Programación Paralela con OpenMP

Una Introducción y Uso en C para Computación Científica

23 de julio de 2025

¿Qué es OpenMP?

- OpenMP (Open Multi-Processing) es un **estándar de programación paralela** que permite la computación simultánea de muchas tareas o procesos.
- Se implementa mediante **directivas de compilador o pragmas** para lenguajes como C, C++ y Fortran, enfocándose en la programación de memoria compartida.
- Su objetivo principal es acelerar el software dividiendo el trabajo en múltiples partes del procesador de un ordenador, haciéndolo más rápido y eficiente.
- Utiliza un modelo de memoria compartida, donde los datos pueden ser accesibles por todos los hilos (compartidos) o solo por el hilo que los posee (privados).

Uso de OpenMP en Computación Científica

- Se utiliza típicamente para **paralelizar bucles (loops)**.
- Es ideal para aprovechar la arquitectura multinúcleo de los procesadores modernos (por ejemplo, CPUs y GPUs), donde cada núcleo es independiente y puede acceder a la misma memoria concurrentemente.
- En la computación científica y las simulaciones, la **paralelización de programas secuenciales** es muy común.
- Las aplicaciones que muestran **paralelismo grueso** (coarse-grained parallelism), donde las subtareas no necesitan comunicarse muy a menudo, son más fáciles de paralelizar con OpenMP.

Uso Básico en C: Regiones Paralelas

- **Incluir la cabecera:** Debes incluir el archivo de cabecera de OpenMP:

```
#include <omp.h>
```

- Especificar la región paralela: Usa la directiva `#pragma omp parallel` para definir un bloque de código que se ejecutará en paralelo:

```
#pragma omp parallel
{
    // Código de la región paralela
}
```

- Funcionamiento: Cuando el compilador encuentra una región paralela, el hilo original (llamado hilo maestro, con ID 0) "forkea"(crea) hilos adicionales. Todo el código dentro de la región paralela es ejecutado concurrentemente por todos los hilos.
- Finalización: Una vez que la región paralela termina, todos los hilos se fusionan de nuevo en el hilo maestro.
- Obtener el ID del hilo: Puedes usar `omp_get_thread_num()` para obtener el número del hilo actual.

```
        #pragma omp parallel
        {
    printf("Hola Mundo... desde el hilo = %d\n",
            omp_get_thread_num());
        }
```

Uso Básico en C: Control y Compilación

- Establecer el número de hilos: Puedes definir cuántos hilos se ejecutarán usando una variable de entorno:

  ```
  export OMP_NUM_THREADS=5
  ```

  En este caso, se crearán 5 hilos.

- Otra forma de especificar número de hilos:

  ```
  #pragma omp parallel for num_threads(4)
  for (int i = 1; i <= 10; i++) {
      int tid = omp_get_thread_num();
  ```

- Nota importante sobre el orden: El orden de ejecución de los hilos no está garantizado y puede variar en cada ejecución del programa.

Variable Privadas y Compartidas

- Las variables **private** son exclusivas de cada hilo, cada hilo tiene su propia copia.

```
int th_id, nthreads;
#pragma omp parallel private(th_id)
  {
    th_id = omp_get_thread_num();
    printf("Hello World from thread %d\n", th_id);
  }
```

- Las variables **shared**, son visibles dentro de todos los hilos y son el comportamiento predeterminado. Hay peligro de Race Conditions.

Recetas Útiles

- Reductions: https://theartofhpc.com/pcse/omp-reduction.html
- Master: https://hpc-tutorials.llnl.gov/openmp/master_directive/
- Directives: https://theartofhpc.com/pcse/omp-basics.html#Codeandexecutionstructure
-