UNIVERSIDAD NACIONAL DE CÓRDOBA
Facultad de Ciencias Exactas, Físicas y Naturales

Cátedra de Sistemas Operativos II

PROGRAMACIÓN DISTRIBUIDA

Autor: Tissot Esteban
e-mail: egtissot@gmail.com

Sistemmas Operativos II
Departamento de Computación
FCEFyN - UNC

Córdoba, 14/03/2017

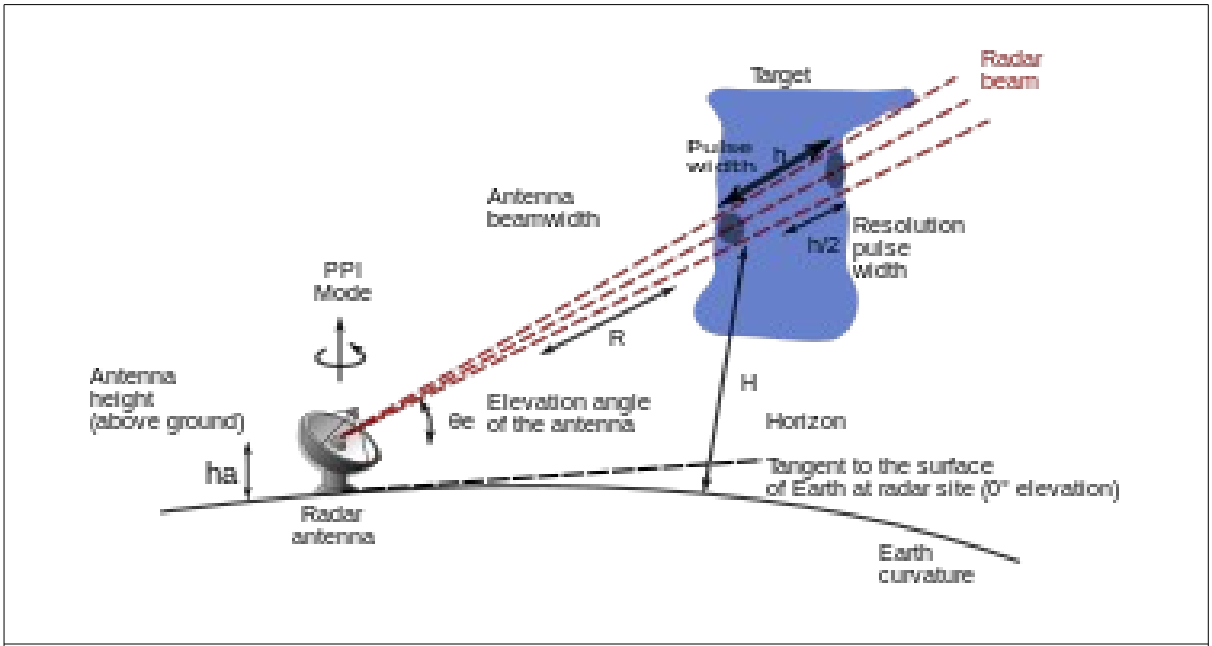
Índice

Introducción
Descripción del problema
Requerimientos y tareas
Diseño y documentación
Implementación y Resultados
Conclusiones

Introducción

La UNC posee un radar Hidro-meteorológico (RMA 1)que opera en Banda C (5,625 Ghz), al cual se le desea introducir un procesador Doppler que permita calcular la componete radial del campo velocidad. La palabra radar, es un acrónimo de radio detection and ranging.

El funcionamiento de un radar, de forma muy resumida y simplificada, consiste en la emisión de una onda electromagnética, que recorre el espacio hasta encontrarse con un blanco. Este blanco, al ser inducido por la onda, se comporta como un espejo, que hace retornar la señal a la antena que emitió la onda. Esta señal se procesa y es permite extraer información del blanco. El funcionamiento del mismo se observa en la ilustración 1.

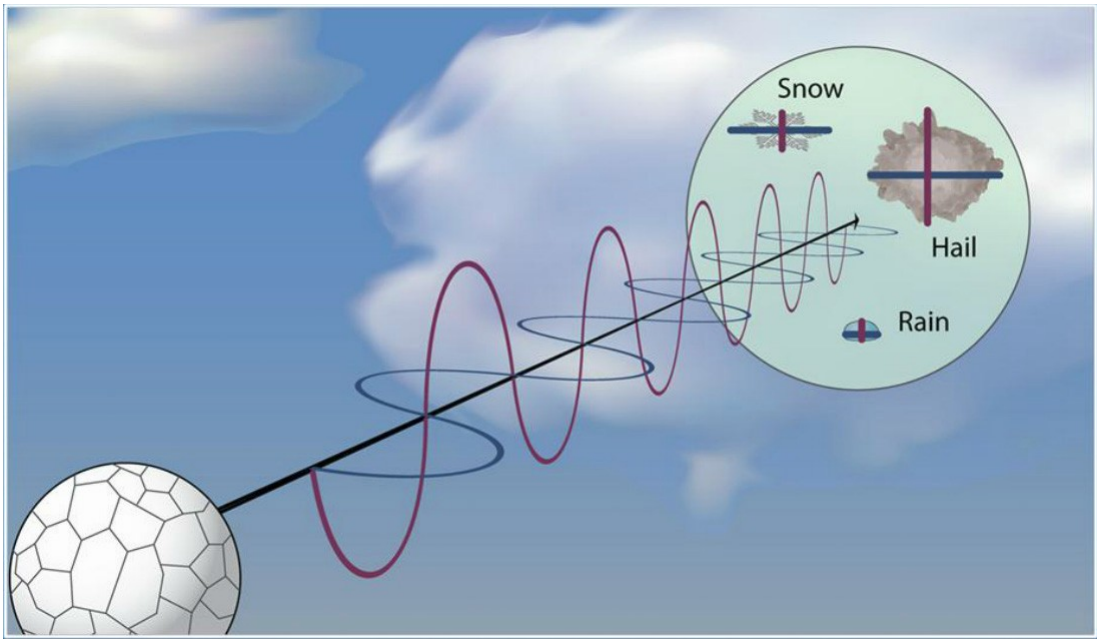


ILISTRACION 1: Radar Hidro-Meteorológico

La antena gira sobre su eje vertical en 360° (una vuelta completa) y la posición de la antena sobre este eje de denomina acimut. A su vez, puede moverse en su eje horizontal, mas comúnmente entre 0° y 90°, que se denomina elevación.

El radar con que se desea trabajar del tipo pulsado, posee un transmisor que genera pulsos electromagnéticos en doble polarización, es decir, transmite un pulso con polarización horizontal (0°) y otro con polarización vertical (90°), en forma conjunta.

Cada canal se denomina H y V respectivamente. A su vez, recordemos que una señal electromagnética es compleja, y esta se puede descomponer en sus componentes en fase y en cuadratura (I + jQ).



ILUSTRACION 2: Pulso de doble polarización

El radar posee una resolución de 0,5 km en rango, y un rango máximo de cobertura de 250km de radio. A cada componente de rango se le denomina gate.

Descripción del problema

Se dispone de un archivo binario (pulsos.iq), que contiene la salida del conversor analógico digital (ADC), que esta ubicado en la entrada del Receptor Digital. El mismo digitaliza el canal H y el canal V, datos que se denominan N1 (Nivel 1). El archivo esta conformado por N pulsos, emitidos a elevación 0,5° y acimut 83°.

Nombre	Tipo	Descripción
ValidSamples	Uint16_t	Cantidad de muestras válidas por canal
V_I[1]	float	Muestra 1 polaridad vertical en fase
V_Q[1]	float	Muestra 1 polaridad vertical en cuadrante
V_I[2]	float	Muestra 2 polaridad vertical en fase
V_Q[2]	float	Muestra 2 polaridad vertical en cuadrante
V_I[...]	float	
V_Q[...]	float	
H_I[1]	float	Muestra 1 polaridad vertical en fase
H_Q[1]	float	Muestra 1 polaridad vertical en cuadrante
H_I[2]	float	Muestra 1 polaridad vertical en fase
H_Q[2]	float	Muestra 1 polaridad vertical en cuadrante

A los fines de implementar el procesador Doppler se pide que:

- Obtenga el valor de cada gate, a partir del cálculo de la media arimética de las n muestras que caen en el rango de resolución. Es decir, si se tienen 5000 validSamples, cada gate se obtendrá a partir de 10 muestras (0,5km * 5000 muestras / 250 km). Esto se debe realizar para el canal H y V. Como resultado, obtendremos una estructura de datos que se puede interpretar como una matriz (gate,pulso), por canal.

- Luego, se debe aplicar la formula de auto correlación en tiempo discreto, sobre todos los valores de de todos los pulsos para un determinado gate, es decir, por columna. La ecuación de autocorrelación es la siguiente:

$$R(T_s)=\frac{1}{M}\cdot\sum_{m=1}^{M-1} V*_m V_{m+1}$$

Donde M es la cantidad de pulsos y V es el valor absoluto de cada gate compleja.

Requerimientos y tareas

- El resultado del procesamiento se debe guardar en un archivo binario indicando en la documentación la estructura del mismo.
- Primero, se debe realizar un diseño que sea solución al problema sin explotar el paralelismo (procedural).
- Reconocer qué tipo de paralelismo exhibe el problema en cuestión.
- Diseñar la solución del mismo determinando cuáles son los datos/operaciones paralelizables.
- Implementar la solucion mediante el uso de la librería OpenMP, explotando el paralelismo del problema.

Diseño y documentación

Diseño Procedural

Se diseñaron cuatro funciones:

- n_pulsos(). Abre el archivo “pulsos.iq” y calcula la cantidad de pulsos y la cantidad máxima de muestras.
- pulsos(). Utiliza la información del archivo “pulsos.iq” para crear una matriz(gate,pulso) por canal.
- autocorrelacion(). Implementa la formula de autocorrelación y guarda el resultado en un vector.
- escribir_archivo(). Escribe un archivo binario con el resultado de la autocorrelación.

Archivos:

Es necesario disponer de un archivo binario (pulsos.iq) que se encuentre en el mismo directorio que el programa. Al finalizar el mismo, se crearan dos archivos binarios, uno por cada canal, llamados “autocorrelacion_canal_H.iq” y “autocorrelacion_canal_V.iq”. Estos archivos contienen números float que se corresponden con los valores de autocorrelación obtenidos.

Diseño en Paralelo

Al tener una matriz (gate,pulso) por canal la inicialización de cada una se puede hacer en paralelo.

La función de autocorrelacion utiliza la matriz(gate,pulso) y se debe ejecutar la para ambos canales. Como la matriz tiene, en este caso, 500 filas se decidió que todos los hilos trabajen sobre el primer canal y luego para el segundo. Este esquema limita a la paralelización a un máximo de 500 hilos, que para el objetivo del trabajo es más que suficiente.

Se decidió escribir un archivo por cada canal, de esta forma se definen dos secciones, con un hilo cada una, que trabajan en forma paralela.

Profiling

El profiling es el análisis de rendimiento de un programa, el mismo se basa en información reunida desde el análisis dinámico del mismo. El objetivo puede ser averiguar el tiempo dedicado a la ejecución de diferentes partes del programa para detectar los puntos problemáticos o para mejorar las áreas dónde sea posible llevar a cabo una amortización del rendimiento (ya sea en velocidad o en consumo de recursos). Un profiler puede proporcionar información como una traza de ejecución o un resumen estadístico de los eventos observados.

Linux proporciona varias herramientas que nos permiten analizar un programa en ejecución: prof, gprof, monitor, time. Gprof puede producir varios estilos de analisis, los más simples son el ”file information”, ”execution count”, ”function and file ordering”. Ademas existen otros importantes como los que se describen a continuación.

- File information: Muestra los registros de histograma, call graph y basic-block.
- Flat Profile: Muestra cuanto tiempo se gasto en cada función.
- Call Graph: muestra cual funcion fue llamada por otra y cuanto tiempo usa cada función cuando las mismas realizan llamadas a subrutinas.
- Line-by- line: gprof puede analizar lineas individuales de codigo fuente.

El comando time es bueno para realizar pruebas de performance en base a tiempo, ya que ejecuta un comando, y luego despliega en la salida estándar la siguiente información:

- El tiempo real transcurrido entre la llamada y la finalización de orden.
- El tiempo de usuario del procesador (tms_utime + tms_cutime).
- El tiempo de sistema del procesador (tms_stime + tms_cstime.)

Implementación

Se utilizo OpenMP para poder realizar la implementación en paralelo, por lo que es necesario incluir la librería <omp.h>. De esta librería se utilizó las siguientes estructuras:

```
 #pragma omp parallel for private(i) num_threads(num_th_write)
        /* bucle for */

#pragma omp parallel for private(i,j) collapse(2)
        /* bucle for para i*/
                /* bucle for para j */

#pragma omp parallel sections
{
         #pragma omp section
         {
                /* sección 1 */
         }
         #pragma omp section
         {
                /* sección 2 */
         }
}
```

```
omp_get_max_threads()  /* Devuelve la cantidad maxima de hilos disponibles */
```

Para calcular los tiempos se optó por usar el comando omp_get_wtime() debido a su simpleza y claridad en los resultados de sus análisis.

Compilación:

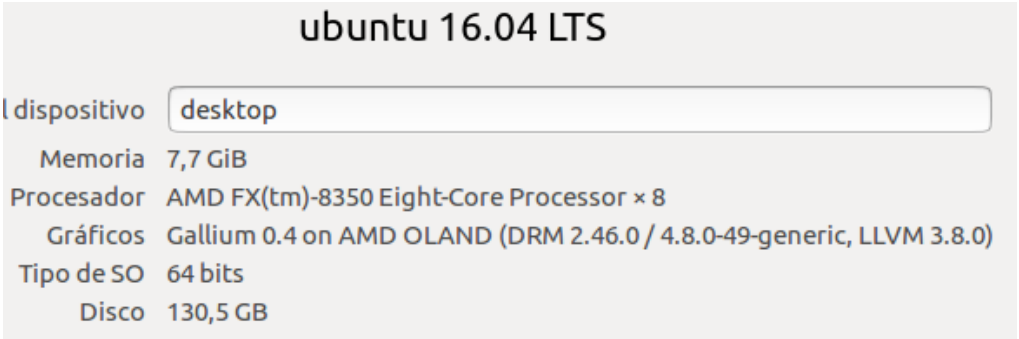
Para poder compilar es necesario agregar la bandera: **-fopenmp**. Se provee de un archivo makefile, por lo que solo es necesario ejecutar el comando **make** para compilar el programa antes de ejecutarlo y si se quiere remover los archivos creados se debe ejecutar make clean. Para ejecutar el programa es necesario abrir una terminal en la carpeta donde se encuentra el programa y, luego de compilar, ejecutar el comando ./radar {cantidad_de_hilos}.

Resultados

PC Desktop

En las computadoras personales hay mucho procesamiento que no se puede separar, es decir que no podemos asignar núcleos específicamente a nuestro programa, por lo que en algunos casos los hilos de nuestro programa competirán con hilos de otros por tiempo de procesador, este tiempo de “espera” es el que hace que los datos no sean fiables. Teniendo en cuenta esto se realizo una ejecución después de iniciar la computadora y se alcanza a observar una mejora utilizando ejecución en paralelo.

Información del Sistema



Ejecución en PC-desktop

```
esteban@desktop:~/Escritorio/SO_II/TP_2/src$ ./radar 1
Ejecucion Procedural
Tiempo: 0.428611
esteban@desktop:~/Escritorio/SO_II/TP_2/src$ ./radar 2
Ejecucion en Paralelo con 2 hilos de 8 disponibles
Tiempo: 0.071064
esteban@desktop:~/Escritorio/SO_II/TP_2/src$ ./radar 4
Ejecucion en Paralelo con 4 hilos de 8 disponibles
Tiempo: 0.069560
esteban@desktop:~/Escritorio/SO_II/TP_2/src$ ./radar 6
Ejecucion en Paralelo con 6 hilos de 8 disponibles
Tiempo: 0.073416
esteban@desktop:~/Escritorio/SO_II/TP_2/src$ ./radar 8
Ejecucion en Paralelo con 8 hilos de 8 disponibles
Tiempo: 0.072949
```

Resultados en el Cluster

Se ejecuto el programa en el Cluster “pulqui” del Laboratorio de Cómputos que cuenta con 24 núcleos intel de 2.30GHz. A continuación se muestran algunas capturas seteando distinta cantidad de hilos para la ejecución.

Ejecución en el cluster “pulqui”

```
[Alumno45@pulqui src]$ ./radar 1
Ejecucion Procedural
Tiempo: 0.329258
```

```
[Alumno45@pulqui src]$ ./radar 2
Ejecucion en Paralelo con 2 hilos de 24 disponibles
Tiempo: 0.210251
```

```
[Alumno45@pulqui src]$ ./radar 4
Ejecucion en Paralelo con 4 hilos de 24 disponibles
Tiempo: 0.175497
```

```
[Alumno45@pulqui src]$ ./radar 8
Ejecucion en Paralelo con 8 hilos de 24 disponibles
Tiempo: 0.155377
```

```
Ejecucion en Paralelo con 16 hilos de 24 disponibles
Tiempo: 0.178041
[Alumno45@pulqui src]$ ./radar 16
```

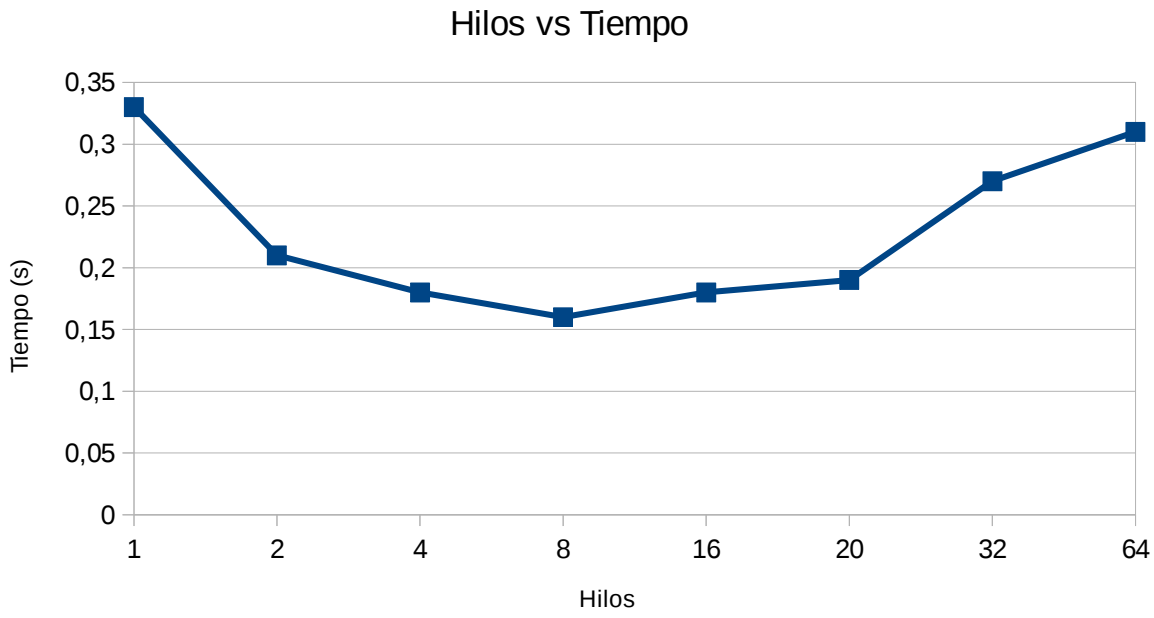
```
[Alumno45@pulqui src]$ ./radar 20
Ejecucion en Paralelo con 20 hilos de 24 disponibles
Tiempo: 0.190511
```

En la siguiente imagen se observa que si forzamos a que el programa se ejecute con mas hilos de los disponibles, el rendimiento es peor que si se ejecutara de forma procedural. Este comportamiento se debe a un conflicto en la cache de los núcleos, debido a que hay mayor cantidad de hilos que núcleos, este problema es conocido con el nombre de “false sharing” .

Ejecución en el cluster “pulqui”

```
[Alumno45@pulqui src]$ ./radar 32
Ejecucion en Paralelo con 32 hilos de 24 disponibles
Tiempo: 0.272446
[Alumno45@pulqui src]$ ./radar 64
Ejecucion en Paralelo con 64 hilos de 24 disponibles
Tiempo: 0.309896
```

A continuación se realizó un gráfico con los resultados obtenidos anteriormente, desde este punto de vista se puede observar la mejora significativa que inserta el paralelismo en cuestiones de tiempo, teóricamente con dos hilos se reduce un 40% del tiempo de ejecución de un hilo. A medida que aumentamos la cantidad de hilos seguimos percibiendo una mejora aunque no tan significativa. Esta mejora se empieza a reducir a medida que nos acercamos a la cantidad de núcleos físicos que tiene el computador y empeora si nos pasamos.



Conclusiones

Con este trabajo se obtuvo una idea básica del concepto de programación con uso de OpenMP, no solo en la etapa de codificación y ejecución sino también en la etapa previa de diseño, en donde se deben tener en cuenta factores como las partes del programa que son posibles de ser paralelizadas y las partes que no.

Se comprendió el funcionamiento de la herramienta OpenMP y la forma en que esta puede ser utilizada con el objeto de mejorar el rendimiento del sistema a la hora de resolver un problema que requiere una gran cantidad de cálculos.

Por último a través de las mediciones realizadas se pudo constatar la mejoría que ofrece la programación paralela a pesar de agregarle o modificar el código original comparando las distintas formas de ejecutar un mismo programa bajo diferentes configuraciones de sistema, obteniendo así mejoras de hasta un 60% en tiempos de ejecución.

Bibliografia

http://supercomputingblog.com/openmp/tutorial-parallel-for-loops-with-openmp/

http://bisqwit.iki.fi/story/howto/openmp/

Archivos adjuntos.