Parallel Scientific Computation

OpenMP 2

J.-H. Parq
IPCST
Seoul National University

Do/For Directive

- Automatic loop division and assignment
  - Used in a parallel block
  - Synchronization at the end: barrier

- Fortran (f90)

```
!$omp do [clauses]
    do i = 1, N
        ......
    end do
!$omp end do [nowait]
```

- C/C++

```
#pragma omp for [clauses]
  for (i=0; i<N; i++) {
        ......
  }
```

Do/For Directive

- Nowait / no wait
  - Finishing threads go out of the loop and ahead.
  - No synchronization at the end

- Fortran (f90)

```
!$omp do
    do i = 1, N
        ......
    end do
!$omp end do nowait
```

- C/C++

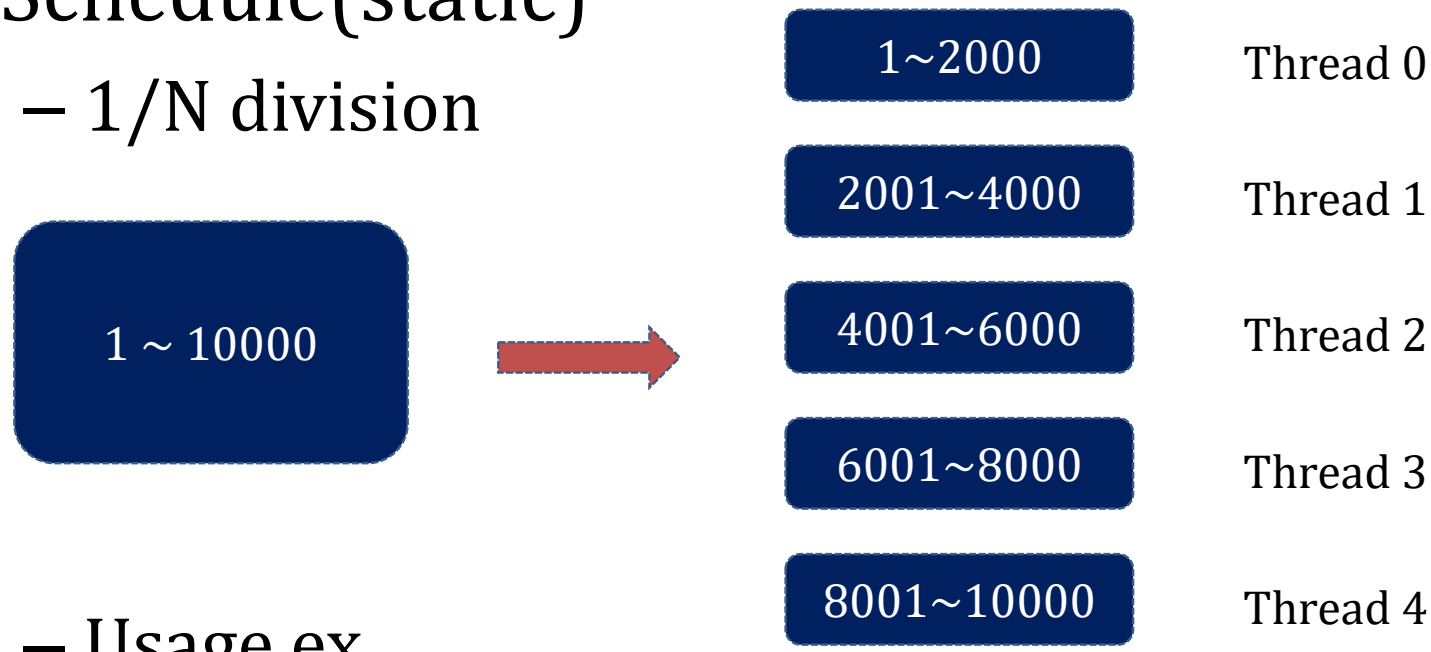
```
#pragma omp for nowait
  for (i=0; i<N; i++) {
        ......
  }
```

Do/For Directive

- Loop division
  - Default: compiler dependent
  - You can control it by ‘schedule’ clause
  - Clause: schedule(type[, chunk size])
    - Static: divided into chunks and then assigned to threads consecutively
    - Dynamic: assigned to threads in order of performance
    - Guided: similar to dynamic but the chunk size changes proportional to (remainder/# of threads)
    - Runtime: controlled by ‘OMP_SCHEDULE’ (external)
      - Ex.) export OMP_SCHEDULE=“static,500”

Do/For Directive

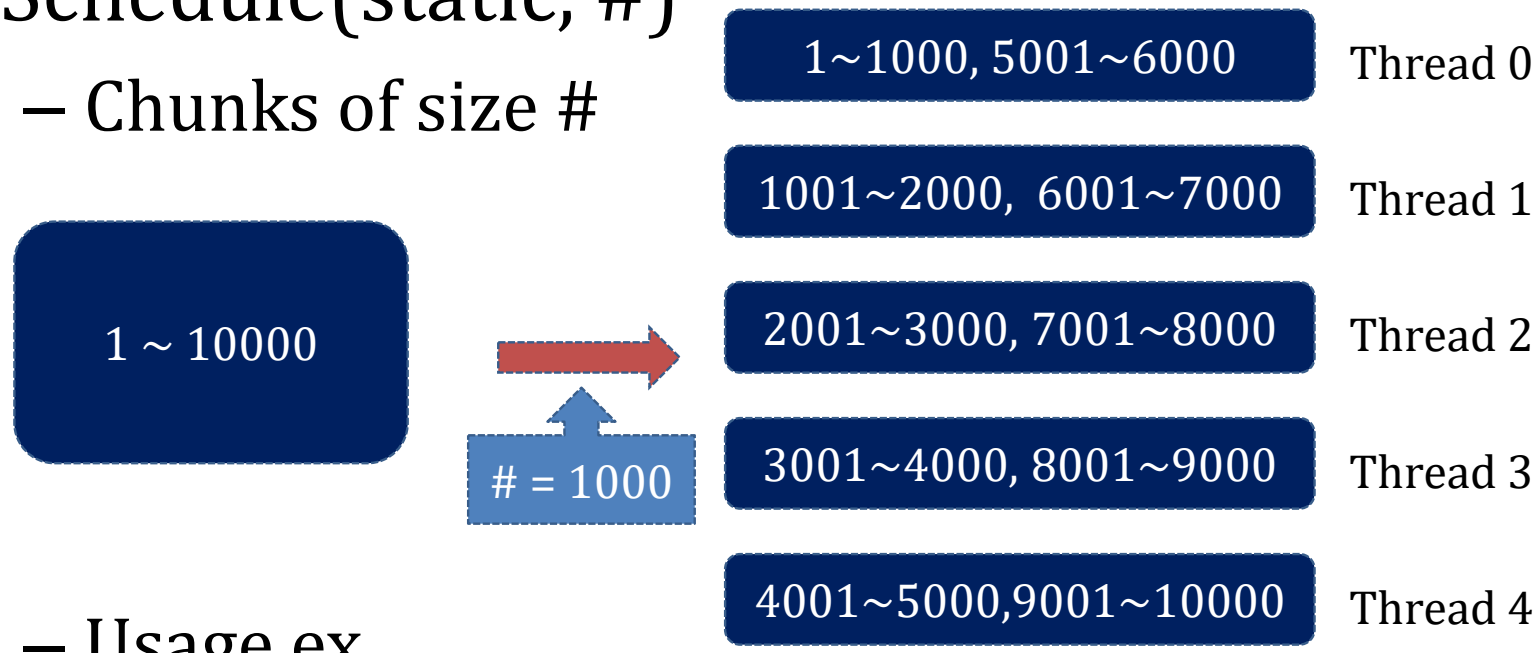
- Schedule(static)
  - 1/N division



  - Usage ex.
    - C: #pragma omp for schedule(static)
    - Fortran: !$omp do schedule(static)

Do/For Directive

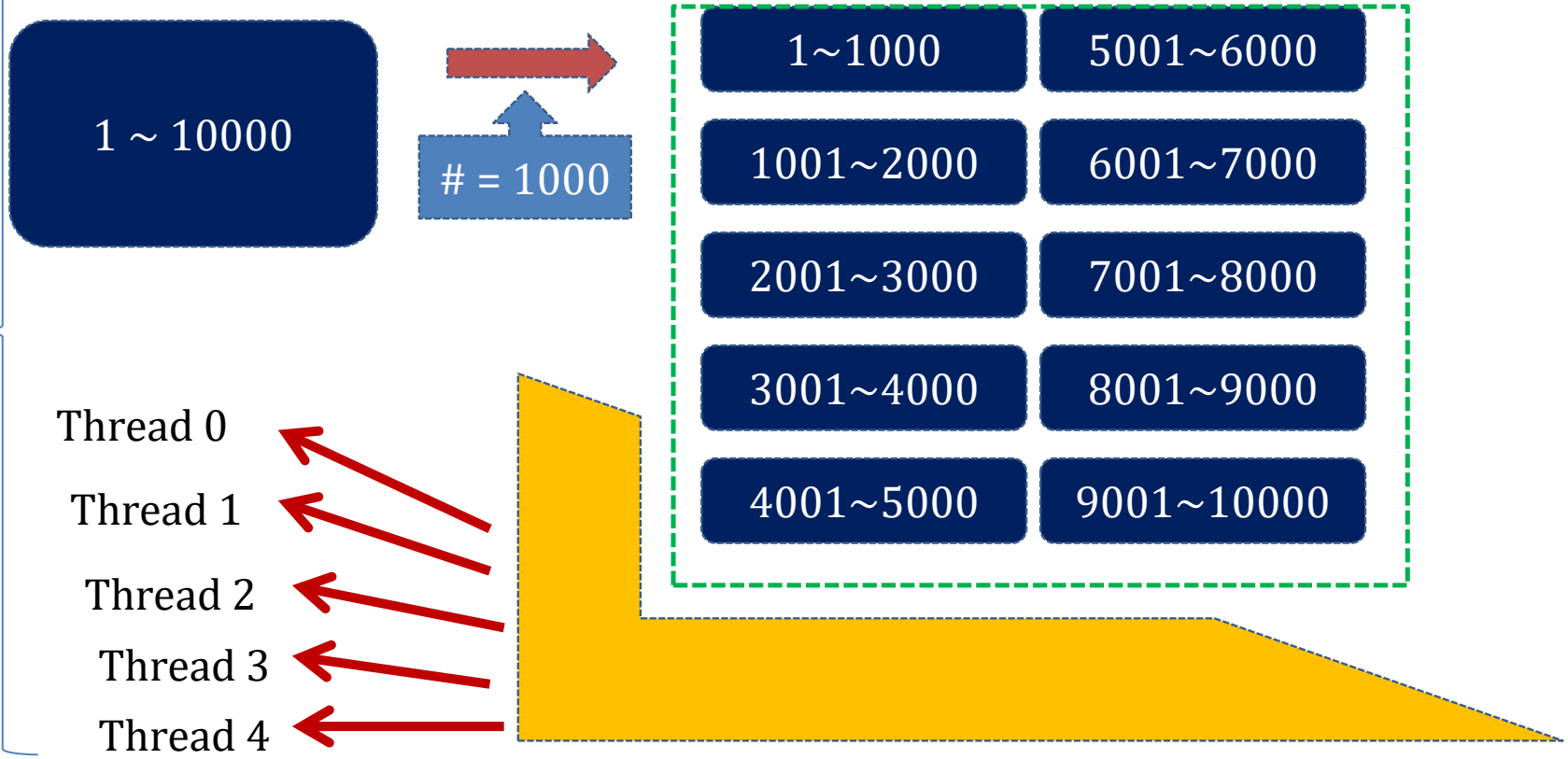
- Schedule(static, #)
  - Chunks of size #



  - Usage ex.
    - C: #pragma omp for schedule(static, 1000)
    - Fortran: !$omp do schedule(static, 1000)

Do/For Directive

- Schedule(dynamic, #)



Section(s) Directive

- Block structure
  - Functional decomposition

- Fortran (f90)

```
!$omp sections [clauses]
    ......
!$omp section
    ......
!$omp section
    ......
!$omp end sections [nowait]
```

- C/C++

```
#pragma omp sections [clauses]
{
        ......
   #pragma omp section
        ......
   #pragma omp section
    {     ......     }
}
```

Clauses for Parallel/Sections/Do/For

- Common
  - **private**, firstprivate, **reduction**
- Parallel
  - **shared**, **default**, **if**, copyin, num_threads, **do/for**, **sections**
- Sections
  - **nowait,** lastprivate
- Do/For
  - **schedule**, ordered, lastprivate, **shared, collapse, nowait**

Variable Sharing Property

- Rules for variables without setting by clauses
  - Mostly ‘shared’
  - Loop index: private in Fortran
  - Stack (automatic) variables: private
    - Local variables in subroutines called from parallel
    - Local pointers, ……
- Default clause
  - It sets default sharing properties.
  - Fortran options: private/firstprivate/shared/none
  - C/C++ options: shared/none

Matrix-Vector Multiplication

Fortran

```
!$omp parallel private(sum)
!$omp do
    do i = 1, N
        sum = 0d0
        do k = 1, M
            sum = sum + A(i,k)*x(k)
        end do
        y(i) = sum
    end do
!$omp end do
!$omp end parallel
```

C

```
#pragma omp parallel
{
   #pragma omp for private(i,k,sum)
   for (i=0; i<N; i++) {
        sum = 0.0;
        for (k=0; k<M; k++)
            sum += A[i][k]*x[k];
        y[i] = sum;
   }
}
```

Jacobi Method

- An iterative method to solve a system of linear equations **Ax = b** (A: matrix, x & b: vectors)

$$
\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\
a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\
&\vdots \\
a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n
\end{aligned}
$$

$$
x_i^{k+1} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1, j \neq i}^{n} a_{ij} x_j^k \right)
$$

  - Iterating until $\| \mathbf{x}^k - \mathbf{x}^{k-1} \| <$ (tolerance)
  - Converges only if diagonally dominant($|a_{ii}| \geq \sum_{j \neq i} |a_{ij}|$)

Jacobi Method

- OpenMP parallelization of Jacobi method
  1. Initialization
     - Parallelization of a do/for loop setting **A**
  2. Main loop
     - Parallelization of a do/for loop of index *i* or *j*
       - Consider use of nowait clause
     - Reduction (summation) for error estimation
  3. Error estimation
     - Parallel do/for reduction

References

- C. Evangelinos,
  Parallel Programming for Multicore Machines Using OpenMP and MPI
- B. Barney, OpenMP
- Wikipedia