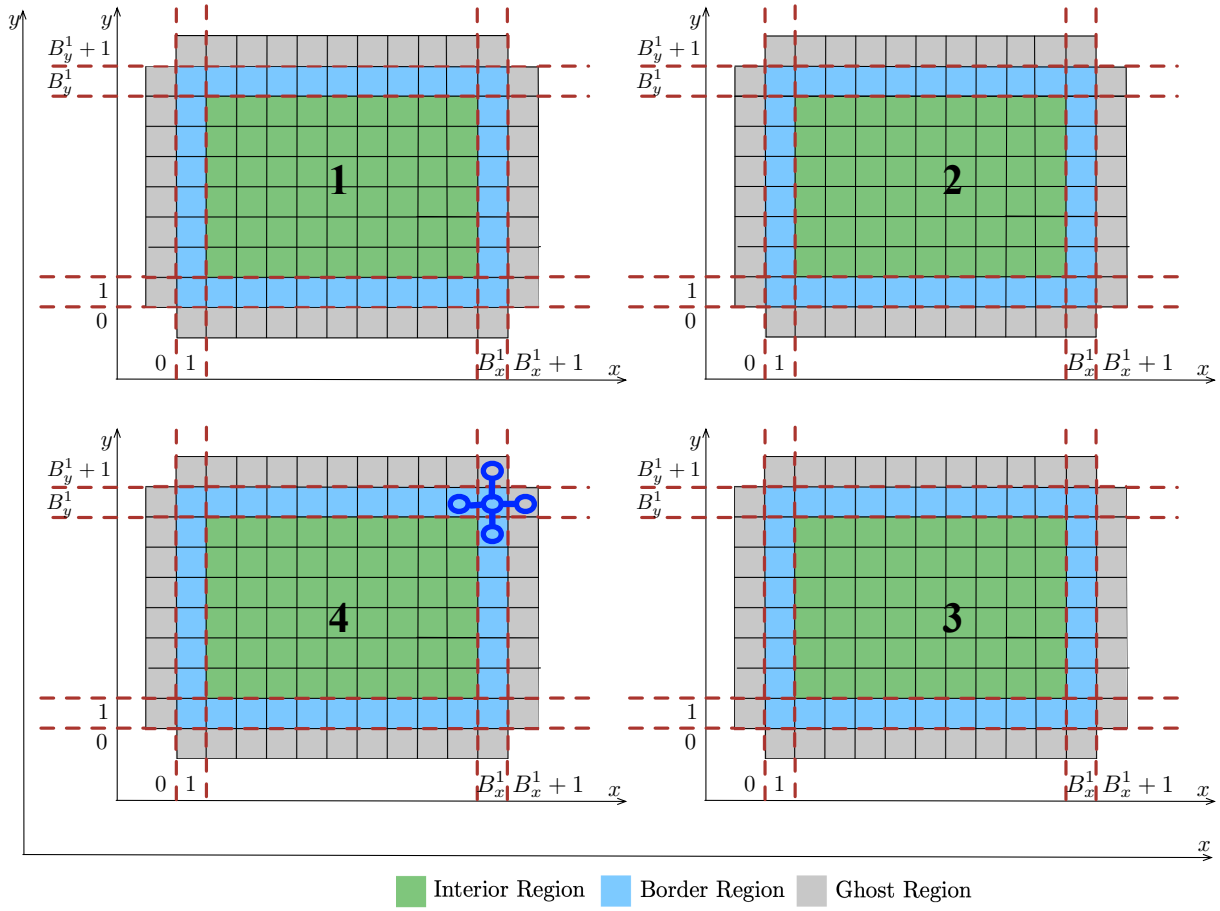


Figure 1: 4 process decomposition of a 22x16 stencil. Process labels start in the top left and proceed clockwise. Local variables of state are also illustrated for indexing a 2D array, 0,1,nxl,etc.

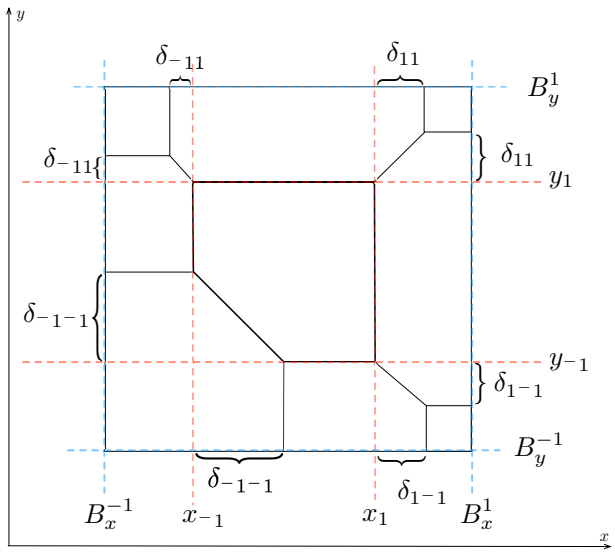
1 Motivation And Basic Terminology

The solution of partial differential equations (PDEs) are classic problems for parallel computers and use what are known as stencil methods. A *stencil* is a dependency structure for a grid cell that is defined according to the discretization of a partial differential equation. The top right corner of process 4 in figure 1 illustrates an example. At grid point (nxl, nyl), there is a single blue dot surrounded by 4 dots. This example illustrates a *5-point stencil*, for the 5 blue dots in the image. The center dot is a spatially dependent value of the PDE that will evolve over time. The adjacent dots are the spatial dependencies for the center cell. All cells can update their value one unit of time when these dependencies are present. On a single process, this dependency structure repeats for every cell in the grid illustrated. When every cell in the grid completes its updates, the process is said to complete a *timestep*.

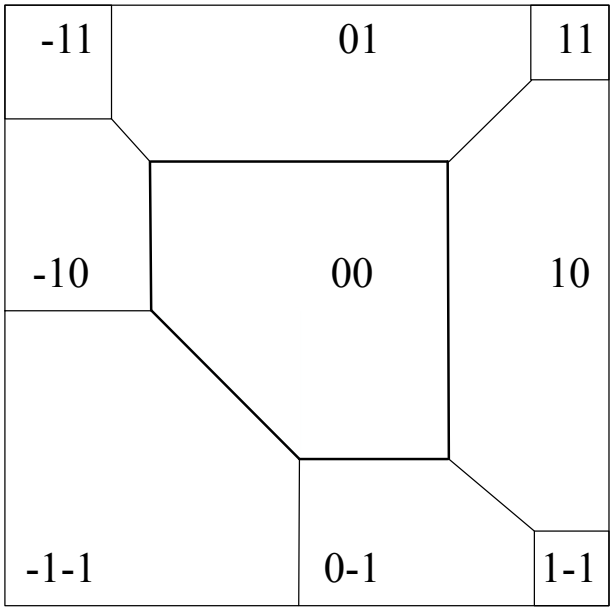
Parallelizing a large 2D stencil using a SIMD strategy starts with the definition of a global area over which computation is required, followed by a subdivision of that area into 2D grids for each processing unit to manage. Figure 1 illustrates the general principle for 4 processing units, each with 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 illustrated 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' 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, and the requirements for updating each regions' grid cells in each timestep, updating grid cells in parallel requires each process to synchronize with neighbor processes. If synchronizations are expensive, it becomes beneficial to develop strategies to relax these synchronization requirements. In order to do so while maintaining stencil depencies



(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 2: (2a) An illustration of state labels that allow each of the 9 regions to relax their boundaries. (2b) Shorthand labels for each region, and for corresponding area vectors.

reqires relaxing the boundaries of the 9 classically static regions of a 2D stencil on each process.

2 State Variables For Region Decomposition

Figure 2b illustrates a more concise labeling of each of the 9 regions described above, whose boundaries have been relaxed to arbitrary positions. Figure 2a illustrates the minimal set of variables needed to relax region boundaries for a single process, and delimits the regions they collectively form. Within each region defined by these new state variables, grid cell updates can continue while respecting the dependencies of the stencil. This is because each state variables captures information about the timestep, and therefore dependency information of each region. To start with region, 00 (the 'interior region') 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 regarless 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,

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

From this definition, every region's timestep can be further defined relative to t_{max} and the state variables drawn in figure 2a. 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 2a 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 2a are the static boudary terms, denoted by the B variables. Each term B^{-1} denotes a lower bound, and B^1 denotes an upper bound for the grid cells along dimensions x and y which are subscripted. 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 their intersection points. For example, where line $y = x_{-1}$ intersects with line $x = y_{-1}$ at $[x_{-1}, y_{-1}]$).

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

generalizes the example. For B terms,

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

Finally, for vectors of the indexes,

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

We'll operationalize these terms later in matrix form, and state them now [1]. For boundary intersections,

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

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

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

For delta terms,

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

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          δ_{-1-1} = δ_{-1-1} + 1; δ_{1-1} = δ_{1-1} + 1; δ_{11} = δ_{11} + 1; δ_{-11} = δ_{-11} + 1;
6      when a_x a_y =⁻ 1⁻1
7          δ_{-1-1} = δ_{-1-1} + 1;
8      when a_x a_y = 1⁻1
9          δ_{1-1} = δ_{1-1} + 1;
10     when a_x a_y = 11
11         δ_{11} = δ_{11} + 1;
12     when a_x a_y =⁻ 11
13         δ_{-11} = δ_{-11} + 1;
14     when a_x a_y =⁻ 10
15         δ_{-1-1} = δ_{-1-1} − 1; δ_{-11} = δ_{-11} − 1;
16         x_{-1} = x_{-1} − 1;
17     when a_x a_y = 10
18         δ_{11} = δ_{11} − 1; δ_{1-1} = δ_{1-1} − 1;
19         x_1 = x_1 + 1;
20     when a_x a_y = 0⁻1
21         δ_{-1-1} = δ_{-1-1} − 1; δ_{1-1} = δ_{1-1} − 1;
22         y_{-1} = y_{-1} − 1;
23     when a_x a_y = 01
24         δ_{-11} = δ_{-11} − 1; δ_{11} = δ_{11} − 1;
25         y_1 = y_1 + 1;
```

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 respectively in the opposite direction.

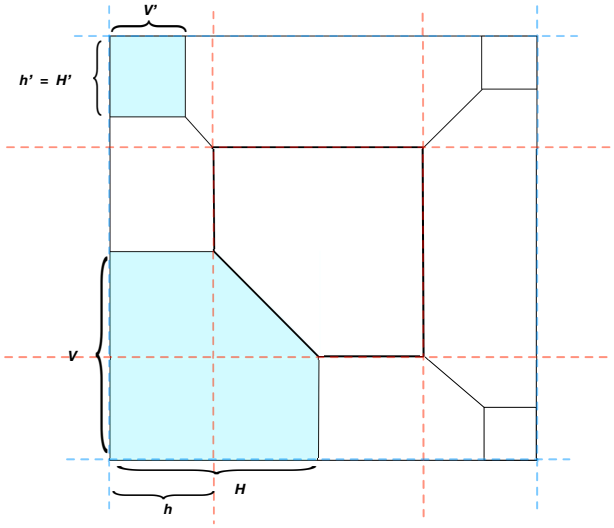
With this summary of state we must track, we have the tools to update each grid cell while respecting their dependency structure. In the next section, a strategy for updating any each region regardless of it's geometry will be presented.

[1] Defining equations 3– 5 allows for the easy manipulation of these terms in matrices, but 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.

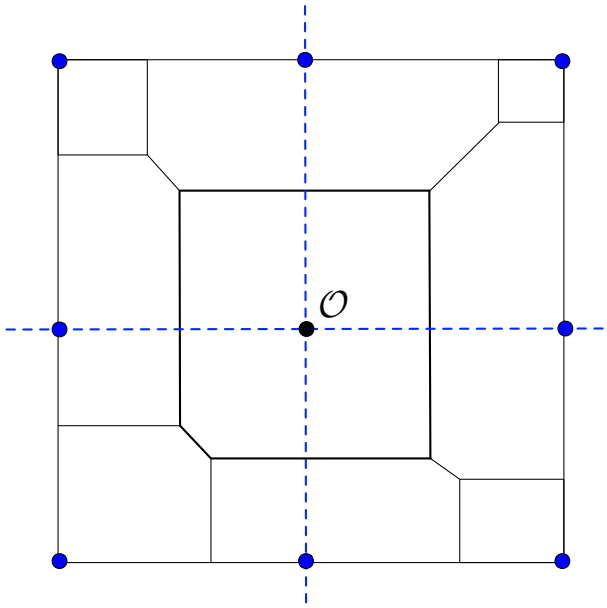
3 Uniform Region Updates

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 $^{-}11$. Figure 3a illustrates dimensions H, h and V for regions $-1-1$ and $^{-}11$ which capture both the pentagon and square shapes.

To update any region's gridcells therefore, we must simply run pseudocode given in algorithm 2.



(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 3a

Figure 3: An illustration of the minimal set of dimensions needed for each region to be described by the same bounds.

Figure 3b rescales figure 3a slightly and draws the subdivion of each region by intersecting an origin term (black center dot),

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

This term cuts accross the center of 00 horizontally and vertically, and cuts 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 3a. Where the dotted blue line divides regions, these terms and associated geometries will have multiple instantiations - 1 for each division, while sharing the origin.

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

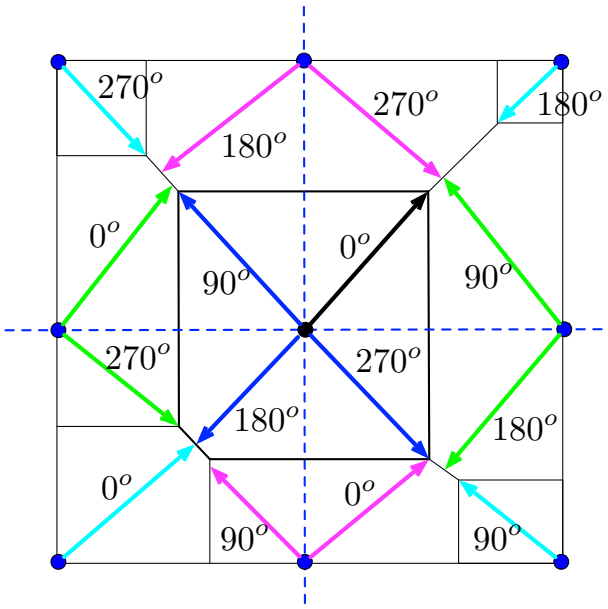
```
for tile ∈ tiles in region a_x a_y do
    for x ← 1 to H(tile) do
        for y ← 1 to min(V(tile), V(tile) − x + h(tile)) do
            update(x, y).
```

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 simply given the functions 12

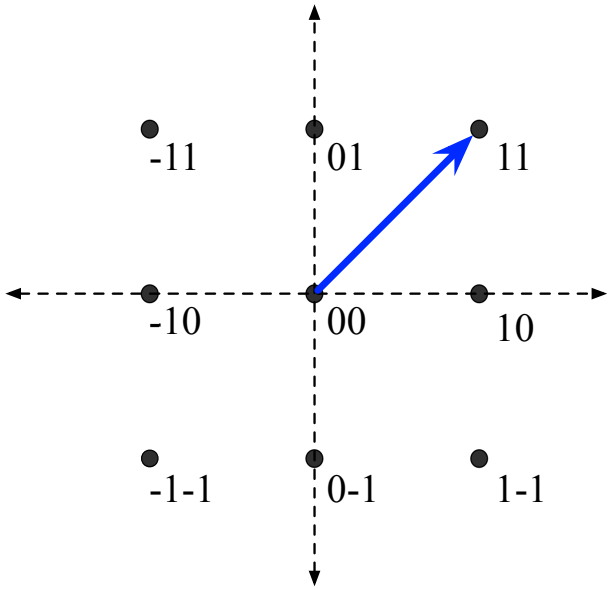
$$T_x(a_x a_y) = B_x^1 \cdot \mathbf{max}(a_x, 0) + (1 - |a_x|) \cdot \mathcal{O}_x \tag{11}$$
$$T_y(a_x a_y) = B_y^1 \cdot \mathbf{max}(a_y, 0) + (1 - |a_y|) \cdot \mathcal{O}_y, \tag{12}$$

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 4b



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

Figure 4: Region angles, and the unit vector

To actually employ rotations properly, we'll need to know what rotations actually need to be performed by region. Figure 4a 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 4a acquire their angle and magnitude. Figure 4b begins to unpack this 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 angles, and we'll call it 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 4a– it is straightforward to verify the derived procedure.

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.

procedure *AngleSearch*($a_x a_y \in \mathcal{R}$) **is**
    $\mathcal{S}' = \{\}$;
    for $s \in \mathcal{S}_u$ **do**
        $\mathcal{S}' = \mathcal{S}' \cup \{\mathbf{s} + [a_x,\ \ a_y]\}$
    $\mathcal{S}' = \mathcal{S}' \cap \mathcal{R}$;
    $\Theta = \{\}$;
    for $s \in \mathcal{S}'$ **do**
        $\mathbf{u} = \mathbf{s} - [a_x,\ \ a_y]$;
        $\Theta = \Theta \cup atan2(\mathbf{u} \cdot \mathbf{v}^u, |\mathbf{u}^T||\mathbf{v}^{uT}|)$;
    return Θ;

To find the angles enumerated in figure 4a, 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 4a along with the unit vector, $\mathbf{v}^u$.

Algorithm 3 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 4a. Finally, the angle between the induced 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 4a). The inputs to this function are the dot produce 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 3a, 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 $h, H, V : (a_x a_y, \theta) \rightarrow \mathbb{R}^+$.

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

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

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 outer 2 dimensions of each tensor starting in the bottom right corner of each matrix (i.e. $\mathbf{P}_{11}$ and $\mathbf{\Delta}_{11}$). 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 $\mathbf{\Delta}_{11}$). This observation on the pattern of matrix indexing according to angle rotations can be generalized for rows (denoted r) and columns (denoted c) of the first 2 dimensions of the tensors 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, from here on out, for any tensor M's outer 2 dimensions rc,

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

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. This transformation is given by the following function, where for state matrix M, region transformation $M_{a_x a_y}$, is defined as:

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

where the pre-multiplication flips the rows of matrix M, and the post-multiplication flips the columns of M. Again, given our relatively small state space, this can be verified as yielding appropriate matrices of state for each region.

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

$$\mathbf{p} = \mathbf{P}_{a_x a_y}[\theta] \tag{21}$$
$$\mathbf{b} = \mathbf{B}_{a_x a_y}[\theta] \tag{22}$$
$$\delta = \mathbf{\Delta}_{a_x a_y}[\theta] \tag{23}$$
$$\mathbf{u} = \mathbf{U}_{a_x a_y}[\theta]. \tag{24}$$

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

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

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

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

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

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

and using using it in equations (25)–(27). Algorithm 4 puts these equations together, with considerations for managing rotations and increments in discrete space.

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 $t - 1$.

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

```
Θ = AngleSearch(a_x a_y);
for θ ∈ Θ do
    for j = 0 to j = H(θ, a) do
        for k = 0 to k = min{V(θ, a), V(θ, a) − j + h(θ, a) do
            i = R_θ · [j,  k]^T;
            g(t, T_x(a) + i_x T_y(a) + i_y)) = f(t − 1, T_x(a) + i_x, T_y(a) + i_y)
```

4 Update Constraints for the Multi-Process Problem

Up until now we've been considering updates on a single process using the region 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 (which is the whole purpose of this research), then a few things immediately follow. In particular, 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 $\theta \in \mathit{AngleSearch}(a_x a_y)$, 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,

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

where $\Theta = \mathit{AngleSearch}(a_x a_y)$. 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

When a process' border cell requires a ghost cell to complete a timestep, it must have this ghost cell communicated to it by another process in order for the border cell to be updated, therefore, there must be means of ensuring that the communication is complete. 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 $\mathcal{N}(a_x a_y, \mathsf{pid}) : (a_x a_y, \mathsf{pid}) \to \{\mathbb{Z}^+\}$ 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 valid access of it.

Given these definitions for communication between neighbor process, 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:

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

4.3 Mirror Process Misalignment

Given a global spatial decomposition as described in section 1, a process whose local spatial decomposition is directly adjacent to it is referred to as a *mirror process*. 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 *mirror state*.

Figure 5 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 -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 box, and a timestep displayed within it. 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 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 exchnaged between

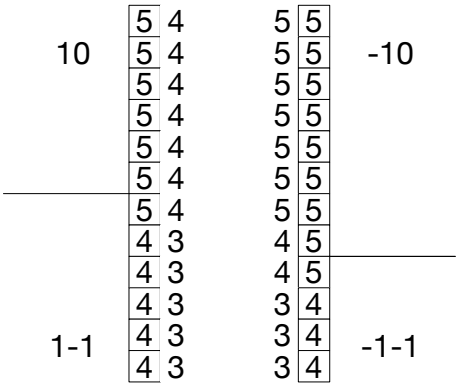


Figure 5: A motivating example for the utility of $\mathcal{G}_{\mathcal{M}}$ and $\mathcal{G}_{\mathcal{T}}$. A subset of the 2D data arrays of a left and right process are illustrated–ghost cells drawn without a border, and border cells given a border. A delimeter between regions on each process perpendicular 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

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

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

processes, and stored as locally in *mirror variables*,

$$\mathbf{\Delta}_n^M \equiv \mathbf{\Delta}\text{matrix of process neighbor process } n \tag{31}$$
$$\mathbf{P}_n^M \equiv \mathbf{P}\text{matrix of process neighbor process } n. \tag{32}$$

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 5, 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 between mirror quantities on the left and right processes. $H(\theta, \mathbf{a})$ is mirrored by $V^M(\theta - 90^o, \mathbf{a})$, and $V^M(\theta, \mathbf{a})$ 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

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 $r = x$ **mod** y, and r is the remainder from dividing x into y). Given its frequency, the shorthand $\overline{x} = x$ **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 being solved for. 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

$$\mathbb{I}_x(q, \theta, \mathbf{a}) = T_x + |a_x| \cdot (a_x - q) \cdot R_\theta \cdot [H(\theta, \mathbf{a}), V(\theta, \mathbf{a})]_x^T \tag{33}$$
$$\mathbb{I}_y(q, \theta, \mathbf{a}) = T_y + |a_y| \cdot (a_y - q) \cdot R_\theta \cdot [H(\theta, \mathbf{a}), V(\theta, \mathbf{a})]_y^T, \tag{34}$$

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

$$\mathbb{I}_x(q, \theta, \mathbf{a})^s = \mathbf{min}(\mathbf{max}(\mathbb{I}_x(q, \theta, \mathbf{a}) - q, 1), B_x^1) \tag{35}$$
$$\mathbb{I}_y(q, \theta, \mathbf{a})^s = \mathbf{min}(\mathbf{max}(\mathbb{I}_y(q, \theta, \mathbf{a}) - q, 1), B_y^1). \tag{36}$$

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

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

/* Initialize Global Variables. */

$t_{max} \leftarrow 1$;
$x_{-1} \leftarrow 1$; $x_1 \leftarrow B_x^1$; $y_{-1} \leftarrow 1$; $y_1 \leftarrow B_y^1$;
$\delta_{-11} \leftarrow 0$; $\delta_{-1-1} \leftarrow 0$; $\delta_{11} \leftarrow 0$; $\delta_{1-1} \leftarrow 0$;
for $a_x a_y \in \mathcal{R} - \{00\}$ **do**
  $\Theta = \text{AngleSearch}(a_x a_y)$;
  for $\theta \in \Theta$ **do**
    for $n \in \mathcal{N}(a_x a_y)$ **do**
      $\mathsf{send}_{a_x a_y}[n] \leftarrow postSend(n, \mathbf{\Delta})$; $\mathsf{rcv}_{a_x a_y}[n] \leftarrow postRecv(n, \mathbf{\Delta}_n^M)$;
      $\mathsf{send}_{a_x a_y}[n] \leftarrow postSend(n, \mathbf{P})$; $\mathsf{rcv}_{a_x a_y}[n] \leftarrow postRecv(n, \mathbf{P}_n^M)$;
      $\mathsf{send}_{a_x a_y}[n] \leftarrow postSend(n, \mathsf{data}[1][\mathbb{I}_y(-1,\theta,\mathbf{a})^s..\mathbb{I}_y(+1,\theta,\mathbf{a})^s][\mathbb{I}_x(-1,\theta,\mathbf{a})^s..\mathbb{I}_x(+1,\theta,\mathbf{a})^s])$;
       $\mathsf{rcv}_{a_x a_y}[n] \leftarrow postRecv(n, \mathsf{data}[0][\mathbb{I}_y(-1,\theta,\mathbf{a})..\mathbb{I}_y(+1,\theta,\mathbf{a})][\mathbb{I}_x(-1,\theta,\mathbf{a})..\mathbb{I}_x(+1,\theta,\mathbf{a})])$;
while $t_{max} < \mathit{nsteps} \vee x_{-1} > 0 \vee y_{-1} > 0 \vee x_1 < B_x^1 \vee y_{-1} > B_y^1$ **do**
  for $a_x a_y \in \mathcal{R}$ **do**
    $\Theta = \text{AngleSearch}(a_x a_y)$;
    when $\mathcal{G}^S(a_x a_y, \Theta) \wedge (a_x a_y = 00 \vee (\mathcal{G}^C(a_x a_y, \mathsf{pid}) \wedge \mathcal{G}^M(a_x a_y, \Theta)))$
      $\theta_c = 0$;
      for $\theta \in \Theta$ **do**
        $\mathbf{p} = \mathbf{P}_{a_x a_y}[\theta]$;
        $\mathbf{b} = \mathbf{B}_{a_x a_y}[\theta]$;
        $\delta = \mathbf{\Delta}_{a_x a_y}[\theta]$;
        $\mathbf{u} = \mathbf{U}_{a_x a_y}[\theta]$;
        $h(\theta,\mathbf{a}) = [R_{-\theta}\cdot\mathbf{p}^T]_x + [R_{-\theta}\cdot\mathbf{a}^T]_x \cdot [R_{-\theta}\cdot\mathbf{b}^T]_x - [R_{-\theta}\cdot\mathcal{O}]_x \cdot (1 - |R_{-\theta}\cdot\mathbf{a}^T|_x) + (1 - |R_{-\theta}\cdot\mathbf{a}^T|_y)\cdot\mathbf{min}_{x,y}(R_{-\theta}\cdot\mathbf{u})\cdot\delta$;
        $H(\theta,\mathbf{a}) = [R_{-\theta}\cdot\mathbf{p}^T]_x + [R_{-\theta}\cdot\mathbf{a}^T]_x \cdot [R_{-\theta}\cdot\mathbf{b}^T]_x - [R_{-\theta}\cdot\mathcal{O}]_x \cdot (1 - |R_{-\theta}\cdot\mathbf{a}^T|_x) + |R_{-\theta}\cdot\mathbf{a}^T|_y\cdot[R_{-\theta}\cdot\mathbf{u}]_x\cdot\delta$;
        $V(\theta,\mathbf{a}) = [R_{-\theta}\cdot\mathbf{p}^T]_y + [R_{-\theta}\cdot\mathbf{a}^T]_y \cdot [R_{-\theta}\cdot\mathbf{b}^T]_y - [R_{-\theta}\cdot\mathcal{O}]_y \cdot (1 - |R_{-\theta}\cdot\mathbf{a}^T|_y) + |R_{-\theta}\cdot\mathbf{a}^T|_x\cdot[R_{-\theta}\cdot\mathbf{u}]_y\cdot\delta$;
        $\mathbf{q} = R_\theta \cdot [1,\ \ 1]^T$;
        for $j = 0$ **to** $j = H(\theta,\mathbf{a}) - \mathbf{q}_x$ **do**
          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**
            $\mathbf{i} = R_\theta \cdot [j,\ \ k]^T$;
            $\mathsf{data}[1 - \overline{t(a_x a_y, j, k)}][T_y(\mathbf{a}) + \mathbf{i}_y) + \mathbf{q}_y\cdot(1-|a_y|)\cdot\mathbf{min}((\theta_c \bmod 3), 1)][T_x(\mathbf{a}) + \mathbf{i}_x + \mathbf{q}_x\cdot(1-|a_x|)\cdot\overline{\theta_c}] =$
             $f'(\mathsf{data}[\overline{t(a_x a_y, j, k)}][T_y(\mathbf{a}) + \mathbf{i}_y + \mathbf{q}_y\cdot(1-|a_y|)\cdot\mathbf{min}((\theta_c \bmod 3), 1)][T_x(\mathbf{a}) + \mathbf{i}_x + \mathbf{q}_x\cdot(1-|a_x|)\cdot\overline{\theta_c}])$
        $\theta_c = \theta_c + 1$;
      when $a_x a_y \neq 00$
        for $\theta \in \Theta$ **do**
          for $n \in \mathcal{N}(a_x a_y)$ **do**
            $\mathsf{send}_{a_x a_y}[n] \leftarrow postSend(n, \mathbf{\Delta})$; $\mathsf{rcv}_{a_x a_y}[n] \leftarrow postRecv(n, \mathbf{\Delta}_n^M)$;
            $\mathsf{send}_{a_x a_y}[n] \leftarrow postSend(n, \mathbf{P})$; $\mathsf{rcv}_{a_x a_y}[n] \leftarrow postRecv(n, \mathbf{P}_n^M)$;
            $\mathsf{send}_{a_x a_y}[n] \leftarrow postSend(n, \mathsf{data}[1 - \overline{t(a_x a_y, j, k)}][\mathbb{I}_y(-1,\theta,\mathbf{a})^s..\mathbb{I}_y(+1,\theta,\mathbf{a})^s][\mathbb{I}_x(-1,\theta,\mathbf{a})^s..\mathbb{I}_x(+1,\theta,\mathbf{a})^s])$;
             $\mathsf{rcv}_{a_x a_y}[n] \leftarrow postRecv(n, \mathsf{data}[\overline{t(a_x a_y, j, k)}][\mathbb{I}_y(-1,\theta,\mathbf{a})..\mathbb{I}_y(+1,\theta,\mathbf{a})][\mathbb{I}_x(-1,\theta,\mathbf{a})..\mathbb{I}_x(+1,\theta,\mathbf{a})])$;
      when $a_x a_y = 00$
        $t_{max} \leftarrow t_{max} + 1$;
    $update\_state(a_x a_y)$;