Universidad Nacional de Córdoba

Cátedra de Sistemas Operativos II

FACULTAD DE CIENCIAS EXACTAS, FÍSICAS Y NATURALES

Trabajo Práctico N°2

Alumno: Oliva Arias, Carlos Agustín
Fecha: 15 de Mayo de 2019

Índice

1. Introducción

1.1. Propósito

El objetivo del presente trabajo práctico es que el estudiante sea capaz diseñar una solución que utilice el paradigma de memoria distribuida, utilizando OpenMP.

1.2. Ámbito del sistema

El práctico se realiza sobre dos sistemas, primero sobre la notebook del alumno, cuyas especificaciones son las siguientes:

- Computadora de propósito general
- Procesador: Intel(R) Core(R) i7-7500U CPU @ 2.70 GHz x 2
- Memoria: 7.7 GiB DDR4 Synchronous 2400 MHz

Luego la ejecución se realiza sobre un nodo del cluster Yaku del Lab de Hidraúlica, cuyas especificaciones son:

- Nodo de cálculo
- Procesador: Intel(R) Xeon(R) CPU E5-2683 v4 @ 2.10GHz 40 MB cache (2 CPU x 16 cores = 32 cores)
- Placa madre: SuperMicro X10DDW-i
- Memoria: 64 GB - DDR4 - 2400 Mhz ECC

1.3. Definiciones, Acrónimos y Abreviaturas

NetCDF: Network Common Data Form (Forma común de datos en red)
CMI: Cloud and Moisture Image (Imágen de nubes y humedad)

1.4. Referencias

- http://200.16.30.250/equipo.html
- https://ark.intel.com/content/www/us/en/ark/products/120492/intel-xeon-gold-6130-processor-22m-cache-2-10-ghz.html
- https://www.unidata.ucar.edu/software/netcdf/netcdf-4/newdocs/netcdf/ncdump.html
- https://stackoverflow.com/questions/8864599/convert-netcdf-to-image

1.5. Descripción general del documento

Las siguientes secciones describen el proceso de realización del trabajo, las herramientas y archivos auxiliares utilizados.

2. Descripción general

El programa debe tomar un archivo en el formato NetCDF y aplicarle un filtro de borde a su variable CMI, que consta de una matriz de 21696 * 21696, donde los valores corresponden al brillo de una imagen monocromática.

2.1. Restricciones

El resultado del procesamiento debe guardarse en otro archivo, por lo que se opta por un archivo del mismo formato, donde se guardan los nuevos datos de la variable CMI. Esta restricción esta dada por el tamaño final del archivo, ya que, mientras un archivo binario con los shorts necesarios tiene un peso de 1.7GB aproximadamente, el archivo final en formato NetCdf tiene un peso de la mitad aproximadamente.

2.2. Suposiciones y dependencias

Para poder utilizar el programa se necesitan las siguientes librerías:

- Netcdf
- zlib
- hdf5
- OpenMP

Donde las primeras tres son provistas por el script de instalación entregado, y la última esta disponible en el sistema operativo utilizado.

3. Diseño de solución

Para la solución utilizamos un código fuente esqueleto, que ponía a disposición la matriz a partir del archivo, y al cual solo hacia falta realizar la convolución. A la hora de verificar si la imagen era visible, se encontró una solución en python que muestrea la imagen para visualizarla en un archivo de menor tamaño. Ahí se pudo verificar dos problemas:

- que solo se veía el cuarto inferior de la imagen
- El brillo de la imagen no era el adecuado

Se imprimió en pantalla algunos valores de los datos ingresados, pudiendo observarse que ninguno tenia parte decimal, a lo cual se investigo con la herramienta ncdump el archivo.

El análisis confirmo que se contaba con una matriz de shorts, y que los valores afuera del planeta eran -1. Esto permitió modificar el código, resolviendo el primer problema expuesto.

La matriz filtro provista no se encuentra normalizada (esto es que la suma de sus componentes sea 1) entonces al aplicárselo a los datos se modifica el valor de brillo resultante, a lo cual se encontraron dos posibles soluciones, por un lado dividir la matriz por el valor de la sumatoria, y por otro modificar el valor central por un 9, permitiendo conservar la calidad de short de las variables. Así se resolvió el segundo problema.

El procesamiento de la imagen requiere de 4 for-loops anidados, siendo los dos más externos correspondientes al pixel sobre el que se trabaja y los dos interiores necesarios para la suma de convolución, vemos como los dos externos se pueden ejecutar como un bucle rectangular perfecto, mientras que los interiores necesitarían compartir, o reducir, la variable de acumulación. Entonces podemos paralelizar el procesamiento de pixeles simplemente con la siguiente directriz del compilador:

```
#pragma omp parallel for collapse(2)
```

4. Implementación y resultados

La implementación final, esta compuesta por un archivo fuente en C, un makefile y algunos scripts varios.

Una muestra de la imagen original y de la filtrada pueden verse a continuación:

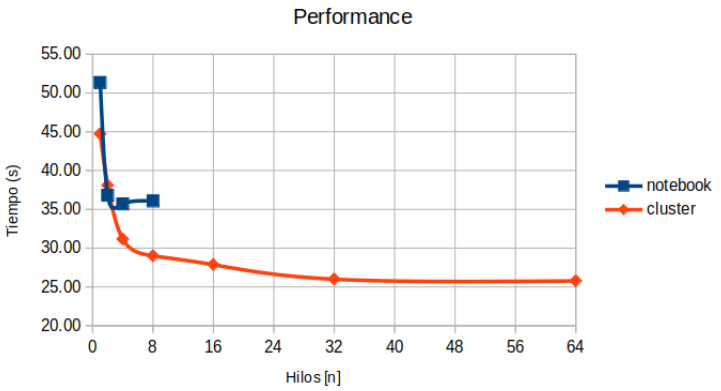
Imagen original

Imagen filtrada

Se escribió un script para la realización de pruebas de tiempo de ejecución, el mismo utiliza el comando time, para las diferentes plataformas y cantidad de hilos a utilizar durante la convolución. Además se utilizo el script en python mencionado en el apartado anterior.

Los resultados de todas las ejecuciones requeridas se encuentran en la siguiente tabla:

N	Cantidad de Threads										
	Notebook				Cluster						
	1	2	4	8	1	2	4	8	16	32	64
1	56.74	36.48	36.54	35.88	46.22	33.91	33.15	27.39	27.29	28.67	26.73
2	51.08	36.19	36.11	37.36	45.43	34.01	31.13	28.02	26.32	25.74	25.46
3	50.59	36.41	35.48	35.55	44.48	35.8	32.68	29.69	26.07	27.54	25.61
4	51.3	37.16	35.36	35.68	44.47	35.63	33.1	28.95	25.91	26.47	25.44
5	50.99	37	35.42	38.24	45.33	36.3	30.88	29.11	25.76	26.51	25.17
6	50.64	35.43	34.58	35.65	43.89	35.97	30.81	29.16	26.28	26.64	25.1
7	51.14	37.03	35.55	36.43	43.69	37.04	30.97	29.25	27.32	25.31	25.08
8	51.68	36.35	35.16	37.53	45.55	35.83	30.8	29.32	26.57	26.04	25.2
9	51.07	35.68	35.18	35.87	45.93	34.69	30.84	31.08	27.32	27.12	26.22
10	51.1	39.96	35.13	35.72	45.42	34.47	30.33	32.16	27.84	29.54	25.91
11	51.35	36.5	35.32	35.28	46.63	35.99	30.34	30.13	27.32	24.81	25.44
12	51.21	36.22	35.05	36.63	44.36	34.16	30.86	29.25	30.4	26.44	25.33
13	51.19	36.83	35.74	35.6	46.97	36.01	30.89	28.34	26.9	25.68	27.37
14	51.06	36.51	35.61	36.98	47.26	36.56	30.88	29.44	27.06	25.77	26.1
15	51.05	38.49	35.15	36.62	46.11	35.5	31.01	29.46	26.65	24.87	25.29
16	51.07	37.16	36.66	37.05	44.19	36.84	31.25	27.99	27.04	24.71	25.82
17	52.03	35.41	36.02	35.47	43.79	35.84	31.65	28.32	25.78	25.12	26.38
18	51.3	37.18	35.72	36.17	43.51	35.09	31.38	28.7	26.49	27.21	26.32
19	51.71	36.45	35.14	37.33	43.53	36.54	31.37	28.89	29.08	26.86	25.45
20	51.21	36.47	35.57	35.93	43.67	40.35	32.25	27.77	29.32	25.68	26.49
21	52.26	36.27	35.78	36.31	43.52	35.38	31.5	29.57	27.52	24.96	26.87
22	51.42	37.39	35.4	37.3	43.55	40.79	32.54	28.21	29.47	25.79	25.62
23	51.1	37.99	36.05	35.46	43.56	34.52	30.9	27.94	28.92	25.94	26.09
24	51.39	36.92	34.79	36.83	43.56	58.27	30.48	27.69	28.51	26.05	24.79
25	51.07	36.56	36.18	35.08	45.79	35.73	30.21	28.15	29.44	28.75	24.67
26	51.09	36.52	36.13	35.4	45.96	57.27	31.71	28.5	30	26.21	24.72
27	51.71	36.34	37.35	35.28	44.93	36.96	31.89	28.83	27.05	25.91	25.82
28	51.06	37.82	35.05	35.26	44.23	36.75	31.31	29.32	26.8	25.35	26.28
29	51.48	35.91	37.44	35.23	43.88	36.8	31.6	28.67	27.96	24.74	26.12
30	51.4	35.32	35.73	34.94	43.84	36.61	31.07	29.04	27.97	24.96	25.17



5. Conclusiones

Con los resultados de ejecución, se observa como la incorporación de threads para el procesamiento consigue menores tiempos de ejecución, en tanto la cantidad de los mismos no supere las capacidades del hardware.

Debido al uso del comando time, que mide el tiempo de toda la ejecución del programa, los tiempos encontrados incluyen porciones de código que no fueron optimizadas con paralelismo, tales como lo relativo a lectura/escritura del archivo .nc, y que afectan negativamente a los tiempos mostrados.

Aún así el trabajo fue una buena primera experiencia de aprendizaje sobre paralelismo en C, herramientas de profiling y el uso de la libreria Netcdf en ciencia.

6. Apéndices

6.1. Código

```
#include <stdlib.h>
#include <stdio.h>
#include <netcdf.h>
#include <omp.h>

/* Handle errors by printing an error message and exiting
   with a
 * non-zero status. */
#define ERRCODE 2
#define ERR(e) {printf("Error: %s\n", nc_strerror(e)); exit(
   ERRCODE);}
```

```
10
11 /* nombre del archivo a leer */
12 #define FILE_NAME "OR_ABI-L2-CMIPF-
       M6C02_G16_s20191011800206_e20191011809514_c20191011809591.
       nc"
13
14 #define OUT_FILE "file.nc"
15
16 /* Lectura de una matriz de 21696 x 21696 */
17 #define NX 21696
18 #define NY 21696
19 #define filter_width 3
20 short filter[3][3] = {{ -1, -1, -1,},
21                       { -1,  9, -1,},
22                       { -1, -1, -1}};
23 int half_fw = (int) filter_width/2;
24
25 int main(int argc, char *argv[] )
26 {
27     if(argc != 2) exit(-4);
28
29     int ncid, varid;
30     int ncid2, varid2;
31     short *data_in = calloc (1,sizeof(short)*NX*NY);
32     short *data_out = calloc (1,sizeof(short)*NX*NY);
33     int retval;
34     int retval2;
35
36     if ((retval = nc_open(FILE_NAME, NC_NOWRITE, &ncid)))
37         ERR(retval);
38     /* Obtenemos el varID de la variable CMI. */
39     if ((retval = nc_inq_varid(ncid, "CMI", &varid)))
40         ERR(retval);
41     if ((retval2 = nc_open(OUT_FILE, NC_WRITE, &ncid2)))
42         ERR(retval2);
43     /* Obtenemos el varID de la variable CMI. */
44     if ((retval2 = nc_inq_varid(ncid2, "CMI", &varid2)))
45         ERR(retval2);
46     /* Leemos la matriz. */
47     if ((retval = nc_get_var_short(ncid, varid, data_in)))
48         ERR(retval);
49
50     /* aplico el filtro */
51     int thread_count = atoi( argv[1] );
52     printf("Se hace con %d hilos.\n",thread_count);
53     omp_set_num_threads(thread_count);
54     int i,j;
55     short sum;
56     #pragma omp parallel for collapse(2) private(i,j,sum)
```

```
57      for(i = 0; i < NX; i++){
58        for(j = 0; j < NY; j++){
59          if(data_in[i*NX+j]==-1){
60            data_out[i*NX+j]=-1;
61          }else{
62            sum =0;
63            for(int k= -half_fw; k<= half_fw ;k++){
64              for(int l= -half_fw ; l<= half_fw ;l++){
65                int r = i+k;
66                int c = j+l;
67                r= (r<0) ? 0: r;
68                c= (c<0) ? 0: c;
69                r = (r>= NY) ? NY-1 :r;
70                c= (c>=NX) ? NX-1 :c;
71                sum += data_in[r*NX+c]*filter[k+half_fw][l+
                      half_fw];
72              }
73            }
74            data_out[i*NX+j]=sum;
75          }
76        }
77      }
78
79      /*Para guardar la imagen en el mismo formato utilizo */
80      if ((retval2 = nc_put_var(ncid2, varid2, data_out)))
81        ERR(retval2);
82
83      /* Se cierra el archivo y liberan los recursos*/
84      if ((retval = nc_close(ncid)))
85          ERR(retval);
86          free(data_in);
87          free(data_out);
88      return 0;
89  }
```

readNetcdf.c

```
1  SHELL := /bin/bash
2
3  compile:
4   gcc -o readNetcdf.o readNetcdf.c -O3 -lm -lnetcdf -fopenmp -
       Wall -Werror -pedantic
5  test:
6   rm file.nc -f
7   cp OR_ABI-L2-CMIPF-
       M6C02_G16_s20191011800206_e20191011809514_c20191011809591
       .nc file.nc
8   time ./readNetcdf.o 1
9  clean:
```

```
10  rm *.o -f
11  rm run_*.txt -f
12 check:
13  cppcheck readNetcdf.c --enable=all
14 show:
15  python3 show.py
16 profile:
17  gcc readNetcdf.c -o profiling.o -g -pg -std=c99 -O3 -lm -
        lnetcdf -fopenmp -Wall -Werror -pedantic
```

Makefile

```
#!/bin/bash
for i in $(seq 0 3);
do
 j=$((2 ** $i))
 for k in $(seq 0 30);
 do
    cp OR_ABI-L2-CMIPF-
       M6C02_G16_s20191011800206_e20191011809514_c20191011809591
       .nc file.nc
  { /usr/bin/time ./readNetcdf.o $j ; } 2>>
     time_parallel_notebook_$j.txt
    rm file.nc
  t=$((k + 1))
  echo pasada n$k
 done
  echo termina con $j hilos
done
```

Script de ejecución

6.2. Profiling

Herramientas de Profiling: Se estudiaron varias herramientas de profilling gratuitas y disponibles para el sistema operativo utilizado (basado en Ubuntu), entre ellas:

- Vtune
- Gprof
- Valgrind

Primeramente se optó por la utilización de gprof que es la más liviana de ellas. Con la herramienta, se confeccionó una tabla que muestra información sobre la ejecución discriminada por líneas del código fuente, pero los tiempos

registrados eran significativamente menores a los que se tomaron como muestra, a lo que luego de la lectura de la documentación se determinó que esa opción no era compatible con la versión de gcc utilizada (7.4.0).

Posteriormente se utilizó vtune, la cual permitio obtener más información.

```
Elapsed Time: 42.822s
    CPU Time: 36.223s
    Total Thread Count: 8
    Paused Time: 0s
```

Tiempo ejecución (profiling)

Function / Call Stack	CPU Time ▼	Module	Function (Full)	Source File
func@0x25450	20.693s	libnetcdf.so.13	func@0x25450	
main._omp_fn.0	8.250s	profiling.o	main._omp_fn.0	readNetcdf.c
nc_get_var_short	7.142s	libnetcdf.so.13	nc_get_var_short	
nc_open	0.080s	libnetcdf.so.13	nc_open	
[Outside any known module]	0.030s		[Outside any known module]	
func@0x18eb0	0.010s	libgomp.so.1	func@0x18eb0	
nc_close	0.010s	libnetcdf.so.13	nc_close	

Profiling

Del tiempo de Cpu, el 57 % esta dedicado a la función detallada como *func@0x25450* que corresponde a la línea del código fuente 80, en la que se vuelca el resultado al archivo de salida. Esto fue verificado por vtune.

Como también se detalla en el siguiente gráfico, la porción del programa que esta parelelizada es solo el 22 % del tiempo de cpu, en donde se hace un uso intensivo del hardware disponible.

Uso del cpu