



# Ciencia y Tecnología

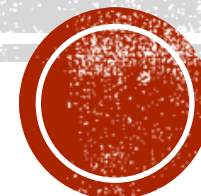
Secretaría de Ciencia, Humanidades, Tecnología e Innovación



CIMAT  
UNIDAD MÉRIDA

# OPENMP

Dr. Francisco J. Hernández López  
SECIHTI – CIMAT-Mérida  
fcoj23@cimat.mx, [www.cimat.mx/~fcoj23](http://www.cimat.mx/~fcoj23)



¿Cómo podemos obtener ventaja de una computadora **multicore** sin necesidad de reescribir el código serial que tenemos disponible o que hemos desarrollado?

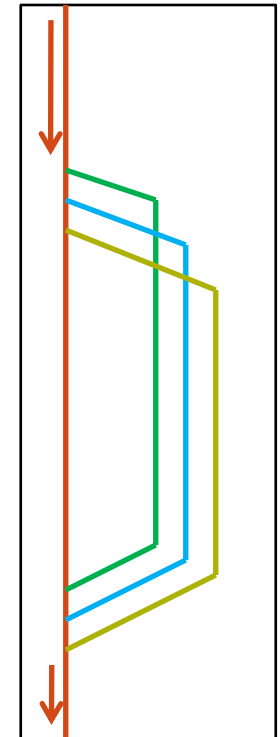


- Estándar para programación en paralelo con memoria compartida controlado por el grupo ARB (Architecture Review Board) sin fines de lucro
- Corre en sistemas multicore
- Existen también versiones de OpenMP para GPUs y Clusters
- Podemos programar usando OpenMP en Fortran, C/C++, Java, Python...
- WebPage: <https://www.openmp.org/>



# “HELLO WORLD” OPENMP C/C++

```
1  #include <stdio.h>
2  #include <iostream>
3  //OpenMP
4  #include <omp.h>
5
6  int main(void){
7
8      int nthreads, tid;
9
10     omp_set_num_threads(4);
11
12     /*Se hace el Fork para generar los hilos y sus propias copias de variables*/
13     #pragma omp parallel private(tid)
14     {
15         /*Se obtiene y se imprime el id de los hilos generados*/
16         tid = omp_get_thread_num();
17         printf("Hola Mundo, soy el Hilo = %d\n", tid);
18
19         /*Unicamente el hilo con id==0 hace esto*/
20         if (tid == 0){
21             nthreads = omp_get_num_threads();
22             printf("Numero de hilos = %d\n", nthreads);
23         }
24     } /* Todos los hilos se sincronizan y terminan aqui con Join */
25
26     //system("pause");
27     return(0);
28 }
```



# "HELLO WORLD" OPENMP FORTRAN

HolaMundo.f

```

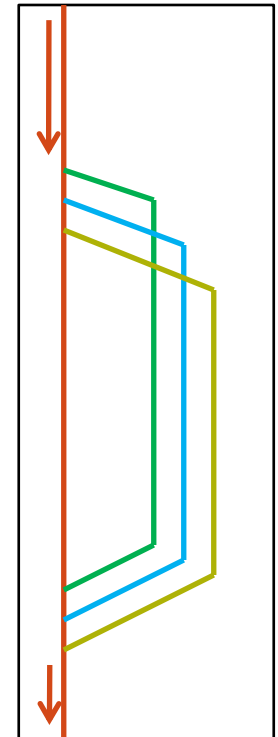
c      Programa: HolaMundo utilizando OpenMP
c
c
PROGRAM HolaMundo
    IMPLICIT NONE
    INTEGER NTHREADS, TID, OMP_GET_NUM_THREADS, OMP_GET_THREAD_NUM

!$OMP PARALLEL PRIVATE(TID)
C      Obtenemos e imprimimos el ID del THREAD.
    TID = OMP_GET_THREAD_NUM()
    PRINT *, 'Hola Mundo... Hilo = ', TID

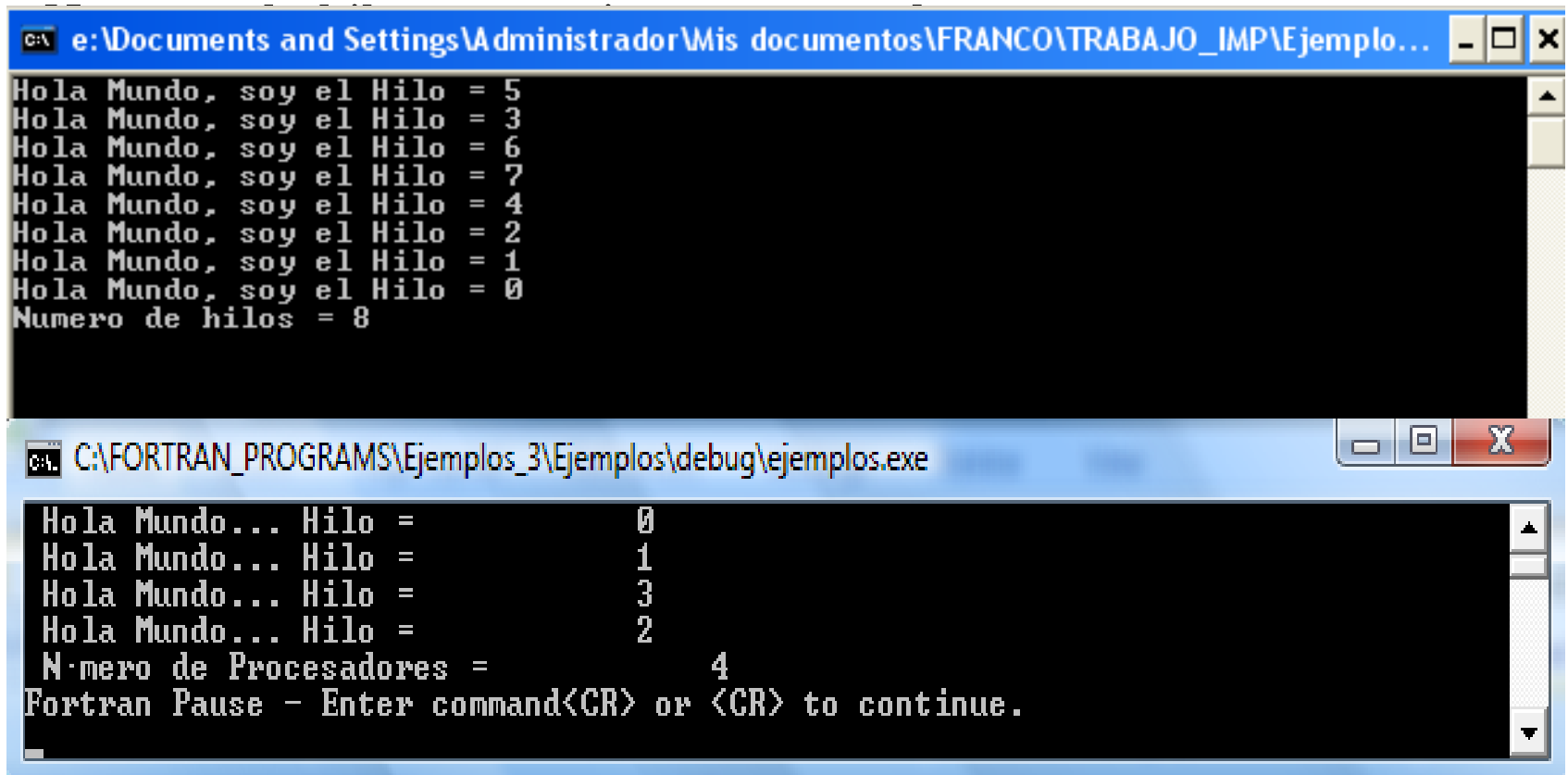
C      Este bloque lo hace solo el THREAD 0
    IF (TID .EQ. 0) THEN
        NTHREADS = OMP_GET_NUM_THREADS()
        PRINT *, 'Número de Procesadores = ', NTHREADS
    END IF
!$OMP END PARALLEL

    pause
END PROGRAM HolaMundo

```



# RESULTADO DEL “HELLO WORLD” EN OPENMP



```
e:\Documents and Settings\Administrador\Mis documentos\FRANCO\TRABAJO_IMP\Ejemplo...  
Hola Mundo, soy el Hilo = 5  
Hola Mundo, soy el Hilo = 3  
Hola Mundo, soy el Hilo = 6  
Hola Mundo, soy el Hilo = 7  
Hola Mundo, soy el Hilo = 4  
Hola Mundo, soy el Hilo = 2  
Hola Mundo, soy el Hilo = 1  
Hola Mundo, soy el Hilo = 0  
Numero de hilos = 8  
  
C:\FORTRAN_PROGRAMS\Ejemplos_3\Ejemplos\debug\ejemplos.exe  
Hola Mundo... Hilo = 0  
Hola Mundo... Hilo = 1  
Hola Mundo... Hilo = 3  
Hola Mundo... Hilo = 2  
Numero de Procesadores = 4  
Fortran Pause - Enter command<CR> or <CR> to continue.
```

# DIRECTIVAS EN OPENMP (C1)

## ■ Región Paralela:

- `!$OMP PARALLEL, !$OMP END PARALLEL.` (Fortran)
- `#pragma omp parallel {}.` (C/C++)

## ■ División del Trabajo:

- `!$OMP DO, !$OMP END DO` (Fortran)
- `#pragma omp for` (C/C++)

} Especifica la ejecución en paralelo de un ciclo de iteraciones.

- `!$OMP SECTIONS, !$OMP END SECTIONS`
- `#pragma omp sections {}`

} Especifica la ejecución en paralelo de algún bloque de código secuencial.

- `!$OMP SINGLE, !$OMP END SINGLE`
- `#pragma omp single`

} Define una sección de código donde exactamente un Hilo tiene permitido ejecutar el código.

# DIRECTIVAS EN OPENMP (C2)

- **Combinaciones de directivas:**
  - !\$OMP **PARALLEL DO**, !\$OMP END PARALLEL DO. (Fortran)
  - #pragma omp **parallel for** {} (C/C++)
  - !\$OMP **PARALLEL SECTIONS**, !\$OMP END PARALLEL SECTIONS (Fortran)
  - #pragma omp **parallel sections** {} (C/C++)

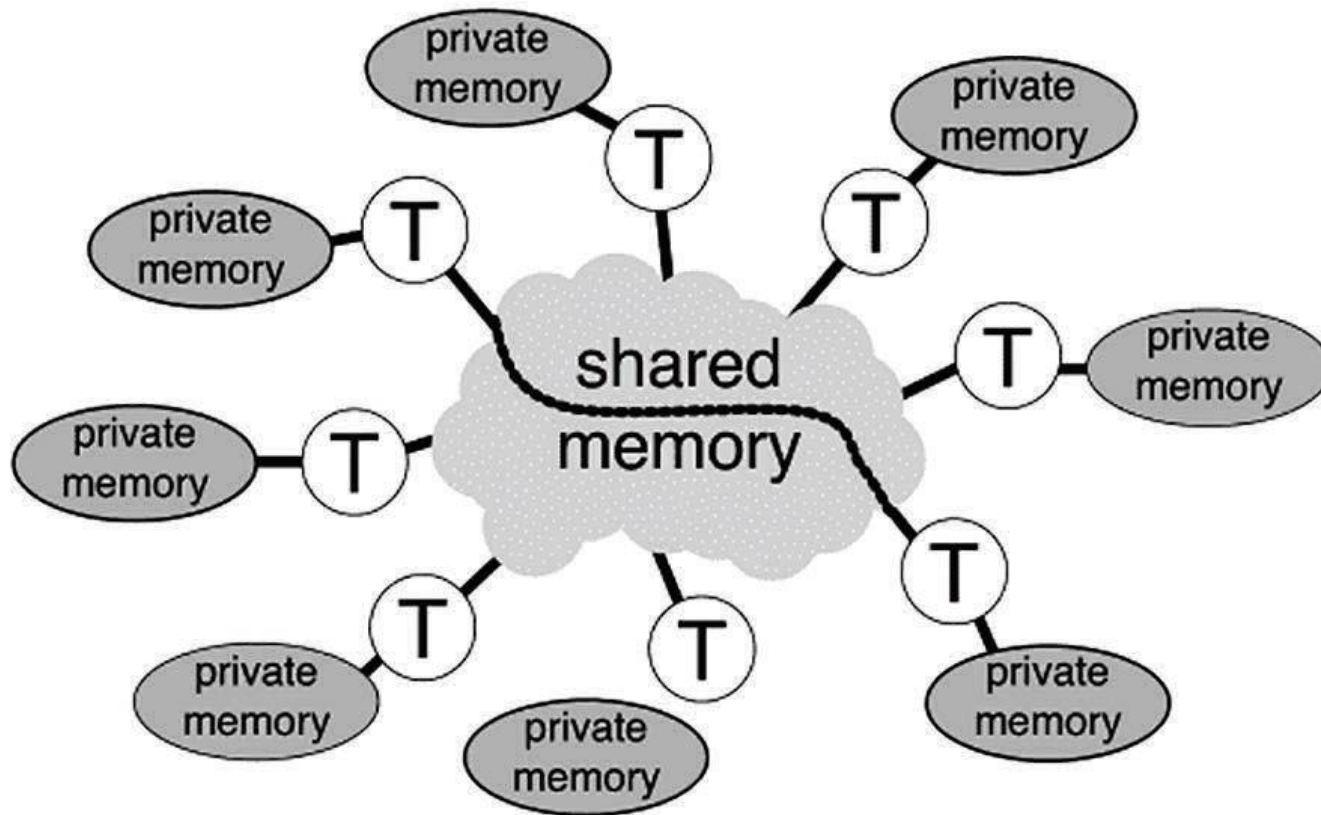
# DIRECTIVAS EN OPENMP (C3)

- **Sincronización:**

- **CRITICAL:** Solo un hilo a la vez tiene permitido ejecutar el código.
- **ORDERED:** Asegura que el código se ejecuta en el orden en que las iteraciones se ejecutan de forma secuencial.
- **ATOMIC:** Asegura que una posición de memoria se modifique sin que múltiples hilos intenten escribir en ella de forma simultánea.
- **FLUSH:** El valor de las variables se actualiza en todos los hilos (ejemplo: banderas).
- **BARRIER:** Sincroniza todos los hilos.
- **MASTER:** El código lo ejecuta sólo el hilo maestro (No implica un Barrier).



# MODELO DE LA MEMORIA EN OPENMP



# DIRECTIVAS PARA MANIPULAR LOS DATOS

- ***private(lista)***: Las variables de la lista son privadas a los hilos, lo que quiere decir que cada hilo tiene una variable privada con ese nombre
- ***firstprivate(lista)***: Las variables son privadas a los hilos, y se inicializan al entrar con el valor que tuviera la variable correspondiente
- ***lastprivate(lista)***: Las variables son privadas a los hilos, y al salir quedan con el valor de la última iteración (si estamos en un bucle do paralelo) o sección

# DIRECTIVAS PARA MANIPULAR LOS DATOS (C1)

- ***shared(lista)***: Indica las variables compartidas por todos los hilos
- ***default(shared|none)***: Indica cómo serán las variables por defecto. Si se pone *none* las que se quiera que sean compartidas habrá que indicarlo con la cláusula *shared*
- ***reduction(operador:lista)***: Las variables de la lista se obtienen por la aplicación del operador, que debe ser asociativo

# EJEMPLO: PRIVATE Y FIRSTPRIVATE

```
1 x = 5;
2 y = 20;
3 #pragma omp parallel private(x) firstprivate(y)
4 {
5     x = 10;           // x is undefined on entry, but now set to 10
6     int z = x + y;    // y was pre-initialized to a value of 20
7     .....
8     y = 30;           // (first)private variables may be modified
9     .....
10 } // End of parallel region
```

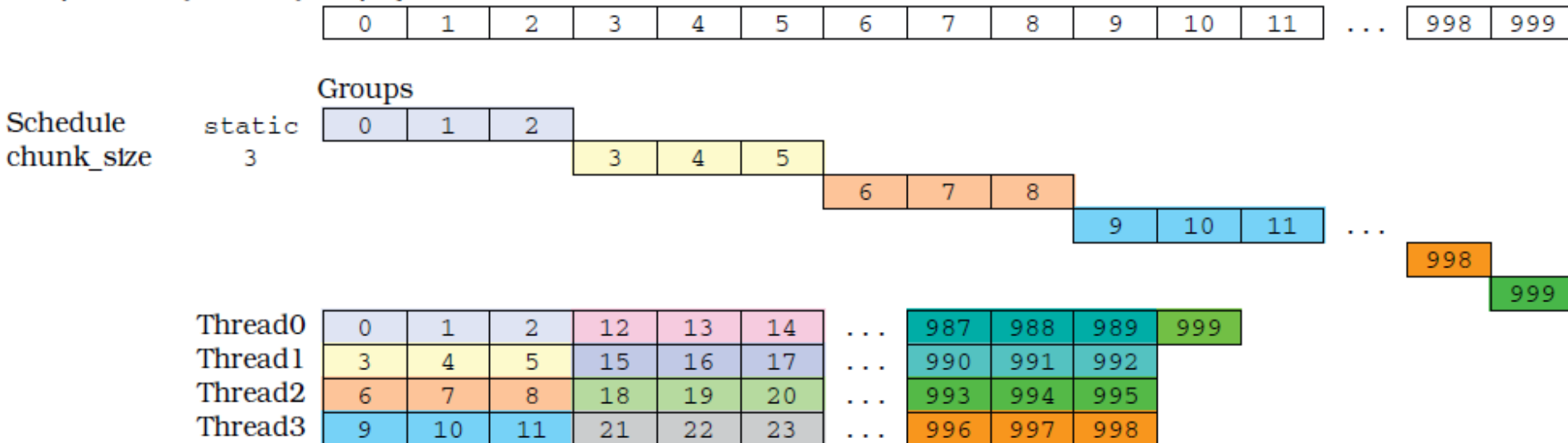
# CLAUSULA SCHEDULE DE OPENMP

- Utilizada con la directiva **DO** o **for**
- Especifica un algoritmo de planificación, que determina de qué forma se van a dividir las iteraciones del ciclo entre los hilos disponibles
- Se debe especificar un tipo y, opcionalmente, un tamaño (*chunk*)
- Tipos:
  - **STATIC.**
  - **DYNAMIC.**
  - **GUIDED.**
  - **RUNTIME.**

# SCHEDULE (STATIC)

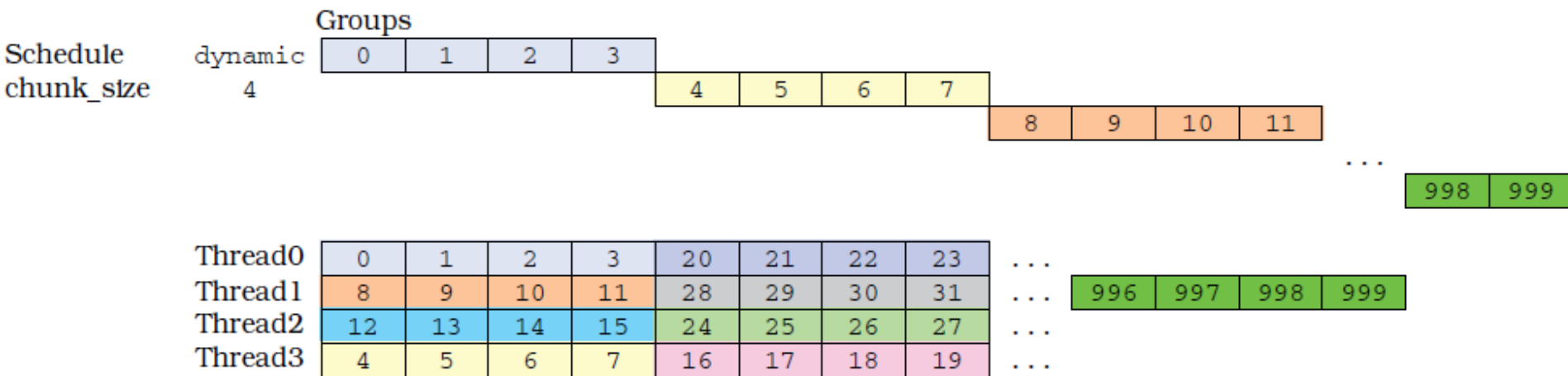
- El ciclo se divide en grupos de  $chunk\_size$  iteraciones
- Si no se especifica el  $chunk\_size$  entonces el valor por default es  $\left\lceil \frac{N}{p} \right\rceil$ , con  $N$  el número de iteraciones y  $p$  el número de hilos

```
for(int i=0;i<1000;i++) {...
```



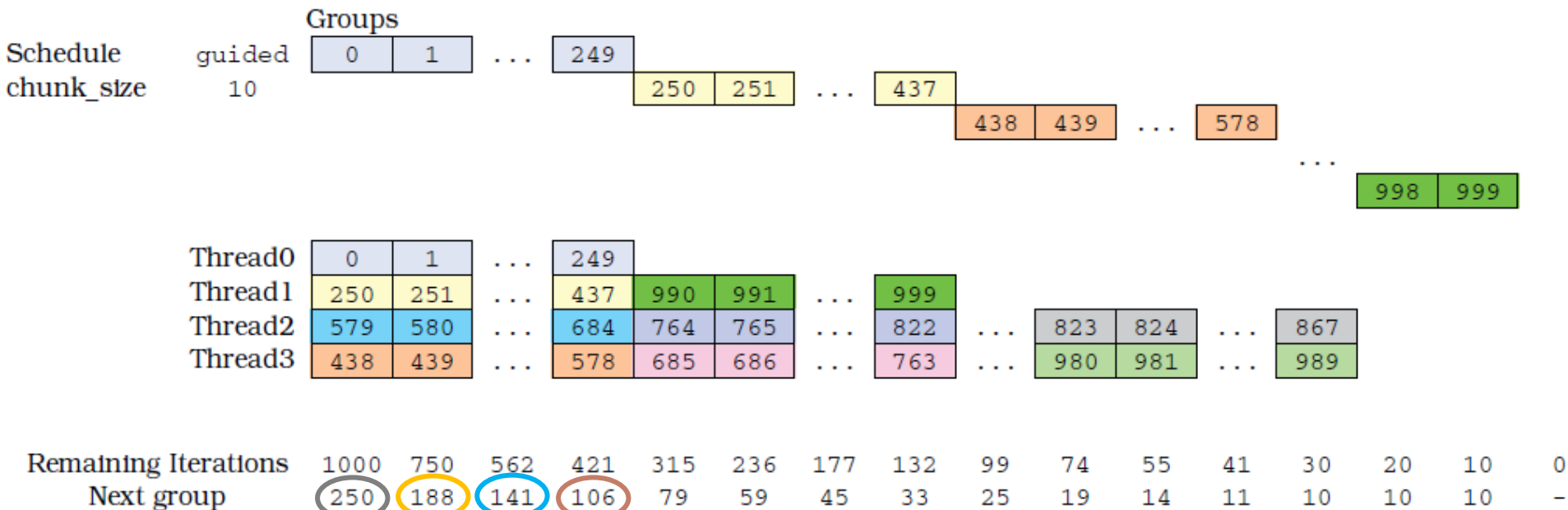
# SCHEDULE (DYNAMIC)

- Imita el funcionamiento de un balanceo de cargas dinámico
- Un grupo de *chunk\_size* es asignado a un hilo de la forma “primero en llegar primero en recibir”
- Si no se especifica el *chunk\_size* entonces el valor por default es 1



# SCHEDULE (GUIDED)

- Utiliza grupos de tamaño variable que se reducen con el tiempo para equilibrar la carga de trabajo entre los hilos.
- El tamaño del grupo dado en cada nueva solicitud es igual al número de iteraciones no asignadas dividido por  $p$ .
- chunk\_size* especifica un límite inferior para el tamaño de un grupo. Si *chunk\_size* no está especificado, el valor predeterminado es 1.





# GRACIAS POR SU ATENCIÓN

Francisco J. Hernández-López

[fcoj23@cimat.mx](mailto:fcoj23@cimat.mx)

WebPage:

[www.cimat.mx/~fcoj23](http://www.cimat.mx/~fcoj23)

