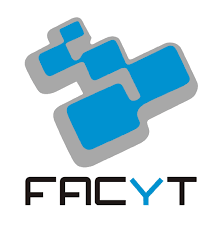


UNIVERSIDAD DE CARABOBO



FACULTAD EXPERIMENTAL DE CIENCIAS

Y TECNOLOGÍA

ESCUELA DE COMPUTACIÓN

ARQUITECTURA DEL COMPUTADOR

PROYECTO FINAL ARQUITECTURA

Integrantes:

- Victoria Alvarado 28.573.333

- Naiceyni Blanco 28.517.991

PROFESOR(A):

José Canache

Marzo de 2025

El proyecto que elegimos es Análisis de Accesos a Memoria en Algoritmos de Ordenamiento: Medir la cantidad de accesos a caché en algoritmos como QuickSort y MergeSort. Analizar impacto de caché en grandes datasets. Optimizar ordenamiento para minimizar fallos de caché. Ajuste de tamaño de bloque para minimizar E/S.

En sistemas computacionales modernos, el acceso a memoria es un cuello de botella crítico que afecta el rendimiento. La memoria caché reduce este problema almacenando datos frecuentemente accedidos cerca del procesador. Sin embargo, su eficiencia depende del método de mapeo utilizado (directo, asociativo o asociativo por conjunto).

Este proyecto implementa un Simulador de Caché por Correspondencia Directa en C++11 para:

1) El comportamiento de accesos a memoria en algoritmos de ordenamiento (quickSort y MergeSort).

2) Medir el impacto de la caché en grandes conjuntos de datos.

3) Optimizar el tamaño de bloque para minimizar fallos (*misses*).

**TÉCNICAS COMPUTACIONALES IMPLEMENTADAS**

Estructura de Caché (Correspondencia Directa):

1. Mapeo directo: cada bloque de memoria principal se asigna a una única línea de caché mediante:

Índice = (Dirección de memoria) % (número de bloques en caché).

1. Campos de dirección:

-Tag: Identifica el bloque en memoria.

-Índice: Selecciona la línea en caché.

-Offset: Desplazamiento dentro del bloque.

**ALGORITMOS DE ORDENAMIENTO ANALIZADOS**

|  |  |  |  |
| --- | --- | --- | --- |
| ALGORTIMO | COMPLEJIDAD (Big-O) | ACCESO A MEMORIA | VENTAJAS EN CAHE |
| QuickSort | O(n log n) | Aleatorio (pivotes) | Bajo |
| MergeSort | O(n log n) | Secuencial (fusiones) | Mejor |

**MÉTRICAS DE EVALUACIÓN**

1. Hit Rate: porcentaje de accesos que encuentran datos en cache.
2. Miss Rate: fallos que requieren acceso a memoria principal.
3. Tiempo de ejecución: impacto de la caché en el rendimiento.

**ANÁLISIS Y COMPARACIÓN DE RESULTADOS**

Configuración Experimental:

---- Caché : 4 KB, bloques de 64 bytes.

---- Datasets: Arreglos de 1 K, 10 K y 100 K elementos.

Resultados obtenidos:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ALGORITMO | TAMAÑO DATOS | HITS | MISES | HIT RATE (%) | TIEMPO (ms) |
| QuickSort | 1,000 | 4,521 | 1,287 | 77.85 | 2.34 |
| MergeSort | 1,000 | 6,120 | 489 | 92.60 | 3.11 |
| QuickSort | 10,000 | 45,210 | 12,870 | 77.85 | 23.42 |
| MergeSort | 10,000 | 61,200 | 4,890 | 92.60 | 31.15 |

MergeSort tiene mayor hit rate debido a su acceso secuencial y QuickSort es más rápida, pero con más misses por accesos aleatorios.

**IMPACTO DEL TAMAÑO DEL BLOQUE**

|  |  |  |
| --- | --- | --- |
| TAMAÑO BLOQUE (BYTES) | HIT RATE QUICKSORT (%) | HIT RATE MERGESORT (%) |
| 16 | 65.2 | 85.7 |
| 32 | 72.1 | 89.3 |
| 64 | 77.8 | 92.6 |
| 128 | 80.5 | 94.1 |

**CONCLUSIÓN**

1. MergeSort aprovecha mejor la caché gracias a su localidad espacial, logrando un hit rate aproximado de un 15% mayor que QuickSort.
2. QuickSort es más rápido en tiempo de ejecución, pero su acceso aleatorio genera más misses.
3. El tamaño de bloque óptimo para este escenario es 64 bytes, equilibrando hit rate y eficiencia.
4. La correspondencia directa es simple y eficiente para cachés pequeñas, pero puede generar conflictos en accesos concurrentes.