Programación de Arquitecturas Multinúcleo

Curso 2024/2025

Trabajo Práctico 2: Programación con OpenMP

1. Razona el comportamiento del `ejemplo_for.c`, al ejecutarlo, con 16 hilos, usando distintos `schedules(static, dynamic, guided)` y tamaños de reparto (1 y 2).

2. Compara razonadamente el comportamiento de `ejemplo_sections.c` vs. `ejemplo_single.c`.

3. Compara razonadamente el comportamiento de `ejemplo_sections.c` vs. `ejemplo_critical.c`.

4. Compara los programas `ejemplo_critical_pi.c` y `ejemplo_atomic_pi.c` ¿Ofrecen el mismo resultado? ¿Qué ocurre con sus tiempos de ejecución conforme aumenta el número de hilos que se utilizan? Razona tus respuestas.

5. En el programa `ejemplo_flush_2.c`, ¿para qué se utiliza la variable `flag`? Muestra y razona el comportamiento al ejecutarse si no se utiliza esta variable.

6. A partir de `ejemplo_flush_2.c`, crea `ejercicio_flush_3.c`, donde un hilo lee 2 datos numéricos desde el teclado, un segundo hilo los suma y un tercer hilo muestra el resultado.

7. Ejecuta varias veces el programa `ejemplo_lastprivate.c`. ¿Muestra siempre el mismo resultado de la variable `x` en el último mensaje en pantalla: "*Después de pragma parallel x=...*"? Cambia el `schedule` del bucle for a `dynamic` y repite el ejercicio. Razona tus respuestas.

8. Modifica el programa `ejemplo_reduction.c` para que en lugar de calcular la suma de los valores de la variable `x` de todos los hilos, muestre el valor máximo de dichos valores.

9. En el programa `ejemplo_nested.c`, ¿qué ocurre si cambias `omp_set_dynamic(0)` por `omp_set_dynamic(1)`? Muestra el comportamiento al ejecutarlo y razona tu respuesta. ¿Y si cambias `omp_nested(1)` por `omp_nested(0)`? Muestra el comportamiento al ejecutarlo y razona tu respuesta.

10. A partir del ejemplo `ejemplo_atomic_pi.c`, crea el programa `ejercicio_lock_pi.c` que realiza el mismo cálculo pero, en lugar de utilizar operaciones atómicas, haga uso de un candado para sincronizar el acceso en exclusión mutua a la variable `sum` que es compartida por todos los hilos.

11. A partir del ejemplo de Fibonacci, `ejemplo_fibo.c`, crea la versión `ejercicio_fibo_2.c`, con el objetivo de reducir la profundidad del árbol de tareas OpenMP que se generan. Para ello, cuando el número a calcular sea menor que una cota dada, (por ejemplo, menor que 20) el cálculo se debe hacer directamente en la tarea correspondiente, mediante las oportunas llamadas recursivas secuenciales, pero sin que se generen más tareas OpenMP a partir de ella. Comprueba razonadamente que el programa tiene este comportamiento.

12. Diseña el programa paralelo, usando OpenMP, `MulMatCua.c` para multiplicar matrices cuadradas de números reales en doble precisión, $C_{nxn}=A_{nxn}B_{nxn}$. Las matrices `A`, `B` y `C` se encuentran almacenadas en memoria por filas, con un *leading dimension* (distancia de almacenamiento entre cada par de elementos consecutivos de una columna) de `ldn` elementos. Se debe repartir el trabajo a realizar de manera que el cálculo de cada conjunto de `F` filas consecutivas de la matriz `C` sea asignado a un hilo de entre los `t` hilos generados.

    SINTAXIS: `MulMatCua <n> <ldn> <F> <t>`

    Toma como punto de partida el ejemplo de multiplicación matriz-vector, `ejemplo_mv.c`, cuya rutina principal, que se muestra a continuación, multiplica la matriz `m` por el vector `v`, dejando el resultado en el vector `w`. En este programa, la matriz `m` tiene `fm` filas y `cm` columnas. Esta matriz se encuentra almacenada en memoria por filas, con un *leading dimension* (distancia de almacenamiento entre cada par de elementos consecutivos de una columna) de `ldm` elementos.

```
void mv(double *m,int fm,int cm,int ldm,double *v,double *w)
{
  int i,j,iam,nprocs;
  double s;
  #pragma omp parallel private(iam,nprocs)
  {
    nprocs=omp_get_num_threads();
    iam=omp_get_thread_num();
    #pragma omp master
      THREADS=nprocs;
    #pragma omp for private(i,s,j) schedule(static)
    for (i = 0; i < fm; i++)
    {
      s=0.;
      for(j=0;j<cm;j++)
        s+=m[i*ldm+j]*v[j];
      w[i]=s;
    }
}
}
```

13. A partir del programa anterior, escribe un programa paralelo usando OpenMP, llamado `MulMat.c`, para calcular el producto de dos matrices rectangulares de números reales de doble precisión, de la forma $C_{mxn}=A_{mxk}B_{kxn}$. Las matrices `A`, `B` y `C` se encuentran almacenadas en memoria por filas, con un *leading dimension* (distancia de almacenamiento entre cada par de elementos consecutivos de una columna) de `lda`, `ldb` y `ldc` elementos, respectivamente. Se debe repartir el trabajo a realizar de manera que el cálculo de cada conjunto de `F` filas consecutivas de la matriz `C` sea asignado a un hilo de entre los `t` hilos generados.

    SINTAXIS: `MulMat <m> <n> <k> <lda> <ldb> <ldc> <F> <t>`

14. A partir del programa anterior, escribe un programa paralelo usando tareas de OpenMP, llamado `TMulMat.c`, para calcular el producto de dos matrices rectangulares de números reales de doble precisión, de la forma $C_{mxn}=A_{mxk}B_{kxn}$. Las matrices `A`, `B` y `C` se encuentran almacenadas en memoria por filas, con un *leading dimension* (distancia de almacenamiento entre cada par de elementos consecutivos de una columna) de `lda`, `ldb` y `ldc` elementos, respectivamente. Se debe repartir el trabajo a realizar de manera que el cálculo de cada conjunto de `F` filas consecutivas de la matriz `C` sea asignado a una tarea.

    SINTAXIS: `TMulMat <m> <n> <k> <lda> <ldb> <ldc> <F> <t>`