UTEC

Reporte Unidad 6

Programacion orientada a objetos 2

Alejandro Goicochea
alejandro.goicochea@utec.edu.pe

Profesor:
Ruben Rivas

June 26, 2019

1 Introducción

En este ejercicio se estuvo trabajando con la programacion concurrente usando la libreria $< thread >$ de C++. Se tiene como objetivo comparar el tiempo de ejecucion de la multiplicacion de dos matrices con dos clases diferentes. Una tiene realiza la multiplicacion de manera secuencial y la otra de manera concurrente.

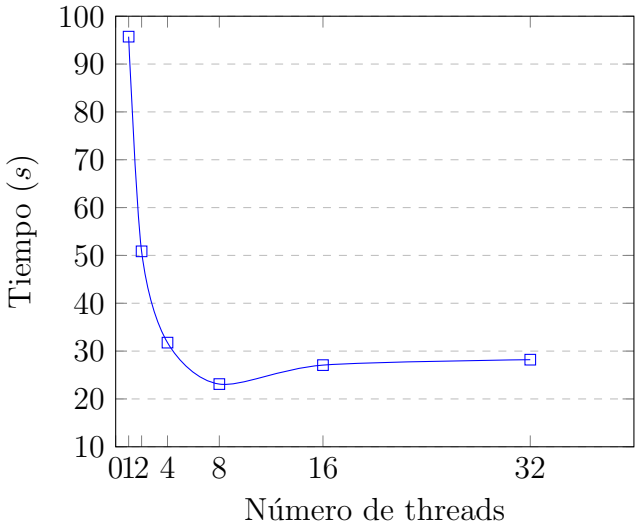
2 Implementación

Se crean dos clases *matriz_reg* y *matriz_thread* ambas con variables n y m que describen la cantidad de filas y columnas de su matriz, respectivamente. Ambas tambien cuentan con metodos para llenar su matriz de diferentes maneras. En el constructor de ambas se genera la matriz con los tama;os indicados en los parametros y se guarda en su variable matriz, un arreglo de tipo $int **$. Para la simplificacion del programa, se uso dos matrices del mismo tama;o (2000 x 2000) y el programa se ejecuto en un procesador Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz con 4 threads y la capacidad de hyperthreading. Para el benchmark se so la funcion *clock*() que mide el tiempo total del CPU. Este metodo para medir el tiempo considera el tiempo que tomo con todos los threads y devuelve el tiempo en la unidad *clock_t* por lo que dividimos nuestro resultado de *inicio* - fin por la cantidad de clocks por segundo almacenado en la variable $CLOCKS\_PER\_SEC$ multiplicado por la cantidad de threads que usamos (si la cantidad de threads usados es mayor a 8, se utiliza 8 ya que el procesador en el que se ejecuto el programa solo tiene esa capacidad). El tiempo de ejecucion de la clase secuencial es el valor al ejecutar el programa con un solo thread.

3 Resultados

Al correr el programa el tiempo de ejecución empezó alto y fue bajando hasta llegar a los 8 threads e incrementó nuevamente al agregar más llegando a un valor que parace ser asintótico.

Tiempo de ejecución de multiplicación de matrices



En este gráfico se ve claramente como el tiempo de ejecución baja hasta llegar a los 8 threads y luego incrementa de nuevo llegando a una asíntota cercana a los 28 segundos. De estos gráficos podemos concluir que el número óptimo de threads para nuestro programa va a ser 8 lo que coincide con el número de threads que tiene nuestro procesador. Es probable que el número óptimo de threads que se debe crear en un programa sea igual a la cantidad de threads del procesador ya que si se crean más threads de los que tiene el procesador van a haber threads esperando a que se libere otro thread del procesador. Si se crea la cantidad de threads que tiene el procesador, se esta repartiendo las instrucciones del programa entre todos los threads por lo que al terminar la ejecución de estos threads se termina la ejecución del programa ya que no había threads esperando para ejecutarse. Este hallazgo concuerda con papers scientificos tal como $ThreadCountPredictionModel$: $DynamicallyAdjustingThreadsforHeterogeneousMany-CoreSystems$ donde se muestra a traves de la ley de Amdahl que la cantidad optima de threads para un proceso es la cantidad de threads que puede correr el procesador de tu máquina simultaneamente. Crear mas threads que esto genera una clase de "bottleneck" por lo que un thread estaria en espera hasta que un thread del procesador se libere.