Parallel approach to Jacobi iteration technique for Numerical Solution of Laplace’s equation

Doniyor Babajanov

Turin Polytechnic University in Tashkent, Uzbekistan

Content

- Laplace's equation
- Jacobi iteration technique
- OpenMP parallelization
- 1d MPI parallelization
- 2d MPI parallelization

Laplace's equation

$$\frac{\partial^2 \varphi}{\partial x^2} + \frac{\partial^2 \varphi}{\partial y^2} = 0$$

$$\varphi(boundaries) = f(x, y)$$

$$\varphi(x, y) = trial\ function$$

Jacobi iteration technique

$$\varphi(x, y) = V_{i,j}$$

$$\frac{V_{i+1,j} - 2V_{i,j} + V_{i-1,j}}{h^2} + \frac{V_{i,j+1} - 2V_{i,j} + V_{i,j-1}}{t^2} = 0$$

$$h = t$$

$$V_{i,j} = \frac{1}{4}\left(V_{i+1,j} + V_{i-1,j} + V_{i,j+1} + V_{i,j-1}\right)$$

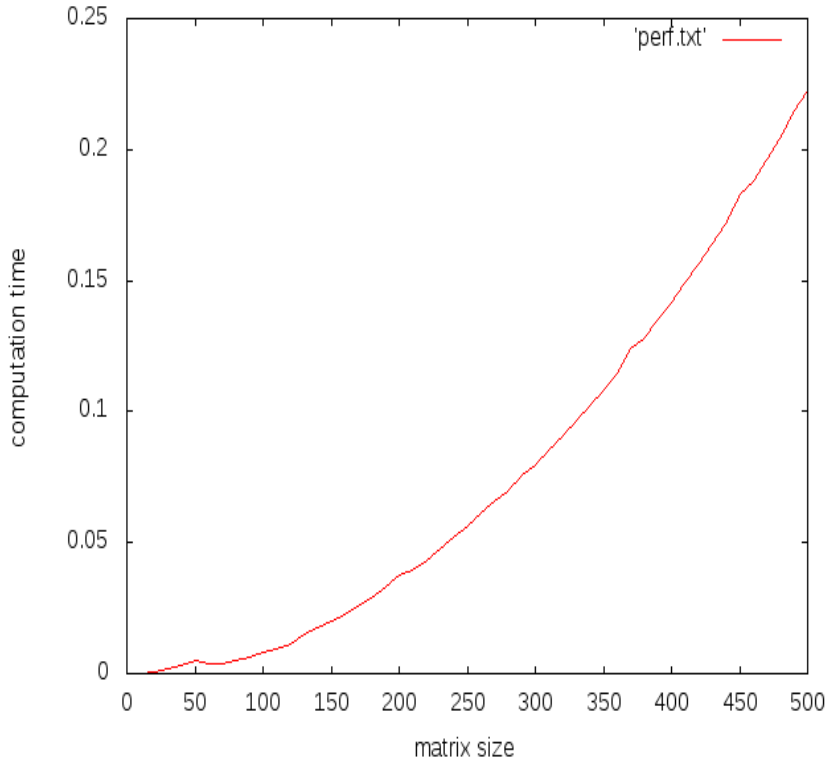
Jacobi iteration technique

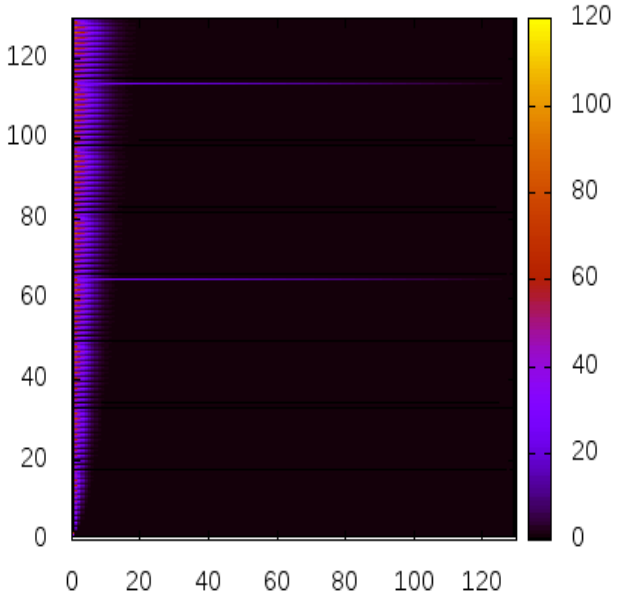
$$V_{i,j} = 0.25\left(V_{i+1,j} + V_{i-1,j} + V_{i,j+1} + V_{i,j-1}\right)$$

0	0	0	0	0	0	0	0	0	0	0
10										0
20										0
30			$V_{i+1,j}$							0
40		$V_{i,j-1}$	$V_{i,j}$	$V_{i,j+1}$						0
50			$V_{i-1,j}$							0
60										0
70										0
80										0
90										0
100	90	80	70	60	50	40	30	20	10	0

Figure 1: A diagram of the Jacobi Relaxation for Solving the Laplace's Equation on an evenly spaced 9x9 grid with the boundary conditions outlined in the text above.

Results from serial program



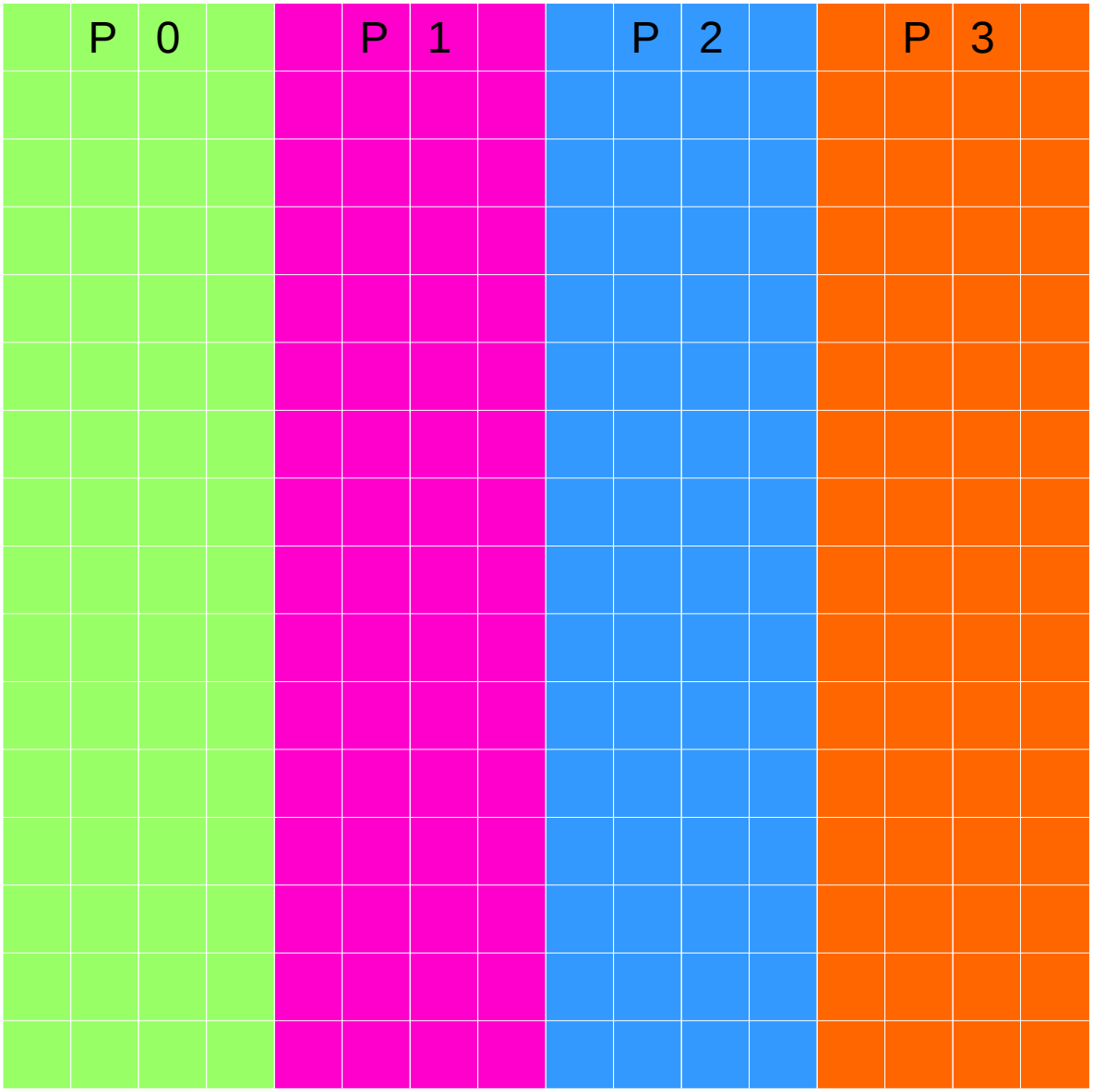


OpenMP parallelization

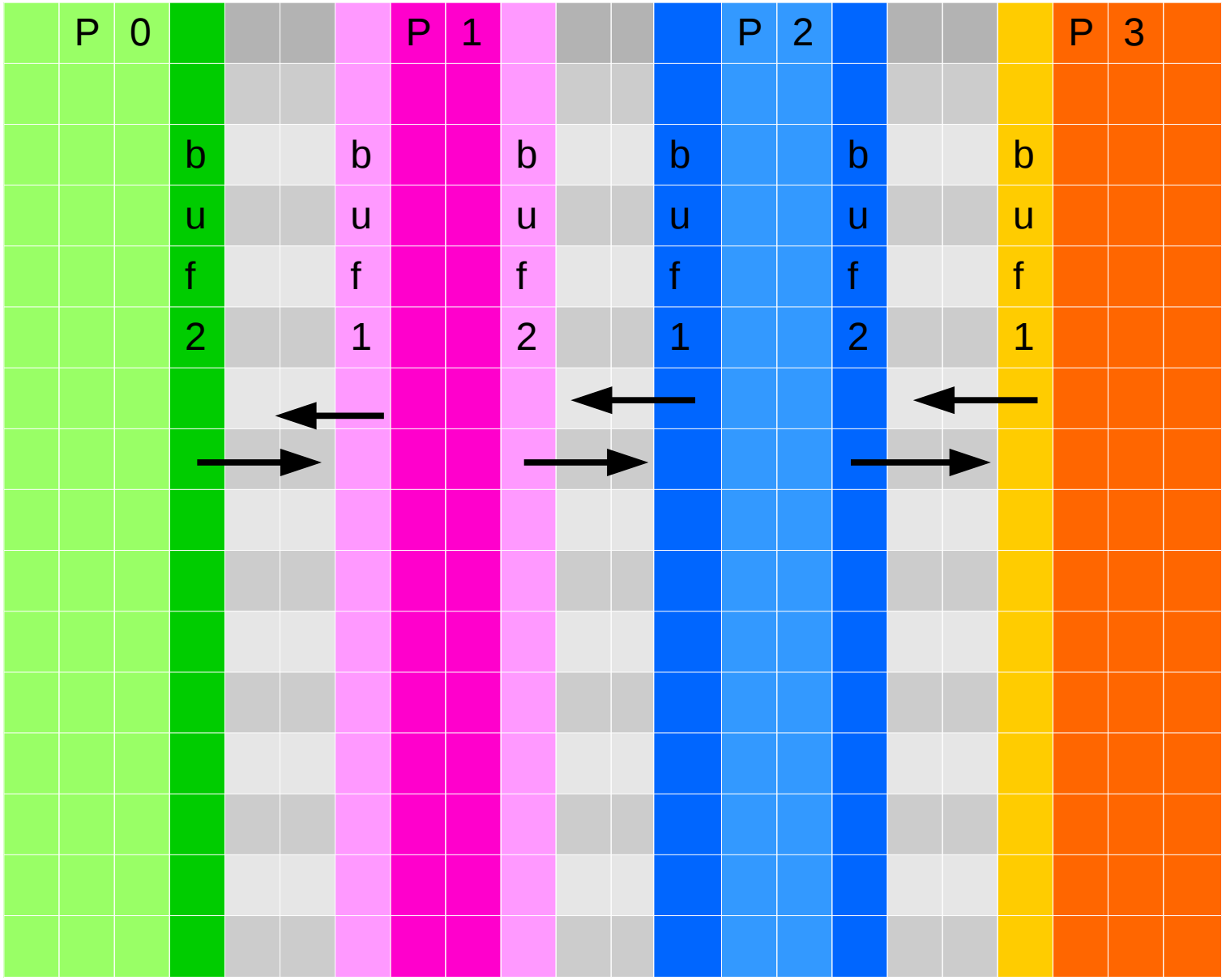
```
!$OMP PARALLEL DO PRIVATE(i,j) NUM_THREADS(nth)
      DO i = 2,Dim+1
            DO j = 2,Dim+1
                        SurfaceMatrix_t(j,i) = 0.25*(SurfaceMatrix(j-1,i) +SurfaceMatrix(j,i+1)
                              +SurfaceMatrix(j+1,i) +SurfaceMatrix(j,i-1))
             ENDDO
       ENDDO
!$OMP END PARALLEL DO
```

```
!$OMP PARALLEL DO PRIVATE(i,j) NUM_THREADS(nth)
      DO i=2,Dim
            DO j=2,Dim
                SurfaceMatrix(j,i) = SurfaceMatrix_t(j,i)
            ENDDO
       ENDDO
!$OMP END PARALLEL DO
```

1D MPI parallelization



1D MPI parallelization



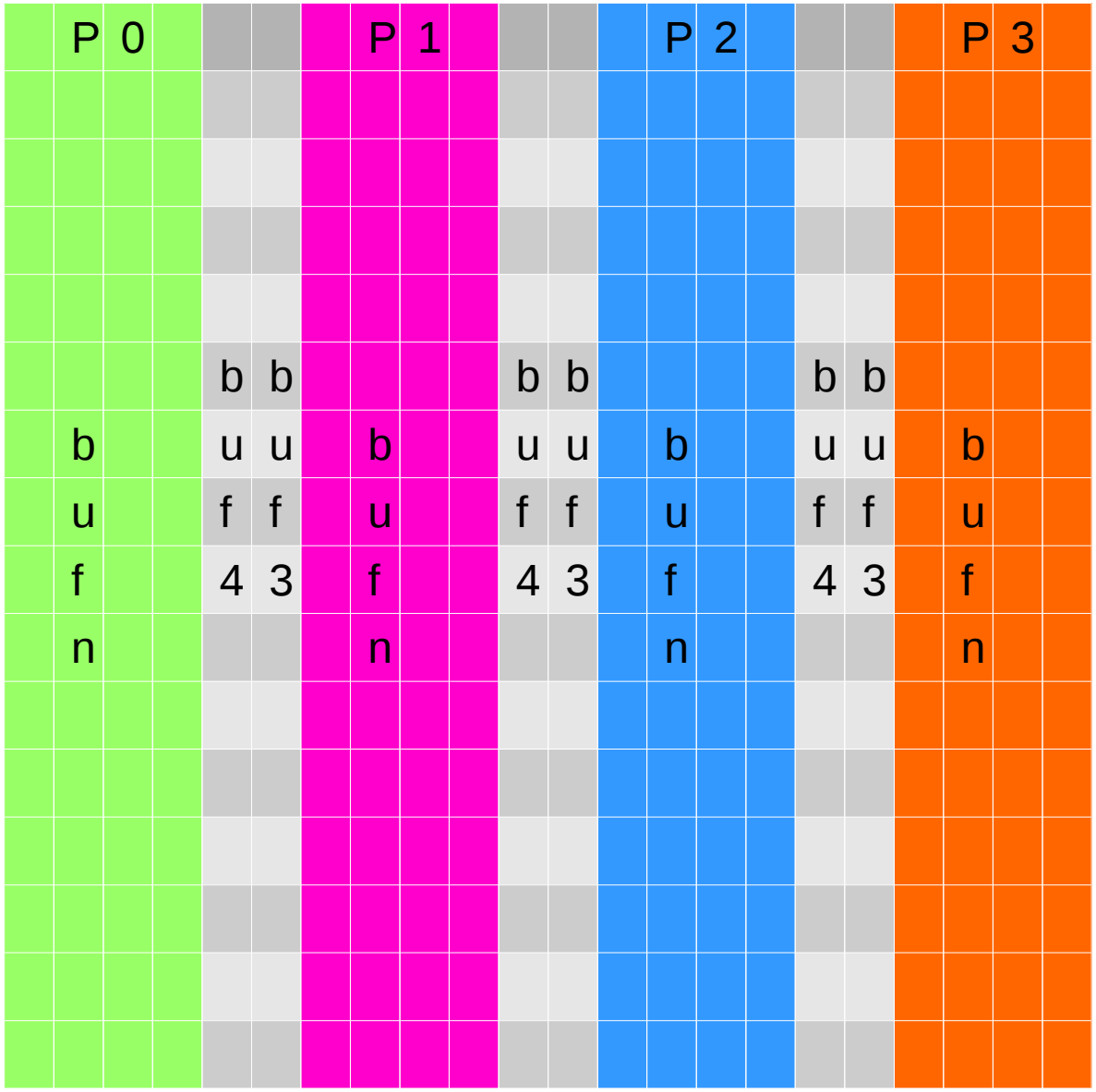
1D MPI parallelization non-blocking communications

```
if(RANK.ne.0) then
  call MPI_ISEND(buf1, Dim, MPI_REAL, RANK-1, 0, MPI_COMM_WORLD,
      REQUEST, IERROR)
endif

if(RANK.ne.CSIZE-1) then
  call MPI_ISEND(buf2, Dim, MPI_REAL, RANK+1, 0, MPI_COMM_WORLD,
      REQUEST, IERROR)
endif
```

1D MPI parallelization non-blocking communications

```
if(RANK.ne.0) then
    call MPI_IRECV(buf3, Dim, MPI_REAL, RANK-1, 0, MPI_COMM_WORLD,
        REQUEST1, IERROR)
    call MPI_WAIT(REQUEST1, MPI_STATUS_IGNORE, IERROR)
  endif

  if(RANK.ne.CSIZE-1) then
    call MPI_IRECV(buf4, Dim, MPI_REAL, RANK+1, 0, MPI_COMM_WORLD,
        REQUEST2, IERROR)
    call MPI_WAIT(REQUEST2, MPI_STATUS_IGNORE, IERROR)
  endif
```

1D MPI parallelization

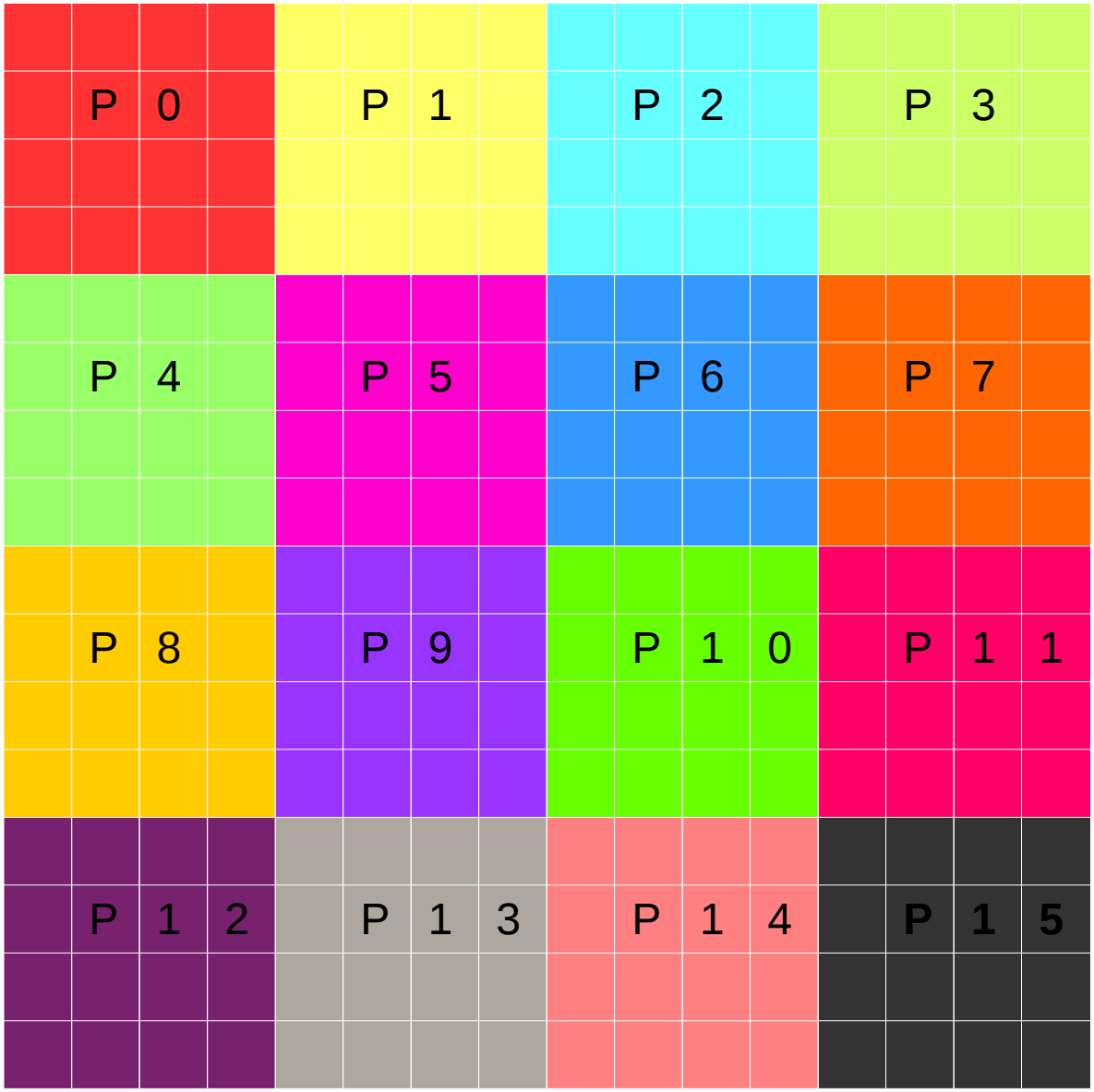


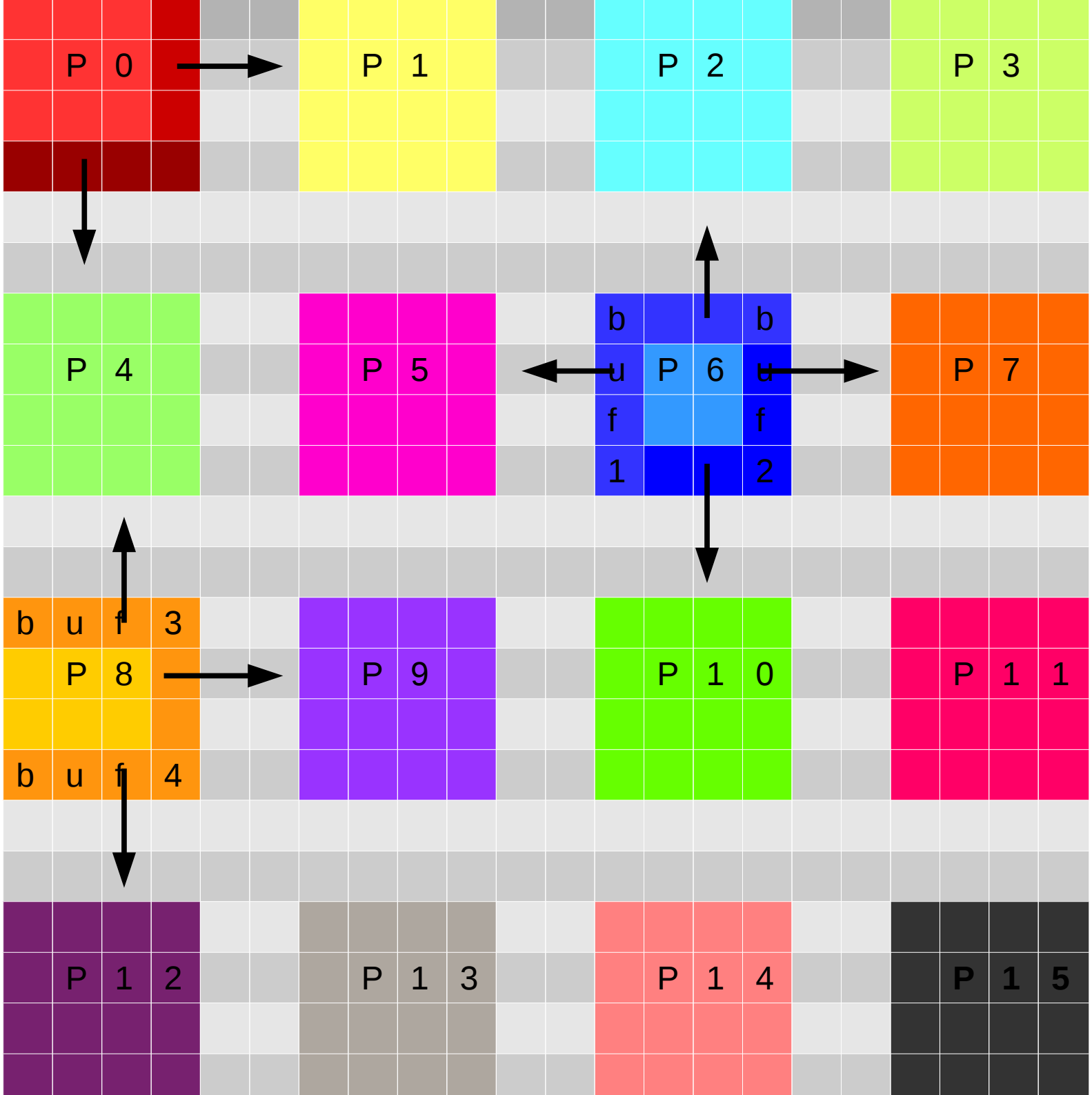
1D MPI parallelization non-blocking communications

```
call MPI_GATHER(bufn, nel*Dim, MPI_REAL, buffin, nel*Dim, MPI_REAL, 0,
      MPI_COMM_WORLD, IERROR)
```

```
if(RANK==0) then
   DO i=1,Dim+2
       DO j=1,Dim+2
           write(7,*) j, i, buffin(j,i)
       ENDDO
    ENDDO
 endif
```

2D MPI parallelization




P 0
P 1
P 2
P 3
P 4
P 5
b u f 1
P 6
b u f 2
P 7
b u f 3
P 8
P 9
P 10
P 11
b u f 4
P 12
P 13
P 14
P 15

2D MPI non-blocking send

```
if(MOD(RANK,nppl).ne.0) then
        call MPI_ISEND(buf1, nel, MPI_REAL, RANK-1, 0, MPI_COMM_WORLD,
            REQUEST, IERROR)
endif

if(MOD((RANK+1),nppl).ne.0) then
        call MPI_ISEND(buf2, nel, MPI_REAL, RANK+1, 0, MPI_COMM_WORLD,
                REQUEST, IERROR)
endif

if(RANK.ge.nppl) then
        call MPI_ISEND(buf3, nel, MPI_REAL, RANK-nppl, 0, MPI_COMM_WORLD,
            REQUEST, IERROR)
endif

if(RANK.lt.(CSIZE-nppl)) then
        call MPI_ISEND(buf4, nel, MPI_REAL, RANK+nppl, 0, MPI_COMM_WORLD,
            REQUEST, IERROR)
endif
```

2D MPI non-blocking receive

```
if(MOD(RANK,nppl).ne.0) then
        call MPI_IRECV(buf5, nel, MPI_REAL, RANK-1, 0, MPI_COMM_WORLD,
                REQUEST1, IERROR)
        call MPI_WAIT(REQUEST1, MPI_STATUS_IGNORE, IERROR)
endif

if(MOD((RANK+1),nppl).ne.0) then
        call MPI_IRECV(buf6, nel, MPI_REAL, RANK+1, 0, MPI_COMM_WORLD,
                REQUEST2, IERROR)
        call MPI_WAIT(REQUEST2, MPI_STATUS_IGNORE, IERROR)
endif

if(RANK.ge.nppl) then
        call MPI_IRECV(buf7, nel, MPI_REAL, RANK-nppl, 0, MPI_COMM_WORLD,
            REQUEST3, IERROR)
        call MPI_WAIT(REQUEST3, MPI_STATUS_IGNORE, IERROR)
endif

if(RANK.lt.(CSIZE-nppl)) then
        call MPI_IRECV(buf8, nel, MPI_REAL, RANK+nppl, 0, MPI_COMM_WORLD,
            REQUEST4, IERROR)
        call MPI_WAIT(REQUEST4, MPI_STATUS_IGNORE, IERROR)
endif
```

2D MPI combining & printing

```
call MPI_GATHER(bufn, nel*nel, MPI_REAL, buffin, nel*nel, MPI_REAL, 0,
      MPI_COMM_WORLD, IERROR)

if(RANK==0) then
   DO i=2,Dim+1
       DO j=2,Dim+1
          write(7,*) j, i, buffin(j,i)
        ENDDO
    ENDDO
endif
```