Informe Proyectos

David Neyra Gutierrez

Universidad Nacional de San Agustin

Résumé En el Presente informe se desarrollara un analisis de lo que se avanzo sobre Three Nested Loop y un analisis sobre los datos de Valgrind y kcachegrind.

Table des matières

1 Introducción

Introduccion Todos los datos están en el siguiente repositorio Link *https* : *//github.com/xxdavidxx*11/*Paralelos*, todos los archivos citados están en el presente repositorio, cualquier duda se puede verificar en ellos.

2 Primera Pregunta

Implement in C the simple three-nested-loop version of the matrix product and try to evaluate its performance for a relatively large matrix size. El presente codigo se encuentra en *preg.hpp* del repositorio.

```
typedef size_t ST;
typedef int** MI;

MI TNL3(MI &A,MI &B,ST &tam)
{
    ST i,j,k;
    MI C;
    C=new int*[tam]; /*Uso de Memoria*/
    for(int i=0;i<tam;i++)
        C[i]=new int[tam]; /*Uso de Memoria*/

    for(i=0;i<tam;i++)
        for(j=0;j<tam;j++)
            for(k=0;k<tam;k++)
            C[i][j]+=A[i][k]*B[k][j]; /*Uso de Memoria*/
    return C;
}
```

3 Segunda Pregunta

Implement the blocked version with six nested loops to check whether you can observe a significant gain. El presente codigo se encuentra en *preg.hpp* del repositorio.

```
MI TNL6(MI &A,MI &B,ST tam,ST tamblock)
{
    ST i,j,k,ib,jb,kb;
    MI C;
    C=new int*[tam];
    for(int i=0;i<tam;i++)
        C[i]=new int[tam]; /*Uso de Memoria*/

    for (i = 0; i < tam; i+=tamblock)
        for (j = 0 ; j < tam; j+=tamblock)
            for (k = 0; k < tam ; k+= tamblock)
                for (ib = i; ib<j+tamblock; ib++)
                    for (kb = k ; kb<k+tamblock; kb++)
                        for (jb = j ; jb<j+tamblock ; jb++)
                            C[ib][jb] += A[ib][kb] *B[kb][jb]; /*
                                Uso de Memoria*/
    return C;
}
```

4 Tercera Pregunta

Execute these algorithms step by step to get a good understanding of data movements between the cache and the memory and try to evaluate their respective complexity in term of distant memory access.

En el TNL3 :
El tiempo computacional es $BIGO(n^3)$ pero utiliza demasiada memoria en su ejecución. La complejidad es cubica ya que recorre 3 ciclos loop invariant, estos ciclos dependen del tamaño de la matriz directamente.

En el TNL6 :
El tiempo computacional es $BIGO(n^3)$ pero utiliza poca memoria en su ejecución. La complejidad es cubica, ya que aunque se utilice 6 ciclos, estos no lo evalúan en toda la matriz, sus ciclos en este caso no dependen del tamaño de la matriz sino del tamaño del bloque en la matriz a ejecutar.

5 Cuarta Pregunta

Execute these two versions of the code with valgring and kcachegrind to get a precise evaluation of their performance in term of cache misses.

Se tiene dos resultados : *primerValgrid.txt* y *segundoValgrid.txt*, el primero podemos observar que en la línea numero 52 comienza un análisis de los datos los cuales se crearon pero no hubo una eliminación de estos datos, es así que me di cuenta que había memoria la cual no había liberado y esto paso, que le reserve espacio de memoria con la función *new* de C y no la libere con la función *delete*, es por ello que cree por último la función *limpiar* la cual muestro en el siguiente espacio, junto con la función llenar la cual reserva espacio de Memoria.

```
void llenar(MI& A,ST &tam)
{
    A=new int*[tam];   /*Uso de Memoria*/
    for(int i=0;i<tam;i++)
        A[i]=new int[tam]; /*Uso de Memoria*/

    ST i,j;
    for(i=0;i<tam;i++)
        for(j=0;j<tam;j++)
            A[i][j]=rand() % 10 + 1; /*Uso de Memoria*/
}

void limpiar(MI& A, ST &tam)
{
    for(int i=0;i<tam;i++)
        delete A[i];
    delete A;
```

```
}
```

Con la implementación de esta función pude liberar la información la cual no era necesaria para el programa, de esta manera se pudo ver en $SegundoValgrind.txt$ en la línea numero 106 se puede ver que pude liberar la información innecesaria para el sistema y de cierto modo optimizar el trabajo hecho. En cuanto al kcachegrind se hizo lo mismo, pero en este caso se pudo optimizar el tiempo de ejecución del antes y después de estos datos, de 2,192,011 a 2,188,939.

6 Conclusión

Cuando se usa *new* se debe utilizar necesariamente *delete* para poder liberar espacio, en cuanto a c puro, si se utiliza *malloc* se debe utilizar $free$, de esta manera podemos optimizar y mejorar el tiempo de ejecución del programa el cual se está realizando, para poder mejorar de una manera el uso del cache que se tiene utilizando.