



UNIVERSIDAD NACIONAL DE SAN AGUSTIN

ESCUELA PROFESIONAL DE  
CIENCIA DE LA COMPUTACIÓN  
ALGORITMOS PARALELOS

---

## Ejercicios en el supercomputador

---

***Alumna :***

*Rosa Yuliana Gabriela*

*Paccotacya Yanque*

***Profesor:***

*Mg. Alvaro Henry Mamani*

*Aliaga*

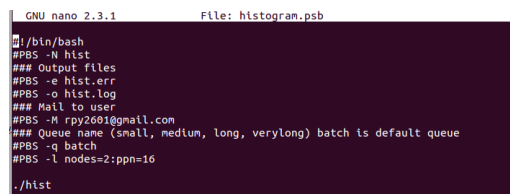
## Índice

<b>1. Script PSB</b>	<b>2</b>
<b>2. OpenMP</b>	<b>2</b>
2.1. Histograma . . . . .	2
2.2. Montecarlo . . . . .	3
2.3. CountSort . . . . .	3
<b>3. MPI</b>	<b>4</b>
3.1. Regla Trapezoidal . . . . .	4
3.2. Distribución y lectura de los datos de un vector . . . . .	4
3.2.1. Scatter . . . . .	5
3.2.2. Gather . . . . .	5
3.3. Output . . . . .	5
<b>4. Multiplicación de matriz con vector</b>	<b>5</b>
<b>5. PThreads</b>	<b>5</b>

Todas las pruebas y ejecuciones de código se realizaron en el supercomputador de la UNSA.

## 1. Script PSB

Script que se usa para realizar la compilación en el supercomputador, donde se indica el nombre del ejecutable.



```
GNU nano 2.3.1 File: histogram.psb
#!/bin/bash
#PBS -N hist
### Output files
#PBS -e hist.err
#PBS -o hist.log
### Mail to user
#PBS -M rpy2601@gmail.com
### Queue name (small, medium, long, verylong) batch is default queue
#PBS -q batch
#PBS -l nodes=2:ppn=16
./hist
```

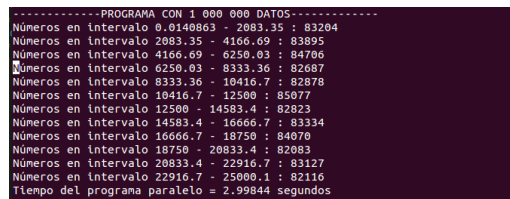
Figura 1: Script de histograma

## 2. OpenMP

Se muestran la comparación en los tiempos de ejecución de los diversos programas realizados con OpenMP

### 2.1. Histograma

Realiza un histograma contando las frecuencias en n intervalos de un conjunto de datos



```
-----PROGRAMA CON 1 000 000 DATOS-----
Números en Intervalo 0.0140863 - 2083.35 : 83204
Números en Intervalo 2083.35 - 4166.69 : 83895
Números en Intervalo 4166.69 - 6250.03 : 84706
Números en Intervalo 6250.03 - 8333.36 : 82687
Números en Intervalo 8333.36 - 10416.7 : 82878
Números en Intervalo 10416.7 - 12500 : 85077
Números en Intervalo 12500 - 14583.4 : 82823
Números en Intervalo 14583.4 - 16666.7 : 83334
Números en Intervalo 16666.7 - 18750 : 84070
Números en Intervalo 18750 - 20833.4 : 82083
Números en Intervalo 20833.4 - 22916.7 : 83127
Números en Intervalo 22916.7 - 25000.1 : 82116
Tiempo del programa paralelo = 2.99844 segundos
```

Figura 2: Histograma en mi PC

Despues de ejecutar en el supercomputador, se puede observar el ejecutable, y la cola en este.

```

-----PROGRAMA CON 1 000 000 DATOS-----
Números en intervalo 0.0140863 - 2083.35 : 83204
Números en intervalo 2083.35 - 4166.69 : 83895
Números en intervalo 4166.69 - 6250.03 : 84706
Números en intervalo 6250.03 - 8333.36 : 82687
Números en intervalo 8333.36 - 10416.7 : 82878
Números en intervalo 10416.7 - 12500 : 85077
Números en intervalo 12500 - 14583.4 : 82823
Números en intervalo 14583.4 - 16666.7 : 83334
Números en intervalo 16666.7 - 18750 : 84070
Números en intervalo 18750 - 20833.4 : 82083
Números en intervalo 20833.4 - 22916.7 : 83127
Números en intervalo 22916.7 - 25000.1 : 82116
Tiempo del programa paralelo = 2.99844 segundos

```

Figura 3: Histograma en el supercomputador

```

[rpaccotacya@inkari ~]$ ls
hist      hist.log      histogram.psb  hw.cpp  monte      montecarlo.cppo
hist.err  histogram.cpp  hw            hw.psb  montecarlo.cpp  montecarlo.psb

```

Figura 4: Archivos en el supercomputador

## 2.2. Montecarlo

Realiza el calculo de pi con millones de iteraciones

```

rose@satellite-555-A:~/CS_AlgoritmosParalelos/OPENMPS$ g++ -std=c++11 -O nm -fope
nmp montecarlo.cpp
rose@satellite-555-A:~/CS_AlgoritmosParalelos/OPENMPS$ ./nm
PI calculado: 3.14205 PI real: 3.14159 error: -0.000457882
Tiempo del programa paralelo = 0.131214 segundos

```

Figura 5: Montecarlo en mi PC

Despues de ejecutar en el supercomputador, se puede observar el ejecutable, y la cola en este.

```

[rpaccotacya@inkari ~]$ ls
hist      hist.log      histogram.psb  hw.cpp  monte      montecarlo.cppo
hist.err  histogram.cpp  hw            hw.psb  montecarlo.cpp  montecarlo.psb

```

Figura 6: Archivos en el supercomputador

Como se observa en la cola (Figura 8) archivo esta siendo ejecutado, por lo que no se pudo realizar mas ejecuciones.

## 2.3. CountSort

Algoritmo para ordenar CountSort, se realizó su implementacion y tambien se uso la brindada por la libreria QSort.

```
[rpaccotacya@lnkari ~]$ qstat
```

Job ID	Name	User	Time Use	S	Queue
350.lnkari	hist	rpaccotacya	0	Q	batch
351.lnkari	monte	rpaccotacya	0	Q	batch
352.lnkari	monte	rpaccotacya	0	Q	batch
353.lnkari	monte	rpaccotacya	0	Q	batch
354.lnkari	monte	rpaccotacya	0	Q	batch
355.lnkari	hist	rpaccotacya	0	Q	batch
356.lnkari	hist	rpaccotacya	0	Q	batch
357.lnkari	hist	rpaccotacya	0	Q	batch
358.lnkari	hist	rpaccotacya	0	Q	batch
359.lnkari	hist	rpaccotacya	0	Q	batch
360.lnkari	hist	rpaccotacya	0	Q	batch
361.lnkari	hist	rpaccotacya	0	Q	batch
362.lnkari	monte	rpaccotacya	0	Q	batch
363.lnkari	hw	rpaccotacya	0	Q	batch
364.lnkari	hist	rpaccotacya	0	Q	batch
365.lnkari	hist	rpaccotacya	0	Q	batch

Figura 7: Cola en el supercomputador

```
rose@Satellite-555-A:~/CS_AlgoritmosParalelos/OPENMPS$ g++ -std=c++11 -fopenmp -o
b count_sort.cpp
rose@Satellite-555-A:~/CS_AlgoritmosParalelos/OPENMPS$ ./b
-----PROGRAMA EN FORMA SERIAL-----
Tiempo del programa serial = 2.0359e-05 segundos
-----PROGRAMA EN FORMA PARALELA-----
Tiempo del programa paralelo = 4.79995e-08 segundos
-----PROGRAMA CON QSORT-----
Tiempo del programa paralelo = 5.99975e-08 segundos
```

Figura 8: Count Sort en mi PC

### 3. MPI

#### 3.1. Regla Trapezoidal

Este programa usa MPI para implementar la versión paralela de la regla trapezoidal, estima la integral de  $a$  a  $b$  de una función  $f(x)$  usando  $n$  trapecios.

1. Cada proceso calcula su intervalo de integración
2. Cada proceso estima la integral de  $f(x)$  sobre su intervalo usando la regla trapezoidal
3. Todos los procesos diferentes de rango 0 mandan su integral al proceso 0.
4. El proceso 0 suma todo lo recibido e imprime el resultado.

#### 3.2. Distribución y lectura de los datos de un vector

Estos programas leen e imprimen un vector entre los procesos usando una distribución de bloques.

```
rose@Satellite-S55-A:~/CS_AlgoritmosParalelos/LAB_3_MPI$ mpicc -std=c99 -o trap
trapezoidal.c
rose@Satellite-S55-A:~/CS_AlgoritmosParalelos/LAB_3_MPI$ mpiexec -n 5 ./trap
Con n = 1024 trapezoides, nuestra estimación
de integrar desde 0.000000 a 3.000000 = 8.894946975633502e+00
rose@Satellite-S55-A:~/CS_AlgoritmosParalelos/LAB_3_MPI$ mpiexec -n 10 ./trap
Con n = 1024 trapezoides, nuestra estimación
de integrar desde 0.000000 a 3.000000 = 8.894946975633502e+00
rose@Satellite-S55-A:~/CS_AlgoritmosParalelos/LAB_3_MPI$ mpiexec -n 100 ./trap
Con n = 1024 trapezoides, nuestra estimación
de integrar desde 0.000000 a 3.000000 = 8.381907362490892e+00
```

### 3.2.1. Scatter

$MPI_{scatter}$  puede ser usada en una función que lee un vector entero en el proceso 0, pero solo manda los componentes necesarios a los otros procesos

### 3.2.2. Gather

Recolecta todos los componentes del vector sobre el proceso 0.

## 3.3. Output

```
Ingresa vector x
1
2
3
4
5
6
7
8
9
10
11
12

El vector x
1.000000 2.000000 3.000000 4.000000 5.000000 6.000000 7.000000 8.000000 9.000000
10.000000 11.000000 12.000000
```

## 4. Multiplicación de matriz con vector

## 5. PThreads