Odd-Even Sort

```
void Odd_even_sort(
      int  a[]  /* in/out */,
      int  n/* in */) {
   int phase, i, temp;

   for (phase = 0; phase < n; phase++)
      if (phase % 2 == 0) { /* Even phase */
      for (i = 1; i < n; i += 2)
            if (a[i-1] > a[i]) {
            temp = a[i];
            a[i] = a[i-1];
            a[i-1] = temp;
            }
      } else { /* Odd phase */
      for (i = 1; i < n-1; i += 2)
            if (a[i] > a[i+1]) {
            temp = a[i];
            a[i] = a[i+1];
            a[i+1] = temp;
            }
      }
}  /* Odd_even_sort */
```

Para paralelizar este algoritmo de ordenamiento se puede hacer de dos formas, la primera es usando un **#pragma `omp for`** y la otra forma es usando un **#pragma `omp parallel for`**.

#pragma omp parallel genera un grupo de subprocesos, mientras que **#pragma omp** divide las iteraciones de bucle entre los subprocesos generados. Puede hacer ambas cosas a la vez con la directiva fusionada **#pragma omp parallel for directive**

DIRECTIVA

Una directiva ejecutable de OpenMP se aplica al bloque estructurado siguiente o una construcción de OpenMP. Un "bloque estructurado" es una declaración única o una declaración compuesta con una sola entrada en la parte superior y una única salida en la parte inferior.

1 #PRAGMA OMP FOR

La construcción del bucle especifica que las iteraciones de los bucles serán distribuidas y ejecutadas por el equipo de hilos.

```
void Odd_even(int a[], int n) {
   int phase, i, tmp;

#  pragma omp parallel num_threads(thread_count) \
      default(none) shared(a, n) private(i, tmp, phase)
   for (phase = 0; phase < n; phase++) {
      if (phase % 2 == 0)
//solo distribuye el trabajo en hilo
#     pragma omp for
      for (i = 1; i < n; i += 2) {
            if (a[i-1] > a[i]) {
            tmp = a[i-1];
            a[i-1] = a[i];
            a[i] = tmp;
            }
      }
      else
#     pragma omp for
      for (i = 1; i < n-1; i += 2) {
            if (a[i] > a[i+1]) {
            tmp = a[i+1];
            a[i+1] = a[i];
            a[i] = tmp;
            }
      }
   }
}
```

2. OMP PARALLEL FOR

La construcción paralela forma un equipo de hilos y comienza la ejecución paralela

```
void Odd_even(int a[], int n) {
   int phase, i, tmp;
   char title[100];

   for (phase = 0; phase < n; phase++) {
      if (phase % 2 == 0)
//COLOCA UNA BARRERA PARA QUE TODOS TERMINEN Y PASEN A LA SIGUIENTE FASE
//INICIALIZA LOS THREADS
#     pragma omp parallel for num_threads(thread_count) \
            default(none) shared(a, n) private(i, tmp)
      for (i = 1; i < n; i += 2) {
            if (a[i-1] > a[i]) {
            tmp = a[i-1];
            a[i-1] = a[i];
            a[i] = tmp;
```

```
            }
      }
      else
#     pragma omp parallel for num_threads(thread_count) \
            default(none) shared(a, n) private(i, tmp)
      for (i = 1; i < n-1; i += 2) {
            if (a[i] > a[i+1]) {
            tmp = a[i+1];
            a[i+1] = a[i];
            a[i] = tmp;
            }
      }
      sprintf(title, "After phase %d", phase);
      Print_list(a, n, title);
   }
}  /* Odd_even */
```

Sección Crítica

- Asegura la serialización de bloques de código.
- Se puede extender para serializar grupos de bloques con el uso adecuado de la etiqueta "nombre".
- Más lento
- El bloque de código adjunto será ejecutado por un solo hilo a la vez, y no será ejecutado simultáneamente por múltiples hilos. A menudo se usa para proteger los datos compartidos de las condiciones de carrera.
- funciona sobre un bloque de codigo

Operación atómica:

- funciona sobre una sola linea de codigo
    - Solo asegura la serialización de una operación en particular.
    - La actualización de memoria (escritura o lectura-modificación-escritura) en la siguiente instrucción se realizará atómicamente. No hace que toda la declaración sea atómica; solo la actualización de memoria es atómica. Un compilador puede usar instrucciones de hardware especiales para un mejor rendimiento que cuando se usa un dispositivo crítico .