

# ALGORITMOS PARALELOS

DÁVILA GUILLÉN GRIMALDO JOSÉ

Universidad Nacional de San Agustín  
Escuela profesional de Ciencia de la Computación

27 de marzo de 2017



Arequipa – Perú

## 1. Introducción

En este trabajo se analiza el impacto de la memoria cache en la aceleración del tiempo de ejecución del producto de matrices. Se ejecutaron dos algoritmos: la versión simple del producto de matrices y la versión de bloques.

Teóricamente se conoce que la versión de bloques es más rápida que la versión simple, pero ¿esto a qué se debe?, la respuesta es: Memoria cache.

Para realizar el análisis se hizo uso de un software llamado Valgrind.

Valgrind tiene diferentes herramientas que nos ayuda a evaluar y mejorar nuestro código. Las más principales son:

- Memcheck es un detector de errores de memoria.
- Cachegrind realiza la simulación de caché capturando los accesos a esta.
- Callgrind crea el gráfico de llamadas de un programa, es decir como las funciones se llaman entre sí y cuantos eventos ocurren mientras se ejecuta una función.
- Helgrind es un detector de errores de paralelización.

## 2. Código

### 2.1 multiplicacion simple:

```
void multiplicacionsimple(int **&A,int **&B,int **&C,int af,int ac,int bf,int bc)
{
    for(int i=0;i<af;i++){
        for(int j=0;j<bc;j++){
            C[i][j]=0;
            for(int k=0;k<ac;k++){
                C[i][j]=(C[i][j]+(A[i][k]*B[k][j]));
            }
        }
    }
}
```

Algoritmo 1: multiplicación simple de matrices.

## 2.2 mutliplicación por bloques:

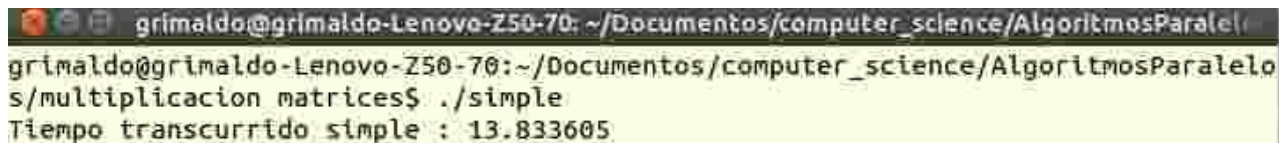
```
int n=1000;
int blockSize=50;
void BlockMultiplication(int **&A,int **&B,int **&C)
{
    for(int i=0; i<n; i+=blockSize)
        for(int j=0; j<n; j+=blockSize)
            for(int k=0; k<n; k+=blockSize)
                for(int x=i; x<i+blockSize; ++x)
                    for(int y=j; y<j+blockSize; ++y)
                        for(int z=k; z<k+blockSize; ++z)
                        {
                            C[x][y]+=A[x][z]*B[z][y];
                        }
}
```

Algoritmo 2: multiplicación por bloques de matrices.

## 3. Análisis de tiempo

En cuanto al tiempo de ejecución del algoritmo de multiplicación simple probado con una matriz de 1000x1000 fue de un total de :

= 13.833605 segundos.

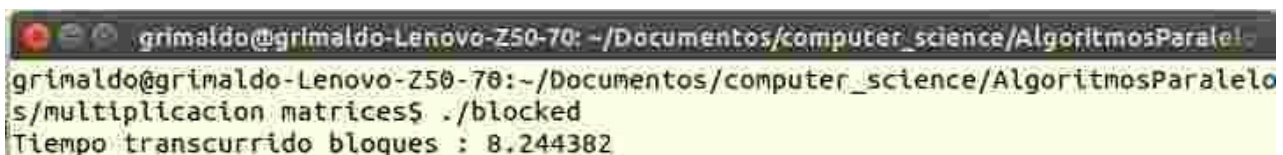


A terminal window showing the command `./simple` being executed in the directory `~/Documentos/computer_science/AlgoritmosParalelos/multiplicacion matrices`. The output is `Tiempo transcurrido simple : 13.833605`.

Figura 1: Resultado del tiempo de ejecución de la multiplicación simple.

En cuanto al tiempo de ejecución del algoritmo de multiplicación por bloques probado con una matriz de 1000x1000 fue de un total de :

= 8.244382 segundos.



A terminal window showing the command `./blocked` being executed in the directory `~/Documentos/computer_science/AlgoritmosParalelos/multiplicacion matrices`. The output is `Tiempo transcurrido bloques : 8.244382`.

Figura 2: Resultado del tiempo de ejecución de la multiplicación por bloques.

Podemos ver que en cuanto a velocidad la multiplicación por bloques es mas rápida que la multiplicación simple.

## 4.Análisis con valgring y kcachegrind

```

grimaldo@grimaldo-Lenovo-Z50-70: ~/Documentos/computer_science/AlgoritmosParalelos
grimaldo@grimaldo-Lenovo-Z50-70:~/Documentos/computer_science/AlgoritmosParalelos
==7768== Cachegrind, a cache and branch-prediction profiler
==7768== Copyright (C) 2002-2013, and GNU GPL'd, by Nicholas Nethercote et al.
==7768== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==7768== Command: ./simple
==7768==
--7768-- warning: L3 cache found, using its data for the LL simulation.
Tiempo transcurrido simple : 318.746230
==7768==
==7768== I   refs:      54,179,712,213
==7768== I1 misses:      1,033
==7768== LLi misses:      1,029
==7768== I1 miss rate:      0.00%
==7768== LLi miss rate:      0.00%
==7768==
==7768== D   refs:      27,076,273,828 (26,058,167,997 rd + 1,018,105,831 wr)
==7768== D1 misses:      1,190,439,875 ( 1,190,247,643 rd +      192,232 wr)
==7768== LLd misses:      62,979,647 (   62,789,023 rd +      190,624 wr)
==7768== D1 miss rate:      4.3% (      4.5% +      0.0% )
==7768== LLd miss rate:      0.2% (      0.2% +      0.0% )
==7768==
==7768== LL refs:      1,190,440,908 ( 1,190,248,676 rd +      192,232 wr)
==7768== LL misses:      62,980,676 (   62,790,052 rd +      190,624 wr)
==7768== LL miss rate:      0.0% (      0.0% +      0.0% )

```

Figura 3: Resultado del tiempo del análisis de la multiplicación simple.

```

grimaldo@grimaldo-Lenovo-Z50-70: ~/Documentos/computer_science/AlgoritmosParalelos/multipli-
==7981== Copyright (C) 2002-2013, and GNU GPL'd, by Nicholas Nethercote et al.
==7981== Using Valgrind-3.10.1 and LibVEX; rerun with -h for copyright info
==7981== Command: ./blocked
==7981==
--7981-- warning: L3 cache found, using its data for the LL simulation.
Tiempo transcurrido bloques : 294.879371
==7981==
==7981== I   refs:      56,444,411,478
==7981== I1 misses:      1,035
==7981== LLi misses:      1,031
==7981== I1 miss rate:      0.00%
==7981== LLi miss rate:      0.00%
==7981==
==7981== D   refs:      28,247,942,789 (27,211,429,531 rd + 1,036,513,258 wr)
==7981== D1 misses:      5,065,290 (   4,935,807 rd +      129,483 wr)
==7981== LLd misses:      1,490,773 (   1,362,897 rd +      127,876 wr)
==7981== D1 miss rate:      0.0% (      0.0% +      0.0% )
==7981== LLd miss rate:      0.0% (      0.0% +      0.0% )
==7981==
==7981== LL refs:      5,066,325 (   4,936,842 rd +      129,483 wr)
==7981== LL misses:      1,491,804 (   1,363,928 rd +      127,876 wr)
==7981== LL miss rate:      0.0% (      0.0% +      0.0% )

```

Figura 4: Resultado del tiempo del análisis de la multiplicación por bloques.

### Multiplicación Simple

Instrucciones ejecutadas (I refs) :	54,179,712,213
Lecturas de memoria (D refs) :	27,076,273,828
Cache Misses % :	4.3%
Lectura de datos de cache (LL refs) :	1,190,440,908
Cache Misses % (Datos) :	0.0%

### Multiplicación por Bloques

Instrucciones ejecutadas (I refs) :	56,444,411,478
Lecturas de memoria (D refs) :	28,247,942,789
Cache Misses % (D1 miss rate):	0.0%
Lectura de datos de cache (LL refs) :	5,066,325
Cache Misses % (Datos)LL miss rate :	0.0%

Podemos notar que la diferencia notable se produce en la lectura de datos a la memoria cache, esta es la causa de que el tiempo de demora de la multiplicación por bloques sea mas rapida que la multiplicación simple a pesar de que la multiplicación simple tenga menos instrucciones ejecutadas y menos lecturas de memoria.