UNIVERSIDAD NACIONAL DE CÓRDOBA

Facultad de Ciencias Exactas, Físicas y Naturales

Cátedra de Sistemas Operativos II

OpenMP

Tarazi Pedro Esequiel

8 de Mayo de 2016

Índice

INTRODUCCION

OpenMP es una API para la programación multiproceso que permite añadir concurrencia y paralelismo a programas escritos en lenguajes C, C++ y Fortran. Se compone de un conjunto de directivas de compilador, rutinas y variables de entorno que modifican el comportamiento en tiempo de ejecución. Es un modelo de programación portable y escalable que proporciona a los programadores una interfaz simple y flexible para el desarrollo de aplicaciones paralelas. OpenMP usa un modelo de paralelismo fork/join, soportando además paralelismo anidado. Permite la programación multihilo en sistemas con memoria compartida. Una gran ventaja de OpenMP es que viene integrado en el compilador GCC, y por lo tanto no es necesario instalar ningún programa o aplicación en el sistema. Otra de las ventaja es que permite definir bloques paralelizables mediante etiquetas “#pragma”, por lo que dentro de un mismo programa, puede haber rutinas que se ejecuten paralelamente y otras donde se hace serialmente.

En el proyecto realizado, se usan algunas instrucciones que permiten paralelizar ciertos bloques de código, para aumentar la eficiencia y rendimiento de la ejecución.

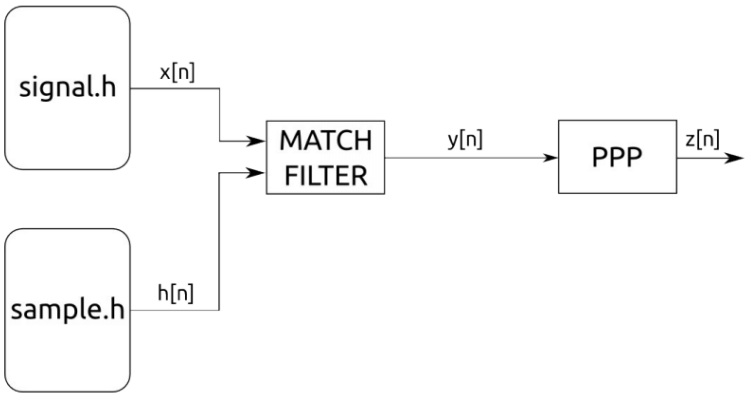
En el presente informe se realizará una breve descripción del problema, se detallarán los requerimientos, además de mostrar el diseño y la documentación del proyecto, para finalmente mostrar los resultados obtenidos.

DESCRIPCION DEL PROBLEMA

El problema consiste en un sistema radar que genera una señal(pulso) coherente. Esta señal es enviada a través de su canal de transmisión y se toma una muestra antes de ser enviada. Una vez transmitida la señal, el sistema comienza a recibir ecos provenientes de objetivos y guarda la señal recibida.

El radar envía 50 pulsos idénticos y la señal recibida la divide en 50 gates o pixeles, formando una matriz de 50x50. Cada gate se contiene 500 datos.

Se pide realizar el procesamiento de la señal recibida por el radar. Este consiste en el filtrado de una señal, mediante un MatchFilter, y luego un procesamiento de par de pulsos.



Se pide realizar un diseño que sea solución al problema sin explotar el paralelismo y luego realizar una implementación mediante el uso de la librería OpenMP. Se requiere conocer qué tipo de paralelismo exhibe el problema, y luego diseñar la solución del mismo.

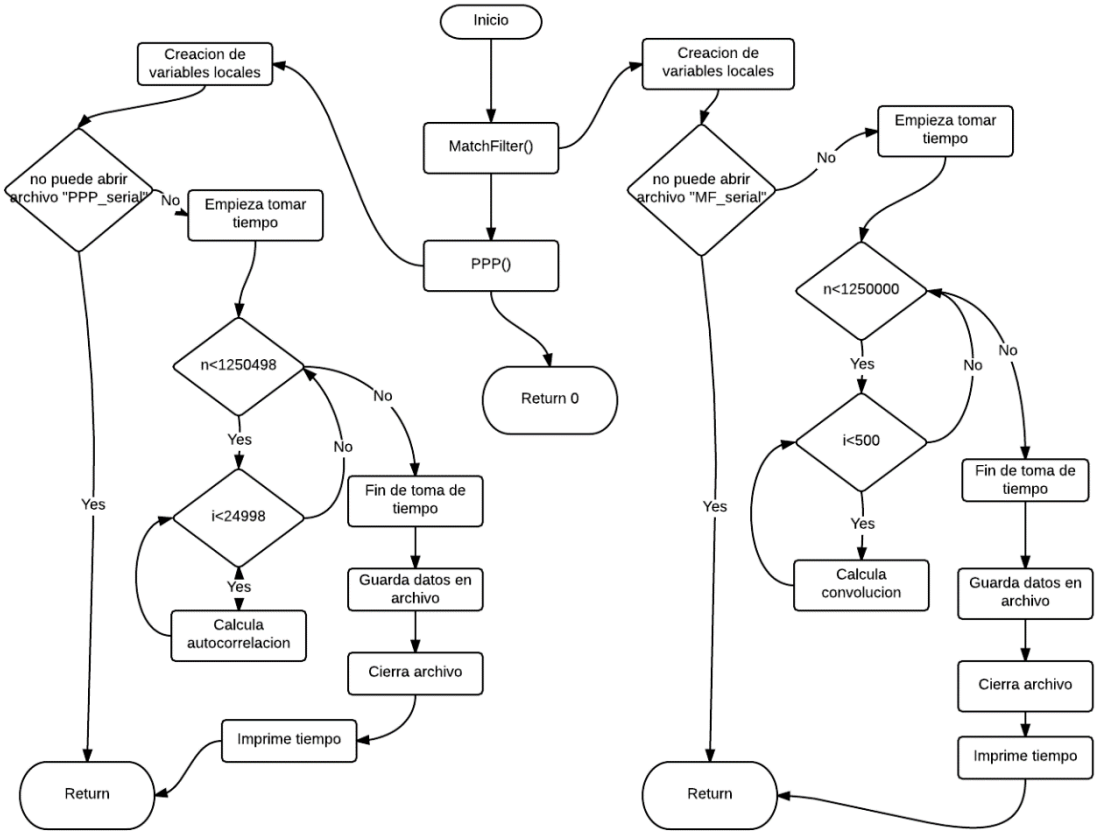
REQUERIMIENTOS Y TAREAS

- Se debe realizar filtrado de la señal mediante un Match Filter.
- Se debe realizar procesamiento de par de pulsos.
- Se debe realizar un diseño sin explotar el paralelismo.
- Se debe realizar un diseño usando la librearia OpenMP.
- Se debe especificar qué tipo de paralelismo exhibe el problema.
- Se debe elaborar un análisis con gráficos/tablas de los datos recopilados de varias ejecuciones del programa en una PC y en el cluster de la facultad.
- Se debe investigar sobre herramientas de profiling.

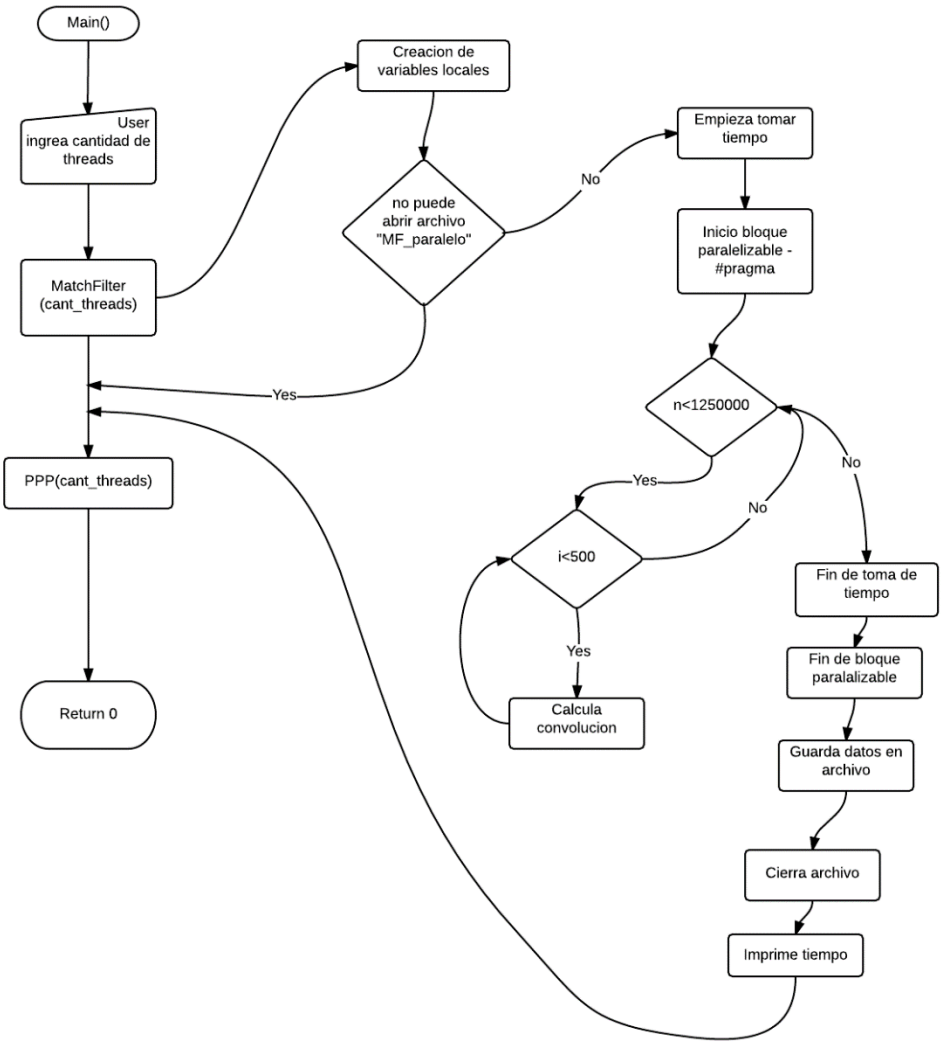
DISEÑO Y DOCUMENTACION

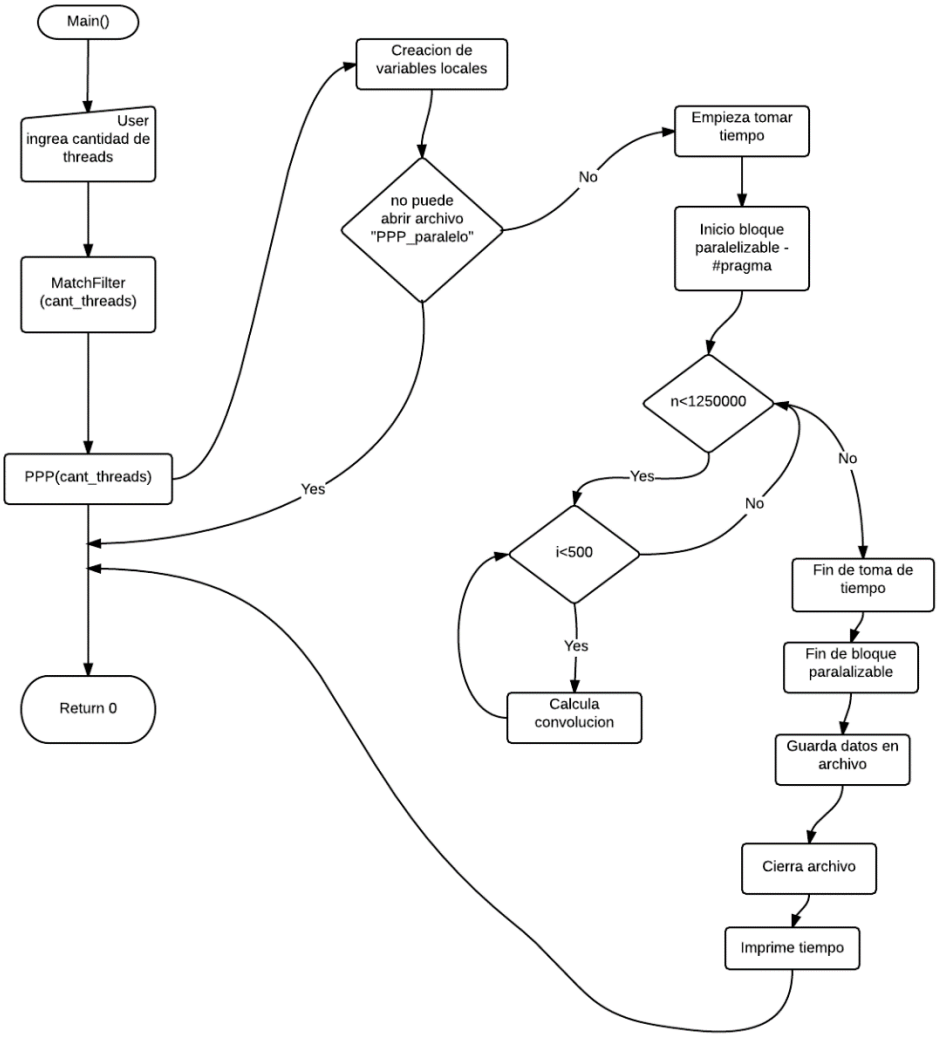
Diseño

A continuación, se muestra el diagrama de flujo del programa “serial.c”, en el cual se realiza el procesamiento de la señal de manera procedural, es decir, con un solo hilo.



En las próximas imágenes, se muestran los diagramas de flujo de las funciones MatchFilter y PPP del programa “paralelo.c”:


Main()
User ingrea cantidad de threads
MatchFilter (cant_threads)
Creacion de variables locales
no puede abrir archivo "MF_paralelo"
No
Yes
Empieza tomar tiempo
Inicio bloque paralelizable - #pragma
n<1250000
Yes
No
i<500
Yes
No
Calcula convolucion
Fin de toma de tiempo
Fin de bloque paralalizable
Guarda datos en archivo
Cierra archivo
Imprime tiempo
PPP(cant_threads)
Return 0



Documentación

El tipo de paralelismo que exhibe el programa es un paralelismo a nivel de datos, donde cada thread toma una porción de datos y realiza una acción sobre ellos. Dicha acción la hace cada uno de los threads sobre la porción de datos asignada.

Las clasulas e instrucciones usadas en el programa son:

- #pragma omp parallel private(n,i) shared(y) num_threads(cant)
  - “#pragma omp parallel” indica que el bloque encerrado por { } será realizado por una cantidad de hilos igual a la que contiene el procesador. El thread master(thread 0) lee esa instrucción y crea

tantos hilos como pueda el procesador, a menos que se le indique una cantidad específica mediante la clausula "num_threads()".
    - "private(n,i)" indica que cada uno de los threads tendrán una copia de las variables i y n.
    - "shared(y)" indica que la variable "y"será compartida por todos los threads. Esta clasusula no es obligatoria, ya que todas las variables están compartidas por defecto.
    - "num_threads(cant_threads)" le indica al thread master la cantidad de threads que debe crear.
- #pragma omp for schedule(dynamic, chunk)
    - "pragma omp for" indica que el for que esta a continuación será paralelizado. Este acepa varias clausulas que pueden modificar su comportamiento.
    - "schedule(dynamic, chunk)" indica que los datos se repartirán dinámicamente en un tamaño indicado por chunk, y una vez que un thread acabe de procesar sus datos, podrá pedir una nueva porción de datos.

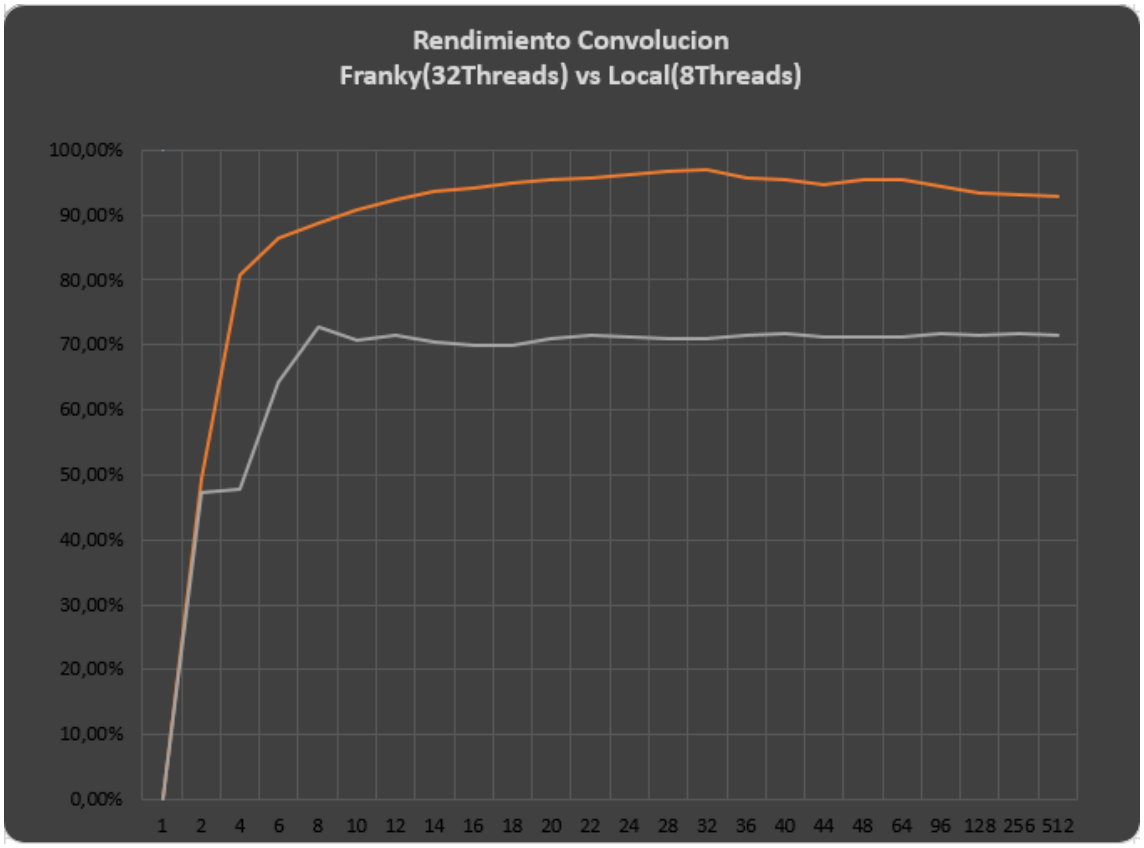
La herramienta usada para contar tiempos es la herramienta que viene implementada en la libreria "omp.h", omp_get_wtime(). Elegí esta herramienta porque tiene una precisión de hasta microsegundos, por lo que es muy buena para comparar tiempos tan pequeños. Además, es muy fácil de usar.

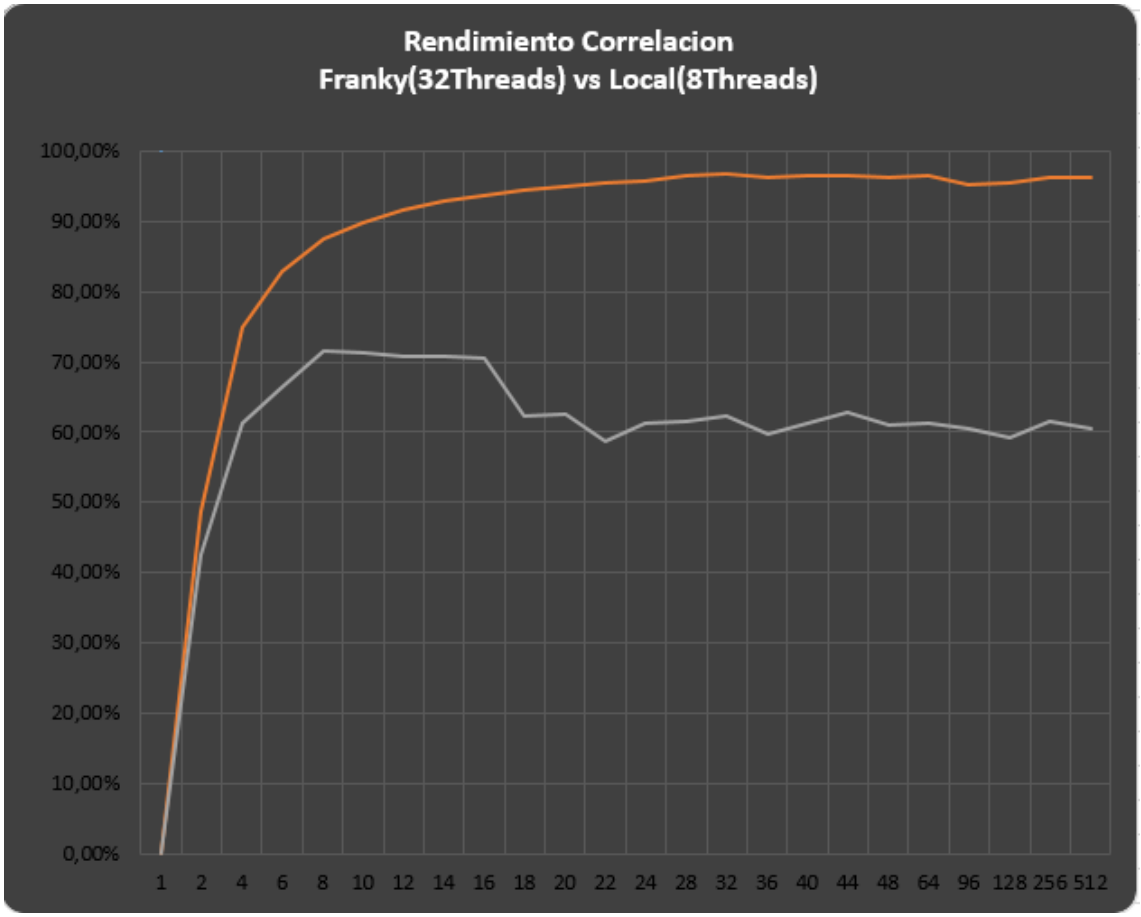
Existen otras funciones para calcular tiempos, como lo son la función clock() de la librería "time.h", o la función gettimeofday de la librearia "sys/time.h", pero eran tan precisas como la función elegida.

IMPLEMENTACION Y RESULTADOS

Se realizaron 5 ejecuciones del programa para distintos números de hilos. El número de ejecuciones fue pequeño debido a que contamos con un cluster para 50 alumnos, y si hubiese realizado muchas más, otros no podrían usarlo, ya que los tiempos se miden cuando el cluster está corriendo solo un programa a la vez. Dicho cluster cuenta con 32 threads, por lo que se supone que la ejecución del programa tendrá su pico de rendimiento cuando se usen todos los threads disponibles.

A continuación, se mostrarán las tablas con todas las mediciones tomadas, y una gráfica donde se puede observar que el pico de rendimiento es cuando se usan todos los threads de hardware disponibles. Cuando el Sistema Operativo empieza a crear threads mediante software, se puede ver que el rendimiento empieza a disminuir. En el anexo se encuentra las tablas con los datos tomados:




Rendimiento Correlacion
Franky(32Threads) vs Local(8Threads)
100,00%
90,00%
80,00%
70,00%
60,00%
50,00%
40,00%
30,00%
20,00%
10,00%
0,00%
1 2 4 6 8 10 12 14 16 18 20 22 24 28 32 36 40 44 48 64 96 128 256 512

CONCLUSIONES

Durante la construcción del proyecto, pude ir conociendo la herramientas e instrucciones que permiten paralelizar un código de una manera rápida y sencilla. Existen una gran cantidad de instrucciones y cláusulas que hacen que el código se ejecute de la manera en que nosotros queremos, y que nos permiten optimizar porciones del mismo en el que se analizan grandes cantidades de datos.

Pude aprender que lo más común es paralelizar un ciclo “for” donde se recorren grandes volúmenes y se realizan cálculos que son una gran carga para el procesador, y que paralelizándolos, se optimiza el uso del mismo. Otra cosa que noté, es que volcar grandes cantidades de datos en un archivo, es una tarea muy complicada para que la hagan varios threads a la vez, ya que los datos llegan de manera desordenada, y esto puede ocasionar errores muy graves en aplicaciones de la vida real, por lo que decidí no paralelizar esa parte del trabajo.

Un problema que tuve durante el trabajo, fue que cuando paralelizaba, los datos tenían diferencias en el orden del micro. No pude solucionar ese error, pero al ser tan pequeño, no me pareció significativo. De todas maneras, el porcentaje de error era mínimo respecto a la gran cantidad de datos que procesaba.

ANEXO

CONVOLUCION - FRANKY(32 Threads)							
Hilos	1	2	3	4	5	Promedio	Rendimiento
1	5,89312	5,88917	5,88377	5,88237	5,88392	5,88647	0,0000%
2	2,99015	2,9828	2,97748	2,96076	2,96365	2,974968	49,4609%
4	1,14478	1,12648	1,12672	1,1286	1,12642	1,1306	80,7932%
6	0,82369	0,81572	0,81421	0,80853	0,75977	0,804383	86,3350%
8	0,66511	0,65223	0,65397	0,65314	0,65416	0,655719	88,8606%
10	0,54113	0,52432	0,53679	0,54129	0,53192	0,535091	90,9098%
12	0,44561	0,43714	0,44254	0,45278	0,44239	0,444093	92,4557%
14	0,38237	0,37682	0,37734	0,38264	0,3783	0,379492	93,5531%
16	0,3589	0,32867	0,35415	0,33789	0,33793	0,343506	94,1645%
18	0,29923	0,31391	0,3048	0,29845	0,29907	0,303091	94,8510%
20	0,27177	0,28801	0,2726	0,26785	0,27031	0,274109	95,3434%
22	0,26438	0,24397	0,2626	0,24366	0,24323	0,251569	95,7263%
24	0,22712	0,22626	0,22785	0,2252	0,2234	0,225964	96,1613%
28	0,20231	0,19707	0,19866	0,19503	0,19348	0,19731	96,6481%
32	0,17582	0,1732	0,17043	0,17246	0,1751	**0,173401**	**97,0543%**
36	0,2369	0,24556	0,25303	0,24544	0,27534	0,251253	95,7317%
40	0,23745	0,26438	0,24236	0,24971	0,34938	0,268654	95,4361%
44	0,27198	0,24039	0,3571	0,40134	0,27386	0,308934	94,7518%
48	0,32081	0,24336	0,29081	0,22772	0,22608	0,261755	95,5533%
64	0,21129	0,24639	0,35217	0,28488	0,24739	0,268421	95,4400%
96	0,30086	0,34957	0,35039	0,30417	0,3008	0,321157	94,5441%
128	0,31545	0,28321	0,569	0,45523	0,29341	0,38326	93,4891%
256	0,53985	0,3052	0,42628	0,27819	0,4471	0,399322	93,2163%
512	0,25953	0,53426	0,57862	0,38676	0,35599	0,423033	92,8135%

CORRELACION - FRANKY(32 Threads)							
Hilos	1	2	3	4	5	Promedio	Rendimiento
1	215,098	215,109	215,094	215,086	215,106	215,0986	0,0000%
2	109,013	108,876	108,927	112,258	112,378	110,2904	48,7257%
4	54,0618	53,9325	53,8861	53,8839	54,1031	53,97348	74,9076%
6	37,5233	37,4364	37,4858	36,2999	36,2997	37,00902	82,7944%
8	26,8811	27,0785	27,1041	27,1167	27,0617	27,04842	87,4251%
10	21,7674	21,6673	21,6182	21,8432	21,7269	21,7246	89,9002%
12	18,0774	17,9274	19,5275	18,0414	18,1076	18,33626	91,4754%
14	15,5009	15,5119	15,5139	15,5131	15,4856	15,50508	92,7916%
16	13,62	13,5607	13,6284	13,5711	13,5511	13,58626	93,6837%
18	12,0485	12,0112	12,1027	12,0971	12,2388	12,09966	94,3748%
20	10,7487	10,9445	10,8502	10,8809	10,8569	10,85624	94,9529%
22	9,90604	9,83109	9,94706	9,87001	9,87695	9,88623	95,4039%
24	9,03281	9,04984	9,04692	9,03602	9,03021	9,03916	95,7977%
28	7,71156	7,74281	7,74067	7,72806	7,7772	7,74006	96,4016%
32	6,79106	6,77579	6,78364	6,77595	6,77299	**6,779886**	**96,8480%**
36	8,22893	8,32859	8,0685	8,07893	8,11126	8,163242	96,2049%
40	7,91898	7,82724	7,85625	7,56722	7,61488	7,756914	96,3938%
44	7,35926	7,47094	7,46109	7,33068	7,37304	7,399002	96,5602%
48	7,55442	7,61039	9,26196	7,50829	7,55446	7,897904	96,3282%
64	7,15135	6,95155	7,99923	7,36597	7,10769	7,315158	96,5992%
96	10,5454	10,3401	10,4202	10,3128	9,97939	10,319578	95,2024%
128	9,80265	9,62204	9,48282	9,77548	9,87655	9,711908	95,4849%
256	6,85003	8,12892	9,16778	8,68831	9,04113	8,375234	96,1063%
512	6,78049	7,74667	8,41262	8,50934	8,85711	8,061246	96,2523%

CONVOLUCION - LOCAL(8 Threads)							
Hilos	1	2	3	4	5	Promedio	Rendimiento
1	1,8937	1,94207	1,95581	1,96637	1,97181	1,945952	0,0000%
2	1,01528	1,01983	1,04889	1,00223	1,04126	1,025498	47,3010%
4	1,0139	0,996025	1,03889	1,01453	1,01675	1,016019	47,7881%
6	0,714911	0,676274	0,694403	0,687495	0,690783	0,6927732	64,3993%
8	0,528622	0,530538	0,521967	0,54158	0,528985	**0,5303384**	**72,7466%**
10	0,581141	0,584117	0,576037	0,562898	0,541643	0,5691672	70,7512%
12	0,566581	0,546583	0,551821	0,544313	0,562225	0,5543046	71,5150%
14	0,562264	0,5842	0,575337	0,571986	0,581164	0,5749902	70,4520%
16	0,601868	0,581291	0,571601	0,591538	0,57229	0,5837176	70,0035%
18	0,579358	0,572058	0,59236	0,589851	0,586267	0,5839788	69,9901%
20	0,570442	0,565332	0,56434	0,569604	0,550884	0,5641204	71,0106%
22	0,546484	0,552495	0,551511	0,556099	0,562135	0,5537448	71,5438%
24	0,572074	0,555357	0,542151	0,575332	0,550875	0,5591578	71,2656%
28	0,567668	0,575399	0,565003	0,552796	0,552796	0,5627324	71,0819%
32	0,559995	0,567058	0,568568	0,558091	0,571363	0,565015	70,9646%
36	0,546274	0,55808	0,546338	0,567725	0,56447	0,5565774	71,3982%
40	0,544502	0,552674	0,562505	0,554076	0,542298	0,551211	71,6740%
44	0,549942	0,553313	0,555779	0,562295	0,575325	0,5593308	71,2567%
48	0,555765	0,556034	0,570943	0,544255	0,5618	0,5577594	71,3375%
64	0,547229	0,576411	0,570172	0,553307	0,563685	0,5621608	71,1113%
96	0,549915	0,542899	0,547945	0,565749	0,530089	0,5473194	71,8740%
128	0,560196	0,542953	0,553962	0,550848	0,564448	0,5544814	71,5059%
256	0,543538	0,553495	0,545482	0,547338	0,551039	0,5481784	71,8298%
512	0,535096	0,554015	0,572726	0,563104	0,545156	0,5540194	71,5296%

CORRELACION - LOCAL(8 Threads)							
Hilos	1	2	3	4	5	Promedio	Rendimiento
1	95,7356	92,8514	95,0696	97,9371	97,4058	95,7999	0,0000%
2	53,4534	55,4801	59,3586	51,6082	55,1186	55,00378	42,5847%
4	35,645	36,9293	38,8399	32,2373	41,8337	37,09704	61,2765%
6	28,9564	33,6256	34,2524	32,4749	32,122	32,28626	66,2982%
8	27,8992	26,6232	26,9099	27,0706	28,2066	**27,3419**	**71,4594%**
10	27,6222	27,1721	26,1573	28,0874	28,1146	27,43072	71,3667%
12	27,3253	27,1178	27,6871	28,2268	29,0382	27,87904	70,8987%
14	28,1437	28,367	27,729	28,4815	27,4578	28,0358	70,7350%
16	28,416	28,8644	27,2748	28,6565	27,7092	28,18418	70,5802%
18	36,0359	35,7087	36,4306	36,1504	36,64	36,19312	62,2201%
20	35,7615	35,859	35,7803	35,7606	35,9136	35,815	62,6148%
22	40,0512	38,8305	38,4077	41,2286	39,4526	39,59412	58,6700%
24	38,3004	39,5653	36,6306	35,672	35,717	37,17706	61,1930%
28	36,9753	35,9245	36,7608	38,6437	35,9939	36,85964	61,5243%
32	33,0643	39,2449	35,7376	35,7604	36,6464	36,09072	62,3270%
36	40,288	39,8223	36,417	35,9186	40,6708	38,62334	59,6833%
40	35,7621	37,1336	36,236	35,7684	40,1291	37,00584	61,3717%
44	39,2631	40,1314	36,9888	31,0323	30,7257	35,62826	62,8097%
48	36,6854	38,3861	36,3397	36,5646	38,4906	37,29328	61,0717%
64	39,6741	36,9279	36,5901	36,0092	35,9364	37,02754	61,3491%
96	35,5662	37,6126	36,115	39,425	40,0217	37,7481	60,5969%
128	37,198	41,1882	41,135	36,2651	39,5552	39,0683	59,2189%
256	35,6511	35,7197	38,4907	35,6633	39,2381	36,95258	61,4273%
512	36,5082	39,1528	35,7609	39,7113	37,6695	37,76054	60,5839%