Universidad Católica San Pablo

Pruebas sobre el comportamiento de la memoria caché: bucles anidados

Computer Science — Parallel and Distributed Computing

Harold Villanueva Borda

1. Primer Par de Bucles (i,j):

En el primer par de bucles, se recorre la matriz A por filas (i) y luego por columnas (j). Esto significa que los elementos de A se acceden secuencialmente en la memoria, lo cual puede aprovechar mejor la localidad espacial y mejorar el rendimiento del caché. Este tipo de acceso secuencial puede resultar en menos fallos de caché y una mejor utilización de la jerarquía de memoria, lo que generalmente conduce a un mejor rendimiento.

2. Segundo Par de Bucles (j,i):

En el segundo par de bucles, se recorre la matriz A por columnas (j) primero y luego por filas (i). Esto implica un acceso no contiguo a los elementos de A, lo que puede resultar en peor rendimiento debido a una menor eficiencia en el uso del caché. Al acceder a los elementos de A de esta manera, es más probable que se produzcan fallos de caché debido a la falta de localidad espacial. Conclusión:

3. Code

```cpp
#include <iostream>
#include <chrono>

const int max = 10000;
double A[max][max], x[max], y[max];

void initialize() {
    for (int i = 0; i < max; i++) {
        x[i] = 1.0;
        y[i] = 0.0;
        for (int j = 0; j < max; j++)
            A[i][j] = 1.0;
    }
}

void loop1() {
    for (int i = 0; i < max; i++)
        for (int j = 0; j < max; j++)
            y[i] += A[i][j] * x[j];
}

```

```cpp
void loop2() {
    for (int j = 0; j < max; j++)
        for (int i = 0; i < max; i++)
            y[i] += A[i][j] * x[j];
}

int main() {
    initialize();

    auto start1 = std::chrono::high_resolution_clock::now();
    loop1();
    auto end1 = std::chrono::high_resolution_clock::now();
    std::chrono::duration<double> elapsed1 = end1 - start1;

    initialize(); // Reset

    auto start2 = std::chrono::high_resolution_clock::now();
    loop2();
    auto end2 = std::chrono::high_resolution_clock::now();
    std::chrono::duration<double> elapsed2 = end2 - start2;

    std::cout << "Time for loop 1: " << elapsed1.count() << " s\n";
    std::cout << "Time for loop 2: " << elapsed2.count() << " s\n";

    if (elapsed1.count() < elapsed2.count()) std::cout << "The loop 1 is the winner\n";
    else if (elapsed1.count() > elapsed2.count()) std::cout << "The second loop is the winner\n";
    else std::cout << "Same time\n";
}
```

main (code)

4. Comparación de tiempos:

Cuadro 1: Tiempos de ejecución de los bucles

Size	Tiempo para el bucle 1 (s)	Tiempo para el bucle 2 (s)	Ganador
1000	0.0029998	0.0033018	Bucle 1
2000	0.0120882	0.0276046	Bucle 1
3000	0.0265921	0.0471333	Bucle 1
4000	0.0471781	0.0867026	Bucle 1
5000	0.0732076	0.138842	Bucle 1
6000	0.105906	0.203319	Bucle 1
7000	0.147737	0.28977	Bucle 1
8000	0.188289	0.471309	Bucle 1
9000	0.240823	0.506272	Bucle 1
10000	0.296267	0.664494	Bucle 1
20000	1.5655	2.8187	Bucle 1
30000	4.73304	7.79636	Bucle 1
40000	10.3511	17.9068	Bucle 1

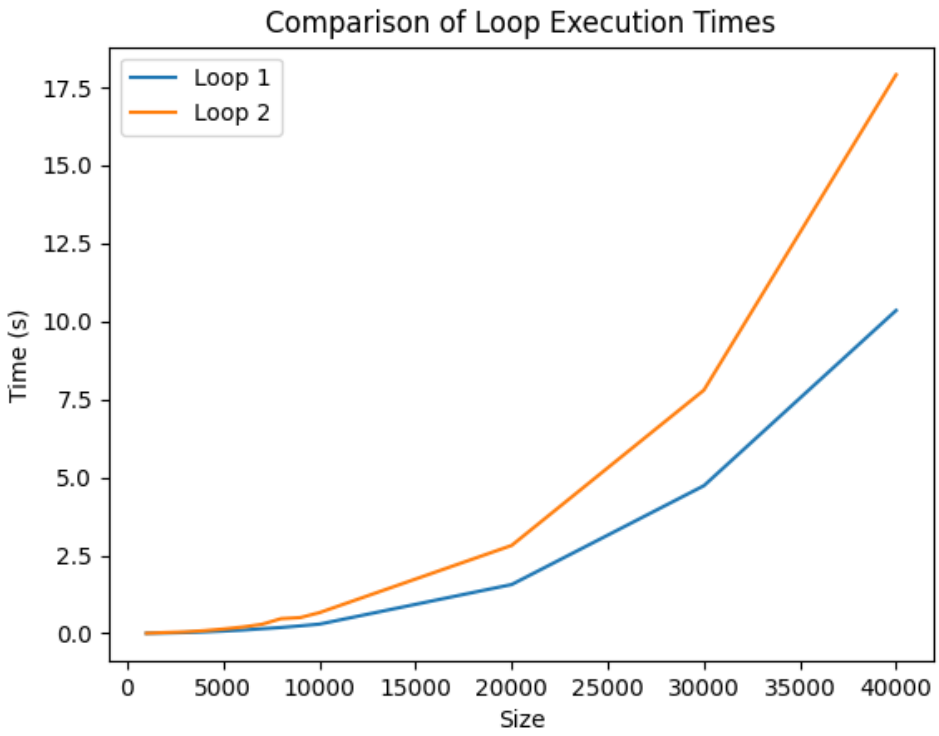

Figura 1: Tamaño vs tiempo.

5. link del repositorio:

GitHub.