Documento de análisis
Ejercicio de perfilado
Multiplicación de matrices con concurrencia

Presentado por: Juan Camilo Rojas Cortés

El objetivo de la presente práctica fue analizar detenidamente el rendimiento de las implementaciones concurrentes del algoritmo de multiplicación de matrices hechas hasta el momento en la asignatura utilizando el método USE. Este método analiza 3 elementos fundamentales a la hora de ver el desempeño de un sistema: Utilización, saturación y errores. Para la presente práctica se hizo énfasis en el método USE aplicado al procesador, ya que es el componente que más se utiliza al aplicarle concurrencia a la multiplicación de matrices y que más podría variar.

Utilización:

Se refiere al tiempo en el que el programa en ejecución tiene ocupado el recurso en cuestión (en este caso el procesador). Dependiendo de la métrica que se utilice, esta variable puede medirse a través de un porcentaje, en el que un valor de 100% indica que el programa en ejecución está utilizando todo el recurso. Para medir este indicador se utilizó el comando de Linux *pidstat 1*. Este muestra, entre otras características, el porcentaje de CPU utilizado por cada proceso del sistema operativo.

Saturación:

La saturación se refiere a la cantidad de trabajo extra que debe realizar el recurso (procesador en este caso) cuando ejecuta un proceso del sistema operativo. Para la presente práctica este indicador se midió con el comando de Linux *dstat -p*. En este, la columna “run” indica el nivel de saturación del procesador. Cuando este número es mayor a la cantidad de núcleos de procesamiento (4 en el caso del equipo en el que se realizaron las pruebas), quiere decir que el procesador tiene un nivel de saturación alto.

Errores:

Este indicador no fue medido en la presente práctica porque no se halló documentación suficiente respecto a cómo se puede medir a través del comando perf (recomendado en la documentación del méetodo USE en linux: http://www.brendangregg.com/USEmethod/use-linux.html).

A continuación, se pasará a analizar el rendimiento con el método USE en cada una de las implementaciones elegidas. Para probar estos indicadores, se ejecutó el algoritmo en cuestión utilizando matrices de 2000 filas y 2000 columnas. De esta forma se tiene un reto que consume exactamente la misma memoria y que puede ser comparable en su complejidad.

Concurrencia con hilos:

El primer método de concurrencia analizado fue la implementación de hilos en el procesador. Este método fue explicado en la primera práctica del presente curso y, si se le compara con el resto de

implementaciones hechas, es el que tiene un speed up rate menor al resto. A continuación se mostrarán los resultados de rendimiento obtenidos:

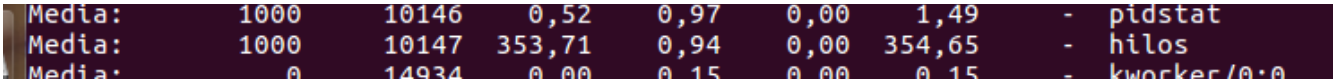

Figura 1. Utilización de CPU en la implementación con hilos.

En la Figura 1 solamente se muestran los resultados de un solo proceso. Esto sucede porque todos los hilos de procesamiento se ejecutan sobre el mismo proceso visible para el sistema operativo, pues es el mismo procesador el que se encarga de administrar la paralelización. Como se puede ver en la Figura 1, la ejecución de este programa consume el 354,65% de la capacidad de la CPU. Esta capacidad que supera al máximo se puede ver reflejada también en los niveles de saturación calculados con el comando *dstat -p*:

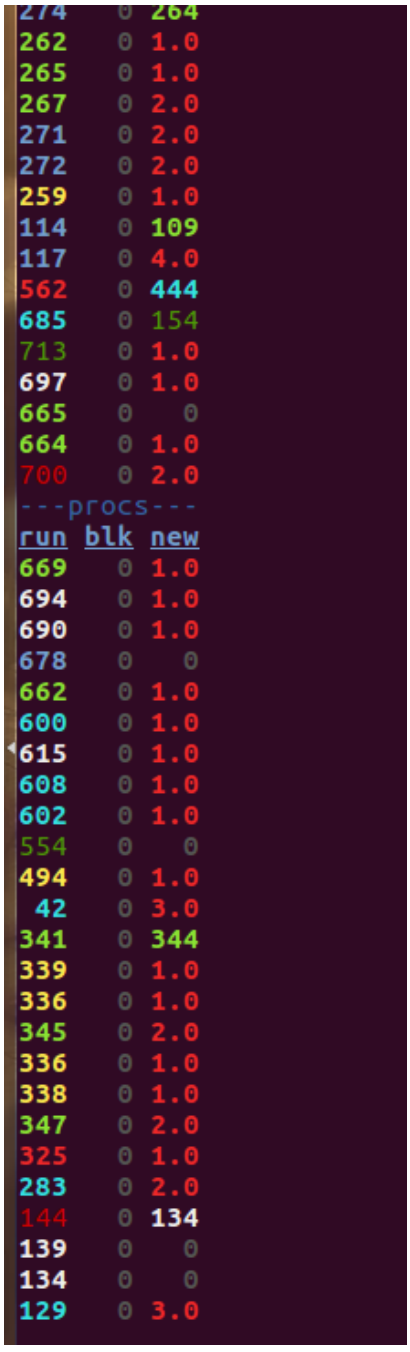

Figura 2. Niveles de saturación de la implementación con hilos calculada con el comando *dstat -p*

Como se puede ver en la Figura 2, los niveles de saturación del procesador cuando se implementa la multiplicación de matrices con hilos es predominantemente alto, mucho más si se tiene en cuenta que, para este caso, el valor normal de la columna “run” (columnda de la izquierda) es de 4, porque la CPU tiene 4 núcleos de procesamiento. Por lo tanto, es posible afirmar que cuando se ejecuta este tipo de concurrencia con muchos hilos de procesamiento (2000 en este caso), el procesador se exige a una capacidad que está por encima de lo normal. Esto se ve reflejado, por ejemplo, en que cuando el programa se está ejecutando el sistema operativo empieza a tener una respuesta característicamente lenta a las órdenes que se le dan. Esto no pasa con otras implementaciones.

Implementación con procesos sin Open MP:

En esta implementación (explicada en una de las entregas anteriores), cada hilo de procesamiento es ejecutado por un proceso independiente del sistema operativa, por lo que hay que configurar la memoria compartida a través de código. De todas las implementaciones hechas hasta ahora, esta es la que tiene un Speed Up Rate mayor. A continuación, se analizará su rendimiento utilizando el método USE:

```
20:40:22     1000     12342    0,00    4,00    0,00    4,00     3  pidstat
20:40:22     1000     12347   97,00    3,00    0,00  100,00     0  procesos
20:40:22     1000     12626   10,00    0,00    0,00   10,00     1  procesos
20:40:22     1000     12627    8,00    0,00    0,00    8,00     3  procesos
20:40:22     1000     12628    8,00    0,00    0,00    8,00     1  procesos
20:40:22     1000     12629    8,00    0,00    0,00    8,00     3  procesos
20:40:22     1000     12630    9,00    0,00    0,00    9,00     1  procesos
20:40:22     1000     12631    8,00    0,00    0,00    8,00     2  procesos
20:40:22     1000     12632    9,00    0,00    0,00    9,00     1  procesos
20:40:22     1000     12633    9,00    0,00    0,00    9,00     3  procesos
20:40:22     1000     12634    8,00    0,00    0,00    8,00     1  procesos
20:40:22     1000     12635    8,00    0,00    0,00    8,00     2  procesos
20:40:22     1000     12636    6,00    0,00    0,00    6,00     1  procesos
```

Figura 3. Utilización de CPU en la paralelización con procesos.

En técnica de concurrencia se genera un proceso por cada fila de la matriz, por lo que en este caso son 2000 procesos independientes para el sistema operativo. Estos no alcanzan a verse completos en la Figura 3, pero sirve como ilustración de lo que sucede. Como se puede ver, hay un proceso que ocupa el 100% de uno de los núcleos de procesamiento (el núcleo 0). El resto de procesos se reparten entre los 3 núcleos restantes y consumen porcentajes pequeños (10% como mucho). Al igual que en la implementación con hilos, aquí se utiliza el procesador por encima de su capacidad. Sin embargo, la diferencia está en la saturación:

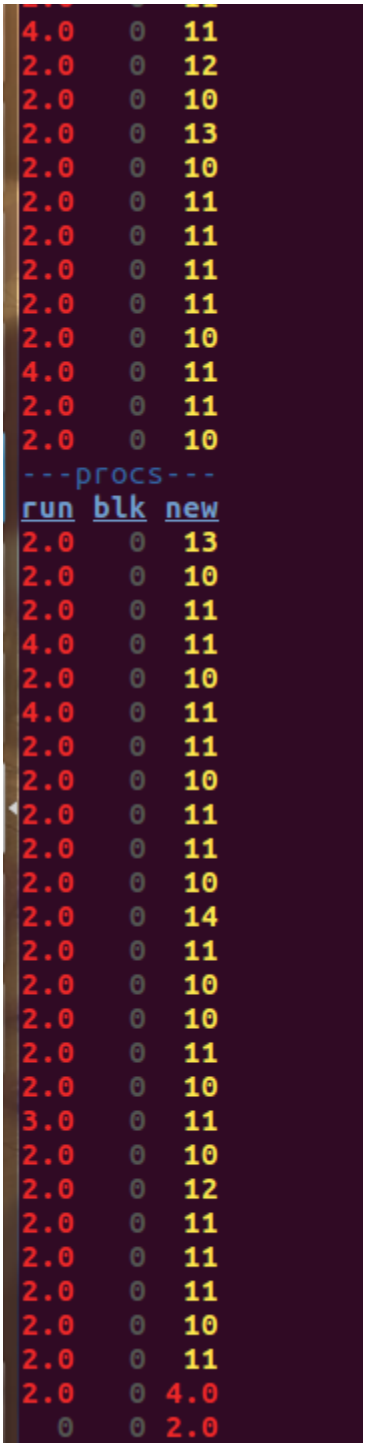

Figura 4. Saturación de CPU en la paralelización con procesos sin Open MP.

Como se puede ver en la figura 4, cuando se implementa paralelización por procesos, la saturación es muchísimo menor que en la paralelización por hilos. De hecho, en todas las instancias esta se encuentra por debajo de 4 (número de núcleos de procesamiento), por lo que es posible afirmar que la CPU no se está sobrecargando de esfuerzo en ningún momento. Esto se ve traducido en un Speed Up Rate mayor y en una mejor respuesta del sistema operativo mientras se está ejecutando el programa.

La saturación de la CPU es notoriamente menor cuando se paraleliza por procesos que cuando se paraleliza con hilos. Esto puede deberse a que, como la ejecución de cada hilo de procesamiento es manejada por el sistema operativo (en el caso de los procesos), estos se reparten de forma tal que no representen una carga mayor para el procesador.

Paralelismo con procesos utilizando Open MP

La tercera implementación analizada fue aquella en la que se utiliza paralelización por procesos haciendo uso de la librería Open MP. Aquí se genera un proceso administrado por el sistema operativo, pero todo esto es transparente al programador, pues es realizado por la librería. En esta implementación hay un Speed Up Rate menor que la paralelización por procesos pura, pero mayor que en la paralelización por hilos. A continuación se presenta el análisis:

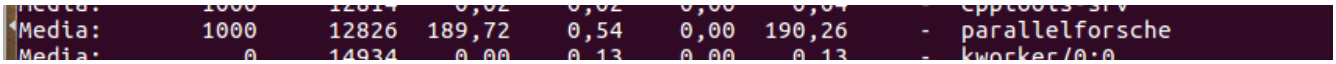

Figura 5. Utilización de CPU en la paralelización con OMP.

En la Figura 5 solamente se ven los resultados de un proceso. Esto ocurre porque, si bien el algoritmo se ejecuta concurrentemente en diferentes procesos, para el sistema operativo solamente es visible uno de ellos, pues toda la gestión de los procesos es realiada por la librería OMP. Como se puede ver en la figura 1, este algoritmo está utilizando el procesador a una capacidad del 190%. Si bien es cierto que es una utilización alta (mayor a lo que el procesador está acostumbrado), esta es menor que en la implementación con hilos. Esto es reforzado por el análisis de la saturación:

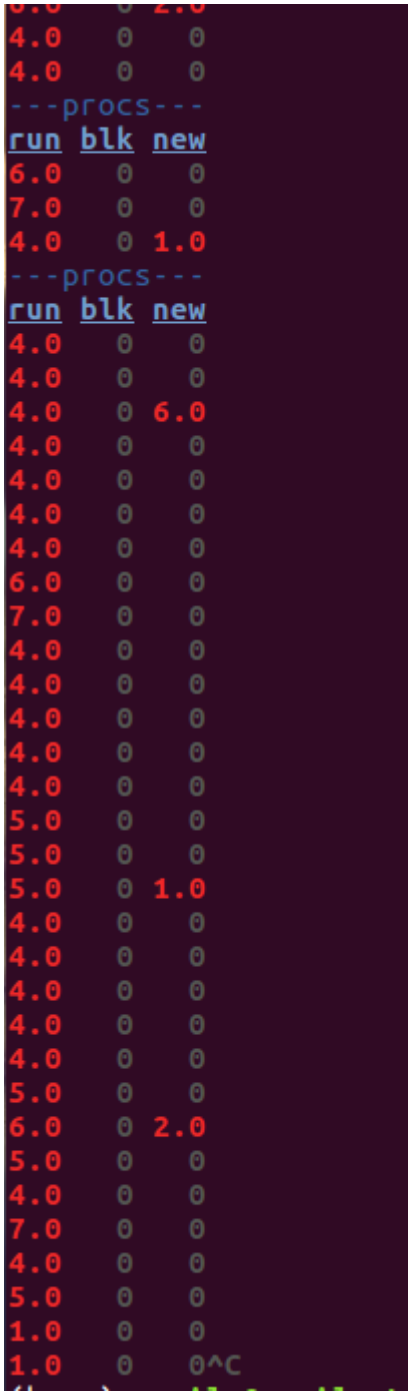

Figura 6. Saturación de CPU al utilizar paralelización por procesos con OMP.

Como se puede ver, la saturación al usar OMP es ligeramente mayor que cuando se hace concurrencia por procesos sin usar esta librería. Esto se ve reflejado en un Speed Up Rate menor, pero mucho mejor que en la concurrencia por hilos.