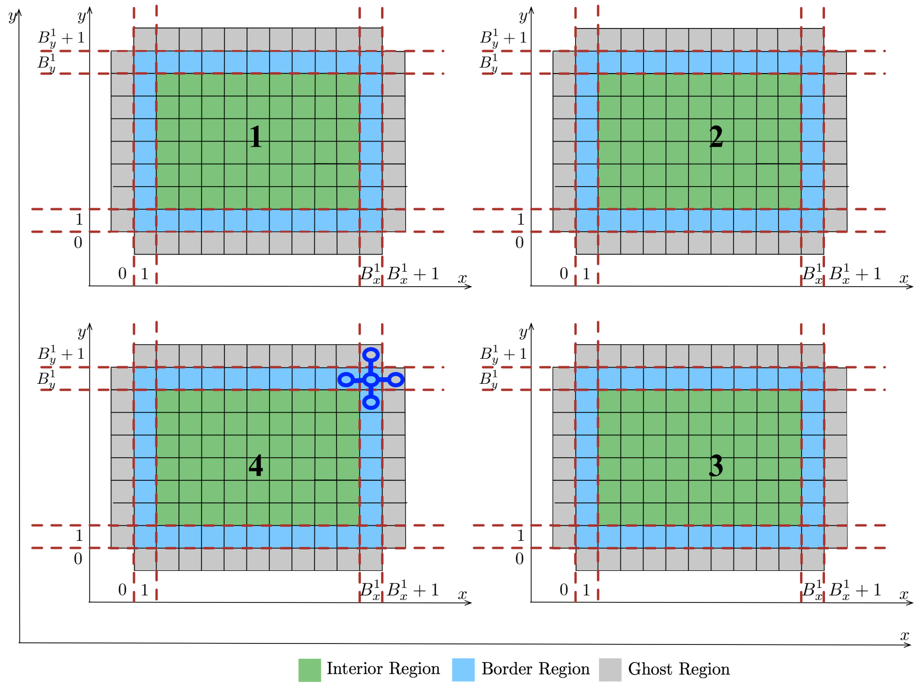


Figure 1: 4 process decomposition of a 22x16 stencil. Process labels start in the top left and proceed clockwise. Local variables of state are also illustrated for indexing a 2D array, 0,1,B_x^1,etc.

1 Motivation And Basic Terminology

The solutions of partial differential equations (PDEs) are classic problems for parallel computers and use what are known as stencil methods. Figure 1 illustrates an example of a *stencil* for a 2D PDE at grid cell (B_x^1, B_y^1) of process 4. At cell (B_x^1, B_y^1) there is a single blue circle surrounded by 4 circles. This structure is known as a *5-point stencil* for the 5 blue circles in the image, and it is derived from the underlying discretization of the PDE being solved. The grid cell containing the center circle is the spatially dependent value of the PDE at a given point in time. The cells of the adjacent circles are the spatial dependencies for the center cell.

Because these spatial dependencies are functions of time, whenever a PDE's value is updated once at a grid cell, the dependency structure changes for that grid cell. The unit of time taken for a cell update is referred to as the *timestep* of that cell. Figure 2 illustrates the timestep at each cell of process 4 from figure 1, for some arbitrary point during the execution of a stencil method. It demonstrates how the spatial dependencies of each cell can change with time. The cell (B_x^1, B_y^1) has a time step of 3, and so do all of its adjacent cells. When a cell has neighbors above, below, left, and right of it which contain values computed from the same timestep, we'll say the cell's dependencies are *valid* at a given point in time. When any one of its neighbors does not contain a value from the same timestep, then its depencies are *invalid*. An example of invalid dependencies is given in cell $(B_x^1, 1)$ in figure 2, where the value of the PDE is only completed up to the 2nd timestep, and it's neighboring cells contain values from different timesteps.

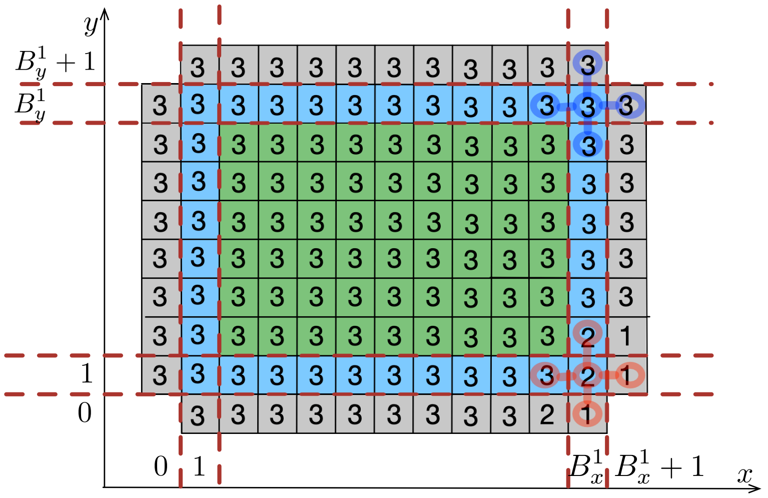
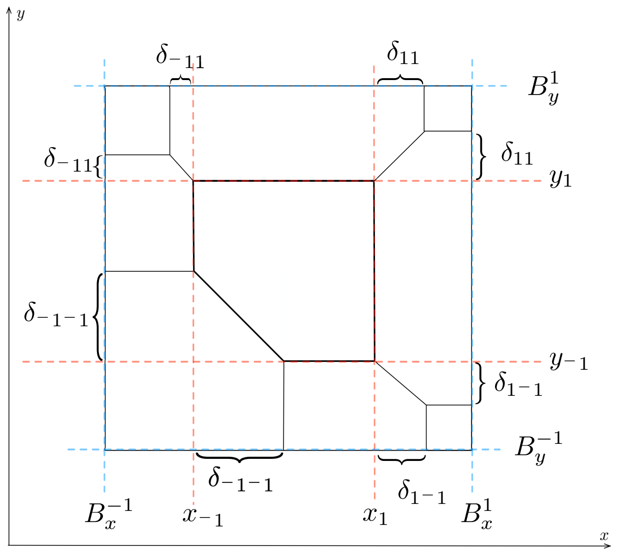
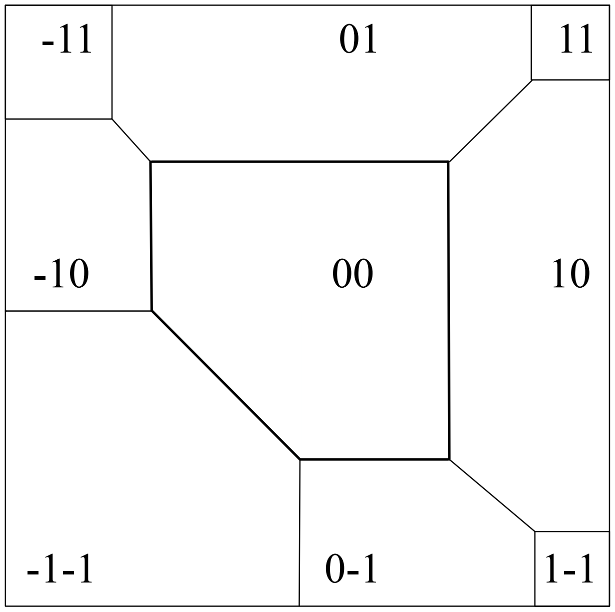


Figure 2: An illustration of grid cell timestep, and its relationship to stencil dependencies. In cell (B_x^1, B_y^1) dependencies are met, and a valid update of the cell can be performed. In cell $(B_x^1, 1)$ the opposite is true. Transparent blue and red circles are overlaid respectively for valid and invalid dependencies.

This dependency structure has an impact on the performance of any parallelization strategy for the stencil method. Parallelizing a large 2D stencil using a SIMD strategy starts with the definition of a global area over which computation is required, followed by a subdivision of that area into 2D grids for each processing unit to manage. Figure 1 illustrates the general principle for 4 processing units, each with separate 11x8 grid portions of the total 22x16 global grid size. While there are indeed 13x10 grid cells for each process subdivision illustrated, the gray cells for each process are in fact redundant, and they hold information which come from the adjacent blue cells held by neighbor processes hence the term *ghost* cells indicated in the diagram key. To make the example explicit, cells $(1,1) - (B_x^1, 1)$ of process 1 need to be reflected in cells $(1, B_y^1+1) - (B_x^1, B_y^1+1)$ on process 4 in order for it to compute cells $(1, B_y^1) - (B_x^1, B_y^1)$. This interprocess dependency is a direct result of the 5 point stencil described above. Because of this dependency structure, each process'



(a) State maintained for the purpose of extrapolating all area boundaries



(b) Region labels, each given by 2 digits in the form $a_x a_y$ throughout the rest of the document. $a_x a_y$ additionally provides a shorthand for the vector $[a_x \quad a_y]^T$, and we therefore refer to these region labels as region vectors, or area vectors throughout the document.

Figure 3: (3a) An illustration of state labels that allow each of the 9 regions to relax their boundaries. (3b) Shorthand labels for each region, and for corresponding area vectors.

illustated blue region (*border region*) must have up to date records in their ghost regions from the corresponding blue region of the nearest process on each side.

This dependency structure between processes is known as a *nearest neighbor* dependency structure, and divides each process' local grid into 9 distinct *regions* depending on where they are positioned relative to ghost cells which must be received from neighbor processes. Continuing to use process 4 in figure 1 as an example, the regions are the green shaded *interior region*, and then 8 blue shaded border regions. The interior region depends only on border regions, and is therefore relatively independent of neighbor processes (that independence will be later quantified). The border regions have 2 types of dependence. The first type depends on only one neighbor process. This is given by cell ranges on the 4 sides of the 2D grid: $(2,1)-(B_x^1-1,1)$, $(1,2)-(1,B_y^1-1)$, $(2,B_y^1)-(B_x^1-1,B_y^1)$, and $(B_x^1,2)-(B_x^1,B_y^1-1)$. In these ranges only the gray ghost cells received from a single process are needed to successfully compute grid cells at each point. The other type of border region depends on 2 processes and exists solely in the corners of the 2D grid at cells: $(1,1)$, $(1,B_y^1)$, (B_x^1,B_y^1), and $(B_x^1,1)$.

Given the 9 static regions described, if a process wishes to execute a complete timestep for each cell in its grid, it must synchronize the pace of its timesteps with each neighbor process . If synchronizations are expensive this creates a performance bottleneck, and it becomes beneficial to develop strategies to relax these synchronization requirements. In order to do so while maintaining valid stencil dependencies reqires relaxing the boundaries of the 9 classically static regions of a 2D stencil on each process. In the next section we'll describe how this is done, before developing a concise approach for performing region updates only when the region contains valid dependencies for each cell.

2 State Variables For Region Decomposition

Figure 3b illustrates a more concise labeling of each of the 9 regions described above, whose boundaries have been relaxed to arbitrary positions. Figure 3a illustrates the minimal set of variables needed to relax region boundaries for a single process, and delimits the regions they collectively form. With these dynamic variables of state, each process has enough information to ensure the grid cells of each region have valid 5 point stencil dependency structures[1]. This is because each state variables captures information about the timestep, and therefore dependency information of each region. To illustrate this point, consider the 'interior region', 00, which has no explicit dependencies. Given that the number of grid cells in 00 dominates the number of grid cells in the border region, we're typically safe to continue shrinking this region and updating it's grid cells regardless of whether neighbor process data has been aquired for the timesteps. Therefore, we define the maximal timestep possible as being the uniform timestep reached by the interior most region, 00,

$$t_{max} \equiv \text{The number of updates completed (timestep) for each cell in region 00.} \tag{1}$$

[1] This isn't proven formally at the moment, but is part of the research plan

From this definition, every region's timestep can be further defined relative to t_{max} and the state variables drawn in figure 3a. For example, variable x_{-1} is the difference in the number of timesteps completed by the interior region and the number of timesteps completed at the leftmost edge of region -10. x_{-1} is also equal to the number of timesteps since the last time a ghost cell was received along that border. To further exemplify, δ_{-1-1} can be thought of as the number of times region $^{-}10$ or region $0^{-}1$ have been updated without a corresponding update to region $^{-}1^{-}1$, making δ_{-1-1} negative as illustrated. On the other hand, figure 3a illustrates a positive δ_{1-1} term. This can occur when 00 has updated several times without any updates from regions $0^{-}1$, 10, or $1^{-}1$. Because δ can be both positive or negative it's intuition is not as straightforward, but it's function as a bookkeeping term is just as important, and is summarized analytically along with the rest of the illustrated variables of state as

$$t(a_x a_y, i_x, i_y) = t_{max} + a_x \cdot x_{a_x} + a_y \cdot y_{a_y} - a_x \cdot i_x - a_y \cdot i_y + |a_x| \cdot |a_y| \cdot \delta_{a_x a_y}, \tag{2}$$

where $t(a_x a_y, i_x, i_y)$ provides the timestep for region $a_x a_y$ in grid cell (i_x, i_y).

The only variables equation 2 doesn't capture from figure 3a are the static boudary terms, denoted by the B variables. Each B^{-1} term denotes a lower bound, and B^{1} term denotes an upper bound for the grid cells along dimensions x and y which are correspondingly subscripted B_x and B_y. These terms will represent array indexing for a process, so $B^{-}1 = 0$ always, and $B^{1} > 0$ always.

To manipulate the state introduced thus far, we'll introduce a more compact notation for a few intersection points. For example, where line $y = x_{-1}$ intersects with line $x = y_{-1}$ is the point $[x_{-1}, y_{-1}]$. In general we'll write,

$$p_{qp} \equiv [x_q, x_p]. \tag{3}$$

For B terms,

$$b_{qp} \equiv [B_x^q, B_y^p]. \tag{4}$$

Finally, for vectors of the subscripts themselves,

$$u_{qp} \equiv [q, p]. \tag{5}$$

ALGORITHM 1: Procedure to update state values for each region $a_x a_y$

```
1  procedure update_state($a_x a_y$) is
2      when $a_x a_y = 00$
3          $x_{-1} = x_{-1} + 1$;   $y_{-1} = y_{-1} + 1$;
4          $x_1 = x_1 - 1$; $y_1 = y_1 - 1$;
5          $\delta_{-1-1} = \delta_{-1-1} + 1$; $\delta_{1-1} = \delta_{1-1} + 1$; $\delta_{11} = \delta_{11} + 1$; $\delta_{-11} = \delta_{-11} + 1$;
6      when $a_x a_y =^{-} 1^{-}1$
7          $\delta_{-1-1} = \delta_{-1-1} + 1$;
8      when $a_x a_y = 1^{-}1$
9          $\delta_{1-1} = \delta_{1-1} + 1$;
10     when $a_x a_y = 11$
11         $\delta_{11} = \delta_{11} + 1$;
12     when $a_x a_y =^{-} 11$
13         $\delta_{-11} = \delta_{-11} + 1$;
14     when $a_x a_y =^{-} 10$
15         $\delta_{-1-1} = \delta_{-1-1} - 1$; $\delta_{-11} = \delta_{-11} - 1$;
16         $x_{-1} = x_{-1} - 1$;
17     when $a_x a_y = 10$
18         $\delta_{11} = \delta_{11} - 1$; $\delta_{1-1} = \delta_{1-1} - 1$;
19         $x_1 = x_1 + 1$;
20     when $a_x a_y = 0^{-}1$
21         $\delta_{-1-1} = \delta_{-1-1} - 1$; $\delta_{1-1} = \delta_{1-1} - 1$;
22         $y_{-1} = y_{-1} - 1$;
23     when $a_x a_y = 01$
24         $\delta_{-11} = \delta_{-11} - 1$; $\delta_{11} = \delta_{11} - 1$;
25         $y_1 = y_1 + 1$;
```

We'll operationalize these terms later in matrix form, and state them now [2]. For boundary intersections,

$$\mathbf{B} \equiv \begin{bmatrix} [B_x^{-1}, B_y^{-1}] & [B_x^{1}, B_y^{-1}] \\ [B_x^{-1}, B_y^{1}] & [B_x^{1}, B_y^{1}] \end{bmatrix} \equiv \begin{bmatrix} b_{-1-1} & b_{1-1} \\ b_{-11} & b_{11} \end{bmatrix} \tag{6}$$

[2] Defining equations 3– 5 allows for the easy manipulation of these terms in matrices, because we're nesting the vector terms so as not to dilute the matrix properties we wish to manipulate by adding a 3rd dimension to each set of state variables. That having been said, a better generalization with more ready extension to the 3D version of this noise tolerance algorithm may be better attained with a tensor representation which is still under study, and would likely change the matrix representations described in this section.

For interior edge intersections,

$$\mathbf{P} \equiv \begin{bmatrix} [x_{-1}, y_{-1}] & [x_1, y_{-1}] \\ [x_{-1}, y_1] & [x_1, y_1] \end{bmatrix} \equiv \begin{bmatrix} p_{-1-1} & p_{1-1} \\ p_{-11} & p_{11} \end{bmatrix}. \tag{7}$$

For a subset of labeling terms which will be explored more in the next section,

$$\mathbf{U} \equiv \begin{bmatrix} [-1, -1] & [1, -1] \\ [-1, 1] & [1, 1] \end{bmatrix} \equiv \begin{bmatrix} u_{-1-1} & u_{1-1} \\ u_{-11} & u_{11} \end{bmatrix}. \tag{8}$$

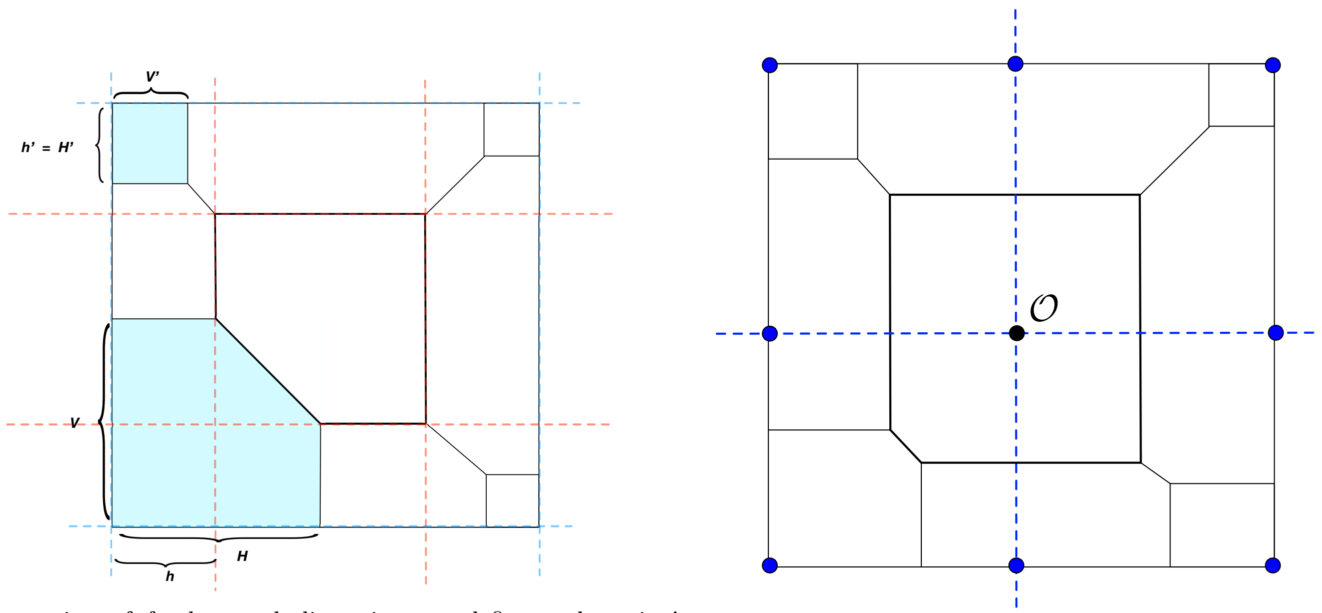
Finally for delta terms, we'll write

$$\mathbf{\Delta} \equiv \begin{bmatrix} \delta_{-1-1} & \delta_{1-1} \\ \delta_{-11} & \delta_{11} \end{bmatrix}. \tag{9}$$

While we've defined the state we must track for each region, we have not yet shown how the state evolves to account for delays in synchronization. The details will be explored more below, but for now consider that algorithm 1 will be run for each region, whenever every cell in that region has received an update. Line 2 handles the interior region, which contracts its boundaries while simultaneously expanding the boundaries of each neigbor region. Lines 6– 12 performs the same operation for each corner region, and lines 14– 23 does the same for the remainder of each region. Note further that if every region update occurs within a timestep, the region boundaries will all return to where they have started with effectively no change in boundaries: For every decrement or increment of a variable of state, there is a corresponding increment or decrement in the opposite direction.

With this summary of the state we must track, we have the tools to update each grid cell while respecting their dependency structures. In the next section, a strategy for updating any each region described by this state will be presented.

3 Uniform Region Updates



(a) Illustration of fundamental dimensions to define each region's boundary: h, H, V and h', H', V'. We'll show in this section how to derive both bounds from the same function

(b) A subdivision of all regions by drawing a horizontal and vertical line through the origin $\mathcal{O}$ - note how each region now looks very similar to the regions annotated in 4a

Figure 4: An illustration of the minimal set of dimensions needed for each region to be described by the same bounds.

ALGORITHM 2: Procedure to update each region $a_x a_y$ according to the number of unit tiles present

```
for *tile* ∈ {*The tiles found in region* $a_x a_y$} do
    for $i_x \leftarrow 1$ to $H(tile)$ do
        for $i_y \leftarrow 1$ to $\min(V(tile), V(tile) - i_x + h(tile))$ do
            *Update the value at grid cell* $(i_x, i_y)$ *1 timestep*;
```

While the 9 regions above do not share the same dependency structure with respect to neighbor process cells, they do share the same geometric forms which are captured by the corner regions: The pentagon of region $-1-1$ and the squares of regions $1^{-}1$, 11, and $^{-}11$ in figure 4a for example. The figure further illustrates dimensions H, h and V for regions $-1-1$ and $^{-}11$ which capture both the pentagon and square shapes. Therefore, to update any region's grid cells, we can consider an approach outlined in the pseudocode of algorithm 2. In line 1 of the pseudo code, we assume that the unit tiles illustrated in figure 4a can be derived as a function of the region $a_x a_y$, and then in lines 2–3 each dimension $(H, h,$ and $V)$ can be derived as a function of the region tiles. This is a very rough outline of what we develop formally below, but it serves as a good guide for our intuition.

To further understand how each region provides the information needed to determine the set of uniform tiles, consider figure 4b. This figure rescales figure 4a slightly and draws a subdivion of each region by intersecting perpendicular lines through an origin term (black center dot) given by,

$$\mathcal{O} \equiv \left[x_{-1} + \frac{x_1 - x_{-1}}{2}, \quad y_{-1} + \frac{y_1 - y_{-1}}{2}\right]. \tag{10}$$

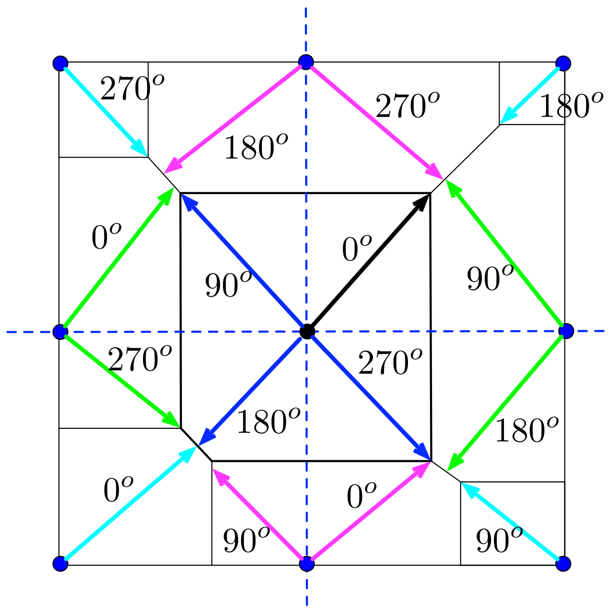
The intersecting lines through this point, cut accross the center of 00 horizontally and vertically, and cut acrross the other side regions (01,0−1,10,−10) either horizontally or vertically. The highlighted blue dots then represent the origins of each unit geometry which can be specified by H, h, and V as illustrated in figure 4a.

With this in mind, as long as we can concisely shift our origin and bounds for each region, we can use the uniform iterates provided by algorithm 2 for every region and region subdivision. We can do this via simple coordinate transformations, and rotations. Transformations are defined by

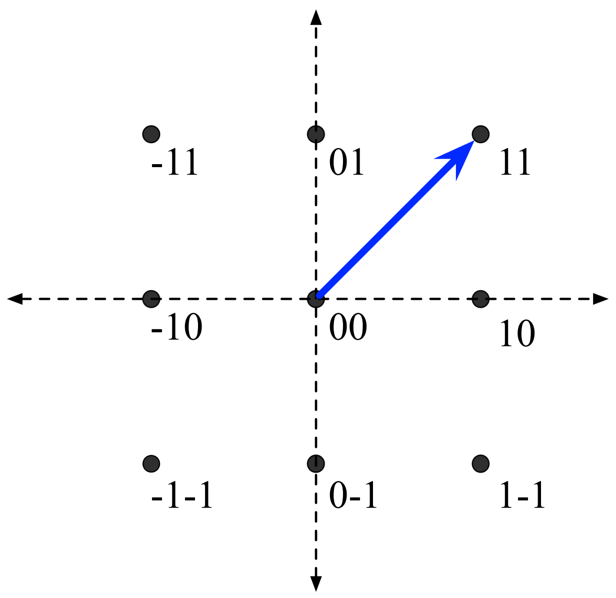
$$T_x(a_x a_y) \quad = B_x^1 \cdot \mathbf{max}(a_x, 0) + (1 - |a_x|) \cdot \mathcal{O}_x \tag{11}$$
$$T_y(a_x a_y) \quad = B_y^1 \cdot \mathbf{max}(a_y, 0) + (1 - |a_y|) \cdot \mathcal{O}_y, \tag{12}$$

where $\mathcal{O}_x$ is the first term of definition 10, and $\mathcal{O}_y$ the second term. To rotate our orientation in each region so that each region may share the same origin, we can employ the standard rotation matrix in 2 dimensions,

$$R_\theta \equiv \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}. \tag{13}$$



(a) Vectors and angles corrsponding to each region, with color coded sets of $0^o - 270^o$. The angle is calculated relative to the unit vector of figure 5b



(b) Unit vector overalayed on coordinates corresponding to region labels

Figure 5: Region angles, and the unit vector

To employ rotations properly, we'll need to know what rotations must be performed in each region, for each subdivision. Figure 5a illustrates the number of rotations present in each region, along with their rotation angles and the vectors they're derived from. In practice, we'll want a procedure to derive these angles using only the region vector, $a_x a_y$. To do so, it's necessary to understand how all of the vectors drawn in figure 5a acquire their angle and magnitude. Figure 5b explains this in part by illustrating the labels of each region overlaid on a cartesian plain, where the labels assemble to form a unit square centered on $[0, \ 0]$. Further, the unit vector whose tail to head goes from $[0, \ 0]$ to $[1, \ 1]$ provides an orientation from which to measure the angles in figure 5a. The vector of figure 5b will be called the *unit vector*. Our procedure for uncovering angles for each region can now proceed as follows:

1. Find the vectors which map to our unit tiles
2. Find the angle between each vector and the unit vector
3. return the set of angles.

Given that we know the angles we're looking for–enumerated in figure 5a– it is straightforward to verify the derived procedure.

To find the angles enumerated in figure 5a, we formalize some notions of our unit labelings of each region:

$$\mathcal{R} \equiv \{-1-1, 0-1, 1-1, -10, 00, 10, -11, 01, 11\} \tag{14}$$
$$\mathcal{S}_u \equiv \{-1-1, 1-1, -11, 11\} \tag{15}$$
$$\mathbf{v}^u \equiv [1, \;\; 1]. \tag{16}$$

$\mathcal{R}$ is just the set of all possible regions given our breakdown where $a_x a_y \in \mathcal{R}$ always. Note further that our notation for $a_x a_y$, can also be thought of as indicating a vector $[a_x, \;\; a_y]$, and we will use the notations interchangably. $\mathcal{S}_u$ is a subset of $\mathcal{R}$, and provides a way to derive the angles of figure 5a along with the unit vector, $\mathbf{v}^u$.

Algorithm 3 formalizes the procedure outlined in steps 1-3 above, and proceeds as follows. Line 4 adds every region vector $a_x a_y$ to each element of $\mathcal{S}_u$, before finding the intersection of these vectors with the set of valid labels in line 5. With these valid label's in hand, line 8 calculates the corresponding vectors of each region $a_x a_y$– in fact the $\mathbf{u}$ calculated here are the vectors displayed and color coded for each region in figure 5a. Finally, the angle between the vector $\mathbf{u}$ and the unit vector $\mathbf{v}^u$ are calculated in line 8. The trigonometric function $atan2$ is used in order to return an angle between 0 and 360 to conform to our target rotations (figure 5a). The inputs to this function are the dot product of $\mathbf{u}$ and $\mathbf{v}^u$ followed by the determinant of the same vectors concatenated together–concatenation denoted by $||$, and the determinant denoted with vertical bars on each side of the expression, $|\cdot|$.

With our state composition defined in section 2, and our ability to isolate uniform tiles by angle with procedure 3, we're now able to derive an expression for H, h, and V illustrated in figure 4a, and therefore implement our update pseudocode outlined by procedure 2. Specifically, each tile's dimensions (H, h, and V) can be expressed as functions where $h, H, V : (a_x a_y, \theta) \to \mathbb{R}^+$. To develop these functions, we'll need a way to index state variables presented in section 2. The first observation that can aid intuition of the indexing scheme derived is to observe the core variables of state 7 and 9,

$$\mathbf{P} \equiv \begin{bmatrix} [x_{-1}, y_{-1}] & [x_1, y_{-1}] \\ [x_{-1}, y_1] & [x_1, y_1] \end{bmatrix},$$

and

$$\boldsymbol{\Delta} \equiv \begin{bmatrix} \delta_{-1-1} & \delta_{1-1} \\ \delta_{-11} & \delta_{11} \end{bmatrix}.$$

Note that for region 00, H, V, and h can all be written in terms of each term in these matrices for each angle θ by moving counter-clockwise through the matrix 7 starting in the bottom right corner of each matrix (i.e. $\mathbf{P}_{11}$ and $\boldsymbol{\Delta}_{11}$). For example at $\theta = 0^o$ for region 00, H, V, and h will all be bounded by $[x_1, \;\; y_1]$ and δ_{11} (i.e. the terms of $\mathbf{P}_{11}$ and $\boldsymbol{\Delta}_{11}$). This observation on the pattern of matrix indexing according to angle rotations can be generalized for rows (denoted r) and columns (denoted c) of matrices as

$$\mathsf{c} = \mathbf{min}(1 + [R_{-\theta} \cdot \mathbf{v}^u]_y, 1) \tag{17}$$
$$\mathsf{r} = \mathbf{min}(1 + [R_{-\theta} \cdot \mathbf{v}^u]_x, 1). \tag{18}$$

To simplify notation however, for any 2x2 matrix M and angle $\theta \in \{0, 90, 180, 360\}$, we write

$$M[\theta] \equiv M_{\mathsf{rc}}, \tag{19}$$

Which allows us to directly index variables of state by angle θ. To do this for any region $a_x a_y$ however, the expressions of state in their current matrix form would not be sufficient to employ this kind of angle based indexing universally without first transforming the matrices of state given by matrices (6)–(9). This transformation is given by the following function, where for state matrix M, region transformation $M_{a_x a_y}$, is defined as:

$$M_{a_x a_y} \equiv \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}^{2-|a_y|} \cdot M \cdot \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}^{2-|a_x|}, \tag{20}$$

ALGORITHM 3: Procedure, *AngleSearch* which takes as input $a_x a_y \in \mathcal{R}$, and returns the set Θ of all the angles associated with it.

```
1  procedure $AngleSearch(a_x a_y \in \mathcal{R})$ is
2      $\mathcal{S}' = \{\}$;
3      for $s \in \mathcal{S}_u$ do
4          $\mathcal{S}' = \mathcal{S}' \cup \{\mathbf{s} + [a_x, \;\; a_y]\}$
5      $\mathcal{S}' = \mathcal{S}' \cap \mathcal{R}$;
6      $\Theta = \{\}$;
7      for $s \in \mathcal{S}'$ do
8          $\mathbf{u} = \mathbf{s} - [a_x, \;\; a_y]$;
9          $\Theta = \Theta \cup atan2(\mathbf{u} \cdot \mathbf{v}^u, |\mathbf{u}^T||\mathbf{v}^{uT}|)$;
10     return $\Theta$;
```

where the pre-multiplication of matrix $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ flips the rows of matrix M, and the post-multiplication of matrix M by $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$ flips its columns. Given our relatively small state space, this can be verified as yielding appropriate matrices of state for each region through an exhaustion of cases.

Taken together, equations 19 and 20, allow us to retrieve the variables of state given θ and $a_x a_y$,

$$\begin{aligned} \mathbf{p} &= \mathbf{P}_{a_x a_y}[\theta] && (21)\\ \mathbf{b} &= \mathbf{B}_{a_x a_y}[\theta] && (22)\\ \delta &= \mathbf{\Delta}_{a_x a_y}[\theta] && (23)\\ \mathbf{u} &= \mathbf{U}_{a_x a_y}[\theta]. && (24) \end{aligned}$$

From equations 21– 24, we can write $h, H, V : (a_x a_y, \theta) \to \mathbb{R}^+$,

$$h(\theta, \mathbf{a}) = [R_{-\theta} \cdot \mathbf{p}^T]_x + [R_{-\theta} \cdot \mathbf{a}^T]_x \cdot [R_{-\theta} \cdot \mathbf{b}^T]_x - [R_{-\theta} \cdot \mathcal{O}]_x \cdot (1 - |R_{-\theta} \cdot \mathbf{a}^T|_x) + (1 - |R_{-\theta} \cdot \mathbf{a}^T|_y) \cdot \mathbf{min}_{x,y}(R_{-\theta} \cdot \mathbf{u}) \cdot \delta \quad (25)$$

$$H(\theta, \mathbf{a}) = [R_{-\theta} \cdot \mathbf{p}^T]_x + [R_{-\theta} \cdot \mathbf{a}^T]_x \cdot [R_{-\theta} \cdot \mathbf{b}^T]_x - [R_{-\theta} \cdot \mathcal{O}]_x \cdot (1 - |R_{-\theta} \cdot \mathbf{a}^T|_x) + |R_{-\theta} \cdot \mathbf{a}^T|_y \cdot [R_{-\theta} \cdot \mathbf{u}]_x \cdot \delta \quad (26)$$

$$V(\theta, \mathbf{a}) = [R_{-\theta} \cdot \mathbf{p}^T]_y + [R_{-\theta} \cdot \mathbf{a}^T]_y \cdot [R_{-\theta} \cdot \mathbf{b}^T]_y - [R_{-\theta} \cdot \mathcal{O}]_y \cdot (1 - |R_{-\theta} \cdot \mathbf{a}^T|_y) + |R_{-\theta} \cdot \mathbf{a}^T|_x \cdot [R_{-\theta} \cdot \mathbf{u}]_y \cdot \delta \quad (27)$$

Making these functions discrete (i.e. $h, H, V : (a_x a_y, \theta) \to \mathbb{Z}^+$) requires reworking equation 10 with the ceiling function,

$$\mathcal{O} \equiv \left[x_{-1} + \left\lceil \frac{x_1 - x_{-1}}{2} \right\rceil, \quad y_{-1} + \left\lceil \frac{y_1 - y_{-1}}{2} \right\rceil\right], \quad (28)$$

and using using it in equations (25)–(27). Algorithm 4 puts these equations together and gives precise analytic form to the pseudocode developed in algorithm 2.

ALGORITHM 4: Uniform Region Update $\forall$ input $a_x a_y = \mathbf{a} \in \mathcal{R}$

```
1 $\Theta = AngleSearch(a_x a_y)$;
2 for $\theta \in \Theta$ do
3     for $j = 0$ to $j = H(\theta, \mathbf{a})$ do
4         for $k = 0$ to $k = \mathbf{min}\{V(\theta, \mathbf{a}), V(\theta, \mathbf{a}) - j + h(\theta, \mathbf{a})$ do
5             $\mathbf{i} = R_\theta \cdot [j, \ k]^T$;
6             $g(t, T_x(\mathbf{a}) + \mathbf{i}_x T_y(\mathbf{a}) + \mathbf{i}_y)) = f(t-1, T_x(\mathbf{a}) + \mathbf{i}_x, T_y(\mathbf{a}) + \mathbf{i}_y)$
```

Algorithm 4 puts everything together ignoring some implemtation details, with equations for H, h, and V. In the interest of concision, line 6 omits a precise definition of a 5 point stencil, but simply states that $g(t, x, y) = f(t, x, y)$ meaning at time t, we can compute the value of a stencil at $g(t, x, y)$ given some function f of the same point x, y at the prior time $t - 1$. Line 1 executes the procedure of algorithm 3, and illustrates the parallel between our notion of a tile and it's apparent encoding as an angle. The potential for an intersection in the mathematical research on *tiling* is intriguing, but has yet to bear analytic fruit beyond the broad observations made in this section.

4 Update Constraints for the Multi-Process Problem

Up until now we've been considering updates on a single process using the regions decomposed by the state laid out in section 2, but we still need to consider coordination between processes. Recall in our discussion of algorithm 1 that if the procedure is called once for every region (i.e. $\forall a_x a_y \in \mathcal{R}$), then the borders introduced in the static region decomposition in section 1 remain unchanged. However when this requirement is relaxed, then a few things immediately follow. In particular, certain conditions must be met for algorithm 4 to be invoked by a process at all. This section enumerates those conditions.

4.1 Nonzero Region Area

This first such condition is that for every region $a_x a_y \in \mathcal{R}$, and for all $\theta \in AngleSearch(a_x a_y)$, it must be the case that $h(\theta, a_x a_y) > 0$, $H(\theta, a_x a_y) > 0$, and $V(\theta, a_x a_y) > 0$. This can be summarized by the boolean function,

$$\mathcal{G}^S(a_x a_y, \Theta) \equiv \bigwedge_{\theta \in \Theta} H(\theta, \mathbf{a}) > 2 \wedge h(\theta, \mathbf{a}) > 2 \wedge V(\theta, \mathbf{a}) > 2, \quad (29)$$

where $\Theta = AngleSearch(a_x a_y)$. Here, 0 is replaced by 2 given that in discrete space, a 5 point stencil must have at minimum a $3x3$ grid of cells to update a single point (or at least 5 cells, with its 4 corners missing). We'll refer to formula 29 as the *spatial update guard*.

4.2 Interprocess Communication Constraints

Any process wishing to update its border cells must receive ghost cells from one or more neighbor processes. Therefore, there must be a means of ensuring that the requisite *communication* (sending and receiving of data) is performed and completed. Given our assumption that communication completion is nondeterministic, the abstraction we will use will mimic MPI's non-blocking communication. Let every process have a unique id $\mathsf{pid} \in \mathbb{Z}^+$. Let the $\mathcal{N}(a_x a_y, \mathsf{pid}) : (a_x a_y, \mathsf{pid}) \rightarrow \{\mathbb{Z}^+\}$ be a mapping from a region $a_x a_y$ and a process id pid to a set of positive integers which themselves are process ids of the processes that pid must communicate with (these ids are referred to as *neighbors*).

Next, we define functions analagous to MPI's ISend and IRecv respectively. $\mathsf{send}_{a_x a_y}[n] \leftarrow \mathit{postSend}(n \in \mathbb{Z}^+, \mathsf{d})$ is the signature for a function which takes a neighbor process id as its first argument, and some arbitrary data structure d as its second argument, and returns a boolean value in the array $\mathsf{send}_{a_x a_y}[0..|\mathcal{N}(a_x a_y, \mathsf{pid})|]$, which is initially set to false, and then nondeterministically set to true when the neighbor process n receives the value d, and copies it to a local data structure. Likewise, $\mathsf{rcv}_{a_x a_y}[n] \leftarrow \mathit{postRecv}(n \in \mathbb{Z}^+, \mathsf{d})$ is the signature for a function which takes a neighbor process id as its first argument, and some arbitrary local data structure d as its second argument, and returns a boolean value in the array $\mathsf{rcv}_{a_x a_y}[0..|\mathcal{N}(a_x a_y, \mathsf{pid})|]$, which is initially set to false, and then nondeterministically set to true when the neighbor process pid receives a new value in its local data structure d from process n, and can perform a validly access the memory for d.

Given these definitions for communication between neighbor processes, it follows for all $a_x a_y \in \mathcal{R} - \{00\}$ that the following boolean function must be true for any valid update to be performed:

$$\mathcal{G}^C(a_x a_y, \mathsf{pid}) \equiv \bigwedge_{n \in \mathcal{N}(a_x a_y, \mathsf{pid})} \mathsf{send}_{a_x a_y}[n] \wedge \mathsf{rcv}_{a_x a_y}[n]. \tag{30}$$

We'll call this the *communication update guard.*

4.3 Mirror Process Misalignment

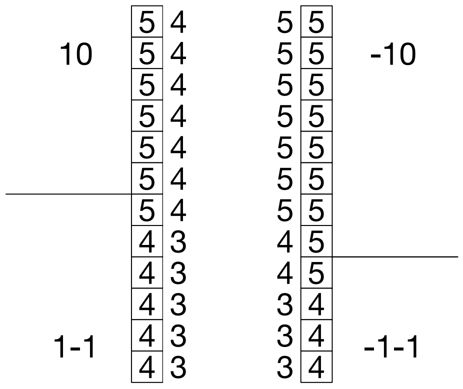


Figure 6: A motivating example for the utility of $\mathcal{G}_{\mathcal{M}}$. A subset of the 2D data arrays of a left and right process are illustrated–ghost cells drawn without a border, and border cells given a border. A delimeter between regions on each process perpendicular To B_x^1 and B_x^{-1} is drawn for the left and right process, respectively. Region labels are shown for the 4 regions of interest to the example, and the rest are omitted

Given a global spatial decomposition as described in section 1, 2 neighbor processes are referred to as *mirror processes* of each other. In figure 1, process 2 and 4 are process 1's mirror processes, process 1 and 3 are process 2's mirror processes, and so on. Further, the state variables outlined in section 2 are referred to as the *mirror state* of the processes adjacent to them.

Figure 6 illustrates an example how the *mirror state* of 2 processes can become misaligned. This discussion will not show how such a state is reachable, but what must be done when it occurs in order to respect the dependency contraints of a cell update at the border. Drawn is a left process with regions -10 and 10 shown, and its *mirror process* on the right with regions -10 and $-1-1$. Border cells of the processes are bordered by a square boxes, and a timesteps are displayed within them. Ghost cell timesteps are displayed with no border. Observe that the border cells of the left process are exactly reflected in the ghost cells of the right process

ALGORITHM 5: Procedure to check mirror state for each $a_x a_y \in \mathcal{R}$

```
1 procedure $\mathcal{G}^M(a_x a_y \in \mathcal{R} - \{00\}, \Theta)$ is
2     $\theta_1 = \Theta[0]$;   $\theta_2 = \Theta[1]$ ? $|\Theta| = 2 : \theta_1$;
3     return $H[\theta_1] \leq V^M[\theta_1 - 90] \wedge V[\theta_2] \leq V^M[\theta_2 + 90]$;
```

and vice versa. This in fact corresponds to the state in each process, $\mathbf{P}$ and $\mathbf{\Delta}$. We assume further that this state is exchanged between processes, and stored as locally in *mirror variables*, where

$$\mathbf{\Delta}_n^M \equiv \mathbf{\Delta} \text{ matrix of neighbor process } n \tag{31}$$

$$\mathbf{P}_n^M \equiv \mathbf{P} \text{ matrix of neighbor process } n. \tag{32}$$

It follows, that any calculation made with these terms will receive a superscript M to denote that it is in fact measuring the term for the mirror process (e.g. $V^M(\theta, \mathbf{a}), h^M(\theta, \mathbf{a})$).

To ensure that a region $a_x a_y$ is not updated when its appropriate ghost cells are not present as in region -10 in figure 6, the boolean function $\mathcal{G}^M$ shown in algorithm 5 can be used. The function returns true if the region is an interior region, and then checks alignment with mirror quantities from its neigbor process(es). $H(\theta, \mathbf{a})$ is mirrored by $V^M(\theta - 90^o, \mathbf{a})$, and $V(\theta, \mathbf{a})$ is mirrored by $V^M(\theta + 90^o, \mathbf{a})$. This is a relationship that holds up for all $a_x a_y \in \mathcal{R} - \{00\}$, and can be checked by drawing out the rotated bounds V and H for all border and corner regions, along side their region bounds.

5 Full Algorithm

ALGORITHM 6: Full algorithm to be run on every process until termination conditions are met

```
/* Initialize Global Variables.                                                      */
1  $t_{max} \leftarrow 1$;
2  $x_{-1} \leftarrow 1$; $x_1 \leftarrow B_x^1$; $y_{-1} \leftarrow 1$; $y_1 \leftarrow B_y^1$;
3  $\delta_{-11} \leftarrow 0$; $\delta_{-1-1} \leftarrow 0$; $\delta_{11} \leftarrow 0$; $\delta_{1-1} \leftarrow 0$;
4  for $a_x a_y \in \mathcal{R} - \{00\}$ do
5      $\Theta = \text{AngleSearch}(a_x a_y)$;
6      for $\theta \in \Theta$ do
7          for $n \in \mathcal{N}(a_x a_y, \mathsf{pid})$ do
8              $\mathsf{send}_{a_x a_y}[n] \leftarrow postSend(n, \mathbf{\Delta})$;   $\mathsf{rcv}_{a_x a_y}[n] \leftarrow postRecv(n, \mathbf{\Delta}_n^M)$;
9              $\mathsf{send}_{a_x a_y}[n] \leftarrow postSend(n, \mathbf{P})$;   $\mathsf{rcv}_{a_x a_y}[n] \leftarrow postRecv(n, \mathbf{P}_n^M)$;
10             $\mathsf{send}_{a_x a_y}[n] \leftarrow postSend(n, \mathsf{data}[1][\mathbb{I}_y(-1, \theta, \mathbf{a})^s..\mathbb{I}_y(+1, \theta, \mathbf{a})^s][\mathbb{I}_x(-1, \theta, \mathbf{a})^s..\mathbb{I}_x(+1, \theta, \mathbf{a})^s])$;
               $\mathsf{rcv}_{a_x a_y}[n] \leftarrow postRecv(n, \mathsf{data}[0][\mathbb{I}_y(-1, \theta, \mathbf{a})..\mathbb{I}_y(+1, \theta, \mathbf{a})][\mathbb{I}_x(-1, \theta, \mathbf{a})..\mathbb{I}_x(+1, \theta, \mathbf{a})])$;
11 while $t_{max} < \mathsf{nsteps} \vee x_{-1} > 0 \vee y_{-1} > 0 \vee x_1 < B_x^1 \vee y_{-1} > B_y^1$ do
12     for $a_x a_y \in \mathcal{R}$ do
13         $\Theta = \text{AngleSearch}(a_x a_y)$;
14         when $\mathcal{G}^S(a_x a_y, \Theta) \wedge ((\mathcal{G}^C(a_x a_y, \mathsf{pid}) \wedge \mathcal{G}^M(a_x a_y, \Theta)) \vee a_x a_y = 00)$
15             $\theta_c = 0$;
16             for $\theta \in \Theta$ do
17                 $\mathbf{p} = \mathbf{P}_{a_x a_y}[\theta]$;
18                 $\mathbf{b} = \mathbf{B}_{a_x a_y}[\theta]$;
19                 $\delta = \mathbf{\Delta}_{a_x a_y}[\theta]$;
20                 $\mathbf{u} = \mathbf{U}_{a_x a_y}[\theta]$;
21                 $h(\theta, \mathbf{a}) = [R_{-\theta} \cdot \mathbf{p}^T]_x + [R_{-\theta} \cdot \mathbf{a}^T]_x \cdot [R_{-\theta} \cdot \mathbf{b}^T]_x - [R_{-\theta} \cdot \mathcal{O}]_x \cdot (1 - |R_{-\theta} \cdot \mathbf{a}^T|_x) + (1 - |R_{-\theta} \cdot \mathbf{a}^T|_y) \cdot \mathbf{min}_{x,y}(R_{-\theta} \cdot \mathbf{u}) \cdot \delta$;
22                 $H(\theta, \mathbf{a}) = [R_{-\theta} \cdot \mathbf{p}^T]_x + [R_{-\theta} \cdot \mathbf{a}^T]_x \cdot [R_{-\theta} \cdot \mathbf{b}^T]_x - [R_{-\theta} \cdot \mathcal{O}]_x \cdot (1 - |R_{-\theta} \cdot \mathbf{a}^T|_x) + |R_{-\theta} \cdot \mathbf{a}^T|_y \cdot [R_{-\theta} \cdot \mathbf{u}]_x \cdot \delta$;
23                 $V(\theta, \mathbf{a}) = [R_{-\theta} \cdot \mathbf{p}^T]_y + [R_{-\theta} \cdot \mathbf{a}^T]_y \cdot [R_{-\theta} \cdot \mathbf{b}^T]_y - [R_{-\theta} \cdot \mathcal{O}]_y \cdot (1 - |R_{-\theta} \cdot \mathbf{a}^T|_y) + |R_{-\theta} \cdot \mathbf{a}^T|_x \cdot [R_{-\theta} \cdot \mathbf{u}]_y \cdot \delta$;
24                 $\mathbf{q} = R_\theta \cdot [1, \;\; 1]^T$;
25                 for $j = 0$ to $j = H(\theta, \mathbf{a}) - \mathbf{q}_x$ do
26                     for $k = 0$ to $k = \mathbf{min}\{V(\theta, \mathbf{a}) - \mathbf{q}_y, V(\theta, \mathbf{a}) - j + h(\theta, \mathbf{a}) - \mathbf{q}_y\}$ do
27                         $\mathbf{i} = R_\theta \cdot [j, \;\; k]^T$;
28                         $\mathsf{data}[1 - \overline{t(a_x a_y, j, k)}][T_y(\mathbf{a}) + \mathbf{i}_y) + \mathbf{q}_y \cdot (1 - |a_y|) \cdot \mathbf{min}((\theta_c \ \mathbf{mod}\ 3), 1)][T_x(\mathbf{a}) + \mathbf{i}_x + \mathbf{q}_x \cdot (1 - |a_x|) \cdot \overline{\theta_c}] =$
                             $f'(\mathsf{data}[\overline{t(a_x a_y, j, k)}][T_y(\mathbf{a}) + \mathbf{i}_y + \mathbf{q}_y \cdot (1 - |a_y|) \cdot \mathbf{min}((\theta_c \ \mathbf{mod}\ 3), 1)][T_x(\mathbf{a}) + \mathbf{i}_x + \mathbf{q}_x \cdot (1 - |a_x|) \cdot \overline{\theta_c}])$
29                 $\theta_c = \theta_c + 1$;
30             when $a_x a_y \neq 00$
31                 for $\theta \in \Theta$ do
32                     for $n \in \mathcal{N}(a_x a_y, \mathsf{pid})$ do
33                         $\mathsf{send}_{a_x a_y}[n] \leftarrow postSend(n, \mathbf{\Delta})$;   $\mathsf{rcv}_{a_x a_y}[n] \leftarrow postRecv(n, \mathbf{\Delta}_n^M)$;
34                         $\mathsf{send}_{a_x a_y}[n] \leftarrow postSend(n, \mathbf{P})$;   $\mathsf{rcv}_{a_x a_y}[n] \leftarrow postRecv(n, \mathbf{P}_n^M)$;
35                         $\mathsf{send}_{a_x a_y}[n] \leftarrow postSend(n, \mathsf{data}[1 - \overline{t(a_x a_y, j, k)}][\mathbb{I}_y(-1, \theta, \mathbf{a})^s..\mathbb{I}_y(+1, \theta, \mathbf{a})^s][\mathbb{I}_x(-1, \theta, \mathbf{a})^s..\mathbb{I}_x(+1, \theta, \mathbf{a})^s])$;
                           $\mathsf{rcv}_{a_x a_y}[n] \leftarrow postRecv(n, \mathsf{data}[\overline{t(a_x a_y, j, k)}][\mathbb{I}_y(-1, \theta, \mathbf{a})..\mathbb{I}_y(+1, \theta, \mathbf{a})][\mathbb{I}_x(-1, \theta, \mathbf{a})..\mathbb{I}_x(+1, \theta, \mathbf{a})])$;
36             when $a_x a_y = 00$
37                 $t_{max} \leftarrow t_{max} + 1$;
38             $update\_state(a_x a_y)$;
```

The full description for a parallel decomposition of a 5 point stencil is provided in algorithm 6. Some details that haven't been explored are provided here for completeness. To start with, how to handle discrete space indexing given rotations within a region $a_x a_y$. The modulo terms below account for this (where $r = x \ \mathbf{mod}\ y$, and r is the remainder from dividing x into y). Given its frequency, the shorthand $\overline{x} = x \ \mathbf{mod}\ 2$ is used as well when $y = 2$. Additionally, a termination condition has not been introduced for the algorithm. Recall that the modeling problem initially set out in section 1 is that of PDEs, which evolves a spatially dependent quantity over time. Therefore the stopping condition will be the number of timesteps to evolve the spatial property over, nsteps.

Finally, we will actually introduce operations on an array of cells being used to model the PDE. $\mathsf{data}[0..1][0..B_y^1+1][0..B_x^1+1]$ will hold all the grid cells regardless of region, including the ghost cells. It is a 3D array with a 'double buffering' strategy for updating in time in the first dimension, and the height and length of the local portion of the process' PDE stored in the rows and columns, respectively. For a set of shorthands for defining the bounds to send and receive in each spatial dimension of data we define

$$\mathbb{I}_x(q,\theta,\mathbf{a}) \quad = T_x + |a_x| \cdot (a_x - q) \cdot R_\theta \cdot [H(\theta,\mathbf{a}), V(\theta,\mathbf{a})]_x^T \tag{33}$$

$$\mathbb{I}_y(q,\theta,\mathbf{a}) \quad = T_y + |a_y| \cdot (a_y - q) \cdot R_\theta \cdot [H(\theta,\mathbf{a}), V(\theta,\mathbf{a})]_y^T, \tag{34}$$

where $q \in \{-1,1\}$ indicating the left bound and right bound respectively. This is a discrete function which works for the receive buffers (ghost cells), but given we're also accessing send buffers (border cells), this won't be sufficient. This combined with the need to account for the case where $\mathbb{I}_x(-1,\theta,\mathbf{a}) = \mathbb{I}_x(+1,\theta,\mathbf{a})$ motivates the additional helper function

$$\mathbb{I}_x(q,\theta,\mathbf{a})^s \quad = \mathbf{min}(\mathbf{max}(\mathbb{I}_x(q,\theta,\mathbf{a}) - q, 1), B_x^1) \tag{35}$$

$$\mathbb{I}_y(q,\theta,\mathbf{a})^s \quad = \mathbf{min}(\mathbf{max}(\mathbb{I}_y(q,\theta,\mathbf{a}) - q, 1), B_y^1). \tag{36}$$

Finally, note that we do not define the entire stencil update method for a 5 point stencil, but continue to denote it by f', which no longer needs a variable of time (as in algorithm 4), as our data buffer has made this access precise. Further, g has been dropped from the original formulation of algorithm 4 as the left hand side of the update need only access the single appropriate center cell of the 5-point dependency structure under study.

6 Acknowledgements

This research used resources of the Argonne Leadership Computing Facility at Argonne National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357.

This research used the University of Delaware's *Chimera* computer, funded by U.S. National Science Foundation award CNS-0958512. S.F. Siegel was supported by NSF award CCF-0953210.

7 License