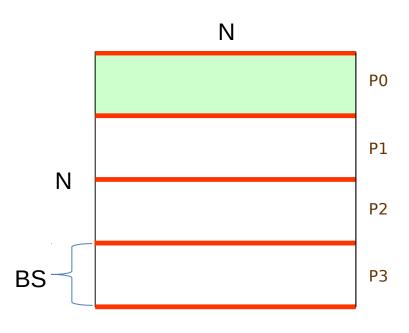
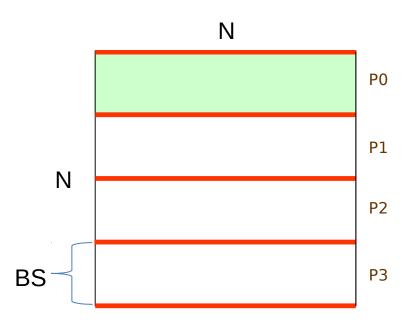
Block geometric data decomposition by rows of input data Matrix_in. Variable sum replicated



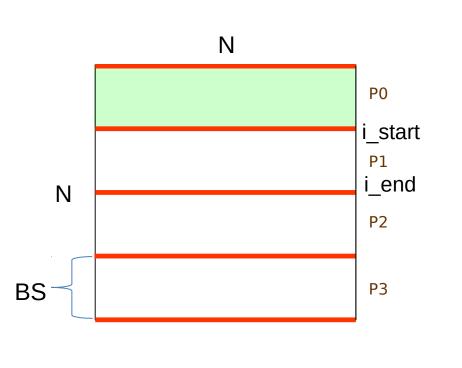
```
// Sequential
Sum = 0;
for(i=0; i<N; i++)</pre>
    for (j=0; j<N; j++)
        sum+= f(Matrix_in[i][j])
```

```
nt = number of threads
BS = number of rows in a block equal to N/nt
```



```
// Parallel version using omp for
sum = 0;
#pragma omp parallel
#pragma omp for private(j) schedule(static) reduction(+:sum)
for(i=0; i<N; i++)
   for (j=0; j<N; j++)
      sum+= f(Matrix_in[i][j])
```

```
nt = number of threads
BS = number of rows in a block equal to N/nt
```



```
nt = number of threads }
BS = number of rows in a block equal to N/nt
```

```
// Parallel version using omp for
sum = 0;
#pragma omp parallel
#pragma omp for private(j) schedule(static) reduction(+:sum)
for(i=0; i<N; i++)
   for (j=0; j<N; j++)
      sum+= f(Matrix_in[i][j])
// Parallel version without omp for
sum = 0;
#pragma omp parallel private(i,j) reduction(+:sum)
   int i_start = ?
   int i_end = ?
   for(i=i_start; i<i_end; i++)</pre>
     for (j=0; j<N; j++)
        sum+= f(Matrix_in[i][j])
```

```
// Parallel version using omp for
                                       sum = 0;
                                       #pragma omp parallel
                                       #pragma omp for private(j) schedule(static) reduction(+:sum)
                Ν
                                       for(i=0; i<N; i++)
                                          for (j=0; j<N; j++)
                             P0
                                             sum+= f(Matrix_in[i][j])
                            i start=1*BS
                                               // Parallel version without omp for
                             Ρ1
                                                sum = 0;
                             i end = i start+BS
   Ν
                                                #pragma omp parallel private(i,j) reduction(+:sum)
                             P2
                                                   int my_id = omp_get_thread_num();
                                                   int BS = N/omp_get_num_threads();
                             Р3
BS
                                                   int i_start = my_id * BS;
                                                   int i end = i start + BS;
                                                   for(i=i_start; i<i_end; i++)</pre>
                                                     for (j=0; j<N; j++)
   nt = number of threads
                                                        sum+= f(Matrix_in[i][j])
   BS = number of rows in a block equal to N/nt
```

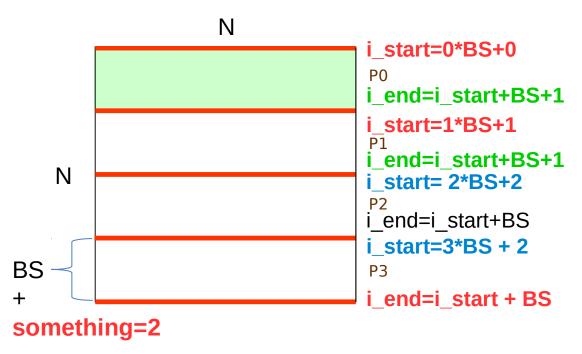
```
// Parallel version using omp for
                                        sum = 0;
                                        #pragma omp parallel
                                        #pragma omp for private(j) schedule(static) reduction(+:sum)
                 Ν
                                        for(i=0; i<N; i++)
                                           for (j=0; j<N; j++)
                             P0
                                              sum+= f(Matrix_in[i][j])
                             i start=1*BS
                                                 // Parallel version without omp for
                             Ρ1
                                                 sum = 0;
                             i end = i start+BS
   Ν
                                                 #pragma omp parallel private(i,j) reduction(+:sum)
                             P2
                                                                                      n();
                                                Is N multiple of nt?
                                                                                      ads();
                             Р3
BS
                                                    for(i=i_start; i<i_end; i++)</pre>
                                                      for (j=0; j<N; j++)
   nt = number of threads
                                                         sum+= f(Matrix_in[i][j])
   BS = number of rows in a block equal to N/nt
```

```
// Parallel version using omp for
                                       sum = 0;
                                       #pragma omp parallel
                                       #pragma omp for private(j) schedule(static) reduction(+:sum)
                Ν
                                       for(i=0; i<N; i++)
                                          for (j=0; j<N; j++)
                             P0
                                             sum+= f(Matrix_in[i][j])
                            i start=1*BS
                                                // Parallel version without omp for
                             Ρ1
                                                sum = 0;
                             i end = i start+BS
   Ν
                                                #pragma omp parallel private(i,j) reduction(+:sum)
                             P2
                                                                                    1();
                                            Is N multiple of nt? No?
                                                                                     ads();
                             Р3
BS
                                   Let's give more rows to the last thread!
                                                   for(i=i_start; i<i_end; i++)</pre>
                                                     for (j=0; j<N; j++)
   nt = number of threads
                                                        sum+= f(Matrix_in[i][j])
   BS = number of rows in a block equal to N/nt
```

```
// Parallel version using omp for
                                       sum = 0;
                                       #pragma omp parallel
                                       #pragma omp for private(j) schedule(static) reduction(+:sum)
                 Ν
                                       for(i=0; i<N; i++)
                                          for (j=0; j<N; j++)
                             P0
                                             sum+= f(Matrix_in[i][j])
                             i start=1*BS
                                                // Parallel version without omp for
                             Ρ1
                                                sum = 0;
                             i end = i start+BS
   Ν
                                                #pragma omp parallel private(i,j) reduction(+:sum)
                             P2
                             i start=3*BS
                                                   int my_id = omp_get_thread_num();
                                                               = N/omp_get_num_threads();
                                                   int BS
BS
                             Р3
                                                   int i_start = my_id * BS;
                             i end = N
                                                   int i_end = i_start + BS;
something
                                                   if (my_id == omp_get_num_threads()-1)
                                                        i end = N;
    nt = number of threads
                                                   for(i=i_start; i<i_end; i++)
    BS = number of rows in a block equal to N/nt
                                                     for (j=0; j<N; j++)
                                                        sum+= f(Matrix_in[i][j])
                                                }
```

```
// Parallel version using omp for
                                       sum = 0;
                                      #pragma omp parallel
                                      #pragma omp for private(j) schedule(static) reduction(+:sum)
                Ν
                                      for(i=0; i<N; i++)
                                          for (j=0; j<N; j++)
                            P0
                                             sum+= f(Matrix_in[i][j])
                            i start=1*BS
                                               // Parallel version without omp for
                            Ρ1
                                               sum = 0;
                            i end = i start+BS
   Ν
                                               #pragma omp parallel private(i,j) reduction(+:sum)
                            P2
                            i_start=3*BS
                                                                                   ads();
BS
                            Р3
                                      Can we maximize load balance?
                            i end:
                                    Maximum one row more per thread?
something
                                                                                   5()-1)
                                                       I_ena - N;
   nt = number of threads
                                                  for(i=i_start; i<i_end; i++)
    BS = number of rows in a block equal to N/nt
                                                    for (j=0; j<N; j++)
                                                       sum+= f(Matrix_in[i][j])
```

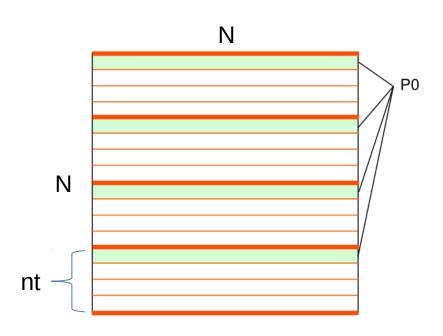
```
// Parallel version using omp for
                                       sum = 0;
                                       #pragma omp parallel
                                       #pragma omp for private(j) schedule(static) reduction(+:sum)
                Ν
                                       for(i=0; i<N; i++)
                                          for (j=0; j<N; j++)
                             P0
                                              sum+= f(Matrix_in[i][j])
                             i start=1*BS
                                                // Parallel version without omp for
                             Ρ1
                                                sum = 0;
                             i end = i start+BS
   Ν
                                                #pragma omp parallel private(i,j) reduction(+:sum)
                             P2
                            i_start=3*BS
                                                                          thread num();
                                                                                    ads();
BS
                             Р3
                                     Assume ... something is equal to 2
                             i end :
something=2
                                                                                     ()-1)
                                                        I_ena - N;
   nt = number of threads
                                                   for(i=i_start; i<i_end; i++)
   BS = number of rows in a block equal to N/nt
                                                     for (j=0; j<N; j++)
                                                        sum+= f(Matrix_in[i][j])
```



nt = number of threads BS = number of rows in a block equal to N/nt

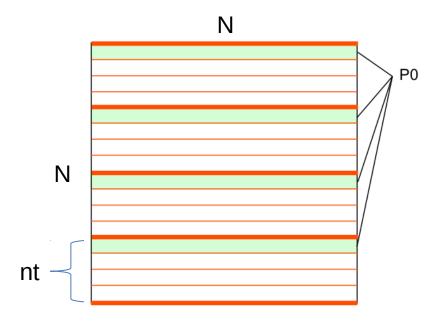
```
// Parallel version without omp for
sum = 0;
#pragma omp parallel private(i,j) reduction(+:sum)
                 = omp_get_thread_num();
   int my_id
   int nt
                 = omp_get_num_threads();
                 = N/nt;
   int BS
   int something = N%nt;
   int i_start
                 = my id * BS;
   if (my_id <something)</pre>
       i_start = i_start + my_id;
   else
       i_start = i_start + something;
   int i end
                 = i start + BS;
   if (my_id <something)</pre>
       i end = i end+1;
   for(i=i_start; i<i_end; i++)</pre>
     for (j=0; j<N; j++)
        sum+= f(Matrix_in[i][j])
```

Cyclic geometric data decomposition by rows of input data Matrix_in. Variable sum replicated



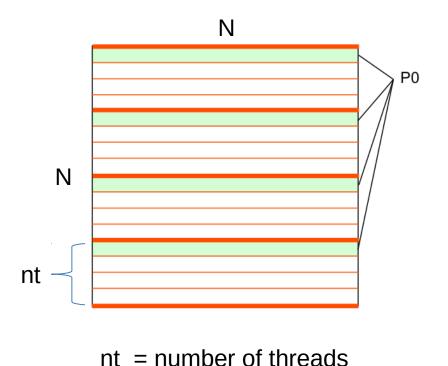
nt = number of threads

```
// Sequential
Sum = 0;
for(i=0; i<N; i++)
    for (j=0; j<N; j++)
        sum+= f(Matrix_in[i][j])</pre>
```

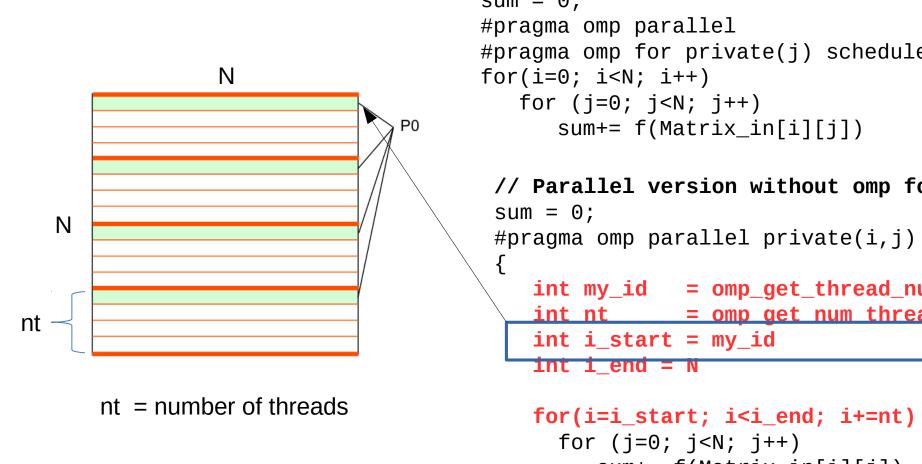


nt = number of threads

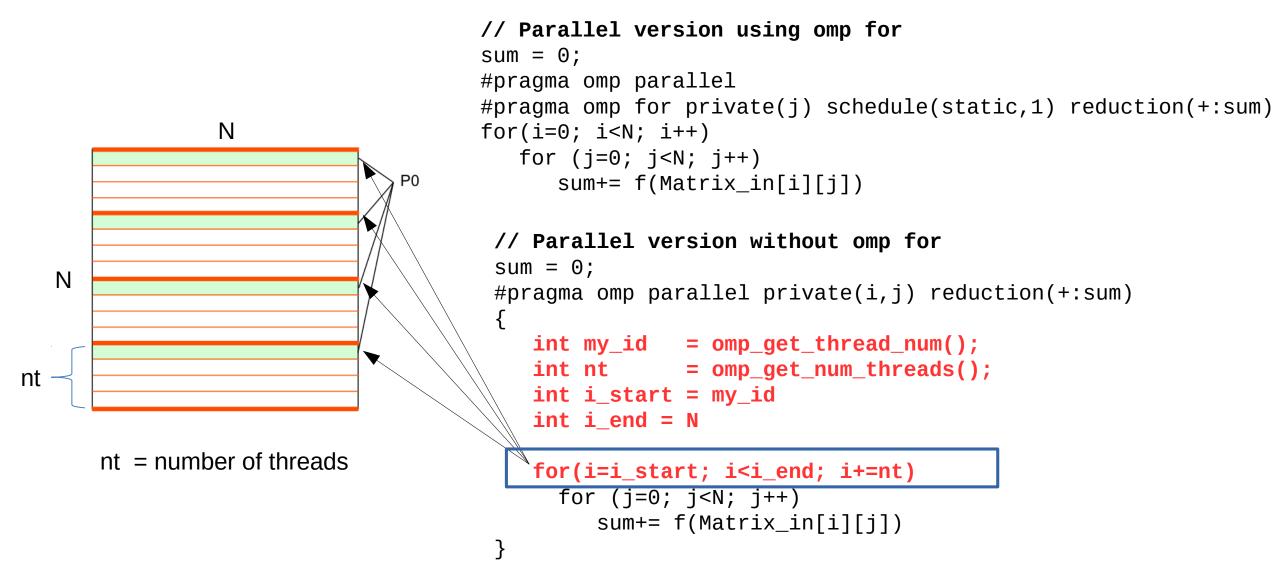
```
// Parallel version using omp for
sum = 0;
#pragma omp parallel
#pragma omp for private(j) schedule(static,1) reduction(+:sum)
for(i=0; i<N; i++)
    for (j=0; j<N; j++)
        sum+= f(Matrix_in[i][j])</pre>
```



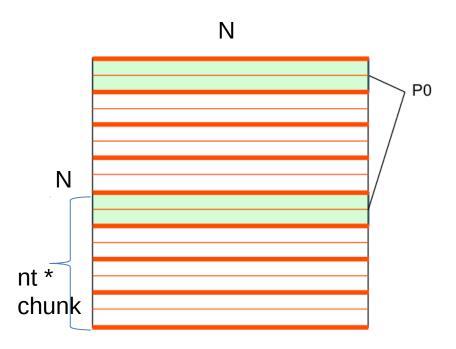
```
// Parallel version using omp for
sum = 0;
#pragma omp parallel
#pragma omp for private(j) schedule(static,1) reduction(+:sum)
for(i=0; i<N; i++)
   for (j=0; j<N; j++)
      sum+= f(Matrix_in[i][j])
 // Parallel version without omp for
 sum = 0;
 #pragma omp parallel private(i,j) reduction(+:sum)
    int i_start = ?
    int i_end = ?
    for(i=i_start; i<i_end; i+= ?)</pre>
      for (j=0; j<N; j++)
         sum+= f(Matrix_in[i][j])
```



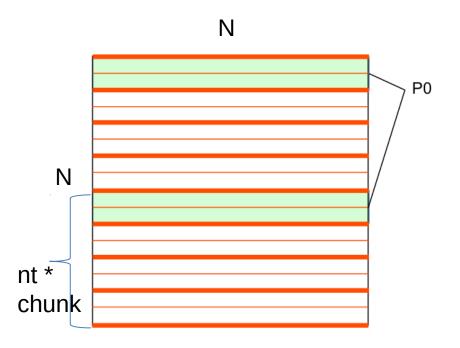
```
// Parallel version using omp for
sum = 0;
#pragma omp for private(j) schedule(static,1) reduction(+:sum)
 // Parallel version without omp for
 #pragma omp parallel private(i,j) reduction(+:sum)
    int my_id = omp_get_thread_num();
               = omp get num threads():
         sum+= f(Matrix_in[i][j])
```



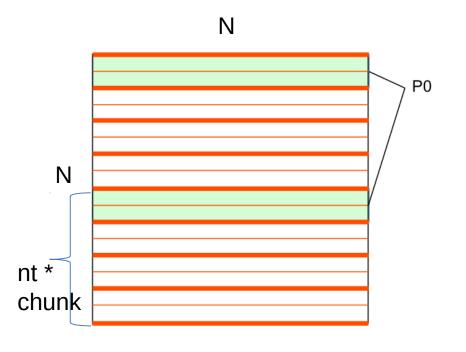
Block-Cyclic geometric data decomposition by rows of input data Matrix_in. Variable sum replicated



```
// Sequential
Sum = 0;
for(i=0; i<N; i++)
    for (j=0; j<N; j++)
        sum+= f(Matrix_in[i][j])</pre>
```



```
// Parallel version using omp for
sum = 0;
#pragma omp parallel
#pragma omp for private(j) schedule(static,2) reduction(+:sum)
for(i=0; i<N; i++)
    for (j=0; j<N; j++)
        sum+= f(Matrix_in[i][j])</pre>
```



```
// Parallel version using omp for
sum = 0;
#pragma omp parallel
#pragma omp for private(j) schedule(static,2) reduction(+:sum)
for(i=0; i<N; i++)
   for (j=0; j<N; j++)
      sum+= f(Matrix_in[i][j])
// Parallel version without omp for
sum = 0;
#pragma omp parallel private(ii,i,j) reduction(+:sum)
   int i_start = ?;
   int i_end = ?;
   for(ii=i_start; ii<i_end; ii+=?)</pre>
   for (?)
     for (j=0; j<N; j++)
        sum+= f(Matrix_in[i][j])
```

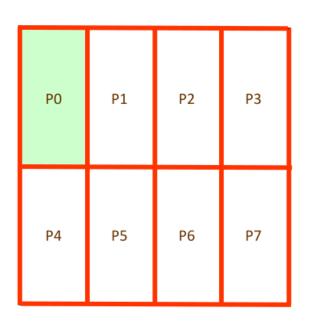
```
// Parallel version using omp for
                                     sum = 0;
                                    #pragma omp parallel
                                    #pragma omp for private(j) schedule(static,2) reduction(+:sum)
                Ν
                                     for(i=0; i<N; i++)
                                        for (j=0; j<N; j++)
                                           sum+= f(Matrix_in[i][j])
                               P0
                                    // Parallel version without omp for
                                    sum = 0;
  Ν
                                    #pragma omp parallel private(ii,i,j) reduction(+:sum)
                                       int nt = omp_get_num_threads();
                                       int my id = omp get thread num();
nt *
                                       int i_start = my_id*2;
chunk
                                       int i_end
                                                   = N;
                                       for(ii=i_start; ii<i_end; ii+=nt*2)</pre>
                                        for (i=ii; i<ii+2; i++)
                                         for (j=0; j<N; j++)
                                            sum+= f(Matrix_in[i][j])
```

```
// Parallel version using omp for
                                    sum = 0;
                                    #pragma omp parallel
                                    #pragma omp for private(j) schedule(static,2) reduction(+:sum)
                                    for(i=0; i<N; i++)
                Ν
                                       for (j=0; j<N; j++)
                                           sum+= f(Matrix_in[i][j])
                               P0
                                    // Parallel version without omp for
                                    sum = 0;
  Ν
                                    #pragma omp parallel private(ii,i,j) reduction(+:sum)
                                       int nt = omp_get_num_threads();
                                       int my_id = omp_get_thread_num();
nt *
                                       int i_start = my_id*2;
chunk
                                       int i_end = N;
                                       for(ii=i_start; ii<i_end; ii+=nt*2)</pre>
                                        for (i=ii; i<ii+2; i++)
                                         for (j=0; j<N; j++)
                                            sum+= f(Matrix_in[i][j])
```

```
// Parallel version using omp for
                                     sum = 0;
                                     #pragma omp parallel
                                     #pragma omp for private(j) schedule(static,2) reduction(+:sum)
                                     for(i=0; i<N; i++)
                Ν
                                        for (j=0; j<N; j++)
                                           sum+= f(Matrix_in[i][j])
                               P0
                                    // Parallel version without omp for
                                    sum = 0;
   Ν
                                    #pragma omp parallel private(ii,i,j) reduction(+:sum)
                                       int nt = omp_get_num_threads();
                                       int my_id = omp_get_thread_num();
nt *
                                       int i_start = my_id*2;
chunk
                                       int i_end = N;
                                        for(ii=i start; ii<i end; ii+=nt*2)</pre>
                                        for (i=ii; i<ii+2; i++)
                                         for (j=0; j<N; j++)
                                            sum+= f(Matrix_in[i][j])
```

Geometric data decomposition by BLOCKS of N/2 rows and N/4 columns (BSi x BSj)

```
// Parallel version using omp for
sum = 0;
#pragma omp parallel
#pragma omp for private(j) schedule(static,2) reduction(+:sum)
for(i=0; i<N; i++)
    for (j=0; j<N; j++)
        sum+= f(Matrix_in[i][j])</pre>
```



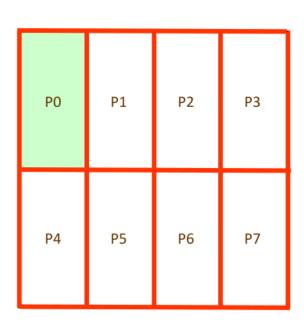
```
// Parallel version without omp for
sum = 0;
#pragma omp parallel private(ii,i,j) reduction(+:sum) num_threads(8)
   int my_id
   int block_i = my_id/4; // 0 or 1 indicates which row of blocks
  int block_j = my_id%4; // 0,1,2,3 indicates which column of blocks
  int BSi = N/2; // Assume N is multiple of 2
  int BSj = N/4; // Assume N is multiple of 4
   int i_start = block_i * Bsi; // where should thread start?
   int i_end = i_start + BSi;
   int j_start = block_j * BSj;
   for(i=i_start; i<i_end; i++)</pre>
     for (j=j_start; j<j_end; j++)</pre>
        sum+= f(Matrix_in[i][j])
```

Geometric data decomposition by BLOCKS of N/2 rows and N/4 columns (BSi x BSj)

```
For the case of
                                     // Parallel version using omp for
 my id=1
                                     sum = 0;
 block_i=my_id/4=0
                                     #pragma omp parallel
 block_j=my_id%4=1
                                     #pragma omp for private(j) schedule(static,2) reduction(+:sum)
                                     for(i=0; i<N; i++)
          block_j
                                        for (j=0; j<N; j++)
                                           sum+= f(Matrix_in[i][j])
block_i
                              // Parallel version without omp for
                              sum = 0;
                              #pragma omp parallel private(ii,i,j) reduction(+:sum) num_threads(8)
         P0
              P1
                                 int my_id
                                 int block_i = my_id/4; // 0 or 1 indicates which row of blocks
                                 int block_j = my_id%4; // 0,1,2,3 indicates which column of blocks
                                 int BSi = N/2; // Assume N is multiple of 2
         P4
              P5
                                 int BSj = N/4; // Assume N is multiple of 4
                                 int i_start = block_i * Bsi; // where should thread start?
                                 int i_end = i_start + BSi;
                                 int j_start = block_j * BSj;
                                 for(i=i_start; i<i_end; i++)</pre>
                                   for (j=j_start; j<j_end; j++)</pre>
                                      sum+= f(Matrix_in[i][j])
```

Geometric data decomposition by BLOCKS of N/2 rows and N/4 columns (BSi \times BSj)

```
For the case of my_id=1 block_i=my_id/4=0 block_j=my_id%4=1
```



```
// Parallel version using omp for
sum = 0;
#pragma omp parallel
#pragma omp for private(j) schedule(static,2) reduction(+:sum)
for(i=0; i<N; i++)
    for (j=0; j<N; j++)
        sum+= f(Matrix_in[i][j])</pre>
```

```
// Parallel version without omp for
sum = 0;
#pragma omp parallel private(ii,i,j) reduction(+:sum) num_threads(2x4)
   int my_id
   int block_i = my_id/4; // 0 or 1 indicates which row of blocks
   int block j = mv id%4; \frac{1}{100} 0,1,2,3 indicates which column of blocks
                          // Assume N is multiple of 2
   int BSi = N/2;
   int BSj = N/4;
                          // Assume N is multiple of 4
   int i start = block i * Bsi; // where should thread start?
   int i_end = i_start + BSi;
   int j_start = block_j * BSj;
   for(i=i_start; i<i_end; i++)</pre>
     for (j=j_start; j<j_end; j++)</pre>
        sum+= f(Matrix_in[i][j])
```

Geometric data decomposition by BLOCKS of N/2 rows and N/4 columns (BSi x BSj)

```
For the case of my_id=1 block_i=my_id/4=0 block_j=my_id%4=1
```

```
P0 P1 P2 P3

P4 P5 P6 P7
```

```
// Parallel version using omp for
sum = 0;
#pragma omp parallel
#pragma omp for private(j) schedule(static,2) reduction(+:sum)
for(i=0; i<N; i++)
    for (j=0; j<N; j++)
        sum+= f(Matrix_in[i][j])</pre>
```

```
// Parallel version without omp for
sum = 0;
#pragma omp parallel private(ii,i,j) reduction(+:sum) num_threads(8)
  int my_id
   int block_i = my_id/4; // 0 or 1 indicates which row of blocks
   int block_j = my_id%4; // 0,1,2,3 indicates which column of blocks
   int BSi = N/2; // Assume N is multiple of 2
   int BSj = N/4;  // Assume N is multiple of 4
   int i start = block i * Bsi; // where should thread start?
   int i_end = i_start + BSi;
   int j_start = block_j * BSj;
   for(i=i_start; i<i_end; i++)</pre>
     for (j=j_start; j<j_end; j++)</pre>
        sum+= f(Matrix in[i][i])
```