are coarser than $b$ fall in the group $\mathbb{G}^-$, and all neighbors that are finer than $b$ fall in the group $\mathbb{G}^+$. Each of these group can either exist on the same rank as $b$, in which case they are denoted as 'local' $\mathbb{G}_L^{0/-/+}$, or on another rank in which case they are denoted as 'global' $\mathbb{G}_G^{0/-/+}$. The ghost point computation is divided into two parts. For a block $b$, the first step combines the values of coarser neighbors $\mathbb{G}^-_{G/L}$ and same level neighbors $\mathbb{G}^0_{G/L}$ with a coarse perspective of the data in $b$ to be able to compute a local refinement of the ghost region and retrieve the required ghost points (see Algorithm 1). This step implements the process outlined in the main section in subsection 2.2.2. In the second step, block $b$ uses the refined region of step 1 (including ghosts) to coarsen again, leading to coarse ghost points that block $b$ then communicates to neighbors $\mathbb{G}^-_{G/L}$ (see Algorithm 2).

*Part 1: Coarser and same-level neighbors* . First, due to the continuity in memory needed by the RMA window (our current memory allocation does not guarantee continuity for a given field across blocks), we start by a copy of all the $b \in \mathbb{M}$ into the buffer reserved for the communication, $\text{buf}_{\text{RMA}}$. The ghost values from same-level neighbors are trivially obtained by accessing the values needed by the ghost region through a memory copy for $\mathbb{G}^0_L$ and using `MPI_Get()` for $\mathbb{G}^0_G$.

To obtain ghost values from coarser and same-level neighbors of our current block $b$, we have to use the refinement operation as described in subsection 2.2. This operation relies on the grid values of $\mathbb{G}^-$ and the even values of our current block $b$ and $\mathbb{G}^0$. We proceed by gathering all the required values for the computation of the ghost into a temporary buffer $\tilde{b}$ (whose size is $1/8^{\text{th}}$ of the block-size, extended with a coarsened representation of the ghost region). The contribution of the coarser neighbors are first obtained through a copy or `MPI_Get()` of the required values into $\tilde{b}$, and we gather the even values of the same-level neighbors, together with the even data of the current block. The data inside $\tilde{b}$ associated with fine-level neighbors is left blank and ignored throughout this process. Once all the needed data are gathered into $\tilde{b}$, we apply any domain boundary conditions at the coarse level. Finally, we perform the refinement operation locally on the rank associated with $b$ and read out the computed ghost values within the regions overlapping with $\mathbb{G}^0$ and $\mathbb{G}^-$ from the refined buffer.

*Part 2: Fine-level neighbors* . In the second part of the algorithm, we use the data in $b$ and its ghost points computed in part 1 to compute coarse-level ghost points for $\mathbb{G}^-$, with the individual steps described in Algorithm 2. We first apply any specified domain boundary conditions on $b$ if required, and then perform the substitution step as detailed in subsection 2.2.4. We then coarsen $b$ along the lines described in subsection 2.2 and store the coarsened data into $\tilde{b}$. Once the coarsening is completed, block $b$ copies the required coarse-level ghost points to $\mathbb{G}^-_L$ or issues a `MPI_Put()` for $\mathbb{G}^-_G$ to write into the coarser neighbor's memory. When completed, we are left with the application of the boundary conditions as all the ghost informations are now complete.

With these two steps, all block interfaces in the domain can be handled. In the way explained above, the ghost reconstruction would correspond to a tree traversal from the coarsest to the finest levels in part 1, and back up to the coarsest level in part 2. However, choosing $N_b$ larger than the support of the wavelet filters together with the 2:1 constraint in resolution jumps guarantees that our algorithm for ghost reconstruction parallelizes efficiently across the different levels, with only independent synchronizations for each block in the `MPI_Win_Wait()` call of Algorithm 1 independent of the level on which it exists.

**Algorithm 1:** Ghosting - Part 1

```
1  $\mathrm{buf}_{\mathrm{RMA}} \leftarrow$ copy($b \in \mathbb{M}$)
2  MPI_Win_Post()
3  MPI_Win_Start()
4  foreach $b \in \mathbb{B}$ do
5      $b \leftarrow$ copy($\mathbb{G}^0_L$), MPI_Get($\mathbb{G}^0_G$)
6      $\tilde{b} \leftarrow$ copy($\mathbb{G}^-_L$), sample($\mathbb{G}^0_L$)
7      $\tilde{b} \leftarrow$ MPI_Get($\mathbb{G}^-_G$), MPI_Get(sample($\mathbb{G}^0_G$))
8  end
9  MPI_Win_Complete()
10 MPI_Win_Wait()
11 foreach $b \in \mathbb{B}$ do
12     $\tilde{b} \leftarrow$ sample($b$)
13     $\tilde{b} \leftarrow$ BoundaryCondition()
14     $b \leftarrow$ WaveletRefine($\tilde{b}$)
15 end
```

**Algorithm 2:** Ghosting - Part 2

```
1  MPI_Win_Post()
2  MPI_Win_Start()
3  foreach $b \in \mathbb{B}$ do
4      $b \leftarrow$ BoundaryCondition()
5      $b \leftarrow$ WaveletSubstitution()
6      $\tilde{b} \leftarrow$ WaveletCoarsen($b$)
7      $\mathbb{G}^-_L \leftarrow$ copy($\tilde{b}$)
8      $\mathbb{G}^-_G \leftarrow$ MPI_Put($\tilde{b}$)
9  end
10 MPI_Win_Complete()
11 MPI_Win_Wait()
12 foreach $b \in \mathbb{B}$ do
13     $b \leftarrow \mathrm{buf}_{\mathrm{RMA}}$
14     $b \leftarrow$ BoundaryCondition()
15 end
```

### SM2.2. Grid adaptation.

The adaptation as detailed in Algorithm 3 is an iterative procedure starting on a grid $\mathbb{B}_k$, where the iteration $k$ is divided in multiple steps:

1. Ghost values computation and detail computation: the ghost values are updated dimension by dimension first on the field used for the detail computation and then on the rest of the fields present on the grid.
2. In `WaveletCriterion(`$b \in \mathbb{B}_k$`)` the detail coefficients are communicated and computed, either directly for scalar fields or on a component-by-component basis for vector and tensor fields.
3. Every block gets a corresponding status that expresses its desired evolution given the detail coefficient values and the refinement $\epsilon_r$ and coarsening $\epsilon_c$ tolerances:
   (a) `M_ADAPT_FINER`: if any of the detail coefficient is $\geq \epsilon_r$ for any dimension,
   (b) `M_ADAPT_COARSER`: if every detail coefficient is $\leq \epsilon_c$ for every dimension,
   (c) `M_ADAPT_SAME`: if no other status is assigned.
4. Once all blocks have their desired status assigned, we check compliance against our grid adaptation policy to ensure that the grid adaptation is valid and consistent with the multiresolution theory; if needed, we change the status of individual block to enforce compliance. The policy consists of the following rules:

   (a) we give priority to the refinement: if a block's coarser neighbor wants to refine, the latter cannot be coarsened;
   (b) the need of a fine block to be refined has to be "propagated" to its coarser neighbor: if a block has a finer neighbor which wants to refine, the latter must refine first to not break the 2:1 condition; similarly, if a block has a coarser neighbor, that neighbor has to be refined first;
   (c) if a block has been refined in a previous iteration, it cannot be coarsened.

   In addition, we enable the option to enforce possible user-defined bounds on the minimum and/or maximum level permitted, if needed; these bounds are not used in the results of this work.
5. Once each block has received its final status, we use the `p4est` library to create the meta-structure associated with a new set of blocks $\mathbb{B}^*_k$.
6. Refine or coarsen the blocks as needed to obtain the updated grid $\mathbb{B}_{k+1}$ through the `WaveletInterpolate()` function.
7. Partition the grid using `p4est` .
8. Synchronize the executed adaptation step of each block to its neighbors.
9. Update the adapted fields in `WaveletUpdateAfterCoarsening(`$b \in \mathbb{B}_{k+1}$`)` to enforce that fine blocks receive updated scaling coefficients if one or more neighbors have been coarsened (subsection 2.2.1).
10. Obtain the number of adapted blocks over the whole grid (implemented through a non-blocking reduce operation). If no blocks have been adapted, we have reached the final grid with $\|\gamma\|_\infty \leq \epsilon_r$. Else, move to the next iteration $k+1$.

Throughout these steps, we rely on `p4est` functions to support the coarsening, the refinement, and the partitioning of the grid; however, this is mostly limited to handling the metadata, as we have implemented the block refinement, coarsening and partitioning ourselves to exploit asynchronous and non-blocking MPI calls.

**Algorithm 3:** grid adaptation

```
1  while adapt do
2      Ghost($b \in \mathbb{B}_k$)
3      WaveletCriterion($b \in \mathbb{B}_k$)
4      SyncStatus()
5      EnforceStatusPolicy($b \in \mathbb{B}_k$)
6      p4estCoarsen()
7      p4estRefine()
8      $\mathbb{B}_{k+1} \leftarrow$ WaveletInterpolate($b^* \in \mathbb{B}^*_{k+1}$)
9      p4estPartition()
10     SyncStatus()
11     WaveletUpdateAfterCoarsening($b \in \mathbb{B}_{k+1}$)
12     $\mathbb{B}_k \leftarrow \mathbb{B}_{k+1}$
13 end
```

## SM3. Time integration scheme.

As a time integrator we are using the RK3-TVD [5, 6] also know as RK3-SSP. This is an explicit in time, 3 step Runge-Kutta scheme, which only requires two temporary buffers. The integration for $\dot{u} = f(t,u)$ from time $t^n$ to $t^{n+1}$ with $\Delta t = t^{n+1} - t^n$ is given by the three stage equations:

$$y_1 = \Delta t f\left(t^n, u^n\right) + u^n \ , \qquad y_2 = \frac{1}{4}\left[\Delta t f\left(t+\Delta t, y_1\right) + y_1\right] + \frac{3}{4} u^n \ , \qquad u^{n+1} = \frac{2}{3}\left[\Delta t f\left(t + \frac{\Delta t}{2} y_2\right) + y_2\right] + \frac{1}{3} u^n \ . \tag{SM3.1}$$

## SM4. Finite differences scheme.

For the spatial discretization of the derivatives in the advection equation, we rely on the `CONS-3` scheme, which is a fixed-weight version of a `WENO` schemes. First, given a divergence-free velocity field, the linear advection equation is considered in conservative form:

$$\frac{\partial \phi}{\partial t} + \mathbf{u} \cdot \nabla \phi = 0 \quad \Leftrightarrow \quad \frac{\partial \phi}{\partial t} + \nabla \cdot \left(\mathbf{u}\, \phi\right) = 0 \ , \tag{SM4.1}$$

with flux function $f = \mathbf{u}\, \phi$. We consider a 1D version of this equation, discretized on a uniform grid, *i.e.* the grid cell $i$ spans $[x_{i-1/2}\ ;\ x_{i+1/2}]$:

$$\frac{\partial \phi_i}{\partial t} = -\left(f_{i+1/2} - f_{i-1/2}\right) \frac{1}{h} \ , \tag{SM4.2}$$

where $f_{i\pm1/2}$ is the flux at the cell boundary. Here we follow [8, section 2.1] to derive a conservative finite-difference form for the flux reconstruction. To maintain stability [8, section 2.1] the flux terms must be decomposed into a positive $f^+$ and negative part

$f^-$ such that

$$\frac{\partial f^+}{\partial \phi} > 0 \quad \text{and} \quad \frac{\partial f^-}{\partial \phi} < 0 \; . \tag{SM4.3}$$

For our equations in 1D we find $f^+ = u\phi$ if $u > 0$ and $f^- = u\phi$ if $u < 0$. Since the velocity information at the interface is not directly available, we use a simple reconstruction, $u_{i+1/2} = 1/2\ (u_{i+1} + u_{i-1})$. This approximation gives the sign of the velocity at the interface from the velocity in each grid cell, which determines the choice between $f^+$ and $f^-$ at each cell interface.

The actual flux computation can be done in different ways, leading to essentially non-oscillatory (ENO) and weighted ENO (WENO) interpolation formulas. Using three points in the flux calculation, one can choose between two stable stencils to compute $f^+_{i+1/2}$

$$S_0: \quad f^+_{i+1/2} = -1/2\, f^+_{i-1} + 3/2\, f^+_i \qquad S_1: \quad f^+_{i+1/2} = 1/2\, f^+_i + 1/2\, f^+_{i+1} \; . \tag{SM4.4}$$

Further, one can associate a smoothness indicator to each stencil $\beta_0 = \left(f^+_i - f^+_{i-1}\right)^2$ and $\beta_1 = \left(f^+_{i+1} - f^+_i\right)^2$. The standard ENO approach relies on the $\beta$ indicators to choose the best stencil, *i.e.* we choose either $S_0$ or $S_1$ to evaluate the flux. The WENO technique instead combines the stencils together relying on weights associated to each of them ($w_0$ and $w_1$ respectively) in order to obtain the most accurate evaluation possible.

In the case of the stencils $S_0$ and $S_1$ one can combine them together to reach a third-order stencil using the "ideal" weights $w_0 = \gamma_0 = 1/3$ and $w_1 = \gamma_1 = 2/3$. By doing so, one obtains the conservative third-order stencil `CONS-3` we used in the manuscript. Along similar lines, the fifth order conservative stencil `CONS-5` can be derived from the optimal weights of `WENO-5`. Tables Table SM4.1 and Table SM4.2 summarize the flux definitions and optimal weights $\gamma$ for each of these schemes. For non-smooth fields the weights can be adapted locally based on the $\beta$ smoothness indicators, and we implemented the WENO-Z version [1, 3] in our code – however for the results in this manuscript we only use the fixed-weight schemes.

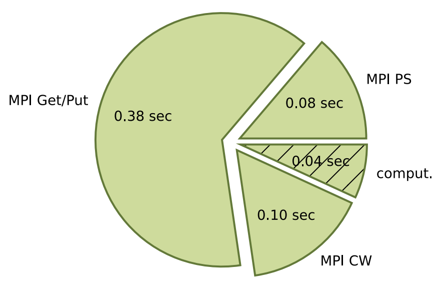
| | | | |
|---|---|---|---|
| $f^+_{i+1/2}$ | $S^+_0 = -1/2 f_{i-1} + 3/2 f_i$ | $\beta^+_0 = \left(f_i - f_{i-1}\right)^2$ | $\gamma^+_0 = 1/3$ |
| | $S^+_1 = 1/2 f_i + 1/2 f_{i+1}$ | $\beta^+_1 = \left(f_{i+1} - f_i\right)^2$ | $\gamma^+_1 = 2/3$ |
| $f^-_{i-1/2}$ | $S^-_0 = 1/2 f_{i-1} + 1/2 f_i$ | $\beta^-_0 = \beta^+_0 = \left(f_i - f_{i-1}\right)^2$ | $\gamma^-_0 = 2/3$ |
| | $S^-_1 = 3/2 f_i - 1/2 f_{i+1}$ | $\beta^-_1 = \beta^+_1 = \left(f_{i+1} - f_i\right)^2$ | $\gamma^-_1 = 1/3$ |

Table SM4.1: WENO stencils - order 3. Setting the weights equal to the $\gamma$'s leads to the third-order `CONS-3` scheme.

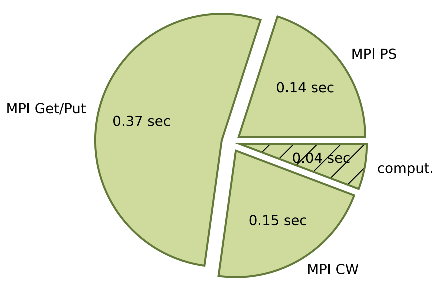
| | | | |
|---|---|---|---|
| $f^+_{i+1/2}$ | $S^+_0 = 1/3 f_{i-2} - 7/6 f_{i-1} + 11/6 f_i$ | $\beta^+_0 = \frac{1}{4}\left(f_{i-2} - 4f_{i-1} + 3f_i\right)^2 + \frac{13}{12}\left(f_{i-2} - 2f_{i-1} + f_i\right)^2$ | $\gamma^+_0 = 1/10$ |
| | $S^+_1 = -1/6 f_{i-1} + 5/6 f_i + 1/3 f_{i+1}$ | $\beta^+_1 = \frac{1}{4}\left(-f_{i-1} + f_{i+1}\right)^2 + \frac{13}{12}\left(f_{i-1} - 2f_i + f_{i+1}\right)^2$ | $\gamma^+_1 = 3/5$ |
| | $S^+_2 = 1/3 f_i + 5/6 f_{i+1} - 1/6 f_{i+2}$ | $\beta^+_2 = \frac{1}{4}\left(-3f_i + 4f_{i+1} - f_{i+2}\right)^2 + \frac{13}{12}\left(f_i - 2f_{i+1} + f_{i+2}\right)^2$ | $\gamma^+_2 = 3/10$ |
| $f^-_{i-1/2}$ | $S^-_0 = -1/6 f_{i-2} + 5/6 f_{i-1} + 1/3 f_i$ | $\beta^-_0 = \beta^+_0 = \frac{1}{4}\left(f_{i-2} - 4f_{i-1} + 3f_i\right)^2 + \frac{13}{12}\left(f_{i-2} - 2f_{i-1} + f_i\right)^2$ | $\gamma^-_0 = 3/10$ |
| | $S^-_1 = 1/3 f_{i-1} + 5/6 f_i - 1/6 f_{i+1}$ | $\beta^-_1 = \beta^+_1 = \frac{1}{4}\left(-f_{i-1} + f_{i+1}\right)^2 + \frac{13}{12}\left(f_{i-1} - 2f_i + f_{i+1}\right)^2$ | $\gamma^-_1 = 3/5$ |
| | $S^-_2 = 11/6 f_i - 7/6 f_{i+1} + 1/3 f_{i+2}$ | $\beta^-_2 = \beta^+_2 = \frac{1}{4}\left(-3f_i + 4f_{i+1} - f_{i+2}\right)^2 + \frac{13}{12}\left(f_i - 2f_{i+1} + f_{i+2}\right)^2$ | $\gamma^-_2 = 1/10$ |

Table SM4.2: WENO stencils - order 5. Setting the weights equal to the $\gamma$'s leads to the fifth-order `CONS-5` scheme.

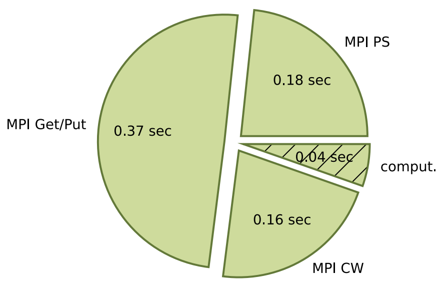
**SM5. Weak scalability analysis.** In this section we provide more details on the scalability of the different sub-operations discussed in section 6 of the main text. In Figure SM5.1 we show the breakdown of the timings during the ghost computation, where the hatched area represent purely computational operations that are expected to scale perfectly. In Figure SM5.2 we decompose similarly the operations involved in the stencil computation, and in Figure SM5.3 we do the same for the grid adaptation.



(a) 1 node - 32 ranks

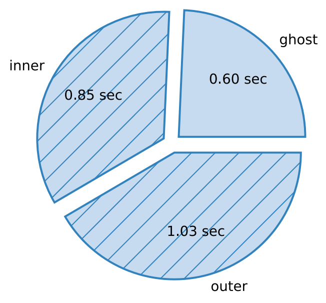


(b) 128 node - 4,096 ranks

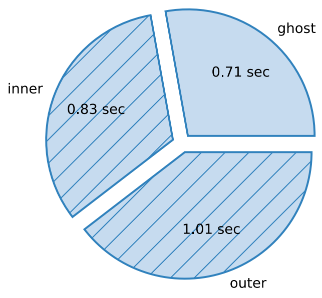


(c) 512 node - 16,384 ranks

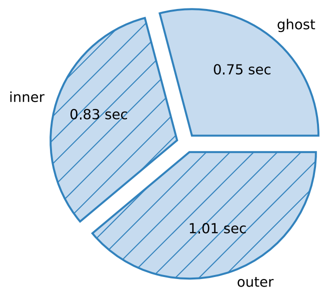
Figure SM5.1: Breakdown of the time spent during ghost reconstruction in a weak scalability test. The hatched region indicates operations only involving computations and no communications. The regions marked with “`MPI PS`” and “`MPI CW`” represent the calls to perform the active synchronization, “`MPI Get/Put`“ contains the put and get accesses, and “`comp.`” is the time spent during the wavelet-based refinement and coarsening required to compute the ghost values.



(a) 1 node - 32 ranks



(b) 128 node - 4,096 ranks



(c) 512 node - 16,384 ranks

Figure SM5.2: Breakdown of the time spent during stencil computations in a weak scalability test. The hatched regions indicates operations only involving computations and no communications. The regions marked “`inner`” and “`outer`” are the computation of the stencil on the the inner and outer points in a block respectively, and “`ghost`” represents the computation of the ghost values overlapping with the “`inner`” computations as detailed in Figure SM5.1

**References.**

[1] R. Borges, M. Carmona, B. Costa, and W. S. Don, *An improved weighted essentially non-oscillatory scheme for hyperbolic conservation laws*, Journal of Computational Physics, 227 (2008), pp. 3191–3211, https://doi.org/https://doi.org/10.1016/j.jcp.2007.11.038, https://www.sciencedirect.com/science/article/pii/S0021999107005232.

[2] I. Daubechies and W. Sweldens, *Factoring wavelet transforms into lifting steps*, Journal of Fourier Analysis and Applications, 4 (1998), pp. 247–269, https://doi.org/10.1007/BF02476026, https://doi.org/10.1007/BF02476026.

[3] W.-S. Don and R. Borges, *Accuracy of the weighted essentially non-oscillatory conservative finite difference schemes*, Journal of Computational Physics, 250 (2013), pp. 347–372, https://doi.org/https://doi.org/10.1016/j.jcp.2013.05.018, https://www.sciencedirect.com/science/article/pii/S0021999113003501.

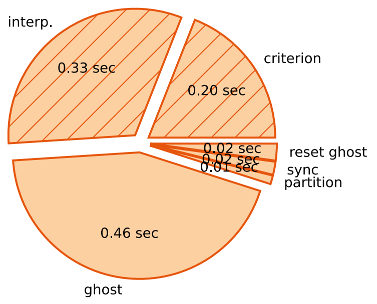
[4] G. Fernandez, S. Periaswamy, and W. Sweldens, *Liftpack: a software package for wavelet transforms using lifting*, in Proc.SPIE, vol. 2825, 10 1996, https://doi.org/10.1117/12.255250, https://doi.org/10.1117/12.255250.

[5] S. Gottlieb and C.-W. Shu, *Total variation diminishing Runge-Kutta schemes*, Mathematics of Computation, 67 (1998), pp. 73–85, https://doi.org/10.1090/S0025-5718-98-00913-2.

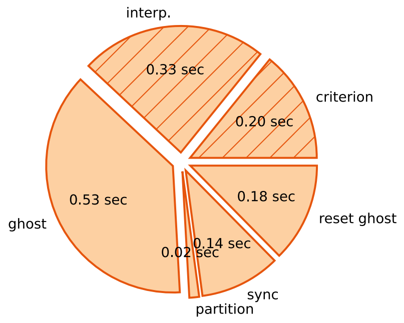
[6] S. Gottlieb, C.-W. Shu, and E. Tadmor, *Strong stability-preserving high-order time discretization methods*, SIAM Review, 43 (2001), pp. 89–112, https://doi.org/10.1137/S003614450036757X, https://doi.org/10.1137/S003614450036757X.

[7] D. Rossinelli, B. Hejazialhosseini, W. van Rees, M. Gazzola, M. Bergdorf, and P. Koumoutsakos, *MRag-I2d: Multi-resolution adapted grids for remeshed vortex methods on multicore architectures*, Journal of Computational Physics, 288 (2015), pp. 1–18, https://doi.org/http://dx.doi.org/10.1016/j.jcp.2015.01.035.

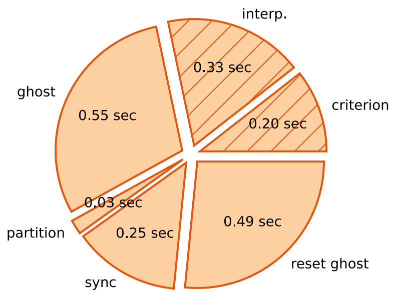
[8] C.-W. Shu, *Essentially non-oscillatory and weighted essentially non-oscillatory schemes for hyperbolic conservation laws*, ICASE 97-65, NASA, November 1997.



(a) 1 node - 32 ranks



(b) 128 node - 4,096 ranks



(c) 512 node - 16,384 ranks

Figure SM5.3: Breakdown of the time spent during grid adaptation operations in a weak scalability test. The hatched regions indicates operations only involving computations and no communications. The region marked "`interp.`" represents the coarsening/refinement of the blocks through the wavelets, "`criterion`" stands for the computation of the detail coefficients to evaluate the desired status of each block (refine, compress, or unaltered), "`reset ghost`" includes the reinitialization of the ghost meta-data including the `MPI_Win_create` and `MPI_Win_free` calls, "`sync`" contains the synchronizations and reductions on the block statuses to enforce the adaptation policy, "`partition`" is the load balancing of the grid and the re-distribution of the blocks, and "`ghost`" represents the computation of the ghost points needed for the coarsening/refinement of the blocks and the computation of the detail coefficients.