LABORATORIO 2: Pruebas sobre el comportamiento de la memoria caché

Uberto García

I. Implementación

Primero la multiplicación clásica que es el uso de 3 bucles tal que:

```
int** Multiplicacion_clasica(int **A,int**B, int a, int b, int n){
    int **resultado;
    resultado = new int*[a];
    for (int i = 0; i < a; i++){
        resultado[i] = new int[b];
        for (int j = 0; j < b; j++){
            resultado[i][j] = 0;
            for (int k = 0; k < n; k++){
                resultado[i][j] += (A[i][k]*B[k][j]);
            }
        }
    }
    return resultado;
}
```

Luego le sigue la multiplicación por bloques que es de 6 bucles, de la siguiente forma:

```
int ** Multiplicacion_por_bloques(int **A,int **B, int a, int b, int n, int tb){
    int ** resultado = new int*[a];
    for (int i = 0; i < a; i++){
        resultado[i] = new int[b];
    }
    int ** s = new int*[tb];
    for (int i = 0; i < tb; i++){
        s[i] = new int[tb];
    }
    for (int i = 0; i < a; i = i + tb){
        for (int j = 0; j < b; j = j + tb){
            /*Matriz 0*/
            for (int p = 0; p < tb; p++)
                for (int q = 0; q < tb; q++)
                    s[p][q] = 0;
            for (int k = 0; k < n; k = k + tb){
                /*multiplicacion*/
                for (int p = i; p < (i+tb); p++){
                    for (int q = j; q < (j+tb); q++){
                        for (int r = k; r < (k+tb); r++){
                            if(p < a && q < b &&r < n){
                                s[p-i][q-j] += (A[p][r] * B[r][q]);
                            }
                        }
                    }
                }
            }
            /*Copiar*/
            for (int x = 0; x < tb; x++){
                for (int y = 0; y < tb; y++){
                    if((i+x) < a && (j+y) < b){
                        resultado[i+x][j+y] = s[x][y];
                    }
                }
            }
        }
    }
    return resultado;
}
```

Una vez obtenido todo, damos valores, en este caso serán 1600 todas las filas y columnas, y se harán por bloques de 4x4:

```
int main(){
    unsigned i0, i1, f0, f1;
    const int a = 1600;
    const int b = 1600;
    const int n = 1600;
    int **A,**B,**R1,**R2;
    A = new int*[a];
    B = new int*[n];
    for (int i = 0; i < n; i++){
```

```
        B[i] = new int[b];
        for (int j = 0; j < b; j++){
            B[i][j] = rand()%21 - 10;
        }
    }
    for (int i = 0; i < a; i++){
        A[i] = new int[n];
        for (int j = 0; j < n; j++){
            A[i][j] = rand()%21 - 10;
        }
    }
    /*Multiplicacion por bucles anidados*/
    i0= clock();
    R1 = Multiplicacion_clasica(A,B,a,b,n);
    f0= clock();
    /*Multiplicaci n por bloques*/
    i1= clock();
    R2 = Multiplicacion_por_bloques(A,B,a,b,n,4);
    f1= clock();
    double result0 = (f0 - i0)/CLOCKS_PER_SEC;
    double result1 = (f1 - i1)/CLOCKS_PER_SEC;
    cout<<"Tiempo clasico: "<<result0<<endl;
    cout<<"Tiempo por bloques: "<<result1<<endl;
    return 0;
}
```

Al terminar, el tiempo se puede visualizar

II. Comparación

Con valores menores, la multiplicación clásica comienza una clara ventaja, pero una vez que se sigue aumentando, la multiplicación por bloques supera a la clásica debido a su complejidad.

III. Conclusion

La primera multiplicación sirve para matrices pequeñas, mientras que la segunda para matrices grandes

Click aquí para ir al repositorio de Github