Práctica de OpenMP

Tecnología de la Programación Paralela (TPP)

José E. Román

Esta práctica se compone de tres partes. La primera está orientada a la paralelización de bucles en el caso de una computación matricial muy regular, mientras que en la otras dos se aborda un problema más irregular (paralelismo de tareas).

Producto de Matrices

Dadas dos matrices cuadradas, $A, B \in \mathbb{R}^{n\times n}$, su producto se define como $C = AB$, de forma que $c_{ij} = \sum_{k=1}^{n} a_{ik}b_{kj}$. El archivo `matmat.c` contiene un programa en lenguaje C que crea dos matrices aleatorias e invoca a la función `matmat` que las multiplica.

E1 Paralelización de Bucles

Tomando como base la versión secuencial, paralelizar el programa del producto de matrices utilizando directivas OpenMP. Incluir en el programa instrucciones para mostrar el tiempo empleado en la realización del cálculo, mediante `omp_get_wtime()` (ver `omp.h`).

Para ejecutar con varios hilos, se puede hacer por ejemplo:

```
$ OMP_NUM_THREADS=4 ./pmatmat 800
```

En este ejercicio, se pretende paralelizar a nivel de bucle. Dado que hay tres bucles anidados, probar diferentes posibilidades. Modificar el programa para que muestre el número de hilos empleados. Ejecutar para diferente número de hilos y comentar la mejora obtenida. Utilizar tallas del problema suficientemente grandes para que los tiempos sean significativos. También es conveniente hacer varias repeticiones del cálculo.

Nota: Si se dispone de un compilador con soporte para OpenMP 3.0 (por ejemplo, GCC 4.4 o superior), realizar alguna prueba con la cláusula `collapse`.

Fractales

El programa contenido en el directorio `fractal` genera fractales de Mandelbrot a partir de los datos proporcionados. Para que el cálculo no sea excesivamente rápido, hay que escoger los parámetros adecuadamente, por ejemplo:

```
$ ./mandel -width 1024 -height 768 -ulx -1.37 -uly 0.04 -lly -0.01
$ ./mandel -width 1024 -height 768 -ulx -1.37 -uly 0.014 -lly 0.0135
$ ./mandel -width 1024 -height 768 -ulx -1.369538 -uly 0.013797 -lly 0.01368
```

Si fuera necesario, se puede incrementar la resolución de la imagen generada.

E2 Paralelización con Regiones Paralelas

Paralelizar el programa utilizando regiones paralelas y construcciones de reparto de trabajo. Realizar pruebas de prestaciones y probar diferentes estrategias de planificación (cláusula `schedule`). Se recomienda utilizar `schedule(runtime)` en combinación con la variable de entorno `OMP_SCHEDULE`.

E3 **Paralelización con Colas de Tareas (opcional)**

Los archivos del directorio `taskqueue` contienen un programa en C++ con una clase `TaskQueue` que implementa una cola de tareas paralelas representada internamente como una lista enlazada de objetos `Task`. La clase `Task` es una clase abstracta con un método `execute()`, de la cual se pueden derivar los diferentes tipos de tareas.

La clase derivada `SimpleTask` es un ejemplo muy básico. También se proporciona la clase `MandelTask`, para generar los fractales de Mandelbrot.

Deberá paralelizarse el programa y utilizarse directivas de sincronización de OpenMP para la implementación de la clase `TaskQueue`.

Ampliación: Probar el caso en que el método `execute()` contenga alguna directiva `parallel`. En este caso, analizar el comportamiento del programa cuando se activa o desactiva el paralelismo anidado y la creación dinámica de hilos.

Nota: Si se dispone de un compilador con soporte para OpenMP 3.0 (por ejemplo, GCC 4.4 o superior), realizar alguna prueba con la directiva `task`.

Paralelismo de tareas con Quark

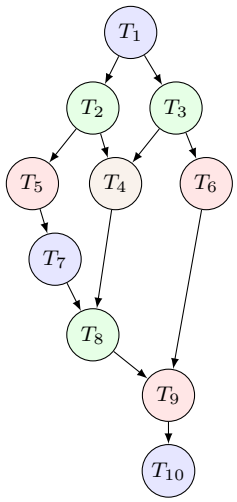
A partir de OpenMP 4.0 se ofrece soporte para activar tareas en base a las dependencias explícitas en el grafo de tareas. Como alternativa a OpenMP se puede usar Quark para esto.

E4 **Cholesky paralelo a partir del grafo de dependencias**

A continuación se muestra un ejemplo de cálculo orientado a bloques de la factorización de Cholesky y el grafo de dependencia de tareas asociado.

$$\begin{bmatrix} A_{11} & & \text{sim} \\ A_{21} & A_{22} & \\ A_{31} & A_{32} & A_{33} \end{bmatrix} = \begin{bmatrix} L_{11} & & \\ L_{21} & L_{22} & \\ L_{31} & L_{32} & L_{33} \end{bmatrix} \begin{bmatrix} L_{11}^T & L_{21}^T & L_{31}^T \\ & L_{22}^T & L_{32}^T \\ & & L_{33}^T \end{bmatrix}$$

$$\begin{array}{lllll} k=1 & A_{11} \leftarrow \text{chol}(A_{11}) & & & \\ & j=2 & A_{21} \leftarrow A_{21}A_{11}^{-T} & & \\ & j=3 & A_{31} \leftarrow A_{31}A_{11}^{-T} & & \\ & j=2 & i=3 & A_{32} \leftarrow A_{32} - A_{31}A_{21}^T & \\ & & & A_{22} \leftarrow A_{22} - A_{21}A_{21}^T & \\ & j=3 & & A_{33} \leftarrow A_{33} - A_{31}A_{31}^T & \\ k=2 & A_{22} \leftarrow \text{chol}(A_{22}) & & & \\ & j=3 & A_{32} \leftarrow A_{32}A_{22}^{-T} & & \\ & j=3 & & A_{33} \leftarrow A_{33} - A_{32}A_{32}^T & \\ k=3 & A_{33} \leftarrow \text{chol}(A_{33}) & & & \end{array}$$



El programa `chol.c` implementa la versión secuencial. Hay que tener en cuenta que para que el cálculo a bloques sea eficiente es necesario que cada bloque esté almacenado de forma contigua en memoria, por eso se realiza una conversión en el formato de almacenamiento. El objetivo del ejercicio es desarrollar una versión paralela con Quark y analizar sus prestaciones, variando el tamaño de la matriz, el tamaño de bloque y el número de hilos.