PX425 Assignment 4

phuwcs (1833194)
Department of Physics, University of Warwick

(Dated: December 10, 2021)

Parallelise a program using a Cartesian domain decomposition with MPI.

CONTENTS

I. GETTING MPI UP AND RUNNING

To get MPI up and running I followed the instructions in the script. This image (1) was saved from the first successful run.

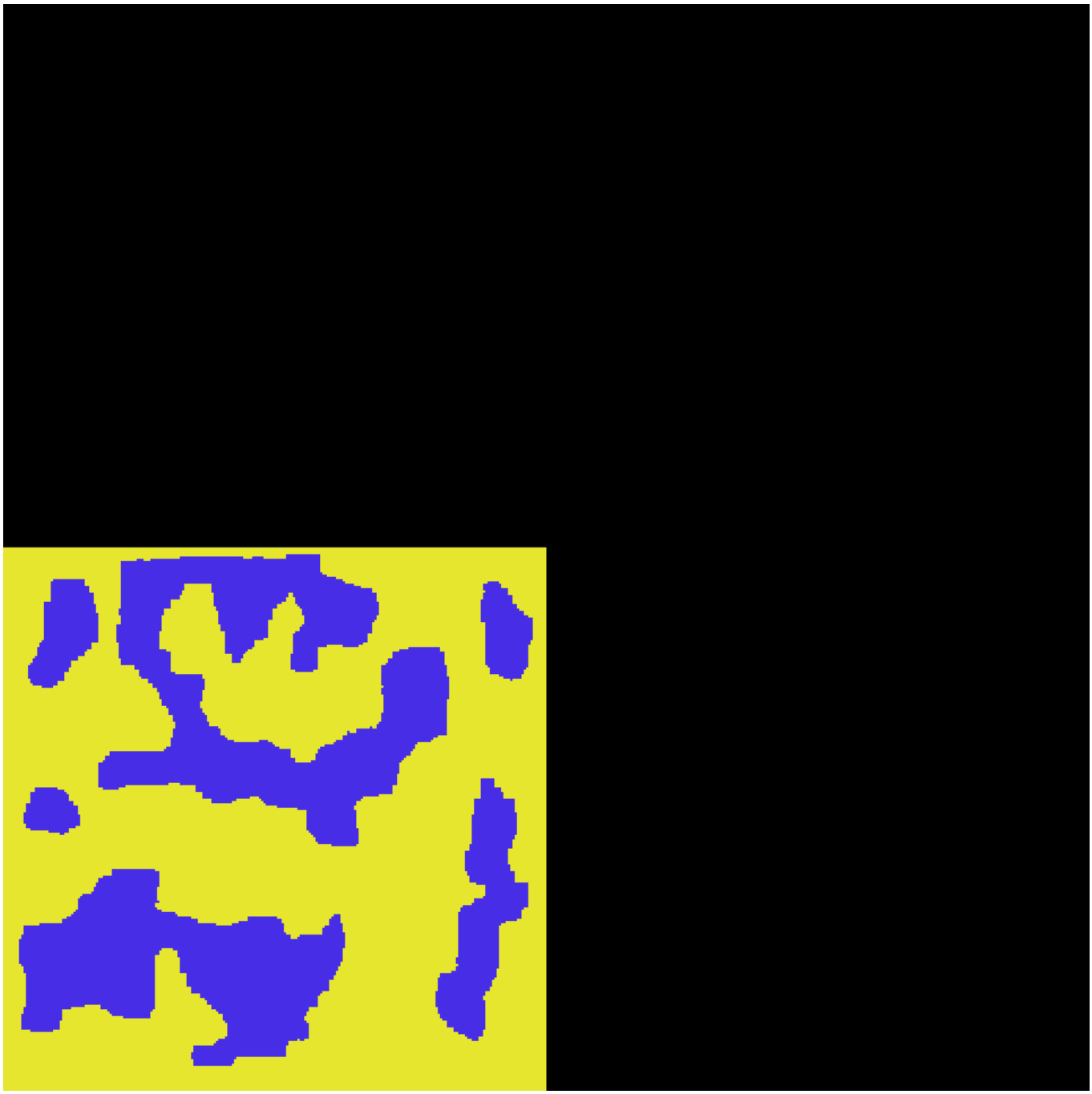

Figure 1. The final snapshot after MPI was first set up. Only the bottom left quarter of the image (coordinates 0, 0) is shown.

II. SHARING THE WORK - SETTING UP A CARTESIAN TOPOLOGY

I used `MPI_Cart_create` to create a Cartesian topology as expected. I also stored the process's current rank in the Cartesian communicator for this step, in case it was reordered from the world rank. While `reorder` is currently set to `0`, I thought this would be more resilient if it needed to be change. With the rank within the communicator at hand, the coordinates of the current rank are available via `MPI_Cart_coords`. The ranks of the neighbours could be found easily enough with `MPI_Cart_rank` after that. Outputs of the print statements are shown below:

```
With 1 processor:
Size of each local processor grid :   480 x   480
```

```
My rank        : 0
My coordinates: 0 0
Up             : 0
Down           : 0
Left           : 0
Right          : 0

With 4 processors:
Size of each local processor grid :   240 x   240
My rank        : 0
My coordinates: 0 0
Up             : 1
Down           : 1
Left           : 2
Right          : 2

My rank        : 1
My coordinates: 0 1
Up             : 0
Down           : 0
Left           : 3
Right          : 3

My rank        : 2
My coordinates: 1 0
Up             : 3
Down           : 3
Left           : 0
Right          : 0

My rank        : 3
My coordinates: 1 1
Up             : 2
Down           : 2
Left           : 1
Right          : 1

With 9 processors:
Size of each local processor grid :   160 x   160
My rank        : 0
My coordinates: 0 0
Up             : 1
Down           : 2
Left           : 6
Right          : 3

My rank        : 1
My coordinates: 0 1
Up             : 2
Down           : 0
Left           : 7
Right          : 4

My rank        : 2
My coordinates: 0 2
Up             : 0
Down           : 1
Left           : 8
```

```
Right          : 5

My rank        : 3
My coordinates: 1 0
Up             : 4
Down           : 5
Left           : 0
Right          : 6

My rank        : 4
My coordinates: 1 1
Up             : 5
Down           : 3
Left           : 1
Right          : 7

My rank        : 5
My coordinates: 1 2
Up             : 3
Down           : 4
Left           : 2
Right          : 8

My rank        : 6
My coordinates: 2 0
Up             : 7
Down           : 8
Left           : 3
Right          : 0

My rank        : 7
My coordinates: 2 1
Up             : 8
Down           : 6
Left           : 4
Right          : 1

My rank        : 8
My coordinates: 2 2
Up             : 6
Down           : 7
Left           : 5
Right          : 2
```

With a quick sketch I was able to convince myself that this was correct.

III. COLLECTING DATA FROM ALL PROCESSORS

When editing `comms_get_global_grid`, in order to ensure that data from ranks cannot be confused, I specified the source and desination rank explicitly in `MPI_Send` and `MPI_Recv`.

For `comms_get_global_mag`, an `MPI_Allreduce` with `MPI_SUM` was appropriate to add up all the local magnisation values. The code already in the function then divided by the number of processors to get the average.

(2) shows the final results from this step with four processors.

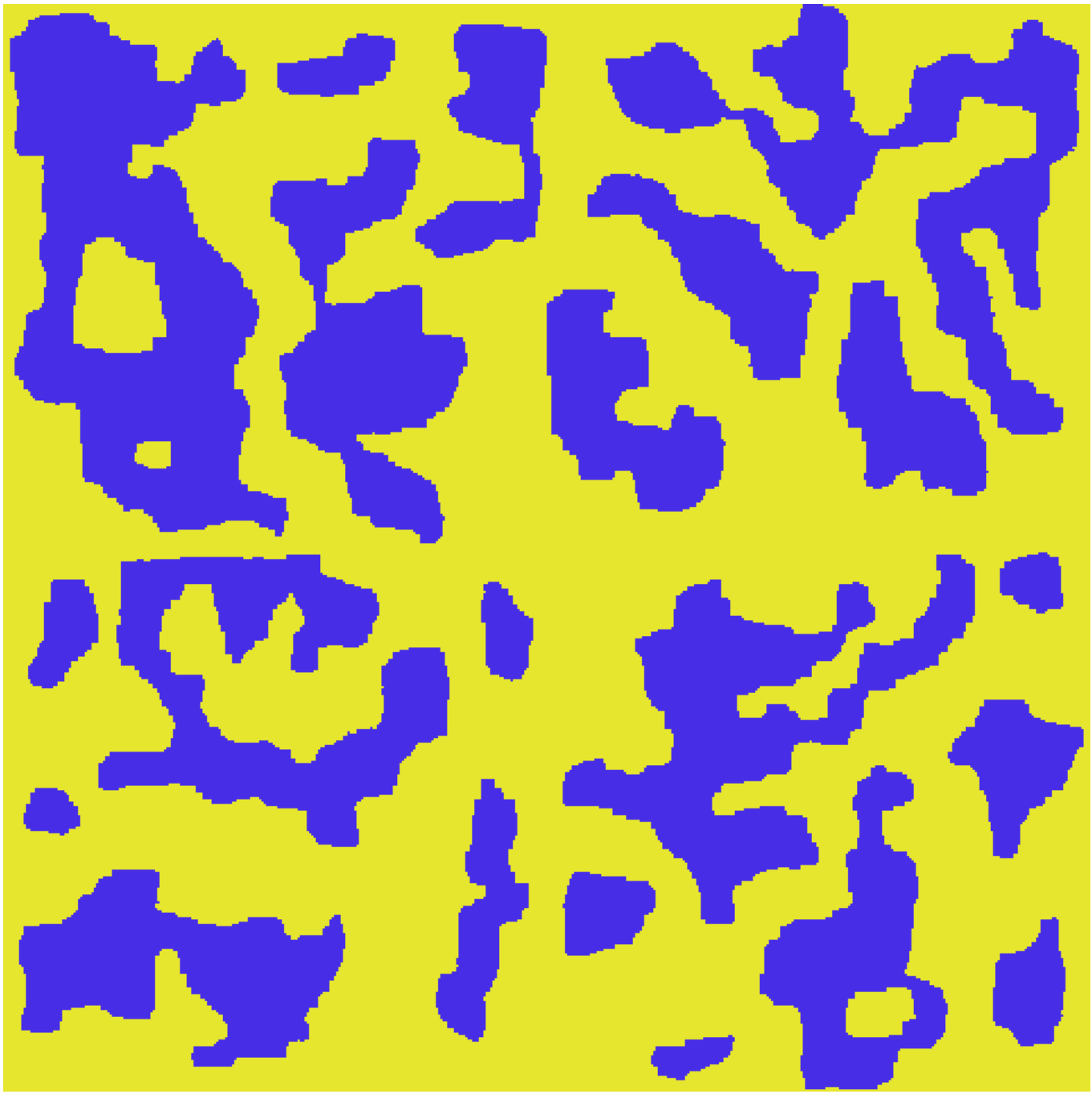

Figure 2. Final results after collecting data from all processors. The borders between the four quadrants are evidence of the number of processors and lack of halo swapping to correctly evolve the system over time.

IV. HALO SWAPPING

This step was a bit fiddly and took a bit of time to get correct. In particular, matching "left", "right", "up", and "down" to the coordinate system required careful thought. I used `MPI_Sendrecv` since the ranks would be sending and receiving data simultaneously, with source and destination ranks set as well as a specific tag for each direction. This ensured that no data would be sent to the wrong place. Where possible I also used `memcpy` to copy to and from `sendbuf` and `recvbuf`. After a few failed attempts, a good looking result popped out: (3).

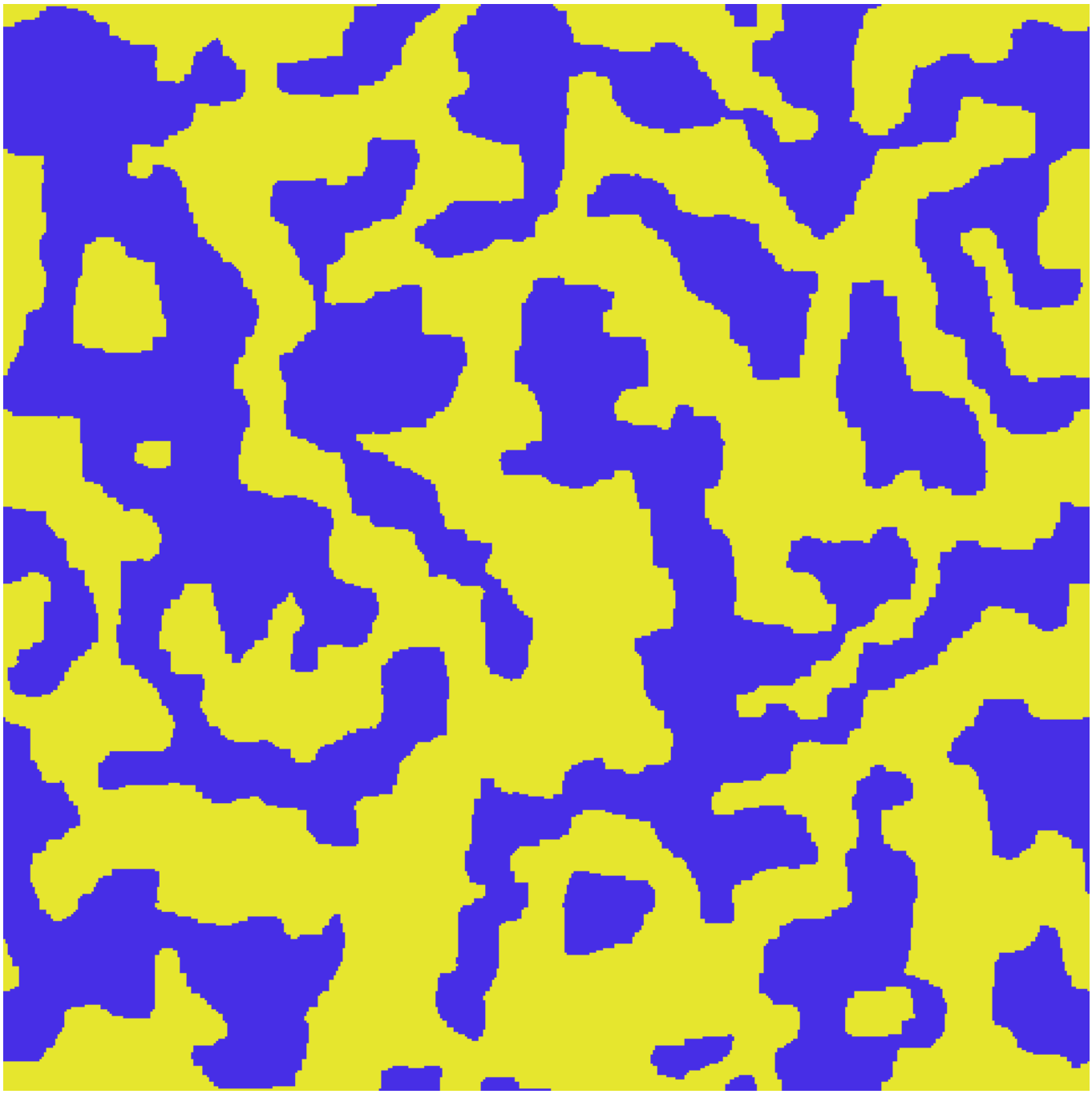

Figure 3. Final data after halo swapping was correctly implemented, with four processors. There are no obvious lines through the middle of the data and the edges also correctly obey periodic boundary conditions.

V. TIMINGS

Table I on the following page shows the timings recorded from the various configurations required. The larger the size of the grid, the slower the simulation runs. The more processors, the faster the simulation runs. The effect that the number of processors has decreases the more processors there are. The grid works the opposite way, with each larger size slowing the overall simulation down more and more (since the total number of points is N_{grid}^2). I didn't notice anything especially unusual about these results.

Ngrid	P	Time / s	$\psi(N)$
480	1	14.5	1.00
	4	3.50	4.14
	9	1.43	10.1
	16	0.787	18.4
	25	0.560	25.9
600	1	31.9	1.00
	4	5.94	5.37
	9	2.38	13.4
	16	1.26	25.3
	25	0.837	38.1
720	1	51.2	1.00
	4	11.4	4.49
	9	3.76	13.6
	16	1.90	26.9
	25	1.64	31.2
840	1	75.0	1.00
	4	16.9	4.44
	9	6.19	12.1
	16	2.78	27.0
	25	2.20	34.1

Table I. Time taken to execute the simulation as measured by `MPI_Wtime`. $\psi(N) = \frac{T(1)}{T(P)}$.

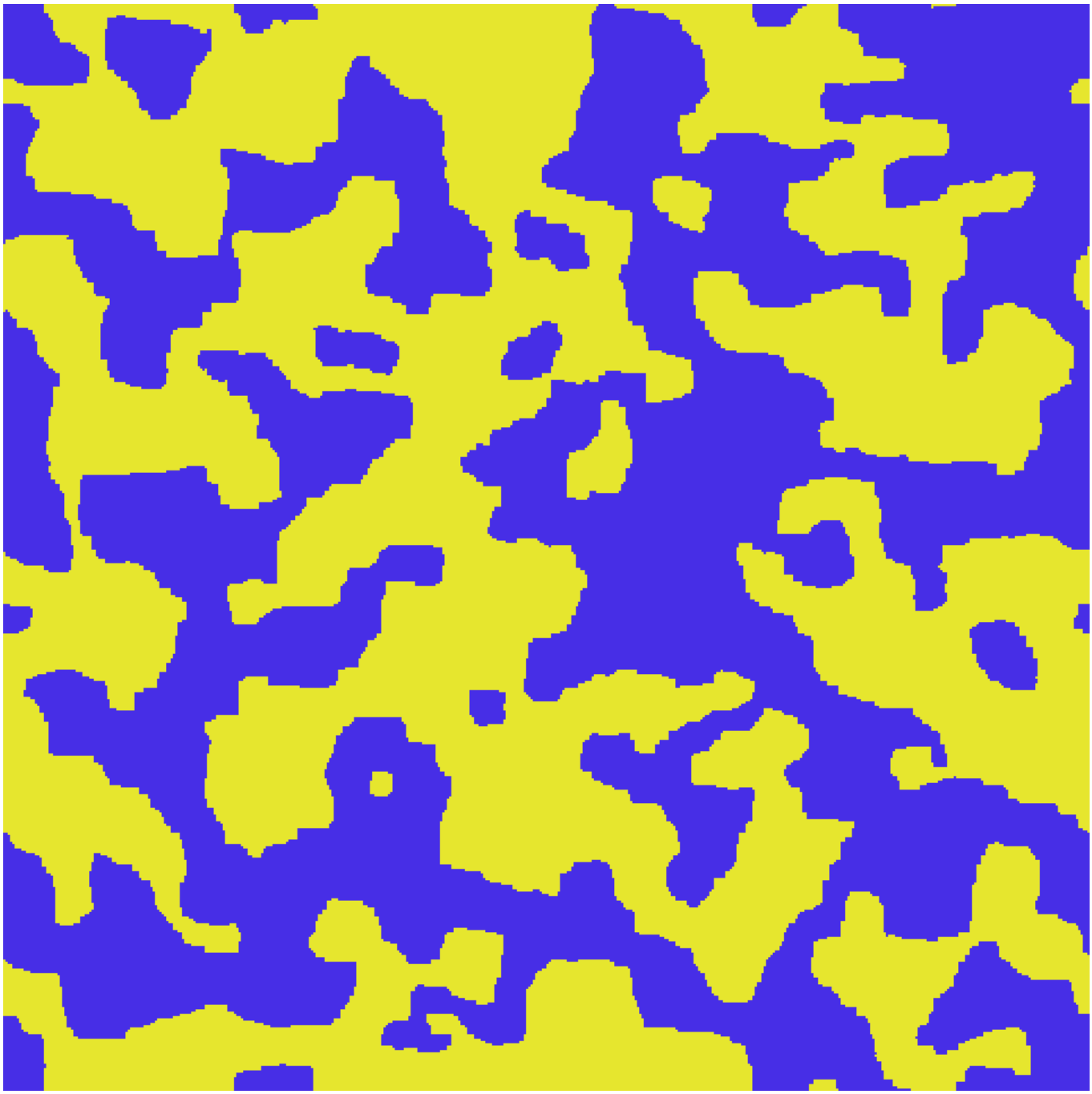

Figure 4. Final image with `Ngrid=480` and `P=9`, attached as `image.png`. Note that the precise pattern is different to figure 3 on page 6 because of the random number generator generating different numbers for each process, so as `P` varies so will the final outcome.