

# Programación III

## Trabajo Práctico N° 1

*“Introducción al Paralelismo”*



**Alumno:** López, Valentín Emmanuel

**Legajo:** LSI001967

**DNI:** 44645057

UNJu - Facultad de Ingeniería

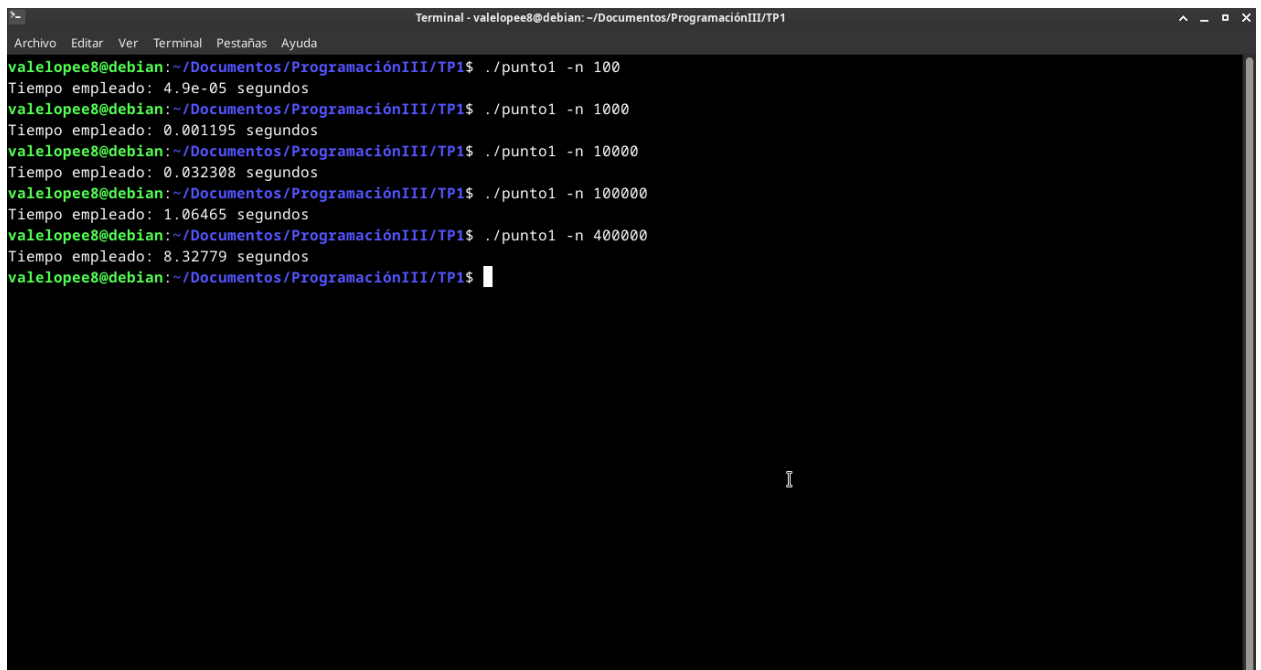
Período Lectivo 2024



1. Implemente en lenguaje C/C++ un programa que permita generar los **n** primeros números primos, donde **n** se enviara como argumento al momento de ejecutar el programa.

**Se pide:**

- Ejecute el programa con los siguientes valores de **n**: 100, 1000, 10000, 100000 y 400000.
- Represente en una tabla comparativa los tiempos de ejecución obtenidos para cada valor de **n**.
- Comente acerca de los resultados obtenidos.



```
Terminal - valelopee8@debian: ~/Documentos/ProgramaciónIII/TP1
valelopee8@debian:~/Documentos/ProgramaciónIII/TP1$ ./punto1 -n 100
Tiempo empleado: 4.9e-05 segundos
valelopee8@debian:~/Documentos/ProgramaciónIII/TP1$ ./punto1 -n 1000
Tiempo empleado: 0.001195 segundos
valelopee8@debian:~/Documentos/ProgramaciónIII/TP1$ ./punto1 -n 10000
Tiempo empleado: 0.032308 segundos
valelopee8@debian:~/Documentos/ProgramaciónIII/TP1$ ./punto1 -n 100000
Tiempo empleado: 1.06465 segundos
valelopee8@debian:~/Documentos/ProgramaciónIII/TP1$ ./punto1 -n 400000
Tiempo empleado: 8.32779 segundos
valelopee8@debian:~/Documentos/ProgramaciónIII/TP1$
```

Valor (n)	n = 100	n = 1000	n = 10000	n = 100000	n = 400000
Tiempo (s)	4.9e-05	0.001195	0.032308	1.06465	8.32779

- Para **n = 100**: Tarea sencilla para el computador.
- Para **n = 1000**: Se toma un poco más de tiempo.
- Para **n = 10000**: Más tiempo que los anteriores.
- Para **n = 100000**: Mucho más tiempo que los anteriores.
- Para **n = 400000**: El mayor tiempo de ejecución.

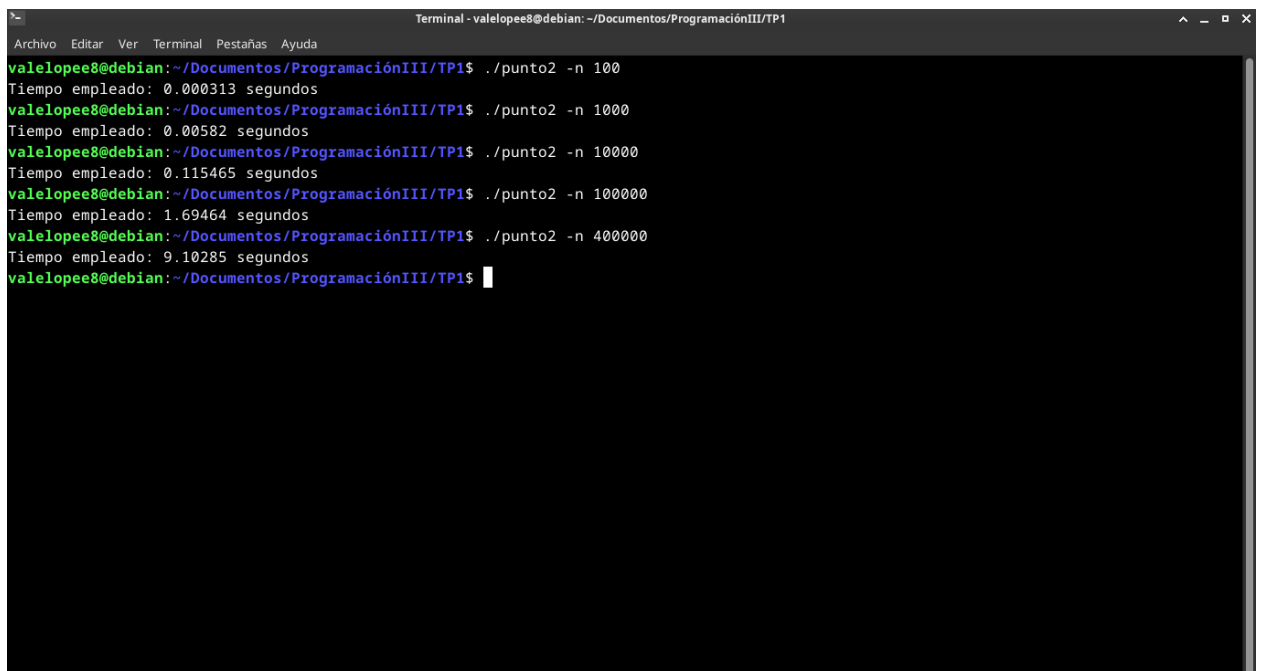
A medida que aumenta el valor de **n**, el cálculo de los primeros **n** números primos se vuelve más costoso computacionalmente. Por esa razón los tiempos de ejecución aumentan cuando queremos calcular una mayor cantidad de números primos.

El programa con estos valores de **n** no representa esfuerzo alguno para el computador. Es el algoritmo que menos recursos consume del trabajo práctico.

2. Implemente en lenguaje C/C++ un programa que permita ordenar un vector de tamaño  $n$ , donde  $n$  se enviara como argumento al momento de ejecutar el programa.

Se pide:

- Ejecute el programa con los siguientes valores de  $n$ : 1000, 10000, 100000 y 400000.
- Represente en una tabla comparativa los tiempos de ejecución obtenidos para cada valor de  $n$ .
- Comente acerca de los resultados obtenidos.



```
Terminal - valelopee8@debian: ~/Documentos/ProgramaciónIII/TP1
valelopee8@debian:~/Documentos/ProgramaciónIII/TP1$ ./punto2 -n 100
Tiempo empleado: 0.000313 segundos
valelopee8@debian:~/Documentos/ProgramaciónIII/TP1$ ./punto2 -n 1000
Tiempo empleado: 0.00582 segundos
valelopee8@debian:~/Documentos/ProgramaciónIII/TP1$ ./punto2 -n 10000
Tiempo empleado: 0.115465 segundos
valelopee8@debian:~/Documentos/ProgramaciónIII/TP1$ ./punto2 -n 100000
Tiempo empleado: 1.69464 segundos
valelopee8@debian:~/Documentos/ProgramaciónIII/TP1$ ./punto2 -n 400000
Tiempo empleado: 9.10285 segundos
valelopee8@debian:~/Documentos/ProgramaciónIII/TP1$
```

Valor (n)	n = 100	n = 1000	n = 10000	n = 100000	n = 400000
Tiempo (s)	0.000313	0.00582	0.115465	1.69464	9.10285

- Para  $n = 100$ : Tiempo corto.
- Para  $n = 1000$ : Un poco más de tiempo que el anterior.
- Para  $n = 10000$ : Más tiempo que el anterior.
- Para  $n = 100000$ : Aumenta el tiempo de ejecución y el consumo de recursos del computador.
- Para  $n = 400000$ : El mayor tiempo de ejecución y consumo de recursos.

El algoritmo de este problema es un poco más complejo que el anterior, sus tiempos de ejecución lo justifican (aunque el tiempo no varía drásticamente). Este programa consume más recursos pero sigue siendo relativamente sencillo para el computador ejecutarlo con los valores de  $n$  empleados con anterioridad.

Haste este punto no notamos un esfuerzo del computador para ejecutar los procesos planteados en el trabajo, solo un leve aumento de tiempo de ejecución.

Veamos que ocurre con el siguiente.

### 3. Descargue del aula virtual el proyecto **motionBrown.tar.gz**:

**Se pide:**

- Descomprima el proyecto en su espacio de trabajo.
- Compile el proyecto y genere el ejecutable/binario correspondiente
- Ejecute el programa con los siguientes valores de n: 50, 100, 500, 1000, 1500 y 2000.
- Represente en una tabla comparativa los tiempos de ejecución obtenidos para cada valor de n.
- Comente acerca de los resultados obtenidos.

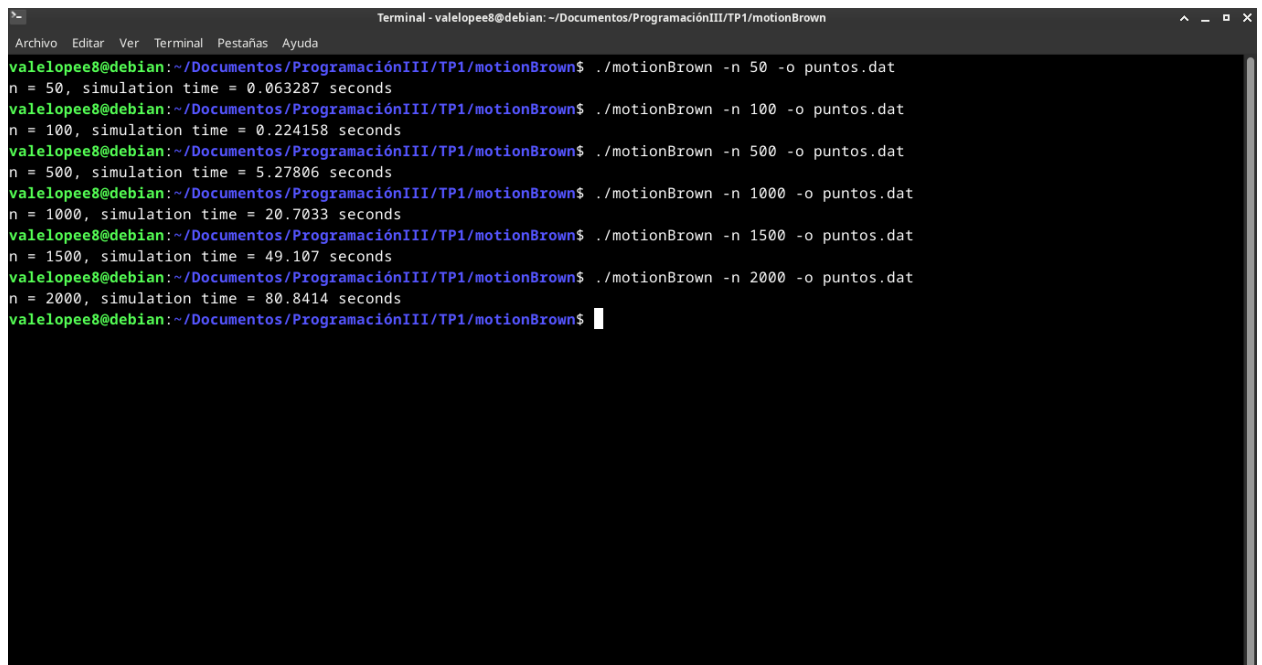
**Nota:**

El programa implementado permite simular un conjunto de partículas en 2D. Básicamente describe, en un archivo de texto, el movimiento aleatorio de una partícula como resultado de las colisiones de dicha partícula con otras partículas en movimiento.

**Ref:**

[https://es.wikipedia.org/wiki/Movimiento\\_browniano](https://es.wikipedia.org/wiki/Movimiento_browniano)

<https://www.turtle.ox.ac.uk/run>



```
Terminal - valelopee8@debian: ~/Documentos/ProgramaciónIII/TP1/motionBrown
valelopee8@debian:~/Documentos/ProgramaciónIII/TP1/motionBrown$ ./motionBrown -n 50 -o puntos.dat
n = 50, simulation time = 0.063287 seconds
valelopee8@debian:~/Documentos/ProgramaciónIII/TP1/motionBrown$ ./motionBrown -n 100 -o puntos.dat
n = 100, simulation time = 0.224158 seconds
valelopee8@debian:~/Documentos/ProgramaciónIII/TP1/motionBrown$ ./motionBrown -n 500 -o puntos.dat
n = 500, simulation time = 5.27806 seconds
valelopee8@debian:~/Documentos/ProgramaciónIII/TP1/motionBrown$ ./motionBrown -n 1000 -o puntos.dat
n = 1000, simulation time = 20.7033 seconds
valelopee8@debian:~/Documentos/ProgramaciónIII/TP1/motionBrown$ ./motionBrown -n 1500 -o puntos.dat
n = 1500, simulation time = 49.107 seconds
valelopee8@debian:~/Documentos/ProgramaciónIII/TP1/motionBrown$ ./motionBrown -n 2000 -o puntos.dat
n = 2000, simulation time = 80.8414 seconds
valelopee8@debian:~/Documentos/ProgramaciónIII/TP1/motionBrown$
```

Valor (n)	n = 50	n = 100	n = 500	n = 1000	n = 1500	n = 2000
Tiempo (s)	0.063287	0.224158	5.27806	20.7033	49.107	80.8414

- Para n = 50: Tiempo corto.

- Para  $n = 100$ : Un poco más de tiempo que el anterior.
- Para  $n = 500$ : Mayor tiempo que los anteriores.
- Para  $n = 1000$ : Incremento considerable y notorio de tiempo de ejecución.
- Para  $n = 1500$ : Crece el tiempo y el consumo de recursos.
- Para  $n = 2000$ : El tiempo de ejecución supera el minuto por primera vez.

Este simulador de partículas lleva consigo un algoritmo matemático mucho más complejo que los programas anteriores y se refleja en el tiempo de ejecución.

Cuando  $n = 1500$  y  $n = 2000$  el computador mostró un consumo de recursos mayor, el más notorio hasta el momento. El procesador aumento su temperatura y los ventiladores trabajaron más que los casos anteriores.

Con el comando: **cat puntos.dat | wc -l** podemos observar la cantidad de entradas/coordenas que produjo el algoritmo, siendo para  $n = 500$  un total de 5001 y para  $n = 2000$  un total de 200001.

También con el comando: **ls -lSh** podemos ordenar los archivos presente de la carpeta y ver su peso, en este caso nos interesa la salida **puntos.dat**. Cuando  $n = 500$  dicho binario pesa 18KiB y cuando  $n = 2000$  pesa unos 3.4Mib.

Podemo concluir que al aumentar  $n$  mayor será la cantidad de entradas/coordenadas y el peso del binario. Por lo tanto, es evidente que el computador se esfuerza más para poder ejecutar el proyecto motionBrown.

4. Implemente en lenguaje C/C++ un programa que permita aproximar la solución de un sistema de ecuaciones lineales. El programa deberá emplear el método numérico de **Jacobi** y al momento de ejecutarse recibirá como argumento el nombre del archivo existente con los datos de la matriz  **$Ax=b$** .

**Se pide:**

- Ejecute el programa con sistemas de: 5x5, 10x10, 20x20, 60x60, 100x100
- Represente en una tabla comparativa los tiempos de ejecución obtenidos para cada sistema.
- Comente acerca de los resultados obtenidos.

**Ref:**

<https://ocw.ehu.eus/file.php/9/pdf/temas/Tema3.pdf>

[https://www.ucg.ac.me/skladiste/blog\\_10701/objava\\_23569/fajlovi/jacobi.pdf](https://www.ucg.ac.me/skladiste/blog_10701/objava_23569/fajlovi/jacobi.pdf)

Valor (n)	$n = 100$	$n = 1000$	$n = 10000$	$n = 100000$	$n = 400000$
Tiempo (s)					

5. ¿Para cada caso el programa implementado aprovecha los procesadores disponibles en su computador? Justifique.

Se podrían ejecutar con mejor rendimiento si se aprovechan todos los recursos del computador. Es decir, pasar de una ejecución secuencial (un procesador) a uno paralelo (más de uno). De esta forma el tiempo de procesamiento se reduce considerablemente.

Cabe recalcar que no todos los programas se pueden paralelizar, se puede lograr en ciertos aspectos del algoritmo, tal vez un 25% o 20%.

Observemos por medio de **htop** el consumo de los procesadores durante la ejecución del proyecto motionBrown:

```
Terminal - valelopee8@debian: ~/Documentos/ProgramaciónIII/TP1/motionBrown
valelopee8@debian:~/Documentos/ProgramaciónIII/TP1/motionBrown$ ./motionBrown -n 2000 -o puntos.dat

Terminal - valelopee8@debian: -
0[|||||] 6.3% Tasks: 117, 360 thr, 98 kthr; 2 running
1[|||||] 9.4% Load average: 1.45 0.94 0.85
2[|||||] 100.0% Uptime: 05:11:13
3[|||||] 8.8%
Mem[|||||] 3.09G/11.6G
Swp[|||||] 0K/1.86G

Main 1/20
PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command
13424 valelopee8 20 0 3352 1172 1072 R 99.4 0.0 1:06.28 ./motionBrown -n 2000 -o puntos.dat
F1Help F2Setup F3Search F4Filter F5Tree F6SortBy F7Nice -F8Nice +F9Kill F10Quit
```

De los cuatro cores existentes solo uno de ellos alcanza el 100% de su capacidad, mientras que los demás están pendientes de otras tareas del sistema. Por lo tanto notamos que no se está aprovechando el máximo del computador para ejecutar el programa y reducir el tiempo de espera.

**En conclusión**, para aprovechar al máximo los procesadores disponibles, es necesario implementar técnicas de paralelización adecuadas en el código. Esto puede incluir el uso de hilos, procesos múltiples o el uso de bibliotecas de paralelización como OpenMP o MPI, dependiendo de la naturaleza del problema y las capacidades del hardware disponible.