Universidad Nacional de San Agustín

Ciencia de la Computación

Algoritmos paralelos

Algoritmos paralelos: Impacto de cache

Integrantes:

- FERNÁNDEZ ZAMORA, FLOR

28 de junio de 2016

Índice

Parte I

Multiplicación de Matrices.

1. Introducción

El objetivo de este ejercicio es realizar una comparación entre una multiplicación tradicional y una multiplicación realizada en bloques (block multiplication), esto con la finalidad de entender cuan importante es entender el desenvolvimiento de la cache en una simple programa.
La primera parte explica la multiplicación tradicional y muestra en código desarrollado en c++. La segunda parte consiste en explicar la multiplicación por bloques y el tiempo de demora usado por el programa. En la última parte se explica ambos programas usando herramientas como valgring y kcachegrind que te permiten evaluar el desevolvimiento de tus programas en término de *cache miss*.

2. Multiplicación tradicional

2.1. Explicación

Esta seccción muestra la implementación de una matriz C=C+A*B. La siguiente imagen muestra el algoritmo que puede ser usado para multiplicar m*m matrices. , ver fig 1, 2.

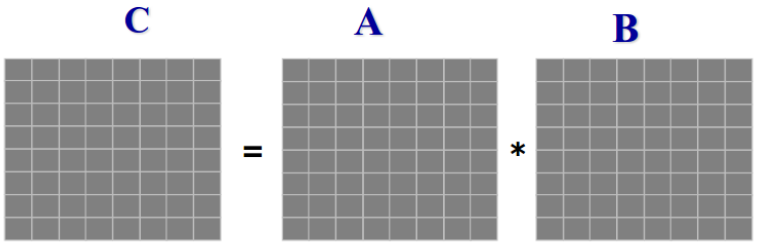

Figura 1: Multiplicación de matrices

2.2. Programa en C++

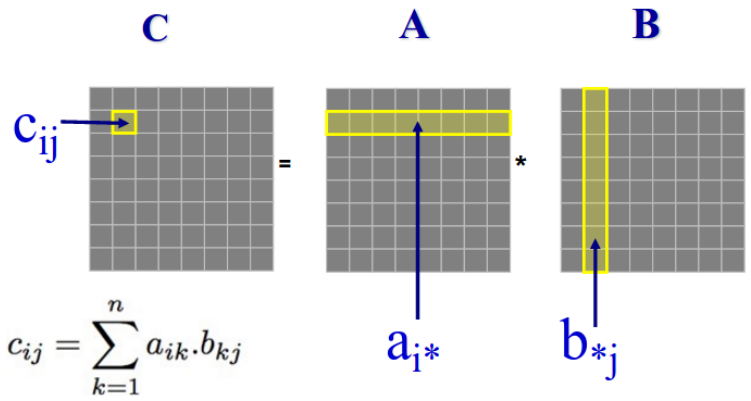

Figura 2: Multiplicación de matrices

```
#include <iostream>
#include <stdlib.h>       /* srand, rand */
#include <time.h>

using namespace std;

const int n=1000;

int AA[n][n];
int BB[n][n];
int C[n][n];

void multiplicacion()
{

    for(int i=0;i<n;++i)
        for(int j=0;j<n;++j)
            C[i][j]=0;

    for(int i=0;i<n;++i)
        for(int j=0;j<n;++j)
            for(int k=0;k<n;++k)
            {
                C[i][j]+=AA[i][k]*BB[k][j];
            }

    cout<<"Finalizo"<<endl;
```

```
}

void print()
{
    int count=0;
    for(int i=0;i<n;++i)
        for(int j=0;j<n;++j)
        {
            cout<<i<<" "<<j<<" : "<<C[i][j]<<endl;
            ++count;

        }

    cout<<"# elementos : "<<count<<endl;
}

int main (){
    multiplicacion();
    print();

    return 0;
}
```

3. Multiplicación en bloques

3.1. Explicación

Memoria Cache es una área de almacenamiento temporal donde la data que se accede frecuentemente puede estar almacenado para un rápido acceso. , ver fig 3, 4, 5, 6 .

3.2. Programa en C++

```
#include <iostream>
#include <algorithm>
#include <stdlib.h>      /* srand, rand */
#include <time.h>

using namespace std;
```

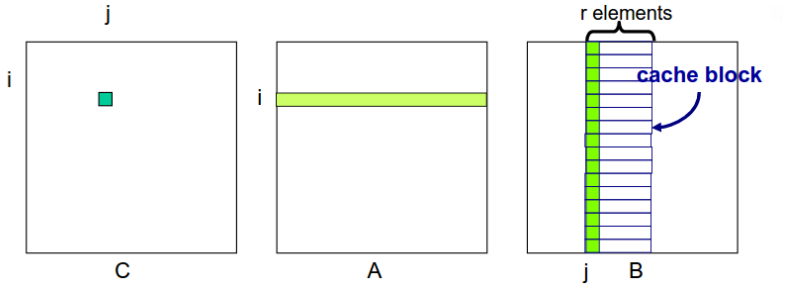

Figura 3: Multiplicación en bloques

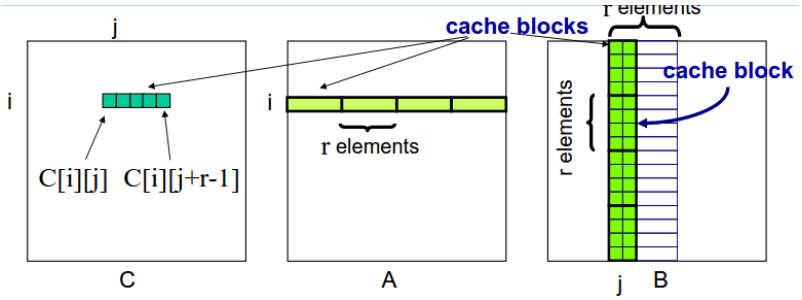

Figura 4: Multiplicación en bloques

```
const int n=10;
const int blockSize=5;

int AA[n][n];
int BB[n][n];
int C[n][n];
```

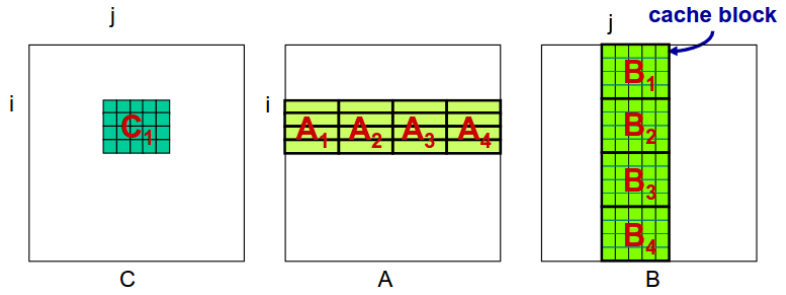

Figura 5: Multiplicación en bloques

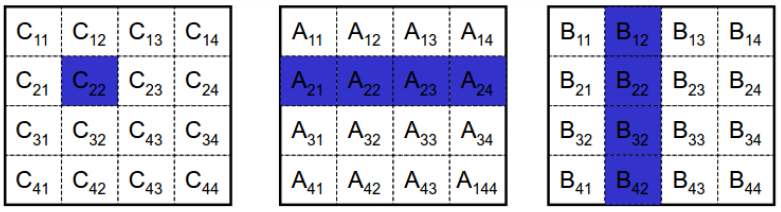

Figura 6: Multiplicación en bloques

```
void InicializarMatrix()
{
        for(int i=0;i<n;++i)
        for(int j=0;j<n;++j)
            C[i][j]=0;
}

void BlockMultiplication()
{
        InicializarMatrix();
    for(int i=0; i<n; i+=blockSize)
        for(int j=0; j<n; j+=blockSize)
                for(int k=0; k<n; k+=blockSize)
                        for(int x=i; x<i+blockSize; ++x)
                                for(int y=j; y<j+blockSize; ++y)
                                        for(int z=k; z<k+blockSize; ++z)
                                        {
                                                C[x][y]+=AA[x][z]*BB[z][y
                                        }

}

void print()
{
    int count=0;
    for(int i=0;i<n;++i)
        for(int j=0;j<n;++j)
        {
            cout<<i<<" "<<j<<" : "<<C[i][j]<<endl;
            ++count;
```

```
        }

    cout<<"# elementos: "<<count<<endl;
}


int main(){

        BlockMultiplication();
        print();

        return 0;
}
```

4. Análisis de rendimiento

Para el análisis de rendimiento se uso $clock_t$ *clock*() esta función retorna el tiempo del procesador usado por el programa desde el inicio de la ejecución.

```
t = clock();
add();
multiplicacion();
print();
t = clock() - t;
cout<<(((float)t1)/CLOCKS_PER_SEC)<<endl;
```

En cuento a la multiplicación tradicional el tiempo que retornó fue: $66e-06$ y la multiplicación por bloques $44e-06$.

5. Usando valgring y kcachegrind

6. Conclusiones

- Los programadores pueden obtimizar el rendimiento de la cache, puede ser como organizamos las estructuras y como accdemos a ella, una de las técnicas más generales es la de *blocking*.

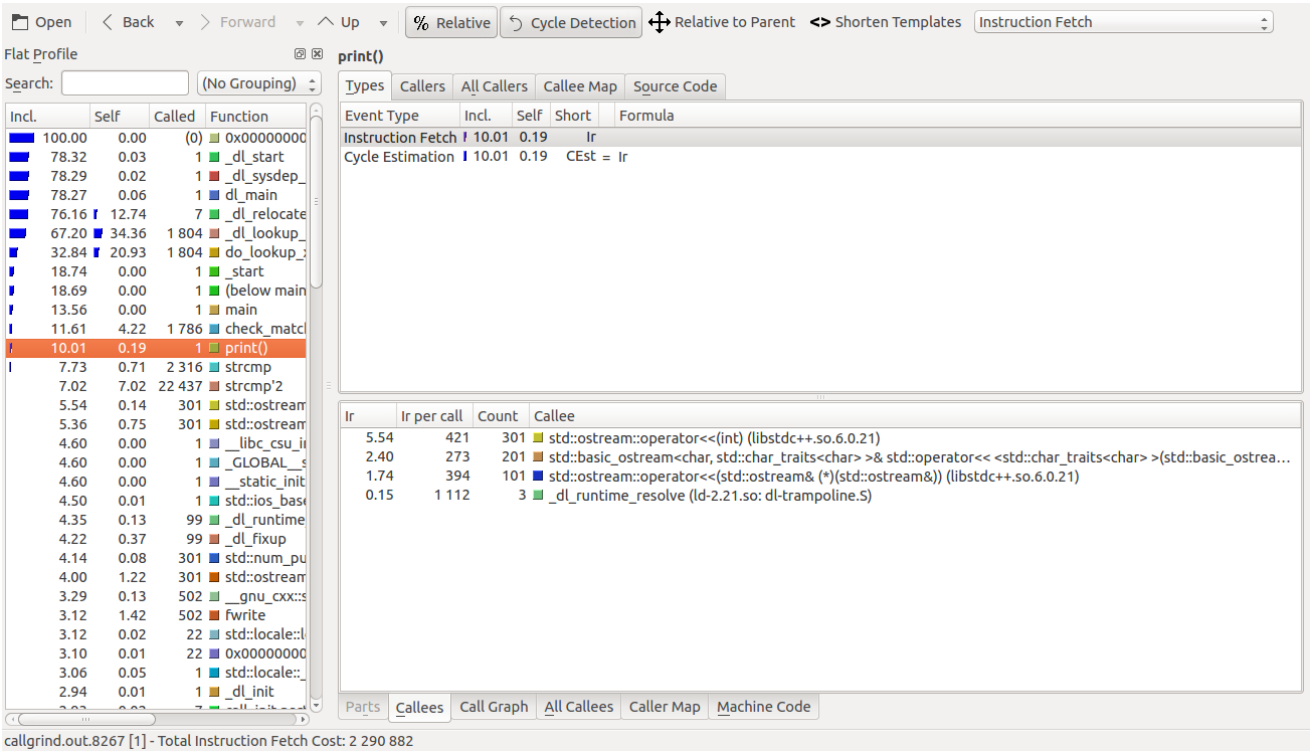

Figura 7: Multiplicación tradicional

Referencias

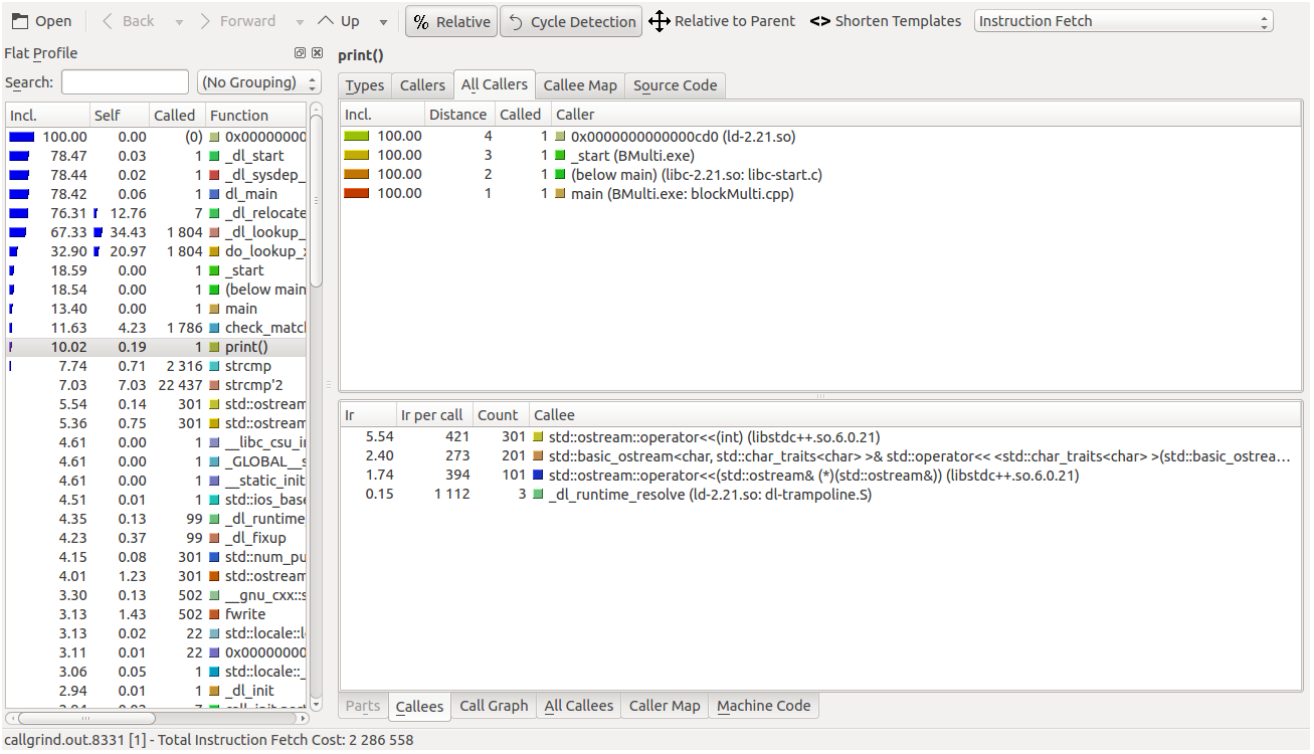

Figura 8: Multiplicación en bloques