Práctica 1: Paralelización con OpenMP
Sesión 3: Números primos

Mónica Chillarón

Computación Paralela (CPA)

Curso 2020/2021

DSIC DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Índice

Recordatorios

- Este documento es únicamente para apoyar la explicación. El material completo de la práctica está en la tarea de PoliformaT. El **boletín** contiene toda la información necesaria y es el documento que se tiene que seguir para realizar el trabajo. Recomendable leerlo antes de cada sesión.

- Nunca utilizar espacios en blanco en los nombres de directorios o ficheros.

- El Front-end sirve para realizar comprobaciones cortas, nunca para medir tiempos o realizar ejecuciones costosas. Los tiempos se toman lanzando los trabajos al sistema de colas mediante qsub.

Algoritmo Secuencial

- Algoritmo para determinar si un número es primo o no:

```
Función primo(n)
  Si  n es par  y  no es el número 2  entonces
    p <- falso
  si no
    p <- verdadero
  Fin si
  Si  p  entonces
    s <- raíz cuadrada de n
    i <- 3
    Mientras  p  y  i <= s
      Si  n es divisible de forma exacta por i  entonces
        p <- falso
      Fin si
      i <- i + 2
    Fin mientras
  Fin si
  Retorna p
Fin función
```

```
int primo(Entero_grande n)
{
  int p;
  Entero_grande i, s;

  p = (n % 2 != 0 || n == 2);

  if (p) {
    s = sqrt(n);

    for (i = 3; p && i <= s; i += 2)
      if (n % i == 0) p = 0;
  }

  return p;
}
```

- Se comprueba si un número impar es divisible por algún número inferior a él y distinto de 1
- El incremento de i es 2 porque nos saltamos los pares

Algoritmo Paralelo

- En el programa **primo_grande.c** vamos a buscar el número primo más grande que cabe en una variable entera sin signo de 8 bytes

- Vamos a paralelizar el bucle for de la función primo:

```
for (i = 3; p && i <= s; i += 2)
  if (n % i == 0) p = 0;
```

- No se puede hacer mediante #**pragma omp parallel for**. Tenemos un **condicional** en el bucle, no sabemos cuántas iteraciones se van a hacer. Si quitamos el condicional el algoritmo es **ineficiente**.

- Solución: región paralela con reparto **explícito** (manual)

Algoritmo Paralelo

- Repartimos manualmente las iteraciones del bucle for
- Modificar el índice inicial y el incremento para hacer un reparto cíclico
- Ejemplo de reparto cíclico para 3 hilos y 19 iteraciones:

Hilo 0	3	9	15
Hilo 1	5	11	17
Hilo 2	7	13	19

- Vamos a paralelizar el bucle for de l a función primo:

```
for (i = 3; p && i <= s; i += 2)
  if (n % i == 0) p = 0;
```

- Calcular la fórmula para sacar el **índice inicial y el incremento** (ayuda: el inicial depende del identificador del hilo, y el incremento del número total de hilos)
- Preparar el programa para que también muestre **número de hilos y tiempo** y probar comparando con el resultado del programa secuencial.
- Nota: Para que el algoritmo sea más eficiente, utilizar **volatile** int p

Contando Primos

- En el programa **primo_numeros.c** vamos a contar cuántos primos hay entre dos números
- Una versión paralela podría ser utilizar la función que hemos paralelizado anteriormente (muy ineficiente, sobretodo para números pequeños)
- Mejor opción: Utilizar la función **primos** secuencial y paralelizar el bucle for del main:

```
int main()
{
  Entero_grande i, n;

  n = 2; /* Por el 1 y el 2 */
  for (i = 3; i <= N; i += 2)
    if (primo(i)) n++;

  printf("Entre el 1 y el %llu hay %llu numeros primos.\n",
         N, n);

  return 0;
}
```

Contando primos

- Tomar tiempos de la versión secuencial
- Tomar tiempos de la versión paralela al menos con tres planificaciones: static, static con chunk 1, dynamic (recordatorio páginas 24 y 25 del Seminario 2)
- Para especificar la planificación hay dos opciones:
  1. En el propio código fuente:
     #pragma omp parallel for schedule(static, 1)
  2. En tiempo de ejecución. Para ello en el código ponemos #pragma omp parallel for **schedule(runtime)** y en la línea de ejecución lo especificamos:
     OMP_NUM_THREADS=4 OMP_SCHEDULE="static,6"./p
     Si elegimos esta opción, para static sin chunk pondremos:
     OMP_SCHEDULE="static,0"