



UNIVERSIDAD
NACIONAL DE
CÓRDOBA

Facultad de Ciencias Exactas, Físicas y Naturales

Cátedra de Sistemas Operativos II

TRABAJO PRÁCTICO N° II

Salas Canalicchio, Sergio Enrique

Córdoba, 13 de Mayo de 2018

Índice

1. Introducción

- 1.1. Propósito
- 1.2. Ámbito del sistema
- 1.3. Definiciones, acrónimos y abreviaturas
- 1.4. Referencias
- 1.5. Descripción General del documentado

2. Descripción General

- 2.1. Perspectiva del producto
- 2.2. Funciones del producto
- 2.3. Características de los usuarios
- 2.4. Restricciones
- 2.5. Suposiciones y dependencias
- 2.6. Requisitos futuros

3. Requisitos específicos

- 3.1. Interfaces externas
- 3.2. Funciones
- 3.3. Requisitos de rendimiento
- 3.4. Restricciones de diseño
- 3.5. Atributos del sistema
- 3.6. Otros requisitos

4. Implementación y resultados

- 4.1. Implementación de aplicación procedural y paralela
- 4.2. Resultados

5. Conclusiones

6. Anexos

1. Introducción

1.1. Propósito

El propósito del presente trabajo consiste en desarrollar una aplicación que funcione como un procesador Doppler, es decir tomar las recepciones de los distintos canales de un radar meteorológico, brindadas a través de un archivo binario, y procesar las misma para obtener como resultado deseado la autocorrelación sobre cada canal que hay por cada componente de rango. El documento está dirigido a personas con conocimiento en programación paralela y especializado en este ámbito, para que pueda comprender a través de la ejecución de diferentes muestras sobre el programa las ventajas de rendimiento que se obtienen con la paralelización de procesos.

1.2. Ámbito del sistema

La aplicación desarrollada en el presente documento, está diseñada para los sistemas operativos Linux, cualquiera de su versiones podrá ejecutar dicho programa a través de la compilación del makefile correspondiente a la misma.

Esta aplicación procesa un archivo binario como entrada y obtiene un archivo binario de salida con resultados de autocorrelación discreta por componente de rango de un radar que es el dato requerido a través del procesamiento de la información. A su vez se podrá visualizar los tiempos de demora que son necesarios para el caso de lograr llevar a cabo la aplicación sin paralelización de procesos y explotando el paralelismo con la paralelización de procesos.

El objetivo es que los tiempos obtenidos explotando la paralelización, sean mucho mejor e impacten de manera significativa al comparar con la versión de la aplicación procedural. Para esto siempre se buscará amoldar los algoritmos para que consuman el menor tiempo posible.

1.3. Definiciones, acrónimos y abreviaturas

Procesador Doppler: Procesador usado en procesamiento de señales "pulse-doppler"

Pulse-Doppler signal procesing: Es una estrategia de mejora de rendimiento de un radar que permite la detección de objetos pequeños que se mueven a gran velocidad en cercanía con objetos más grandes también en movimiento.

Acimut: posición en grados sobre el eje horizontal.

Elevación: posición en grados sobre el eje vertical.

Fase: componente real de la señal electromagnética compleja.

Cuadratura: componente imaginaria de la señal electromagnética compleja.

Rango de resolución: rango en kilómetros de avance de cobertura mínimo, el mismo determina la cantidad de componentes de rango total de acuerdo al tamaño del rango de cobertura total.

Rango máximo de cobertura: cantidad de kilómetros máxima de cobertura alcanzada por el radar.

Gate: cada componente de rango. Por ejemplo 0.5km, 1 km, 1.5 km, y así sucesivamente hasta alcanzar el rango de cobertura máximo.

Librería OpenMP: Es una API para la programación multiproceso de memoria compartida.

API: interfaz de programación de aplicaciones, conjunto de funciones y subrutinas que ofrece una biblioteca.

Biblioteca: Conjunto de implementaciones funcionales.

Memoria compartida: Se refiere a un conjunto de procesadores que comparten memoria principal, como en el caso de las computadoras multinúcleos y a diferencia de un clúster.

Hardware: Referido a productos tangibles donde corre un software, computadora, clúster, etc.

Software: Todo lo referente a programas, código.

Conversor A/D: Conversor analógico digital, convierte una señal analógica en un valor digital que consta de palabras binarias

Radar: Acrónimo de RAdio Detection And Ranging, emite ondas electromagnética para la medición de distancias y velocidades entre otras cosas.

Procedural: Referido a un programa que posee solo un hilo de ejecución, todas las tareas se realizan secuencialmente.

Paralelo: Las tareas se pueden ejecutar en paralelo siempre y cuando no haya una dependencia fuerte entre una y otra.

Pulsos: Emisión de energía electromagnética de alta intensidad y corta duración

Clúster: Conjunto de computadoras autónomas que operan juntas para proporcionar mayor potencia que cada computadora individualmente.

Gate: Es una medida de la resolución de un radar

Autocorrelación: Herramienta estadística para el procesamiento de señales

Archivo binario: Archivo que posee palabras binarias, (valor 0 ó 1), no puede ser interpretado como un archivo de texto, sino que debe ser abierto con un programa dedicado.

Estructura: Se refiere a un bloque de código que puede representar como un nuevo tipo de datos que es definido fuera de los tipos de un lenguaje.

C90: Es un estándar del lenguaje C, existe además C89, C95, C99, y C11. La idea de estos estándares es mejorar la portabilidad de código.

UML: Unified Modeling Language, lenguaje de modelado unificado, es un estándar para el diseño de productos, que cuenta con diferentes tipos de diagramas.

Profiling: Análisis dinámico de programas

1.4. Referencias

[1] IEEE 830. (22 de Octubre, 2008). Especificación de requisitos según el estándar IEEE 830. [URL](#).

[2] Williams Stallings – Sistemas Operativos 5ta Edición.

1.5. Descripción General del documentado

Este documento se compone de seis secciones principales, siendo la primera la introducción, donde se explica brevemente el fin del proyecto como así también sus requerimientos de forma muy general. En la segunda sección la descripción general del sistema con el fin de conocer las principales funciones que debe ser capaz de realizar, conocer a quién va dirigido, además de restricciones y supuestos que puedan afectar el desarrollo del mismo. En la sección tres se definen de manera detallada los requerimientos del sistema a desarrollar, los que definen el comportamiento del sistema como así también otros requerimientos que puedan ser deseados considerando el uso que el sistema va a tener. En la cuarta sección se discuten los resultados de implementación del diseño de la aplicación final donde se muestra como el sistema opera. Por último en la sección cinco se elabora una breve conclusión acerca de la experiencia en la elaboración del proyecto. La sección seis, de Anexos, contiene información asociada a cierto tipos de temas mencionados en el trabajo que se extienden para mayor detalle.

2. Descripción General

2.1. Perspectiva del producto

La aplicación tiene un objetivo muy claro. Se trata de explotar el mayor paralelismo posible para visualizar las ventajas que proporciona el mismo. Con estos algoritmos se puede comprender que utilizándolos en otros sistemas o aplicaciones de mayor tamaño se puede buscar mejorar su rendimiento logrando explotar el paralelismo. La aplicación es utilizada como una interfaz que existe entre un radar meteorológico y los resultados de datos necesarios que se requieren para visualizarlos en pantalla, ya sea de una computadora, un smartphone o una página web, pero para todos los casos primero es necesario de esta aplicación y mejor será si explota de manera eficiente el paralelismo ya que los datos serán obtenidos directamente apenas sean creados sus archivos de recepción.

2.2. Funciones del producto

La aplicación nos permitirá las siguientes funciones:

- Funcionar como un procesador Doppler, capaz de tomar como entrada un archivo binario de pulsos el cual será procesado, devolviendo un archivo binario de salida como resultado y en el menor tiempo posible.
- La aplicación permitirá procesar la información a través de diferentes algoritmos con el fin de visualizar las mejoras cuando se usan algoritmos que explotan el paralelismo.
- La aplicación mostrará un menú de opciones con las distintas funcionalidades que se podrán realizar sobre los diferentes procesos vinculados a un algoritmo utilizado para llevar a cabo la resolución requerida. Las opciones serán para amoldar los parámetros de operación, como número de iteraciones sobre el proceso a ejecutar, el número de hilos a utilizar sobre el algoritmo de un proceso y el tipo de proceso a ejecutar referido a un tipo de algoritmo procedural o paralelo. Como también para visualizar todos los datos obtenidos, por ejemplo lectura e interpretación del archivo binario resultante.
- Se podrá ejecutar un proceso que utiliza en algoritmo procedural que obtendrá los resultados sobre el archivo binario deseado mostrando los tiempos de demora.
- Se podrá ejecutar un proceso que utiliza en algoritmo paralelo que obtendrá los resultados sobre el archivo binario deseado mostrando los tiempos de demora.
- La aplicación además permitirá poder ejecutarse de manera iterativa. Es decir si se desea ejecutar un mismo proceso 30 veces, esto se podrá indicar a través de un menú de opciones y de esa manera se podrá también saber el promedio de tiempo de demora sobre las 30 muestras.
- Para los procesos que utilizan algoritmos paralelos se podrá definir el número de hilos que se desean utilizar paralelamente en su ejecución.
- Toda información de estadísticas será accedida a través de una opción para visualizar los resultados estadísticos.
- También toda la información estadística podrá eliminarse para tomar nuevos análisis de muestras.

2.3. Características de los usuarios

La aplicación está destinada a usuarios especializados y con conocimientos sobre la programación paralela para que puedan comprender las mejoras significativas que se pueden obtener al explotar los algoritmos con la paralelización de procesos. Los usuarios deberán tener conocimientos sobre conceptos relacionados a los radares meteorológicos para poder interpretar los resultados obtenidos.

2.4. Restricciones

Como restricción de lenguaje de programación se tiene que la aplicación debe ser en su totalidad elaborada bajo lenguaje C. Se explota el paralelismo con la API OpenMP. Además como se mencionó el entorno donde corre la aplicación debe ser cualquier distribución basada en GNU/Linux.

En cuanto a hardware se requieren más de un procesador lógico, es decir puede ser un equipo con un procesador físico pero con capacidad de hyperthreading, con lo cual brinda la capacidad de ejecutar más de un proceso paralelamente. Cualquier computadora que pueda correr un GNU/Linux con las características anteriores es apta.

Para visualizar y concluir adecuadamente sobre las mejoras significativas que trae el paralelismo de procesos se deberá contar con un equipo con una cantidad significativa de núcleos físicos o con la capacidad de núcleos lógicos para poder visualizar la diferencia y ganancias de tiempo que existe entre un algoritmo procedural (un núcleo lógico/físico) y un algoritmo paralelo. Un equipo ideal para esta aplicación sería contar con un cluster o por lo menos con un acceso a estos para ejecutar la aplicación sobre el mismo.

2.5. Suposiciones y dependencias

La aplicación se diseña con el fin de ser ejecutada en cualquier sistema operativo Linux, por lo cual, esta es dependiente de estos sistemas operativos. En caso de cambiar de sistema operativo deberá revisarse si se adaptan todas las condiciones para el correcto funcionamiento de la aplicación.

Se supone que la cantidad de gates que cubren el rango máximo son 500. Esto es así debido a que el rango de cobertura máximo es de 250 km de radio, entonces si cada componente fuera cada un km se tendría 250 gates de cantidad, pero como en nuestro caso se tiene una resolución 0.5 km por componente en las 250 km entran 500 componentes. Es decir que cada km se tienen 2 componentes de 0,5 km cada una por lo cual en las 250 km da un total de 500 componentes.

Cargando...

La cantidad de pulsos contenidos en el archivo binario *pulsos.iq* es una constante estática, ya que por grado de acimut se tienen 100 pulsos emitidos y como también se proporciona el dato de que los pulsos emitidos desde los 83° a los 90° en acimut, se determina que hay información de 8 grados diferentes con lo que se concluye que los pulsos del archivo son 800.

- Grados de acimut considerados: 8.
- Cantidad de pulsos por grado de acimut: 100.
- Total de pulsos en archivo: $8 \times 100 = 800$.

El tipo de archivo que contiene la información a procesar por la aplicación es binario por lo cual si en un futuro se cambiará la estructura de los datos del archivo binario deberán ajustarse los cambios a la aplicación cuando analiza el archivo.

Las características del archivo del que depende la aplicación son las siguientes:

- Tipo de archivo: binario.
- Nombre de archivo: pulsos.iq
- Estructura de archivo binario:
 - validSamples (tipo unit16_t).
 - Para $i=0,1,2,\dots$, validSamples se tiene: $V_I[i]$ $V_Q[i]$ (tipo float).
 - Para $i=0,1,2,\dots$, validSamples se tiene: $H_I[i]$ $H_Q[i]$ (tipo float).

Las características del archivo de salida son las siguientes:

- Tipo de archivo: binario.
- Nombre de archivo: resultados_adpc
- Estructura de archivo binario:
 - Cantidad_de_datos (tipo int).
 - Para $i=0,1,2,\dots, 500$ se tiene: $R_V[i]$ (tipo double).
 - Para $i=0,1,2,\dots, 500$ se tiene: $R_H[i]$ (tipo double).

En caso de que cambie el nombre, el tipo o la estructura del archivo de entrada o de salida deberán revisarse los requerimientos asociados a la lectura y/o escritura de los mismos.

2.6. Requisitos futuros

Como requerimientos futuros para lograr mejorar más la implementación y adecuar cada vez mejor la aplicación implementada se proponen los siguiente requerimientos:

- Mejorar el algoritmo de lectura de archivos. Sabemos que los mayores costos están dados por el nivel de jerarquía más bajo de la memoria, es por lo cual es importante mantener un correcto algoritmo sobre la lectura y escritura de los ficheros para de esta manera mantener optimizado los tiempos de acceso a los espacios de memoria de menor jerarquía.
- Adaptaciones a diferentes tipos de ficheros con tamaños variables, puede presentarse la situación de tener que tomar un mismo archivo pero sobre diferentes tamaños de datos, para esto se propone utilizar algoritmos que detectan estas lecturas de archivos de manera de determinar los tamaños de dimensiones necesarios para adecuar sus recursos en tiempo de ejecución.
- La interpretación de archivos binarios con diferentes nombres podría ser otro requisito futuro, lo que permitirá seleccionar que archivo se desea utilizar para de esta manera procesar la información en archivos diferentes.

3. Requisitos específicos

3.1. Interfaces externas

Interfaz de software:

La interfaz de software, es por selección de opciones a través de un menú de opciones proporcionado por la ejecución de la aplicación. Al ejecutar la aplicación se muestran las opciones y se deberá seleccionar la opción de acuerdo al número que se le corresponde.

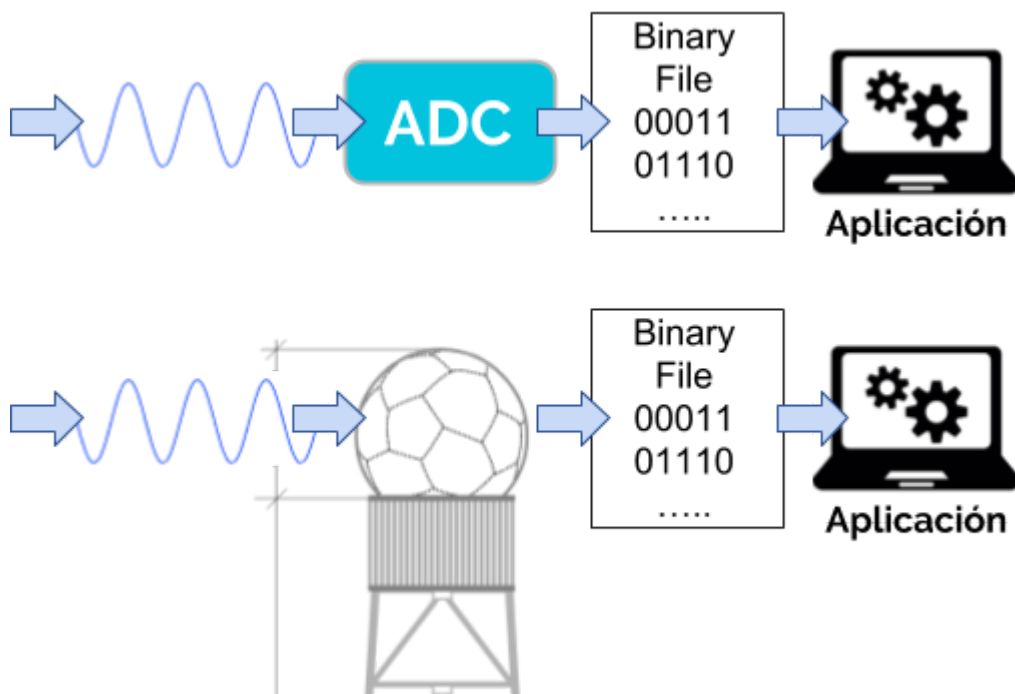
```
>>>_                MENU                _<<<

1. Cambiar numero de iteraciones por algoritmo.
2. Cambiar numero de hilos paralelos.
3. Ejecutar algoritmo procedural.
4. Ejecutar algoritmo procedural con solicitud de resultados.
5. Ejecutar algoritmo paralelo.
6. Ejecutar algoritmo paralelo con solicitud de resultados.
7. Ver estadísticas de algoritmo procedural.
8. Ver estadísticas de algoritmo paralelo.
9. Borrar estadísticas.
10. Mostrar resultados desde archivo binario resultados_adpc.
11. Salir.

Introduzca opcion (1-11): 7
```

Interfaz de hardware y comunicación:

No se tiene interfaz de hardware ya que existe otro dispositivo del sistema que conforma el radar encargado de obtener un archivo binario que es el que se utiliza como entrada de la aplicación del presente documento.



3.2. Funciones

Como esta aplicación es utilizada por personal capacitado y especializado en el manejo de software avanzado (como único tipo de usuario), se detallan las funciones de la aplicación de acuerdo a los objetivos que se deben llevar a cabo en las mismas para que funcione todo adecuadamente.

La aplicación diseñada cuenta con las siguientes funciones, que se seccionan de acuerdo a los tipos de procesos que serán ejecutados. En primer lugar se describen las funciones de los procesos que usaran el algoritmo paralelo o procedural y por último se describirán las funciones que deberá cumplir la interfaz para ejecutar de manera adecuada estos procesos.

Funciones de los procesos ejecutados por la aplicación principal

Los procesos que serán ejecutados por la aplicación principal serán los que se asocian con los programas que contendrán los algoritmos de resolución de la autocorrelación y que podrán ser procedural o paralelo. En ambos casos se deberán cumplir con las siguientes funciones:

- El programa deberá ser capaz de leer un archivo binario e interpretar la información para luego poder procesarla de acuerdo a las necesidades. Se tiene en cuenta que este archivo tiene una estructura proporcionada por la cátedra.
- Por cada canal (vertical y horizontal), se deberá obtener el valor de cada gate de acuerdo a la media aritmética de muestras que existan para el mismo. El resultado es una matriz gate-pulso que contendrá por cada pulso (columnas de matriz) el valor de gate asociado a cada componente de rango (filas de matriz). Para lograr esto se creó la función *gate_por_canal()* que se encarga de construir la estructura de datos gate-pulso. Para esta función se presentan distintas situaciones que dependen de la cantidad de muestras (samples) que existan por pulso, según las indicaciones del archivo binario. Esto es un problema ya que si el número de muestras se debe repartir para todas las componentes de rango que conforma a cada gate. El problema surge del hecho de que repartir puede variar de acuerdo a la cantidad muestras y se presentan tres casos diferentes:
 - El caso ideal es que el número de repartida para cada componente asociada a un gate sea igual para cada componente de rango, es decir sea un entero, por ejemplo sea de 10 en todos los casos igual.
 - Otro caso es que no sea entero y se debe redondear para abajo, por ejemplo si el número a repartir por cada componente de rango sea de 10.1, se redondea a 10 pero surge el conflicto de que se introduce un pequeño error a medida que se avanza en los promedios por componentes asociadas a cada gate. La solución es corregir el error a medida que su acumulación empieza a ser significativa.
 - Y el último caso es el que se presenta opuestamente al anterior, es decir cuando se debe redondear hacia arriba el número a repartir por componente de rango asociada a un gate, por ejemplo si es de 10.9, hay menos error si se redondea a 11 pero a medida que se avanza por promedios de componentes de rangos diferentes se va teniendo un pequeño error que deberá solucionarse corrigiéndose cuando este empieza a ser significativo como el caso anterior.

Para resolver todos los problemas anteriores se utiliza en la función *gate_por_canal()* una codificación capaz de poder seleccionar diferentes algoritmo de corrección de error de

acuerdo al caso que se presente para de esta manera tener un cálculo de los resultados de la manera más adecuada.

- Por último el programa deberá ser capaz de resolver una autocorrelación discreta por canal para de esta manera obtener la información requerida. Para esto se utiliza la función *autocorrelacion()* que se encarga de obtener la autocorrelación discreta para cada uno de los canales a través de un algoritmo encargado de dicha tarea.
- Todos los resultados de la autocorrelación por canal deberá ser almacenada en un archivo binario de acuerdo a una estructura indicada en la presente documentación. El archivo que contendrá los resultados se denomina *resultados_adpc*, haciendo referencia a los resultados de la autocorrelación discreta por pulsos de cada canal.

Funciones de la aplicación de interfaz que utilizará el usuario final

La aplicación principal involucra a los programas anteriores que cumplen con las requerimientos descritos asociados a los casos del algoritmo procedural y el algoritmo paralelo. Esta aplicación es una interfaz de usuario final automatizada y con flexibilidad de configuración de parámetros para poder realizar las muestras y determinar las estadísticas de los resultados como el análisis de los archivos resultantes. La aplicación contará con las siguientes funciones:

- La aplicación deberá permitir configurar el número de iteraciones que se deseen llevar a cabo al ejecutar un programa procedural o un programa paralelo.
- La aplicación deberá permitir configurar el número de hilos a utilizar por el programa que hará uso de los mismo a través de su explotación de paralelismo.
- La aplicación nos permitirá, por medio de una opción, ejecutar el programa procedural.
- La aplicación nos permitirá, por medio de una opción, ejecutar el programa procedural con solicitud de los resultados que obtenga en la autocorrelación.
- La aplicación nos permitirá, por medio de una opción, ejecutar el programa paralelo.
- La aplicación nos permitirá, por medio de una opción, ejecutar el programa paralelo con solicitud de los resultados que obtenga en la autocorrelación.
- La aplicación nos permitirá ver las estadísticas del programa procedural.
- La aplicación nos permitirá ver las estadísticas del programa paralelo.
- La aplicación nos permitirá borrar las estadísticas almacenadas.
- La aplicación nos permitirá abrir el archivo binario de resultados.
- La aplicación nos permitirá poder finalizar con la misma.

3.3. Requisitos de rendimiento

La aplicación deberá soportar un número significativos de muestras para poder determinar promedios de tiempo. En este caso con 30 o más de 30 son un conjunto de muestras suficientes para la ejecución de los diferentes algoritmos que se podrán ejecutar. Todos los resultados obtenidos por los algoritmos serán almacenados en un archivo binario de salida al cual denominamos *resultados_adpc*, haciendo referencia a los resultados de auto correlación discreta de cada pulso por canal que contiene el radar.

3.4. Restricciones de diseño

Las restricciones de diseño son que el mismo deberá ser llevado a cabo mediante el lenguaje de programación C. Además se deberá contar con un equipo con un sistema operativo Linux y que cuente con más de un núcleo lógico para poder explotar el paralelismo. Para explotar el paralelismo se utiliza la API OpenMP.

3.5. Atributos del sistema

La aplicación busca tener el mejor rendimiento posible. Es por lo cual no se trata de una aplicación que requiera seguridad en cuanto a la protección de datos, por lo que podrá ser utilizado por cualquier usuario sin necesidad de contener una clave o usuario habilitado.

En base a la busca del mejor rendimiento se determinara que este será mejor cuando los resultados de tiempos obtenidos por el programa paralelo muestran mejoras de rendimiento en comparación a los resultados de tiempo obtenidos por el programa procedural.

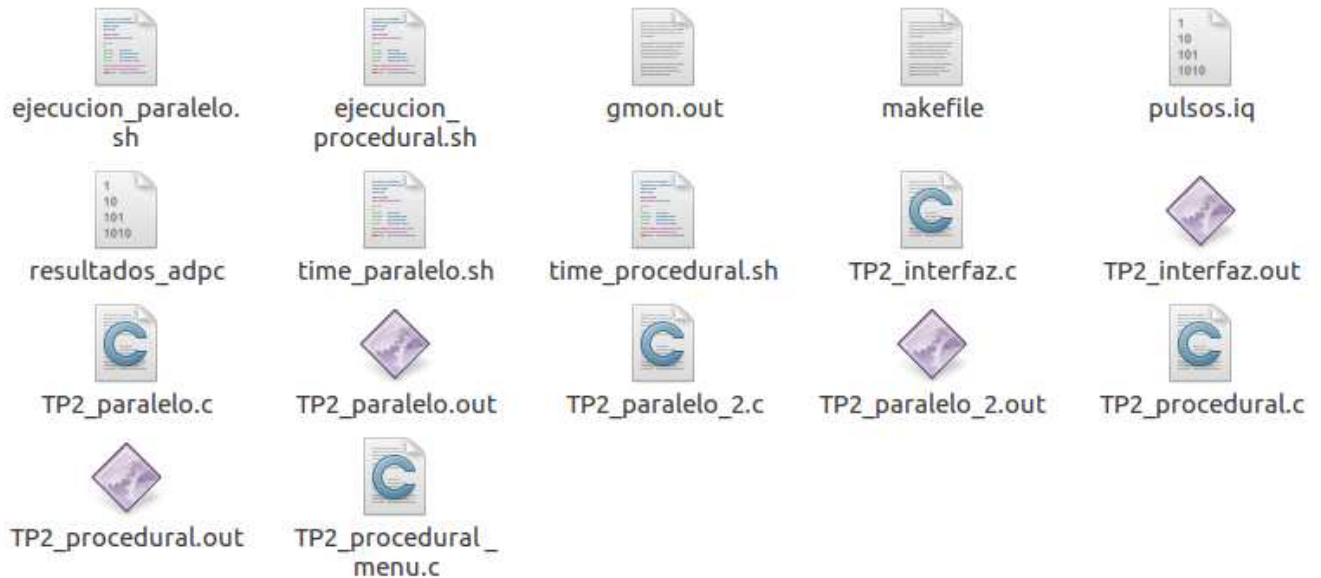
3.6. Otros requisitos

La aplicación que será utilizada un usuario se diseña de tal forma que se pueda unificar la mayor cantidad de funcionalidades en un solo lugar, por este motivo se detectaron un par de requisitos extras que contribuyen al mejor funcionamiento de la aplicación. Estos son:

- Proporcionar un menú de opciones para poder realizar de manera adecuada la configuración de los diferentes parámetros que afectan al sistema con el fin de adecuar todas las necesidades a través de opciones.
- Se deberá contar con una opción que permita ejecutar el algoritmo que determina los resultados sobre un archivo binario.
- Se deberá contar con una opción que permita poder configurar el número de iteraciones que se desean realizar sobre una misma aplicación.
- Se deberá contar con una opción que permita visualizar los valores de tiempos promedios de acuerdo a las ejecuciones previas de los diferentes algoritmos,
- Se deberá contar con una opción que permita poder leer e interpretar los valores contenidos en el archivo binario resultante.

4. Implementación y resultados

4.1. Implementación de aplicación procedural y paralela



Mediante el archivo makefile se logra compilar el programa. Ejecutando el siguiente comando:

```
make
```

Se obtiene la compilación de los programas TP2_procedural.c, TP2_paralelo_2.c, TP2_interfaz.c Devolviendo los programas ejecutables TP2_procedural.out, TP2_paralelo_2.out, TP2_interfaz.out. Luego ejecutando

```
./TP2_interfaz.out
```

Se iniciará la ejecución del proceso ejecutable TP2_interfaz.out.

Este proceso principal será el encargado de llamar al resto de los procesos, en este caso se encarga de llamar al proceso TP2_procedural.out y al proceso TP2_paralelo_2.out cuando se lo solicite un usuario.

Tanto el proceso TP2_procedural.out como también el proceso TP2_paralelo_2.out utilizan el archivo *pulsos.iq* para poder tomar como entrada todos los pulsos proporcionados por el radar meteorológico. Luego de procesar los datos se devuelve el archivo binario de salida resultados_adpc, que contiene todos los resultados obtenidos por el algoritmo procedural o paralelo según sea el caso.

4.2. Resultados

Durante el inicio de ejecución de la aplicación se obtiene la siguiente pantalla:

```
$ ./TP2_interfaz.out

>>>_          MENU          _<<<

1. Cambiar numero de iteraciones por algoritmo.
2. Cambiar numero de hilos paralelos.
3. Ejecutar algoritmo procedural.
4. Ejecutar algoritmo procedural con solicitud de resultados.
5. Ejecutar algoritmo paralelo.
6. Ejecutar algoritmo paralelo con solicitud de resultados.
7. Ver estadisticas de algoritmo procedural.
8. Ver estadisticas de algoritmo paralelo.
9. Borrar estadisticas.
10. Mostrar resultados desde archivo binario resultados_adpc.
11. Salir.

Introduzca opcion (1-11): █
```

Para ejecutar el proceso con el algoritmo procedural, como vemos en la captura anterior, se deberá elegir la opción 3. Luego de la ejecución se devuelven los resultados como muestra la siguiente captura:

```
>>>_          MENU          _<<<

1. Cambiar numero de iteraciones por algoritmo.
2. Cambiar numero de hilos paralelos.
3. Ejecutar algoritmo procedural.
4. Ejecutar algoritmo procedural con solicitud de resultados.
5. Ejecutar algoritmo paralelo.
6. Ejecutar algoritmo paralelo con solicitud de resultados.
7. Ver estadisticas de algoritmo procedural.
8. Ver estadisticas de algoritmo paralelo.
9. Borrar estadisticas.
10. Mostrar resultados desde archivo binario resultados_adpc.
11. Salir.

Introduzca opcion (1-11): 3

    * El numero de pulsos que tiene el archivo pulsos.iq es: 800
    * Las componetes de rango asociadas a cada gate son: 500
    * Se utilizan estructuras de datos del tipo matriz[GATES][PULSO]
canal[500][800]

    * Tiempo autocorrelación: 0.00348
    * Tiempo requerido para calculos (calculo de GATES y autocorrela
on): 0.34245
    * Tiempo de demora total de iteracion 1: 0.35742

    * Archivo binario de salida con resultados resultados_adpc
```


La ejecución anterior se realiza con una sola iteración, para cambiar el número de muestras en la ejecución de los procesos se deberá elegir la opción 1. La misma pedirá que se ingrese el número deseado de iteraciones/muestras y quedará configurado hasta un próximo cambio.

```
>>>_          MENU          _<<<

1. Cambiar numero de iteraciones por algoritmo.
2. Cambiar numero de hilos paralelos.
3. Ejecutar algoritmo procedural.
4. Ejecutar algoritmo procedural con solicitud de resultados.
5. Ejecutar algoritmo paralelo.
6. Ejecutar algoritmo paralelo con solicitud de resultados.
7. Ver estadísticas de algoritmo procedural.
8. Ver estadísticas de algoritmo paralelo.
9. Borrar estadísticas.
10. Mostrar resultados desde archivo binario resultados_adpc.
11. Salir.

Introduzca opcion (1-11): 1

> Introduzca el numero de iteraciones deseado: 10
> Se establecieron 10 iteraciones.
```

Al ejecutar nuevamente con la opción 3 ahora se realizará con el número de veces ingresado en la opción anterior:

```
Introduzca opcion (1-11): 3

* El numero de pulsos que tiene el archivo pulsos.iq es: 800
* Las componetes de rango asociadas a cada gate son: 500
* Se utilizan estructuras de datos del tipo matriz[GATES][PULSO] = canal[500][800]

* Tiempo autocorrelación: 0.00352
* Tiempo requerido para calculos (calculo de GATES y autocorrelacion): 0.34748
* Tiempo de demora total de iteracion 1: 0.36426

* Tiempo autocorrelación: 0.00367
* Tiempo requerido para calculos (calculo de GATES y autocorrelacion): 0.30474
* Tiempo de demora total de iteracion 2: 0.31836

* Tiempo autocorrelación: 0.00362
* Tiempo requerido para calculos (calculo de GATES y autocorrelacion): 0.30601
* Tiempo de demora total de iteracion 3: 0.32031

* Tiempo autocorrelación: 0.00362
* Tiempo requerido para calculos (calculo de GATES y autocorrelacion): 0.27570
* Tiempo de demora total de iteracion 4: 0.28027

* Tiempo autocorrelación: 0.00350
* Tiempo requerido para calculos (calculo de GATES y autocorrelacion): 0.29499
* Tiempo de demora total de iteracion 5: 0.30469
```

Para cambiar el número de hilos que se requieren en la ejecución del proceso con el algoritmo paralelo se deberá seleccionar la opción 2. Con la misma se configura el número de hilos para la paralización el proceso que explota el paralelismo y permanecerá dicho valor hasta una nueva modificación:

```
>>>_          MENU          _<<<

1. Cambiar numero de iteraciones por algoritmo.
2. Cambiar numero de hilos paralelos.
3. Ejecutar algoritmo procedural.
4. Ejecutar algoritmo procedural con solicitud de resultados.
5. Ejecutar algoritmo paralelo.
6. Ejecutar algoritmo paralelo con solicitud de resultados.
7. Ver estadisticas de algoritmo procedural.
8. Ver estadisticas de algoritmo paralelo.
9. Borrar estadisticas.
10. Mostrar resultados desde archivo binario resultados_adpc.
11. Salir.

Introduzca opcion (1-11): 2

> Introduzca el numero de hilos paralelos deseados: 2
  * Comando 1:./TP2_paralelo_2.out 2
  * Comando 2:./TP2_paralelo_2.out 2 1
  * hilos: 2
```

Si se desea ejecutar el algoritmo paralelo se deberá elegir la opción 5. La salida resultante se muestra en la siguiente captura (solo para una iteración):

```
>>>_          MENU          _<<<

1. Cambiar numero de iteraciones por algoritmo.
2. Cambiar numero de hilos paralelos.
3. Ejecutar algoritmo procedural.
4. Ejecutar algoritmo procedural con solicitud de resultados.
5. Ejecutar algoritmo paralelo.
6. Ejecutar algoritmo paralelo con solicitud de resultados.
7. Ver estadisticas de algoritmo procedural.
8. Ver estadisticas de algoritmo paralelo.
9. Borrar estadisticas.
10. Mostrar resultados desde archivo binario resultados_adpc.
11. Salir.

Introduzca opcion (1-11): 5

  * El numero de pulsos que tiene el archivo pulsos.iq es: 800
  * Las componetes de rango asociadas a cada gate son: 500
  * Se utilizan estructuras de datos del tipo matriz[GATES][PULSO]

  * Tiempo autocorrelación: 0.00204
  * Tiempo requerido para calculos (calculo de GATES y autocorrel.
tiempo de demora total de iteracion 1: 0.29102

archivo binario de salida con resultados resultados_adpc
```

Para mostrar los resultados del archivo binario resultante se deberá elegir la opción 10. La misma interpreta la estructura que mantiene el archivo binario de salida.

```
>>>_                               MENU                               _<<<

1. Cambiar numero de iteraciones por algoritmo.
2. Cambiar numero de hilos paralelos.
3. Ejecutar algoritmo procedural.
4. Ejecutar algoritmo procedural con solicitud de resultados.
5. Ejecutar algoritmo paralelo.
6. Ejecutar algoritmo paralelo con solicitud de resultados.
7. Ver estadisticas de algoritmo procedural.
8. Ver estadisticas de algoritmo paralelo.
9. Borrar estadisticas.
10. Mostrar resultados desde archivo binario resultados_adpc.
11. Salir.

Introduzca opcion (1-11): 10
cantidad 500
R_V[0] = 0.0608889020, R_H[0] = 0.0209738172
R_V[1] = 0.0271063099, R_H[1] = 0.0089635151
R_V[2] = 0.0203369489, R_H[2] = 0.0069478597
R_V[3] = 0.0182197291, R_H[3] = 0.0064507396
R_V[4] = 0.0152895394, R_H[4] = 0.0055833892
R_V[5] = 0.0123240922, R_H[5] = 0.0044057295
R_V[6] = 0.0108640468, R_H[6] = 0.0037330334
R_V[7] = 0.0081694692, R_H[7] = 0.0028925044
R_V[8] = 0.0072433521, R_H[8] = 0.0023225721
R_V[9] = 0.0068533877, R_H[9] = 0.0022683151
R_V[10] = 0.0062679001, R_H[10] = 0.0020786164
R_V[11] = 0.0043748245, R_H[11] = 0.0013370025
```

Podemos notar cómo e interpreta el archivo binario resultante de acuerdo a la cantidad de valores que contiene y con cada uno de los datos sobre los diferentes canales obtenidos para cada caso. Por último para validar si los resultados son correctos se puede verificar eligiendo la opción 4 o 6 la cual nos permite poder comparar los valores que hay en el archivos de salida escrito por los procesos, con los valores contenidos en las estructuras de datos que mantienen los procesos en tiempo de ejecución. Seguramente se notará que los valores son los mismos (caso en el cual no hay errores en la escritura de datos), o en caso de errores serán distintos por lo que se deberá revisar el código.

Resultados de estadísticas

Sobre el cluster de la facultad se utilizó el nodo pulqui.

Se tomaron 30 muestras para la ejecución de cada uno de los procesos paralelo y procedural y además para el caso del algoritmo paralelo se varió número de hilos de manera incremental.

Primero ejecutamos el proceso con el algoritmo procedural para poder comparar los resultados en base a este.


```

>>>_ ESTADISTICAS DE ALGORITMO _<<<
>>>_ ALGORITMO PROCEDURAL _<<<
| Cores | Tiempo | Muestras |
| 1 | 0.1417 | 30 |

```

Luego realizamos las muestras sobre el proceso paralelo con el número de hilos incrementando hasta observar que no escala el algoritmo paralelo. En promedio el tiempo de demora por muestras es el siguiente:

```

Introduzca opcion (1-11): 8
>>>_ ESTADISTICAS DE ALGORITMOS _<<<
>>>_ ALGORITMO PARALELO _<<<
| Cores | Tiempo | SpeedUp | Porcentaje | Muestras |
| 1 | 0.1417 | 1.0000 | 0% | 30 |
| 2 | 0.1417 | 1.0000 | 0% | 30 |
| 4 | 0.1083 | 1.3077 | 31% | 30 |
| 8 | 0.0875 | 1.6190 | 62% | 30 |
| 16 | 0.0958 | 1.4783 | 48% | 30 |
| 32 | 0.3667 | 0.3864 | -61% | 30 |

```

Podemos notar que para el caso de 16 hilos y para 32 empieza a descender el escalado del algoritmo paralelo.

Para comparar resultados con otros de los nodos de la facultad se utilizó también el nodo franky. Los resultados son los siguientes:

```

Introduzca opcion (1-11): 7

>>>_          ESTADISTICAS DE ALGORITMO          _<<<
-----
>>>_          ALGORITMO PROCEDURAL          _<<<
-----
|  Cores  |  Tiempo  |  Muestras  |
-----
|    1    |  0.3500  |    30      |
-----

```

Luego de tomar la muestras para el proceso paralelo para las diferentes cantidades de hilos se obtuvieron los siguientes resultados:

```

Introduzca opcion (1-11): 8

>>>_          ESTADISTICAS DE ALGORITMOS          _<<<
-----
>>>_          ALGORITMO PARALELO          _<<<
-----
|  Cores  |  Tiempo  |  SpeedUp  |  Porcentaje  |  Muestras  |
-----
|    1    |  0.3500  |  1.0000   |    0%        |    30      |
-----
|    2    |  0.3500  |  1.0000   |    0%        |    30      |
-----
|    4    |  0.2958  |  1.1831   |    18%       |    30      |
-----
|    8    |  0.2500  |  1.4000   |    40%       |    30      |
-----
|   16    |  0.2292  |  1.5273   |    53%       |    30      |
-----
|   32    |  0.2375  |  1.4737   |    47%       |    30      |
-----
|   48    |  1.1833  |  0.2958   |   -70%      |    30      |
-----

```

En este nodo se nota que los resultados de tiempo son mayores a comparación del anterior. Pero en cuanto a la escalabilidad se puede decir que en este caso recién deja de escalar cuando se utilizan 32 hilos para el algoritmo.

Por último sobre el nodo cabecera de la facultad, nodo rocky se tienen los siguientes resultados.

```
Introduzca opcion (1-11): 7

>>>_          ESTADISTICAS DE ALGORITMO          _<<<
-----
>>>_          ALGORITMO PROCEDURAL          _<<<
-----
| Cores | Tiempo | Muestras |
-----
|   1   | 0.1833 |    30    |
-----
```

```
Introduzca opcion (1-11): 8

>>>_          ESTADISTICAS DE ALGORITMOS          _<<<
-----
>>>_          ALGORITMO PARALELO          _<<<
-----
| Cores | Tiempo | SpeedUp | Porcentaje | Muestras |
-----
|   1   | 0.1833 | 1.0000 |    0%     |    30    |
-----
|   2   | 0.1917 | 0.9565 |   -4%     |    30    |
-----
|   4   | 0.1708 | 1.0732 |    7%     |    30    |
-----
|   8   | 0.1375 | 1.3333 |   33%     |    30    |
-----
|  16   | 0.2625 | 0.6984 |  -30%     |    30    |
-----
```

Se observa que el algoritmo paralelo deja de escalar cuando se utilizan 16 hilos.

5. Conclusiones

El presente trabajo nos deja en claro cómo podemos paralelizar distintos tipos de problemas como también cuales son las dificultades que existen al paralelizar sobre los mismos. En este caso se explota la paralelización haciendo uso de OpenMP que es una interfaz de programación de aplicaciones (API) para poder manipular explícitamente el multiprocesamiento y el paralelismo de memoria compartida. Con esta API pudimos conocer básicamente cuáles son algunas de las directivas para interactuar en con las aplicaciones que se quieren convertir de secuencial a paralelo. Conocimos también las cláusulas que brinda la API necesarias para indicar cómo debe manipularse las variables a las que se asociará cada hilos. Sobre el manejo de los for conocimos que por medio los `schedule` podemos indicarle cual debe ser la manipulación sobre las iteraciones que recibirá cada uno de los hilos. Además la API nos brinda funciones para poder manejar los hilos tanto como qué cantidad utilizar como también controlar el tiempo de ejecución o capturar información de `get` como por ejemplo cuántos hilos existe como máximo en un equipo entre muchas otras funcionalidades.

6. Anexos

6.1. Mediciones de tiempo en ejecución procesos

Para la medición de los tiempos en la ejecución de los procesos se realizó una investigación acerca de diferentes herramientas que puedan ser útiles para dicha tarea dentro de las cuales nos centramos en tres casos, en donde concluimos que uno es el que más se adapta a nuestras necesidades. Las tres alternativas analizadas son :

- El comando time.
- El comando date.
- La función omp_get_wtime().

A continuación describimos cada una de estas.

Comando time

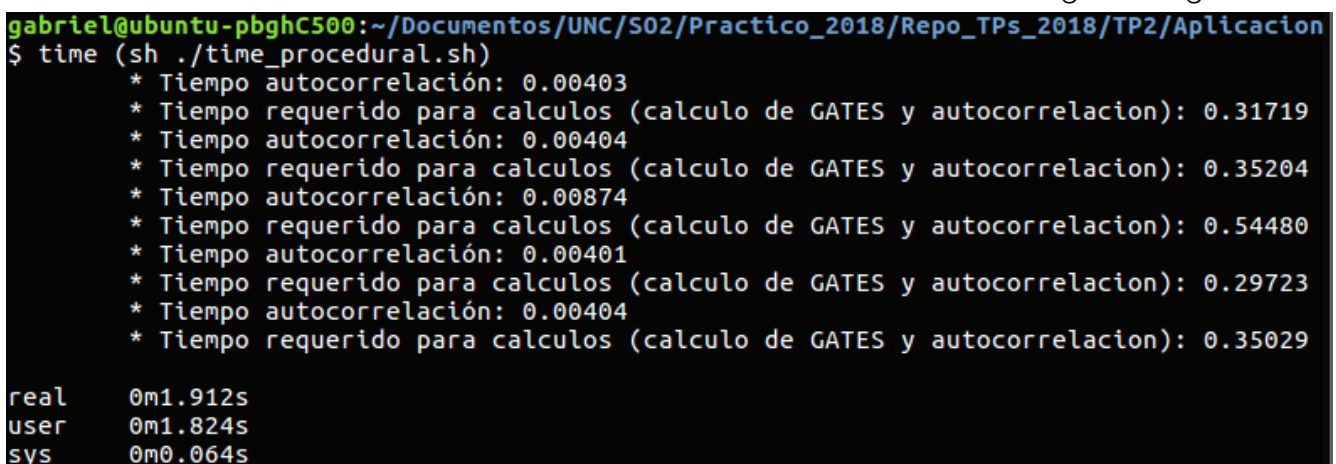
Esta herramienta consiste en ejecutar un proceso y medir el tiempo que demora su ejecución. Para esto simplemente debe ejecutar sobre la terminal el siguiente comando:

```
time proceso_a_ejecutar
```

Por ejemplo si combinamos esta herramienta con el uso de un script bash como el siguiente:

```
1. #!/bin/bash
2. # Variables
3. muestras=5
4. hilos=4
5.
6. for i in $(seq 1 $muestras); do
7.     ./TP2_procedural.out
8. done
```

Para nuestro caso una salida de esta herramientas es como se observa en la siguiente figura:



```
gabriell@ubuntu-pbghC500:~/Documentos/UNC/S02/Practico_2018/Repo_TPs_2018/TP2/Aplicacion
$ time (sh ./time_procedural.sh)
    * Tiempo autocorrelación: 0.00403
    * Tiempo requerido para calculos (calculo de GATES y autocorrelacion): 0.31719
    * Tiempo autocorrelación: 0.00404
    * Tiempo requerido para calculos (calculo de GATES y autocorrelacion): 0.35204
    * Tiempo autocorrelación: 0.00874
    * Tiempo requerido para calculos (calculo de GATES y autocorrelacion): 0.54480
    * Tiempo autocorrelación: 0.00401
    * Tiempo requerido para calculos (calculo de GATES y autocorrelacion): 0.29723
    * Tiempo autocorrelación: 0.00404
    * Tiempo requerido para calculos (calculo de GATES y autocorrelacion): 0.35029

real    0m1.912s
user    0m1.824s
sys     0m0.064s
```

Podemos notar en la herramienta que para nuestro ejemplo nuestro propio programa lanza mediciones de tiempos a las que este puede detallar, pero por fuera de nuestro programa, se requiere de la herramienta para determinar su tiempo de demora total.

Ventajas:

En cuanto a las ventajas que pudimos determinar con esta herramientas son las siguientes:

- Herramienta simple y rápida para utilizar.
- Previamente instalada en ubuntu 16.04 LTS. Por lo cual no es necesario descargar nada.
- Precisión de medición de tiempo (hasta milésimas de segundos).
- Flexibilidad para el manejo de cantidad de muestras y cantidad de hilos a utilizar.

Desventajas:

- Se debe combinar con el uso de scripts bash. Los cuales son muy susceptibles a errores en su programación.
- Se deberá poseer conocimientos en scripts bash.
- Modificación de parámetros estática., donde se debe actualizar los códigos de scripts bash cada vez que se desea modificar el número de hilos o cantidad de muestras a ejecutar.
- Y se debe procesar la salida obtenida para determinar el cálculo del tiempo por muestra que existe.
- Poco flexibilidad para adaptar a los programas a resultados de tiempos y estadísticas deseados.
- Escasa información de complemento en la nube.

Comando date

Esta herramienta puede ser utilizada para medir el tiempo ya que la misma nos proporciona el tiempo actual. Con esta función se la puede utilizar para tomar el tiempo actual todo en segundos, y luego procesar otro proceso para luego al final tomar el tiempo actual nuevamente y con estas dos muestras del tiempo si se realiza la diferencia se debería obtener la demora del proceso ejecutado.

Por ejemplo en combinación con los script bash se puede obtener el siguiente script para determinar el tiempo de demora de un proceso:

```

1.  #!/bin/bash
2.  # Variables
3.  muestras=5
4.  resta=0
5.  start_time=$(date +%s)
6.
7.  for i in $(seq 1 $muestras); do
8.      ./TP2_procedural.out
9.  done
10.
11. finish_time=$(date +%s)
12. resta=$((finish_time - start_time))
13. tiempo_total=$(( calc $resta/$muestras))
14.
15. echo
16. echo "> El tiempo de demora total para las $muestras muestras es: $resta segundos."
17. echo "> El tiempo promedio por cada muestra es:" $tiempo_total "segundos."

```

Un ejemplo de su ejecución sería como sigue:

```

gabriel@ubuntu-pbghC500:~/Documentos/UNC/S02/Practico_2018/Repo_TPs_2018/TP2/Aplicacion
$ sh ./ejecucion_procedural.sh
* Tiempo autocorrelación: 0.01296
* Tiempo requerido para calculos (calculo de GATES y autocorrelacion): 0.94189
* Tiempo autocorrelación: 0.01302
* Tiempo requerido para calculos (calculo de GATES y autocorrelacion): 0.95414
* Tiempo autocorrelación: 0.01474
* Tiempo requerido para calculos (calculo de GATES y autocorrelacion): 0.97085
* Tiempo autocorrelación: 0.01352
* Tiempo requerido para calculos (calculo de GATES y autocorrelacion): 1.02723
* Tiempo autocorrelación: 0.01464
* Tiempo requerido para calculos (calculo de GATES y autocorrelacion): 0.97333
> El tiempo de demora total para las 5 muestras es: 5 segundos.
> El tiempo promedio por cada muestra es: 1 segundos.

```

Ventajas:

- Fácil y rápida de utilizar.
- Su salida es fácil de procesar. Ya que devuelve la cantidad de segundos (un dato entero).
- Si el tiempo es de varios segundos se puede obtener buena precisión.
- Flexible para la configuración de la cantidad de muestras y cantidad de hilos a utilizar por proceso.

Desventajas:

- Será necesario instalar `apcalc` (`sudo apt install apcalc`). El programa es necesario para procesar los datos más fácilmente con operaciones aritméticas.
- Se debe combinar con el uso de scripts `bash`. Los cuales son muy susceptibles a errores en su programación.
- Se deberá poseer conocimientos en scripts `bash`.
- Modificación de parámetros estática, donde se debe actualizar los códigos de scripts `bash` cada vez que se desea modificar el número de hilos o cantidad de muestras a ejecutar.
- Si se quiere medir tiempos menores a 1 segundo no es una herramienta recomendada.
- Poca flexibilidad para adaptar a los programas a resultados de tiempos y estadísticas deseados.
- Escasa información de complemento en la nube.

Función `omp_get_wtime()`

Esta herramienta a diferencia de las anteriores, no es un comando es por lo cual no se vincula con los scripts `bash`. Se trata de una de las funciones que brinda la biblioteca de `openMP` para medir el tiempo con precisión en bloques de código de nuestros programas. Una de las formas de usar esta herramienta es crear un programa de C principal encargado de ejecutar otros programas independientes y medir el tiempo de su ejecución por fuera. Esta es la alternativa que más se adapta a lo que buscamos porque sobre la misma podemos tener todo en un mismo lugar y sacar estadísticas con los datos obtenidos.

Un ejemplo del uso de esta herramienta es como se nota en la siguiente sección de código:

```

1. case 3: /* Ejecutar algoritmo procedural. */
2.     for(i = 0; i < iteraciones; i++)
3.     {
4.         total_i=omp_get_wtime();
5.         /* Ejecucion de proceso con algoritmo procedural*/

```

```

6.         system(comando_procedural_modol);
7.         total_f=omp_get_wtime();
8.         t_est = t_est + (total_f-total_i);
9.         imp++;
10.        printf("\t* Tiempo de demora total de iteracion %d: %.5f\n\n",i+1, total_f-total_i);
11.    }
12.    im_parallel[0] = imp;
13.    tp_est[0]= t_est;
14.    break;

```

Esta es la sección de nuestro programa interfaz que se encarga de realizar las ejecuciones de los procesos, a través de la función `system()`, y de determinar los valores de demoras en tiempo a través de la función `omp_get_wtime()`.

En nuestro programa principal utilizamos esta forma para brindarle al usuario la opción que le permita ejecutar un proceso, procedural o paralelo, y a la vez se llevan a cabo las mediciones de los tiempos por último con una opción que permite visualizar los tiempos se puede observar con precisión de cuanto han sido las demoras por muestra de ejecución de procesos en promedio y muchas más estadísticas de utilidad como el speedup.

Ventajas:

- Fácil de utilizar. Solo se debe llamar una función al inicio y final y realizar la diferencia de tiempos.
- Los tiempos medidos son muy precisos. Se adapta a todo tipo de mediciones ya sean en segundo o milisegundos.
- Flexibilidad en configuración de cantidad de muestras a ejecutar y cantidad de hilos a utilizar por procesos.
- Modificación de parámetros en tiempo de ejecución. Por lo cual no hace falta tener que adaptar el código ante un nuevo cambio. Esto proporciona mejor automatización.
- Todo queda en un mismo lugar y sobre una misma forma de programación.
- No hay susceptibilidad de errores en la programación ya que se proporciona con un gran compilador y flags de complemento para la detección de errores.
- Entorno robusto. Por la gran cantidad de alos del lenguaje C y la librería openMP se tiene herramientas de confianza y calidad por lo que garantizan buenos resultados.
- Grandes cantidades de información en la nube.

Desventajas:

- Se debe tener conocimientos en programación del lenguaje C.
- No tan rápido de utilizar. Se requerirá armar un código que se adapte al uso deseado.



UNIVERSIDAD
NACIONAL DE
CÓRDOBA

Facultad de Ciencias Exactas, Físicas y Naturales

Cátedra de Sistemas Operativos II

TRABAJO PRÁCTICO N° II

Salas Canalicchio, Sergio Enrique

Córdoba, 13 de Mayo de 2018

Índice

1. Introducción

- 1.1. Propósito
- 1.2. Ámbito del sistema
- 1.3. Definiciones, acrónimos y abreviaturas
- 1.4. Referencias
- 1.5. Descripción General del documentado

2. Descripción General

- 2.1. Perspectiva del producto
- 2.2. Funciones del producto
- 2.3. Características de los usuarios
- 2.4. Restricciones
- 2.5. Suposiciones y dependencias
- 2.6. Requisitos futuros

3. Requisitos específicos

- 3.1. Interfaces externas
- 3.2. Funciones
- 3.3. Requisitos de rendimiento
- 3.4. Restricciones de diseño
- 3.5. Atributos del sistema
- 3.6. Otros requisitos

4. Implementación y resultados

- 4.1. Implementación de aplicación procedural y paralela
- 4.2. Resultados

5. Conclusiones

6. Anexos

1. Introducción

1.1. Propósito

El propósito del presente trabajo consiste en desarrollar una aplicación que funcione como un procesador Doppler, es decir tomar las recepciones de los distintos canales de un radar meteorológico, brindadas a través de un archivo binario, y procesar las misma para obtener como resultado deseado la autocorrelación sobre cada canal que hay por cada componente de rango. El documento está dirigido a personas con conocimiento en programación paralela y especializado en este ámbito, para que pueda comprender a través de la ejecución de diferentes muestras sobre el programa las ventajas de rendimiento que se obtienen con la paralelización de procesos.

1.2. Ámbito del sistema

La aplicación desarrollada en el presente documento, está diseñada para los sistemas operativos Linux, cualquiera de su versiones podrá ejecutar dicho programa a través de la compilación del makefile correspondiente a la misma.

Esta aplicación procesa un archivo binario como entrada y obtiene un archivo binario de salida con resultados de autocorrelación discreta por componente de rango de un radar que es el dato requerido a través del procesamiento de la información. A su vez se podrá visualizar los tiempos de demora que son necesarios para el caso de lograr llevar a cabo la aplicación sin paralelización de procesos y explotando el paralelismo con la paralelización de procesos.

El objetivo es que los tiempos obtenidos explotando la paralelización, sean mucho mejor e impacten de manera significativa al comparar con la versión de la aplicación procedural. Para esto siempre se buscará amoldar los algoritmos para que consuman el menor tiempo posible.

1.3. Definiciones, acrónimos y abreviaturas

Procesador Doppler: Procesador usado en procesamiento de señales "pulse-doppler"

Pulse-Doppler signal procesing: Es una estrategia de mejora de rendimiento de un radar que permite la detección de objetos pequeños que se mueven a gran velocidad en cercanía con objetos más grandes también en movimiento.

Acimut: posición en grados sobre el eje horizontal.

Elevación: posición en grados sobre el eje vertical.

Fase: componente real de la señal electromagnética compleja.

Cuadratura: componente imaginaria de la señal electromagnética compleja.

Rango de resolución: rango en kilómetros de avance de cobertura mínimo, el mismo determina la cantidad de componentes de rango total de acuerdo al tamaño del rango de cobertura total.

Rango máximo de cobertura: cantidad de kilómetros máxima de cobertura alcanzada por el radar.

Gate: cada componente de rango. Por ejemplo 0.5km, 1 km, 1.5 km, y así sucesivamente hasta alcanzar el rango de cobertura máximo.

Librería OpenMP: Es una API para la programación multiproceso de memoria compartida.

API: interfaz de programación de aplicaciones, conjunto de funciones y subrutinas que ofrece una biblioteca.

Biblioteca: Conjunto de implementaciones funcionales.

Memoria compartida: Se refiere a un conjunto de procesadores que comparten memoria principal, como en el caso de las computadoras multinúcleos y a diferencia de un clúster.

Hardware: Referido a productos tangibles donde corre un software, computadora, clúster, etc.

Software: Todo lo referente a programas, código.

Conversor A/D: Conversor analógico digital, convierte una señal analógica en un valor digital que consta de palabras binarias

Radar: Acrónimo de RAdio Detection And Ranging, emite ondas electromagnética para la medición de distancias y velocidades entre otras cosas.

Procedural: Referido a un programa que posee solo un hilo de ejecución, todas las tareas se realizan secuencialmente.

Paralelo: Las tareas se pueden ejecutar en paralelo siempre y cuando no haya una dependencia fuerte entre una y otra.

Pulsos: Emisión de energía electromagnética de alta intensidad y corta duración

Clúster: Conjunto de computadoras autónomas que operan juntas para proporcionar mayor potencia que cada computadora individualmente.

Gate: Es una medida de la resolución de un radar

Autocorrelación: Herramienta estadística para el procesamiento de señales

Archivo binario: Archivo que posee palabras binarias, (valor 0 ó 1), no puede ser interpretado como un archivo de texto, sino que debe ser abierto con un programa dedicado.

Estructura: Se refiere a un bloque de código que puede representar como un nuevo tipo de datos que es definido fuera de los tipos de un lenguaje.

C90: Es un estándar del lenguaje C, existe además C89, C95, C99, y C11. La idea de estos estándares es mejorar la portabilidad de código.

UML: Unified Modeling Language, lenguaje de modelado unificado, es un estándar para el diseño de productos, que cuenta con diferentes tipos de diagramas.

Profiling: Análisis dinámico de programas

1.4. Referencias

[1] IEEE 830. (22 de Octubre, 2008). Especificación de requisitos según el estándar IEEE 830. [URL](#).

[2] Williams Stallings – Sistemas Operativos 5ta Edición.

1.5. Descripción General del documentado

Este documento se compone de seis secciones principales, siendo la primera la introducción, donde se explica brevemente el fin del proyecto como así también sus requerimientos de forma muy general. En la segunda sección la descripción general del sistema con el fin de conocer las principales funciones que debe ser capaz de realizar, conocer a quién va dirigido, además de restricciones y supuestos que puedan afectar el desarrollo del mismo. En la sección tres se definen de manera detallada los requerimientos del sistema a desarrollar, los que definen el comportamiento del sistema como así también otros requerimientos que puedan ser deseados considerando el uso que el sistema va a tener. En la cuarta sección se discuten los resultados de implementación del diseño de la aplicación final donde se muestra como el sistema opera. Por último en la sección cinco se elabora una breve conclusión acerca de la experiencia en la elaboración del proyecto. La sección seis, de Anexos, contiene información asociada a cierto tipos de temas mencionados en el trabajo que se extienden para mayor detalle.

2. Descripción General

2.1. Perspectiva del producto

La aplicación tiene un objetivo muy claro. Se trata de explotar el mayor paralelismo posible para visualizar las ventajas que proporciona el mismo. Con estos algoritmos se puede comprender que utilizándolos en otros sistemas o aplicaciones de mayor tamaño se puede buscar mejorar su rendimiento logrando explotar el paralelismo. La aplicación es utilizada como una interfaz que existe entre un radar meteorológico y los resultados de datos necesarios que se requieren para visualizarlos en pantalla, ya sea de una computadora, un smartphone o una página web, pero para todos los casos primero es necesario de esta aplicación y mejor será si explota de manera eficiente el paralelismo ya que los datos serán obtenidos directamente apenas sean creados sus archivos de recepción.

2.2. Funciones del producto

La aplicación nos permitirá las siguientes funciones:

- Funcionar como un procesador Doppler, capaz de tomar como entrada un archivo binario de pulsos el cual será procesado, devolviendo un archivo binario de salida como resultado y en el menor tiempo posible.
- La aplicación permitirá procesar la información a través de diferentes algoritmos con el fin de visualizar las mejoras cuando se usan algoritmos que explotan el paralelismo.
- La aplicación mostrará un menú de opciones con las distintas funcionalidades que se podrán realizar sobre los diferentes procesos vinculados a un algoritmo utilizado para llevar a cabo la resolución requerida. Las opciones serán para amoldar los parámetros de operación, como número de iteraciones sobre el proceso a ejecutar, el número de hilos a utilizar sobre el algoritmo de un proceso y el tipo de proceso a ejecutar referido a un tipo de algoritmo procedural o paralelo. Como también para visualizar todos los datos obtenidos, por ejemplo lectura e interpretación del archivo binario resultante.
- Se podrá ejecutar un proceso que utiliza en algoritmo procedural que obtendrá los resultados sobre el archivo binario deseado mostrando los tiempos de demora.
- Se podrá ejecutar un proceso que utiliza en algoritmo paralelo que obtendrá los resultados sobre el archivo binario deseado mostrando los tiempos de demora.
- La aplicación además permitirá poder ejecutarse de manera iterativa. Es decir si se desea ejecutar un mismo proceso 30 veces, esto se podrá indicar a través de un menú de opciones y de esa manera se podrá también saber el promedio de tiempo de demora sobre las 30 muestras.
- Para los procesos que utilizan algoritmos paralelos se podrá definir el número de hilos que se desean utilizar paralelamente en su ejecución.
- Toda información de estadísticas será accedida a través de una opción para visualizar los resultados estadísticos.
- También toda la información estadística podrá eliminarse para tomar nuevos análisis de muestras.

2.3. Características de los usuarios

La aplicación está destinada a usuarios especializados y con conocimientos sobre la programación paralela para que puedan comprender las mejoras significativas que se pueden obtener al explotar los algoritmos con la paralelización de procesos. Los usuarios deberán tener conocimientos sobre conceptos relacionados a los radares meteorológicos para poder interpretar los resultados obtenidos.

2.4. Restricciones

Como restricción de lenguaje de programación se tiene que la aplicación debe ser en su totalidad elaborada bajo lenguaje C. Se explota el paralelismo con la API OpenMP. Además como se mencionó el entorno donde corre la aplicación debe ser cualquier distribución basada en GNU/Linux.

En cuanto a hardware se requieren más de un procesador lógico, es decir puede ser un equipo con un procesador físico pero con capacidad de hyperthreading, con lo cual brinda la capacidad de ejecutar más de un proceso paralelamente. Cualquier computadora que pueda correr un GNU/Linux con las características anteriores es apta.

Para visualizar y concluir adecuadamente sobre las mejoras significativas que trae el paralelismo de procesos se deberá contar con un equipo con una cantidad significativa de núcleos físicos o con la capacidad de núcleos lógicos para poder visualizar la diferencia y ganancias de tiempo que existe entre un algoritmo procedural (un núcleo lógico/físico) y un algoritmo paralelo. Un equipo ideal para esta aplicación sería contar con un cluster o por lo menos con un acceso a estos para ejecutar la aplicación sobre el mismo.

2.5. Suposiciones y dependencias

La aplicación se diseña con el fin de ser ejecutada en cualquier sistema operativo Linux, por lo cual, esta es dependiente de estos sistemas operativos. En caso de cambiar de sistema operativo deberá revisarse si se adaptan todas las condiciones para el correcto funcionamiento de la aplicación.

Se supone que la cantidad de gates que cubren el rango máximo son 500. Esto es así debido a que el rango de cobertura máximo es de 250 km de radio, entonces si cada componente fuera cada un km se tendría 250 gates de cantidad, pero como en nuestro caso se tiene una resolución 0.5 km por componente en las 250 km entran 500 componentes. Es decir que cada km se tienen 2 componentes de 0,5 km cada una por lo cual en las 250 km da un total de 500 componentes.

Cargando...

La cantidad de pulsos contenidos en el archivo binario *pulsos.iq* es una constante estática, ya que por grado de acimut se tienen 100 pulsos emitidos y como también se proporciona el dato de que los pulsos emitidos desde los 83° a los 90° en acimut, se determina que hay información de 8 grados diferentes con lo que se concluye que los pulsos del archivo son 800.

- Grados de acimut considerados: 8.
- Cantidad de pulsos por grado de acimut: 100.
- Total de pulsos en archivo: $8 \times 100 = 800$.

El tipo de archivo que contiene la información a procesar por la aplicación es binario por lo cual si en un futuro se cambiará la estructura de los datos del archivo binario deberán ajustarse los cambios a la aplicación cuando analiza el archivo.

Las características del archivo del que depende la aplicación son las siguientes:

- Tipo de archivo: binario.
- Nombre de archivo: pulsos.iq
- Estructura de archivo binario:
 - validSamples (tipo unit16_t).
 - Para $i=0,1,2,\dots$, validSamples se tiene: $V_I[i]$ $V_Q[i]$ (tipo float).
 - Para $i=0,1,2,\dots$, validSamples se tiene: $H_I[i]$ $H_Q[i]$ (tipo float).

Las características del archivo de salida son las siguientes:

- Tipo de archivo: binario.
- Nombre de archivo: resultados_adpc
- Estructura de archivo binario:
 - Cantidad_de_datos (tipo int).
 - Para $i=0,1,2,\dots, 500$ se tiene: $R_V[i]$ (tipo double).
 - Para $i=0,1,2,\dots, 500$ se tiene: $R_H[i]$ (tipo double).

En caso de que cambie el nombre, el tipo o la estructura del archivo de entrada o de salida deberán revisarse los requerimientos asociados a la lectura y/o escritura de los mismos.

2.6. Requisitos futuros

Como requerimientos futuros para lograr mejorar más la implementación y adecuar cada vez mejor la aplicación implementada se proponen los siguiente requerimientos:

- Mejorar el algoritmo de lectura de archivos. Sabemos que los mayores costos están dados por el nivel de jerarquía más bajo de la memoria, es por lo cual es importante mantener un correcto algoritmo sobre la lectura y escritura de los ficheros para de esta manera mantener optimizado los tiempos de acceso a los espacios de memoria de menor jerarquía.
- Adaptaciones a diferentes tipos de ficheros con tamaños variables, puede presentarse la situación de tener que tomar un mismo archivo pero sobre diferentes tamaños de datos, para esto se propone utilizar algoritmos que detectan estas lecturas de archivos de manera de determinar los tamaños de dimensiones necesarios para adecuar sus recursos en tiempo de ejecución.
- La interpretación de archivos binarios con diferentes nombres podría ser otro requisito futuro, lo que permitirá seleccionar que archivo se desea utilizar para de esta manera procesar la información en archivos diferentes.

3. Requisitos específicos

3.1. Interfaces externas

Interfaz de software:

La interfaz de software, es por selección de opciones a través de un menú de opciones proporcionado por la ejecución de la aplicación. Al ejecutar la aplicación se muestran las opciones y se deberá seleccionar la opción de acuerdo al número que se le corresponde.

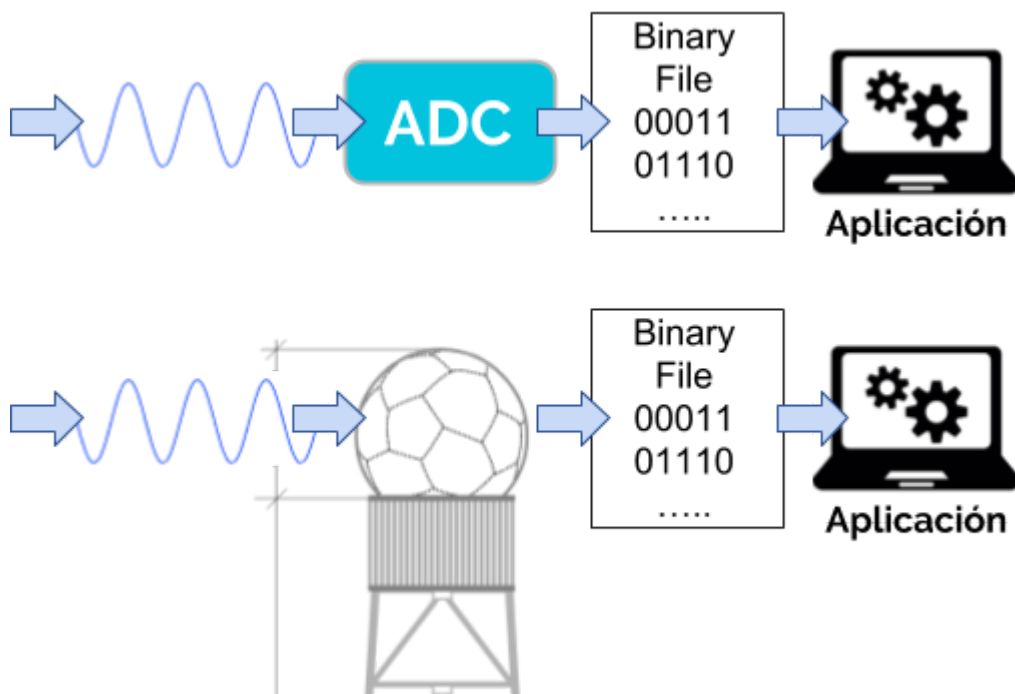
```
>>>_                MENU                _<<<

1. Cambiar numero de iteraciones por algoritmo.
2. Cambiar numero de hilos paralelos.
3. Ejecutar algoritmo procedural.
4. Ejecutar algoritmo procedural con solicitud de resultados.
5. Ejecutar algoritmo paralelo.
6. Ejecutar algoritmo paralelo con solicitud de resultados.
7. Ver estadísticas de algoritmo procedural.
8. Ver estadísticas de algoritmo paralelo.
9. Borrar estadísticas.
10. Mostrar resultados desde archivo binario resultados_adpc.
11. Salir.

Introduzca opcion (1-11): 7
```

Interfaz de hardware y comunicación:

No se tiene interfaz de hardware ya que existe otro dispositivo del sistema que conforma el radar encargado de obtener un archivo binario que es el que se utiliza como entrada de la aplicación del presente documento.



3.2. Funciones

Como esta aplicación es utilizada por personal capacitado y especializado en el manejo de software avanzado (como único tipo de usuario), se detallan las funciones de la aplicación de acuerdo a los objetivos que se deben llevar a cabo en las mismas para que funcione todo adecuadamente.

La aplicación diseñada cuenta con las siguientes funciones, que se seccionan de acuerdo a los tipos de procesos que serán ejecutados. En primer lugar se describen las funciones de los procesos que usaran el algoritmo paralelo o procedural y por último se describirán las funciones que deberá cumplir la interfaz para ejecutar de manera adecuada estos procesos.

Funciones de los procesos ejecutados por la aplicación principal

Los procesos que serán ejecutados por la aplicación principal serán los que se asocian con los programas que contendrán los algoritmos de resolución de la autocorrelación y que podrán ser procedural o paralelo. En ambos casos se deberán cumplir con las siguientes funciones:

- El programa deberá ser capaz de leer un archivo binario e interpretar la información para luego poder procesarla de acuerdo a las necesidades. Se tiene en cuenta que este archivo tiene una estructura proporcionada por la cátedra.
- Por cada canal (vertical y horizontal), se deberá obtener el valor de cada gate de acuerdo a la media aritmética de muestras que existan para el mismo. El resultado es una matriz gate-pulso que contendrá por cada pulso (columnas de matriz) el valor de gate asociado a cada componente de rango (filas de matriz). Para lograr esto se creó la función *gate_por_canal()* que se encarga de construir la estructura de datos gate-pulso. Para esta función se presentan distintas situaciones que dependen de la cantidad de muestras (samples) que existan por pulso, según las indicaciones del archivo binario. Esto es un problema ya que si el número de muestras se debe repartir para todas las componentes de rango que conforma a cada gate. El problema surge del hecho de que repartir puede variar de acuerdo a la cantidad muestras y se presentan tres casos diferentes:
 - El caso ideal es que el número de repartida para cada componente asociada a un gate sea igual para cada componente de rango, es decir sea un entero, por ejemplo sea de 10 en todos los casos igual.
 - Otro caso es que no sea entero y se debe redondear para abajo, por ejemplo si el número a repartir por cada componente de rango sea de 10.1, se redondea a 10 pero surge el conflicto de que se introduce un pequeño error a medida que se avanza en los promedios por componentes asociadas a cada gate. La solución es corregir el error a medida que su acumulación empieza a ser significativa.
 - Y el último caso es el que se presenta opuestamente al anterior, es decir cuando se debe redondear hacia arriba el número a repartir por componente de rango asociada a un gate, por ejemplo si es de 10.9, hay menos error si se redondea a 11 pero a medida que se avanza por promedios de componentes de rangos diferentes se va teniendo un pequeño error que deberá solucionarse corrigiéndose cuando este empieza a ser significativo como el caso anterior.

Para resolver todos los problemas anteriores se utiliza en la función *gate_por_canal()* una codificación capaz de poder seleccionar diferentes algoritmo de corrección de error de

acuerdo al caso que se presente para de esta manera tener un cálculo de los resultados de la manera más adecuada.

- Por último el programa deberá ser capaz de resolver una autocorrelación discreta por canal para de esta manera obtener la información requerida. Para esto se utiliza la función *autocorrelacion()* que se encarga de obtener la autocorrelación discreta para cada uno de los canales a través de un algoritmo encargado de dicha tarea.
- Todos los resultados de la autocorrelación por canal deberá ser almacenada en un archivo binario de acuerdo a una estructura indicada en la presente documentación. El archivo que contendrá los resultados se denomina *resultados_adpc*, haciendo referencia a los resultados de la autocorrelación discreta por pulsos de cada canal.

Funciones de la aplicación de interfaz que utilizará el usuario final

La aplicación principal involucra a los programas anteriores que cumplen con las requerimientos descritos asociados a los casos del algoritmo procedural y el algoritmo paralelo. Esta aplicación es una interfaz de usuario final automatizada y con flexibilidad de configuración de parámetros para poder realizar las muestras y determinar las estadísticas de los resultados como el análisis de los archivos resultantes. La aplicación contará con las siguientes funciones:

- La aplicación deberá permitir configurar el número de iteraciones que se deseen llevar a cabo al ejecutar un programa procedural o un programa paralelo.
- La aplicación deberá permitir configurar el número de hilos a utilizar por el programa que hará uso de los mismo a través de su explotación de paralelismo.
- La aplicación nos permitirá, por medio de una opción, ejecutar el programa procedural.
- La aplicación nos permitirá, por medio de una opción, ejecutar el programa procedural con solicitud de los resultados que obtenga en la autocorrelación.
- La aplicación nos permitirá, por medio de una opción, ejecutar el programa paralelo.
- La aplicación nos permitirá, por medio de una opción, ejecutar el programa paralelo con solicitud de los resultados que obtenga en la autocorrelación.
- La aplicación nos permitirá ver las estadísticas del programa procedural.
- La aplicación nos permitirá ver las estadísticas del programa paralelo.
- La aplicación nos permitirá borrar las estadísticas almacenadas.
- La aplicación nos permitirá abrir el archivo binario de resultados.
- La aplicación nos permitirá poder finalizar con la misma.

3.3. Requisitos de rendimiento

La aplicación deberá soportar un número significativos de muestras para poder determinar promedios de tiempo. En este caso con 30 o más de 30 son un conjunto de muestras suficientes para la ejecución de los diferentes algoritmos que se podrán ejecutar. Todos los resultados obtenidos por los algoritmos serán almacenados en un archivo binario de salida al cual denominamos *resultados_adpc*, haciendo referencia a los resultados de auto correlación discreta de cada pulso por canal que contiene el radar.

3.4. Restricciones de diseño

Las restricciones de diseño son que el mismo deberá ser llevado a cabo mediante el lenguaje de programación C. Además se deberá contar con un equipo con un sistema operativo Linux y que cuente con más de un núcleo lógico para poder explotar el paralelismo. Para explotar el paralelismo se utiliza la API OpenMP.

3.5. Atributos del sistema

La aplicación busca tener el mejor rendimiento posible. Es por lo cual no se trata de una aplicación que requiera seguridad en cuanto a la protección de datos, por lo que podrá ser utilizado por cualquier usuario sin necesidad de contener una clave o usuario habilitado.

En base a la busca del mejor rendimiento se determinara que este será mejor cuando los resultados de tiempos obtenidos por el programa paralelo muestran mejoras de rendimiento en comparación a los resultados de tiempo obtenidos por el programa procedural.

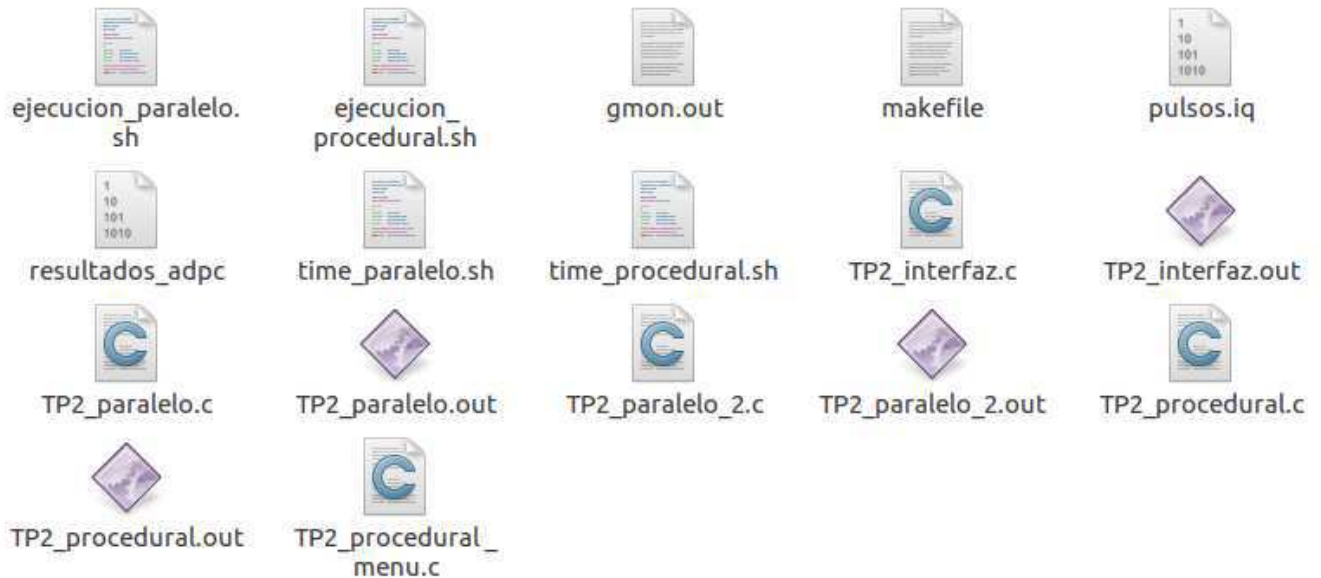
3.6. Otros requisitos

La aplicación que será utilizada un usuario se diseña de tal forma que se pueda unificar la mayor cantidad de funcionalidades en un solo lugar, por este motivo se detectaron un par de requisitos extras que contribuyen al mejor funcionamiento de la aplicación. Estos son:

- Proporcionar un menú de opciones para poder realizar de manera adecuada la configuración de los diferentes parámetros que afectan al sistema con el fin de adecuar todas las necesidades a través de opciones.
- Se deberá contar con una opción que permita ejecutar el algoritmo que determina los resultados sobre un archivo binario.
- Se deberá contar con una opción que permita poder configurar el número de iteraciones que se desean realizar sobre una misma aplicación.
- Se deberá contar con una opción que permita visualizar los valores de tiempos promedios de acuerdo a las ejecuciones previas de los diferentes algoritmos,
- Se deberá contar con una opción que permita poder leer e interpretar los valores contenidos en el archivo binario resultante.

4. Implementación y resultados

4.1. Implementación de aplicación procedural y paralela



Mediante el archivo makefile se logra compilar el programa. Ejecutando el siguiente comando:

```
make
```

Se obtiene la compilación de los programas TP2_procedural.c, TP2_paralelo_2.c, TP2_interfaz.c Devolviendo los programas ejecutables TP2_procedural.out, TP2_paralelo_2.out, TP2_interfaz.out. Luego ejecutando

```
./TP2_interfaz.out
```

Se iniciará la ejecución del proceso ejecutable TP2_interfaz.out.

Este proceso principal será el encargado de llamar al resto de los procesos, en este caso se encarga de llamar al proceso TP2_procedural.out y al proceso TP2_paralelo_2.out cuando se lo solicite un usuario.

Tanto el proceso TP2_procedural.out como también el proceso TP2_paralelo_2.out utilizan el archivo *pulsos.iq* para poder tomar como entrada todos los pulsos proporcionados por el radar meteorológico. Luego de procesar los datos se devuelve el archivo binario de salida resultados_adpc, que contiene todos los resultados obtenidos por el algoritmo procedural o paralelo según sea el caso.

4.2. Resultados

Durante el inicio de ejecución de la aplicación se obtiene la siguiente pantalla:

```
$ ./TP2_interfaz.out

>>>_          MENU          _<<<

1. Cambiar numero de iteraciones por algoritmo.
2. Cambiar numero de hilos paralelos.
3. Ejecutar algoritmo procedural.
4. Ejecutar algoritmo procedural con solicitud de resultados.
5. Ejecutar algoritmo paralelo.
6. Ejecutar algoritmo paralelo con solicitud de resultados.
7. Ver estadisticas de algoritmo procedural.
8. Ver estadisticas de algoritmo paralelo.
9. Borrar estadisticas.
10. Mostrar resultados desde archivo binario resultados_adpc.
11. Salir.

Introduzca opcion (1-11): █
```

Para ejecutar el proceso con el algoritmo procedural, como vemos en la captura anterior, se deberá elegir la opción 3. Luego de la ejecución se devuelven los resultados como muestra la siguiente captura:

```
>>>_          MENU          _<<<

1. Cambiar numero de iteraciones por algoritmo.
2. Cambiar numero de hilos paralelos.
3. Ejecutar algoritmo procedural.
4. Ejecutar algoritmo procedural con solicitud de resultados.
5. Ejecutar algoritmo paralelo.
6. Ejecutar algoritmo paralelo con solicitud de resultados.
7. Ver estadisticas de algoritmo procedural.
8. Ver estadisticas de algoritmo paralelo.
9. Borrar estadisticas.
10. Mostrar resultados desde archivo binario resultados_adpc.
11. Salir.

Introduzca opcion (1-11): 3

    * El numero de pulsos que tiene el archivo pulsos.iq es: 800
    * Las componetes de rango asociadas a cada gate son: 500
    * Se utilizan estructuras de datos del tipo matriz[GATES][PULSO]
canal[500][800]

    * Tiempo autocorrelación: 0.00348
    * Tiempo requerido para calculos (calculo de GATES y autocorrela
on): 0.34245
    * Tiempo de demora total de iteracion 1: 0.35742

    * Archivo binario de salida con resultados resultados_adpc
```


La ejecución anterior se realiza con una sola iteración, para cambiar el número de muestras en la ejecución de los procesos se deberá elegir la opción 1. La misma pedirá que se ingrese el número deseado de iteraciones/muestras y quedará configurado hasta un próximo cambio.

```
>>>_          MENU          _<<<

1. Cambiar numero de iteraciones por algoritmo.
2. Cambiar numero de hilos paralelos.
3. Ejecutar algoritmo procedural.
4. Ejecutar algoritmo procedural con solicitud de resultados.
5. Ejecutar algoritmo paralelo.
6. Ejecutar algoritmo paralelo con solicitud de resultados.
7. Ver estadísticas de algoritmo procedural.
8. Ver estadísticas de algoritmo paralelo.
9. Borrar estadísticas.
10. Mostrar resultados desde archivo binario resultados_adpc.
11. Salir.

Introduzca opcion (1-11): 1

> Introduzca el numero de iteraciones deseado: 10
> Se establecieron 10 iteraciones.
```

Al ejecutar nuevamente con la opción 3 ahora se realizará con el número de veces ingresado en la opción anterior:

```
Introduzca opcion (1-11): 3

* El numero de pulsos que tiene el archivo pulsos.iq es: 800
* Las componetes de rango asociadas a cada gate son: 500
* Se utilizan estructuras de datos del tipo matriz[GATES][PULSO] = canal[500][800]

* Tiempo autocorrelación: 0.00352
* Tiempo requerido para calculos (calculo de GATES y autocorrelacion): 0.34748
* Tiempo de demora total de iteracion 1: 0.36426

* Tiempo autocorrelación: 0.00367
* Tiempo requerido para calculos (calculo de GATES y autocorrelacion): 0.30474
* Tiempo de demora total de iteracion 2: 0.31836

* Tiempo autocorrelación: 0.00362
* Tiempo requerido para calculos (calculo de GATES y autocorrelacion): 0.30601
* Tiempo de demora total de iteracion 3: 0.32031

* Tiempo autocorrelación: 0.00362
* Tiempo requerido para calculos (calculo de GATES y autocorrelacion): 0.27570
* Tiempo de demora total de iteracion 4: 0.28027

* Tiempo autocorrelación: 0.00350
* Tiempo requerido para calculos (calculo de GATES y autocorrelacion): 0.29499
* Tiempo de demora total de iteracion 5: 0.30469
```

Para cambiar el número de hilos que se requieren en la ejecución del proceso con el algoritmo paralelo se deberá seleccionar la opción 2. Con la misma se configura el número de hilos para la paralización el proceso que explota el paralelismo y permanecerá dicho valor hasta una nueva modificación:

```
>>>_          MENU          _<<<

1. Cambiar numero de iteraciones por algoritmo.
2. Cambiar numero de hilos paralelos.
3. Ejecutar algoritmo procedural.
4. Ejecutar algoritmo procedural con solicitud de resultados.
5. Ejecutar algoritmo paralelo.
6. Ejecutar algoritmo paralelo con solicitud de resultados.
7. Ver estadisticas de algoritmo procedural.
8. Ver estadisticas de algoritmo paralelo.
9. Borrar estadisticas.
10. Mostrar resultados desde archivo binario resultados_adpc.
11. Salir.

Introduzca opcion (1-11): 2

> Introduzca el numero de hilos paralelos deseados: 2
  * Comando 1:./TP2_paralelo_2.out 2
  * Comando 2:./TP2_paralelo_2.out 2 1
  * hilos: 2
```

Si se desea ejecutar el algoritmo paralelo se deberá elegir la opción 5. La salida resultante se muestra en la siguiente captura (solo para una iteración):

```
>>>_          MENU          _<<<

1. Cambiar numero de iteraciones por algoritmo.
2. Cambiar numero de hilos paralelos.
3. Ejecutar algoritmo procedural.
4. Ejecutar algoritmo procedural con solicitud de resultados.
5. Ejecutar algoritmo paralelo.
6. Ejecutar algoritmo paralelo con solicitud de resultados.
7. Ver estadisticas de algoritmo procedural.
8. Ver estadisticas de algoritmo paralelo.
9. Borrar estadisticas.
10. Mostrar resultados desde archivo binario resultados_adpc.
11. Salir.

Introduzca opcion (1-11): 5

  * El numero de pulsos que tiene el archivo pulsos.iq es: 800
  * Las componetes de rango asociadas a cada gate son: 500
  * Se utilizan estructuras de datos del tipo matriz[GATES][PULSO]

  * Tiempo autocorrelación: 0.00204
  * Tiempo requerido para calculos (calculo de GATES y autocorrel.
tiempo de demora total de iteracion 1: 0.29102

archivo binario de salida con resultados resultados_adpc
```

Para mostrar los resultados del archivo binario resultante se deberá elegir la opción 10. La misma interpreta la estructura que mantiene el archivo binario de salida.

```
>>>_                               MENU                               _<<<

1. Cambiar numero de iteraciones por algoritmo.
2. Cambiar numero de hilos paralelos.
3. Ejecutar algoritmo procedural.
4. Ejecutar algoritmo procedural con solicitud de resultados.
5. Ejecutar algoritmo paralelo.
6. Ejecutar algoritmo paralelo con solicitud de resultados.
7. Ver estadisticas de algoritmo procedural.
8. Ver estadisticas de algoritmo paralelo.
9. Borrar estadisticas.
10. Mostrar resultados desde archivo binario resultados_adpc.
11. Salir.

Introduzca opcion (1-11): 10
cantidad 500
R_V[0] = 0.0608889020, R_H[0] = 0.0209738172
R_V[1] = 0.0271063099, R_H[1] = 0.0089635151
R_V[2] = 0.0203369489, R_H[2] = 0.0069478597
R_V[3] = 0.0182197291, R_H[3] = 0.0064507396
R_V[4] = 0.0152895394, R_H[4] = 0.0055833892
R_V[5] = 0.0123240922, R_H[5] = 0.0044057295
R_V[6] = 0.0108640468, R_H[6] = 0.0037330334
R_V[7] = 0.0081694692, R_H[7] = 0.0028925044
R_V[8] = 0.0072433521, R_H[8] = 0.0023225721
R_V[9] = 0.0068533877, R_H[9] = 0.0022683151
R_V[10] = 0.0062679001, R_H[10] = 0.0020786164
R_V[11] = 0.0043748245, R_H[11] = 0.0013370025
```

Podemos notar cómo e interpreta el archivo binario resultante de acuerdo a la cantidad de valores que contiene y con cada uno de los datos sobre los diferentes canales obtenidos para cada caso. Por último para validar si los resultados son correctos se puede verificar eligiendo la opción 4 o 6 la cual nos permite poder comparar los valores que hay en el archivos de salida escrito por los procesos, con los valores contenidos en las estructuras de datos que mantienen los procesos en tiempo de ejecución. Seguramente se notará que los valores son los mismos (caso en el cual no hay errores en la escritura de datos), o en caso de errores serán distintos por lo que se deberá revisar el código.

Resultados de estadísticas

Sobre el cluster de la facultad se utilizó el nodo pulqui.

Se tomaron 30 muestras para la ejecución de cada uno de los procesos paralelo y procedural y además para el caso del algoritmo paralelo se varió número de hilos de manera incremental.

Primero ejecutamos el proceso con el algoritmo procedural para poder comparar los resultados en base a este.


```

>>>_ ESTADISTICAS DE ALGORITMO _<<<
>>>_ ALGORITMO PROCEDURAL _<<<

```

Cores	Tiempo	Muestras
1	0.1417	30

Luego realizamos las muestras sobre el proceso paralelo con el número de hilos incrementando hasta observar que no escala el algoritmo paralelo. En promedio el tiempo de demora por muestras es el siguiente:

```

Introduzca opcion (1-11): 8
>>>_ ESTADISTICAS DE ALGORITMOS _<<<
>>>_ ALGORITMO PARALELO _<<<

```

Cores	Tiempo	SpeedUp	Porcentaje	Muestras
1	0.1417	1.0000	0%	30
2	0.1417	1.0000	0%	30
4	0.1083	1.3077	31%	30
8	0.0875	1.6190	62%	30
16	0.0958	1.4783	48%	30
32	0.3667	0.3864	-61%	30

Podemos notar que para el caso de 16 hilos y para 32 empieza a descender el escalado del algoritmo paralelo.

Para comparar resultados con otros de los nodos de la facultad se utilizó también el nodo franky. Los resultados son los siguientes:

```

Introduzca opcion (1-11): 7

>>>_          ESTADISTICAS DE ALGORITMO          _<<<
-----
>>>_          ALGORITMO PROCEDURAL          _<<<
-----
|  Cores  |  Tiempo  |  Muestras  |
-----
|    1    |  0.3500  |    30      |
-----

```

Luego de tomar la muestras para el proceso paralelo para las diferentes cantidades de hilos se obtuvieron los siguientes resultados:

```

Introduzca opcion (1-11): 8

>>>_          ESTADISTICAS DE ALGORITMOS          _<<<
-----
>>>_          ALGORITMO PARALELO          _<<<
-----
|  Cores  |  Tiempo  |  SpeedUp  |  Porcentaje  |  Muestras  |
-----
|    1    |  0.3500  |  1.0000   |    0%        |    30      |
-----
|    2    |  0.3500  |  1.0000   |    0%        |    30      |
-----
|    4    |  0.2958  |  1.1831   |    18%       |    30      |
-----
|    8    |  0.2500  |  1.4000   |    40%       |    30      |
-----
|   16    |  0.2292  |  1.5273   |    53%       |    30      |
-----
|   32    |  0.2375  |  1.4737   |    47%       |    30      |
-----
|   48    |  1.1833  |  0.2958   |   -70%      |    30      |
-----

```

En este nodo se nota que los resultados de tiempo son mayores a comparación del anterior. Pero en cuanto a la escalabilidad se puede decir que en este caso recién deja de escalar cuando se utilizan 32 hilos para el algoritmo.

Por último sobre el nodo cabecera de la facultad, nodo rocky se tienen los siguientes resultados.

```
Introduzca opcion (1-11): 7
-----
>>>_          ESTADISTICAS DE ALGORITMO          _<<<
-----
>>>_          ALGORITMO PROCEDURAL          _<<<
-----
| Cores | Tiempo | Muestras |
-----
|   1   | 0.1833 |    30    |
-----
```

```
Introduzca opcion (1-11): 8
-----
>>>_          ESTADISTICAS DE ALGORITMOS          _<<<
-----
>>>_          ALGORITMO PARALELO          _<<<
-----
| Cores | Tiempo | SpeedUp | Porcentaje | Muestras |
-----
|   1   | 0.1833 | 1.0000 |    0%      |    30    |
-----
|   2   | 0.1917 | 0.9565 |   -4%      |    30    |
-----
|   4   | 0.1708 | 1.0732 |    7%      |    30    |
-----
|   8   | 0.1375 | 1.3333 |   33%      |    30    |
-----
|  16   | 0.2625 | 0.6984 |  -30%      |    30    |
-----
```

Se observa que el algoritmo paralelo deja de escalar cuando se utilizan 16 hilos.

5. Conclusiones

El presente trabajo nos deja en claro cómo podemos paralelizar distintos tipos de problemas como también cuales son las dificultades que existen al paralelizar sobre los mismos. En este caso se explota la paralelización haciendo uso de OpenMP que es una interfaz de programación de aplicaciones (API) para poder manipular explícitamente el multiprocesamiento y el paralelismo de memoria compartida. Con esta API pudimos conocer básicamente cuáles son algunas de las directivas para interactuar en con las aplicaciones que se quieren convertir de secuencial a paralelo. Conocimos también las cláusulas que brinda la API necesarias para indicar cómo debe manipularse las variables a las que se asociará cada hilos. Sobre el manejo de los for conocimos que por medio los `schedule` podemos indicarle cual debe ser la manipulación sobre las iteraciones que recibirá cada uno de los hilos. Además la API nos brinda funciones para poder manejar los hilos tanto como qué cantidad utilizar como también controlar el tiempo de ejecución o capturar información de `get` como por ejemplo cuántos hilos existe como máximo en un equipo entre muchas otras funcionalidades.

6. Anexos

6.1. Mediciones de tiempo en ejecución procesos

Para la medición de los tiempos en la ejecución de los procesos se realizó una investigación acerca de diferentes herramientas que puedan ser útiles para dicha tarea dentro de las cuales nos centramos en tres casos, en donde concluimos que uno es el que más se adapta a nuestras necesidades. Las tres alternativas analizadas son :

- El comando time.
- El comando date.
- La función omp_get_wtime().

A continuación describimos cada una de estas.

Comando time

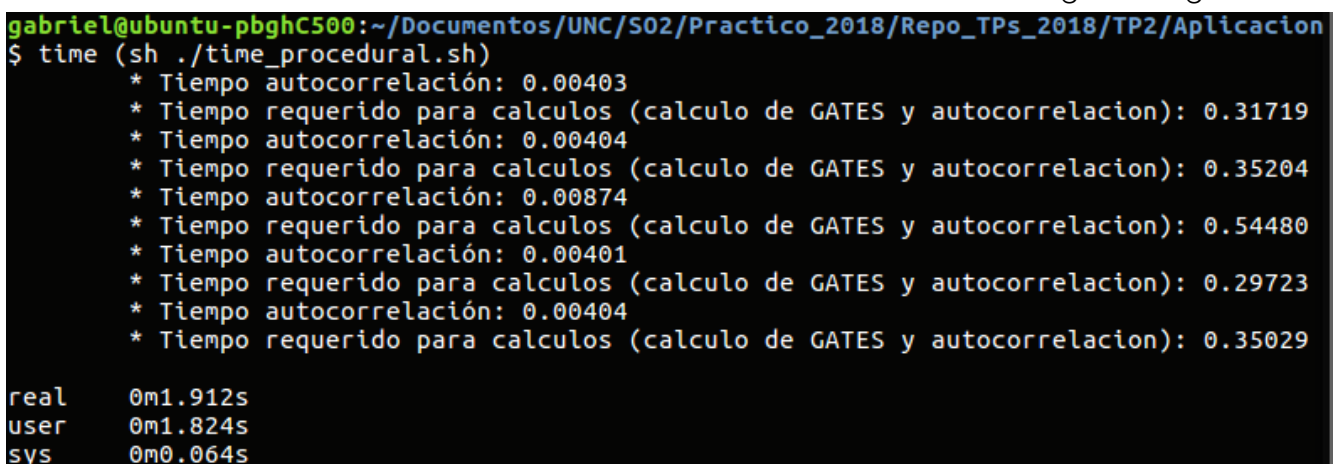
Esta herramienta consiste en ejecutar un proceso y medir el tiempo que demora su ejecución. Para esto simplemente debe ejecutar sobre la terminal el siguiente comando:

```
time proceso_a_ejecutar
```

Por ejemplo si combinamos esta herramienta con el uso de un script bash como el siguiente:

```
1. #!/bin/bash
2. # Variables
3. muestras=5
4. hilos=4
5.
6. for i in $(seq 1 $muestras); do
7.     ./TP2_procedural.out
8. done
```

Para nuestro caso una salida de esta herramientas es como se observa en la siguiente figura:



```
gabriel@ubuntu-pbghC500:~/Documentos/UNC/S02/Practico_2018/Repo_TPs_2018/TP2/Aplicacion
$ time (sh ./time_procedural.sh)
* Tiempo autocorrelación: 0.00403
* Tiempo requerido para calculos (calculo de GATES y autocorrelacion): 0.31719
* Tiempo autocorrelación: 0.00404
* Tiempo requerido para calculos (calculo de GATES y autocorrelacion): 0.35204
* Tiempo autocorrelación: 0.00874
* Tiempo requerido para calculos (calculo de GATES y autocorrelacion): 0.54480
* Tiempo autocorrelación: 0.00401
* Tiempo requerido para calculos (calculo de GATES y autocorrelacion): 0.29723
* Tiempo autocorrelación: 0.00404
* Tiempo requerido para calculos (calculo de GATES y autocorrelacion): 0.35029

real    0m1.912s
user    0m1.824s
sys     0m0.064s
```

Podemos notar en la herramienta que para nuestro ejemplo nuestro propio programa lanza mediciones de tiempos a las que este puede detallar, pero por fuera de nuestro programa, se requiere de la herramienta para determinar su tiempo de demora total.

Ventajas:

En cuanto a las ventajas que pudimos determinar con esta herramientas son las siguientes:

- Herramienta simple y rápida para utilizar.
- Previamente instalada en ubuntu 16.04 LTS. Por lo cual no es necesario descargar nada.
- Precisión de medición de tiempo (hasta milésimas de segundos).
- Flexibilidad para el manejo de cantidad de muestras y cantidad de hilos a utilizar.

Desventajas:

- Se debe combinar con el uso de scripts bash. Los cuales son muy susceptibles a errores en su programación.
- Se deberá poseer conocimientos en scripts bash.
- Modificación de parámetros estática., donde se debe actualizar los códigos de scripts bash cada vez que se desea modificar el número de hilos o cantidad de muestras a ejecutar.
- Y se debe procesar la salida obtenida para determinar el cálculo del tiempo por muestra que existe.
- Poco flexibilidad para adaptar a los programas a resultados de tiempos y estadísticas deseados.
- Escasa información de complemento en la nube.

Comando date

Esta herramienta puede ser utilizada para medir el tiempo ya que la misma nos proporciona el tiempo actual. Con esta función se la puede utilizar para tomar el tiempo actual todo en segundos, y luego procesar otro proceso para luego al final tomar el tiempo actual nuevamente y con estas dos muestras del tiempo si se realiza la diferencia se debería obtener la demora del proceso ejecutado.

Por ejemplo en combinación con los script bash se puede obtener el siguiente script para determinar el tiempo de demora de un proceso:

```

1. #!/bin/bash
2. # Variables
3. muestras=5
4. resta=0
5. start_time=$(date +%s)
6.
7. for i in $(seq 1 $muestras); do
8.     ./TP2_procedural.out
9. done
10.
11. finish_time=$(date +%s)
12. resta=$((finish_time - start_time))
13. tiempo_total=$(( calc $resta/$muestras))
14.
15. echo
16. echo "> El tiempo de demora total para las $muestras muestras es: $resta segundos."
17. echo "> El tiempo promedio por cada muestra es:" $tiempo_total "segundos."

```

Un ejemplo de su ejecución sería como sigue:

```

gabriel@ubuntu-pbghC500:~/Documentos/UNC/S02/Practico_2018/Repo_TPs_2018/TP2/Aplicacion
$ sh ./ejecucion_procedural.sh
* Tiempo autocorrelación: 0.01296
* Tiempo requerido para calculos (calculo de GATES y autocorrelacion): 0.94189
* Tiempo autocorrelación: 0.01302
* Tiempo requerido para calculos (calculo de GATES y autocorrelacion): 0.95414
* Tiempo autocorrelación: 0.01474
* Tiempo requerido para calculos (calculo de GATES y autocorrelacion): 0.97085
* Tiempo autocorrelación: 0.01352
* Tiempo requerido para calculos (calculo de GATES y autocorrelacion): 1.02723
* Tiempo autocorrelación: 0.01464
* Tiempo requerido para calculos (calculo de GATES y autocorrelacion): 0.97333
> El tiempo de demora total para las 5 muestras es: 5 segundos.
> El tiempo promedio por cada muestra es: 1 segundos.

```

Ventajas:

- Fácil y rápida de utilizar.
- Su salida es fácil de procesar. Ya que devuelve la cantidad de segundos (un dato entero).
- Si el tiempo es de varios segundos se puede obtener buena precisión.
- Flexible para la configuración de la cantidad de muestras y cantidad de hilos a utilizar por proceso.

Desventajas:

- Será necesario instalar `apcalc` (`sudo apt install apcalc`). El programa es necesario para procesar los datos más fácilmente con operaciones aritméticas.
- Se debe combinar con el uso de scripts `bash`. Los cuales son muy susceptibles a errores en su programación.
- Se deberá poseer conocimientos en scripts `bash`.
- Modificación de parámetros estática, donde se debe actualizar los códigos de scripts `bash` cada vez que se desea modificar el número de hilos o cantidad de muestras a ejecutar.
- Si se quiere medir tiempos menores a 1 segundo no es una herramienta recomendada.
- Poca flexibilidad para adaptar a los programas a resultados de tiempos y estadísticas deseados.
- Escasa información de complemento en la nube.

Función `omp_get_wtime()`

Esta herramienta a diferencia de las anteriores, no es un comando es por lo cual no se vincula con los scripts `bash`. Se trata de una de las funciones que brinda la biblioteca de `openMP` para medir el tiempo con precisión en bloques de código de nuestros programas. Una de las formas de usar esta herramienta es crear un programa de C principal encargado de ejecutar otros programas independientes y medir el tiempo de su ejecución por fuera. Esta es la alternativa que más se adapta a lo que buscamos porque sobre la misma podemos tener todo en un mismo lugar y sacar estadísticas con los datos obtenidos.

Un ejemplo del uso de esta herramienta es como se nota en la siguiente sección de código:

```

1. case 3: /* Ejecutar algoritmo procedural. */
2.     for(i = 0; i < iteraciones; i++)
3.     {
4.         total_i=omp_get_wtime();
5.         /* Ejecucion de proceso con algoritmo procedural*/

```



```

6.         system(comando_procedural_modol);
7.         total_f=omp_get_wtime();
8.         t_est = t_est + (total_f-total_i);
9.         imp++;
10.        printf("\t* Tiempo de demora total de iteracion %d: %.5f\n\n",i+1, total_f-total_i);
11.    }
12.    im_parallel[0] = imp;
13.    tp_est[0]= t_est;
14.    break;

```

Esta es la sección de nuestro programa interfaz que se encarga de realizar las ejecuciones de los procesos, a través de la función `system()`, y de determinar los valores de demoras en tiempo a través de la función `omp_get_wtime()`.

En nuestro programa principal utilizamos esta forma para brindarle al usuario la opción que le permita ejecutar un proceso, procedural o paralelo, y a la vez se llevan a cabo las mediciones de los tiempos por último con una opción que permite visualizar los tiempos se puede observar con precisión de cuanto han sido las demoras por muestra de ejecución de procesos en promedio y muchas más estadísticas de utilidad como el speedup.

Ventajas:

- Fácil de utilizar. Solo se debe llamar una función al inicio y final y realizar la diferencia de tiempos.
- Los tiempos medidos son muy precisos. Se adapta a todo tipo de mediciones ya sean en segundo o milisegundos.
- Flexibilidad en configuración de cantidad de muestras a ejecutar y cantidad de hilos a utilizar por procesos.
- Modificación de parámetros en tiempo de ejecución. Por lo cual no hace falta tener que adaptar el código ante un nuevo cambio. Esto proporciona mejor automatización.
- Todo queda en un mismo lugar y sobre una misma forma de programación.
- No hay susceptibilidad de errores en la programación ya que se proporciona con un gran compilador y flags de complemento para la detección de errores.
- Entorno robusto. Por la gran cantidad de años del lenguaje C y la librería openMP se tiene herramientas de confianza y calidad por lo que garantizan buenos resultados.
- Grandes cantidades de información en la nube.

Desventajas:

- Se debe tener conocimientos en programación del lenguaje C.
- No tan rápido de utilizar. Se requerirá armar un código que se adapte al uso deseado.