Universidad Nacional de San Agustín
Facultad de Producción y Servicios
Escuela Profesional Ciencia de la Computación

Laboratorio 01

17 de septiembre de 2023

Docente:
Alvaro Henry Mamani Aliaga

Curso:
Computacion Paralela y Distriuida

Estudiante:
Angie Alexandra Pino Huarsaya

Índice general

0.1. Introducción

En este informe, analizaremos el funcionamiento y el rendimiento de dos pares de bucles anidados. Además tambien veremos como el diseño de acceso a memoria afecta al rendimiento debido a la jerarquía de caché de la CPU.

0.2. Implementación

En el siguiente código veremos dos pares de bucles anidados que operan en una matriz bidimensional 'A', un vector 'x', y un vector 'y'.

El primer par de bucles accede a la matriz 'A' iterando filas por columnas, mientras que el segundo par de bucles accede a la matriz 'A' iterando columnas por filas.
El código es el siguiente:

```
// Primer par de bucles
    for (i = 0; i < MAX; i++) {
        for (j = 0; j < MAX; j++) {
            y[i] += A[i][j] * x[j];
        }
    }

// Segundo par de bucles
    for (j = 0; j < MAX; j++) {
        for (i = 0; i < MAX; i++) {
            y[i] += A[i][j] * x[j];
        }
    }
```

0.3. Resultados

Para entender mejor esto, supongamos que MAX es cuatro y los elementos de 'A' están almacenados en la memoria de la siguiente manera:

Cache Line	Elements of A			
0	A[0][0]	A[0][1]	A[0][2]	A[0][3]
1	A[1][0]	A[1][1]	A[1][2]	A[1][3]
2	A[2][0]	A[2][1]	A[2][2]	A[2][3]
3	A[3][0]	A[3][1]	A[3][2]	A[3][3]

Figura 1: Almacenamiento de elementos de A en caché

Supongamos que ninguno de los elementos de A está en el caché cuando cada par de los bucles comienzan a ejecutarse. Supongamos también que una línea de caché consta de cuatro elementos. Finalmente, supondremos que el caché está mapeado directamente y solo puede almacenar ocho elementos de A, o dos líneas de caché.

Ambos pares de bucles intentan acceder primero a A[0][0], como no está en el caché, esto resultará en una pérdida de caché y el sistema leerá la línea que consta de la primera fila de A, A[0][0], A[0][1], A[0] [2], A[0][3], en el caché.

0.3.1. Primer par de bucles

El primer par de bucles accede a A[0][1], A[0][2], A[0][3], todos los cuales están en la memoria caché, y el siguiente error en el primer par de bucles ocurre cuando el código accede a A[1][0].
Siguiendo de esta manera, vemos que el primer par de bucles dará como resultado un total de cuatro fallos cuando acceda a los elementos de A, uno para cada fila. Dado que nuestro caché hipotético solo puede almacenar dos líneas u ocho elementos de A, cuando leemos el primer elemento de la fila dos y el primer elemento de la fila tres, una de las líneas que ya está en el caché tendrá que ser desalojada del caché, pero una vez que se expulsa una línea, el primer par de bucles no necesitará acceder a los elementos de esa línea nuevamente.

0.3.2. Segundo par de bucles

Después de leer la primera fila en el caché, el segundo par de bucles debe acceder a A[1][0], A[2][0], A[3][0], ninguno de los cuales está en el caché. Entonces los próximos tres accesos de A también resultarán en errores. Además, debido a que el caché es pequeño, las

lecturas de A[2][0] y A[3][0] requerirán que se expulsen las líneas que ya están en el caché.

Dado que A[2][0] está almacenado en la línea de caché 2, leer su línea desalojará la línea 0, y leer A[3][0] desalojará la línea 1. Después de terminar el primer paso por el bucle externo, A continuación, deberá acceder a A[0][1], que fue desalojado con el resto de la primera fila. Entonces vemos que cada vez que leemos un elemento de A, fallaremos, y el segundo par de bucles resulta en 16 errores.

0.4. Análisis

Ejecutando el código con MAX = 1000, el primer par de bucles anidados es aproximadamente tres veces más rápido que el segundo par.

El primer par de bucles tiene un mejor rendimiento porque accede a los elementos de 'A' en un patrón contiguo de fila principal, lo que aprovecha la localidad espacial. El segundo par de bucles tiene un peor rendimiento debido al patrón de acceso no contiguo de columna principal, lo que resulta en una mayor cantidad de fallos de caché.

Como conclusión podemos decir que el diseño del acceso a memoria en el código puede tener un impacto significativo en el rendimiento debido a la jerarquía de caché de la CPU. Conociendo el principio de espacialidad y la localidad temporal nos permite tener cierto control indirecto sobre el almacenamiento en caché. Es importante optimizar el acceso a datos para aprovechar esta localidad espacial y temporal y así minimizar los fallos de caché.

El código lo puedes encontrar en mi repositorio en github:
https://github.com/GiedraAlexandra19/CPyD/tree/main/LAB01