Universidad Nacional de San Luis

Práctico Final Integrador

San Luis - 2024

Asignatura:	Sistemas Distribuidos y Paralelos
Estudiante:	Kwist Daniel
Docentes:	María Fabiana Piccoli
	César Ochoa

Índice

Introducción

El presente informe es realizado con la intención de documentar e informar aspectos sobre las decisiones del desarrollo de los algoritmos secuenciales y paralelos, como así también analizar las métricas obtenidas de la ejecución de los mismos sobre imágenes de diferentes dimensiones solicitadas por la cátedra.

El objetivo es poder realizar una transformación de una imagen origen a una imagen destino mediante la técnica crossfading. En la cual se hacen iteraciones según la cantidad de imágenes intermedias que se quiera. Para esto se genera un factor P (con rango [1..0]) el cual es multiplicado cada canal rgb de cada pixel de la imagen.

El presente informe se dividirá en 2 ejes principales, los cuales son el desarrollo y el análisis de resultados. En estos se busca primero exponer la motivación de la resolución y luego poder comparar y analizar los resultados obtenidos mediante la ejecución con distintos parámetros.

Por último, se hace la aclaración que la generación del video se ha realizado mediante el software DaVinci Resolve. La cual simplifico la generación de la transición de los frames gracias a sus herramientas y facilidad de uso que ofrece.

Desarrollo

Aclaraciones previas sobre la implementación

- Manejo de imágenes:

  Los píxeles de cada imagen se cargan y guardan como puntero unsigned char, ya que es la forma en la que trabaja la librería stb.
  Por lo tanto se decidió crear la estructura Pixel que guardara los tres canales (rgb) de la imagen, de tipo unsigned char cada uno.

- Captura del tiempo de ejecución:

  Se añade el cálculo del tiempo de ejecución mediante la librería time.h haciendo uso de la función clock_gettime. La captura del tiempo se realiza desde el momento previo a la iteración que hace el llamado a la rutina de crossfade, hasta su finalización (previo a realizar la limpieza de memoria).

- Parámetro P:

  El parámetro P es un valor que se ubica en el rango [1,0] por el cual es multiplicado cada canal de ambas imágenes para obtener la imagen correspondiente del frame actual. Revisar los lineamientos del práctico para un ejemplo más detallado.

El desarrollo del trabajo se divide en los siguientes 4 algoritmos:

Secuencial

El desarrollo del algoritmo secuencial se ha decidido resolver mediante 2 pasos:

1. Preparación y carga de las imágenes

   Se especifican distintas variables y se prepara el sistema para la ejecución. Aquí se define la cantidad de frames a generar, y se cargan y validan las imágenes. La carga se realiza mediante el uso de la librería stb, capturando en la estructura Pixel los valores de los canales rgb de la imagen.
2. Iteración de crossfade y resultados :

   Aquí se captura el tiempo y se hace una iteración por la cantidad de frames que se quieran generar, calculando en cada paso de la iteración el P correspondiente, llamando a la función crossfade que se encarga de iterar por alto y por ancho de la imagen (la cual se encarga de la transformación de cada uno de los 3 canales RGB mediante el P actual) y por último para luego guardar el resultado con el nombre del frame correspondiente.

A partir de este algoritmo, se hacen las adaptaciones necesarias para resolver el resto de ítems según la necesidad de paralelismo.

MPI

Como se menciona anteriormente, se ha adaptado el algoritmo secuencial para la resolución mediante MPI. Aquí entra en juego la decisión de qué parte del algoritmo se va a paralelizar.

Primero se intentó resolver mediante la paralelización de la función crossfade, para que cada proceso se encargue de una parte diferente de la imagen, pero al no poder obtener resultados prometedores (ya que se generaban escrituras sucias en los resultados, resolviendo en una transición para nada suave y con mucho ruido entre los frames) se decidió distribuir los valores de P entre los procesos y que cada proceso se encargue de iterar sobre un rango de P.

Por lo tanto, previamente se calcula y divide P en P_Valores y luego se calcula cuantos frames va a tener cada proceso (dividiendo la cantidad de frames / cantidad de procesos y creando un subset llamado local_P_valores el cual guardará qué valores de P usará cada proceso.

Mediante MPI_Scatter se hace la dispersión o distribución de P_valores desde el master (rank 0) al resto de procesos, en los que cada uno recibe dentro de su local_P_valores los P_valores que corresponden segun el valor de frames_por_proceso.

Ahora, para la iteración, se realiza el crossfade iterando desde 0 hasta los frames que correspondan al proceso, tomando el P correspondiente a la posición i que corresponde de local_P_valores (el cual se obtuvo por la distribución mediante MPI_Scatter).

Por ejemplo, con 4 procesos, el proceso 0 recibe en local_P_valores los valores de P_valores[0..23], el proceso 1 los valores de P_valores[24..47] y así para cada proceso.

A la hora de guardar los resultados, se guardan dentro de la variable local_frame, el nombre se calcula desde el P correspondiente a la ejecución, siendo “frame(P + (rank * frames_por_procesos))” y se añade extra “_rank.bmp”. Véase el código del algoritmo.

Al resolverlo de esta forma, el problema se volvió más simple de programar y corregir errores, y ahora cada proceso se va a pasar a encargar de generar un rango de imágenes, eliminando también así la necesidad de sincronizar resultados.

Además para la captura de tiempos de ejecución, cada proceso indica cuánto tiempo tardó en resolver las iteraciones que le corresponden.

OpenMP

Haciendo uso de la simplicidad de la librería, se hace bastante sencillo la implementación de este algoritmo, añadiendo solamente al algoritmo secuencial únicamente la definición del número de threads mediante la directiva:
omp_set_num_threads(n) (donde según la ejecución se especifica en n deseado),
y luego se paraleliza la iteración que llama al crossfade mediante la directiva:
#pragma omp parallel for schedule(static,1) private (i).

Se decide resolverlo con schedule ya que es la forma de paralelizar a la cual nos acostumbramos mediante los prácticos de aula. Por otro lado, respecto a los parámetros de la sección schedule, se utiliza

- Static debido a que se asume que la carga va a ser balanceada para las iteraciones, ya que se trabaja siempre sobre las mismas imágenes y lo que varía es el P
- Chunk size = 1, para que cada hilo reciba de una sola iteración. Al incrementar el chunksize se observan errores en la generación de imágenes los cuales generaban que las transiciones no sean suaves alterando la solución del problema.

Hibrido (MPI + OpenMP)

Por último, la resolución híbrida se resuelve tomando el algoritmo de MPI y añadiendo la misma directiva utilizada en el algoritmo de OpenMP, por lo tanto, cada proceso de MPI utiliza los threads que ofrece OpenMP.

Análisis de resultados

Al momento de ejecutar, se han realizado 5 ejecuciones sobre cada algoritmo por cada tamaño de imagen deseado (800x800, 2000x2000 y 5000x5000 pixeles). Cabe aclarar que las imágenes han sido previamente formateadas mediante la herramienta image magick, para convertirlas a formato .bmp, especificar la resolución deseada y también configurar el bit depth (cantidad de bits utilizados para indicar el color de un solo píxel) de las mismas a 24 (TrueColor), realizando esto último bajo la necesidad que imágenes con diferente bit depth no eran compatibles al momento de generar el crossfade.

Las ejecuciones de MPI se han resuelto mediante 4, 16 y 32 procesos, los threads de OpenMP se han declarado en 4,16 y 32 threads y para el algoritmo híbrido se han utilizado combinaciones de 4,16 y 32 procesos con 4 y 16 threads cada proceso, con una ejecución excepcional de 32 procesos y 32 threads.

El número de frames generados ha sido 96 (24*4) qué son los solicitados para generar un video de 4 segundos a 24 frames por segundo. Siendo las iteraciones de los extremos con P=1 y P=0, por lo tanto se tiene 94 imágenes intermedias.

Para realizar las ejecuciones secuenciales y de OpenMP se ha utilizado el nodo 3 del clúster. En cambio para las de MPI e Híbrido se ha especificado un archivo machinefile que indica la utilización de los nodos 3 y 4. Esto se ha decidido para dejar a disposición otros nodos y no monopolizar el uso del cluster.

A continuación se realizan los cálculos de promedios de tiempos, speedup y eficiencia, como también luego el análisis de los resultados, al final se incluyen tablas completas de todos los tiempos de las ejecuciones con sus respectivos parámetros.

Se aclara que la representación de los tiempos es en unidades de segundos.

Cálculo de promedios

Prom	Seq	MPI - 4P	MPI - 16P	MPI - 32P
800x800	63.9542999442	18.1840582202	5.9585044308	4.4441288382
2000x2000	448.994456492	134.121934599	39.6695089482	27.4510788546
5000x5000	2834.3299667	834.377834626	221.318218368	126.207134419

Prom	OpenMP - 4T	OpenMP - 16T	OpenMP - 32T
800x800	17.0655796502	6.0057347908	4.7603770992
2000x2000	115.883440273	34.8803142908	24.8774012132
5000x5000	714.9771833	201.680607403	134.293836672

Prom	Hyb - 4P-4T	Hyb - 4P-16T	Hyb - 16P-4T	Hyb - 16P-16T
800x800	5.6934333018	4.8172195918	4.50476283	4.9563462432
2000x2000	36.9436943844	21.460209738	21.3096661334	20.3862173042
5000x5000	220.57038065	109.638004435	107.693769872	103.211056005

Prom	Hyb - 32P-4T	Hyb - 32P-16T	Hyb - 32P 32T
800x800	4.6104234786	4.67075015	5.210986047
2000x2000	20.9127873184	20.6349093098	20.6281759282
5000x5000	102.699251074	100.927183809	103.243369771

Cálculo de speedup

Se utilizará la siguiente fórmula: T(Mejor Secuencial) / T(Prom(Paralelo))

SpeedUp	Mejor Seq	MPI - 4P	MPI - 16P	MPI - 32P
800x800	63.3492983170	3.48378219811	10.6317447696	14.2546043608
2000x2000	447.5241461860	3.33669617519	11.2813129795	16.3026068504
5000x5000	2824.2623770560	3.38487224834	12.7610930446	22.3779930513

SpeedUp	OpenMP - 4T	OpenMP - 16T	OpenMP - 32T
800x800	3.71210938131	10.5481344954	13.3076218537
2000x2000	3.86184725904	12.8302784905	17.9891839325
5000x5000	3.95014336544	14.0036387902	21.0304690598

SpeedUp	Hyb - 4P-4T	Hyb - 4P-16T	Hyb - 16P-4T	Hyb - 16P-16T
800x800	11.1267305611	13.1505938456	14.0627377528	12.781451337
2000x2000	12.1136814724	20.8536706607	21.0009928539	21.9522896037
5000x5000	12.8043591743	25.7598849196	26.2249374352	27.3639519483

SpeedUp	Hyb - 32P-4T	Hyb - 32P-16T	Hyb - 32P 32T
800x800	13.7404510911	13.5629815945	12.1568735256
2000x2000	21.3995456164	21.6877205258	21.6947997605
5000x5000	27.500321059	27.9831683642	27.3553874047

Cálculo de eficiencia

Se utilizará la siguiente fórmula: SpeedUp / P
Donde P será la cantidad de procesos en MPI, la cantidad de threads en OpenMP o el producto de Procesos * Threads en Híbrido.

Eficiencia	MPI - 4P	MPI - 16P	MPI - 32P
800x800	0.87094554952	0.6644840481	0.44545638627
2000x2000	0.83417404379	0.70508206121	0.50945646407
5000x5000	0.84621806208	0.79756831528	0.69931228285

Eficiencia	OpenMP - 4T	OpenMP - 16T	OpenMP - 32T
800x800	0.92802734532	0.65925840596	0.41586318292
2000x2000	0.96546181476	0.80189240565	0.56216199789
5000x5000	0.98753584136	0.87522742438	0.65720215811

Eficiencia	Hyb - 4P-4T	Hyb - 4P-16T	Hyb - 16P-4T	Hyb - 16P-16T
800x800	0.69542066006	0.20547802883	0.21973027738	0.04992754428
2000x2000	0.75710509202	0.32583860407	0.32814051334	0.08575113126
5000x5000	0.80027244839	0.40249820186	0.40976464742	0.10689043729

Eficiencia	Hyb - 32P-4T	Hyb - 32P-16T	Hyb - 32P 32T
800x800	0.10734727414	0.02649019842	0.0118719468
2000x2000	0.16718395012	0.04235882915	0.02118632789
5000x5000	0.21484625827	0.05465462571	0.02671424551

Análisis de resultados

Podemos observar inicialmente la gran ventaja que se obtiene al paralelizar la ejecución con los mejores tiempos obtenidos de cada algoritmo, específicamente obtenemos que se resuelve con MPI en 32 procesos alrededor de 22 veces más rápido, con OpenMP con 32 threads alrededor 21 veces más rápido y en el algoritmo Híbrido con 16 procesos y 16 threads hasta 28 veces más rápido.

Estos resultados se relacionan directamente en los SpeedUps obtenidos, logrando resultados cerca de la aceleración lineal en distintas configuraciones, siendo la mejor configuración mediante OpenMP con 4 threads, mostrando el mejor SpeedUp (3.71, 3.86 y 3.95), para la cual se ve una eficiencia altísima (0.92,0.96 y 0.98) para las 3 dimensiones de imágenes, logrando ser la configuración que mejores resultados obtiene respecto a la aceleración y la eficiencia, dentro de todas las que se han probado.

Por otro lado, por más que el tiempo para resolver el problema disminuye aumentando la cantidad de procesos o threads, podemos observar un altísimo deterioro en la medida de eficiencia (y un alejamiento de la aceleración lineal hablando del speedup), esta medida disminuye drásticamente indicando que en estas ejecuciones se están malgastando los recursos, quedando ociosos o aumentando drásticamente los esfuerzos de comunicación y sincronización de procesos.

De todas formas, cabe aclarar que puede resultar ser útil elegir una configuración con más procesos pero no tan buen speedup o eficiencia, ya que estos pasan a ser a veces hasta casi 4 veces más rápidos al tener 4 veces más procesos/threads (resumidamente, pasar de 4 procesos a 16 en MPI o 4 threads a 16 en OpenMP resuelve en ejecuciones mucho más rápidas a coste de un poco de eficiencia).

Por otra parte, la configuración Híbrida puede ser útil en situaciones donde no importe malgastar recursos, ya que los tiempos de ejecución obtenidos son demasiado bajos. Aunque hay que tener en cuenta que al agregar demasiados procesos o threads, no se consigue una mejora muy significativa, ya que, por ejemplo, en las imágenes de 5000x5000, con 4 procesos y 16 threads se obtienen 109.6 segundos de ejecución, y el mejor promedio registrado es de 100.9 segundos con 32 procesos y 16 threads, por lo que aumentar en tal número los procesos y threads para una ganancia de 9 segundos no es muy significante.

Como punto positivo, se puede elegir si se desea mantener una alta eficiencia y aceleración con tendencia lineal al utilizar pocos procesos y/o threads, o en cambio, si se requiere una alta velocidad de ejecución, se puede aumentar drásticamente el número de estos para lograr ejecuciones inmediatas.

Pero de todas formas, se puede lograr un equilibrio dentro de los procesos y/o threads a utilizar para mantener una buena eficiencia, reduciendo los tiempos de ejecución y evitando que se malgasten recursos con configuraciones intermedias.

Detalle de los resultados obtenidos

Todos los tiempos son medidos en unidad de segundos.

800x800	Seq	MPI - 4P	MPI - 16P	MPI - 32P
I	64.2233023790	18.2048311750	5.9890450910	4.4235148320
II	63.8301026970	18.2146324800	5.7584387950	4.2260899320
III	63.8099660580	18.0462353520	5.7039155010	4.6146521930
IV	64.5588302700	18.1379773000	6.1432787070	4.4453315000
V	63.3492983170	18.3166147940	6.1978440600	4.5110557340

800x800	OpenMP - 4T	OpenMP - 16T	OpenMP - 32T
I	17.3850564410	5.8430122830	4.4019087570
II	16.8926737140	6.1908873240	4.5884598120
III	16.7154584790	5.7002332750	4.9938979810
IV	17.0541062740	5.8681714150	5.0088052460
V	17.2806033430	6.4263696570	4.8088137000

800x800	Hyb - 4P-4T	Hyb - 4P-16T	Hyb - 16P-4T	Hyb - 16P-16T
I	5.8690355880	6.2252016250	4.5646159640	6.6015178080
II	5.7058600690	4.5748556580	4.5380063170	4.8467217910
III	5.5805961610	4.4767503660	4.3728244000	4.3689709290
IV	5.6198465870	4.5515151400	4.3735577340	4.5165493170
V	5.6918281040	4.2577751700	4.6748097350	4.4479713710

800x800	Hyb - 32P-4T	Hyb - 32P-16T	Hyb - 32P 32T
I	5.2825132700	4.5745794010	6.7234771760
II	4.4425930740	4.4608398530	5.3296787240
III	4.5194763300	4.6473822580	4.4816147610
IV	4.4521175870	4.6829709230	5.2164697130
V	4.3554171320	4.9879783150	4.3036898610

2000x2000	Seq	MPI - 4P	MPI - 16P	MPI - 32P
I	448.9942327300	133.7561773620	39.2713143670	31.1102290240
II	447.5241461860	133.9938597820	40.1466029570	26.8695646700
III	450.4316296390	134.2704919940	40.4764719330	26.2001366150
IV	447.6355078110	133.7402585200	39.9863998450	26.3869288390
V	450.3867660950	134.8488853380	38.4667556390	26.6885351250

2000x2000	OpenMP - 4T	OpenMP - 16T	OpenMP - 32T
I	116.2989682190	35.1699989830	24.5969033110
II	115.9001463870	34.6891666370	25.0194017840
III	115.4507495450	34.9056837700	25.0566427480
IV	115.4666442320	34.7059185680	24.7596685560
V	116.3006929800	34.9308034960	24.9543896670

2000x2000	Hyb - 4P-4T	Hyb - 4P-16T	Hyb - 16P-4T	Hyb - 16P-16T
I	39.0804097740	20.8798037690	21.5482678190	20.5281434550
II	35.9137340090	21.4495186410	21.5581876840	20.8841479050
III	36.6511348780	21.7179124080	21.6541093640	20.9376802750
IV	36.8229045490	22.3140983690	20.6942783900	19.8215861870
V	36.2502887120	20.9397155030	21.0934874100	19.7595286990

2000x2000	Hyb - 32P-4T	Hyb - 32P-16T	Hyb - 32P 32T
I	20.6257807100	20.6519148480	20.1723867780
II	21.7153173440	21.4301771200	21.1833030020
III	20.8025184440	20.3358218660	20.2282981380
IV	20.2089952340	19.9701604340	20.2977785520
V	21.2113248600	20.7864722810	21.2591131710

5000x5000	Seq	MPI - 4P	MPI - 16P	MPI - 32P
I	2832.7940080520	833.1384642480	217.5229936630	127.6143921750
II	2840.5600346700	835.1788157830	220.2922751290	124.9649851340
III	2824.2623770560	834.5151099260	220.8547269070	127.9827738910
IV	2837.3019590770	834.5779432580	223.9472995620	124.1851303580
V	2836.7314546560	834.4788399150	223.9737965800	126.2883905370

5000x5000	OpenMP - 4T	OpenMP - 16T	OpenMP - 32T
I	712.6024227750	194.3839473070	119.5562760600
II	712.7303441930	202.7776748830	129.9807953090
III	714.9536494660	203.1361531280	141.8623081380
IV	713.7741053850	203.3490331840	139.5178670930
V	720.8253946820	204.7562285110	140.5519367580

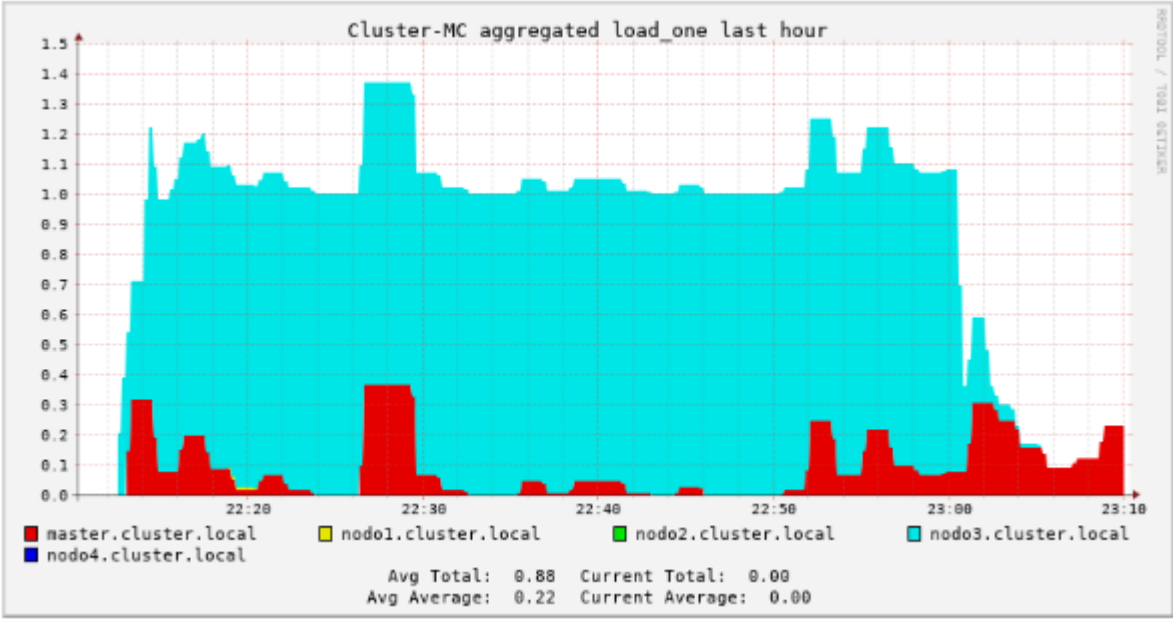
5000x5000	Hyb - 4P-4T	Hyb - 4P-16T	Hyb - 16P-4T	Hyb - 16P-16T
I	221.8102834070	106.3191084950	106.0346361710	102.4842095250
II	219.6330924580	111.3986569220	110.6601893310	100.4300298250
III	219.4737909670	109.8946458600	106.5597282370	103.8003757070
IV	220.5942364600	108.3747044210	109.0639310190	105.8986700600
V	221.3404999590	112.2029064770	106.1503646040	103.4419949070

5000x5000	Hyb - 32P-4T	Hyb - 32P-16T	Hyb - 32P 32T
I	100.9762417830	102.1101710960	99.1520270620
II	108.6320377920	99.5862973600	102.4393624130
III	99.3234449860	102.7495827210	105.5605832120
IV	101.0516667770	102.0856166980	104.3436374020
V	103.5128640310	98.1042511710	104.7212387660

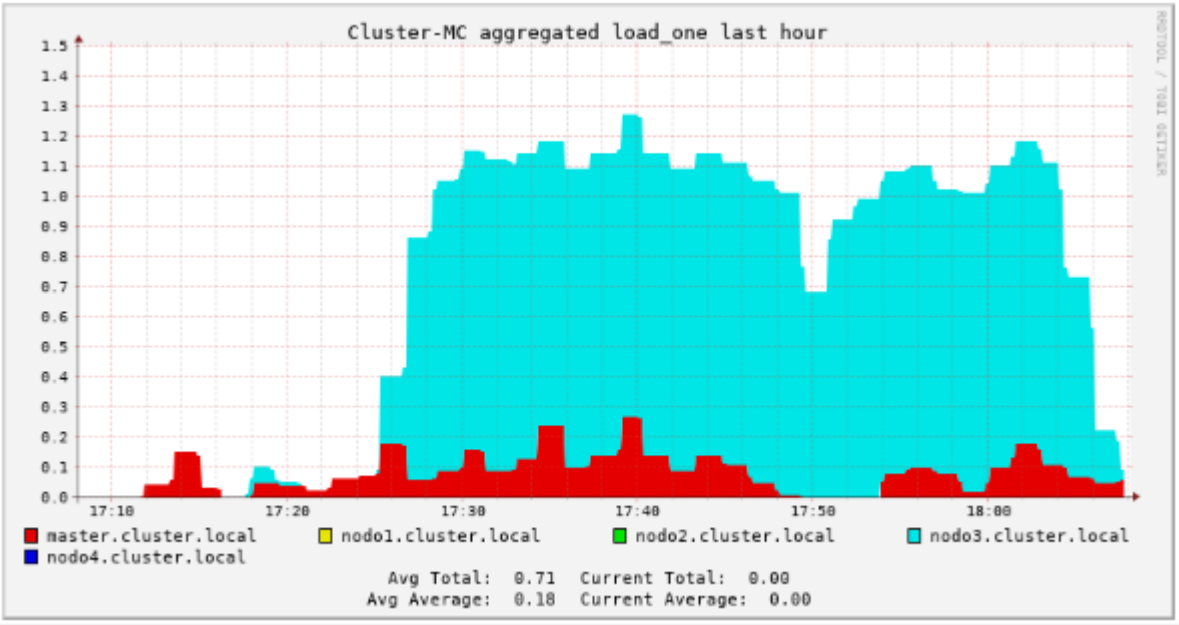
Capturas

Se incluyen diversas capturas del monitor ganglia del cluster durante algunas ejecuciones.

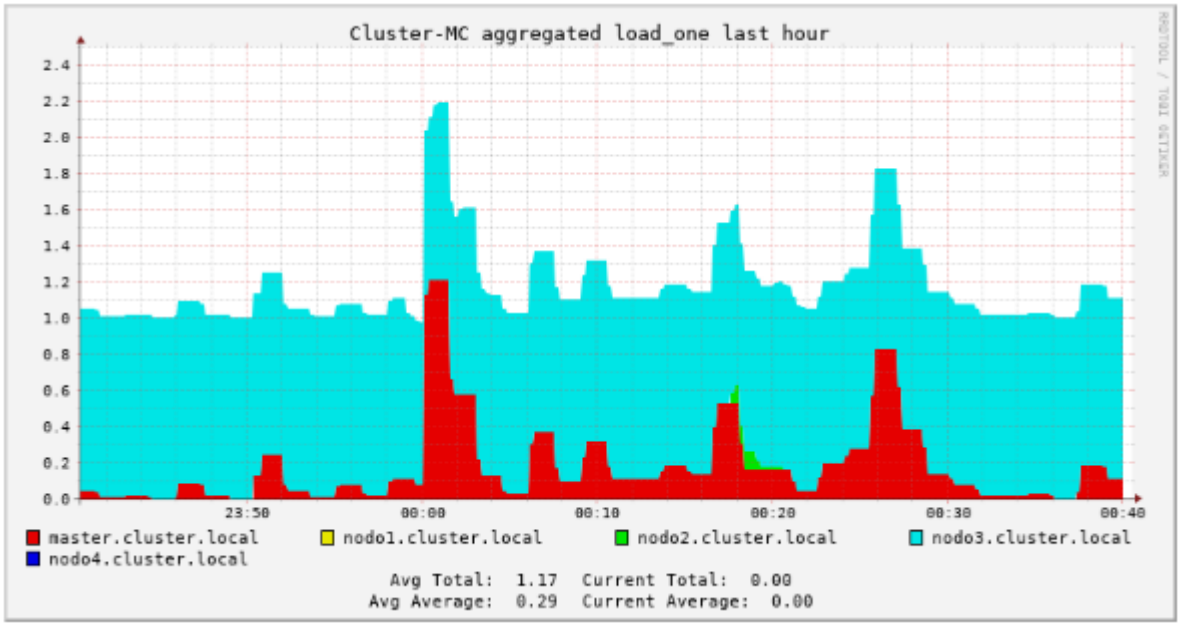
Secuencial sobre nodo 3, imagenes de 5000x5000



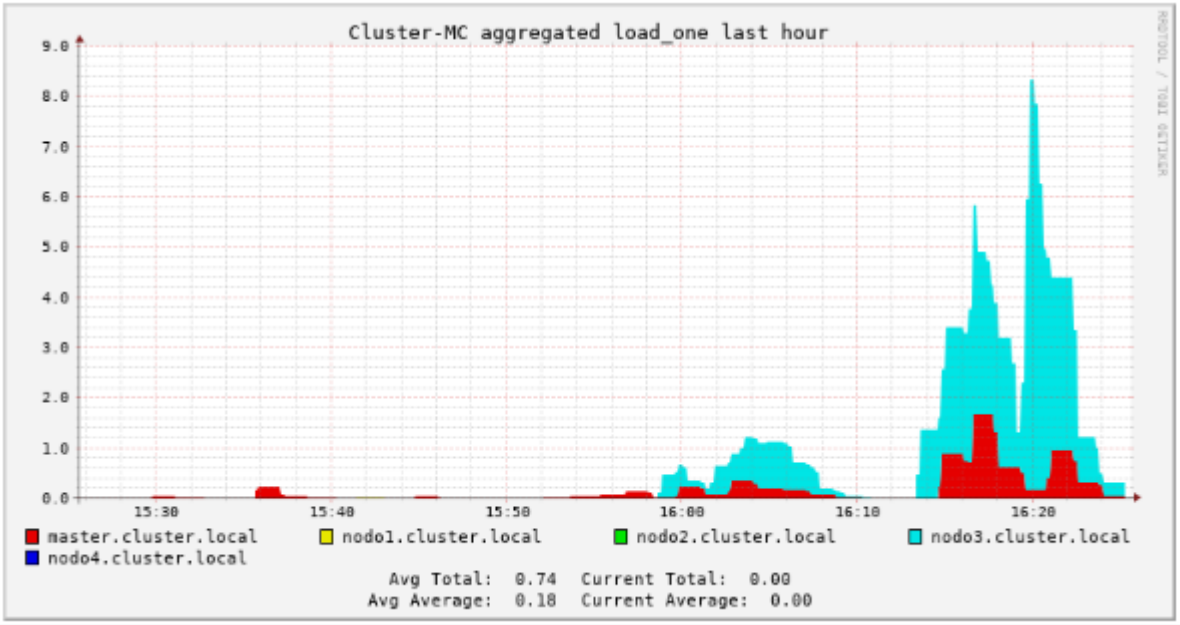
Secuencial sobre nodo 3, imagenes de 2000x2000



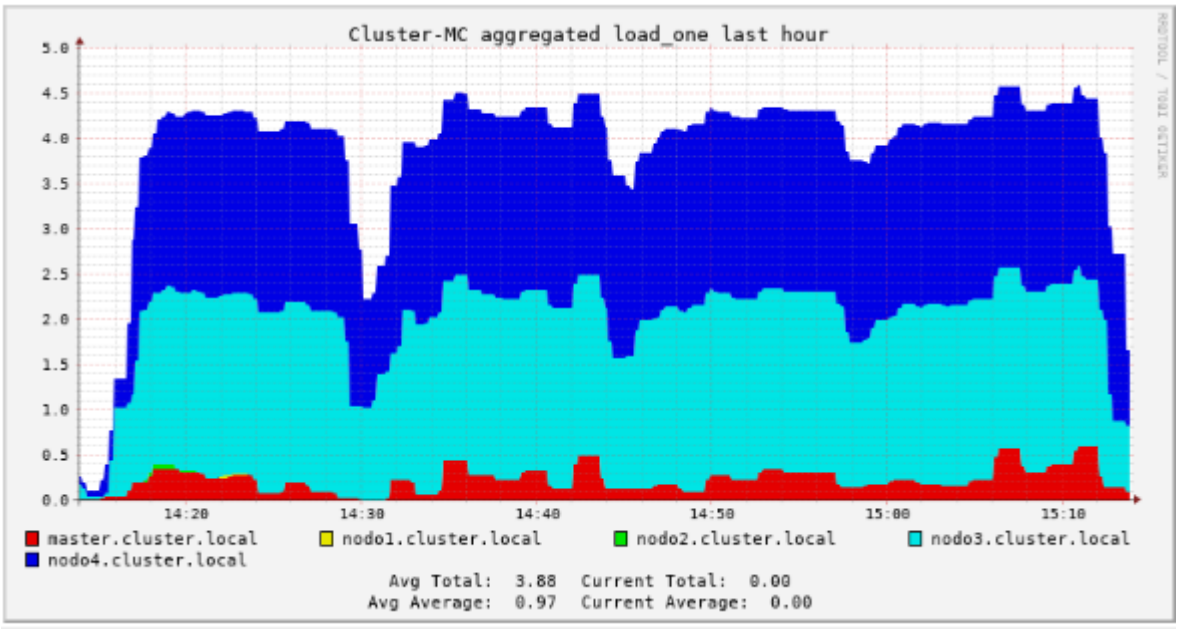
Secuencial sobre nodo 3, imagenes de 5000x5000



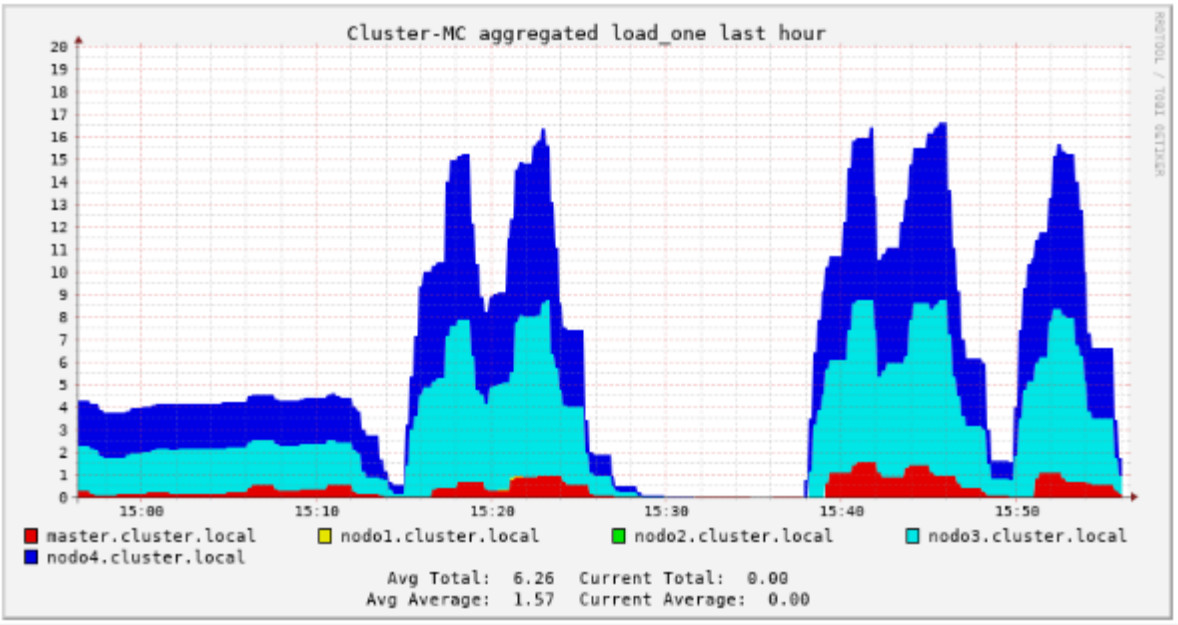
OpenMP sobre nodo 3 con 4,16 y 32 threads, imagenes de 800x800



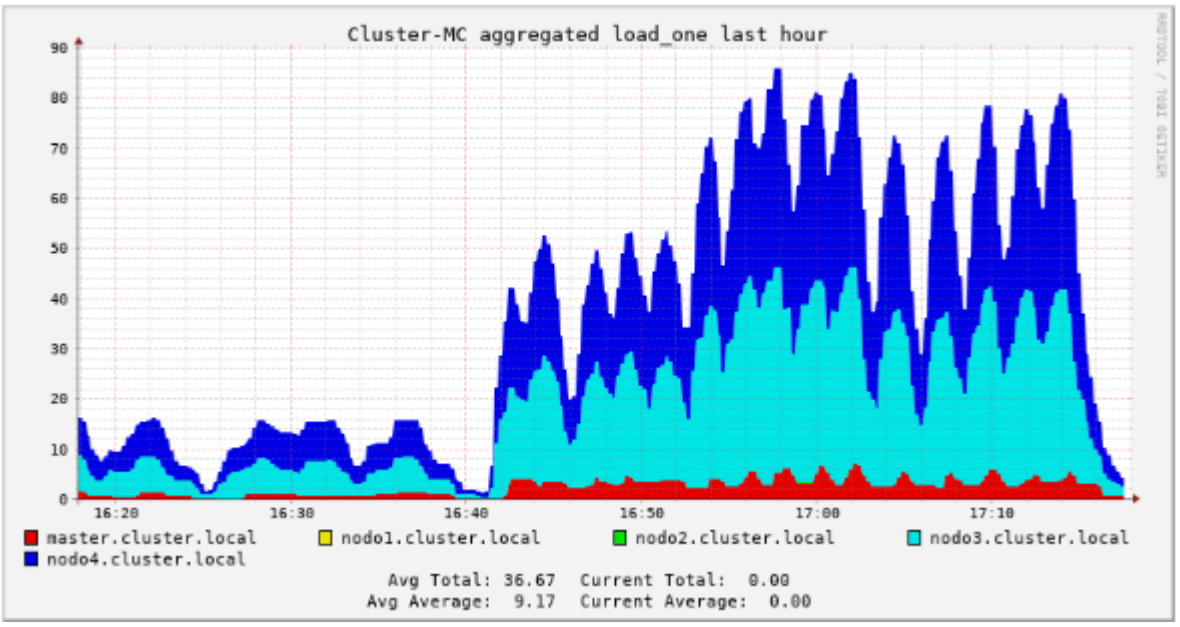
MPI, sobre nodos 3 y 4 con 4 procesos, imagenes de 5000x5000



MPI sobre nodos 3 y 4, los 5 picos corresponden a 5000x5000 con 16 procesos, la sección de la izquierda corresponde a la finalización de la captura anterior



Híbrido sobre nodos 3 y 4, con 4,16 y 32 procesos y 16 threads c/u (Desde 16:40), imágenes de 5000x5000



Estado del nodo 4 al ejecutar algoritmo Híbrido de 32 procesos y 32 threads, imagenes de 5000x5000

Conclusiones

Para concluir cabe aclarar que los algoritmos han sido resueltos mediante una serie de parámetros que el alumno consideró suficientes, esto se ha debido a que en total se han hecho 210 ejecuciones.

Por cada configuración nueva que se agregaba se resultaba en 15 nuevas ejecuciones (5 por cada tamaño de imagen, se eligió 5 para no escalar demasiado en el número de las ejecuciones y aun así salvarse medianamente de datos anómalos).

Al principio se utilizacion ejecuciones de imágenes de 100x100 para poder resolver de forma local rápidamente (dado a que los recursos locales no son muy poderosos) y comprobar que los algoritmos compilaban y ejecutaban de forma correcta, una vez resueltos los 3 algoritmos, se pasó con la ejecución con los tamaños de imágenes solicitados por la cátedra dentro del cluster de la universidad.

Por otro lado, se puede destacar que el transcurso del diseño de los algoritmos no llevó demasiadas complicaciones. Una vez se consiguió el algoritmo secuencial funcional, el resto de algoritmos fueron adaptaciones que llevaron poco esfuerzo de programación. Las mayores dificultades se encontraron en cambio a la hora de elegir y utilizar una librería de carga y guardado de imágenes, como una representación de los píxeles de las imágenes. Para lo cual ha llevado varias horas de ejecución y hasta depuración por consola para ver los valores de cada pixel registrado validando así la información que se leía o que las transformaciones de las imágenes eran correctas.

Una vez solucionado ese problema, se dio continuación con la implementación de los algoritmos a un ritmo más rápido y con dificultades que se resolvían pronta y satisfactoriamente, llegando a ser la mayor complicación el problema comentado previamente con MPI a la hora de paralelizar la transformación de las imágenes por secciones, lo cual se resolvió al dividir el P entre procesos y no la imagen.

Como apreciación final cabe destacar que resulta positiva la posibilidad de trabajar sobre el cluster de la universidad, permitiéndonos integrar los conceptos vistos durante el dictado de la asignatura y ver como estos realmente ayudan a un problema práctico y real al contar con la ventaja de contar con la paralelización del mismo.

También se agradece al cuerpo docente por mantener un seguimiento del trabajo y responder a cuestiones que surgieron inclusive fuera de horario como en fines de semana y feriados.

Apartados, Apéndices o Anexos

Cómo apartado, se aprovecha también a hacer mención sobre el software, herramientas utilizadas para el desarrollo y otros aspectos de la implementación:

- CLion: IDE utilizado para el desarrollo de los algoritmos y la compilación y ejecución local de los mismos.
- Image Magick: Herramienta utilizada para dar formato y convertir a bmp las imágenes previo a su utilización.
- iloveimg: Herramienta online para convertir imágenes a formato jpg.
- FileZilla: Aplicación FPT para el traspaso de archivos al cluster.
- DaVinci Resolve: Aplicación para la generación del video a partir de los frames obtenidos luego de la ejecución.
- Google Docs: Software utilizado para la redacción del presente informe.
- Imágenes utilizadas:
    - https://www.albertdros.com/post/2017/12/03/shooting-the-same-location-over-and-over
    - McDonald's Big Mac (google images)
    - Burger King Whopper (google images)
    - Planeta Tierra (google images)
    - Luna (google images)
- Librerías:
    - MPI: Utilizada durante el transcurso del dictado.
    - OpenMP: Utilizada durante el transcurso del dictado.
    - STB: En especial stb_image y stb_image_write para la lectura y escritura de imágenes en C.

Bibliografía

En el presente informe no se han utilizado referencias a bibliografías externas o de terceros más que la referencia a la documentación de las librerías detalladas en la sección de Apartados, Apéndices o Anexos.