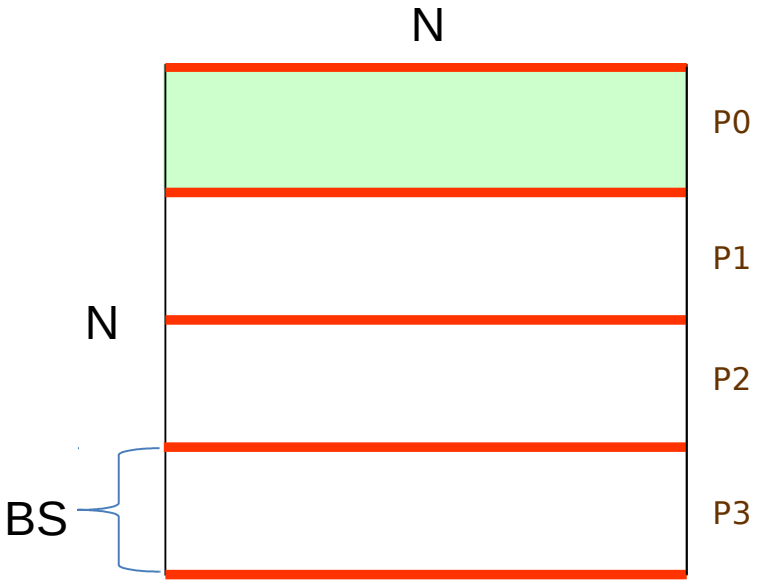
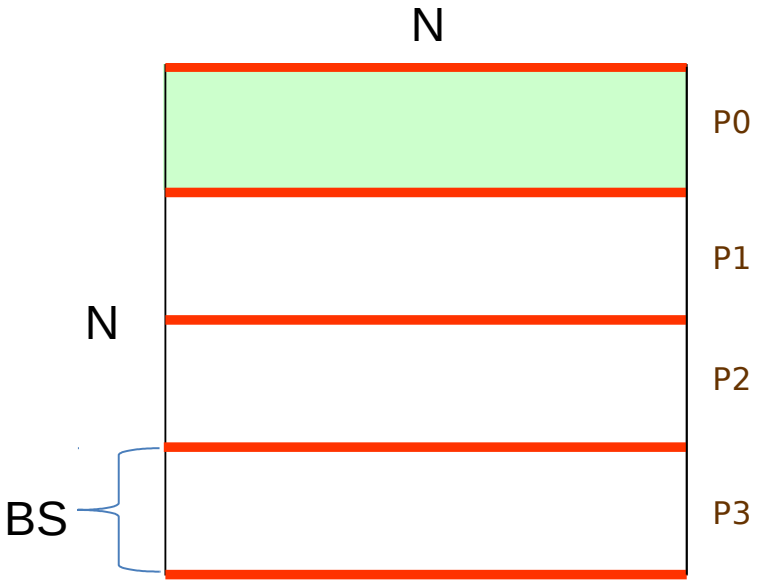
# Block geometric data decomposition by rows of input data Matrix_in. Variable sum replicated




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

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

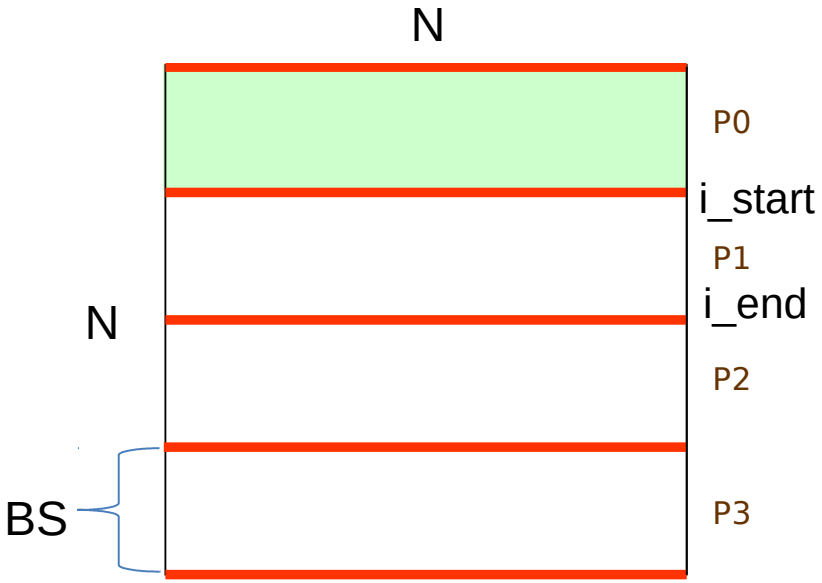
# Block geometric data decomposition by rows




```
// 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

# Block geometric data decomposition by rows

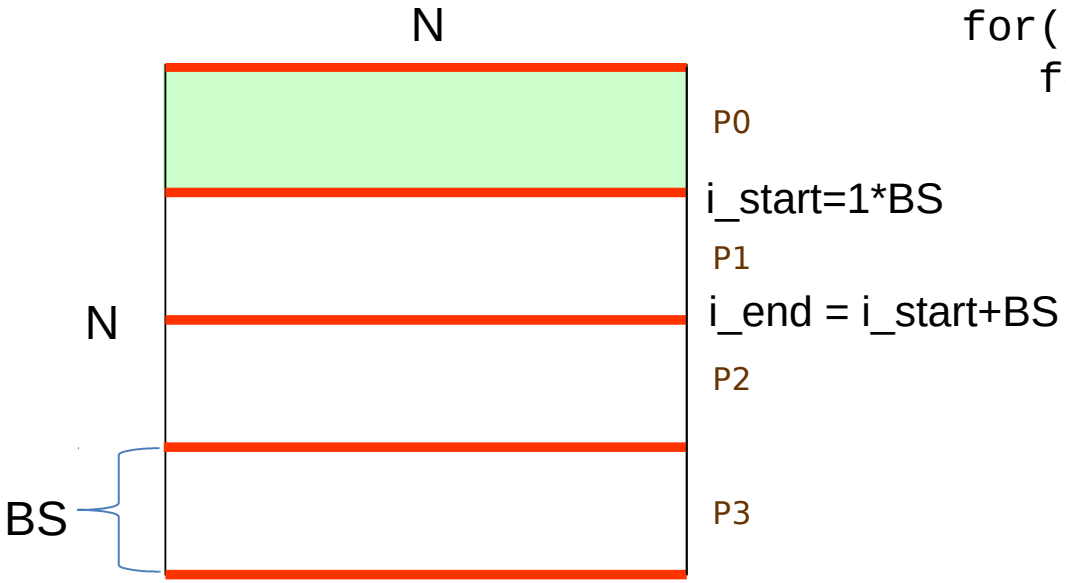



```
// Parallel version using omp for
sum = 0;
#pragma omp parallel
#pragma omp for private(j) schedule(static) reduction(+:sum)
for(i=0; i<N; i++)
   for (j=0; j<N; j++)
      sum+= f(Matrix_in[i][j])

// 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++)
     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

# Block geometric data decomposition by rows

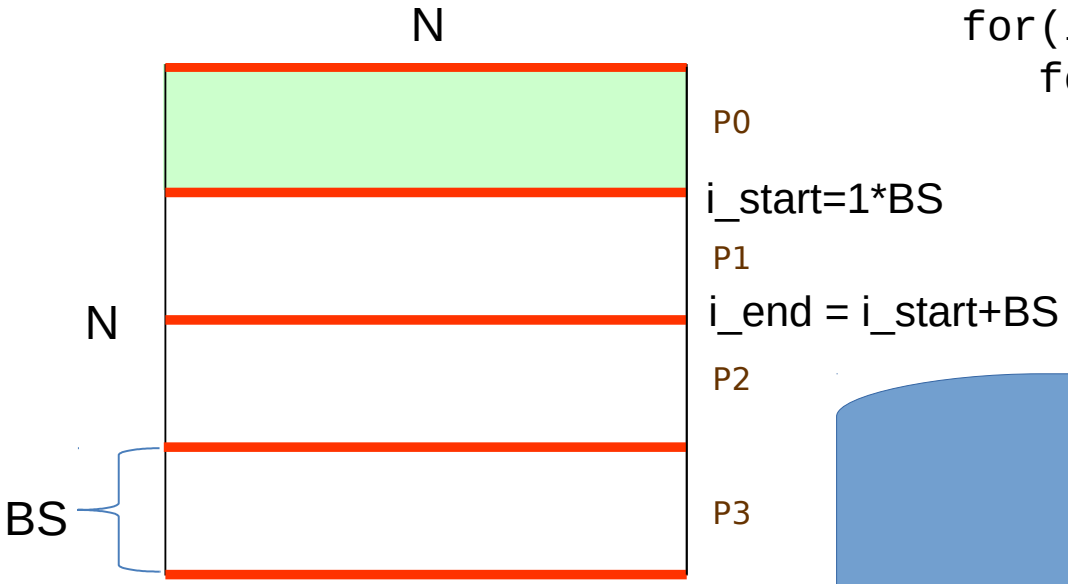



nt = number of threads
BS = number of rows in a block equal to N/nt

```
// 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 my_id   = omp_get_thread_num();
   int BS      = N/omp_get_num_threads();
   int i_start = my_id * BS;
   int i_end   = i_start + BS;

   for(i=i_start; i<i_end; i++)
     for (j=0; j<N; j++)
        sum+= f(Matrix_in[i][j])
}
```

# Block geometric data decomposition by rows




```
// 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)
                                                    n();
                                                   eads();
```

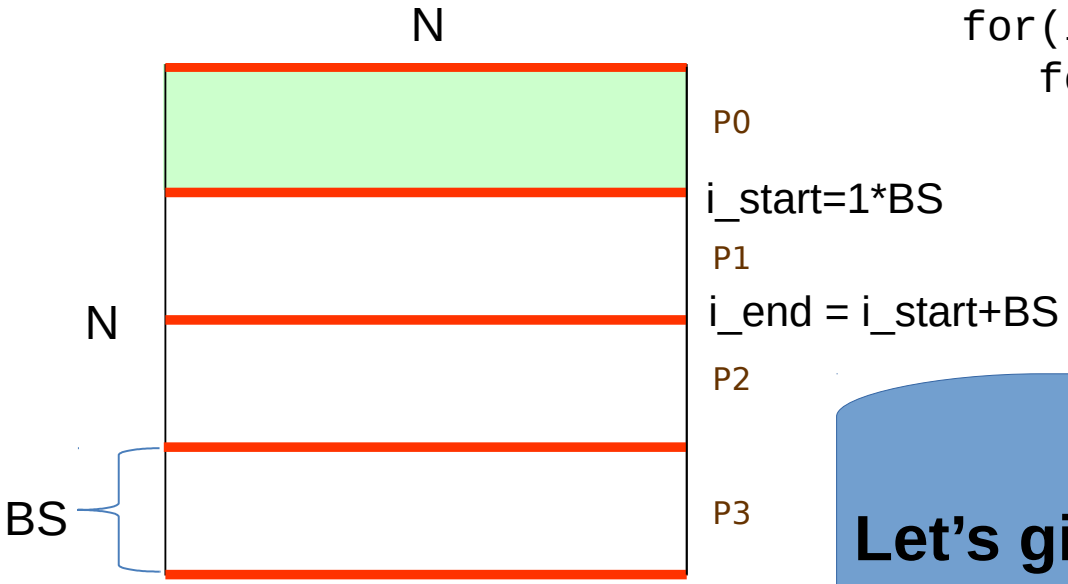
**Is N multiple of nt?**

```
   for(i=i_start; i<i_end; 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

# Block geometric data decomposition by rows




```
// 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])
```

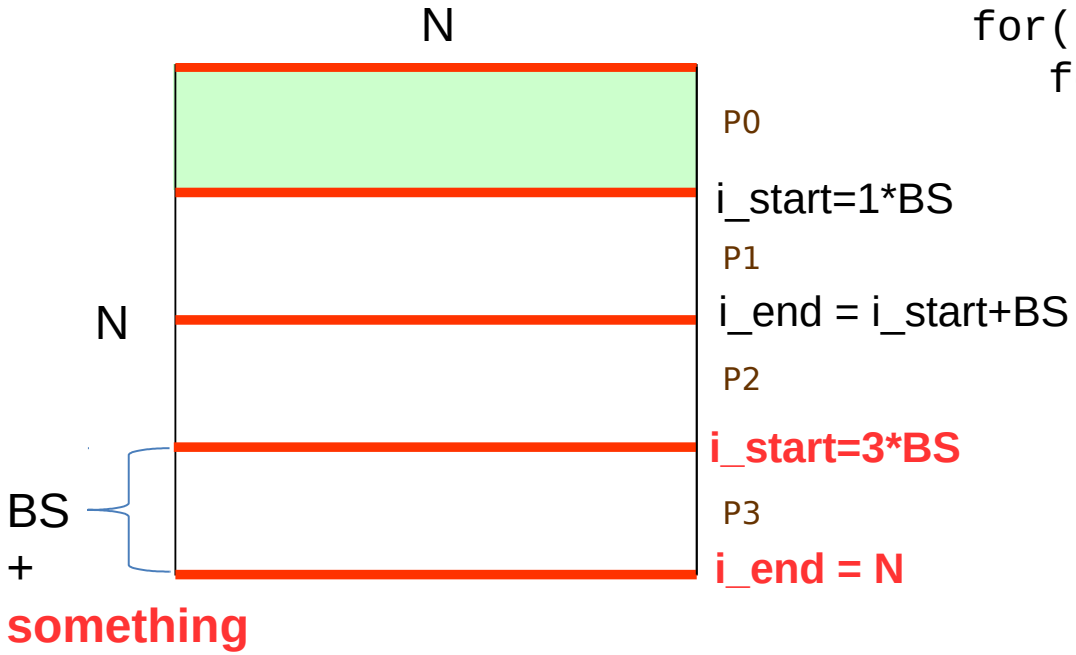
```
// Parallel version without omp for
sum = 0;
#pragma omp parallel private(i,j) reduction(+:sum)
                                                    n();
                                                   eads();
```

**Is N multiple of nt? No?**
**Let's give more rows to the last thread!**

```
for(i=i_start; i<i_end; 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

## Block geometric data decomposition by rows

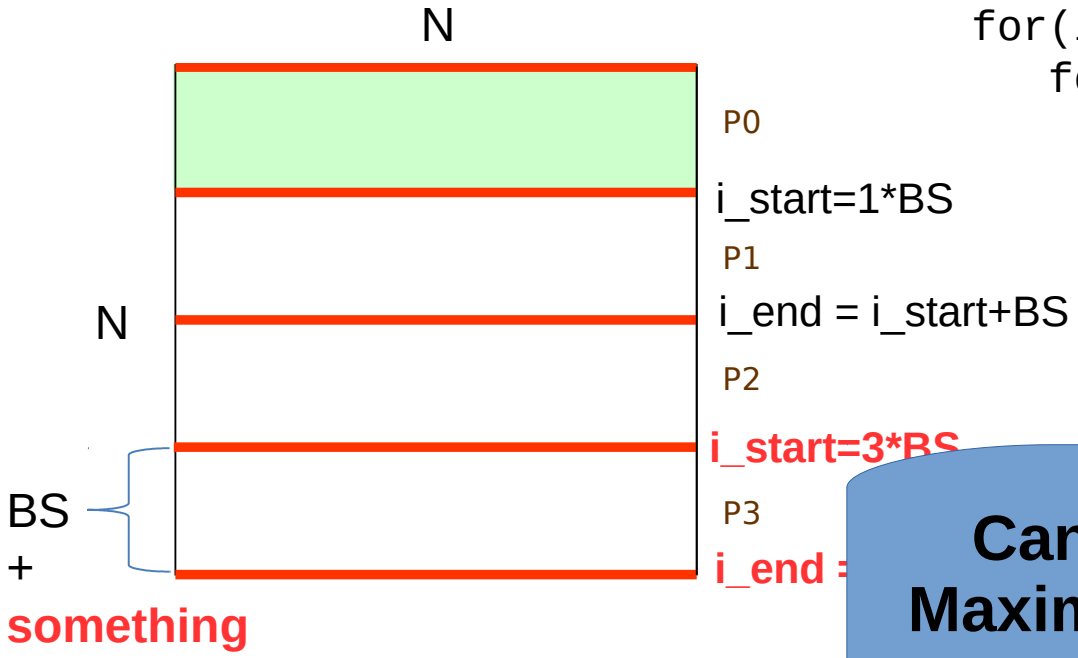



nt = number of threads
BS = number of rows in a block equal to N/nt

```
// 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 my_id   = omp_get_thread_num();
   int BS      = N/omp_get_num_threads();
   int i_start = my_id * BS;
   int i_end   = i_start + BS;
   if (my_id == omp_get_num_threads()-1)
       i_end = N;

   for(i=i_start; i<i_end; i++)
     for (j=0; j<N; j++)
        sum+= f(Matrix_in[i][j])
}
```

## Block geometric data decomposition by rows



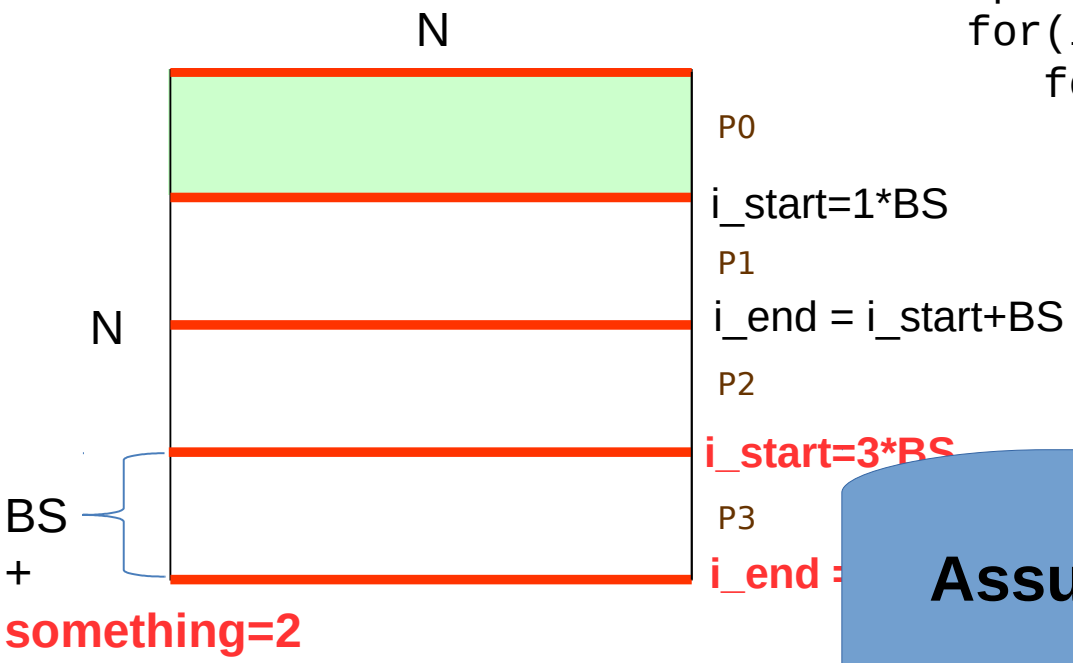

```
// Parallel version using omp for
sum = 0;
#pragma omp parallel
#pragma omp for private(j) schedule(static) reduction(+:sum)
for(i=0; i<N; i++)
   for (j=0; j<N; j++)
      sum+= f(Matrix_in[i][j])
```

```
// Parallel version without omp for
sum = 0;
#pragma omp parallel private(i,j) reduction(+:sum)
{
   int my_id, = omp_get_thread_num();
                                    eads();
                                    s()-1)
   i_end = N;

   for(i=i_start; i<i_end; i++)
     for (j=0; j<N; j++)
        sum+= f(Matrix_in[i][j])
}
```

**Can we maximize load balance?**
**Maximum one row more per thread?**

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

## Block geometric data decomposition by rows



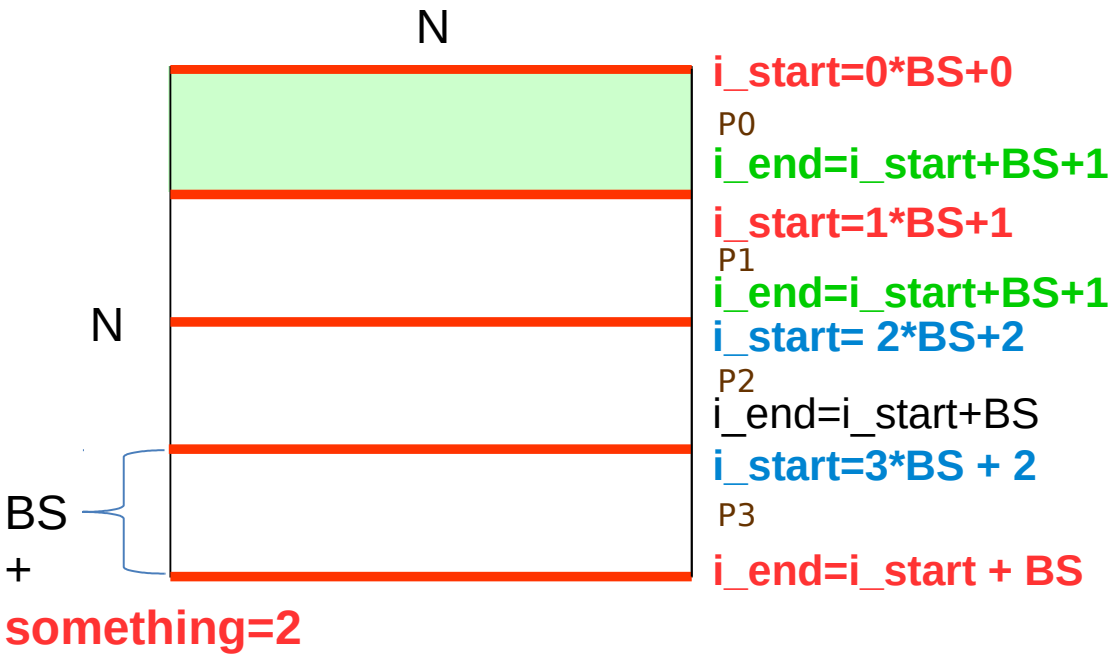

```
// Parallel version using omp for
sum = 0;
#pragma omp parallel
#pragma omp for private(j) schedule(static) reduction(+:sum)
for(i=0; i<N; i++)
   for (j=0; j<N; j++)
      sum+= f(Matrix_in[i][j])
```

```
// Parallel version without omp for
sum = 0;
#pragma omp parallel private(i,j) reduction(+:sum)
{
   int my_id, = omp_get_thread_num();
                                    eads();
                                    s()-1)
   i_end = N;

   for(i=i_start; i<i_end; i++)
     for (j=0; j<N; j++)
        sum+= f(Matrix_in[i][j])
}
```

**Assume ... something is equal to 2**

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

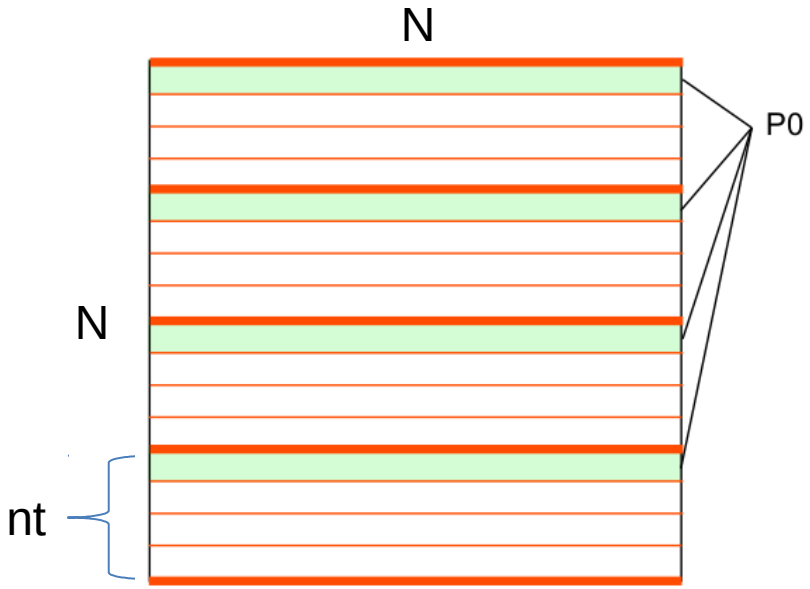
## Block geometric data decomposition by rows




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)
{
    int my_id     = omp_get_thread_num();
    int nt        = omp_get_num_threads();
    int BS        = N/nt;
    int something = N%nt;
    int i_start   = my_id * BS;
    if (my_id <something)
        i_start = i_start + my_id;
    else
        i_start = i_start + something;
    int i_end     = i_start + BS;
    if (my_id <something)
        i_end = i_end+1;
    for(i=i_start; i<i_end; i++)
      for (j=0; j<N; j++)
          sum+= f(Matrix_in[i][j])
}
```

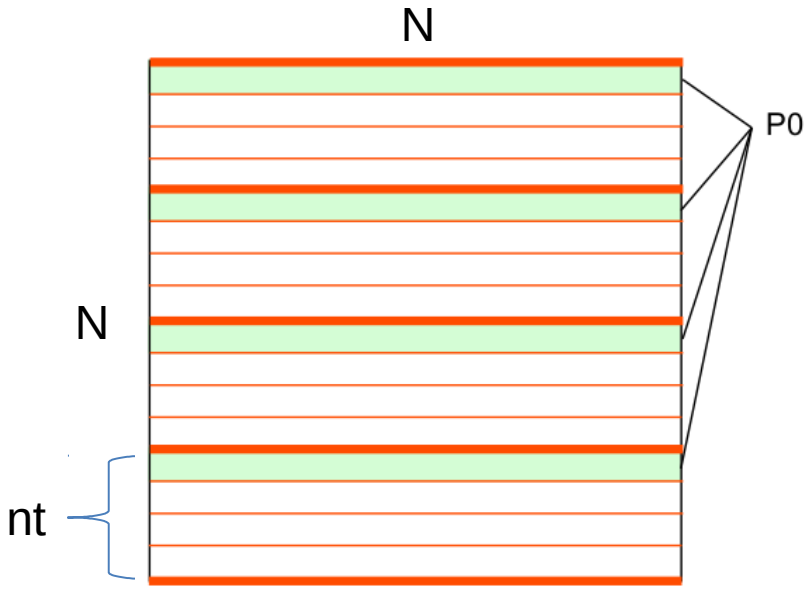
# Cyclic geometric data decomposition by rows of input data Matrix_in. Variable sum replicated




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])
```

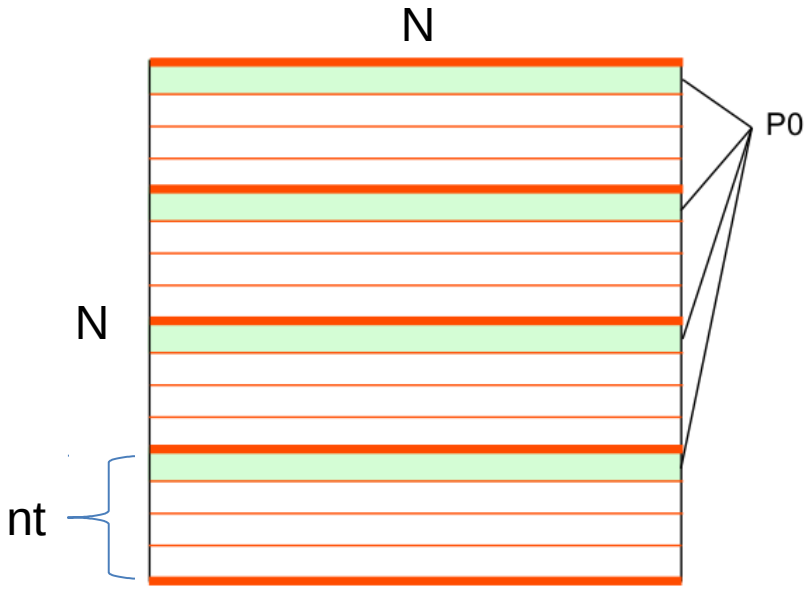
# Cyclic geometric data decomposition by rows




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])
```

# Cyclic geometric data decomposition by rows

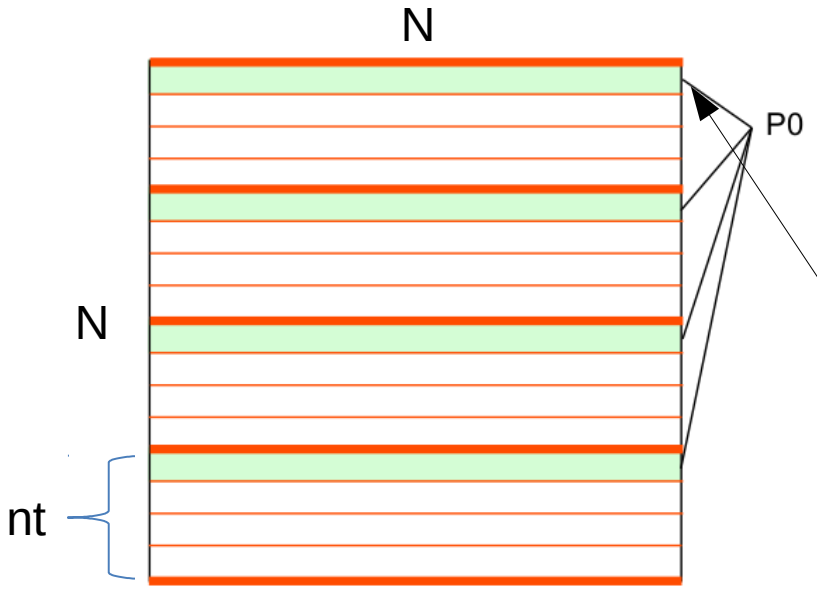



nt = number of threads

```
// 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])
```

```
// 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+= ?)
     for (j=0; j<N; j++)
        sum+= f(Matrix_in[i][j])
}
```

# Cyclic geometric data decomposition by rows

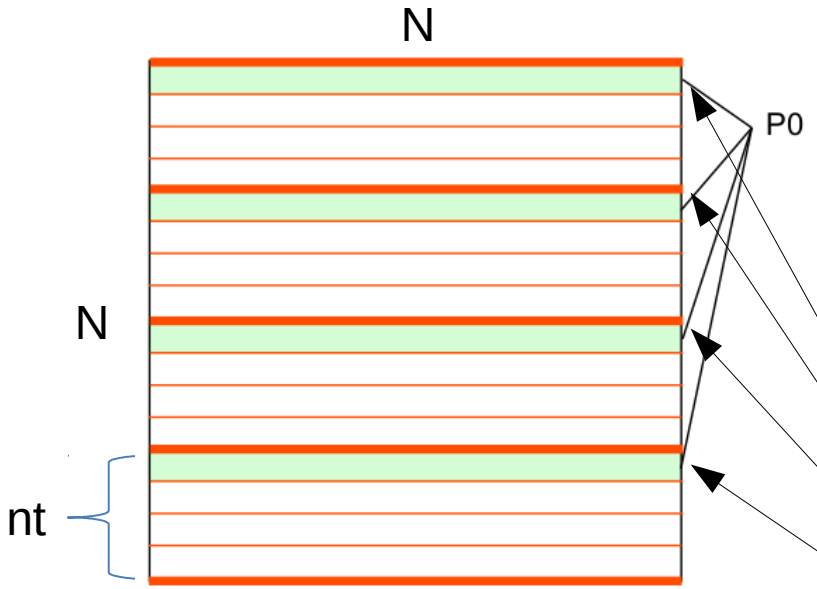



nt = number of threads

```
// 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])
```

```
// Parallel version without omp for
sum = 0;
#pragma omp parallel private(i,j) reduction(+:sum)
{
   int my_id   = omp_get_thread_num();
   int nt      = omp_get_num_threads();
   int i_start = my_id
   int i_end = N

   for(i=i_start; i<i_end; i+=nt)
     for (j=0; j<N; j++)
        sum+= f(Matrix_in[i][j])
}
```

# Cyclic geometric data decomposition by rows

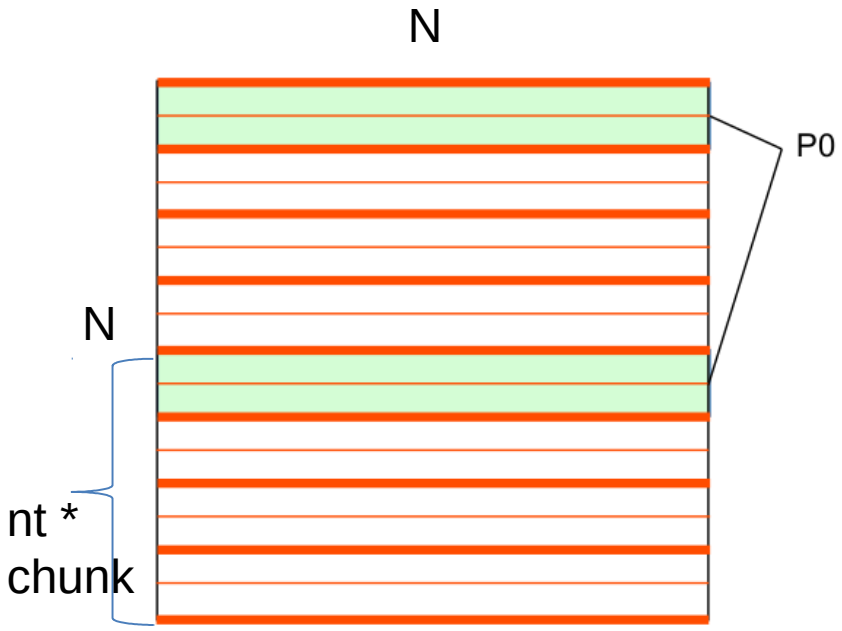



nt = number of threads

```
// 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])
```

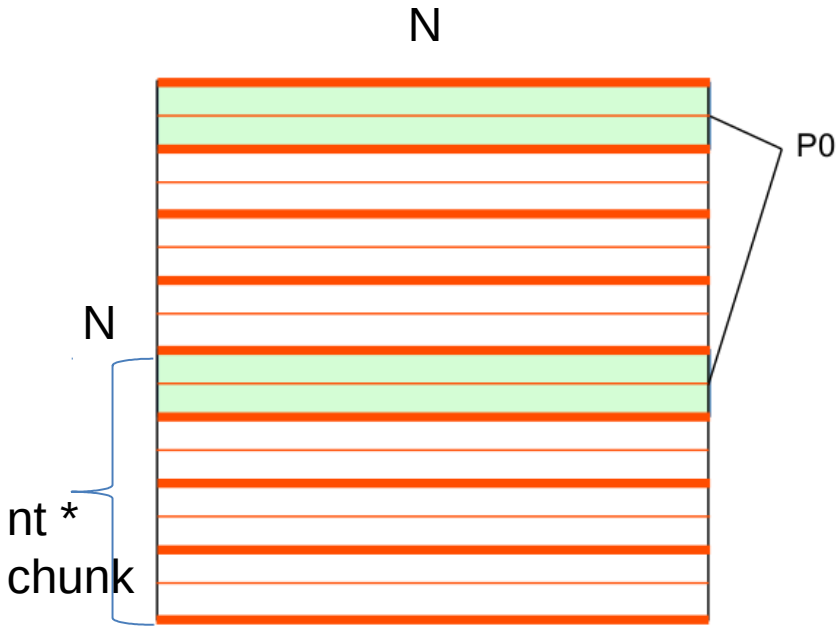
```
// Parallel version without omp for
sum = 0;
#pragma omp parallel private(i,j) reduction(+:sum)
{
   int my_id   = omp_get_thread_num();
   int nt      = omp_get_num_threads();
   int i_start = my_id
   int i_end = N

   for(i=i_start; i<i_end; i+=nt)
      for (j=0; j<N; j++)
         sum+= f(Matrix_in[i][j])
}
```

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



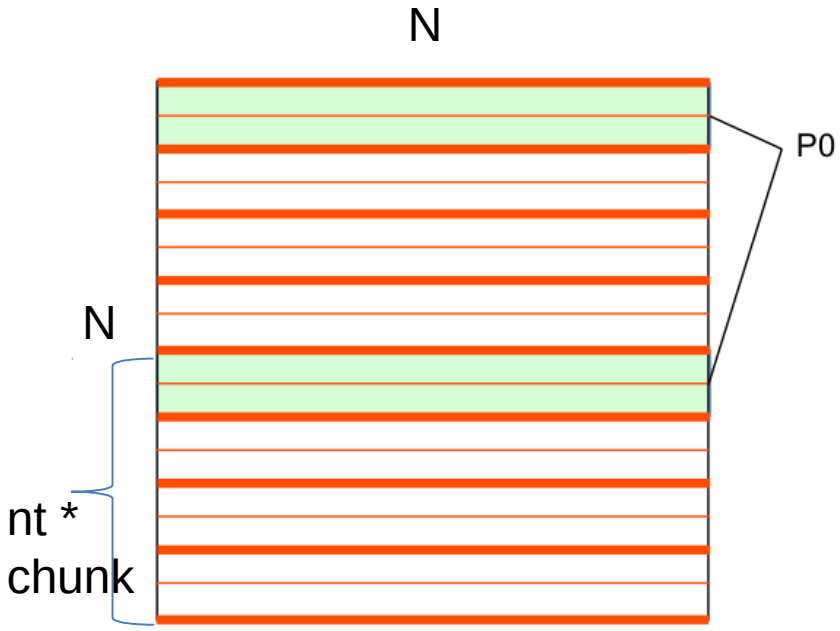

```
// Sequential
Sum = 0;
for(i=0; i<N; i++)
    for (j=0; j<N; j++)
        sum+= f(Matrix_in[i][j])
```

# Block-Cyclic geometric data decomposition by rows




```
// 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])
```

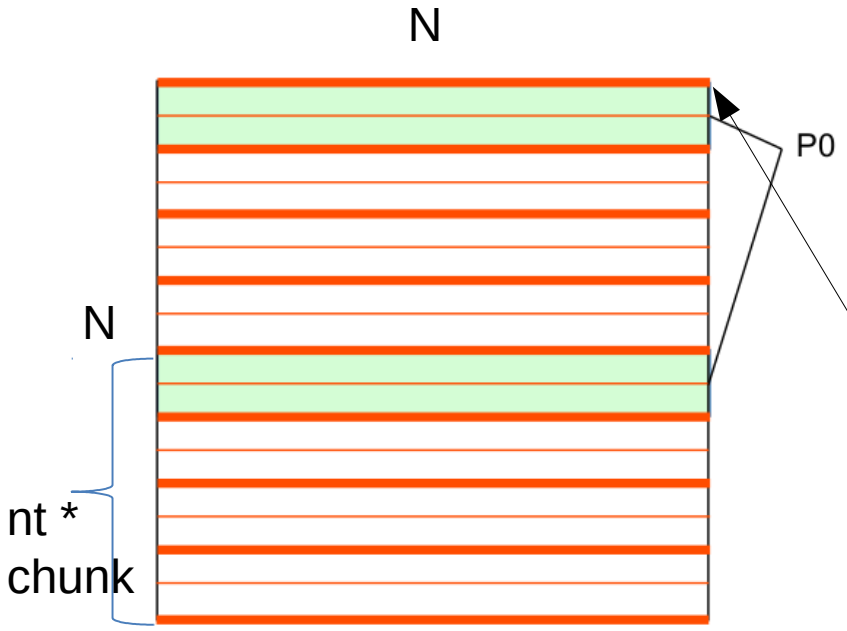
# Block-Cyclic geometric data decomposition by rows




```
// 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+=?)
    for (?)
      for (j=0; j<N; j++)
         sum+= f(Matrix_in[i][j])
}
```

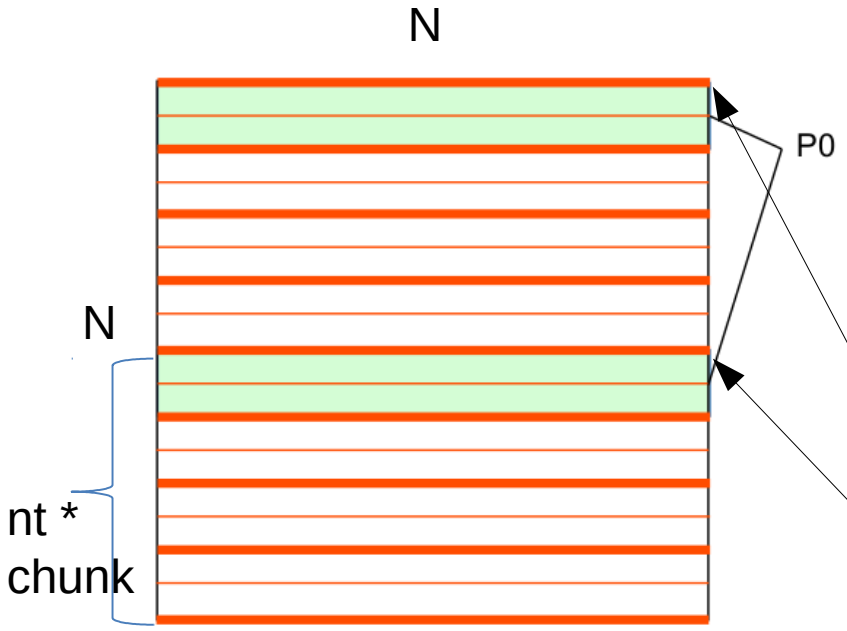
# Block-Cyclic geometric data decomposition by rows




```
// 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 nt      = omp_get_num_threads();
   int my_id   = omp_get_thread_num();
   int i_start = my_id*2;
   int i_end   = N;

   for(ii=i_start; ii<i_end; ii+=nt*2)
    for (i=ii; i<ii+2; i++)
      for (j=0; j<N; j++)
         sum+= f(Matrix_in[i][j])
}
```

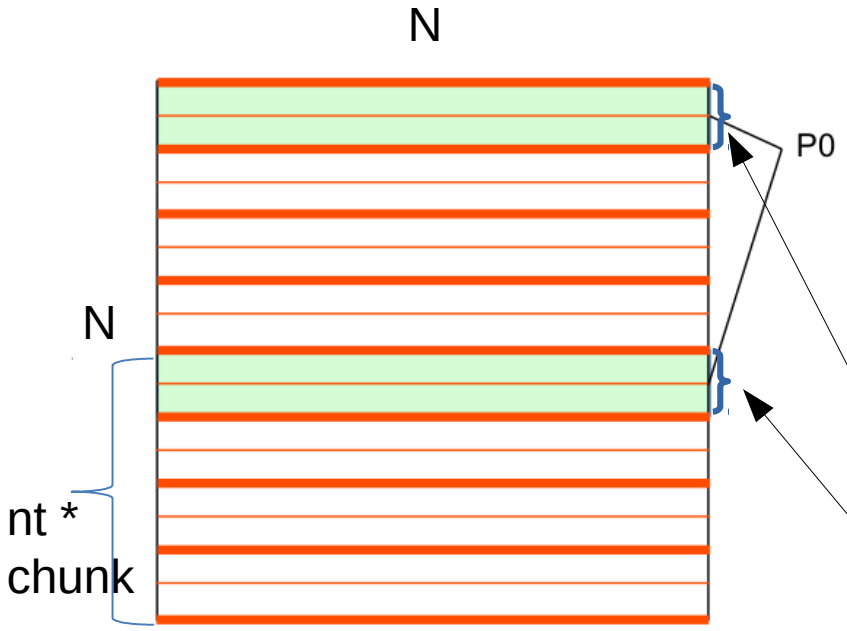
# Block-Cyclic geometric data decomposition by rows




```
// 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 nt      = omp_get_num_threads();
   int my_id   = omp_get_thread_num();
   int i_start = my_id*2;
   int i_end   = N;

   for(ii=i_start; ii<i_end; ii+=nt*2)
    for (i=ii; i<ii+2; i++)
      for (j=0; j<N; j++)
         sum+= f(Matrix_in[i][j])
}
```

# Block-Cyclic geometric data decomposition by rows

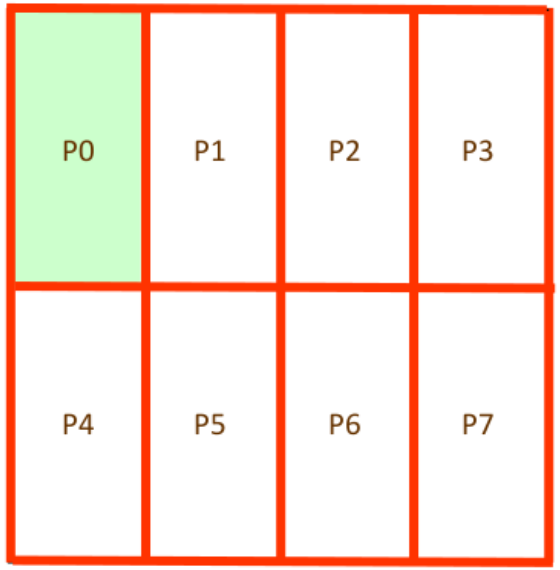



```
// Parallel version using omp for
sum = 0;
#pragma omp parallel
#pragma omp for private(j) schedule(static,2) reduction(+:sum)
for(i=0; i<N; i++)
   for (j=0; j<N; j++)
      sum+= f(Matrix_in[i][j])
```

```
// Parallel version without omp for
sum = 0;
#pragma omp parallel private(ii,i,j) reduction(+:sum)
{
   int nt      = omp_get_num_threads();
   int my_id   = omp_get_thread_num();
   int i_start = my_id*2;
   int i_end   = N;

   for(ii=i_start; ii<i_end; ii+=nt*2)
   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)

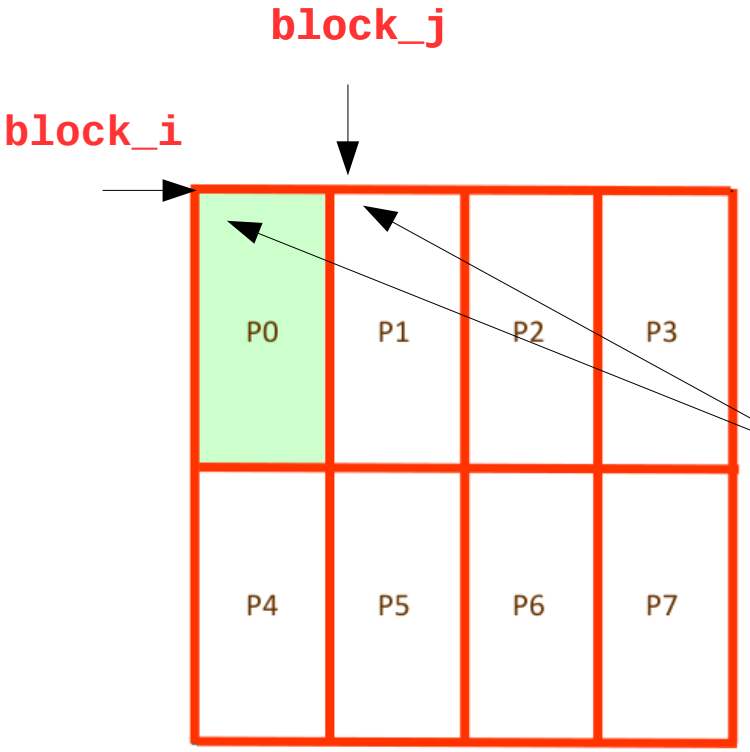
```
// Parallel version using omp for
sum = 0;
#pragma omp parallel
#pragma omp for private(j) schedule(static,2) reduction(+:sum)
for(i=0; i<N; i++)
   for (j=0; j<N; j++)
      sum+= f(Matrix_in[i][j])
```




```
// 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++)
     for (j=j_start; j<j_end; j++)
        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

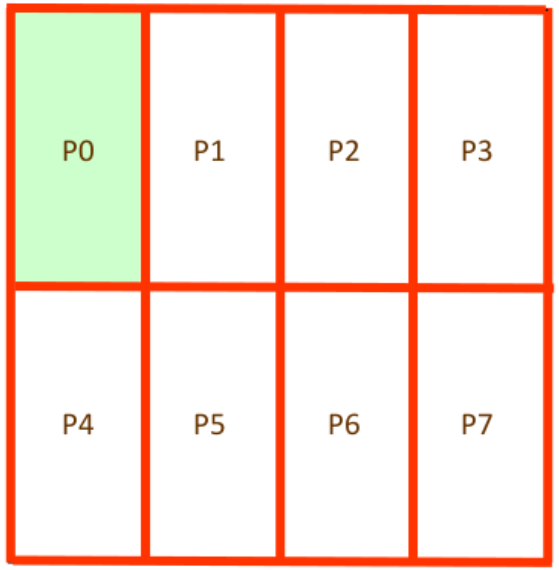



```
// Parallel version using omp for
sum = 0;
#pragma omp parallel
#pragma omp for private(j) schedule(static,2) reduction(+:sum)
for(i=0; i<N; i++)
   for (j=0; j<N; j++)
      sum+= f(Matrix_in[i][j])
```

```
// Parallel version without omp for
sum = 0;
#pragma omp parallel private(ii,i,j) reduction(+:sum) 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++)
     for (j=j_start; j<j_end; j++)
        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



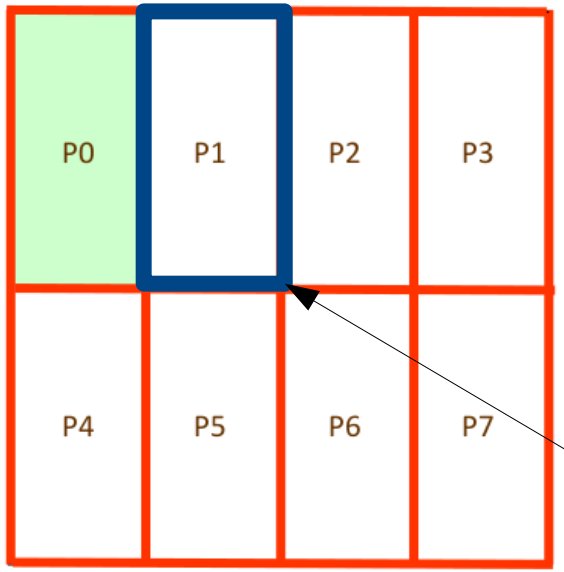

```
// Parallel version using omp for
sum = 0;
#pragma omp parallel
#pragma omp for private(j) schedule(static,2) reduction(+:sum)
for(i=0; i<N; i++)
   for (j=0; j<N; j++)
      sum+= f(Matrix_in[i][j])
```

```
// Parallel version without omp for
sum = 0;
#pragma omp parallel private(ii,i,j) reduction(+:sum) num_threads(2x4)
{
   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++)
     for (j=j_start; j<j_end; j++)
        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

```
// 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) 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++)
     for (j=j_start; j<j_end; j++)
        sum+= f(Matrix_in[i][j])
}
```