Centro de Investigación en Matemáticas (CIMAT).
Unidad Monterrey
Programación y Análisis de Algoritmos

Tarea 1 de Cómputo Paralelo

Gustavo Hernández Angeles

3 de noviembre de 2024

1 Problema 1

Implementar de forma secuencial y en paralelo usando OpenMP la suma de los elementos de un vector de tamaño N.

SOLUCIÓN

En este ejercicio y en los posteriores utilizamos la libreria `iostream` para las funciones `cin` y `cout`, la librería `stdlib.h` para funciones como `malloc()` y `free()`, la librería `omp.h` para la paralelización y la librería `time.h` para la medición de los tiempos de ejecución. Además, la implementación de los problemas tanto de forma secuencial como en paralelo será muy similar, puesto que utilizamos siempre la directiva `for` de OpenMP, haciendo que el ciclo secuencial se ejecute de forma paralela.

La entrada de los programas podrá ser de manera automática (son entradas predeterminadas, utilizadas para la comparación de tiempos) o de forma manual por el usuario (que utilizaremos para verificar ejemplos). El resultado de los calculos se mostrarán en pantalla únicamente cuando la entrada sea manual.

Código paralelo

Se calcula la suma de los elementos del vector con un ciclo `for` en una sección paralela añadiendo la división de trabajo tipo `for` y la directiva `reduction(+ : suma)` para realizar la sumatoria. Además, se utilizan las directivas `private` y `shared` para declarar las variables privadas y compartidas para los hilos. La función toma de entrada el vector A y el tamaño N, se retorna el valor de la suma.

```
double suma_paralelo(double *A, long int N)
{
    double suma = 0;
    long int i;
#pragma omp parallel for shared(A, N) private(i) reduction(+ : suma)
    for (i = 0; i < N; i++)
    {
        suma += A[i];
    }
    return suma;
}
```

Código secuencial

Únicamente quitaremos la linea #pragma y el código se ejecutará de forma secuencial.

```
double suma_secuencial(double *A, long int N)
{
    double suma = 0;
    long int i;
    for (i = 0; i < N; i++)
    {
        suma += A[i];
    }
    return suma;
}
```

Ejemplos

A continuación, se comprueba mediante ejemplos el funcionamiento del código dadas varias entradas distintas.

```
Llenado manual (1) o automatico (0)?: 1
N: 10
Vector:
1 2 3 4 5 6 7 8 9 10
Resultado:
55
```

```
Llenado manual (1) o automatico (0)?: 1
N: 3
Vector:
5 5 5
Resultado:
15
```

```
Llenado manual (1) o automatico (0)?: 1
N: 4
Vector:
1 2 3 4
Resultado:
10
```

Comparación de velocidad

Para este problema, la prueba de rendimiento se hizo generando un vector de 10 millones de elementos de la siguiente forma:

```
N = 10'000'000;
A = (double *)malloc(N * sizeof(double));
for (i = 0; i < N; i++)
{
    A[i] = static_cast<double>(i);
}
```

Y se calcularon los tiempos mediante la libreria `time.h` midiendo los tiempos de inicio y final con la función `clock()`, y calculando su diferencia en segundos. El resultado para esta prueba se muestra cuando hacemos el llenado automático.

```
Llenado manual (1) o automatico (0)?: 0
Tiempo en paralelo: 0.0150 segundos.
Tiempo en secuencial: 0.0460 segundos.
```

El tiempo de mediciones varía en cada ejecución, pero se observa siempre que el tiempo de ejecución en paralelo es menor al secuencial.

2 Problema 2

Implementar de forma secuencial y en paralelo usando OpenMP la multiplicación de dos matrices cuadradas de tamaño $N \times N$.

SOLUCIÓN

Código paralelo

A diferencia del primer problema, aquí tendremos ciclos anidados, ya que para cada elemento sobre la matriz resultante debemos realizar una sumatoria, constituyendo 3 ciclos anidados. Utilizaremos `collapse(2)` para combinar dos ciclos anidados (sobre i y j), y dejaremos que cada hilo haga la sumatoria para asegurar que múltiples hilos no intenten acceder a un mismo elemento de la matriz. También definiremos manualmente las variables privadas y compartidas para cada hilo. Los índices se calculan a partir de la forma lineal en que creamos los arrays con `malloc(N*N*sizeof(float))`.

La función toma como entrada las matrices A y B a multiplicar, la matriz resultante C y el tamaño N de las matrices cuadradas. Se modifica el valor de C para obtener el resultado.

```
void multiplicacion_matriz_p(float *A, float *B, float *C, long int N)
  {
      long int i, j, k, indiceA, indiceB, indiceC;

  #pragma omp parallel for collapse(2) default(none) \
      shared(A, B, C, N) private(i, j, k, indiceA, indiceB, indiceC)
      for (i = 0; i < N; i++)
      {
          for (j = 0; j < N; j++)
          {
              indiceC = i * N + j;
              C[indiceC] = 0;
              for (k = 0; k < N; k++)
              {
                  indiceA = i * N + k;
                  indiceB = j + N * k;
                  C[indiceC] += A[indiceA] * B[indiceB];
              }
          }
      }
  }
```

Código secuencial

Se eliminan las directivas de OpenMP de la función anterior para realizar su versión secuencial.

```
  void multiplicacion_matriz_s(float *A, float *B, float *C, long int N)
  {
      long int i, j, k, indiceA, indiceB, indiceC;
      for (i = 0; i < N; i++)
      {
          for (j = 0; j < N; j++)
          {
              indiceC = i * N + j;
```

```
                C[indiceC] = 0;
                for (k = 0; k < N; k++)
                {
                    indiceA = i * N + k;
                    indiceB = j + N * k;
                    C[indiceC] += A[indiceA] * B[indiceB];
                }
            }
        }
    }
```

Ejemplos

Se prueba el resultado del código con distintas entradas.

```
Llenado manual (1) o automatico (0)?: 1
N: 2
Leemos la matriz A:
1 2
3 4
Leemos la matriz B:
5 6
7 8
Resultado:
19      22
43      50
```

```
Llenado manual (1) o automatico (0)?: 1
N: 3
Leemos la matriz A:
1 2 3
4 5 6
7 8 9
Leemos la matriz B:
9 8 7
6 5 4
3 2 1
Resultado:
30      24      18
84      69      54
138     114     90
```

Comparación de velocidad

La prueba consiste en realizar la multiplicación de dos matrices cuadradas $N \times N$ con $N = 1000$. Se realiza el llenado automáticamente mediante el siguiente código.

```
N = 1000;
A = (float *)malloc(N * N * sizeof(float));
B = (float *)malloc(N * N * sizeof(float));
for (i = 0; i < N; i++)
{
    for (j = 0; j < N; j++)
    {
        A[i * N + j] = (float)(i * j);
        B[i * N + j] = (float)(i + j);
```

```
        }
    }
```

Podemos ver los resultados ejecutando el programa con entrada automática. Podemos observar que el tiempo de ejecución en paralelo suele ser 10 veces menor al secuencial.

```
Llenado manual (1) o automatico (0)?: 0
Tiempo en paralelo: 0.471 segundos.
Tiempo en secuencial: 4.888 segundos.
```

3 Problema 3

Dado un vector de números reales V de tamaño N, implementar de forma secuencial y en paralelo usando OpenMP lo siguiente:

a) $S_1[i] = V[i] + V[i+1]$ para $i = 0, 1, ..., N-2$, con S_1 otro vector de tamaño $N-1$.

b) $S_2[i] = (V[i-1] + V[i+1])/2$ para $i = 1, 2, ..., N-2$, con S_2 otro vector de tamaño $N-2$.

SOLUCIÓN

Código paralelo

La programación de ambos problemas es directa mediante un ciclo `for` estableciendo las condiciones dadas en el problema. Se definen manualmente las variables compartidas y las privadas.

```
void inciso_a_p(float *V, float *A, long int N)
{
    long int i;

    #pragma omp parallel for default(none) shared(V, A, N) private(i)
    for (i = 0; i < N - 1; i++)
    {
        A[i] = V[i] + V[i + 1];
    }
}
```

```
void inciso_b_p(float *V, float *A, long int N)
{
    long int i;
    #pragma omp parallel for default(none) shared(V, A, N) private(i)
    for (i = 1; i < N - 1; i++)
    {
        A[i - 1] = (V[i - 1] + V[i + 1]) / 2;
    }
}
```

Código secuencial

La versión secuencial de las funciones paralelizadas únicamente no incluyen la línea para abrir la región paralela.

```
void inciso_a_s(float *V, float *A, long int N)
{
    long int i;
    for (i = 0; i < N - 1; i++)
    {
        A[i] = V[i] + V[i + 1];
    }
}
```

```
void inciso_b_s(float *V, float *A, long int N)
{
    long int i;
    for (i = 1; i < N - 1; i++)
    {
        A[i - 1] = (V[i - 1] + V[i + 1]) / 2;
    }
}
```

Ejemplos

Se realizan varias pruebas para comprobar el funcionamiento del código.

```
Llenado manual (1) o automatico (0)?: 1
N: 5
Vector V:
1 2 3 4 5
        INCISO A)
Resultado:
3 5 7 9
        INCISO B)
Resultado:
2 3 4
```

```
Llenado manual (1) o automatico (0)?: 1
N: 7
Vector V:
7 14 21 28 35 42 49
        INCISO A)
Resultado:
21 35 49 63 77 91
        INCISO B)
Resultado:
14 21 28 35 42
```

Comparación de velocidad

La prueba de rendimiento se hace con un array de con 100 millones de elementos, generado de la siguiente forma. Los tiempos se calculan exactamente igual que en los problemas anteriores, los resultados no son diferentes a los anteriores; la ejecución en paralelo es más eficiente, aunque en este caso, la diferencia no es muy grande.

```
Llenado manual (1) o automatico (0)?: 0
        INCISO A)
Tiempo en paralelo: 0.04 segundos.
Tiempo en secuencial: 0.14 segundos.
        INCISO B)
Tiempo en paralelo: 0.042 segundos.
Tiempo en secuencial: 0.253 segundos.
```

4 Problema 4

Dadas dos matrices A y B de tamaño $N \times M$ con valores enteros positivos, programar lo siguiente:

a) $C_1(i,j) = A(i,j) + B(N-i-1, M-j-1)$, para $i = 0, 1, ..., N-1$ y $j = 0, 1, ..., M-1$, con C_1 otra matriz de tamaño $N \times M$.

b) $C_2(i,j) = \alpha A(i,j) + (1-\alpha)B(i,j)$, para $i = 0, 1, ..., N-1$ y $j = 0, 1, ..., M-1$, con $\alpha \in (0,1)$ y C_2 otra matriz de tamaño $N \times M$.

SOLUCIÓN

Código paralelo

Al igual que en el problema 2 utilizaremos la directiva `collapse(2)` al tener dos ciclos anidados y sin peligros de que múltiples hilos modifiquen la misma unidad de memoria. Las funciones toma como entrada las matrices A, B y C en forma de punteros, junto a sus tamaños N y M. Adicionalmente, para el inciso b) toma como entrada también al valor α introducido por el usuario.

```
void incisoA_paralelo(unsigned int *A, unsigned int *B,
                      float *C, long int N, long int M)
{
    long int i, j, idx, idx_b;

#pragma omp parallel for collapse(2) default(none) \
    shared(A, B, C, N, M) private(i, j, idx, idx_b)
    for (i = 0; i < N; i++)
    {
        for (j = 0; j < M; j++)
        {
            idx = i * M + j;
            idx_b = (N - i - 1) * M + (M - j - 1);
            C[idx] = A[idx] + B[idx_b];
        }
    }
}
```

```
void incisoB_paralelo(unsigned int *A, unsigned int *B,
                      float *C, float a, long int N, long int M)
{
    long int i, j, idx;

#pragma omp parallel for default(none) \
    shared(A, B, C, N, M, a) private(i, j, idx)
    for (i = 0; i < N; i++)
    {
        for (j = 0; j < M; j++)
        {
            idx = i * M + j;
            C[idx] = a * A[idx] + (1 - a) * B[idx];
        }
    }
}
```

Código secuencial

Eliminamos la línea de OpenMP que abre la región paralela.

```
void incisoA_secuencial(unsigned int *A, unsigned int *B,
                        float *C, long int N, long int M)
{
    long int i, j, idx, idx_b;

    for (i = 0; i < N; i++)
    {
        for (j = 0; j < M; j++)
        {
            idx = i * M + j;
            idx_b = (N - i - 1) * M + (M - j - 1);
            C[idx] = A[idx] + B[idx_b];
        }
    }
}
```

```
void incisoB_paralelo(unsigned int *A, unsigned int *B,
                      float *C, float a, long int N, long int M)
{
    long int i, j, idx;

#pragma omp parallel for default(none) \
    shared(A, B, C, N, M, a) private(i, j, idx)
    for (i = 0; i < N; i++)
    {
        for (j = 0; j < M; j++)
        {
            idx = i * M + j;
            C[idx] = a * A[idx] + (1 - a) * B[idx];
        }
    }
}
```

Ejemplos

Se prueba el código dadas distintas entradas.

```
Llenado manual (1) o automatico (0)?: 1
N x M: 2 3
Matriz A:
1 2 3
4 5 6
Matriz B:
6 5 4
3 2 1
Alfa: 0.5
        INCISO A)
Resultado:
2       4       6
8       10      12
        INCISO B)
Resultado:
3.5     3.5     3.5
3.5     3.5     3.5
```

```
Llenado manual (1) o automatico (0)?: 1
N x M: 2 2
Matriz A:
10 20
30 40
Matriz B:
40 30
20 10
Alfa: 0.7
        INCISO A)
Resultado:
20      40
60      80
        INCISO B)
Resultado:
19      23
27      31
```

Comparación de velocidad

Se realiza la prueba de rendimiento con matrices cuadradas con dimensiones $N = M = 10,000$ y con un valor de $\alpha = 0.5$ definido de la siguiente manera:

```
long int i, j;
N = M = 10000;
A = (unsigned int *)malloc(N * M * sizeof(unsigned int));
B = (unsigned int *)malloc(N * M * sizeof(unsigned int));
for (i = 0; i < N; i++)
{
    for (j = 0; j < N; j++)
    {
        A[i * N + j] = (float)(i * j);
        B[i * N + j] = (float)(i + j);
    }
}
a = 0.5;
```

Las mediciones nos muestran, una vez más, que la ejecución en paralelo es mucho más eficiente que la secuencial alcanzando una reducción del tiempo en casi 10 veces.

```
Llenado manual (1) o automatico (0)?: 0
        INCISO A)
Tiempo en paralelo: 0.145 segundos.
Tiempo en secuencial: 1.136 segundos.
        INCISO B)
Tiempo en paralelo: 0.263 segundos.
Tiempo en secuencial: 2.492 segundos.
```