Universidad Nacional de Córdoba

Cátedra de Sistemas Operativos II

Trabajo Práctico N° II

Integrante: Choquevilca Gustavo
(g.choquevilca@gmail.com)

13 de Mayo del 2018

Índice

troducción

OpenMP[1] es una API para la programación multiproceso que permite añadir concurrencia y paralelismo a programas escritos en lenguajes C, C++ y Fortran. Se compone de un conjunto de directivas de compilador, rutinas y variables de entorno que modifican el comportamiento en tiempo de ejecución. Es un modelo de programación portable y escalable que proporciona a los programadores una interfaz simple y flexible para el desarrollo de aplicaciones paralelas. OpenMP usa un modelo de paralelismo fork/join, soportando además paralelismo anidado. Permite la programación multihilo en sistemas con memoria compartida. Una gran ventaja de OpenMP es que viene integrado en el compilador GCC, y por lo tanto no es necesario instalar ningún programa o aplicación en el sistema. Otra de las ventaja es que permite definir bloques paralelizables mediante etiquetas “#pragma”, por lo que dentro de un mismo programa, puede haber rutinas que se ejecuten de manera paralela y otras que se realicen de manera serial.
En el proyecto realizado, se usan algunas instrucciones que permiten paralelizar ciertos bloques de código, para aumentar la eficiencia y rendimiento de la ejecución.
Este documento está formado por la Especificación de Requisitos Software (ERS), el diseño de solución y la implementación y resultados para el Proyecto Comunicación de procesos en computación paralela(CPCP). Esta especificación se ha estructurado inspirándose en las directrices dadas por el estándar “IEEE Recommended Practice for Software Requirements Specification ANSI/IEEE 830 1998”

Propósito

Este documento tiene como propósito conocer y especificar el funcionamiento general del proyecto CPCP. Planteando desde los requerimientos de sistemas hasta la implementación y Resultados. Este documento está dirigido a los usuarios que deseen utilizar el software para la comunicación entre procesos y para aquellas personas que necesiten definir nuevos requerimientos.

Ámbito del Sistema

Este proyecto será utilizado cuando dado una cierta cantidad datos de pulsos se desee calcular un promedio de gate por pulso y calcular la autocorrelación sobre todos los valores de los pulsos para un determinado gate. Además si deseamos conocer la diferencia entre un programa que utiliza paralelismo y otro que no utiliza paralelismos pudiendo así observar la performance del sistema.

Definiciones, Acrónimos y Abreviaturas

- CPCP: Comunicación de procesos en computación paralela
- IEEE : Instituto de Ingeniería Eléctrica y Electrónica
- gate: componentes de rango (I +jQ)

Referencias

[1] OpenMP, https://es.wikipedia.org/wiki/OpenMP, [Mayo 2018]
[2]Gprof, http://www.chuidiang.org/clinux/herramientas/profiler.php, [Mayo 2018]

Descripción General del Documento

El presente Documento describe cómo se implementa la comunicación entre procesos en computación paralela, además se realizará una breve descripción del problema, se expondrán los requerimientos utilizados, se mostrará el diseño y la documentación del proyecto, para finalmente exponer los resultados que se obtienen.

Descripción General

Se presenta una descripción de los requisitos del sistema, con el fin de conocer las principales funciones que debe realizar, perspectiva del producto, funciones del producto, características del usuario , supuestos y dependencias que afectan al desarrollo.

Perspectiva del Producto

Se proyecta implementar un sistema de comunicación entre procesos en computación paralela Cabe aclarar que el producto resultante es independiente de otros productos,

Funciones del Producto

Las funciones que conforman el sistema son los siguientes:

- Lectura de datos de un archivo binario.
- Sistema de cálculo de media aritmética de n muestras que caen en el rango de resolución.
- Sistema de matrices para el canal vertical y horizontal
- Sistema de Autocorrelación para el canal vertical y canal horizontal
- Sistema de almacenamiento en forma de archivo de datos para la autocorrelación

Características de los Usuarios

Tener conocimiento medio o superior de manejo SO Unix, es decir ya sea para poder ejecutar y compilar los programas hasta reconocer los resultados que obtendrá por pantalla el cliente.

Restricciones

- En cuanto a las restricciones de software se exige que el proyecto CPCP funciones bajo SO tipo Unix.
- En cuanto a las restricciones de hardware: disponer de computadoras con memoria RAM superior a 1 GB, procesador mínimo dos cores para poder aprovechar el paralelismo.

Suposiciones y Dependencias

Utilizar SO Unix, en SO Windows u otros no funciona el proyecto ya que existen secciones de código que implementan paralelismo OpenMP. Así por ejemplo aplicar el proyecto en un SO Windows implicaría agregar requerimientos.

Requisitos Futuros

Una futura mejora es que el proyecto pueda funcionar en cualquier plataforma de SO

Requisitos Específicos

En este apartado se presentan los requisitos funcionales que deberán ser satisfechos por el sistema. Todos los requisitos aquí expuestos son esenciales, es decir, no sería aceptable un sistema que no satisfaga alguno de los requisitos aquí presentados.

Interfaces Externas

- **Interfaz de usuario:** es el terminal del SO, en donde se mostrará todo la información necesaria para el usuario.
- **Interfaz de Hardware:** es la pantalla del monitor ya que por ella se mostraran los resultados obtenidos. Otra interfaz es el mouse y teclado por donde se ingresara los comando a ejecutar.

Requisitos funcionales

- **RF1** El sistema debe poder leer de un archivo los datos referidos a los pulsos
- **RF2** El sistema debe poder calcular el promedio aritmético de n muestras.
- **RF3** El sistema debe poder generar un matriz (gate,pulso) para el canal vertical y horizontal respectivamente
- **RF4** El sistema debe poder calcular la autocorrelación para la matriz vertical y horizontal
- **RF5** El sistema debe guardar los cálculos de la autocorrelación en un archivo binario.
- **RF6** EL sistema debe proveer una solución sin paralelismo (procedural) y con paralelismo (librerías OpenMP)

Requisitos no funcionales

- **RNF1** La aplicación debe ser desarrollada en C
- **RNF2** La aplicación debe ponerse en funcionamiento en un cluster de la facultad y en una computadora personal del usuario.
- **RNF3** disponer de computadoras con memoria RAM superior a 1 GB, procesador mínimo dos cores.
- **RNF4** El sistema deberá ser sencillo e intuitivo.

Diseño de solución

Para el realizar la solución del proyecto primero se implementó el modelo procedural es decir sin la utilización de paralelismo, luego se analizó el programa y se definió qué regiones se realizarán en paralelo para implementar el modelo con OpenMP.

Diagramas UML

A continuación se muestra los diagramas UML necesarios para la explicación de los requerimientos de software. Solo se realizaron los diagramas de caso de uso, actividad y secuencia.

Diagrama de caso de uso

Una vez conocidos los requerimientos del sistema se construye un modelo de requerimientos utilizando casos de uso. El sistema cuenta con un actor y 7 caso de usos.

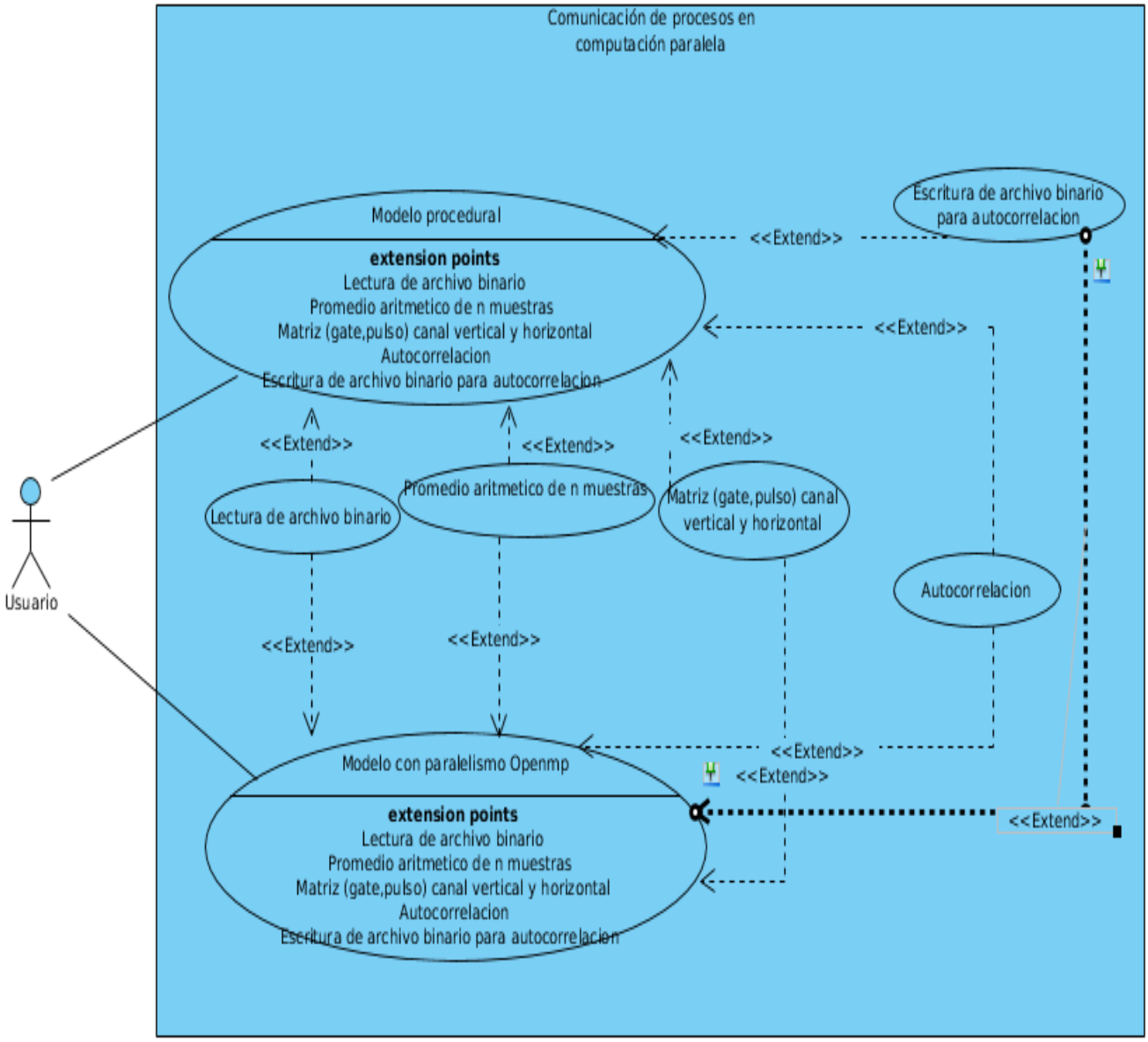
Comunicación de procesos en computación paralela
Modelo procedural
extension points
Lectura de archivo binario
Promedio aritmetico de n muestras
Matriz (gate,pulso) canal vertical y horizontal
Autocorrelacion
Escritura de archivo binario para autocorrelacion
Escritura de archivo binario para autocorrelacion
<<Extend>>
<<Extend>>
<<Extend>>
<<Extend>>
<<Extend>>
Lectura de archivo binario
Promedio aritmetico de n muestras
Matriz (gate,pulso) canal vertical y horizontal
Autocorrelacion
Usuario
<<Extend>>
<<Extend>>
<<Extend>>
<<Extend>>
Modelo con paralelismo Openmp
extension points
Lectura de archivo binario
Promedio aritmetico de n muestras
Matriz (gate,pulso) canal vertical y horizontal
Autocorrelacion
Escritura de archivo binario para autocorrelacion
<<Extend>>

figura.1

Diagrama de Actividades

Otro diagrama que muestra el comportamiento del sistema es el diagrama de actividad.

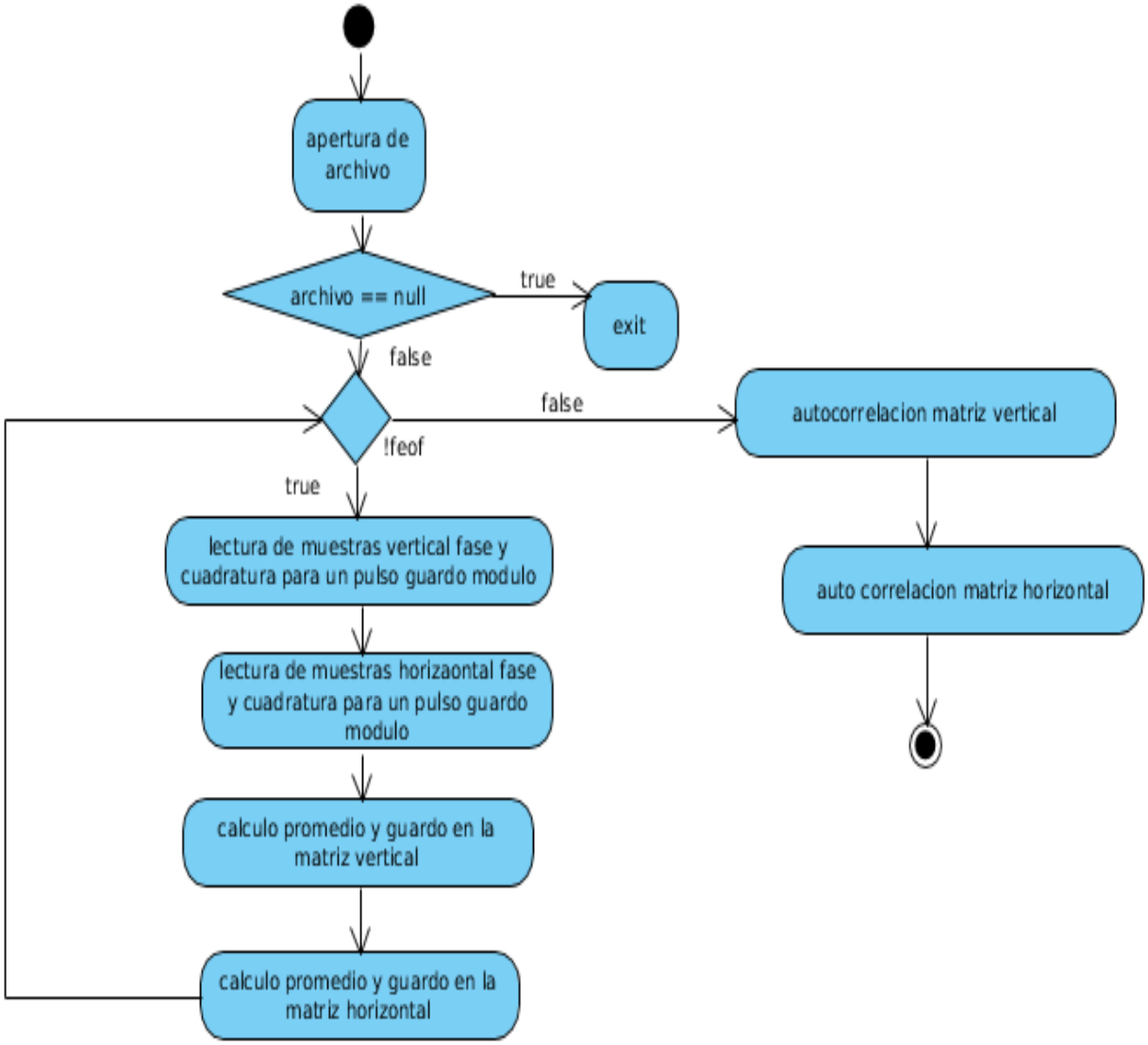

figura.2

Diagrama de Secuencia

Por último se muestra el diagrama de secuencia que implementa el proyecto.

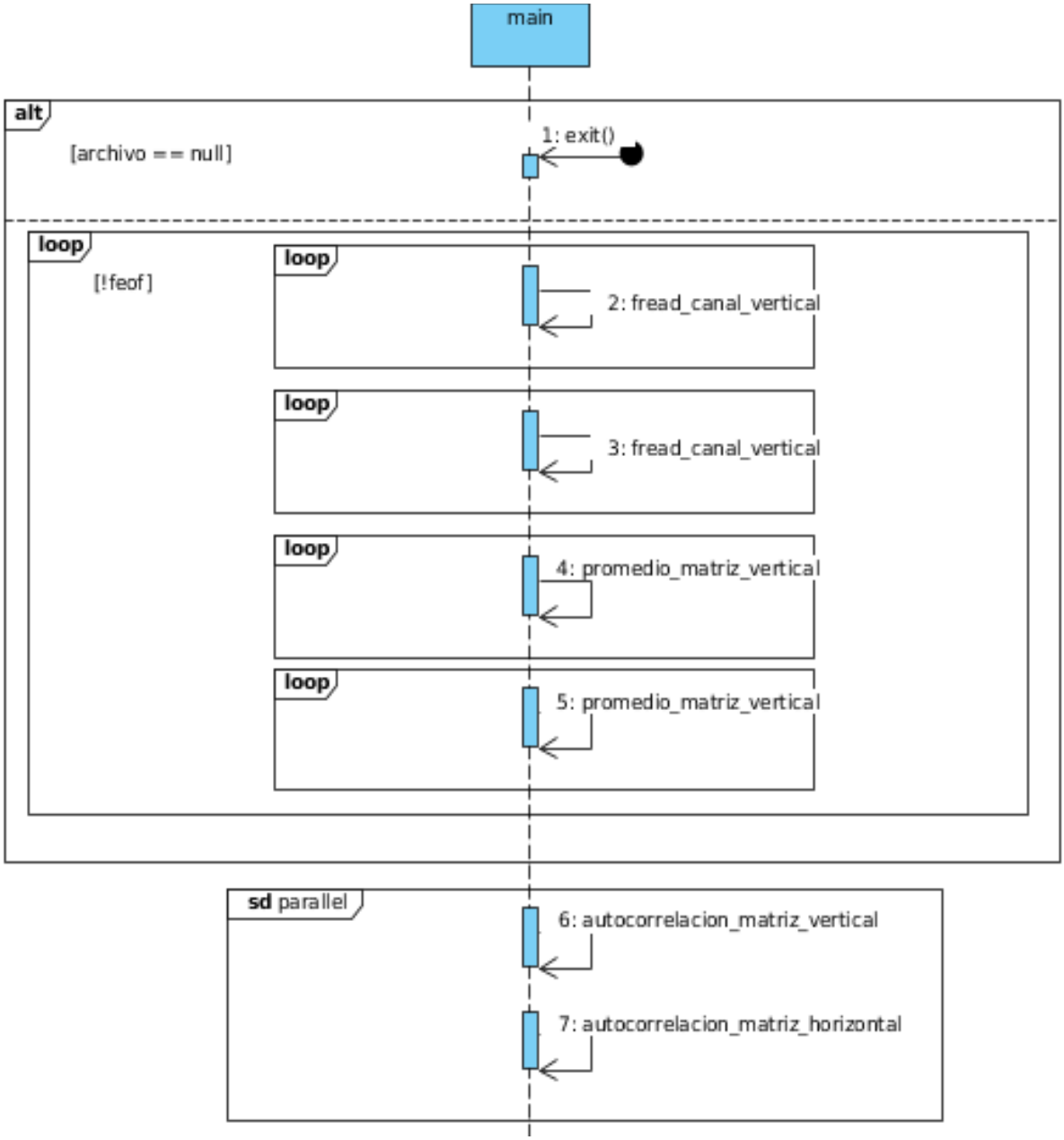

figura.3

Podemos observar en el diagrama de secuencia que existe una sección paralela y esta se realiza para calcular la autocorrelacion.

Implementación y Resultados

La implementación y resultado se puede observar en las siguientes imágenes.

En figura 4 se observa cómo está formado el archivo binario para la autocorrelacion.

La estructura del los archivos binario de la autocorrelacion para el canal vertical y el canal horizontal se compone de la siguiente manera: al disponer de 500 gates existirá 500 valores disponibles de autocorrelacion.

canal vertical[1]
dato autocorrelacion para el gate 1 recorriendo los 800 pulsos
canal vertical[2]
dato autocorrelacion para el gate 2 recorriendo los 800 pulsos
canal vertical[...]
dato autocorrelacion para el gate ... recorriendo los 800 pulsos
canal vertical[500]
dato autocorrelacion para el gate 500 recorriendo los 800 pulsos

figura.4 Estructura autocorrelacion canal vertical

canal horizontal[1]
dato autocorrelación para el gate 1 recorriendo los 800 pulsos
canal horizontal[2]
dato autocorrelación para el gate 2 recorriendo los 800 pulsos

canal horizontal[...]
dato autocorrelación para el gate ... recorriendo los 800 pulsos
canal horizontal[500]
dato autocorrelacion para el gate 500 recorriendo los 800 pulsos

figura.5 Estructura autocorrelacion canal horizontal

En la figura 6. se puede observar los valor obtenidos de autocorrelacion

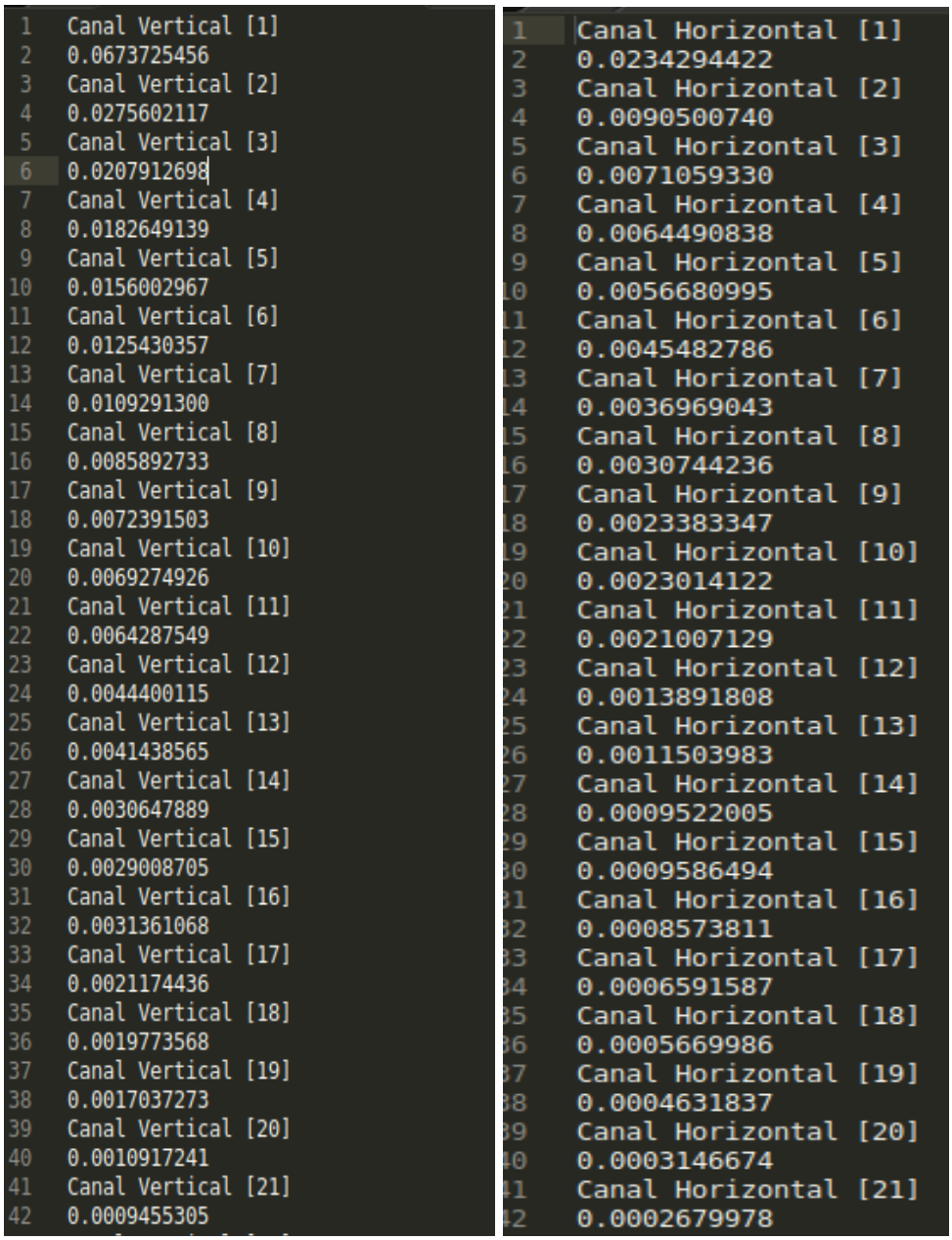

figura.6

Para realizar la implementación utilizando paralelismo se optó por de definir como región paralela el cálculo de la autocorrelacion. En la pc local se obtuvieron los siguientes resultados que corresponde a la ejecución en un procesador icore 3 con 4 cores.

EJECUCIÓN	HILOS		
	4	8	16
1	0,724335	0,929770	0,798550
2	7,22E-01	0,754719	0,892524
3	0,721637	0,773455	0,998485
4	0,733046	0,752243	1,069492
5	0,767119	0,784766	0,904891
6	0,759024	0,755219	0,905751
7	0,730125	0,749367	0,922075
8	0,730004	0,791182	0,855981
9	0,803626	0,759037	0,833223
10	0,724516	0,749728	0,933500
11	0,746539	0,757757	0,892809
12	0,730137	0,753986	0,888028
13	0,818281	0,756470	0,864230
14	0,858083	0,787968	0,836410
15	0,742891	0,786509	1,030516
16	0,724512	0,757083	0,976821
17	0,724721	0,750198	0,966348
18	0,732795	0,759078	0,871711
19	0,754386	0,756754	0,888195
20	0,729722	0,754967	0,864466
21	0,726381	0,762271	0,925766
22	0,726220	0,754427	1,060587
23	0,746631	0,781982	0,965043
24	0,837158	0,763172	0,898457

25	0,734977	0,869417	0,991906
26	0,711537	0,760941	1,031348
27	0,713834	0,758469	0,805955
28	0,760593	0,768759	0,794141
29	0,740587	0,770234	0,786062
30	0,722264	0,754553	0,781811
promedio	**0,746597 seg**	0,772149 seg	0,907836 seg

figura.7 Tabla de tiempos de programa en paralelo en pc local

Se puede observar que la mayor performance se presenta cuando el programa se ejecuta con 4 hilos obteniendo un promedio de **0,746597 segundos.**
En la ejecución procedural en la pc local se obtienen los siguientes tiempos:

EJECUCIÓN	TIEMPOS
1	0,939792
2	7,24E-01
3	0,750071
4	0,718349
5	0,720233
6	0,725182
7	0,733707
8	0,752443
9	0,721395
10	0,719468
11	0,729238
12	0,963124
13	0,766071
14	0,762405
15	0,730129

16	0,793866
17	0,752841
18	0,767152
19	0,760769
20	0,783244
21	0,757792
22	0,768287
23	0,835709
24	1,103605
25	0,848590
26	0,888308
27	0,75868
28	0,759041
29	0,767163
30	0,748736
promedio	**0,784988** seg

figura.8 Tabla de tiempos de programa procedural de pc local

Se puede observar que la performance disminuye con respecto a la ejecución paralela y esto lo podemos ver en el promedio de tiempo.

Otra forma de verificar la performance es a través de gráfico hilos vs tiempo.
En donde podemos apreciar que a medida que incrementa la cantidad de los hilos la performance disminuye y esto lo notamos con el aumento del tiempo de ejecución.

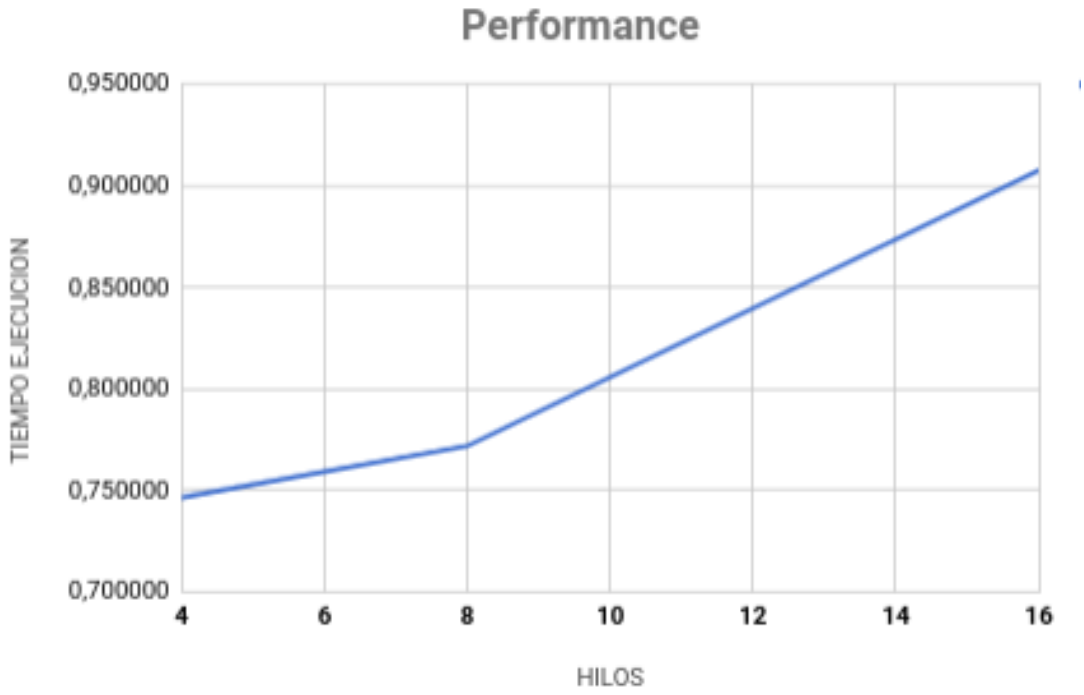

figura.9 Gráfico de performance tiempo vs hilos PC local

Para realizar la parte de ejecución y compilación en el cluster se utilizó : rocky que contiene 8 cores.

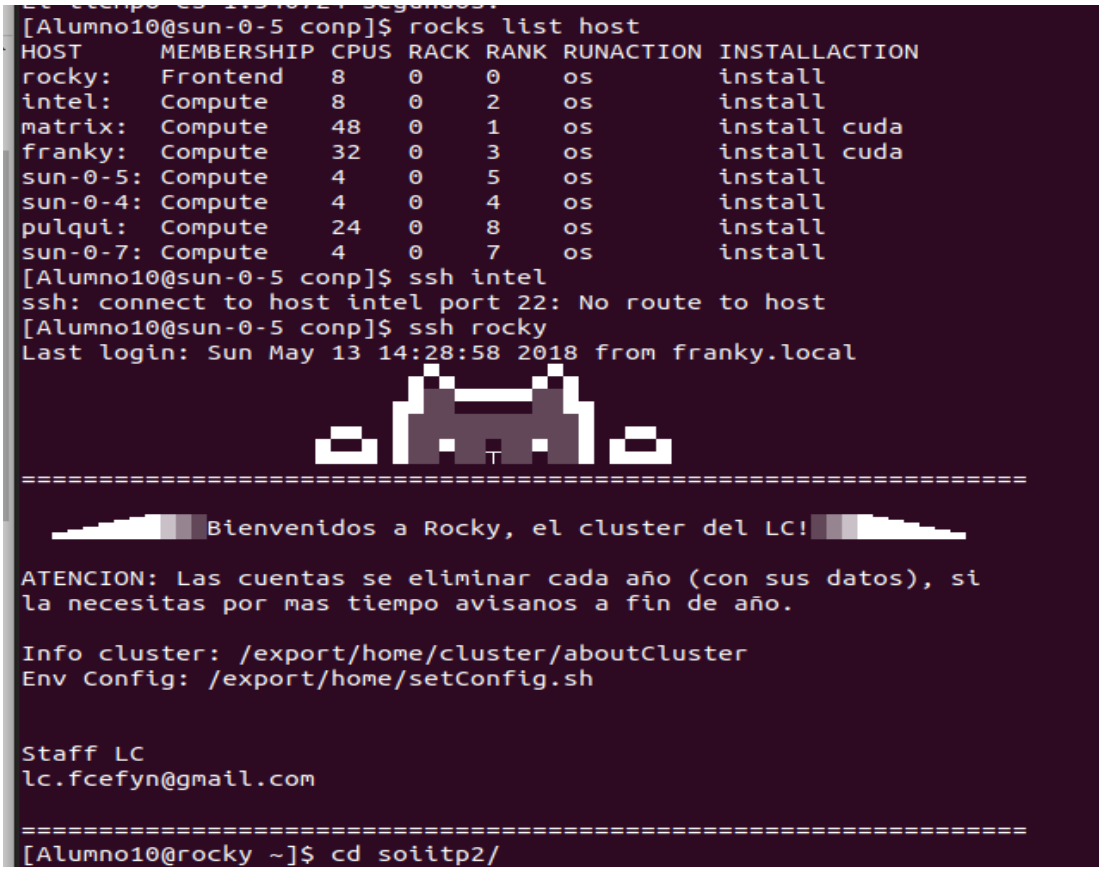

figura.10 interfaz de cluster

Los tiempos obtenidos para la aplicación que implementa paralelismo en este cluster fueron:

		HILOS	
EJECUCIÓN	8	16	32
1	0,813282	0,834549	0,869935
2	8,18E-01	0,849283	0,869497
3	0,818624	0,842281	0,877305
4	0,824421	0,839363	0,874258
5	0,817985	0,845724	0,871289
6	0,810152	0,840558	0,845313

7	0,822485	0,840134	0,864788
8	0,81029	0,846461	0,875757
9	0,812888	0,835551	0,866465
10	0,818884	0,841901	0,874260
11	0,816653	0,848200	0,875724
12	0,826751	0,874895	0,868453
13	0,866492	0,843914	0,866091
14	0,911592	0,847425	0,873027
15	0,812395	0,842832	0,982487
16	0,818873	0,876633	0,905548
17	0,821225	0,842772	0,869449
18	0,820073	0,852149	0,866389
19	0,805671	0,851740	0,867873
20	0,822172	0,838321	0,861655
21	0,817575	0,848803	0,964038
22	0,826517	0,838298	0,906502
23	0,811408	0,836512	0,868171
24	0,817015	0,857041	0,871044
25	0,816586	0,828274	0,889321
26	0,815962	0,832939	0,86836
27	0,818423	0,836325	0,872343
28	0,828164	0,838848	0,875958
29	0,820486	0,841157	0,881146
30	0,810538	0,83721	0,868524
promedio	**0,822371 seg**	0,844336 seg	0,879699 seg

figura.11 Tabla de tiempos de programa en paralelo en cluster

Se puede observar que la mayor performance se presenta cuando el programa se ejecuta con 8 hilos obteniendo un promedio de **0,822371 segundos.**
En la ejecución procedural en el cluster se obtienen los siguientes tiempos:

EJECUCIÓN	**TIEMPOS**
1	0,8064620
2	8,084E-01
3	0,797703
4	0,905309
5	0,798937
6	0,801339
7	0,801528
8	0,851796
9	0,801398
10	0,910011
11	0,82504
12	0,894426
13	0,805701
14	0,801638
15	0,807976
16	0,805237
17	0,805511
18	0,828719
19	0,803259
20	0,850719
21	0,807127
22	0,927968
23	0,802873
24	0,805236

25	0,918981
26	0,804348
27	0,838115
28	0,808993
29	0,835171
30	0,804623
promedio	**0,8288176 seg**

figura.12 Tabla de tiempos de programa procedural en cluster

Se puede observar que la performance de ambos programas ejecutados en el cluster es más bien similar quizás un poco mejor la solución que implementa paralelismo.

Otra forma de verificar la performance es a través de gráfico hilos vs tiempo.
En donde podemos apreciar que a medida que incrementa la cantidad de los hilos la performance disminuye y esto lo notamos con el aumento del tiempo de ejecución.

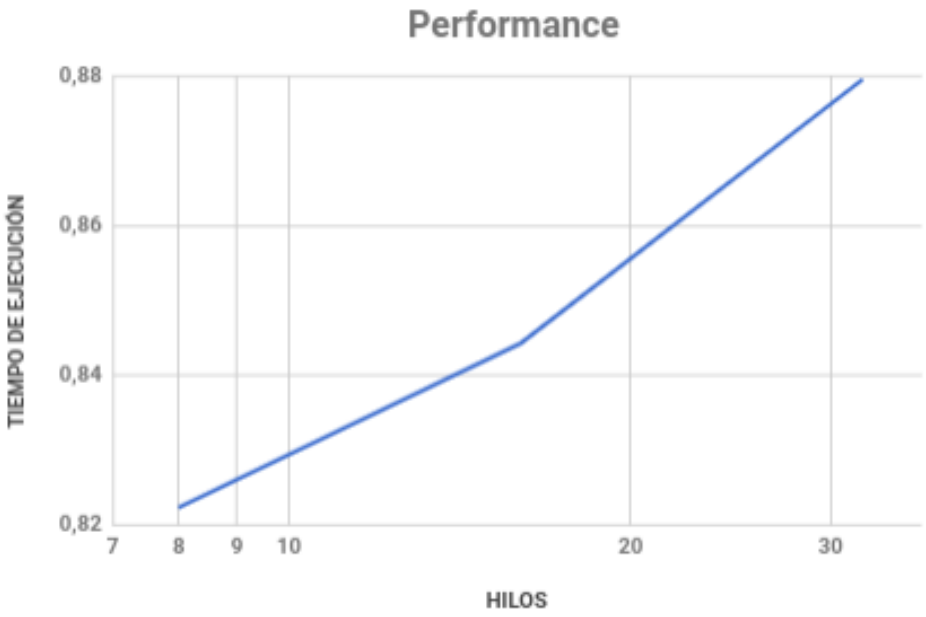

figura.13 Gráfico de performance tiempo vs hilos Cluster

Para realizar las ejecuciones de los programa se utilizo un script para hacer más fácil y más automático la obtención de los tiempo de ejecución.

Profiling

Las pruebas de perfilado (profile testing) se centran precisamente en analizar mediante una herramienta apropiada conocida como profiling, el comportamiento de una aplicación en tiempo de ejecución. La información que se rastrea durante una prueba de perfilado es muy variada, pero suele incluir lo siguiente campos:

- Secuencia, duración y frecuencia de las llamadas a uno o varios métodos dentro del sistema.
- Utilización del CPU en un intervalo
- Utilización de memoria

Se disponen de las siguientes herramientas para realizar las pruebas de profiling:

- Allinea MAP, es un generador de perfiles de aplicaciones uno de los primeros perfiladores capaces de analizar y mostrar visualmente el rendimiento cuando se ejecuta a gran escala (incluidos muchos miles de núcleos).
- AQtime es un perfilador de rendimiento y conjunto de herramientas de depuración de memoria y recursos.
- Gprof[2] es una herramienta de análisis de rendimiento para aplicaciones Unix. Utiliza un híbrido de instrumentación y muestreo y se creó como una versión extendida de la herramienta "prof" anterior. A diferencia de prof, gprof es capaz de recopilar e imprimir gráficos de llamadas limitadas.
- OProfile es una herramienta de generación de perfiles estadísticos para todo el sistema para Linux. consiste en un módulo kernel, un daemon de espacio de usuario y varias herramientas de espacio de usuario. Puede perfilar un sistema completo o sus partes, desde rutinas de interrupción o controladores, hasta procesos de espacio de usuario. Tiene poca sobrecarga.
- Amplifier intel Vtune: obtiene datos precisos poca sobrecarga, obtiene datos de CPU, GPU, FPU, subprocesos memoria y mucho más. Se caracteriza por tener respuestas rápidas. Con el amplificador Intel® VTune ™, obtiene todas estas capacidades avanzadas de creación de perfiles con una única interfaz de análisis amigable. Y para aplicaciones multimedia, también obtiene herramientas poderosas para ajustar las aplicaciones OpenCL ™ y la GPU.

Para este práctico se utilizó la herramienta gprof por su sencillez y flexibilidad con UNIX.

Para compilar con gprof se debe implementar la bandera -pg es decir se debe seguir los siguientes pasos:

- se compila el programa de la siguiente manera:
  gcc -o tp2p tp2p.c -lm -fopenmp -pg -no-pie
- se ejecuta el programa: ***./tp2p***

- luego se muestra los resultados: ***grpof tp2p***

```
ggg@ggg:~/Documentos/SISTEMAS_OPERATIVOS_II/TRABAJOS_PRACTICOS/TP2/TP2V6/CON PAR
ALELISMO$ gcc -o tp2p tp2p.c -lm -fopenmp -pg -no-pie
ggg@ggg:~/Documentos/SISTEMAS_OPERATIVOS_II/TRABAJOS_PRACTICOS/TP2/TP2V6/CON PAR
ALELISMO$ ./tp2p
El tiempo es 0.725663 segundos.
ggg@ggg:~/Documentos/SISTEMAS_OPERATIVOS_II/TRABAJOS_PRACTICOS/TP2/TP2V6/CON PAR
ALELISMO$ gprof tp2p
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 88.39      0.15     0.15                             main
 11.78      0.17     0.02        1    20.03    20.03  correlacion_h
  0.00      0.17     0.00        1     0.00     0.00  correlacion_v
```

figura.10 Resultado de gprof

- % time: el porcentaje del tiempo de ejecución total del programa utilizando por esa función.
- Cumulative seconds: una suma continua de la cantidad de segundos contabilizados por esta función y los enumerados anteriormente.
- Self seconds: la cantidad de segundo representados sólo por esta función, si esta función está perfilada.
- Calls: el número de veces que se invocó esta función, si esta función está perfilada, sino en blanco
- Self ms/call: la cantidad promedio de milisegundos gastados en esta función por llamada, si esta función está perfilada , sino en blanco.
- Total ms/call: el número promedio de milisegundos gastados en esta función y sus descendientes por llamada, si esta función está perfilada, en blanco
- Name : nombre de la función

Conclusión

En general, podemos concluir que es posible realizar aplicaciones que nos permiten la comunicación entre procesos utilizando un sistema procedural y un sistema que implemente paralelismo.

Se comprendió el funcionamiento de la herramienta OpenMp, la cual es de gran utilidad a la hora de generar sistemas que utilicen paralelismo. También podemos concluir que el máximo paralelismo se obtendrá cuando cada core de procesador ejecute un solo hilo, para realizar este proyecto se utilizó un procesador icore3 lo que representa como máximo 4 cores, y la performance para este caso es máxima, esto lo podemos ver en la figura 9. Se pudo utilizar la herramienta de Profiling gprof que proporciona los tiempo porcentaje y llamadas de ejecución de programa, estos datos se pudieron leer sin problemas.

Se puede mencionar que uno de los inconveniente a la hora de realizar el proyecto fue cuando se agregó el paralelismo a la aplicación ya que en un principio los valores obtenidos de autocorrelacion no coinciden con los del modelo procedural, analizando y modificando el programa que implementa paralelismo se pudo corregir este error.