



UNIVERSIDAD NACIONAL DE CÓRDOBA

Facultad de Ciencias Exactas, Físicas y Naturales

Cátedra de Sistemas Operativos II

TRABAJO PRÁCTICO 2

Programación paralela con OpenMP

Mauricio Aguilar

7 de Mayo de 2016

Contenido

Introducción	3	2
Descripción del problema	4	
Requerimientos y tareas	7	
Diseño y documentación.....	8	
Implementación y Resultados.....	11	
Conclusión.....	22	

Introducción

OpenMP es una interfaz de programación de aplicaciones (API) para la programación multiproceso de memoria compartida en múltiples plataformas. Permite añadir concurrencia a los programas escritos en C, C++ y Fortran sobre la base del modelo de ejecución fork-join. Está disponible en muchas arquitecturas, incluidas las plataformas de Unix y de Microsoft Windows. Se compone de un conjunto de directivas de compilador, rutinas de biblioteca, y variables de entorno que influyen el comportamiento en tiempo de ejecución.

Definido conjuntamente por proveedores de hardware y de software, OpenMP es un modelo de programación portable y escalable que proporciona a los programadores una interfaz simple y flexible para el desarrollo de aplicaciones paralelas, para plataformas que van desde las computadoras de escritorio hasta supercomputadoras. Una aplicación construida con un modelo de programación paralela híbrido se puede ejecutar en un cluster de computadoras utilizando OpenMP y MPI, o a través de las extensiones de OpenMP para los sistemas de memoria distribuida.

En este desarrollo se utilizó OpenMP para paralizar y optimizar un procesamiento de señales

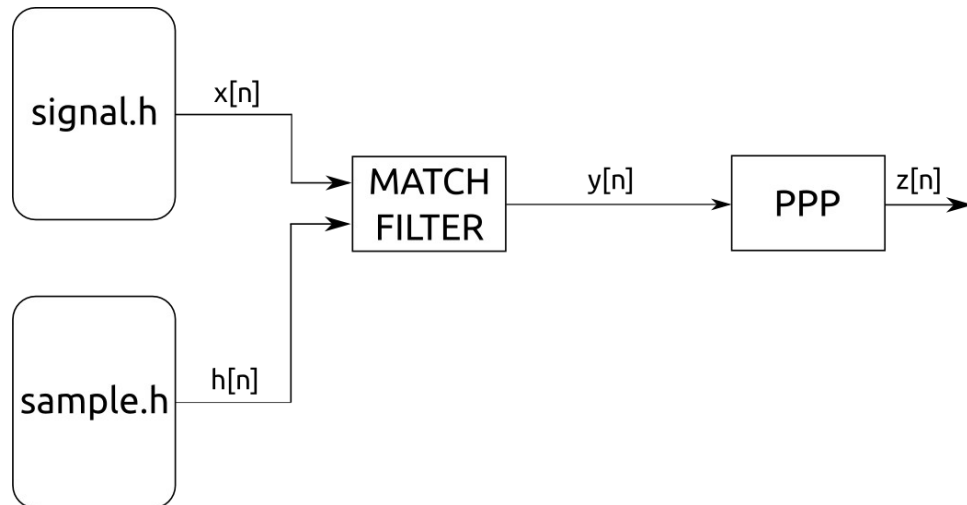
Descripción del problema

En un sistema radar, se genera una señal (pulso) coherente. Esta señal es enviada a través de su canal de transmisión y se toma una muestra de la misma antes de ser enviada. Una vez transmitida la señal, el sistema comienza a recibir ecos provenientes de objetivos y guarda la señal recibida. Se desea conocer cuanto se ha modificado la señal enviada y aumentar su relación señal/ruido. También se le desea realizar un procesamiento Doppler, denominado Procesamiento de Pares de Pulsos (PPP). Para esto se implementa un Match Filter entre la muestra de la señal transmitida y la señal recibida y luego se realiza el procesamiento PPP.

Suponemos que la antena de radar no se mueve y envía pulsos a una única dirección (acimut y elevación fijos). El sistema envía 50 pulsos idénticos (por tratarse de un sistema coherente), y a la señal recibida la divide en 50 gates (píxeles) de misma duración que el pulso transmitido, formando una matriz de 50 pulsos por 50 gates. Donde cada columna se denomina haz, y cada elemento fila es un gate (un único dato).

Cabe destacar que tanto la señal muestreada y la recibida pasaron por el mismo sistema de recepción y fueron digitalizadas con el mismo bitrate. Esto quiere decir que la muestra y cada gate tienen la misma cantidad de muestras digitales.

Por esto, se pide realizar un algoritmo en C, que realice el procesamiento de la señal recibida, por el radar. El procesamiento consiste en realizar el filtrado de una señal entrante (simulada, con ruido desfasaje aleatorio, entregada en el archivo *signal.h*), mediante un filtro Adaptado (Match Filter, con la señal muestreada y dada en el archivo *sample.h*) a los fines de detectar la similitud entre dos señales, y luego realizarle un procesamiento de par de pulsos, tal como se muestre en el siguiente diagrama esquemático.



Sea $h[n]$ la muestra de la señal transmitida (*sample.h*) y $x[n]$ la señal recibida (*signal.h*), se define un Match Filter como:

$$y[n] = \sum_{i=-\infty}^{+\infty} h[n-i]x[i]$$

Luego, esta señal $y[n]$ ingresa al PPPn y realiza el procesamiento:

$$z[n] = \sum_{i=-0}^{M-2} y[n+1]y[i]$$

Donde M son la cantidad de muestra en todo un haz. Los valores calculados se deben guardar en un archivo.

Para esta tarea se cuenta con un set de datos disponible en el LEV, junto a este archivo. Tal como se dijo anteriormente, se entregan dos archivos: *signal.h* y *sample.h*. El primero contiene las muestras almacenadas de la señal recibida y el segundo las muestras de la señal transmitida. Este último contiene 500 muestras digitales.

Se debe realizar, primero, un diseño que sea solución al problema sin explotar el paralelismo (procedural). Luego, a partir de este, realizar una nueva implementación que realice el proceso mediante el uso de la librería OpenMP, explotando el paralelismo del problema. Para ello, se requiere reconocer qué tipo de paralelismo exhibe el problema en cuestión y luego, diseñar la solución del mismo determinando cuáles son los datos/operaciones paralelizables. Se tendrá en cuenta, el nivel de paralelismo alcanzado.

Además, se deberá elaborar un informe con gráficos/tablas de los datos recopilados de varias ejecuciones del programa, tanto en el clúster de la facultad como localmente en una sola PC, indicando qué ganancia en *performance* existe al distribuir este proceso en comparación a la ejecución local. También se deberá investigar acerca de qué utilidades de *profiling* gratuitas existen y qué brinda cada una, optando por una para realizar las mediciones de tiempo de ejecución de la aplicación diseñada.

NOTA 1: tanto la señal muestreada y la recibida son datos brindados en `datosTP2.rar`. Sólo se requiere que implementen el filtrado de manera local y paralela para comparar las ventajas de la ejecución distribuida.

NOTA2: todavía no se encuentra habilitada la cuenta de acceso al clúster de la facultad. Cuando esté disponible la misma, se informará por LEV los datos necesarios.

Requerimientos y tareas

Requerimientos Funcionales:

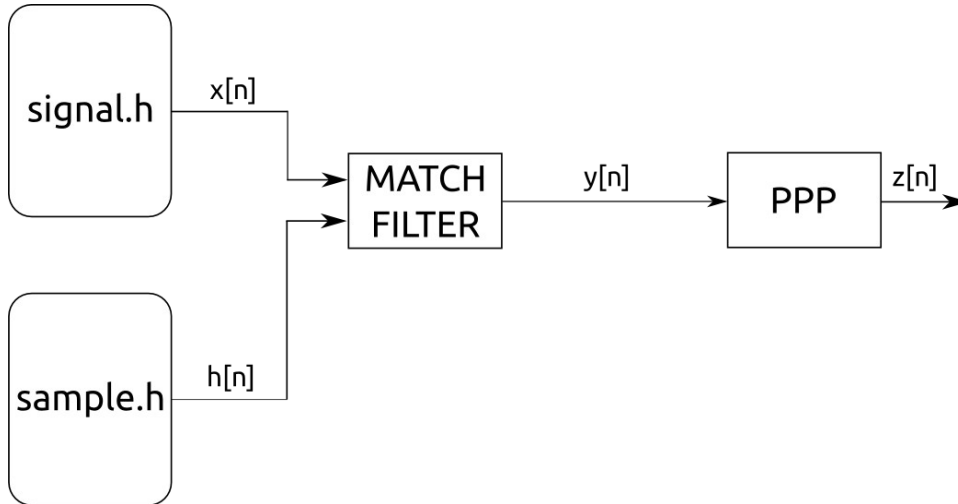
- Realizar un programa con una función que calcule la convolución discreta entre dos señales, `signal` y `sample`, para implementar así el Match Filter.
- Realizar una función que realice el procesamiento PPP con la salida obtenida en el Match Filter.
- Guardar la señal procesada en un archivo `ppp.h`.

Requerimientos no funcionales:

- Analizar el Rendimiento del programa, para ello:
 - Se deberá realizar un programa procedural (hilo único) en el que se resuelva el problema en cuestión, y luego un programa paralelo utilizando técnicas de OpenMP.
 - Se deberá comparar los resultados entre ambas situaciones tanto en equipo local como en el nodo Franky.
 - Mostrar resultados con tablas y gráficos que muestren los puntos de óptima ejecución.
- Desarrollar el programa de tal modo que pueda ejecutarse sobre consola, tanto en el **equipo local** como en el nodo **franky**.

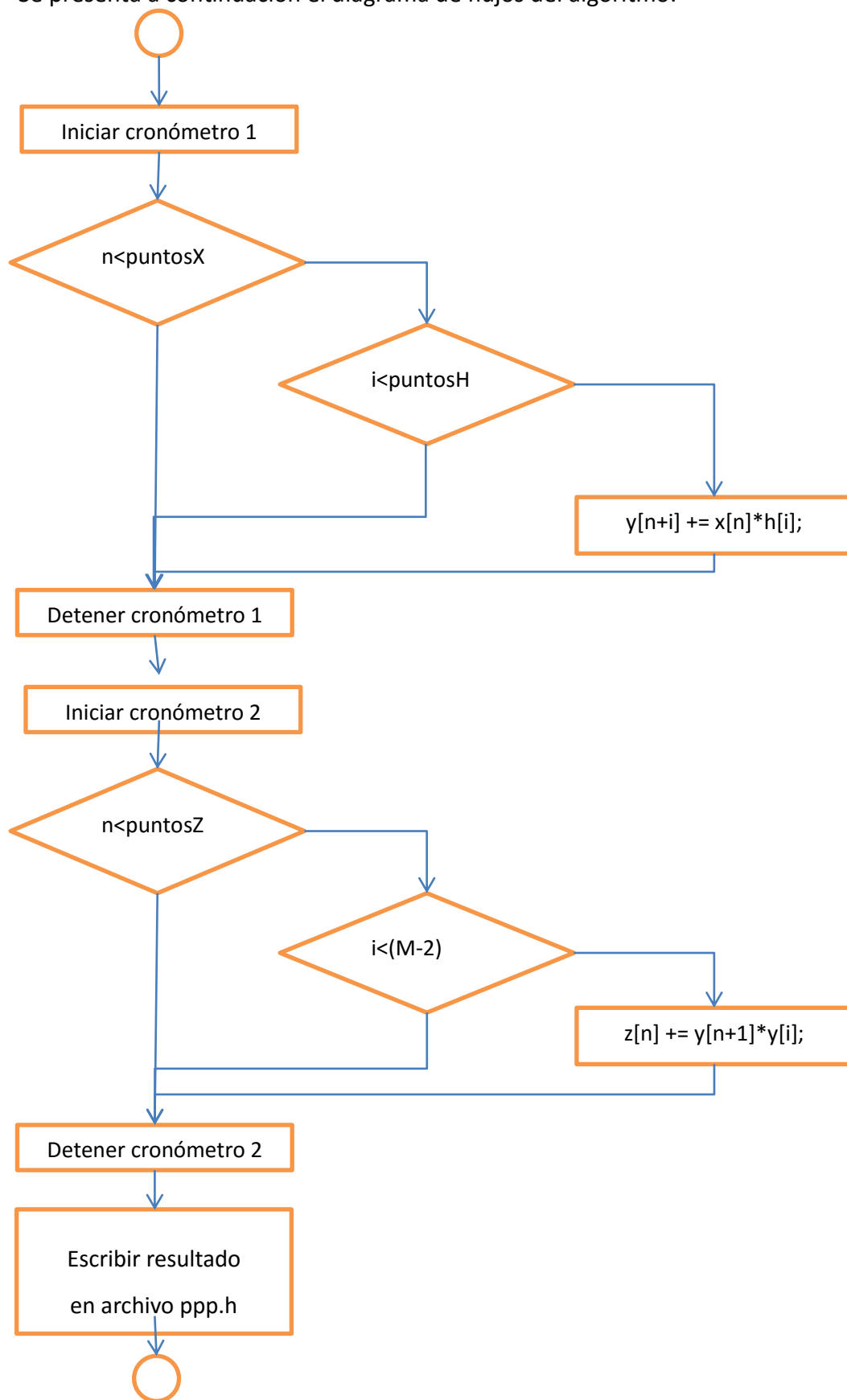
Diseño y documentación

Según lo antes planteado, se ilustra un simple diagrama de componentes para ilustrar la manera en que se relacionan las partes del sistema:



Tanto el Match Filter como el PPP se realizaron de manera procedural y paralela.

Se presenta a continuación el diagrama de flujos del algoritmo:



Se entregan los archivos mencionados en el diagrama de componentes.

- Primero procedemos a compilar y ejecutar en el **equipo local**:

1 Abrir terminal y situarse en la carpeta que contiene los archivos, con el comando `cd`.

2 Compilar con el comando **make**.

3 Para ejecutar hay dos opciones, para el procedural y para el paralelo:

>Para ejecutar una vez el procedural: **./tp2pr**

>Para ejecutar n veces el procedural: **for i in {1..n}; do ./tp2pr; done**

>Para ejecutar una vez el paralelo: **./tp2pa**

>Para ejecutar n veces el paralelo: **for i in {1..n}; do ./tp2a; done**

4 El programa mostrará los tiempos requeridos por cada una de las dos etapas importantes del proceso.

- A continuación, para ejecutar el archivo en el nodo **franky**:

1 Copiar archivos desde el equipo local a franky con el comando `scp`:

Desde una terminal local:

```
>scp /mnt/749261FA9261C16A/so/2tp/tp2procedural.c
Alumno2@200.16.19.197:/export/home/Alumno2/
>scp /mnt/749261FA9261C16A/so/2tp/tp2paralelo.c
Alumno2@200.16.19.197:/export/home/Alumno2/
>scp /mnt/749261FA9261C16A/so/2tp/sample.h
Alumno2@200.16.19.197:/export/home/Alumno2/
>scp /mnt/749261FA9261C16A/so/2tp/signal.h
Alumno2@200.16.19.197:/export/home/Alumno2/
>scp /mnt/749261FA9261C16A/so/2tp/Makefile
Alumno2@200.16.19.197:/export/home/Alumno2/
```

Se solicitará la contraseña del usuario Alumno2 para confirmar el envío.

2 Loguease por ssh a **rocky** y desde ahí seleccionar el nodo **franky**:

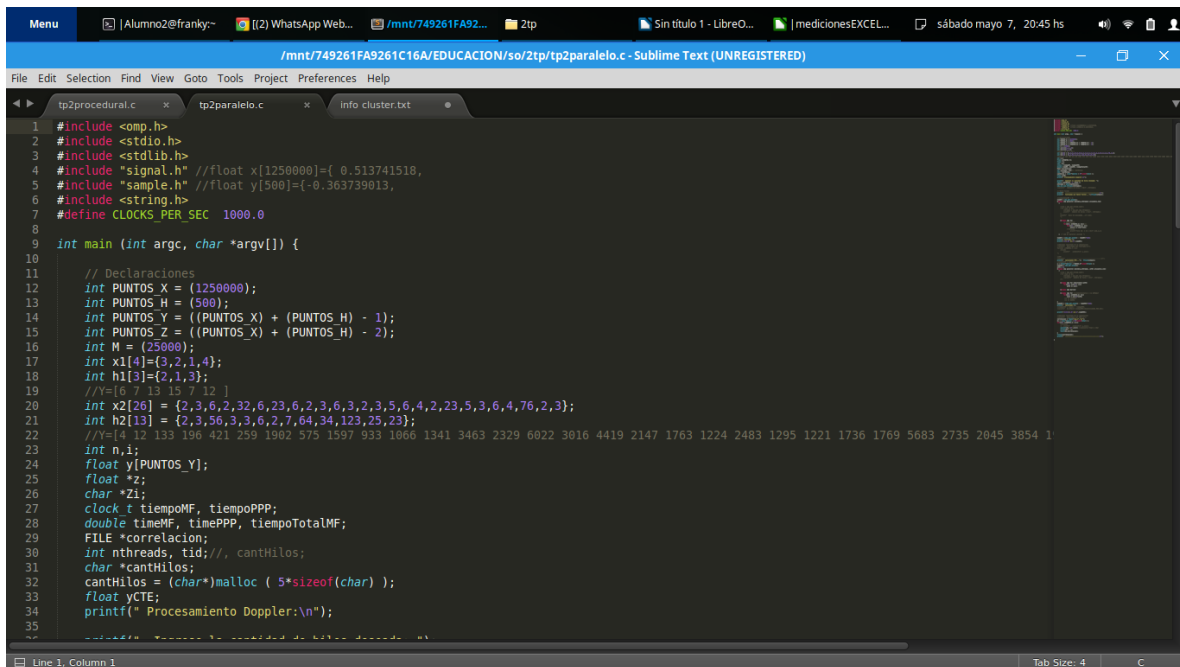
```
>ssh Alumno2@200.16.19.197
>contraseña: (contraseña facilitada)
>ssh franky
```

2 Si los archivos se copiaron correctamente, vamos a verlos con el comando `ls`, y lo compilamos y ejecutamos de la misma manera que se hizo en el modo procedural.

Implementación y Resultados

11

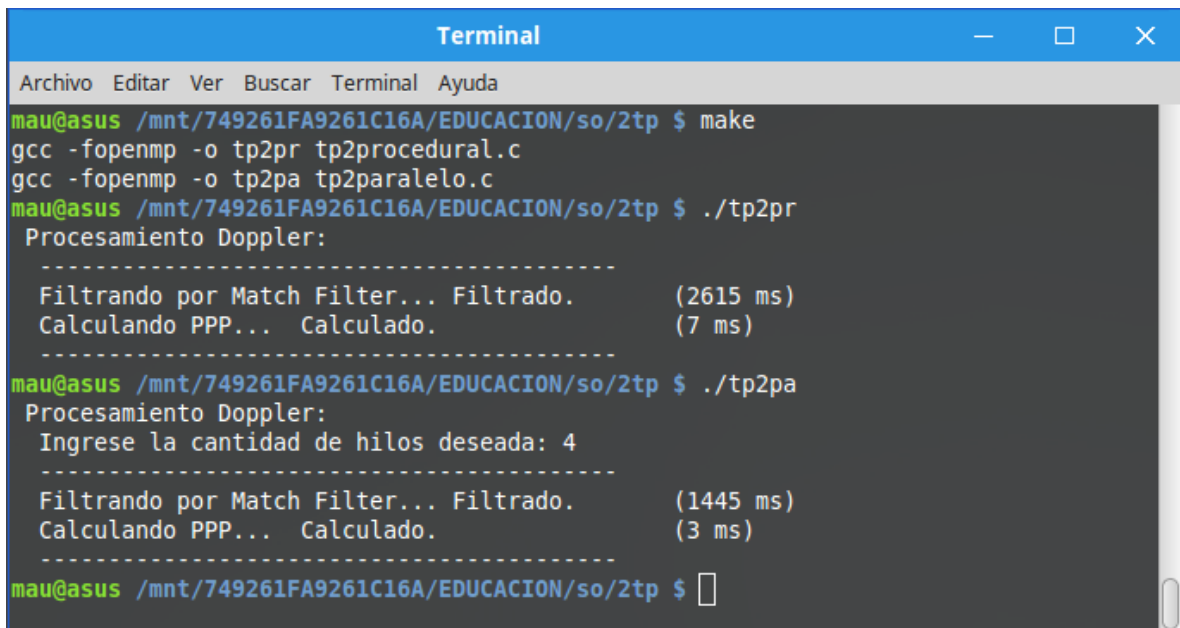
Para codificar se utilizó la herramienta Sublime Text 3:



```

1 #include <omp.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include "signal.h" //float x[1250000]={ 0.513741518,
5 #include "sample.h" //float y[500]={-0.363739013,
6 #include <string.h>
7 #define CLOCKS_PER_SEC 1000.0
8
9 int main (int argc, char *argv[]) {
10
11     // Declaraciones
12     int PUNTOS_X = (1250000);
13     int PUNTOS_H = (500);
14     int PUNTOS_Y = ((PUNTOS_X) + (PUNTOS_H) - 1);
15     int PUNTOS_Z = ((PUNTOS_X) + (PUNTOS_H) - 2);
16     int M = (25000);
17     int x1[4]={3,2,1,4};
18     int h1[3]={2,1,3};
19     //Y=[6 7 13 15 7 12 ]
20     int x2[26] = {2,3,6,2,32,6,23,6,2,3,6,3,2,3,5,6,4,2,23,5,3,6,4,76,2,3};
21     int h2[13] = {2,3,56,3,3,6,2,7,64,34,123,25,23};
22     //Y=[4 12 133 196 421 259 1902 575 1597 933 1066 1341 3463 2329 6022 3016 4419 2147 1763 1224 2483 1295 1221 1736 1769 5683 2735 2045 3854 1
23     int n,i;
24     float y[PUNTOS_Y];
25     float *z;
26     char *Zi;
27     clock_t tiempoMF, tiempoPPP;
28     double timeMF, timePPP, tiempoTotalMF;
29     FILE *correlacion;
30     int nthreads, tid;//, cantHilos;
31     char *cantHilos;
32     cantHilos = (char*)malloc ( 5*sizeof(char) );
33     float yCTE;
34     printf(" Procesamiento Doppler:\n");
35
36     //----- Te voy a dar la cantidad de hilos deseada.
  
```

Para compilar, se utilizó gcc por terminal de Linux.



```

mau@asus /mnt/749261FA9261C16A/EDUCACION/so/2tp $ make
gcc -fopenmp -o tp2pr tp2procedural.c
gcc -fopenmp -o tp2pa tp2paralelo.c
mau@asus /mnt/749261FA9261C16A/EDUCACION/so/2tp $ ./tp2pr
Procesamiento Doppler:
-----
Filtrando por Match Filter... Filtrado.          (2615 ms)
Calculando PPP... Calculado.                      (7 ms)
-----
mau@asus /mnt/749261FA9261C16A/EDUCACION/so/2tp $ ./tp2pa
Procesamiento Doppler:
Ingrese la cantidad de hilos deseada: 4
-----
Filtrando por Match Filter... Filtrado.          (1445 ms)
Calculando PPP... Calculado.                      (3 ms)
-----
mau@asus /mnt/749261FA9261C16A/EDUCACION/so/2tp $
  
```

Cabe destacar que las características básicas de los equipos son:

Equipo Local: Procesar Intel i3 5010U 2.10Ghz, 4GB Memoria RAM

Franky:

Para la ejecución procedural, se realizaron 5 iteraciones en el equipo local y 5 en el nodo franky, para luego promediar los tiempos obtenidos:

PROCEDURAL				
Run	EQUIPO LOCAL		FRANKY	
	Match Filter	PPP	Match Filter	PPP
1	2.621 ms	7 ms	6.146 ms	14 ms
2	2.754 ms	7 ms	6.144 ms	13 ms
3	2.630 ms	7 ms	6.134 ms	14 ms
4	2.663 ms	7 ms	6.141 ms	13 ms
5	2.631 ms	7 ms	6.136 ms	14 ms
Prom	2.660 ms	7 ms	6.140 ms	14 ms

Para la ejecución paralela, se realizaron 5 iteraciones para 1,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,33,34,64,128,256 hilos, para luego promediarlos:

PARALELO (1 hilo)				
Run	EQUIPO LOCAL		FRANKY	
	Match Filter	PPP	Match Filter	PPP
1	2.825 ms	6 ms	4.956 ms	13 ms
2	1.976 ms	3 ms	4.934 ms	14 ms
3	1.514 ms	3 ms	5.008 ms	12 ms
4	1.435 ms	4 ms	4.949 ms	14 ms
5	1.463 ms	4 ms	4.959 ms	14 ms
Prom	1.843 ms	4 ms	4.961 ms	13 ms

PARALELO (2 hilos)				
Run	EQUIPO LOCAL		FRANKY	
	Match Filter	PPP	Match Filter	PPP
1	1.832 ms	3 ms	2.488 ms	8 ms
2	1.446 ms	3 ms	2.666 ms	10 ms
3	1.461 ms	3 ms	4.984 ms	13 ms
4	1.434 ms	3 ms	2.488 ms	7 ms
5	1.826 ms	3 ms	2.484 ms	7 ms
Prom	1.600 ms	3 ms	3.022 ms	9 ms

PARALELO (4 hilos)				
Run	EQUIPO LOCAL		FRANKY	
	Match Filter	PPP	Match Filter	PPP
1	1.429 ms	3 ms	1.254 ms	4 ms
2	1.437 ms	3 ms	1.382 ms	7 ms
3	1.445 ms	3 ms	1.248 ms	4 ms
4	1.434 ms	3 ms	1.248 ms	5 ms
5	1.445 ms	3 ms	1.249 ms	4 ms
Prom	1.438 ms	3 ms	1.276 ms	5 ms

PARALELO (6 hilos)				
Run	EQUIPO LOCAL		FRANKY	
	Match Filter	PPP	Match Filter	PPP
1	1.445 ms	3 ms	920 ms	5 ms
2	1.452 ms	4 ms	841 ms	3 ms
3	1.427 ms	4 ms	837 ms	3 ms
4	1.438 ms	3 ms	848 ms	3 ms
5	1.476 ms	4 ms	941 ms	5 ms
Prom	1.448 ms	4 ms	877 ms	4 ms

PARALELO (8 hilos)				
Run	EQUIPO LOCAL		FRANKY	
	Match Filter	PPP	Match Filter	PPP
1	1.440 ms	3 ms	626 ms	2 ms
2	1.437 ms	3 ms	625 ms	2 ms
3	1.329 ms	4 ms	626 ms	2 ms
4	1.436 ms	3 ms	625 ms	2 ms
5	1.450 ms	3 ms	635 ms	2 ms
Prom	1.418 ms	3 ms	627 ms	2 ms

PARALELO (10 hilos)				
Run	EQUIPO LOCAL		FRANKY	
	Match Filter	PPP	Match Filter	PPP
1	1.435 ms	4 ms	526 ms	2 ms
2	1.423 ms	3 ms	515 ms	3 ms
3	1.439 ms	3 ms	561 ms	2 ms
4	1.436 ms	3 ms	522 ms	2 ms
5	1.483 ms	3 ms	523 ms	2 ms
Prom	1.443 ms	3 ms	529 ms	2 ms

PARALELO (12 hilos)				
Run	EQUIPO LOCAL		FRANKY	
	Match Filter	PPP	Match Filter	PPP
1	1.467 ms	12 ms	430 ms	2 ms
2	1.453 ms	3 ms	426 ms	2 ms
3	1.430 ms	4 ms	423 ms	3 ms
4	1.435 ms	3 ms	426 ms	2 ms
5	1.436 ms	3 ms	419 ms	3 ms
Prom	1.444 ms	5 ms	425 ms	2 ms

PARALELO (14 hilos)				
Run	EQUIPO LOCAL		FRANKY	
	Match Filter	PPP	Match Filter	PPP
1	1.440 ms	4 ms	368 ms	2 ms
2	1.430 ms	5 ms	399 ms	2 ms
3	1.443 ms	4 ms	366 ms	2 ms
4	1.428 ms	3 ms	357 ms	2 ms
5	1.457 ms	4 ms	370 ms	2 ms
Prom	1.440 ms	4 ms	372 ms	2 ms

PARALELO (16 hilos)				
Run	EQUIPO LOCAL		FRANKY	
	Match Filter	PPP	Match Filter	PPP
1	1.480 ms	3 ms	358 ms	2 ms
2	1.463 ms	3 ms	325 ms	3 ms
3	1.440 ms	4 ms	324 ms	2 ms
4	1.440 ms	4 ms	315 ms	3 ms
5	1.444 ms	3 ms	342 ms	2 ms
Prom	1.453 ms	3 ms	333 ms	2 ms

PARALELO (18 hilos)				
Run	EQUIPO LOCAL		FRANKY	
	Match Filter	PPP	Match Filter	PPP
1	1.435 ms	3 ms	293 ms	3 ms
2	1.434 ms	3 ms	311 ms	3 ms
3	1.438 ms	3 ms	286 ms	2 ms
4	1.444 ms	3 ms	291 ms	3 ms
5	1.442 ms	4 ms	291 ms	2 ms
Prom	1.439 ms	3 ms	294 ms	3 ms

PARALELO (20 hilos)				
Run	EQUIPO LOCAL		FRANKY	
	Match Filter	PPP	Match Filter	PPP
1	1.432 ms	4 ms	268 ms	2 ms
2	1.438 ms	4 ms	265 ms	2 ms
3	1.460 ms	4 ms	266 ms	2 ms
4	1.439 ms	5 ms	261 ms	3 ms
5	1.458 ms	4 ms	276 ms	2 ms
Prom	1.445 ms	4 ms	267 ms	2 ms

PARALELO (22 hilos)				
Run	EQUIPO LOCAL		FRANKY	
	Match Filter	PPP	Match Filter	PPP
1	1.451 ms	5 ms	235 ms	2 ms
2	1.451 ms	4 ms	238 ms	2 ms
3	1.437 ms	3 ms	240 ms	2 ms
4	1.442 ms	4 ms	236 ms	2 ms
5	1.435 ms	4 ms	239 ms	2 ms
Prom	1.443 ms	4 ms	238 ms	2 ms

PARALELO (24 hilos)				
Run	EQUIPO LOCAL		FRANKY	
	Match Filter	PPP	Match Filter	PPP
1	1.433 ms	3 ms	214 ms	2 ms
2	1.436 ms	4 ms	212 ms	2 ms
3	1.444 ms	3 ms	214 ms	2 ms
4	1.437 ms	5 ms	213 ms	2 ms
5	1.437 ms	5 ms	218 ms	2 ms
Prom	1.437 ms	4 ms	214 ms	2 ms

PARALELO (26 hilos)				
Run	EQUIPO LOCAL		FRANKY	
	Match Filter	PPP	Match Filter	PPP
1	1.430 ms	3 ms	211 ms	2 ms
2	1.432 ms	3 ms	209 ms	4 ms
3	1.446 ms	3 ms	222 ms	2 ms
4	1.442 ms	4 ms	205 ms	2 ms
5	1.431 ms	4 ms	206 ms	3 ms
Prom	1.436 ms	3 ms	211 ms	3 ms

PARALELO (28 hilos)				
Run	EQUIPO LOCAL		FRANKY	
	Match Filter	PPP	Match Filter	PPP
1	1.431 ms	3 ms	192 ms	2 ms
2	1.434 ms	3 ms	185 ms	3 ms
3	1.434 ms	4 ms	187 ms	4 ms
4	1.434 ms	4 ms	189 ms	4 ms
5	1.436 ms	3 ms	187 ms	3 ms
Prom	1.434 ms	3 ms	188 ms	3 ms

PARALELO (30 hilos)				
Run	EQUIPO LOCAL		FRANKY	
	Match Filter	PPP	Match Filter	PPP
1	1.435 ms	3 ms	173 ms	2 ms
2	1.435 ms	4 ms	178 ms	2 ms
3	1.433 ms	4 ms	180 ms	4 ms
4	1.435 ms	6 ms	179 ms	2 ms
5	1.449 ms	10 ms	176 ms	4 ms
Prom	1.437 ms	5 ms	177 ms	3 ms

PARALELO (32 hilos)				
Run	EQUIPO LOCAL		FRANKY	
	Match Filter	PPP	Match Filter	PPP
1	1.436 ms	6 ms	167 ms	2 ms
2	1.442 ms	5 ms	171 ms	4 ms
3	1.464 ms	6 ms	168 ms	3 ms
4	1.434 ms	4 ms	170 ms	2 ms
5	1.450 ms	6 ms	185 ms	2 ms
Prom	1.445 ms	5 ms	172 ms	3 ms

PARALELO (33 hilos)				
Run	EQUIPO LOCAL		FRANKY	
	Match Filter	PPP	Match Filter	PPP
1	1.449 ms	4 ms	251 ms	4 ms
2	1.430 ms	5 ms	271 ms	6 ms
3	1.442 ms	4 ms	594 ms	5 ms
4	1.445 ms	4 ms	270 ms	4 ms
5	1.476 ms	4 ms	264 ms	5 ms
Prom	1.448 ms	4 ms	330 ms	5 ms

PARALELO (34 hilos)				
Run	EQUIPO LOCAL		FRANKY	
	Match Filter	PPP	Match Filter	PPP
1	1.436 ms	4 ms	249 ms	6 ms
2	1.450 ms	5 ms	255 ms	4 ms
3	1.468 ms	4 ms	277 ms	4 ms
4	1.436 ms	3 ms	263 ms	5 ms
5	1.441 ms	6 ms	366 ms	5 ms
Prom	1.446 ms	4 ms	282 ms	5 ms

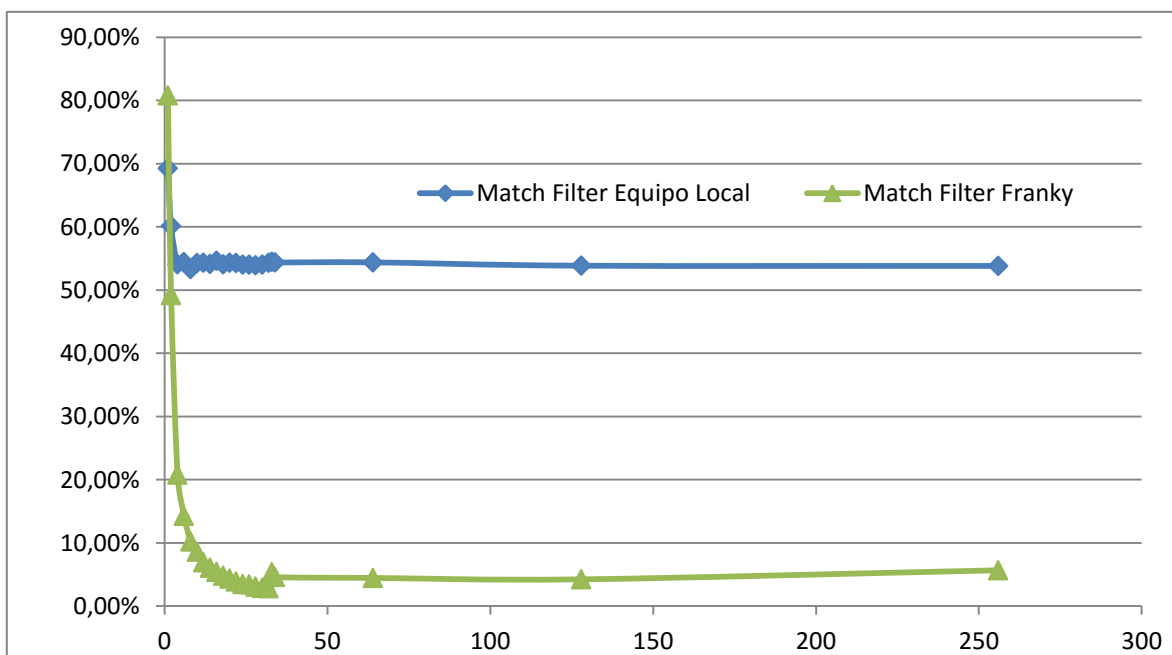
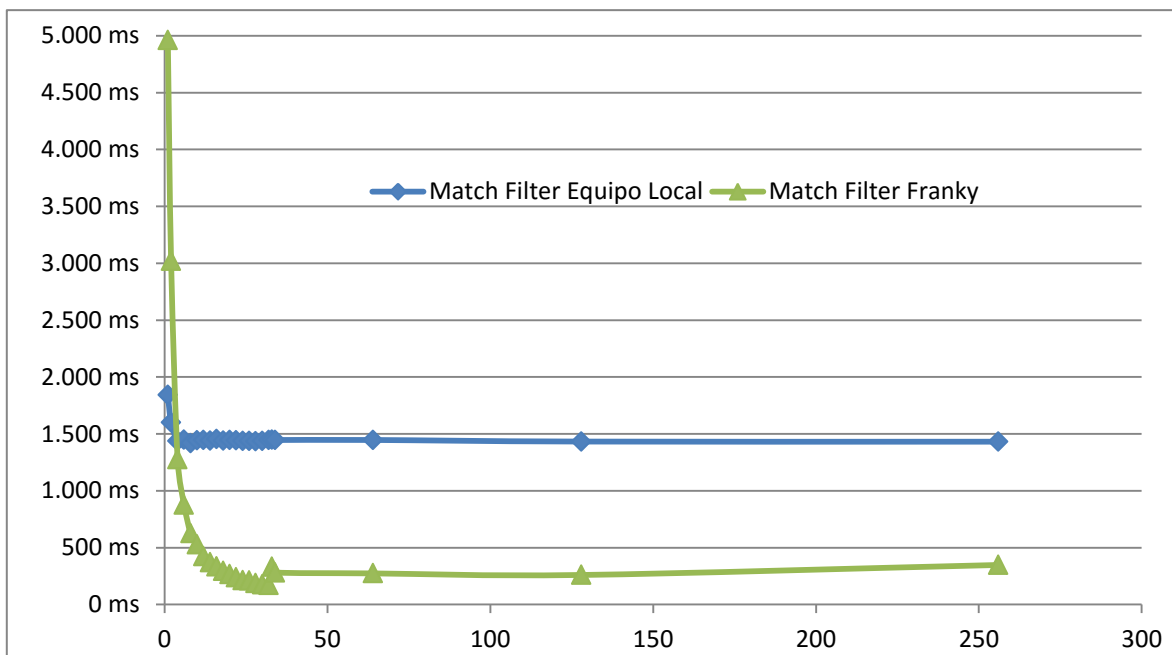
PARALELO (64 hilos)				
Run	EQUIPO LOCAL		FRANKY	
	Match Filter	PPP	Match Filter	PPP
1	1.439 ms	5 ms	402 ms	8 ms
2	1.432 ms	8 ms	254 ms	6 ms
3	1.447 ms	3 ms	244 ms	6 ms
4	1.468 ms	8 ms	248 ms	9 ms
5	1.447 ms	5 ms	221 ms	9 ms
Prom	1.447 ms	6 ms	274 ms	8 ms

PARALELO (128 hilos)				
Run	EQUIPO LOCAL		FRANKY	
	Match Filter	PPP	Match Filter	PPP
1	1.439 ms	9 ms	201 ms	11 ms
2	1.444 ms	6 ms	228 ms	10 ms
3	1.426 ms	5 ms	241 ms	12 ms
4	1.426 ms	5 ms	403 ms	10 ms
5	1.428 ms	5 ms	227 ms	10 ms
Prom	1.433 ms	6 ms	260 ms	11 ms

PARALELO (256 hilos)				
Run	EQUIPO LOCAL		FRANKY	
	Match Filter	PPP	Match Filter	PPP
1	1.429 ms	8 ms	516 ms	12 ms
2	1.434 ms	7 ms	218 ms	17 ms
3	1.428 ms	9 ms	243 ms	14 ms
4	1.439 ms	9 ms	459 ms	13 ms
5	1.428 ms	8 ms	303 ms	16 ms
Prom	1.432 ms	8 ms	348 ms	14 ms

A continuación presentamos una serie de gráficos representativos de estos datos.

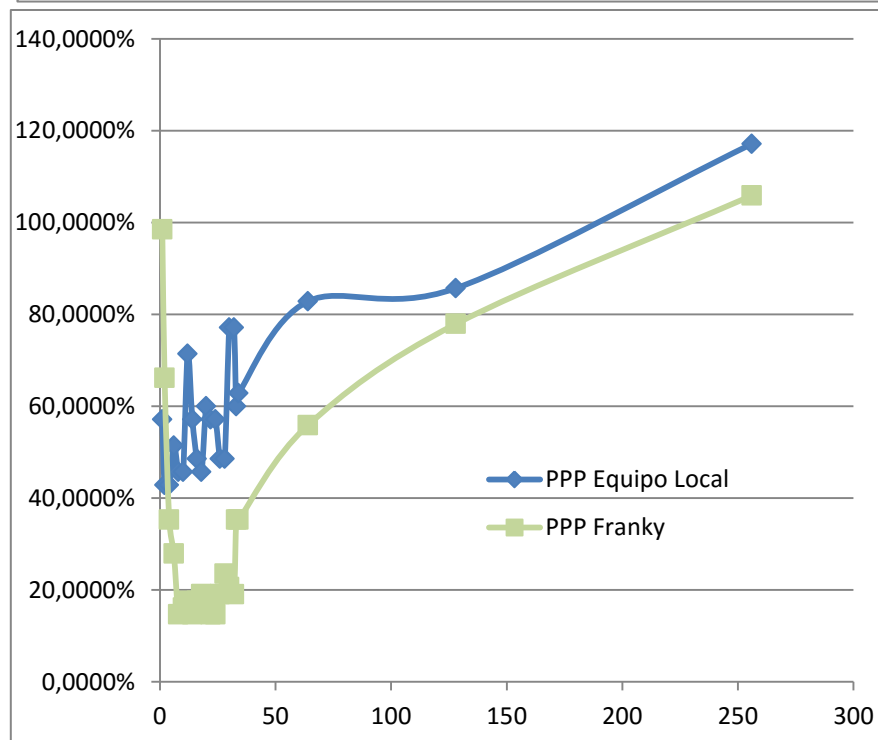
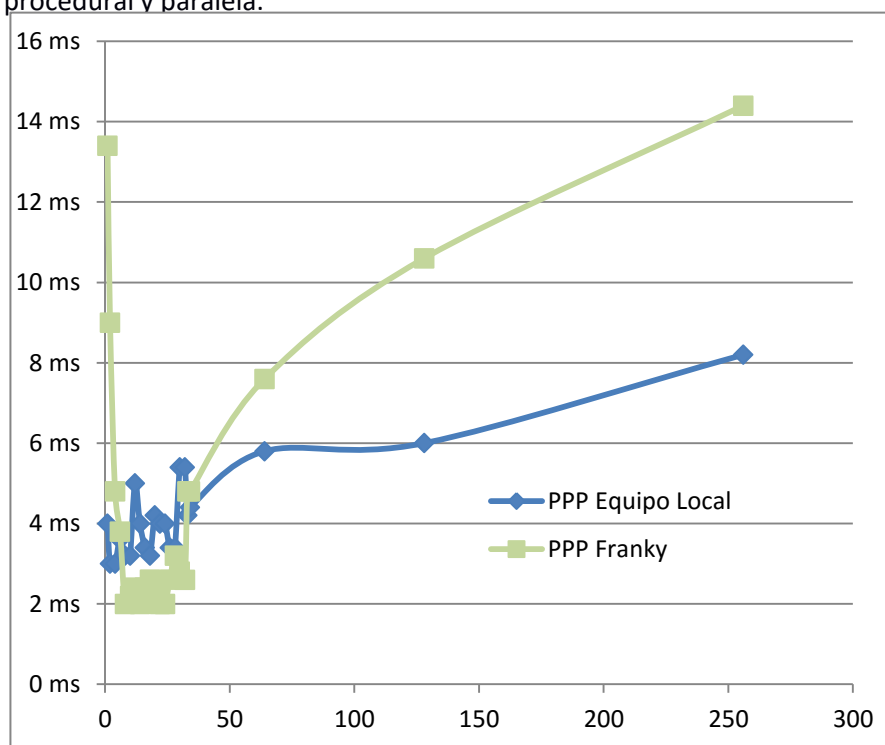
Evolución de los tiempos de la ejecución de Match Filter para ambos equipos, donde se compara ejecución procedural y paralela:



La ejecución del Match Filter es óptima cuando el número de hilos es 4 es el equipo local, y 32 en franky.

Evolución de los tiempos de la ejecución de PPP para ambos equipos, donde se compara ejecución procedural y paralela:

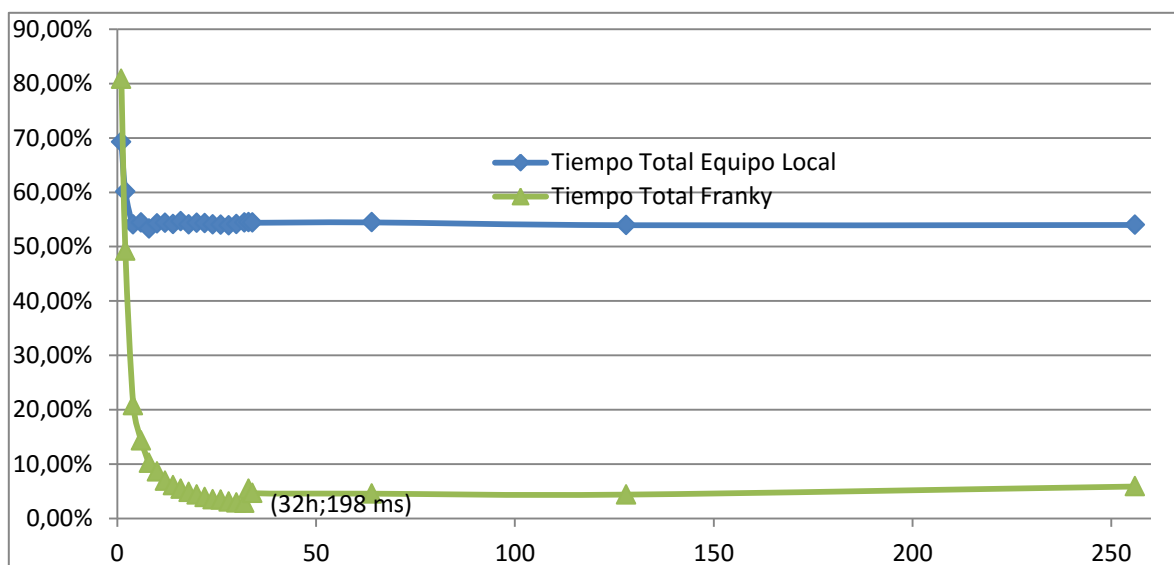
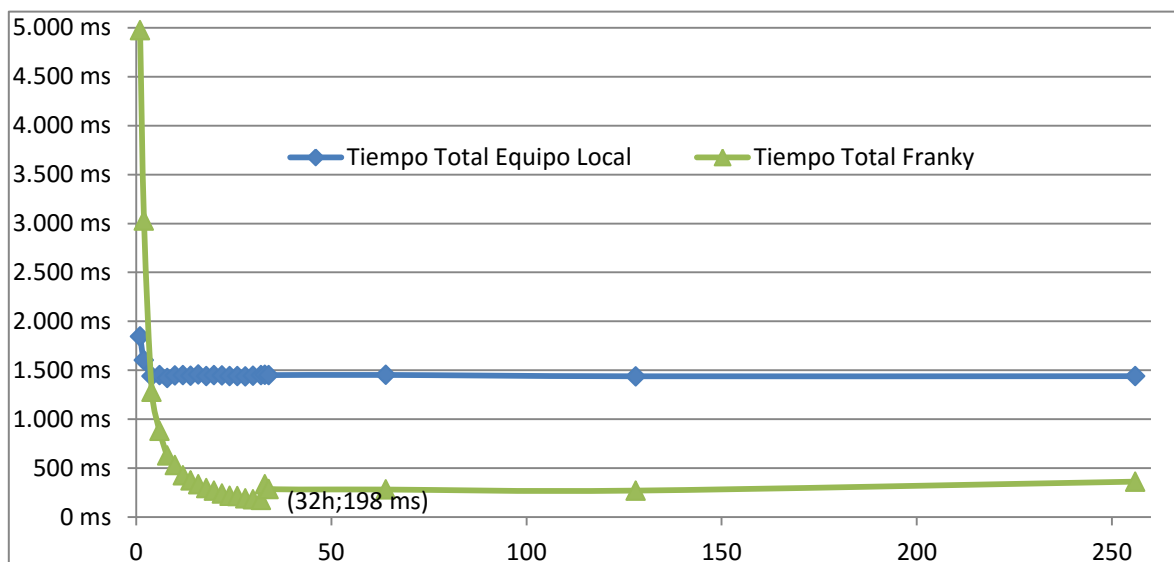
18



El procesamiento PPP disminuye en tiempo a medida que el número de hilos crece y alcanza al número de hilos físicos, aunque presenta mucha mayor variabilidad. Luego, si sigue creciendo el número de hilos programables, el tiempo aumenta exponencialmente.

Evolución de los tiempos de la ejecución Totales para ambos equipos, donde se compara ejecución procedural y paralela:

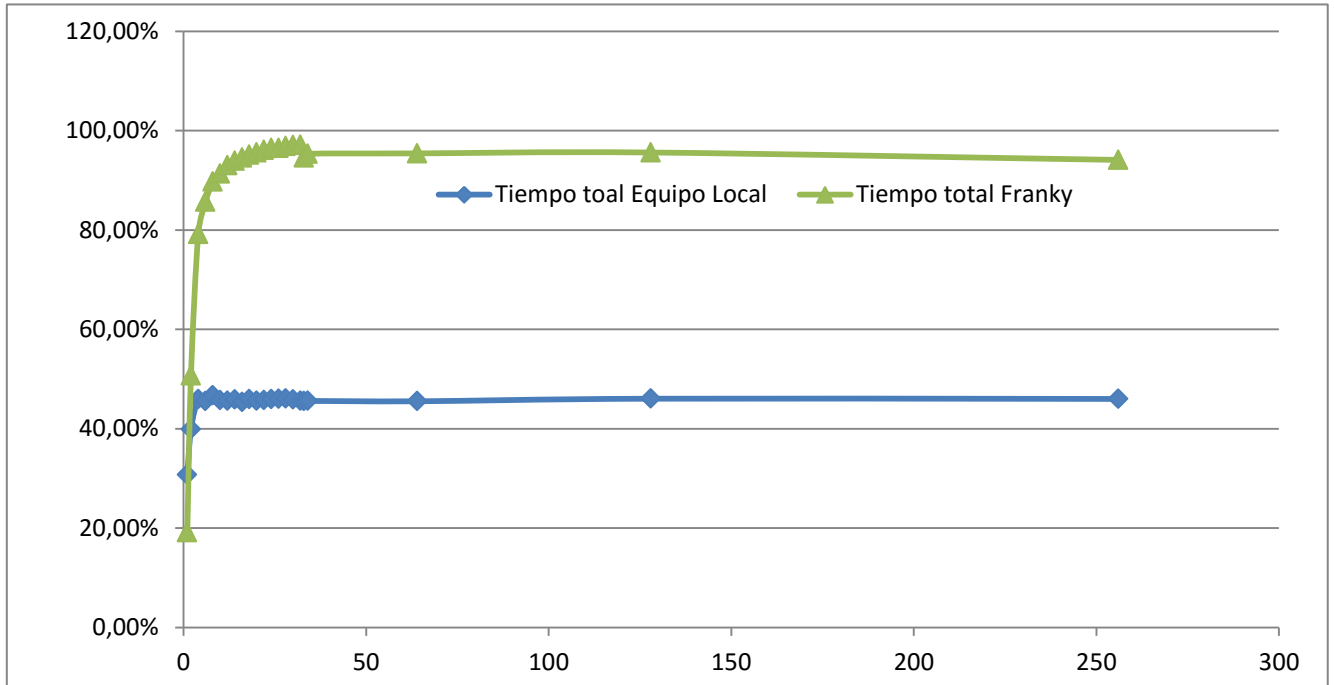
19



Finalmente podemos ver la evolución del tiempo total de ejecución del programa, donde se observa que el punto óptimo es 32 hilos, 198ms para franky, mientras que para el equipo local es 4 hilos, 3ms.

Procedemos a analizar el rendimiento

20



Donde se observa, guiados por las tablas de datos, que el punto de rendimiento óptimo es:

- **para equipo local: 4 hilos, 45,97%, 1441ms**
- **para franky es: 32 hilos, 97,16%, 172ms**

Si bien para el equipo local, hay otros puntos donde el rendimiento es levemente superior, no se los considera, ya si el número de muestras fuera cerca de 30 ejecuciones por cantidad de hilos, se podría observar que hay menos variabilidad y 4 hilos es el punto óptimo.

Los resultados se guardan en el archivo ppp.h.

Para realizar las mediciones se utilizó la herramienta que brinda la librería de OpenMP: La función **omp_get_wtime()**. OpenMP también cuenta con la función **omp_get_wtick()**, para obtener la precisión del temporizador.

Esta nos permitió tomar los tiempos en cada módulo con exactitud, para modelarlo, tabularlo y graficar.

Fuera de esto, existen herramientas de profiling, que nos permiten analizar el rendimiento de nuestras aplicaciones, como por ejemplo:

TAU: Tuning and Analysis Utilities

Un set de herramientas para profiling y trazado de programas paralelos escritos en C, C++, Fortran, y otros. Soporta compilación dinámica e instrumentación a nivel de fuente.

https://wiki.mpi4devs.org/mpich/index.php/TAU_by_example

<https://www.alcf.anl.gov/user-guides/tuning-and-analysis-utilities-tau>

<https://www.cs.uoregon.edu/research/tau/downloads.php>

<http://lsi.ugr.es/jmantas/pdp/ayuda/instalacion.php?ins=tau>

Valgrind

Es un framework para construir herramientas de análisis dinámico.

Puede detectar automáticamente muchos bugs de administración de memoria y de manejo de hilos, y analiza los programas en detalle.

<http://valgrind.org/>

ompp

Es una herramienta de profiling para aplicaciones OpenMP, que hace un reporte inmediatamente al terminar el programa en formato ASCII legible por humanos.

<http://www.ompp-tool.com/>

Intel® VTune™ Amplifier XE

Herramienta de profiling, muy útil para encontrar los cuellos de botella en los códigos de OpenMP.

Encuentra problemas de rendimiento. Hay buenas referencias de este producto, que lo muestran como útil y fácil de usar. Gratuito para software académico.

<https://software.intel.com/en-us/articles/profiling-openmp-applications-with-intel-vtune-amplifier-xe>

<https://software.intel.com/en-us/intel-vtune-amplifier-xe>

Google CPU Profiler (gperftools)

<https://github.com/gperftools/gperftools>

Scalasca

Herramienta de profiling OpenMP y MPI.

<http://www.scalasca.org/>

Allinea Map

Profiler para C/C++ y Fortran para código Linux de alto rendimiento.

<http://www.allinea.com/products/map>

<http://www.allinea.com/blog/201502/profiling-openmp-allinea-map-50>

fuentes:

<http://stackoverflow.com/questions/7181078/how-to-profile-openmp-bottlenecks>

<http://stackoverflow.com/questions/375913/what-can-i-use-to-profile-c-code-in-linux/378024#378024>

Conclusión

Este trabajo dejó la experiencia de poder realizar una misma solución de manera procedural y paralela, y ver como mejora el rendimiento en el caso de la ejecución paralela cuando esta se realiza en un equipo de procesamiento multi-hilo y memoria compartida.

Se observó que en general, cuando el número de hilos que se están ejecutando es el mismo que el número de hilos que soporta el equipo en paralelo, el rendimiento es óptimo y cada procesador se aprovecha al máximo.

A medida que el número de hilos programados aumenta y supera al número de hilos físicos que soporta el equipo, el tiempo de ejecución aumenta levemente y se estanca en un cierto punto o crece muy lentamente. Esto evidencia que no tiene sentido crear demasiados hilos si éstos no puede ejecutarse en paralelo, especialmente para este caso, en que se hacen cálculos independientes y no se necesitan esperar sincronizaciones una vez se le ha asignado la tarea al hilo. En otros casos sí, será mas productivo agregar hilos, donde se involucre otro tipo de tareas que requieran mas condiciones de sincronización.

El uso de OpenMP resultó muy sencillo de programar. Además es un estándar gratuito y multiplataforma, lo que hace de OpenMP una excelente elección para la programación paralela.