PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN

IIC2343 - Arquitectura de Computadores
Marzo 2022

Entrega el 17 de junio hasta las 23:59

Tarea 3

Objetivos

Para esta tarea, esperamos que apliquen lo aprendido sobre paralelización y coherencia de memoria caché, junto que experimenten mejorando el rendimiento de un programa a través de la paralelización de este.

Enunciado

Parte 1: Paralelize!

Existen librerías de código que tienen un muy mal rendimiento en términos de tiempo. Una forma fácil de conseguir una mejora sustancial en rendimiento se logra paralelizando procesos, que en muchos casos es relativamente fácil de lograr, sobre todo en casos en que se debe iterar sobre un arreglo, y se realizan operaciones que son independientes entre ellas. Para esto, se subirá al repositorio un script escrito en Python que deberán paralelizar con la librería Numba (`pip install numba`), y luego deberán contrastar el tiempo de ejecución del programa no paralelizado y el programa paralelizado. A modo de apoyo, también se subirá un script en que se explica a grandes rasgos como utilizar la librería Numba, con un ejemplo de una función paralelizada y su versión no paralelizada. Como la librería paraleliza a nivel de CPU, dependiendo del computador en que ejecuten su tarea, el factor en el que disminuya su tiempo de ejecución podría variar.

2 puntos.

Evaluación parte 1

Se entregará 0.5 puntos por entregar un script que ejecuta sin problemas e importa la librería Numba, 1 punto por paralelizar adecuadamente el programa, demostrando una mejora en el runtime, y 0.5 puntos por no cambiar el resultado de las operaciones realizadas, respecto a la versión no paralelizada del programa.

Parte 2: Simparallel

Un aspecto importante de los sistemas paralelos es mantener la coherencia entre el cache L1, L2 y memoria, en sistemas con paralelismo a nivel de chip. Para familiarizarse con estos sistemas, deberán escribir un programa en Python capaz de detectar cuándo ha ocurrido un error de consistencia en la memoria caché. El programa recibirá como input un archivo `.csv` donde las primeras 4 columnas representan a cada procesador, y a una secuencia de instrucciones que ejecuta en ella. Las siguientes columnas representan a direcciones de memoria, y estas tienen el valor inicial y final de dicha dirección luego de la ejecución del programa, en su

primera y segunda fila. Su programa deberá leer este archivo y verificar que el resultado presente en cada dirección sea el correcto para un sistema que mantiene consistencia de caché, de lo contrario, deberá indicar en que linea(s) se perdió la coherencia del caché. Por ejemplo, en el siguiente caso, hay dos direcciones de memoria que inician en 0 y su valor final aparece anotado como 1, y el procesador 2 no ejecuta ninguna tarea.

```
P0;                 P1;                 P2;         P3;                    0x000; 0x0001;
MOV A, 1;           MOV A, (0x0000);    NOP;        MOV A, (0x0001);       0;     0;
MOV B, 1;           MOV B, 1;           NOP;        MOV B, 1;              1;     1;
NOP;                ADD A, B;           NOP;        ADD A, B;;;
MOV A, (0x0001);    NOP;                NOP;        NOP;;;
ADD A, B;           NOP;                NOP;        NOP;;;
```

El valor final para la dirección no es correcto, y esto error se debe haber producido cuando `P0` llamó el valor de dicha dirección, pero recibió 0 en lugar de 1. El valor esperado para ambas direcciones al finalizar esta ejecución era `0x0000=1;0x0001=2`.

Pueden asumir que los archivos `.csv` con las descripciones de las ejecuciones nunca presentarán *race-conditions*, en las que dos o más procesadores intentan modificar o acceder a un mismo recurso a la vez y los tests tampoco serán ambiguos, es decir, habrá a lo más un único error por dirección y este se producirá en una única llamada a memoria. Además se irán subiendo más ejemplos a lo largo del plazo para tarea.

4 puntos.

Evaluación parte 2

Se evaluará 1 punto por entregar un script que corre sin problemas, lee el archivo y entrega algún tipo de output. Los restantes 3 puntos corresponderán a 6 tests, en que cada uno vale 0.5 puntos, y se entregarán 0.25 puntos por resultados parcialmente correctos.

Nota tarea

La nota se calculará de la siguiente manera:

$$Puntos + 1 = Nota\ tarea$$

La nota de la tarea se redondea al primer decimal.

Entrega

La fecha de entrega es el viernes 17 de junio, hasta las 23:59 por Canvas. El formato a entregar será un archivo `.py` por item, con sus respectivas soluciones para cada item de la tarea.