

Instituto Tecnológico de Estudios Superiores Monterrey Campus Puebla



Juan Manuel Rendón García

A01731524

Proyecto Final

Multiprocesadores Gpo 1

Emmanuel Torres Ríos

Tabla de Contenidos

Introducción	3
Especificaciones y aclaraciones	4
Desarrollo.....	8
Actividad 1.3.....	8
3 Threads.....	9
6 Threads.....	9
9 Threads.....	10
12 Threads.....	11
15 Threads.....	11
18 Threads.....	12
Actividad 1.4.....	13
Actividad 1.5.....	15
Velocidad Disco Duro.....	20
Velocidad RAM	20
Actividad 2.1	24
Actividad 2.2.....	31
Conclusión	45
Referencias.....	46

Introducción

Con el propósito de identificar y señalar las diferencias que existen entre equipos de cómputo, procesadores, procesamiento y tiempos de ejecución es que se presenta el siguiente proyecto. Como parte de nuestras actividades a través del semestre se contabilizaron tiempos de ejecución de múltiples ejercicios, los cuales serán puestos a comparación con una maquina virtual prevista por Google Cloud la cual ejecutará los ejercicios selectos para demostrar las diferencias entre mi computadora personal y dicho servicio.

Especificaciones y aclaraciones


A continuación, se presentarán las especificaciones de mi computadora personal:

Device specifications	
Alienware m15 R3	
Device name	DESKTOP-36DU0QG
Processor	Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz 2.59 GHz
Installed RAM	16.0 GB (15.8 GB usable)
Device ID	7ECCD0AF-8487-4AA1-9531-6904ABC61ACA
Product ID	00325-81759-91483-AAOEM
System type	64-bit operating system, x64-based processor
Pen and touch	No pen or touch input is available for this display

Por otra parte, las de la máquina virtual (Incluyendo instalaciones):

Name *
multiproyecto

Labels ?
[+ ADD LABELS](#)

Region *
us-east1 (South Virginia)
us-east4 (Northern Virginia)
us-east5 (Columbus)
us-south1 (Dallas)
us-west1 (Oregon)  Low CO2
us-west2 (Los Angeles)
us-west3 (Salt Lake City)
us-west4 (Las Vegas)
Europe

Zone *
OPTIMIZED GPU

- (1) Declaración de ubicación de los servidores (Seleccionando uno de emisiones reducidas de carbono)



(2) Zonas internas dentro del servidor (Predeterminado en la mejor zona)

Machine configuration

Machine family

GENERAL-PURPOSE

COMPUTE-OPTIMIZED

MEMORY-OPTIMIZED

GPU

High-performance machine types for compute-intensive workloads

Series

C2

Powered by Intel Cascade Lake CPU platform

Machine type

c2-standard-4 (4 vCPU, 16 GB memory)



vCPU

4

Memory

16 GB

✓ CPU PLATFORM AND GPU

(3) Es importante resaltar que se selecciono la mejor opción (predeterminada) para procesamiento y computación.

Display device

Enable to use screen capturing and recording tools.

☒ Enable display device

Confidential VM service ?

Confidential Computing is disabled on this VM instance

ENABLE

Container ?

Deploy a container image to this VM instance

DEPLOY CONTAINER

(4) Características extras de configuraciónn

Boot disk

Select an image or snapshot to create a boot disk; or attach an existing disk. Can't find what you're looking for? Explore hundreds of VM solutions in [Marketplace](#)

PUBLIC IMAGES

CUSTOM IMAGES

SNAPSHOTS

ARCHIVE SNAPSHOTS

EXISTING DISKS

Operating system
Ubuntu

Version *
Ubuntu 18.04 LTS

x86_64, amd64 bionic image built on 2022-10-18, supports Shielded VM features

Boot disk type *
Balanced persistent disk

[COMPARE DISK TYPES](#)

Size (GB) *
10

[SHOW ADVANCED CONFIGURATION](#)

SELECT

CANCEL

(5) Sistema Operativo Selecto con 10 GB de memoria.

Identity and API access ?

Service accounts ?

Service account
Compute Engine default service account

Requires the Service Account User role (roles/iam.serviceAccountUser) to be set for users who want to access VMs with this service account. [Learn more](#)

Access scopes ?

- ☐ Allow default access
- ☒ Allow full access to all Cloud APIs
- ☐ Set access for each API

Firewall ?

Add tags and firewall rules to allow specific network traffic from the Internet

- ☐ Allow HTTP traffic
- ☐ Allow HTTPS traffic

Advanced options

Networking, disks, security, management, sole-tenancy



Your free trial credit will be used for this VM instance. [Google Cloud Free Tier](#)

(6) Configuraciones seleccionadas para accesos e identificaciones

Monthly estimate

\$122.94

That's about \$0.17 hourly

Pay for what you use: No upfront costs and per second billing

Item	Monthly estimate
4 vCPU + 16 GB memory	\$152.43
10 GB balanced persistent disk	\$1.00
Sustained use discount	-\$30.49
Total	\$122.94

[Compute Engine pricing](#)

[^ LESS](#)

(7) Costo total de la renta de dicho equipo (Reducido debido al servidor selecto en 1)

```
juanmanuelrg98@multiproyecto:~$ sudo apt-get install git
Reading package lists... Done
Building dependency tree
Reading state information... Done
git is already the newest version (1:2.17.1-1ubuntu0.13).
The following package was automatically installed and is no longer required:
  libnuma1
Use 'sudo apt autoremove' to remove it.
0 upgraded, 0 newly installed, 0 to remove and 6 not upgraded.
```

(8) Preinstalación de GitHub

Desarrollo

Se seleccionaron 5 actividades a ser comparadas, las cuales se presentarán individualmente para demostrar las comparaciones por separado y se pueda entender mejor las diferencias entre ellos.

Se utilizará la siguiente fórmula para comparar los tiempos entre equipos:

$$(1) n = \frac{\text{Tiempo de ejecución en } y}{\text{Tiempo de ejecución en } x} = \frac{\text{Rendimiento } x}{\text{Rendimiento } y}$$

Donde x es el equipo más rápido que y

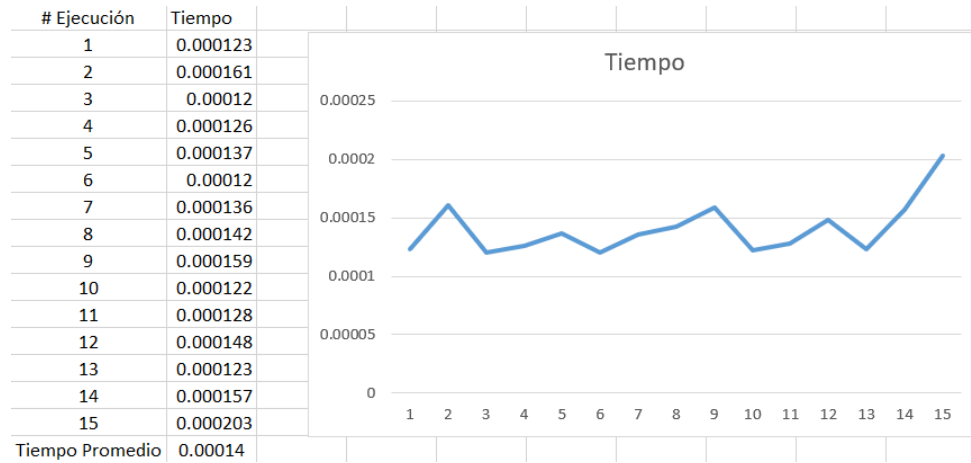
Actividad 1.3

La primera actividad consiste en reconocer el funcionamiento de multithreading por medio de la asignación de threads en series de 3, del 3 al 18. Repitiendo cada serie 15 veces para denotar si existen variaciones en el tiempo de ejecución notables y resaltar el funcionamiento “aleatorio” de asignación de threads.

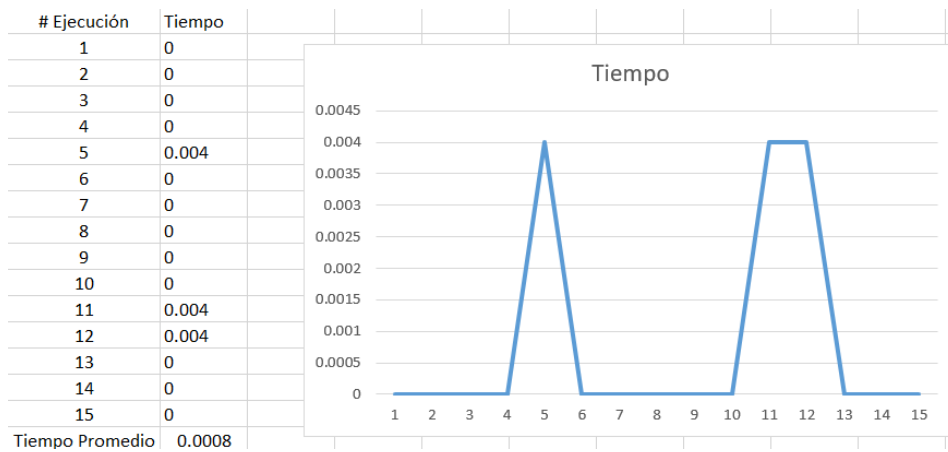
Se uso el siguiente código:

```
#include <stdio.h>
#include <omp.h>
#define NUM_THREADS 3 //Cambiar los threads
int main(){
    double tiempo;
    omp_set_num_threads(NUM_THREADS);
    const double startTime = omp_get_wtime();
    #pragma omp parallel
    {
        int ID = omp_get_thread_num();
        printf("Procesadores (%d)", ID);
        printf("Multiples (%d)", ID);
        printf(",en accion --- %d\n", ID);
    }
    const double endTime = omp_get_wtime();
    tiempo = endTime-startTime;
    printf("Tiempo total = %lf\n", (tiempo));
}
```


3 Threads

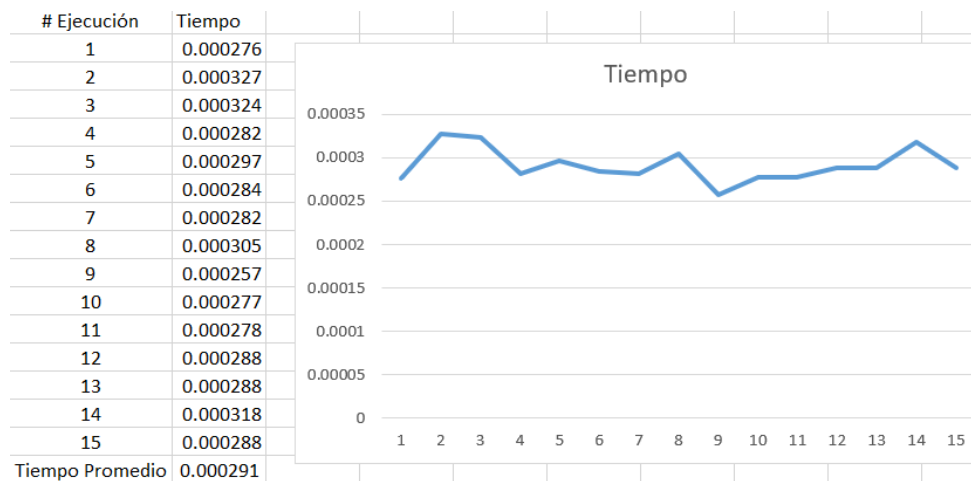


(1.3.1) Tiempo VM

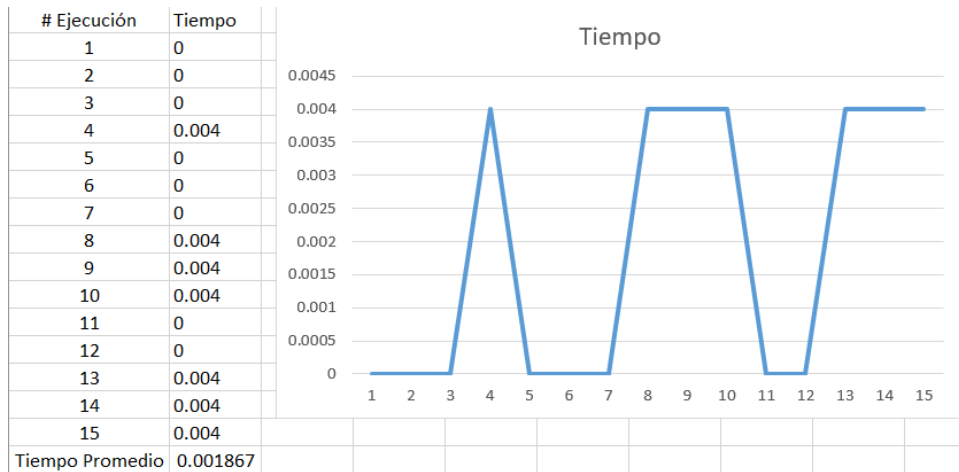


(1.3.2) Tiempo LAPTOP

6 Threads

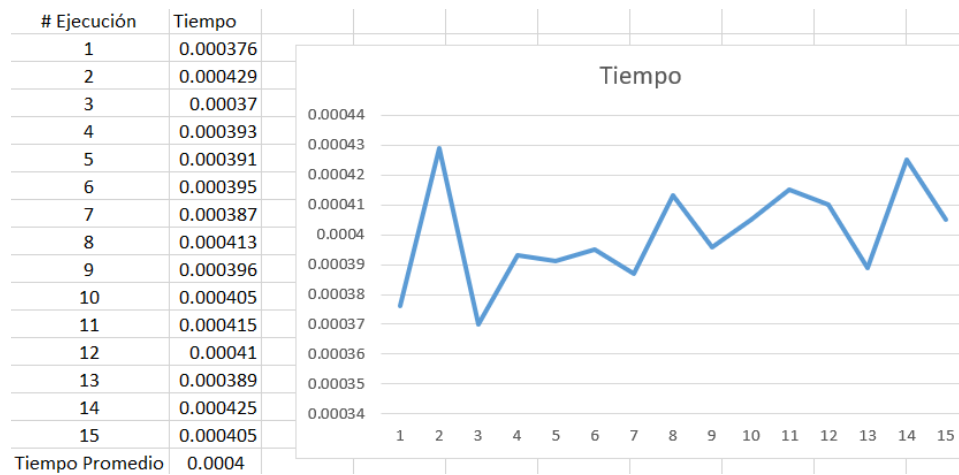


(1.3.3) Tiempo VM

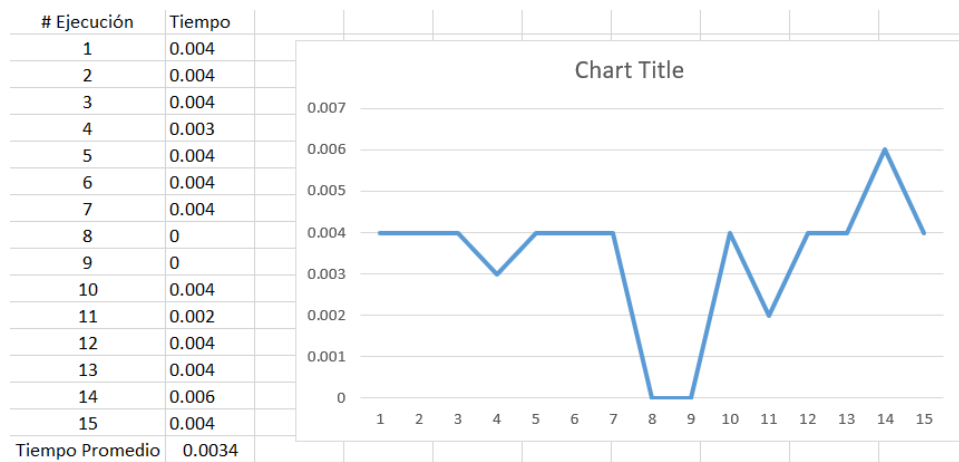


(1.3.4) Tiempo Laptop

9 Threads

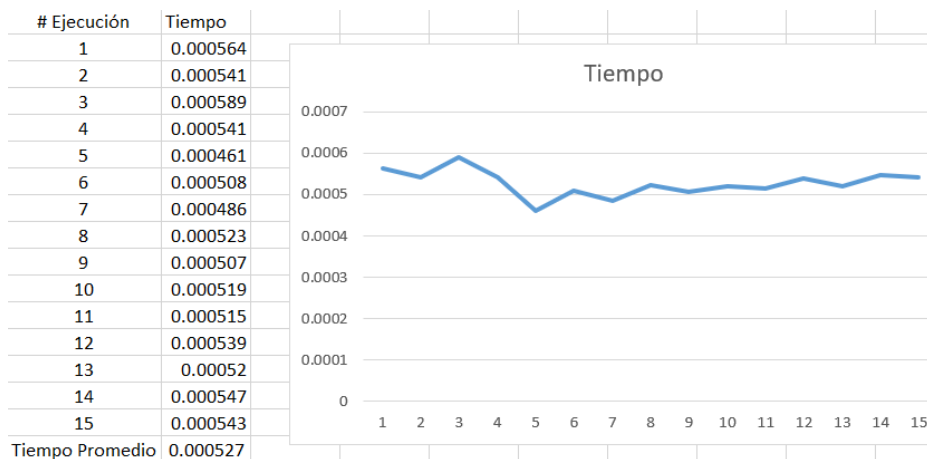


(1.3.5) Tiempo VM

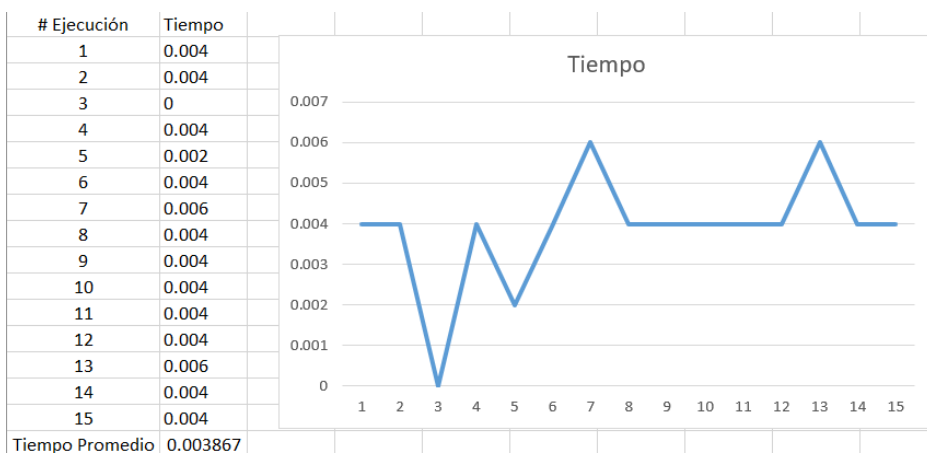


(1.3.6) Tiempo LAPTOP

12 Threads

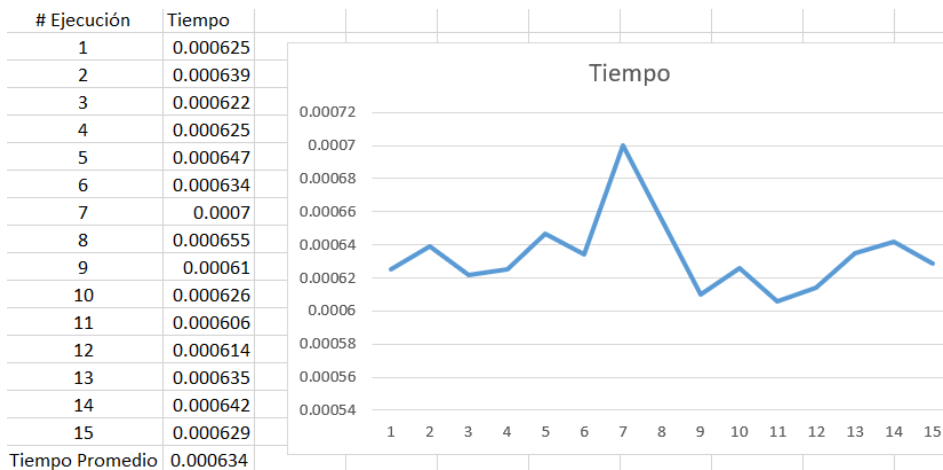


(1.3.7) Tiempo VM

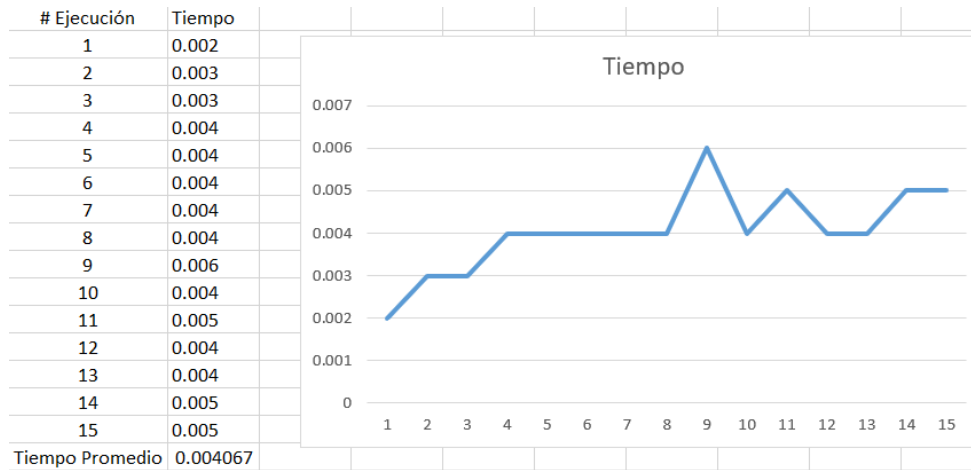


(1.3.8) Tiempo LAPTOP

15 Threads

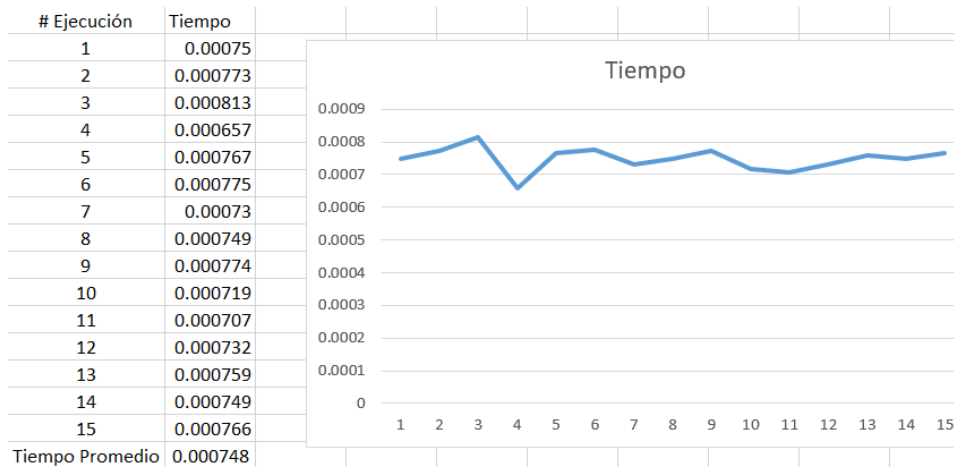


(1.3.9) Tiempo VM

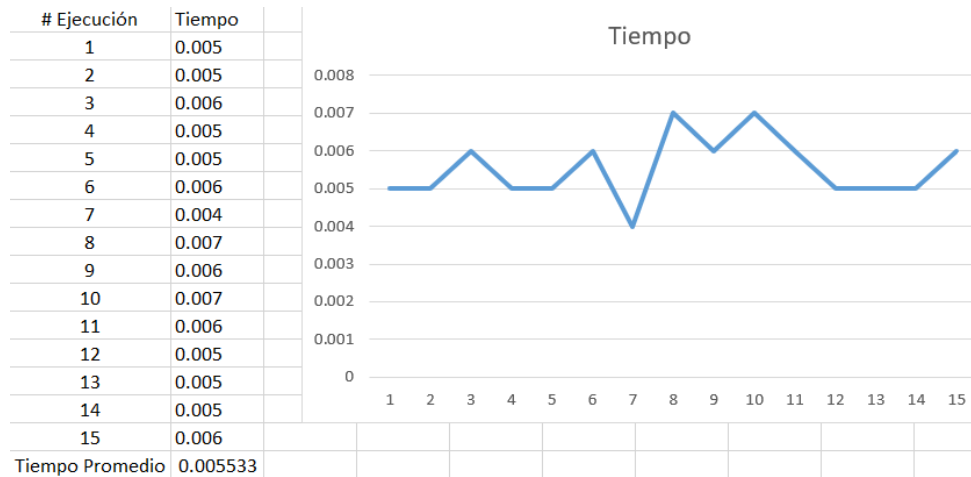


(1.3.10) Tiempo LAPTOP

18 Threads



(1.3.11) Tiempo VM



(1.3.12) Tiempo LAPTOP

Demostrando las siguientes comparaciones (Basándose en el tiempo promediado):

Ejecución	VM	Laptop	Resultado	Interpretación
3 Threads	0.00014	0.0008	0.175	17.5% Rendimiento de mi laptop
6 Threads	0.000291	0.001867	0.1558	15.58% Rendimiento de mi laptop
9 Threads	0.0004	0.0034	0.1176	11.76% Rendimiento de mi laptop
12 Threads	0.000527	0.003867	0.1362	13.62% Rendimiento de mi laptop
15 Threads	0.000629	0.004067	0.1546	15.46% Rendimiento de mi laptop
18 Threads	0.000748	0.005533	0.1351	13.51% Rendimiento de mi laptop

Podemos concluir de forma directa que el procesamiento de la máquina virtual es superior en todas las pruebas, sobrepasando en más del 80% el rendimiento de mi laptop en las pruebas. Lo cual nos da un promedio de rendimiento por parte de mi laptop en un 14.57%, lo cual representa que mi equipo personal es apenas una décima parte de lo que es el procesamiento y rendimiento de la máquina virtual.

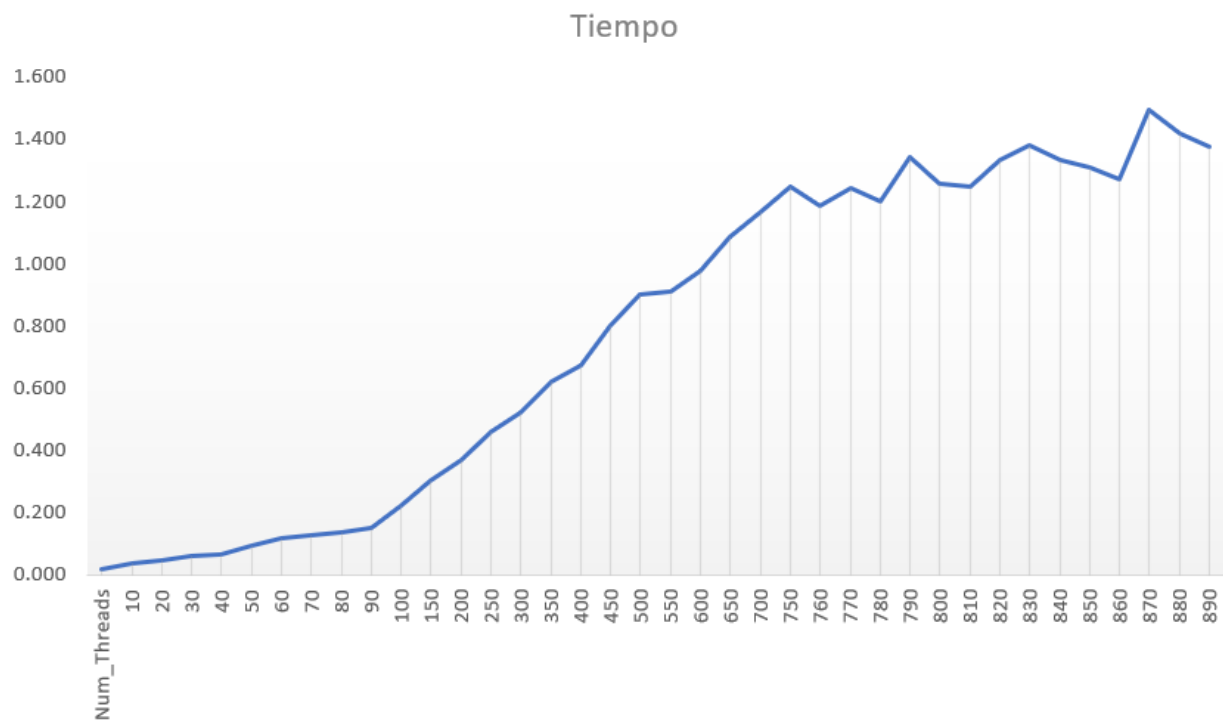
Actividad 1.4

El segundo ejercicio, se enfoca en los tiempos de ejecución en base a los threads disponibles, utilizando la función del área bajo la curva.

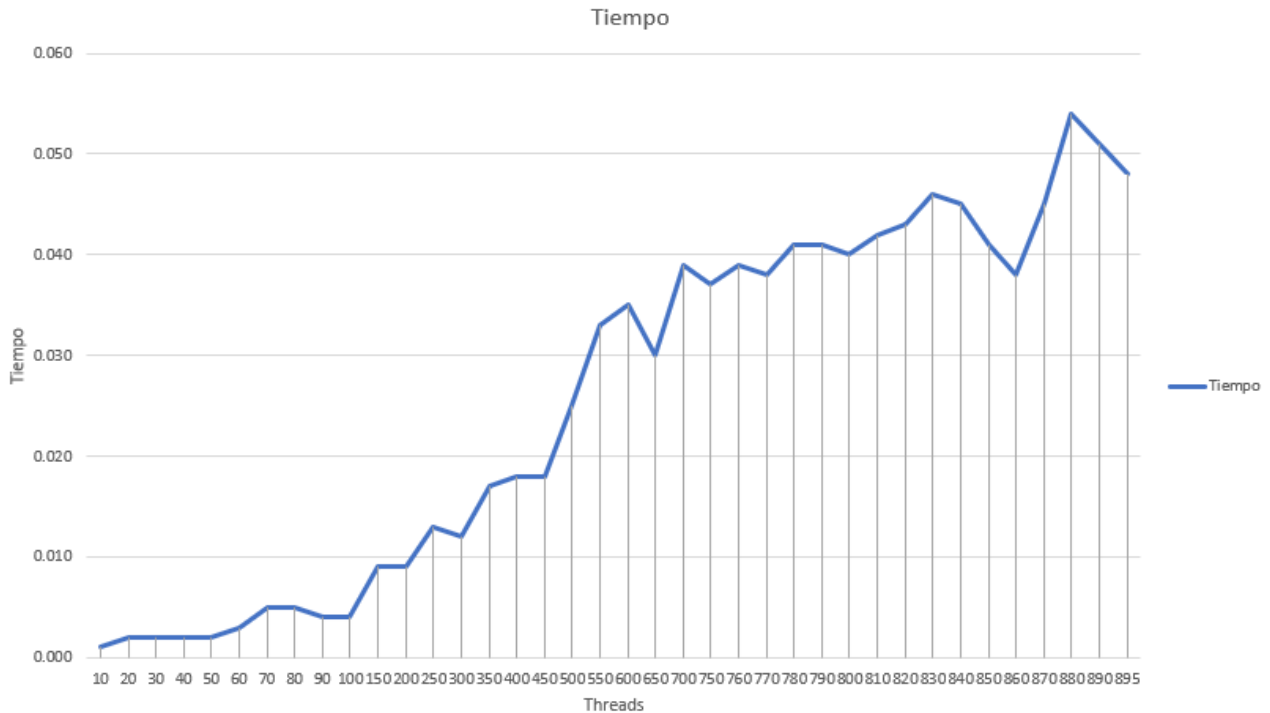
Se utilizará el siguiente código:

```
#include <stdio.h>
#include <omp.h>
#include <math.h>
#define NUM_THREADS 10 //Cambiar los threads
int main(){
    double tiempo, t1, t2;
    double divisiones = 100000; //10 ^ 5
    double divisor = 1.0 / divisiones;
    double total = 0.0;
    omp_set_num_threads(NUM_THREADS);
    const double startTime = omp_get_wtime();
    t1 = omp_get_wtime();
    #pragma omp parallel
    {
        for (double i = 0.0; i < 1.0; i = i + divisor){
            double funcion = (4.0 / (1.0 + pow(i,2))) * divisor;
            total = total + funcion;
        }
    }
    const double endTime = omp_get_wtime();
    tiempo = endTime-startTime;
    printf("Tiempo total = %lf\n", (tiempo));
}
```

Se inicializa con un incremento pequeño en series de 10, hasta llegar a 100 threads de donde las series saltan de 50.



(1.4.1) Tiempo VM



(1.4.2) Tiempo LAPTOP

Obteniendo los siguientes resultados para los tiempos

Ejecución	VM	Laptop	Resultado	Interpretación
Promedio	0.777	0.026	0.0334	3.34% Rendimiento de mi laptop

Si bien concluimos que mi laptop supera totalmente a la máquina virtual por un 96.66% cabe resaltar que la máquina virtual todavía tiene la posibilidad de ir más allá de los threads límites de mi laptop. Lo cual significa que en un ambiente reducido de pruebas (Numero finito de pruebas) mi laptop resulta efectiva pues es capaz de procesar con eficacia un número máximo de threads, sin embargo, si se trata de una prueba de rendimiento mi laptop no sería capaz de competir pues la máquina virtual puede ir mucho más allá a cambio de tiempos de ejecución. Demostrando lo significativo que es el elegir un procesador cuando se trata de actividades específicas como lo es el procesamiento de datos y funciones en cantidades altas.

Actividad 1.5

El tercer ejercicio se enfoca en tiempos de procesamiento con diferentes funciones matemáticas, con el objetivo de resaltar la función en la cual tenga menor tiempo. Teniendo el enfoque de resaltar los tiempos de la memoria RAM en la ejecución, del disco duro y como impacta en ejecuciones multithreads.

El siguiente código fue proporcionado por Abril Jiménez Alatraste A01732412 [2], el cual tiene el propósito de medir la velocidad RAM por ejecución de función.

```

#include <stdio.h>
#include <stdlib.h>
#include <omp.h>
#include <math.h>
#include <malloc.h>
FILE *fptr;
double** tab1;
double** tab2;
double** tab3;
double** tab4;
double** tab5;
double** tab6;

#define NUM_THREADS 6

double** euler_method(int N)
{
    printf("Numero de pasos:%d Atendido por thread:%d\n",
N,omp_get_thread_num());
    double w0=0.5,a=0,b=2;
    int i;
    double** data = malloc(sizeof(double*) * 2);
    double* w = malloc(sizeof(double) * N);
    double* t = malloc(sizeof(double) * N);
    double h,ab;

    h=(b-a)/N;
    w[0] = w0;
    t[0] = a;
    for(i=0;i<N;i++)
    {
        t[i]=a+(h*i);
        w[i]=w[i-1]+h*(1+t[i-1]*t[i-1]-w[i-1]); //Operaciones matematicas
    }
    *(data) = t;
    *(data+1) = w;
    return data;
}

void writeArr(double** arr, FILE *fp){
    size_t size = _msize(*arr)/sizeof arr[0][0];
    printf("%d\n", size);
    fprintf(fp, "Datos que encuentra el metodo de Euler:\nN\t\tX\t\t\t\t\t Y\n");
    for(int j = 0; j < size; j++){
        fprintf(fp, "%03i\t%.4lf\t%.4lf\n",j, arr[0][j], arr[1][j]);
    }
}

```



```

    }
}

int main(){
    omp_set_num_threads(NUM_THREADS); //TIEMPO RAM
    const double startTime = omp_get_wtime();
    #pragma omp parallel
    {
        #pragma omp sections
        {
            #pragma omp section
            tab1 = euler_method(185529);
            #pragma omp section
            tab2 = euler_method(183674);
            #pragma omp section
            tab3 = euler_method(164190);
            #pragma omp section
            tab4 = euler_method(121522);
            #pragma omp section
            tab5 = euler_method(119667);
            #pragma omp section
            tab6 = euler_method(225418);
        }
    }
    const double endTime = omp_get_wtime();
    printf("En un tiempo total de (%lf)\n", (endTime - startTime));

    fptr=fopen("Euler_n_0.txt","w");
    writeArr(tab1,fptr);
    fclose(fptr);

    fptr=fopen("Euler_n_1.txt","w");
    writeArr(tab2,fptr);
    fclose(fptr);

    fptr=fopen("Euler_n_2.txt","w");
    writeArr(tab3,fptr);
    fclose(fptr);

    fptr=fopen("Euler_n_3.txt","w");
    writeArr(tab4,fptr);
    fclose(fptr);

    fptr=fopen("Euler_n_4.txt","w");
    writeArr(tab5,fptr);

```

```

fclose(fptr);

fptr=fopen("Euler_n_5.txt","w");
writeArr(tab6,fptr);
fclose(fptr);
}

```

Cabe resaltar que la línea comentada con “Operaciones matemáticas” fue editada manualmente para incluir las siguientes funciones por ejecución.

```

// "exp"
w[i] = exp2((w[i-1]+h)*(1+t[i-1]*(t[i-1]-w[i-1])));
// "pow"
w[i] = w[i-1]+h*(1+t[i-1]*pow(t[i-1]-w[i-1], 2));
// "sin"
w[i] = sin(w[i-1]+h*(1+t[i-1]*t[i-1]-w[i-1]));
// "cos"
w[i] = cos(w[i-1]+h*(1+t[i-1]*t[i-1]-w[i-1]));
// "sqrt"
w[i] = sqrt(w[i-1]+h*(1+t[i-1]*t[i-1]-w[i-1]));
// "sinh"
w[i] = sinh(w[i-1]+h*(1+t[i-1]*t[i-1]-w[i-1]));
// "cosh"
w[i] = cosh(w[i-1]+h*(1+t[i-1]*t[i-1]-w[i-1]));

```

Como caso de lectura de disco duro base se usó el siguiente código, proporcionado por el profesor en las clases/actividades de salón:

```

#include <stdio.h>
#include <omp.h>
#include <math.h>
#define NUM_THREADS 6
FILE *fptr;
FILE *fptr1;
FILE *fptr2;
FILE *fptr3;
FILE *fptr4;
FILE *fptr5;
void iteracion(int N,FILE *x);
int main()
{
    omp_set_num_threads(NUM_THREADS); //TIEMPO DISCO DURO
    fptr=fopen("Euler_n_0.txt","w");
    fptr1=fopen("Euler_n_1.txt","w");
    fptr2=fopen("Euler_n_2.txt","w");
    fptr3=fopen("Euler_n_3.txt","w");
    fptr4=fopen("Euler_n_4.txt","w");
    fptr5=fopen("Euler_n_5.txt","w");

```

```

const double startTime = omp_get_wtime();
#pragma omp parallel
{
    #pragma omp sections
    {
        #pragma omp section
        (void)iteracion(80,fptr); //20//2000900
        #pragma omp section
        (void)iteracion(85,fptr1); //200//2000400
        #pragma omp section
        (void)iteracion(82,fptr2); //2000//2000700
        #pragma omp section
        (void)iteracion(83,fptr3); //20000//2000800
        #pragma omp section
        (void)iteracion(84,fptr4); //200000//2000500
        #pragma omp section
        (void)iteracion(86,fptr5); //2000000//2000100
    }
}
fclose(fptr);
fclose(fptr1);
fclose(fptr2);
fclose(fptr3);
fclose(fptr4);
fclose(fptr5);
const double endTime = omp_get_wtime();
printf("En un tiempo total de (%lf)\n", (endTime - startTime));
return (0);
}

void iteracion(int N, FILE *x)
{
    printf("Numero de pasos:%d Atendido por thread:%d\n",
N,omp_get_thread_num());
    fprintf(x, "Datos que encuentra el metodo de Euler(variable ind.\t variable
dep.\t numero de thread)\n");
    double h,t,w,ab;
    double w0=0.5,a=0,b=2;
    int i;
    w=w0;
    fprintf(x, "%f\t %f\n", a, w);
    for(i=0;i<N;i++){
        h=(b-a)/N;
        t=a+(h*i);
        ab=t*t;
    }
}

```

```

        w=w+h*(1+pow(t-w,2));
        fprintf(x, "%f\t %f \t numero de thread:%d\n", t+h,
w,omp_get_thread_num());
    }
}

```

Velocidad Disco Duro

```

Numero de pasos:80 Atendido por thread:1
Numero de pasos:85 Atendido por thread:2
Numero de pasos:82 Atendido por thread:0
Numero de pasos:83 Atendido por thread:3
Numero de pasos:84 Atendido por thread:5
Numero de pasos:86 Atendido por thread:4
En un tiempo total de (0.002000)

```

(1.5.1) Velocidad Disco Duro

```

Numero de pasos:80 Atendido por thread:1
Numero de pasos:85 Atendido por thread:2
Numero de pasos:82 Atendido por thread:4
Numero de pasos:83 Atendido por thread:5
Numero de pasos:84 Atendido por thread:3
Numero de pasos:86 Atendido por thread:1
En un tiempo total de (0.000492)

```

(1.5.2) Velocidad Disco Duro VM

Velocidad RAM

Base

```

PS C:\Users\juanm\Documents\Tec2021\Semestre9\Multi\Act 1.5> ./act15
Numero de pasos:183674 Atendido por thread:4
Numero de pasos:185529 Atendido por thread:1
Numero de pasos:164190 Atendido por thread:0
Numero de pasos:121522 Atendido por thread:2
Numero de pasos:119667 Atendido por thread:3
Numero de pasos:225418 Atendido por thread:5
En un tiempo total de (0.008000)

```

(1.5.3) Tiempo LAPTOP

```

Numero de pasos:183674 Atendido por thread:2
Numero de pasos:164190 Atendido por thread:5
Numero de pasos:121522 Atendido por thread:3
Numero de pasos:119667 Atendido por thread:0
Numero de pasos:225418 Atendido por thread:3
Numero de pasos:185529 Atendido por thread:1
En un tiempo total de (0.006423)

```

(1.5.4) Tiempo VM

Exponente

```
PS C:\Users\juanm\Documents\Tec2021\Semestre9\Multi\Act 1.5> ./act15
Numero de pasos:121522 Atendido por thread:3
Numero de pasos:185529 Atendido por thread:1
Numero de pasos:183674 Atendido por thread:0
Numero de pasos:119667 Atendido por thread:4
Numero de pasos:164190 Atendido por thread:2
Numero de pasos:225418 Atendido por thread:5
En un tiempo total de (0.012000)
```

(1.5.5) Tiempo LAPTOP

```
Numero de pasos:183674 Atendido por thread:2
Numero de pasos:185529 Atendido por thread:1
Numero de pasos:164190 Atendido por thread:4
Numero de pasos:121522 Atendido por thread:5
Numero de pasos:119667 Atendido por thread:3
Numero de pasos:225418 Atendido por thread:0
En un tiempo total de (0.013245)
```

(1.5.6) Tiempo VM

Cuadrados

```
PS C:\Users\juanm\Documents\Tec2021\Semestre9\Multi\Act 1.5> ./act15
Numero de pasos:185529 Atendido por thread:1
Numero de pasos:183674 Atendido por thread:2
Numero de pasos:164190 Atendido por thread:4
Numero de pasos:121522 Atendido por thread:0
Numero de pasos:119667 Atendido por thread:3
Numero de pasos:225418 Atendido por thread:5
En un tiempo total de (0.020000)
```

(1.5.7) Tiempo LAPTOP

```
Numero de pasos:185529 Atendido por thread:1
Numero de pasos:183674 Atendido por thread:2
Numero de pasos:225418 Atendido por thread:2
Numero de pasos:121522 Atendido por thread:4
Numero de pasos:164190 Atendido por thread:5
Numero de pasos:119667 Atendido por thread:3
En un tiempo total de (0.008581)
```

(1.5.8) Tiempo VM

Raíz Cuadrada

```
PS C:\Users\juanm\Documents\Tec2021\Semestre9\Multi\Act 1.5> ./act15
Numero de pasos:185529 Atendido por thread:1
Numero de pasos:183674 Atendido por thread:2
Numero de pasos:164190 Atendido por thread:0
Numero de pasos:121522 Atendido por thread:4
Numero de pasos:119667 Atendido por thread:3
Numero de pasos:225418 Atendido por thread:5
En un tiempo total de (0.012000)
```

(1.5.9) Tiempo LAPTOP

```

Numero de pasos:185529 Atendido por thread:1
Numero de pasos:183674 Atendido por thread:2
Numero de pasos:164190 Atendido por thread:4
Numero de pasos:225418 Atendido por thread:0
Numero de pasos:121522 Atendido por thread:5
Numero de pasos:119667 Atendido por thread:3
En un tiempo total de (0.007237)

```

(1.5.10) Tiempo VM

Seno

```

PS C:\Users\juanm\Documents\Tec2021\Semestre9\Multi\Act 1.5> ./act15
Numero de pasos:183674 Atendido por thread:1
Numero de pasos:185529 Atendido por thread:0
Numero de pasos:164190 Atendido por thread:2
Numero de pasos:121522 Atendido por thread:3
Numero de pasos:119667 Atendido por thread:4
Numero de pasos:225418 Atendido por thread:5
En un tiempo total de (0.016000)

```

(1.5.11) Tiempo LAPTOP

```

Numero de pasos:185529 Atendido por thread:1
Numero de pasos:183674 Atendido por thread:2
Numero de pasos:164190 Atendido por thread:4
Numero de pasos:225418 Atendido por thread:0
Numero de pasos:121522 Atendido por thread:5
Numero de pasos:119667 Atendido por thread:3
En un tiempo total de (0.011936)

```

(1.5.12) Tiempo VM

Coseno

```

PS C:\Users\juanm\Documents\Tec2021\Semestre9\Multi\Act 1.5> ./act15
Numero de pasos:164190 Atendido por thread:2
Numero de pasos:183674 Atendido por thread:4
Numero de pasos:185529 Atendido por thread:1
Numero de pasos:121522 Atendido por thread:0
Numero de pasos:119667 Atendido por thread:3
Numero de pasos:225418 Atendido por thread:5
En un tiempo total de (0.016000)

```

(1.5.13) Tiempo LAPTOP

```

Numero de pasos:185529 Atendido por thread:1
Numero de pasos:183674 Atendido por thread:2
Numero de pasos:164190 Atendido por thread:4
Numero de pasos:121522 Atendido por thread:0
Numero de pasos:119667 Atendido por thread:5
Numero de pasos:225418 Atendido por thread:3
En un tiempo total de (0.010934)

```

(1.5.14) Tiempo VM

Secante

```
PS C:\Users\juanm\Documents\Tec2021\Semestre9\Multi\Act 1.5> ./act15
Numero de pasos:185529 Atendido por thread:1
Numero de pasos:183674 Atendido por thread:4
Numero de pasos:164190 Atendido por thread:2
Numero de pasos:121522 Atendido por thread:3
Numero de pasos:119667 Atendido por thread:0
Numero de pasos:225418 Atendido por thread:5
En un tiempo total de (0.100000)
```

(1.5.15) Tiempo LAPTOP

```
Numero de pasos:185529 Atendido por thread:1
Numero de pasos:183674 Atendido por thread:2
Numero de pasos:164190 Atendido por thread:5
Numero de pasos:225418 Atendido por thread:1
Numero de pasos:121522 Atendido por thread:4
Numero de pasos:119667 Atendido por thread:3
En un tiempo total de (0.007629)
```

(1.5.16) Tiempo VM

Cosecante

```
PS C:\Users\juanm\Documents\Tec2021\Semestre9\Multi\Act 1.5> ./act15
Numero de pasos:164190 Atendido por thread:0
Numero de pasos:185529 Atendido por thread:1
Numero de pasos:183674 Atendido por thread:2
Numero de pasos:121522 Atendido por thread:4
Numero de pasos:119667 Atendido por thread:3
Numero de pasos:225418 Atendido por thread:5
En un tiempo total de (0.100000)
```

(1.5.17) Tiempo LAPTOP

```
Numero de pasos:185529 Atendido por thread:1
Numero de pasos:164190 Atendido por thread:4
Numero de pasos:121522 Atendido por thread:3
Numero de pasos:183674 Atendido por thread:2
Numero de pasos:119667 Atendido por thread:5
Numero de pasos:225418 Atendido por thread:0
En un tiempo total de (0.007749)
```

(1.5.18) Tiempo VM

Ejercicio	VM	LAPTOP	Resultado	Interpretación
Disco Duro	0.00492	0.002	0.4065	40.65% Rendimiento de la VM
RAM Base	0.00632	0.008	0.79	79% Rendimiento de mi laptop
RAM Exp	0.01324	0.012	0.9063	90.63% Rendimiento de la VM
RAM Pow	0.00858	0.02	0.429	42.9% Rendimiento de mi laptop
RAM Raiz	0.00723	0.012	0.6025	60.25% Rendimiento de mi laptop
RAM COS	0.01093	0.016	0.6831	68.31% Rendimiento de mi laptop
RAM SIN	0.00119	0.016	0.0743	7.43% Rendimiento de mi laptop
RAM COSEC	0.00774	0.1	0.0774	7.74% Rendimiento de mi laptop
RAM SEC	0.00762	0.1	0.0762	7.62% Rendimiento de mi laptop

Seleccionando el tiempo más rápido, encontramos que la diferencia sobrepasa al 92% mi laptop, pero en cambio, la ejecución mas lenta de la máquina virtual existe una diferencia negativa en el rendimiento del 10%. Es posible concluir que la máquina virtual, si bien funciona y sobrepasa mi laptop en 6 de 8 pruebas, existen tareas pequeñas o específicas en las cuales mi laptop puede estar mejor optimizada para el manejo de cálculos como lo son las funciones exponenciales. También se toma en cuenta que el disco duro de mi laptop rinde un 60% mas efectivo que la máquina virtual, sin embargo, en ejecuciones RAM es que no satisface la velocidad de la máquina virtual.

Actividad 2.1

El cuarto ejercicio se basa en procesamiento de imágenes y rotaciones con multithreading. Destacando la velocidad en que puede manejar mapas de bits para la asignación de bit por bit en un determinado numero de threads y que tan eficaz o certero es en la asignación de dichos bits. Pues se manejan imágenes a color que deben ser segmentadas en R, G, B para así asignar cada uno de sus bits a la posición correspondiente y degradar los colores a grises.

El siguiente código fue proporcionado por Alan Fernández Valdivieso A00825627 [1], el cual obtiene la imagen, la segmenta, reasigna valores y posiciones. Creando una imagen en blanco y negro para entonces invertir la imagen de forma vertical y horizontal.

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <math.h>
#include <malloc.h>
#include "omp.h"

// Standard values located at the header of an BMP file
#define MAGIC_VALUE    0X4D42
//Bit depth
#define BITS_PER_PIXEL 24
#define NUM_PLANE      1
#define COMPRESSION    0
#define BITS_PER_BYTE  8
//OpenMP var
#define NUM_THREADS 200

#pragma pack(1)

/*Section used to declare structures*/
typedef struct{
    uint16_t type;
    uint32_t size;
    uint16_t reserved1;
    uint16_t reserved2;
    uint32_t offset;
    uint32_t header_size;
```



```

uint32_t width;
uint32_t height;
uint16_t planes;
uint16_t bits;
uint32_t compression;
uint32_t imagesize;
uint32_t xresolution;
uint32_t yresolution;
uint32_t importantcolours;
}BMP_Header;

typedef struct{
    BMP_Header header;
    unsigned int pixel_size;
    unsigned int width;
    unsigned int height;
    unsigned int bytes_per_pixel;
    unsigned char * pixel; //For future allocation in memory
}BMP_Image;

/*Section used to declare functions*/
int checkHeader(BMP_Header *);
BMP_Image* cleanUp(FILE *, BMP_Image *);
BMP_Image* BMP_open(const char *);
int BMP_save(const BMP_Image *img, const char *filename);
void BMP_destroy(BMP_Image *img);
static int RGB2Gray(unsigned char, unsigned char, unsigned char);
void BMP_gray(BMP_Image*);

/*End section*/

int checkHeader(BMP_Header *hdr){
    if((hdr -> type) != MAGIC_VALUE)                {printf("No es un bmp\n"); return
0;}
    if((hdr -> bits) != BITS_PER_PIXEL)              {printf("Revisa bit depth\n");
return 0;}
    if((hdr -> planes) != NUM_PLANE)                  {printf("Array de diferente
dimensiones\n"); return 0;}
    if((hdr -> compression) != COMPRESSION)         {printf("Hay compresion\n"); return
0;}
    return 1;
}

/*Funtion used as cleaner, in incorrect format of an image*/
BMP_Image * cleanUp(FILE * fptr, BMP_Image * img)

```

```

{
    if (fptr != NULL)
    {
        fclose (fptr);
    }
    if (img != NULL)
    {
        if (img -> pixel != NULL)
        {
            free (img -> pixel);
        }
        free (img);
    }
    return NULL;
}

BMP_Image* BMP_open(const char *filename){
    FILE *fptr = NULL;
    BMP_Image *img = NULL;
    fptr = fopen(filename, "rb");
    if(fptr == NULL){printf("Archivo no existe\n"); return cleanUp(fptr,img);}
    img = malloc(sizeof(BMP_Image));
    if(img == NULL){return cleanUp(fptr,img);}
    if(fread(&(img -> header), sizeof(BMP_Header),1,fptr) != 1) {printf("Header no
disponible\n"); return cleanUp(fptr,img);}
    if(checkHeader(&(img -> header)) == 0) {printf("Header fuera del estandar\n");
return cleanUp(fptr,img);}
    img -> pixel_size      = (img -> header).size - sizeof(BMP_Header);
    img -> width           = (img -> header).width;
    img -> height          = (img -> header).height;
    img -> bytes_per_pixel = (img -> header).bits/BITS_PER_BYTE;
    img -> pixel = malloc(sizeof(unsigned char) * (img -> pixel_size));
    if((img -> pixel) == NULL){printf("Imagen vacia\n"); return cleanUp(fptr,img);}
    if(fread(img->pixel, sizeof(char), img -> pixel_size,fptr) != (img ->
pixel_size)){printf("Imagen con contenido irregular \n");return
cleanUp(fptr,img);}
    char onebyte;
    if(fread(&onebyte,sizeof(char),1,fptr) != 0) {printf("Hay pixeles
residuales\n"); return cleanUp(fptr,img);}
    fclose(fptr);
    return img;
}

int BMP_save(const BMP_Image *img, const char *filename){
    FILE *fptr = NULL;

```

```

    fptr = fopen(filename, "wb");
    if(fptr == NULL) {return 0;} //Maybe you should write the header first
    if(fwrite(&(img -> header), sizeof(BMP_Header), 1, fptr) != 1) {fclose(fptr);
return 0;}
    if(fwrite(img->pixel, sizeof(char), img -> pixel_size, fptr) != (img ->
pixel_size)) {fclose(fptr); return 0;}
    fclose(fptr);
    return 1;
}

void BMP_destroy(BMP_Image *img){
    free (img -> pixel);
    free (img);
}

void specs(BMP_Image* img){
    printf("Image width: %i\n", img->width);
    printf("Image height: %i\n", abs(img->height));
    printf("Image BPP: %i\n", img->bytes_per_pixel);
    printf("Image size: %i\n", img->pixel_size);
}

static int RGB2Gray(unsigned char red, unsigned char green, unsigned char blue){
    double gray = 0.2989*red + 0.5870*green + 0.1140*blue;
    // double gray = 0.21*red + 0.72*green + 0.07*blue;
    //0.21*r+0.72*g+0.07*b
    return (int) gray;
}

void BMP_gray(BMP_Image *img)
{
    specs(img);
    omp_set_num_threads(NUM_THREADS);
    const double startTime = omp_get_wtime();

    #pragma omp parallel
    {
        #pragma omp for /*No wait*/
        for (int px1 = 0; px1 < (img -> pixel_size); px1 += 3)
        {
            unsigned char gray = RGB2Gray(img -> pixel[px1 + 2], img -> pixel[px1 +
1], img -> pixel[px1]);
            img -> pixel[px1 + 2] = gray; //Red pixel
            img -> pixel[px1 + 1] = gray; //Green pixel
            img -> pixel[px1] = gray; //Blue pixel
        }
    }
}

```

```

    }
}
if (BMP_save(img, "GrayScale.bmp") == 0)
{
    printf("Output file invalid!\n");
    BMP_destroy(img);
}
// Destroy the BMP image
BMP_destroy(img);
const double endTime = omp_get_wtime();
printf("En un tiempo total de (%lf)\n", (endTime - startTime));
}

void BMP_horFlip(BMP_Image *img){
    // printf("Crear la matriz de pixeles\n");
    unsigned char * output = (unsigned char *) malloc(img->pixel_size *
sizeof(unsigned char)); //Allocating memory for matrix
    unsigned int r = abs(img->height);
    unsigned int c = img->width*3;
    omp_set_num_threads(NUM_THREADS);
    const double startTime = omp_get_wtime();
    // printf("(%ix%i)", r, c);
    // Modificar el array
    // printf("Modificar la posicion de pixeles\n");
    #pragma omp parallel
    {
        #pragma omp for
        for(int i = 0; i < r; i++){
            for(int j = c; j > 0; j--){
                output[i*c+(c-j)] = img->pixel[i*c+j];
                // printf("(%ix%i)", i, j);
            }
            // printf("\n");
        }
        // Modificar la imagen
        // printf("Sobrescribir la modificacion\n");
        #pragma omp for
        for(int i = 0; i < r; i++){
            for(int j = 0; j < c; j++){
                img->pixel[i*c+j] = output[i*c+j];
            }
        }
    }
    // Guardar la imagen
    if (BMP_save(img, "HorizontalRot.bmp") == 0)

```

```

    {
        printf("Output file invalid!\n");
        BMP_destroy(img);
        free(output);
    }
    // Destroy the BMP image
    BMP_destroy(img);
    free(output);
    const double endTime = omp_get_wtime();
    printf("En un tiempo total de (%lf)\n", (endTime - startTime));
}

void BMP_verFlip(BMP_Image *img){
    // printf("Crear la matriz de pixeles\n");
    unsigned char * output = (unsigned char *) malloc(img->pixel_size *
    sizeof(unsigned char)); //Allocating memory for matrix
    int r = abs(img->height);
    int c = img->width*3;
    omp_set_num_threads(NUM_THREADS);
    const double startTime = omp_get_wtime();
    // printf("(%ix%i)", r, c);
    //Modificar el array
    // printf("Modificar la posicion de pixeles\n");
    #pragma omp parallel
    {
        #pragma omp for
        for(int i = r-1; i>0; i--){
            for(int j = 0; j<=c; j++){
                // if(j%c == 0) printf("(%ix%i)", i, j);
                output[(r-i)*c+j] = img->pixel[i*c+j];
            }
            // printf("\n");
        }
        //Modificar la imagen
        // printf("Sobrescribir la modificacion\n");
        #pragma omp for
        for(int i = 0; i<r; i++){
            for(int j = 0; j<c; j++){
                img->pixel[i*c+j] = output[i*c+j];
            }
        }
    }
    //Guardar la imagen
    if (BMP_save(img, "VerticalRot.bmp") == 0)
    {

```

```

        printf("Output file invalid!\n");
        BMP_destroy(img);
        free(output);
    }
    // Destroy the BMP image
    BMP_destroy(img);
    free(output);
    const double endTime = omp_get_wtime();
    printf("En un tiempo total de (%lf)\n", (endTime - startTime));
}

/*Debugging functions*/
void printArr(unsigned char * array, unsigned int size, unsigned int EPR,
unsigned int BPP){
    int i, k, h;
    // printf("\nRow = %i",size/(BPP*EPR));
    // printf("\nCol = %i",BPP*EPR);
    for(i = 0, h = -1, k=-1; i < size; i++){
        if(i%(size/(EPR*BPP)) == 0) k++; //Get column(w) index
        if(i%(EPR*BPP) == 0) h++; //Get row(h) index
    }
    printf("%i x %i",k,h);
}

int main(){
    BMP_gray(BMP_open("700.bmp"));
    // BMP_gray(BMP_open("hollow.bmp"));
    BMP_horFlip(BMP_open("GrayScale.bmp"));
    BMP_verFlip(BMP_open("GrayScale.bmp"));
    return 0;
}

```

Como parte de la actividad se debían comparar los tiempos de ejecución para una imagen estándar de 700x700 pixeles y una de mayor demanda arriba de los 2000 pixeles.

```

Image width: 3072
Image height: 2051
Image BPP: 3
Image size: 18902104
En un tiempo total de (0.030000)
En un tiempo total de (0.028000)
En un tiempo total de (0.028000)

```

(2.1.1) Tiempo LAPTOP 2000

```
Image width: 3072
Image height: 2051
Image BPP: 3
Image size: 18902104
En un tiempo total de (0.039410)
En un tiempo total de (0.054932)
En un tiempo total de (0.055761)
```

(2.1.2) Tiempo VM imagen 2000

```
Image width: 700
Image height: 700
Image BPP: 3
Image size: 1470088
En un tiempo total de (0.013000)
En un tiempo total de (0.004000)
En un tiempo total de (0.004000)
```

(2.1.3) Tiempo LAPTOP imagen 700

```
Image width: 700
Image height: 700
Image BPP: 3
Image size: 1470088
En un tiempo total de (0.007639)
En un tiempo total de (0.005414)
En un tiempo total de (0.005912)
```

(2.1.4) Tiempo VM imagen 700

Obteniendo los siguientes resultados:

Ejercicio	VM	LAPTOP	Resultado	Interpretación
700	0.019265	0.021	0.9173	91.73% Rendimiento de mi laptop
2000	0.150103	0.086	0.5729	57.29% Rendimiento de la VM

Podemos notar que, en una imagen de dimensiones pequeñas, tanto la máquina virtual como mi laptop trabajan casi a la par, salvo por una diferencia de un 9%. Sin embargo, al trabajar con mayores dimensiones es que la máquina virtual pierde a comparación de la laptop, teniendo casi una diferencia del 60%. Este resultado puede ser afectado por la cantidad de threads que se manejan o el procesador de cada equipo. Recordemos que la máquina virtual tiene un límite superior a la laptop en términos de threads y bajo estas limitaciones no funciona a su límite o de forma óptima.

Actividad 2.2

La última actividad reutiliza el código de la actividad previa, con la distinción que ahora se implementa una difuminación de imagen para crear un efecto de imágenes borrosas. Las cuales, unidas en secuencia por medio de un gif, es que se presenta la difuminación y efecto borroso de

estas. Este efecto debe ser hecho en una imagen blanco y negro (Que es reutilizada del ejercicio previo) y a la vez con dicha imagen rotada. Creando 2 gifs.

El siguiente código fue proporcionado por Alan Fernández Valdivieso A00825627 [1], el cual obtiene la imagen, la difumina en 11 imágenes separadas y al final son unidas por medio de un gif (De acceso libre en internet).

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <math.h>
#include <malloc.h>
#include "omp.h"
#include <string.h>

// Standard values located at the header of an BMP file
#define MAGIC_VALUE    0X4D42
//Bit depth
#define BITS_PER_PIXEL 24
#define NUM_PLANE      1
#define COMPRESSION    0
#define BITS_PER_BYTE  8
//OpenMP var
#define NUM_THREADS 200

#pragma pack(1)

/*Section used to declare structures*/
typedef struct{
    uint16_t type;
    uint32_t size;
    uint16_t reserved1;
    uint16_t reserved2;
    uint32_t offset;
    uint32_t header_size;
    uint32_t width;
    uint32_t height;
    uint16_t planes;
    uint16_t bits;
    uint32_t compression;
    uint32_t imagesize;
    uint32_t xresolution;
    uint32_t yresolution;
    uint32_t importantcolours;
}BMP_Header;
```



```

typedef struct{
    BMP_Header header;
    unsigned int pixel_size;
    unsigned int width;
    unsigned int height;
    unsigned int bytes_per_pixel;
    unsigned char * pixel; //For future allocation in memory
}BMP_Image;

/*Section used to declare functions*/
int checkHeader(BMP_Header *);
BMP_Image* cleanUp(FILE *, BMP_Image *);
BMP_Image* BMP_open(const char *);
int BMP_save(const BMP_Image *img, const char *filename);
void BMP_destroy(BMP_Image *img);
static int RGB2Gray(unsigned char, unsigned char, unsigned char);
void BMP_gray(BMP_Image*);

/*End section*/

int checkHeader(BMP_Header *hdr){
    if((hdr -> type) != MAGIC_VALUE)                {printf("No es un bmp\n"); return
0;}
    if((hdr -> bits) != BITS_PER_PIXEL)              {printf("Revisa bit depth\n");
return 0;}
    if((hdr -> planes) != NUM_PLANE)                  {printf("Array de diferente
dimensiones\n"); return 0;}
    if((hdr -> compression) != COMPRESSION)          {printf("Hay compresion\n"); return
0;}
    return 1;
}

/*Funtion used as cleaner, in incorrect format of an image*/
BMP_Image * cleanUp(FILE * fptr, BMP_Image * img)
{
    if (fptr != NULL)
    {
        fclose (fptr);
    }
    if (img != NULL)
    {
        if (img -> pixel != NULL)
        {
            free (img -> pixel);
        }
    }
}

```

```

        free (img);
    }
    return NULL;
}

BMP_Image* BMP_open(const char *filename){
    FILE *fptr = NULL;
    BMP_Image *img = NULL;
    fptr = fopen(filename, "rb");
    if(fptr == NULL){printf("Archivo no existe\n"); return cleanUp(fptr,img);}
    img = malloc(sizeof(BMP_Image));
    if(img == NULL){return cleanUp(fptr,img);}
    if(fread(&(img -> header), sizeof(BMP_Header),1,fptr) != 1) {printf("Header no
disponible\n"); return cleanUp(fptr,img);}
    if(checkHeader(&(img -> header)) == 0) {printf("Header fuera del estandar\n");
return cleanUp(fptr,img);}
    img -> pixel_size      = (img -> header).size - sizeof(BMP_Header);
    img -> width           = (img -> header).width;
    img -> height          = (img -> header).height;
    img -> bytes_per_pixel = (img -> header).bits/BITS_PER_BYTE;
    img -> pixel = malloc(sizeof(unsigned char) * (img -> pixel_size));
    if((img -> pixel) == NULL){printf("Imagen vacia\n"); return cleanUp(fptr,img);}
    if(fread(img->pixel, sizeof(char), img -> pixel_size,fptr) != (img ->
pixel_size)){printf("Imagen con contenido irregular \n");return
cleanUp(fptr,img);}
    char onebyte;
    if(fread(&onebyte,sizeof(char),1,fptr) != 0) {printf("Hay pixeles
residuales\n"); return cleanUp(fptr,img);}
    fclose(fptr);
    return img;
}

int BMP_save(const BMP_Image *img, const char *filename){
    FILE *fptr = NULL;
    fptr = fopen(filename, "wb");
    if(fptr == NULL) {return 0;}//Maybe you should write the header first
    if(fwrite(&(img -> header), sizeof(BMP_Header),1,fptr) != 1) {fclose(fptr);
return 0;}
    if(fwrite(img->pixel, sizeof(char), img -> pixel_size, fptr) != (img ->
pixel_size)) {fclose(fptr); return 0;}
    fclose(fptr);
    return 1;
}

void BMP_destroy(BMP_Image *img){

```

```

    free (img -> pixel);
    free (img);
}

void specs(BMP_Image* img){
    printf("Image width: %i\n", img->width);
    printf("Image height: %i\n", abs(img->height));
    printf("Image BPP: %i\n", img->bytes_per_pixel);
    printf("Image size: %i\n", img->pixel_size);
}

static int RGB2Gray(unsigned char red, unsigned char green, unsigned char blue){
    double gray = 0.2989*red + 0.5870*green + 0.1140*blue;
    // double gray = 0.21*red + 0.72*green + 0.07*blue;
    //0.21*r+0.72*g+0.07*b
    return (int) gray;
}

void BMP_gray(BMP_Image *img)
{
    specs(img);
    omp_set_num_threads(NUM_THREADS);
    const double startTime = omp_get_wtime();

    #pragma omp parallel
    {
        #pragma omp for /*No wait*/
        for (int pxl = 0; pxl < (img -> pixel_size); pxl += 3)
        {
            unsigned char gray = RGB2Gray(img -> pixel[pxl + 2],img -> pixel[pxl +
1],img -> pixel[pxl]);
            img -> pixel[pxl + 2] = gray; //Red pixel
            img -> pixel[pxl + 1] = gray; //Green pixel
            img -> pixel[pxl]      = gray; //Blue pixel
        }
    }
    if (BMP_save(img, "GrayScale.bmp") == 0)
    {
        printf("Output file invalid!\n");
        BMP_destroy(img);
    }
    // Destroy the BMP image
    BMP_destroy(img);
    const double endTime = omp_get_wtime();
    printf("En un tiempo total de (%lf)\n", (endTime - startTime));
}

```

```

}

void BMP_horFlip(BMP_Image *img){
    // printf("Crear la matriz de pixeles\n");
    unsigned char * output = (unsigned char *) malloc(img->pixel_size *
sizeof(unsigned char)); //Allocating memory for matrix
    unsigned int r = abs(img->height);
    unsigned int c = img->width*3;
    omp_set_num_threads(NUM_THREADS);
    const double startTime = omp_get_wtime();
    // printf("(%ix%i)", r, c);
    // Modificar el array
    // printf("Modificar la posicion de pixeles\n");
    #pragma omp parallel
    {
        #pragma omp for
        for(int i = 0; i < r; i++){
            for(int j = c; j > 0; j--){
                output[i*c+(c-j)] = img->pixel[i*c+j];
                // printf("(%ix%i)", i, j);
            }
            // printf("\n");
        }
        // Modificar la imagen
        // printf("Sobrescribir la modificacion\n");
        #pragma omp for
        for(int i = 0; i < r; i++){
            for(int j = 0; j < c; j++){
                img->pixel[i*c+j] = output[i*c+j];
            }
        }
    }
    // Guardar la imagen
    if (BMP_save(img, "HorizontalRot.bmp") == 0)
    {
        printf("Output file invalid!\n");
        BMP_destroy(img);
        free(output);
    }
    // Destroy the BMP image
    BMP_destroy(img);
    free(output);
    const double endTime = omp_get_wtime();
    printf("En un tiempo total de (%lf)\n", (endTime - startTime));
}

```

```

void BMP_verFlip(BMP_Image *img){
    // printf("Crear la matriz de pixeles\n");
    unsigned char * output = (unsigned char *) malloc(img->pixel_size *
sizeof(unsigned char)); //Allocating memory for matrix
    int r = abs(img->height);
    int c = img->width*3;
    omp_set_num_threads(NUM_THREADS);
    const double startTime = omp_get_wtime();
    // printf("(%ix%i)",r,c);
    //Modificar el array
    // printf("Modificar la posicion de pixeles\n");
    #pragma omp parallel
    {
        #pragma omp for
        for(int i = r-1; i>0; i--){
            for(int j = 0; j<=c; j++){
                // if(j%c == 0) printf("(%ix%i)",i,j);
                output[(r-i)*c+j] = img->pixel[i*c+j];
            }
            // printf("\n");
        }
        //Modificar la imagen
        // printf("Sobrescribir la modificacion\n");
        #pragma omp for
        for(int i = 0; i<r; i++){
            for(int j = 0; j<c; j++){
                img->pixel[i*c+j] = output[i*c+j];
            }
        }
    }
    //Guardar la imagen
    if (BMP_save(img, "VerticalRot.bmp") == 0)
    {
        printf("Output file invalid!\n");
        BMP_destroy(img);
        free(output);
    }
    // Destroy the BMP image
    BMP_destroy(img);
    free(output);
    const double endTime = omp_get_wtime();
    printf("En un tiempo total de (%lf)\n", (endTime - startTime));
}

```

```

/*Debugging functions*/
float ** kernel(unsigned int size){
    unsigned int height = size;
    unsigned int width = size*3;
    float ** matrix = malloc(sizeof(float*)*height);
    for(int i = 0; i < height; i++){
        matrix[i] = malloc(sizeof(float)*width);
    }

    for(int i = 0; i<height; i++){
        for(int j = 0; j<width; j++){
            matrix[i][j] = (float)1.0/(size*size);
        }
    }
    return matrix;
}

char ** pixelMat(BMP_Image * img){
    unsigned int height = img->height;
    unsigned int width = img->width*3;
    char** mat = malloc(sizeof(char*) * (height));
    for(int i = 0; i < height; i++){
        mat[i] = malloc(sizeof(char)*(width));
    }

    #pragma omp parallel
    {
        #pragma omp for schedule(dynamic, NUM_THREADS) collapse(2)
        for(int i=0; i < height; i++){
            for(int j=0; j< width; j++){
                mat[i][j] = img->pixel[i*width+j];
            }
        }
    }

    return mat;
}

void BMP_blur(char* open, unsigned int size){
    BMP_Image * img = BMP_open(open);
    char** out_buffer = pixelMat(img);
    float** kerSn = kernel(size);

    unsigned int height = img->height;
    unsigned int width = img->width * 3;

```

```

int M = (size-1)/2;

omp_set_num_threads(NUM_THREADS);
const double startTime = omp_get_wtime();

#pragma omp parallel
{
    #pragma omp for schedule(dynamic, NUM_THREADS) collapse(1)
    for(int x=M;x<height-M;x++)
    {
        for(int y=M;y<width-M;y++)
        {
            float sum= 0.0;
            for(int i=-M;i<=M;++i)
            {
                for(int j=-M;j<=M;++j)
                {
                    sum+=(float)kerSn[i+M][j+M]*img->pixel[(x+i)*width+(y+j)]; //matrix
multiplication with kernel
                }
            }
            out_buffer[x][y]=(char)sum;
        }
    }

    #pragma omp for schedule(dynamic, NUM_THREADS)
    for(int i = 1; i<height-1; i++){
        for(int j = 1; j<width-1; j++){
            img->pixel[i*width+j] = out_buffer[i][j];
        }
    }
}
char* name;
if(strcmp(open,"HorizontalRot.bmp") == 0){
    char filename[] = "Rblur0X.bmp";
    if(size < 10) filename[6]=size+'0';
    else{
        filename[5]= (size/10)+'0';
        filename[6]= (size%10)+'0';
    }
    name = filename;
}
else{
    char filename[] = "Blur0X.bmp";

```

```

    if(size < 10) filename[5]=size+'0';
    else{
        filename[4]= (size/10)+'0';
        filename[5]= (size%10)+'0';
    }
    name = filename;
}

//Guardar la imagen
if (BMP_save(img, name) == 0)
{
    printf("Output file invalid!\n");
    BMP_destroy(img);
    free(kerSn);
    free(out_buffer);
    // free(buffer);
}
// Destroy the BMP image
BMP_destroy(img);
free(kerSn);
free(out_buffer);
const double endTime = omp_get_wtime();
printf("%s teminado en un tiempo total de (%lf)\n",name, (endTime -
startTime));
}

/*Debugging functions*/
void printArr(float ** array, int size){
    for(int i = 0; i < size; i++){
        for(int j = 0; j < size*3; j++){
            if((j+1)%3 == 0) printf("%f\t", array[i][j]);
            else printf("%f,", array[i][j]);
        }
        printf("\n");
    }
}

int main(){
    omp_set_num_threads(NUM_THREADS);
    const double startTime = omp_get_wtime();

    #pragma omp sections
    {
        #pragma omp section
        BMP_blur("GrayScale.bmp",3);
    }
}

```



```

#pragma omp section
BMP_blur("GrayScale.bmp",5);
#pragma omp section
BMP_blur("GrayScale.bmp",7);
#pragma omp section
BMP_blur("GrayScale.bmp",9);
#pragma omp section
BMP_blur("GrayScale.bmp",11);
#pragma omp section
BMP_blur("GrayScale.bmp",13);
#pragma omp section
BMP_blur("GrayScale.bmp",15);
#pragma omp section
BMP_blur("GrayScale.bmp",17);
#pragma omp section
BMP_blur("GrayScale.bmp",19);
#pragma omp section
BMP_blur("GrayScale.bmp",21);
#pragma omp section
BMP_blur("GrayScale.bmp",23);
#pragma omp section
BMP_blur("HorizontalRot.bmp",23);
#pragma omp section
BMP_blur("HorizontalRot.bmp",21);
#pragma omp section
BMP_blur("HorizontalRot.bmp",19);
#pragma omp section
BMP_blur("HorizontalRot.bmp",17);
#pragma omp section
BMP_blur("HorizontalRot.bmp",15);
#pragma omp section
BMP_blur("HorizontalRot.bmp",13);
#pragma omp section
BMP_blur("HorizontalRot.bmp",11);
#pragma omp section
BMP_blur("HorizontalRot.bmp",9);
#pragma omp section
BMP_blur("HorizontalRot.bmp",7);
#pragma omp section
BMP_blur("HorizontalRot.bmp",5);
#pragma omp section
BMP_blur("HorizontalRot.bmp",3);
}
// BMP_gray(BMP_open("opossum.bmp"));
// BMP_horFlip(BMP_open("GrayScale.bmp"));

```

```

// BMP_Image * img = BMP_open("rata.bmp");
// specs(&(img->header));
return 0;
}

```

Obteniendo los siguientes resultados en ejecución:

```

Blur03.bmp teminado en un tiempo total de (0.422484)
Blur05.bmp teminado en un tiempo total de (0.907127)
Blur07.bmp teminado en un tiempo total de (1.688347)
Blur09.bmp teminado en un tiempo total de (2.619320)
Blur11.bmp teminado en un tiempo total de (3.790884)
Blur13.bmp teminado en un tiempo total de (5.216176)
Blur15.bmp teminado en un tiempo total de (6.901528)
Blur17.bmp teminado en un tiempo total de (8.814731)
Blur19.bmp teminado en un tiempo total de (11.523646)
Blur21.bmp teminado en un tiempo total de (13.944764)
Blur23.bmp teminado en un tiempo total de (16.592646)
Rblur23.bmp teminado en un tiempo total de (16.601597)
Rblur21.bmp teminado en un tiempo total de (13.995001)
Rblur19.bmp teminado en un tiempo total de (11.541724)
Rblur17.bmp teminado en un tiempo total de (8.793095)
Rblur15.bmp teminado en un tiempo total de (6.887861)
Rblur13.bmp teminado en un tiempo total de (5.213208)
Rblur11.bmp teminado en un tiempo total de (3.801613)
Rblur09.bmp teminado en un tiempo total de (2.623238)
Rblur07.bmp teminado en un tiempo total de (1.682332)
Rblur05.bmp teminado en un tiempo total de (0.907001)
Rblur03.bmp teminado en un tiempo total de (0.411439)

```

(2.2.1) Tiempo VM 2000

```

Blur03.bmp teminado en un tiempo total de (0.189000)
Blur05.bmp teminado en un tiempo total de (0.433000)
Blur07.bmp teminado en un tiempo total de (0.816000)
Blur09.bmp teminado en un tiempo total de (1.392000)
Blur11.bmp teminado en un tiempo total de (2.110000)
Blur13.bmp teminado en un tiempo total de (2.685000)
Blur15.bmp teminado en un tiempo total de (3.821000)
Blur17.bmp teminado en un tiempo total de (4.765000)
Blur19.bmp teminado en un tiempo total de (5.964000)
Blur21.bmp teminado en un tiempo total de (6.976000)
Blur23.bmp teminado en un tiempo total de (8.006000)
Rblur23.bmp teminado en un tiempo total de (7.970000)
Rblur21.bmp teminado en un tiempo total de (6.956000)
Rblur19.bmp teminado en un tiempo total de (5.810000)
Rblur17.bmp teminado en un tiempo total de (4.437000)
Rblur15.bmp teminado en un tiempo total de (3.482000)
Rblur13.bmp teminado en un tiempo total de (2.639000)
Rblur11.bmp teminado en un tiempo total de (1.888000)
Rblur09.bmp teminado en un tiempo total de (1.306000)
Rblur07.bmp teminado en un tiempo total de (0.825000)
Rblur05.bmp teminado en un tiempo total de (0.437000)
Rblur03.bmp teminado en un tiempo total de (0.188000)

```

(2.2.2) Tiempo LAPTOP 2000

```

Blur03.bmp teminado en un tiempo total de (0.032933)
Blur05.bmp teminado en un tiempo total de (0.076022)
Blur07.bmp teminado en un tiempo total de (0.135480)
Blur09.bmp teminado en un tiempo total de (0.207294)
Blur11.bmp teminado en un tiempo total de (0.299069)
Blur13.bmp teminado en un tiempo total de (0.406025)
Blur15.bmp teminado en un tiempo total de (0.530801)
Blur17.bmp teminado en un tiempo total de (0.676256)
Blur19.bmp teminado en un tiempo total de (0.897361)
Blur21.bmp teminado en un tiempo total de (1.086429)
Blur23.bmp teminado en un tiempo total de (1.278076)
Rblur23.bmp teminado en un tiempo total de (1.278398)
Rblur21.bmp teminado en un tiempo total de (1.102233)
Rblur19.bmp teminado en un tiempo total de (0.899130)
Rblur17.bmp teminado en un tiempo total de (0.675765)
Rblur15.bmp teminado en un tiempo total de (0.522995)
Rblur13.bmp teminado en un tiempo total de (0.403206)
Rblur11.bmp teminado en un tiempo total de (0.296898)
Rblur09.bmp teminado en un tiempo total de (0.205920)
Rblur07.bmp teminado en un tiempo total de (0.143418)
Rblur05.bmp teminado en un tiempo total de (0.073587)

```

(2.2.3) Tiempo VM 700

```

Blur03.bmp teminado en un tiempo total de (0.027000)
Blur05.bmp teminado en un tiempo total de (0.061000)
Blur07.bmp teminado en un tiempo total de (0.150000)
Blur09.bmp teminado en un tiempo total de (0.157000)
Blur11.bmp teminado en un tiempo total de (0.257000)
Blur13.bmp teminado en un tiempo total de (0.362000)
Blur15.bmp teminado en un tiempo total de (0.461000)
Blur17.bmp teminado en un tiempo total de (0.523000)
Blur19.bmp teminado en un tiempo total de (0.802000)
Blur21.bmp teminado en un tiempo total de (0.912000)
Blur23.bmp teminado en un tiempo total de (0.974000)
Rblur23.bmp teminado en un tiempo total de (1.054000)
Rblur21.bmp teminado en un tiempo total de (0.954000)
Rblur19.bmp teminado en un tiempo total de (0.779000)
Rblur17.bmp teminado en un tiempo total de (0.532000)
Rblur15.bmp teminado en un tiempo total de (0.484000)
Rblur13.bmp teminado en un tiempo total de (0.325000)
Rblur11.bmp teminado en un tiempo total de (0.250000)
Rblur09.bmp teminado en un tiempo total de (0.161000)
Rblur07.bmp teminado en un tiempo total de (0.095000)
Rblur05.bmp teminado en un tiempo total de (0.054000)
Rblur03.bmp teminado en un tiempo total de (0.031000)

```

(2.2.4) Tiempo LAPTOP 700

Ejercicio	VM	LAPTOP	Resultado	Interpretación
700	11.227296	9.405	0.8376	83.76% Rendimiento de la VM
2000	144.879762	73.095	0.5045	50.45% Rendimiento de la VM

Nuevamente notamos que la maquina virtual en una imagen pequeña esta cerca del rendimiento de la laptop, sin embargo, tiene un 16% de diferencia que no es necesariamente negativa, pero perjudicaría en el multiprocesamiento de imágenes (Como lo puede ser un cortometraje). Una

vez que se procesan imágenes de dimensiones grandes la máquina virtual palidece en contra de mi laptop pues rinde la mitad de esta. Entonces queda claro que en el procesamiento de imágenes mi equipo esta mejor optimizado, sin tomar en cuenta que se dejo un numero de threads estáticos que podrían impactar el procesamiento tanto de mi equipo como el de la máquina virtual.

Conclusión

Si bien en gran mayoría de las actividades destaca la máquina virtual por su velocidad y rendimiento al procesar números e imágenes, existen tareas específicas o de poca escalabilidad las cuales no necesariamente requieren un equipo optimizado para multitareas, si no para cálculos o procesos cortos que dan resultados efectivos y rápidos. Es posible inferir que el uso de una máquina virtual que ha sido optimizada para el procesamiento es una herramienta confiable y útil para todo caso, además que los recursos como puede ser el tiempo o memoria no se ven sacrificados. Por otra parte, es una fuerte inversión de recursos económicos los cuales no siempre se tienen a la mano y por ende se deben utilizar equipos de este estilo para tareas de proporciones enormes o cuya naturaleza en los cálculos, procesos y resultados requieran de una alta cantidad de recursos. Demostrando una diferencia tangible en cada una de estas actividades y como el procesador es un elemento crucial al momento de trabajar con equipos de cómputo, pues varía nuestra selección en torno a los recursos que se tienen a la mano o desean reducir. Tiempo, dinero, energía, etc...

Puedo concluir que mi equipo no esta totalmente optimizado para el manejo de multithreading y la velocidad que este tiene para las ejecuciones de tareas. Esto es debido a que por elección personal elegí un equipo que fuera capaz de soportar el procesamiento de unidades graficas y se mantuviese a la par sin sacrificar el rendimiento de mi unidad de procesamiento. Optando por un equipo que sea capaz de satisfacer mi criterio sin sacrificar su rendimiento general en otros ámbitos que no fuesen el procesamiento gráfico.

Referencias

[1] Fernández Valdivieso, A. (2022, noviembre). GitHub - AlFeVval/BMP-rotational-functions. GitHub. <https://github.com/AlFeVval/BMP-rotational-functions>

[2] Jiménez Alatríste, A. (2022, noviembre). GitHub - Xuxumin99/Multiprocesadores2022: Repositorio para la clase de Multiprocesadores Agosto-Diciembre 2022. GitHub. <https://github.com/Xuxumin99/Multiprocesadores2022>