Informe de Memoria Cache en Multiplicación de Matriz

Eduardo Segovia Pamo
Departamento de Ciencia de la Computación
Universidad Católica San Pablo
eduardo.segovia@ucsp.edu.pe

24 de Agosto, 2022

1. Introducción

El programa desarrollado se basa en realizar la multiplicación de una matriz con un vector, en un primer caso se recorre la matriz por filas y en el segundo caso se recorre la matriz por columnas para comprobar cual de los dos es mas rápido en base al tiempo y un análisis del resultado tomando en cuenta la cache.

2. Implementación

La implementación se divide en 3 secciones:

2.1. Asignación de datos

La asignación de la matriz 'A', el arreglo 'x' y el arreglo 'y' se asigna de la siguiente forma:

```
vector<vector<double>> A(MAX_SIZE, vector<double>(MAX_SIZE,
    0));
vector<double> x(MAX_SIZE, 0), y(MAX_SIZE, 0);
```

donde MAX_SIZE es 500, el tamaño de la matriz 'A' y el arreglo 'x' y el arreglo .

2.2. Caso 1

El primer caso se recorre la matriz por filas, por cada fila se recorre cada uno de sus datos y se toma el tiempo de ejecución utilizando la librería chrono tomando el tiempo en milisegundos:

```
auto t_start = std::chrono::high_resolution_clock::now();

for (int i = 0; i < MAX_SIZE; ++i) {
    for (int j = 0; j < MAX_SIZE; ++j) {
        y[i] += A[i][j] * x[j];
    }
}

auto t_end = std::chrono::high_resolution_clock::now();
double elapsed_time_ms = std::chrono::duration<double, std::
    milli>(t_end - t_start).count();
std::cout << "Primer Caso (ms): " << elapsed_time_ms << "\n"
    ;
```

2.3. Caso 2

El segundo caso se recorre la matriz por columnas, por cada columna se recorre cada uno de sus datos y se toma el tiempo de ejecución:

```
t_start = std::chrono::high_resolution_clock::now();

for (int j = 0; j < MAX_SIZE; ++j) {
    for (int i = 0; i < MAX_SIZE; ++i) {
        y[i] += A[i][j] * x[j];
    }
}
t_end = std::chrono::high_resolution_clock::now();
elapsed_time_ms = std::chrono::duration<double, std::milli>(
    t_end - t_start).count();
std::cout << "Segundo Caso (ms): " << elapsed_time_ms << "\n
    ";
```

3. Resultados

Para tomar los resultados se midió el tiempo de ejecución del caso 1 y 2 con una matriz cuadrada de tamaños 500 y 1000:
MAX_SIZE: 500
Tiempo de CASO 1: 1.2151 ms
Tiempo de CASO 2: 4.293 ms

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: guardado de la matriz en la cache

MAX_SIZE: 1000
Tiempo de CASO 1: 13.4477 ms
Tiempo de CASO 2: 33.031 ms

4. Análisis

Como observamos, el caso 1 nos da mejores tiempos, dado que la cache al trabajar con una matriz, guardara cada fila en cada linea de la cache, como se ve en la Fig. 1, cada fila de esta matriz; asumiendo que las direcciones de cada fila, no necesariamente toda la matriz, serán contiguas tomando en cuenta la localidad espacial al estar las direcciones de la fila juntas o la localidad temporal asumiendo que se accederá la matriz según las filas y no las columnas, teniendo una mayor probabilidad de realizar un cache hit a un cache miss.

5. Link del GitHub

Link del GitHub: https://github.com/wairesp/Computacion-Paralela.