Parallelization of Dense Matrix-Vector Multiplications

東京大学情報基盤中心　准教授　片桐孝洋

Takahiro Katagiri, Associate Professor,
Information Technology Center, The University of Tokyo

台大数学科学中心　科学計算冬季学校

Agenda

1. Execute sample program of dense matrix-vector multiplications
2. Note of Parallelization
3. Lesson of Parallelization
4. Lessons and Homework

Execute Sample Program of Parallelization of Dense Matrix-Vector Multiplications

Commands of EMACS

- C- :With Control Key
- M- : With Esc Key
- C-x C-s : Save text
- C-x C-c : Exit
- C-g : Reset mode. In case of that If you are confusing.
- C-k : Delete one line, and the line is stored in a buffer. You can delete multiple lines to memorize the buffer.
- C-y : Copy contents of the above buffer to location of current cursor.
- C-s : Search input character stream, and move to location of that. Move next candidate if you enter "C-s". For debugging, you can use this to input name of functions that you want to search.
- M-x goto-line : Go to line you want. After entering the command, system asks you the number of line.

Note: Sample program of dense matrix-vector multiplications

- File name for C/Fortran languages:

  Mat-vec-fx.tar

- Change queue name from lecture to lecture6 in job script file "mat-vec.bash". Then type "pjsub" to submit the job.
  - lecture : Queue name in out of time of this lecture.
  - lecture6 : Queue name in time of this lecture.

Execute Sample Program of Parallelization of Dense Matrix-Vector Multiplications

- Type the following commands:
  ```
  $ cp /home/z30082/Mat-vec-fx.tar ./
  $ tar xvf Mat-vec-fx.tar
  $ cd Mat-vec
  ```
- Choose the follows:

  $ cd C : For C language.
  $ cd F : For Fortran.
- The follows are common.
  ```
  $ make
  $ pjsub mat-vec.bash
  ```
- After finalizing execution, type the follow:
  ```
  $ cat mat-vec.bash.oXXXXXX
  ```

Output (C Language)

- If it runs successfully, then you see the followings.

```
N  = 10000
Mat-Vec time  = 0.171097 [sec.]
 1168.927027 [MFLOPS]
OK!
```

Output (Fortran Language)

- If it runs successfully, then you see the followings.

```
N = 10000
Mat-Vec time[sec.] = 0.16659261297900023
MFLOPS = 1200.5334205320200
OK!
```

Explanation of the sample program (C Language)

- #define N 10000

  By varying the number, you can change matrix size.

- #define DEBUG 1

  With compiling with "1", you can verify computation error with a test matrix.

- Recompiling can be adapted by:

  % make clean

  % make

Explanation of the sample program (Fortran Language)

- Declaration of size of matrix NN is located in the following file:

  mat-vec.inc

- You can change size of matrix by changing the following NN:

```
integer  NN
parameter (NN=10000)
```

Homework 2

- Parallelize loop in MyMatVec function (procedure).
  - For debugging, you use:
    - #define N 192

    to reduce execution time.

    In addition, you need to specify:
    - #define DEBUG 1

    to verify result.

Note: To do Homework (1/2)

- Start with distributed data for each process.
  No need to implement data distribution.
- The follows are explanation of program for verification:
  - Elements of matrices and vector are set to "1", if you use the verification.
  - If you do not use the verification, the elements are set with random number in this sample program.
    - The program do not support same sequences of random number in each process.
    - If you want to use random matrix, you need to use same sequences of random number to sequential execution.

Note: To do Homework (2/2)

- In this homework,
  we DO NOT NEED MPI communication functions.
- In this sample program,

  parallelization of verification part is not implemented.

Hence, you need an extra parallelization for the verification part, in addition to the part of MatVec function to pass the verification.

- Parallelization of the verification is as same way as part of MatVec part.

Confirmation of MPI Parallelization (Again)

- SPMD
  - Target program (mat-vec.c, mat-vec.f) is:
    - used for all processes, and
    - invocated simultaneously,

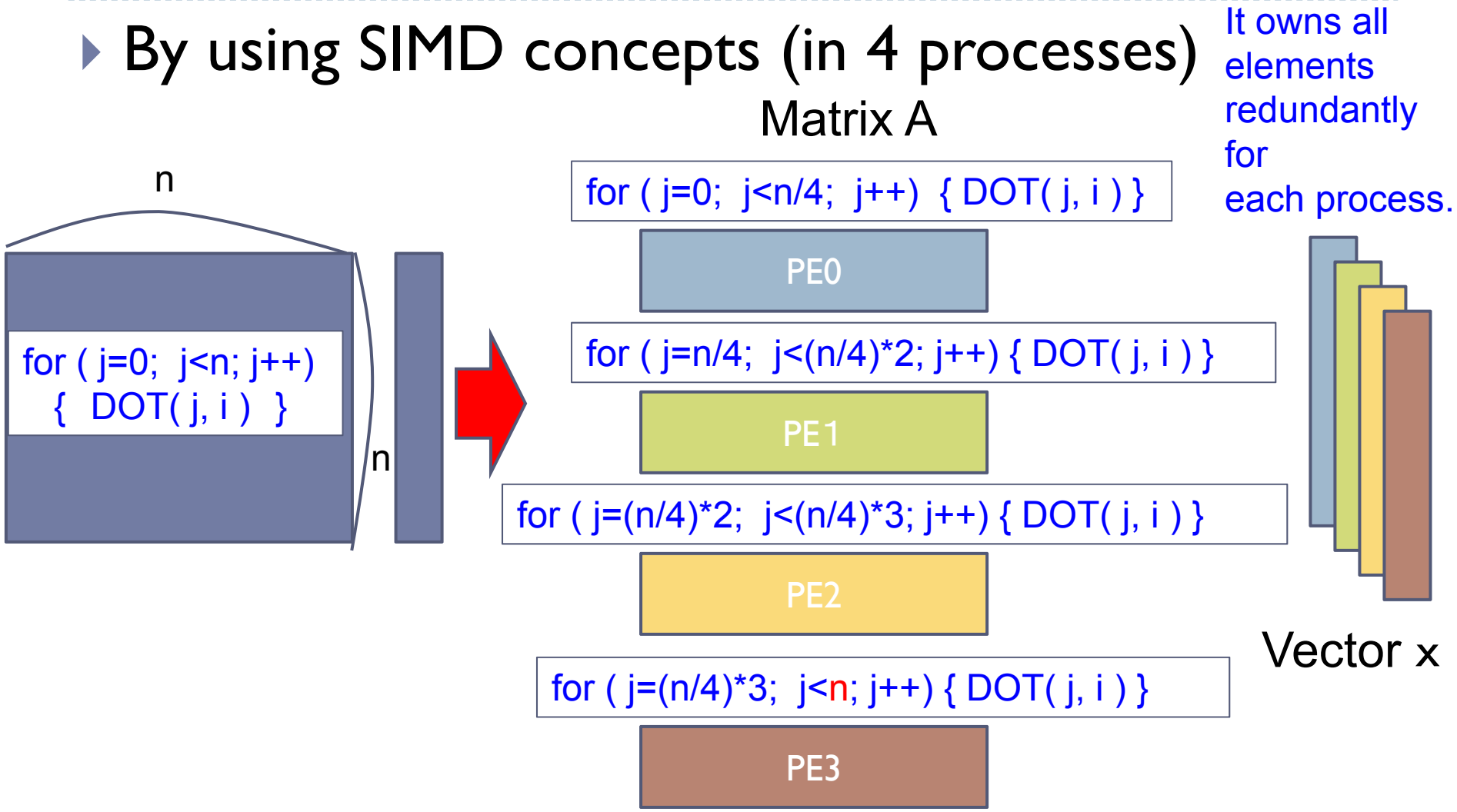
  in time of starting.
- Distributed Memory Parallel Computers
  - In each process, there is an independent memory. This is NOT shared memory.

TIPS for the sample program

- **myid** and **numprocs** are global variables.
  - **myid** (= my identification number)
  - **numprocs** (= number of all processes in the communicator).
    :These variables can be used without declaration inner MyMatVec function.
- **myid** and **numprocs** should be used to parallelize the program.
  - To parallelize MyMatVec function, using **myid** and **numprocs** is needed.

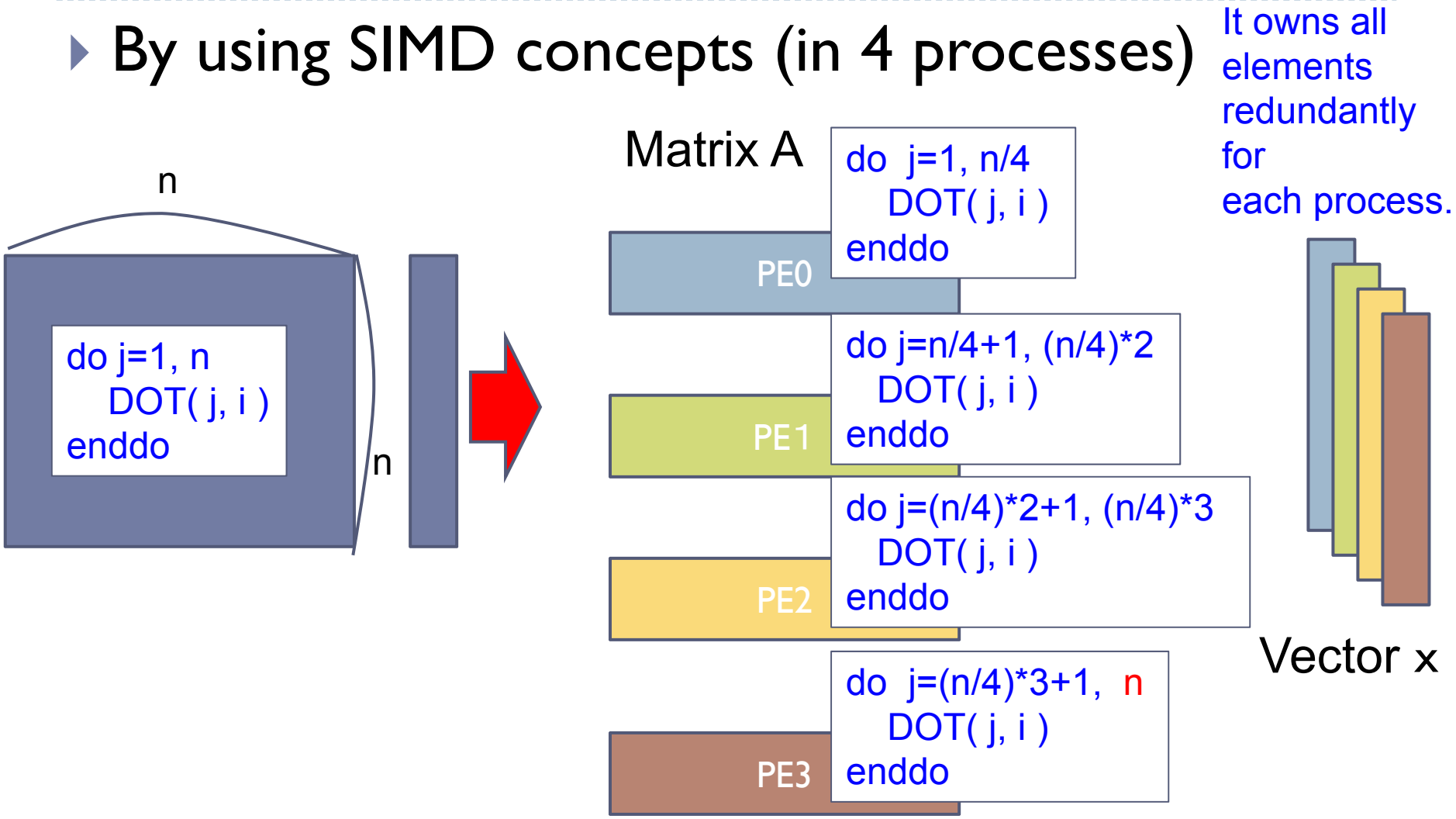
Idea of Parallelization (C Language)

- By using SIMD concepts (in 4 processes)



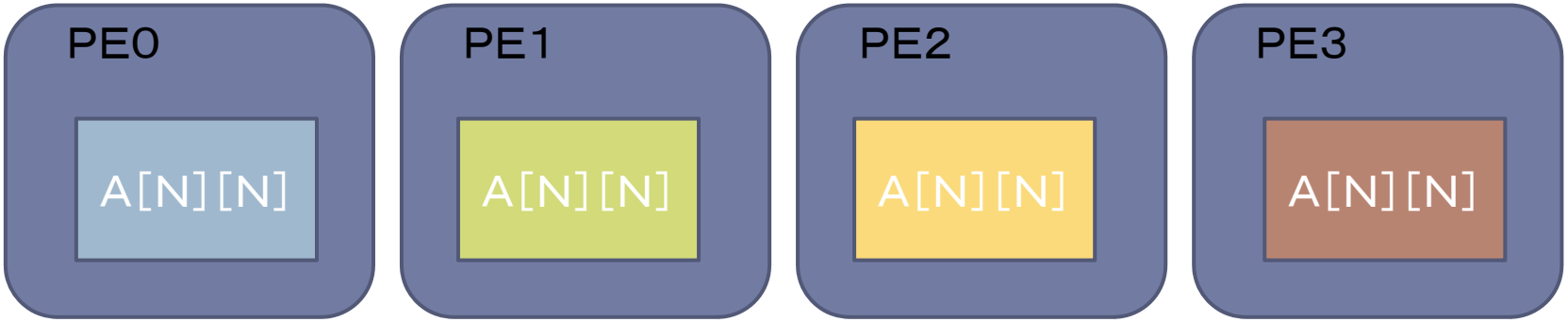
Idea of Parallelization (Fortran Language)

- By using SIMD concepts (in 4 processes)

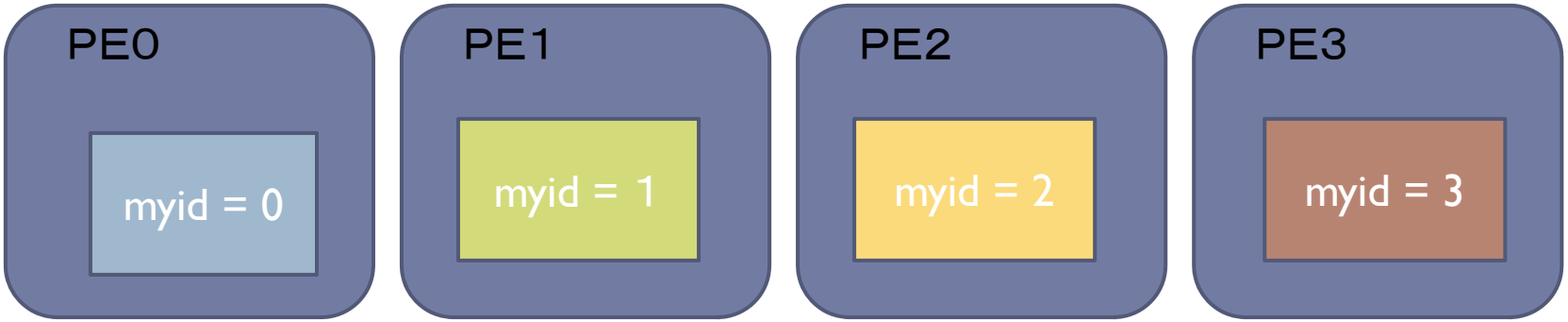


Note: Points to parallelization for beginners

- An independent array is allocated in each process.



- Value of myid is fixed after calling MPI_Comm_rank() function.



Strategy of parallelization (C Language)

1. Let matrix *A* with N × N be allocated, and vectors *x* and y with N be allocated for each process.
2. Modify initial numbers of loop for starting and ending to compute allocated region.
   - If we use block distribution, we obtain:
     （In case of that **n** can be devisable for **numprocs**. ）

```
ib = n / numprocs;
for ( j=myid*ib;  j<(myid+1)*ib;  j++)    { … }
```

3. (After finalizing parallelization of step 2) modify memory allocation for arrays to have allocated data only. Then modify loops to compute it.
   - For the above loop, we can adapt as follows:

```
for ( j=0;  j<ib;  j++)  { … }
```

Strategy of parallelization (Fortran Language)

1. Let matrix *A* with N × N be allocated, and vectors *x* and *y* with N be allocated for each process.
2. Modify initial numbers of loop for starting and ending to compute allocated region.
   - If we use block distribution, we obtain:
     （In case of that **n** can be devisable for **numprocs.** ）

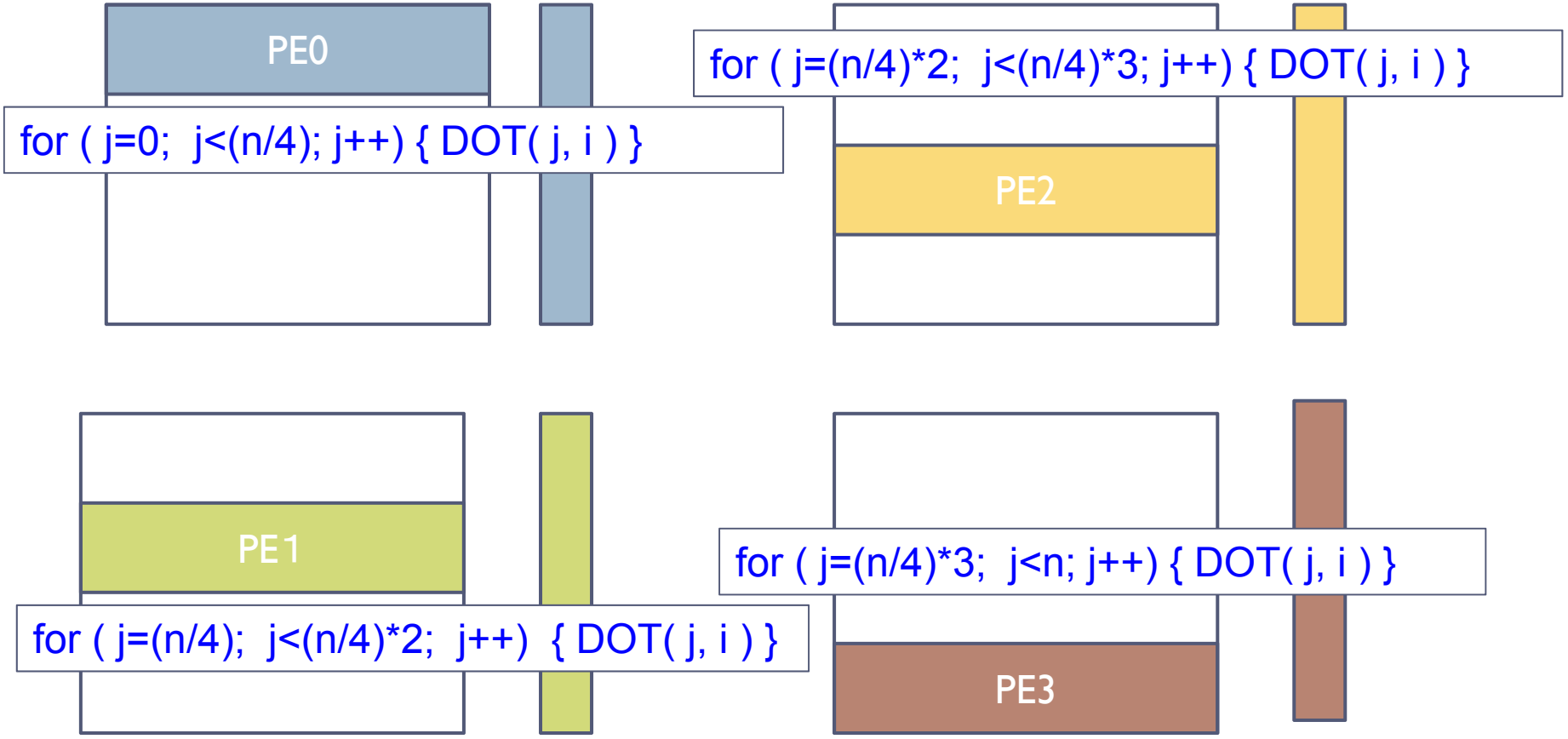
```
ib = n / numprocs
do j=myid*ib+1, (myid+1)*ib  ....  enddo
```

3. (After finalizing parallelization of step 2) modify memory allocation for arrays to have allocated data only. Then modify loops to compute it.
   - For the above loop, we can adapt as follows:

```
do  j=1, ib  ....  enddo
```

Strategy of parallelization (C Language)

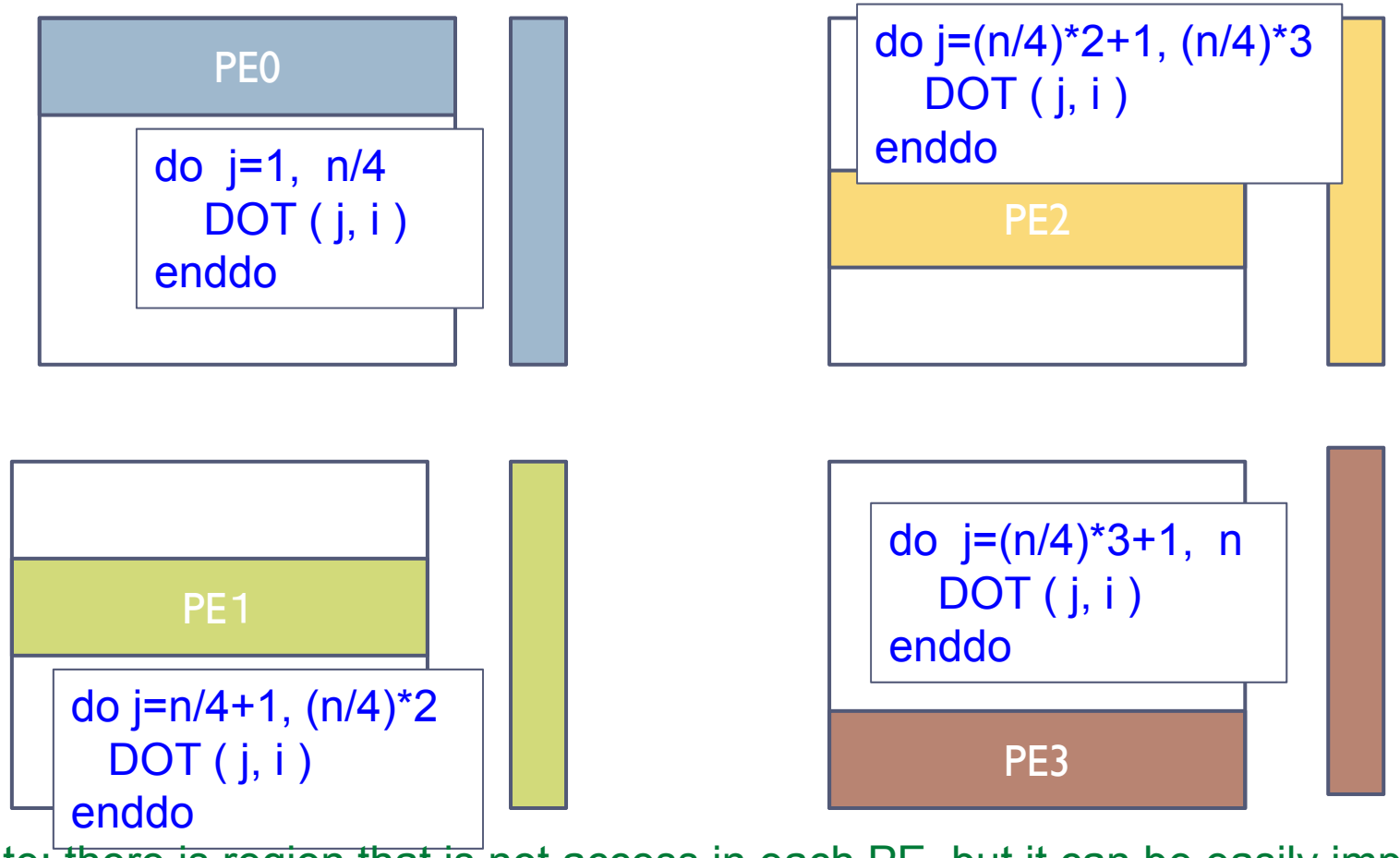
- In case of having all elements of matrix A with N × N:



Note: there is region that is not access in each PE, but it can be easily implemented due to easy specification of length of loops.

Strategy of parallelization (Fortran Language)

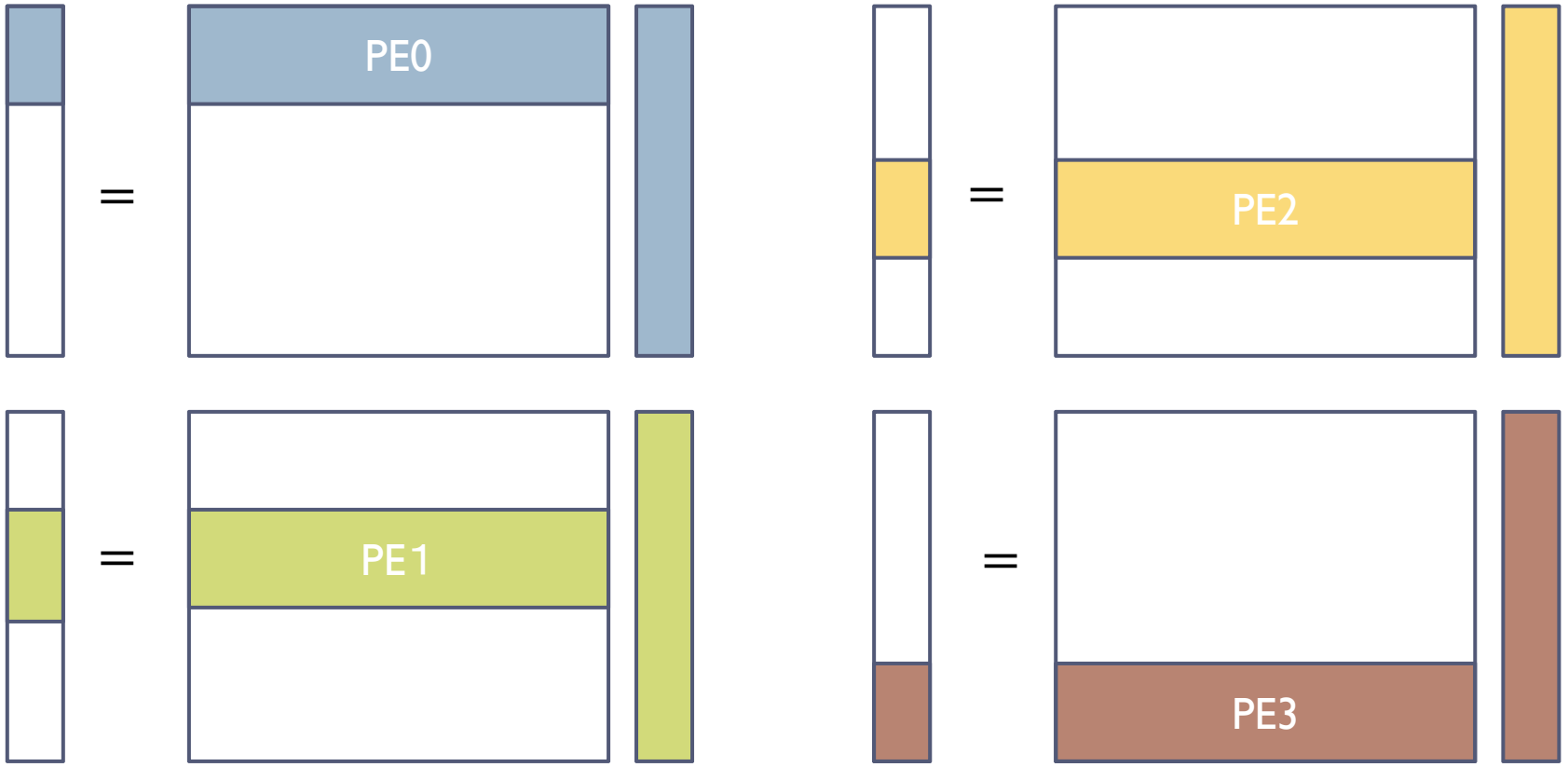
- In case of having all elements of matrix A with N × N:



Note: there is region that is not access in each PE, but it can be easily implemented due to easy specification of length of loops.

Strategy of parallelization (Dense matrix-vector multiplications)

- In this strategy, vector y for y = A x is partially calculated in each PE as follows.



Note: Parallel Environment

- 192 processes can be used in the lecture environment.
- To verify result, use debug function that is including in sample program.
  - There may have a bug that you think that parallelizing is finished.
  - For the sample program in initial state, all elements of y are stored in rank #0, since it is sequential program. Hence:

Parallelize verification processes with respect to loop of sequential execution.

In case that N is not divisible for the number of processors

- If N is not divisible for 192 which is maximum number of cores in the lecture environment, then we need to set end index of loop to rank #191, such as:

```
ib = n / numprocs;
if ( myid == (numprocs - 1) ) {
      i_end = n;
} else {
     i_end = (myid+1)*ib;
}
for ( i=myid*ib;  i<i_end;  i++) { … }
```

An extended implementation (In case of having distributed data only.)

- In case of having distributed data only, we need to know:
  - Local index from 1 to n/192, or 0 to (n/192+(N-(N/192)*192)))
  - Global index from 0 to N:
    - After gathering data of vector x, we need to access vector x with:

      A, y: access for local index.

      x: access for global index.
  - If we use block distribution, it is easy to implement it.
  - If we use cyclic distribution, we need something to consider:
    - By using modulo function (a%b).

Lessons

1. Compare performance of row-wise computation and column-wise computation for dense matrix-vector multiplication. It is not needed to parallelize the code.
2. [Homework 2] Parallelize the sample program. You can allocate matrix A with N × N, and vectors x and y with N for each process.
3. Parallelize sample program. In this lesson, you can only allocate distributed data for matrix A, and vectors x and y. Hence, total amount of memory for each process is reduced with 1/192 compared to that of sequential. You can use extra work area for parallelization if you need.

Lessons

4. After parallelizing, make a parallel code that can execute pure MPI and hybrid MPI execution. Evaluate performance with the code with respect to environmental condition, such as maximum12 nodes (192 cores).
   - There are many combinations for hybrid MPI. In case of 12 processes MPI execution, we can execute a 1 MPI+16 OpenMP threads/node, 2 MPIs + 8 OpenMP threads/node, and 4 MPIs + 4 OpenMP threads/node, etc.

Answer codes of hybrid MPI/OpenMP for matrix-vector multiplication

Answer Code of dense matrix-vector multiplications (Hybrid MPI/OpenMP)

- File name of C/Fortran languages for answer codes of Hybrid MPI/OpenMP:

  Mat-vec-fx_ans.tar

- Change queue name from lecture to lecture6 in job script files.
- The job script files are as follows:
  - mat-vec.bash.P192T1 : Sample job script of Pure MPI.
  - mat-vec.bash.P96T8 : Sample job script of Hybrid MPI with 96 MPIs + 8 Threads/MPI.
  - mat-vec.bash.P12T16 : Sample job script of Hybrid MPI with 12 MPIs + 16 Threads/MPI.
- Then type "pjsub" to submit the job.
  - lecture : Queue name in out of time of this lecture.
  - lecture6 : Queue name in time of this lecture.

Execute Answer Program of Parallelization of Dense Matrix-Vector Multiplication

- Type the following commands:

```
$ cp /home/z30082/Mat-vec-fx_ans.tar ./
$ tar xvf Mat-vec-fx_ans.tar
$ cd Mat-vec
```

- Choose the follows:

```
$ cd C : For C language.
$ cd F : For Fortran.
```

- The follows are common.

```
$ make
```

- After finalizing execution, type the follow. This is in case of Hybrid MPI/OpenMP execution with 12 MPIs + 16 Threads/MPI.

```
$ cp ./mat-vec.bash.P12T16 ./mat-vec.bash
```

- Modify queue name in mat-vec.bash.

```
$ pjsub mat-vec.bash
```