**ANEXO A**

El presente anexo corresponde a la ejecución del experimento de evaluación de rendimiento de la ejecución de 4 algoritmos de multiplicación de matrices. Los fuentes de los algoritmos se encuentran desarrollados en lenguaje C. Para el ejercicio, se utilizó la terminal de Linux para la ejecución de los comandos que se encuentran descritos a continuación.

**Evaluación de desempeño OpenMP**

OBJETIVO

Hacer un estudio de rendimiento de 4 algoritmos de multiplicación de matrices utilizando OpenMP.

**Desarrollo del experimento**:

1. Localización de archivos en el directorio del repositorio:

Los fuentes utilizados para realización de este experimento se encuentran disponibles dentro del repositorio en la ruta que se muestra a continuación en la Figura 1:

/OpenMP/src

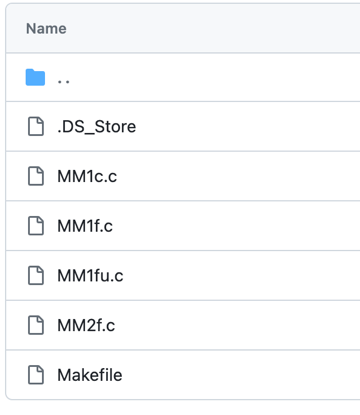
****

Figura 1: Directorio /OpenMP/src

Adicional a los archivos.c, se incluye el archivo makefile, utilizado para compilar los fuentes.

1. Compilar los archivos fuente:

Para el experimento vamos a compilar los archivos fuentes de cada uno de los algoritmos de multiplicación de matrices MM1c.c, MM1f.c, MM1fu.c y MM2f.c.

Desde el directorio src compilar los programas usando los siguientes comandos:

make MM1c

make MM1f

make MM1fu

make MM2f

A modo de ejemplo, la Figura 2 muestra resultado de la ejecución de los comandos indicados:

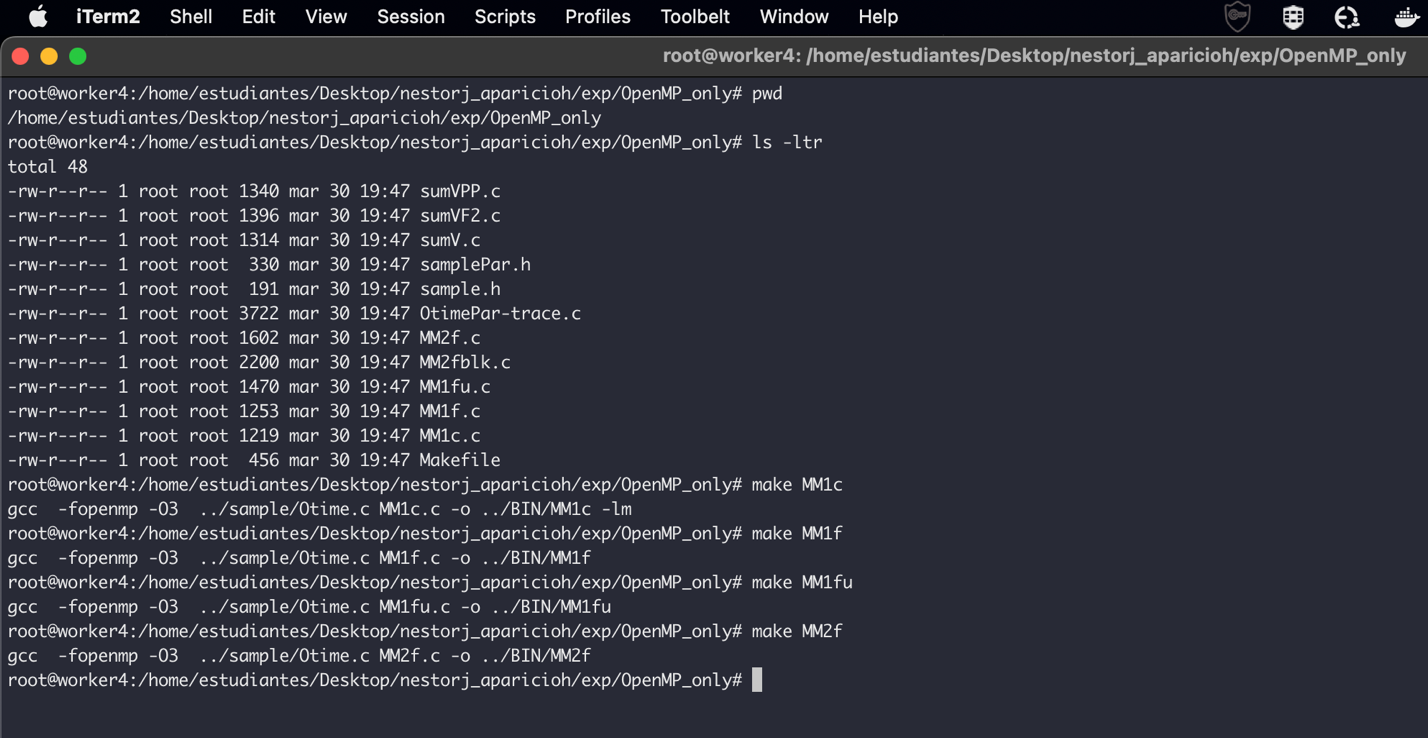


Figura 2 Ejemplo compilación archivos fuente

1. Validar creación de archivos ejecutables:

Una vez se compilan los archivos fuentes, los ejecutables se almacenan en la ruta:

/OpenMP/bin. Los archivos se generan como se muestra en la Figura 3:

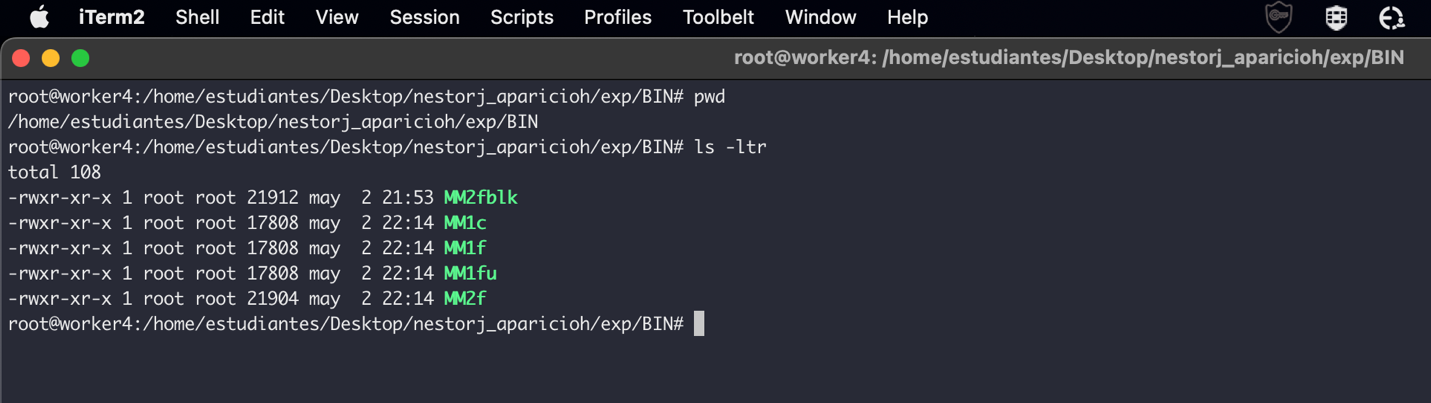


Figura 3 Ejemplo archivos compilados

1. Archivos para lanzar el experimento:

En la carpeta /OpenMP/tools se incluyen los archivos utilizados para la ejecución del experimento, tal como lo ilustra la Figura 4:

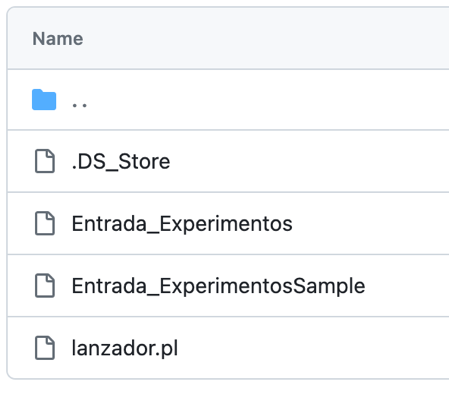


Figura 4 directorio /OpenMP/tools

Descripción de los archivos:

* 1. **Entrada\_Experimentos**: En este archivo se configuran los parametros para la ejecución para el experimento, así:

OpenMP // Kind of Experiment – Tipo de experimento

MM1c,MM1f,MM1fu,MM2f // Binarys – Archivos ejecutables a utilizar en la prueba

2,4,6,8,10,12,14,16,18,20 // Threads – Cantidad de hilos que se utilizará en cada ciclo

30 // Number of Repetitions per experiment – Número de repeticiones

Adicional se configurará el tamaño de las matrices cuadradas de n\*n así:

:3500

:4000

:4500

:5000

:5500

:6000

:6500

:7000

:7500

Ejemplo de configuración del archivo Entrada\_Experimentos para la ejecución del experimento:

####################################################################################

# PONTIFICIA UNIVERSIDAD JAVERIANA

# ZINE Centro de Alto Rendimiento Computacional Javeriano

#

####################################################################################

#####################################################################################

# Inputs file experiments

#####################################################################################

OpenMP // Kind of Experiment

MM1c,MM1f,MM1fu,MM2f // Binarys

2,4,6,8,10,12,14,16,18,20 // Threads

30 // Number of Repetitions per experiment

#####################################################################################

# Inputs file => Matrix Size NxN

#####################################################################################

:3500

:4000

:4500

:5000

:5500

:6000

:6500

:7000

:7500

La Figura 5 muestra un ejemplo del archivo configurado para la ejecución del experimento:

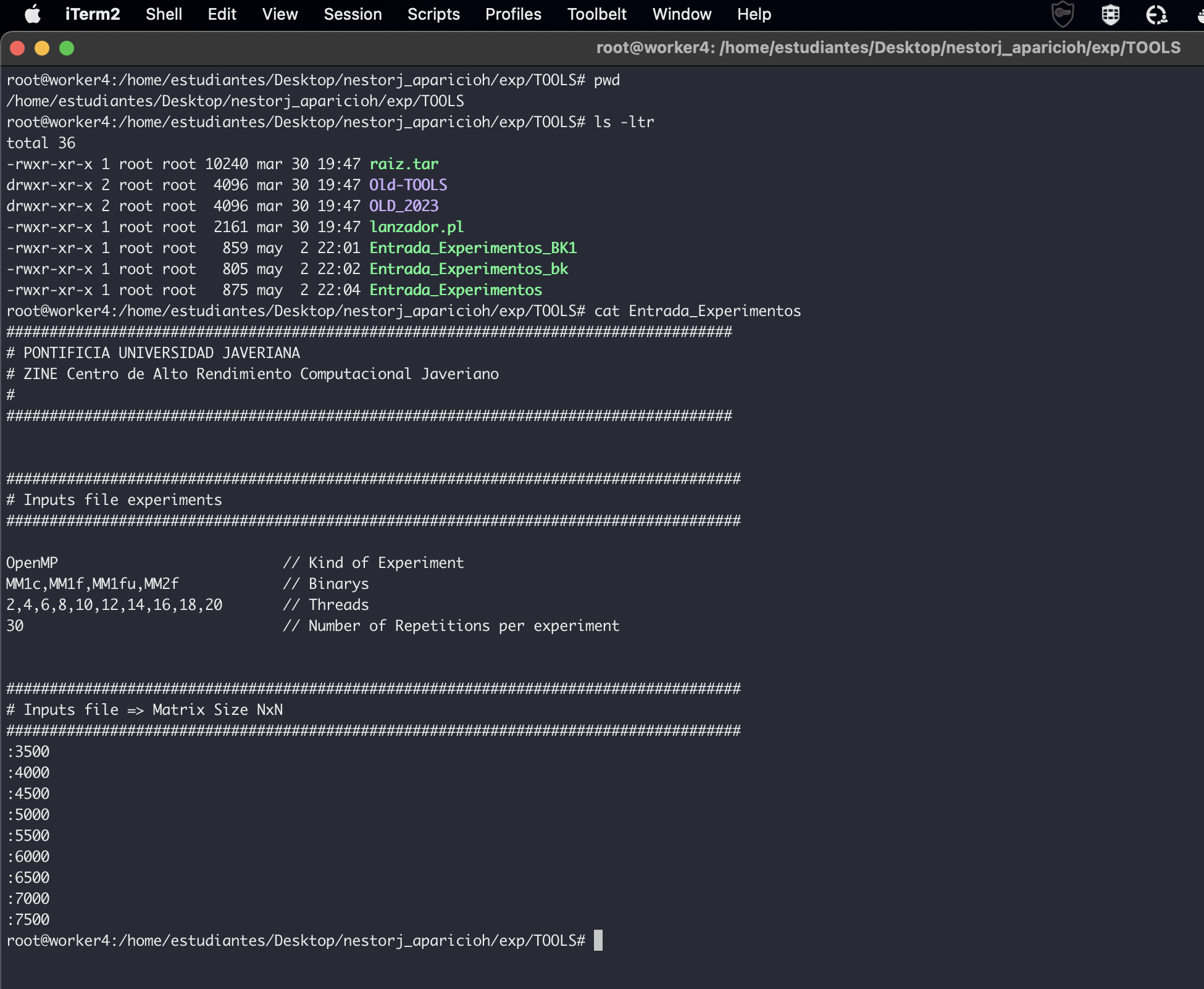


Figura 5 Ejemplo archivo Entrada\_Experimentos

* 1. **lanzador.pl**: Este archivo se encarga de ejecutar dados los parámetros definidos en el archivo Entrada\_Experimentos, cada una de las iteraciones propuestas en el experimento.

Este archivo fue desarrollado por el docente John Corredor y facilitado para la realización de este experimento:

#!/usr/bin/perl

# Created by: John Corredor UAB-CAOS

# john@aopcjc.uab.es

# Julio 2008

#

# Lanza el binary juntos con los argumentos desde argumentos.txt

# y los eventos, desde el archivo Entrada\_Experimentos

if (@ARGV[0]) {

$FInput = "$ARGV[0]";

} else {

usage();

}

use Switch;

$path0 = `pwd`;

chomp($path0);

$T = index($path0,"T")-1;

$Path = substr($path0,0,$T);

@Labels = ("Experiment","Binarys","Threads","Repetitions");

&Search(\@Labels);

$Experiment = $Inputs[0];

@Binarys = split(/,/,$Inputs[1]);

@Threads = split(/,/,$Inputs[2]);

$Repet = $Inputs[3];

@Separate = ("-",":");

#Capture SIZE MATRIZ in vector @Size

$Size = &Argument($Separate[1],1); @Size = split(/\n/,$arg);

#################################################################

# Main program: WARNING Binary x Threads x Size x Repetitions #

#################################################################

foreach $executable (@Binarys) {

$CARP = $executable."-".$Experiment."-";

chop($CARP);

system ("mkdir $Path/Soluciones/$CARP");

foreach $thread (@Threads) {

foreach $size (@Size) {

$file = "$Path/Soluciones/$CARP/$executable-".$size."-TH-".$thread.".txt";

print "$executable $size $thread ... on $file\n";

for ($k = 0; $k < $Repet; $k++) {

system("$Path/BIN/$executable $size $thread 0 0 >> $file");

#print "$Path/BIN/$executable $size $thread 0 0 \n";

}

close($file);

}

}

}

##################################################

# Functions #

##################################################

sub Argument {

$lab = $\_[0];

$pos = $\_[1]; # pos: 1 2 3 4 5

$arg = `cat $FInput | grep $lab | cut -f2 -d$lab | cut -f$pos -d' '`;

chomp($arg);

}

sub Search {

my($label) = @\_;

foreach (@ {$label}) {

$name = `grep $\_ $FInput | cut -f1`;

chomp($name);

push(@Inputs, $name);

}

}

sub Formato {

$in = $\_[0];

if ($in < 1000) {

$Id = $in;

} elsif($in < 10000) {

$ddk = $in/10000;

$Id = $ddk."K";

} elsif($in < 1000000){

$ddk = $in/1000000;

$Id = $ddk."M";

}

}

sub usage() {

print "\n test.pl: Error incorrect usage \n\n";

print "\t <directory> Is directory start point to check \n\n\n";

exit(1);

}

1. Ejecución del experimento: Una vez revisados y configurados los archivos listados en el punto 4, se procede a hacer la ejecución del experimento, para ello, se debe ejecutar el archivo lanzador.pl, utilizando el siguiente comando:

./lanzador.pl <archivo\_con\_parametros>

./lanzador.pl Entrada\_Experimentos

El comando anterior, ejecuta el proceso y mostrará su avance en la terminal de la sesión desde la que se ejecute, sin embargo, si la sesión se cierra, el experimento finalizará su ejecución. Para las ejecuciones de extensa duración como la propuesta en para este experimento, se sugiere utilizar el **nohup** para que la ejecución se haga en segundo plano y continue su tarea aún cuando la sesión desde la que se lanzó la instrucción haya finalizado (ejemplo en la Figura 6). De esta manera, el comando utilizado fue:

nohup ./lanzador.pl Entrada\_Experimentos &

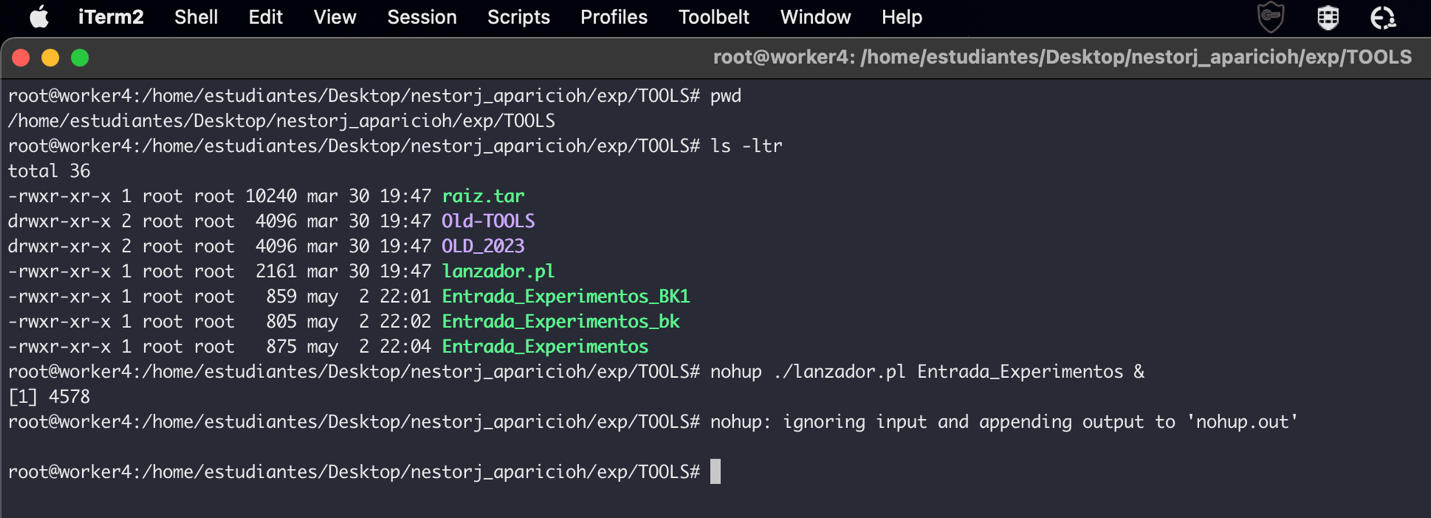


Figura 6 Ejemplo ejecución lanzador.pl con nohup

Una vez lanzado el proceso con el comando nohup, generará un archivo nohup.out que contendrá el log de ejecución del comando ejecutado. De ser requerido, mientras se procesa el experimento, se puede hacer seguimiento del avance del proceso usando el comando tail para verificar el contenido del archivo nohup.out como se muestra a modo de ejemplo en la Figura 7.

Tail -100f nohup.out

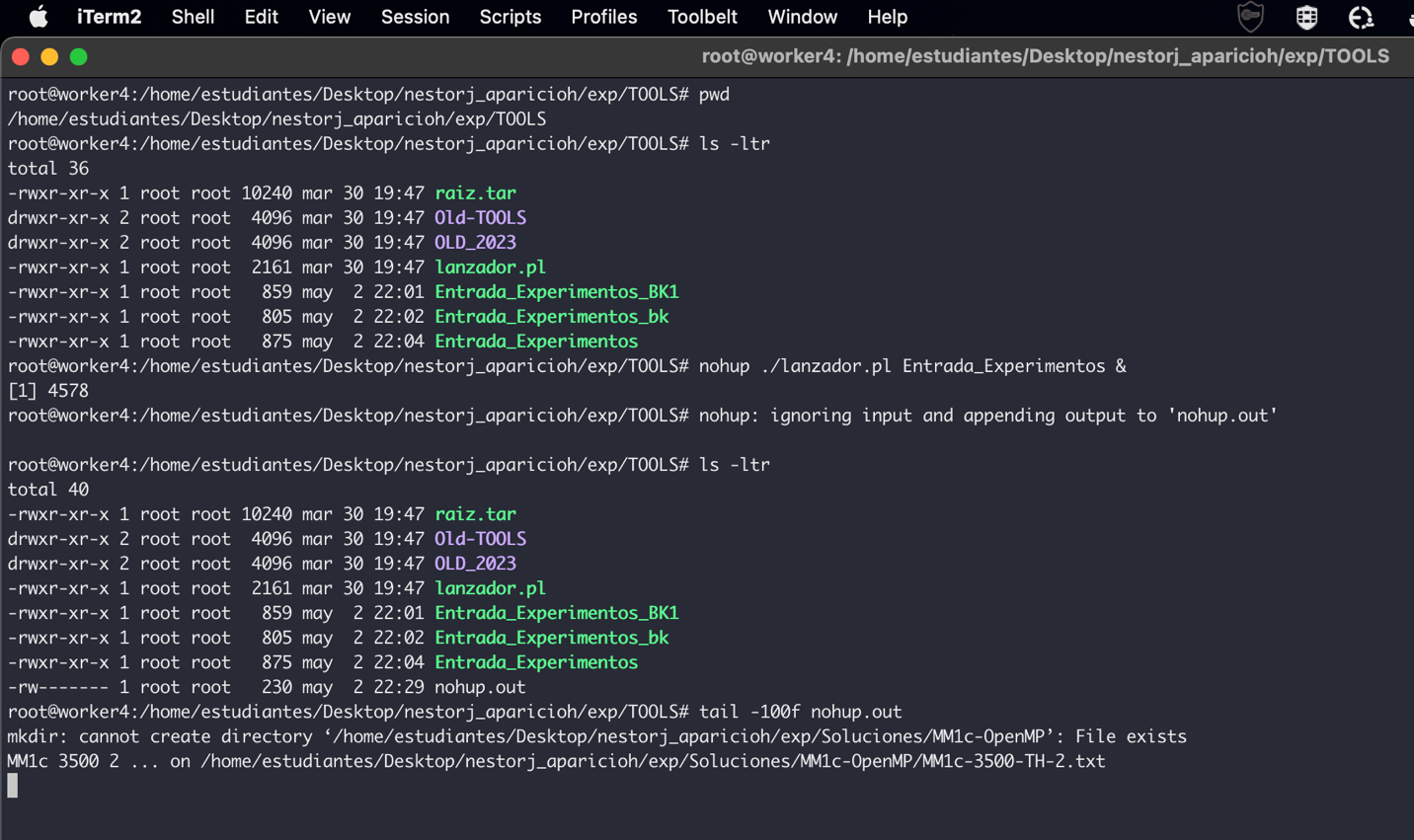


Figura 7 Vista del contenido del archivo nohup.out

Finalizada la ejecución, el contenido del archivo nohup.out se muestra similar al que se muestra en la Figura 8. El archivo nohup.out, se crea en la ruta desde la cual se haya lanzado el comando, para el ejemplo de la Figura 8, el archivo se generó en la ruta:

/home/estudiantes/Desktop/nestorj\_aparicioh/exp/TOOLS

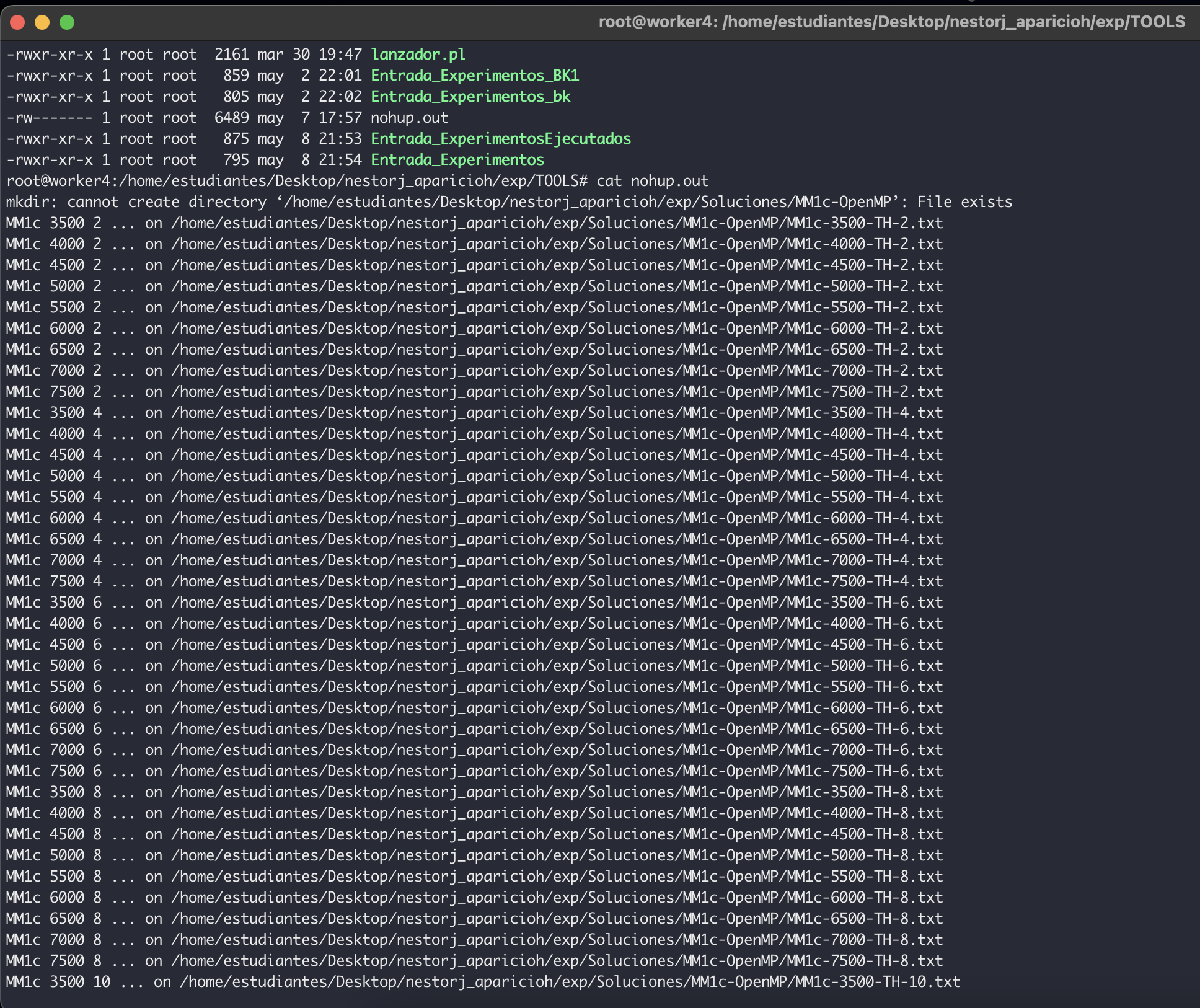


Figura 8 Ejemplo contenido archivo nohup.out

1. Revisión de resultados: De la ejecución anterior se creará el directorio Soluciones, al mismo nivel de directorio de la carpeta tools. Dentro de este directorio se almacenan las salidas producto de la ejecución del lanzador (lanzador.pl). Para cada algoritmo se crea una carpeta con el nombre de este. Así mismo, dentro de cada directorio de algoritmo, Por cada combinación de Algoritmo-TamañoMatriz-CantidadHilos que se haya configurado en el archivo Entrada\_Experimentos, se crea un archivo con el resultado de cada una de las ejecuciones o cantidad de experimentos configurados en el mismo archivo Entrada\_Experimentos. A modo de ejemplo, en la Figura 9 se muestra uno de estos directorios con los archivos generados para el experimento ejecutado:

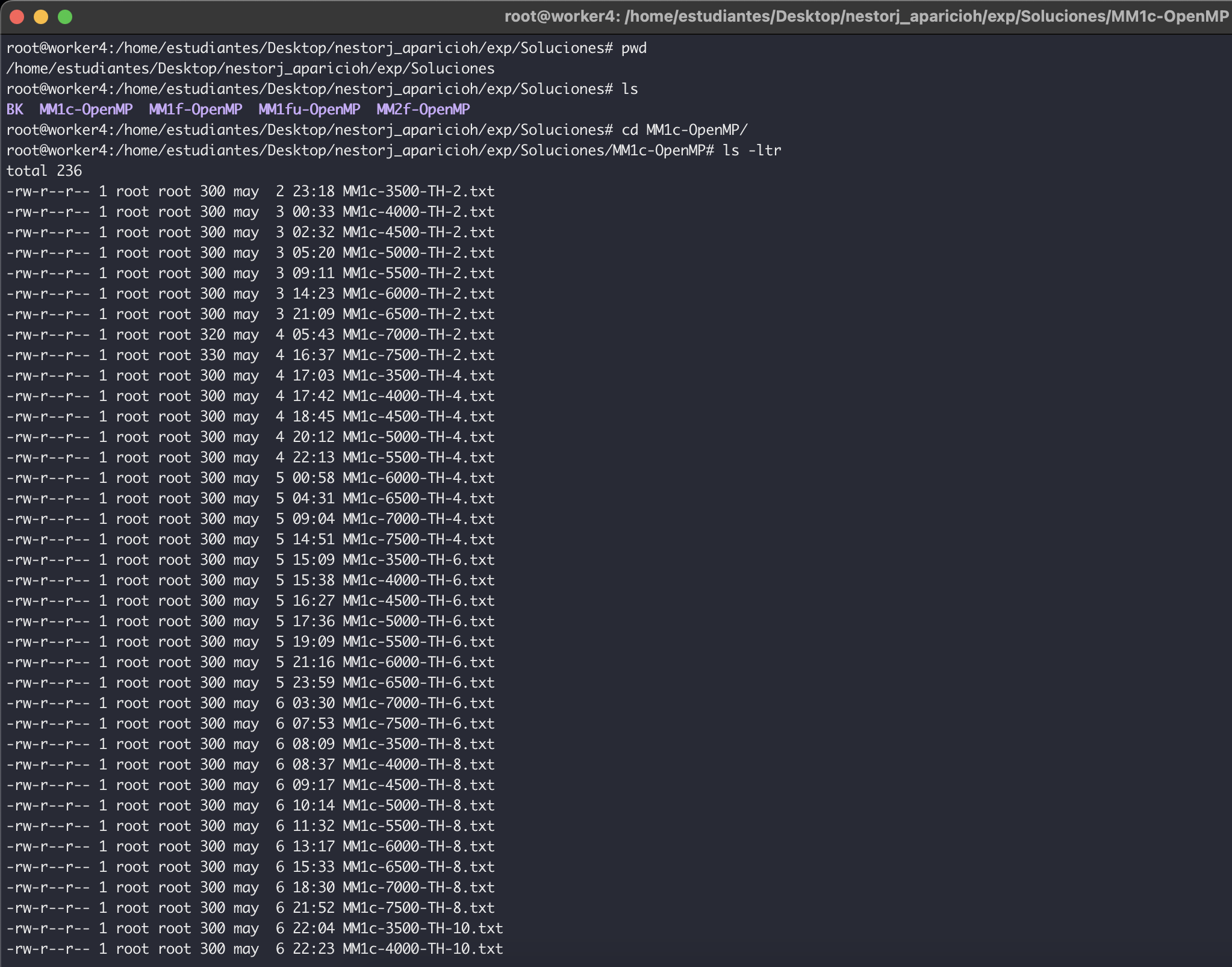
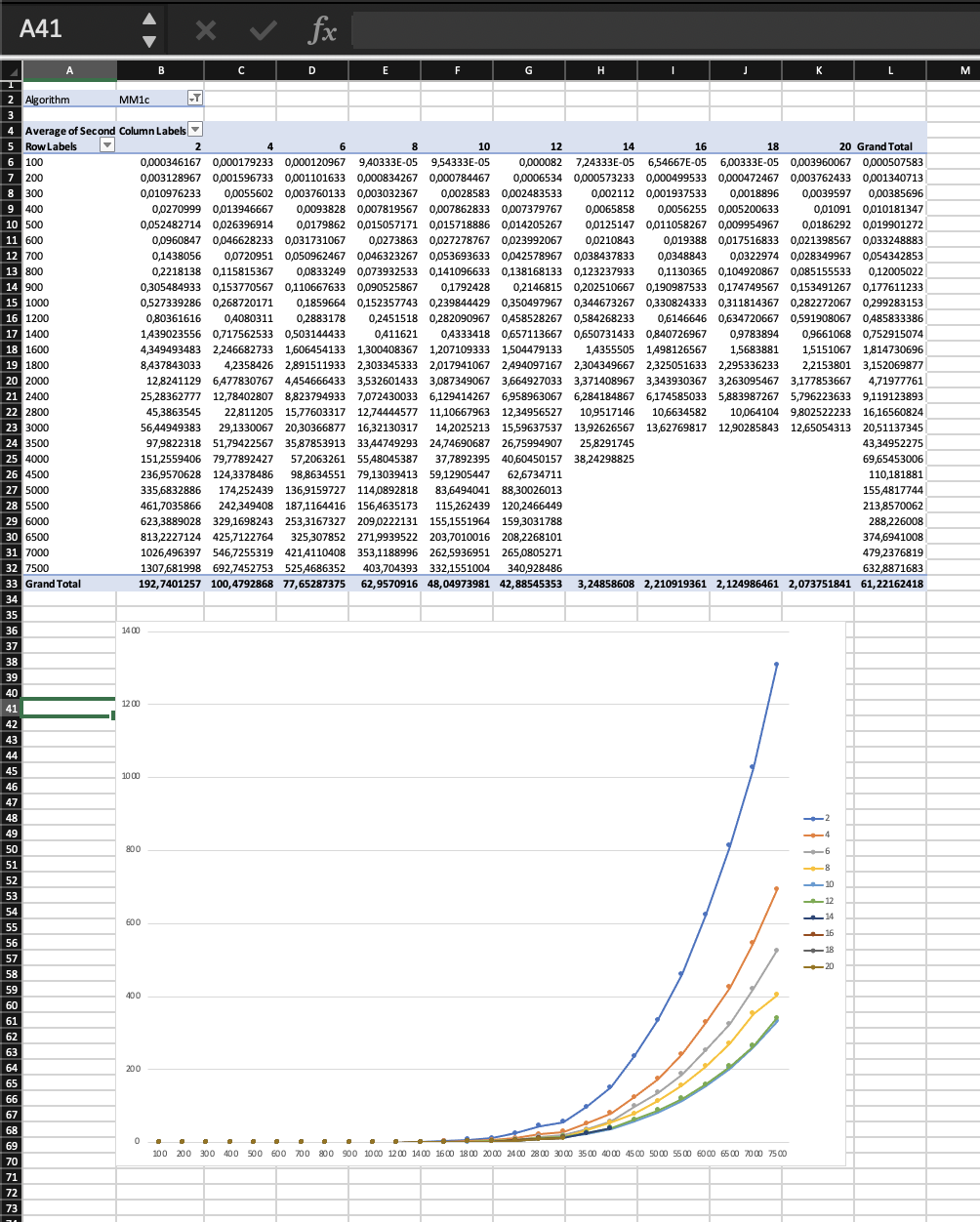


Figura 9 Ejemplo archivos generados. Algoritmo MM1c

Se realiza el proceso de extracción de la información. Se tabula y se grafica mediante Excel:



**Taller 2**

------------------------------------------------------------------------------------------------------------

Ejecutar algoritmos de multiplicación de Matrices en MPI.

Realizar el ejercicio con 30 ejecuciones.

Tabular los resultados.

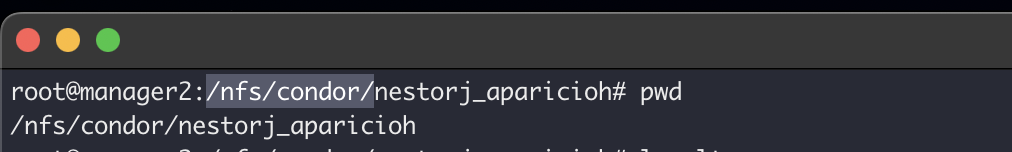
Hacer comparativa con mpi vs OpenMP

**Desarrollo del taller:**

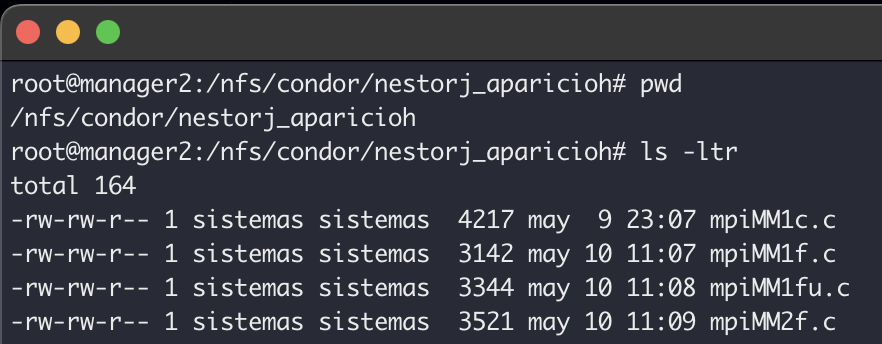
Partiendo de la configuración establecida, el server 10.195.34.31 (manager2) es el master del cluster donde ejecutaremos los procesos mpi de este experimento.

Inicialmente se accede al manager2 a través de una conexión ssh.

Dentro de la ruta /nfs/condor/ se crea la carpeta con el nombre de usuario de uno de los integrantes del grupo, para el ejercicio se utilizó el usuario nestorj\_aparicioh



Dentro de esta carpeta se copian los fuentes de los algoritmos de multiplicación de matrices MM1c, MM1f, MM1fu y MM2f en lenguaje C y con estructura de MPI. Dado que ninguno de los miembros del grupo maneja el lenguaje C, se utilizó ChatGPT para apoyar la generación de este código en lenguaje C:



Se procede a compliar estos fuentes utilizando openMPI, para ello se utilizaron los siguientes comandos:

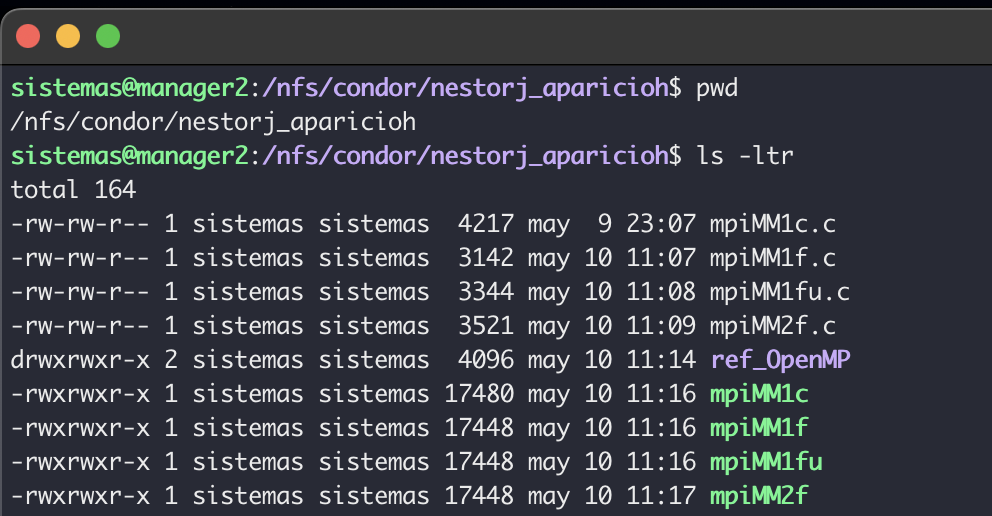
mpicc mpiMM1c.c -o mpiMM1c

mpicc mpiMM1f.c -o mpiMM1f

mpicc mpiMM1fu.c -o mpiMM1fu

mpicc mpiMM2f.c -o mpiMM2f

Con la ejecución de estos comandos, se generaron los siguientes ejecutables:



Se realizan pruebas locales de cada uno de los algoritmos recién compilados:

**Prueba1**: Ejecución local de cada algoritmo con 2 hilos y una matriz de 300\*300:

mpirun -np 2 mpiMM1c 300

mpirun -np 2 mpiMM1f 300

mpirun -np 2 mpiMM1fu 300

mpirun -np 2 mpiMM2f 300



**Prueba2**: Ejecución desde el manager2 a un nodo externo. 2 hilos, matriz de 300\*300:

mpirun --host worker12:2 ./mpiMM1c 300

mpirun --host worker12:2 ./mpiMM1f 300

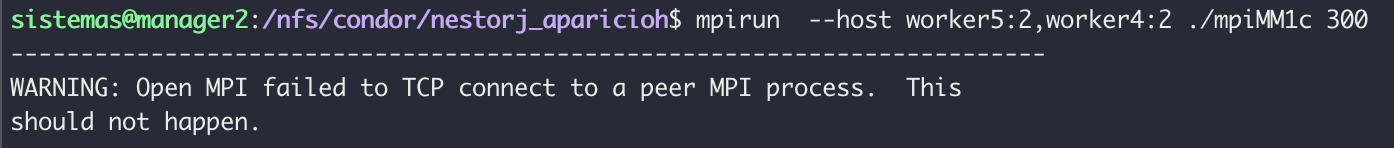
mpirun --host worker12:2 ./mpiMM1fu 300

mpirun --host worker12:2 ./mpiMM2f 300



**Prueba3**: Ejecución desde el manager2 a dos nodos externos. 2 hilos, matriz de 300\*300

Para los 4 ejecutables falló esta prueba.



Luego de realizar la revisión correspondiente se evidencia que a los server se les instaló Docker por lo que el comando no logra establecer las conexiones TCP a los servidores remotos, debido a la configuración de la tarjeta de red necesaria para que funcione la virtualización de Docker.

Referencias:

<https://github.com/open-mpi/ompi/blob/main/opal/mca/btl/tcp/help-mpi-btl-tcp.txt#L116>

<https://www.mail-archive.com/users@lists.open-mpi.org/msg34182.html>

Para solventar la situación, como se indica en los comandos anteriores, se hace necesario incluir el parámetro

--mca btl\_tcp\_if\_include eth0

Donde “eth0” corresponde a la interfaz de red que se va a utilizar en ambos nodos externos.

De esta manera el comando para poder hacer la ejecución de MPI en 2 nodos externos, a 2 hilos, para una matriz de 300\*300 es el siguiente:

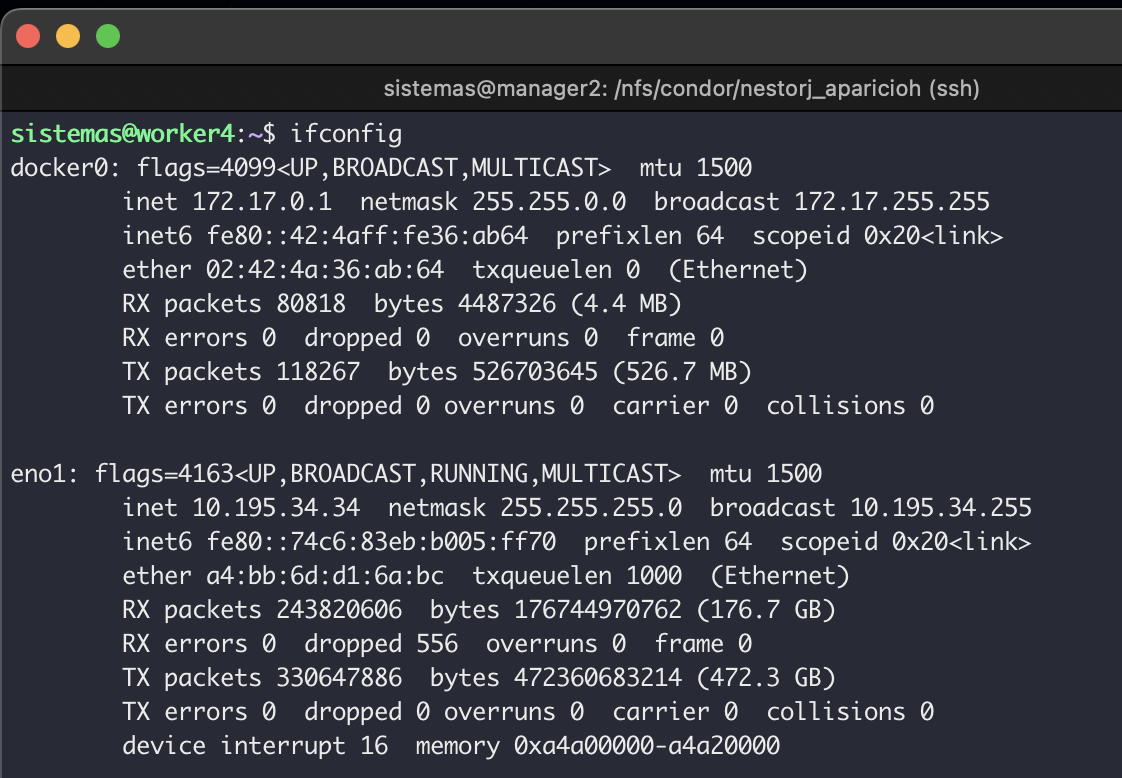
mpirun --mca btl\_tcp\_if\_include eno1 --host worker4:2,worker5:2 ./mpiMM1c 300

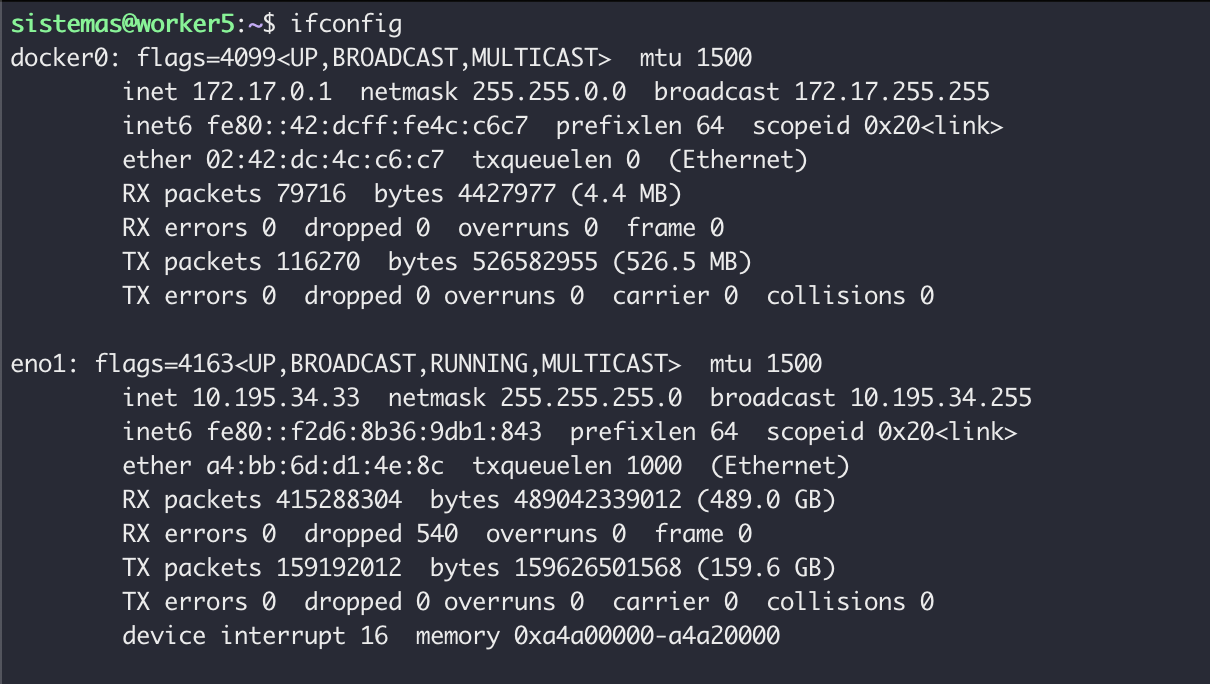
mpirun --mca btl\_tcp\_if\_include eno1 --host worker4:2,worker5:2 ./mpiMM1f 300

mpirun --mca btl\_tcp\_if\_include eno1 --host worker4:2,worker5:2 ./mpiMM1fu 300

mpirun --mca btl\_tcp\_if\_include eno1 --host worker4:2,worker5:2 ./mpiMM2f 300

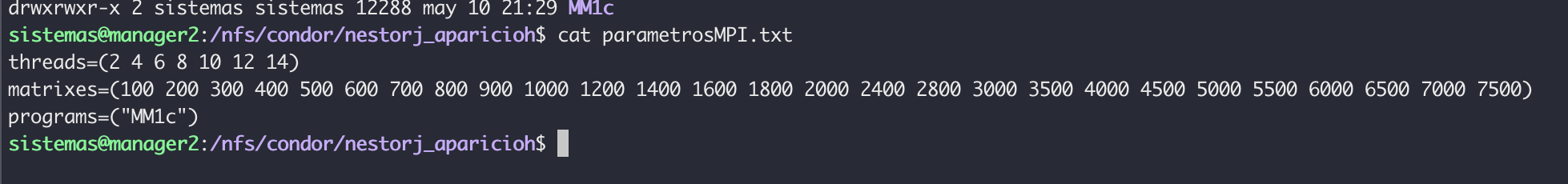
Donde “eno1” corresponde a la interfaz de red a utilizar en los nodos “worker4” y “worker5”





Se procede a crear el lanzador para la ejecución de los ejecutables en mpi. El programa estará divido en 2 partes. Un archivo txt que contendrá los parámetros de ejecución del experimento a nivel de hilos, tamaño de matriz y algotirmo a ejecutar.

Este es el contenido a modo de referencia del archivo para el ejercicio nombrado como “parametrosMPI.txt”:



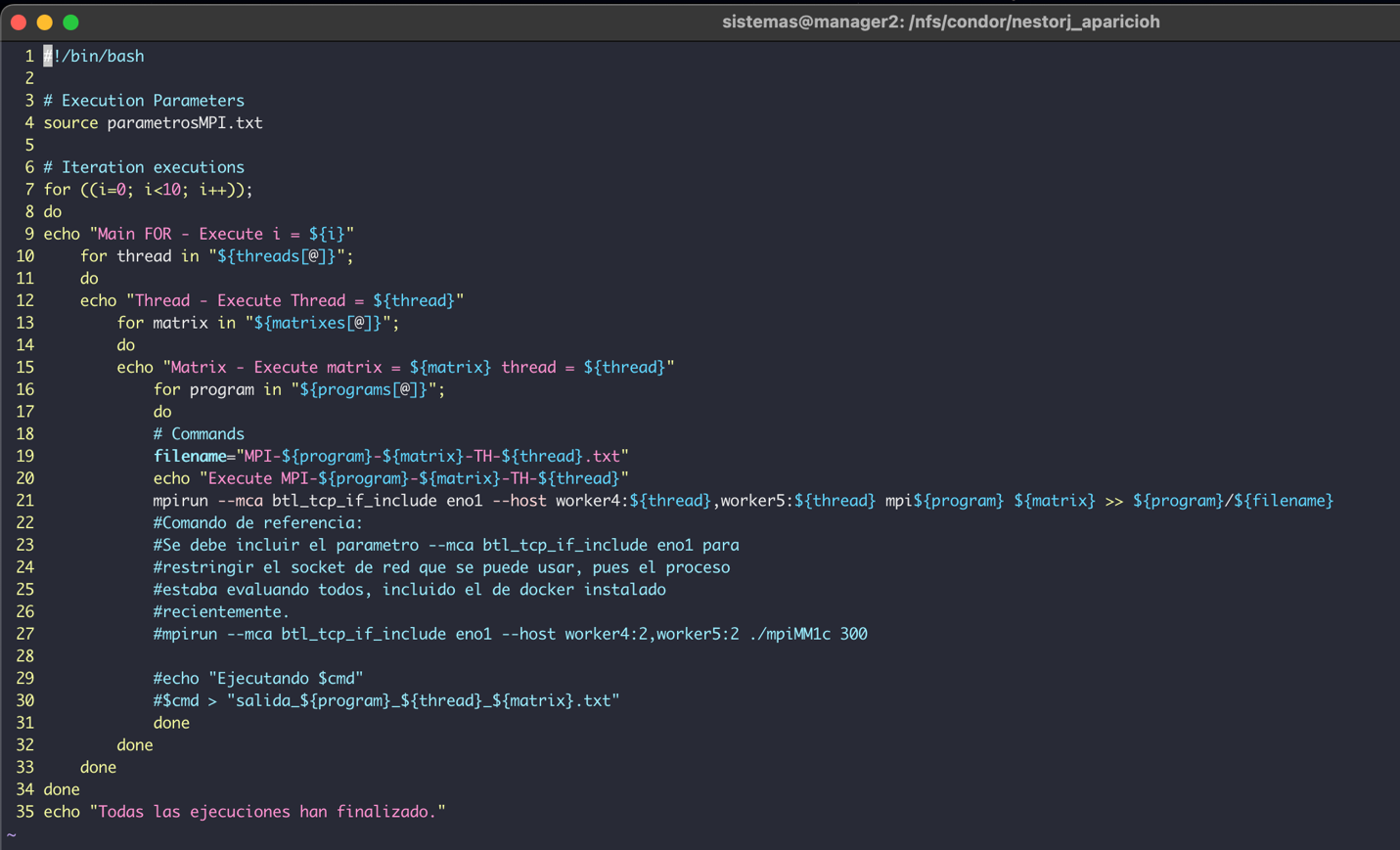
**Threads**: Corresponde a la cantidad de hilos a utilizar, se cargan como un arreglo, separados por espacios.

**Matrixes**: Se ingresa la lista del tamaño de matrices que se desea procesar.

**Programs**: Lista el o los programas que se correrán.

El archivo lanzador será el encargado de ejecutar las iteraciones programadas para llevar a cabo el experimento.

Se crea el archivo lanzadorMPI.sh



1 #!/bin/bash

2

3 # Execution Parameters

4 source parametrosMPI.txt

5

6 # Iteration executions

7 for ((i=0; i<10; i++));

8 do

9 echo "Main FOR - Execute i = ${i}"

10 for thread in "${threads[@]}";

11 do

12 echo "Thread - Execute Thread = ${thread}"

13 for matrix in "${matrixes[@]}";

14 do

15 echo "Matrix - Execute matrix = ${matrix} thread = ${thread}"

16 for program in "${programs[@]}";

17 do

18 # Commands

19 filename="MPI-${program}-${matrix}-TH-${thread}.txt"

20 echo "Execute MPI-${program}-${matrix}-TH-${thread}"

21 mpirun --mca btl\_tcp\_if\_include eno1 --host worker4:${thread},worker5:${thread} mpi${program} ${matrix} >> ${program}/${filename}

22 #Comando de referencia:

23 #Se debe incluir el parametro --mca btl\_tcp\_if\_include eno1 para

24 #restringir el socket de red que se puede usar, pues el proceso

25 #estaba evaluando todos, incluido el de docker instalado

26 #recientemente.

27 #mpirun --mca btl\_tcp\_if\_include eno1 --host worker4:2,worker5:2 ./mpiMM1c 300

28

29 #echo "Ejecutando $cmd"

30 #$cmd > "salida\_${program}\_${thread}\_${matrix}.txt"

31 done

32 done

33 done

34 done

35 echo "Todas las ejecuciones han finalizado."

**El archivo sh funciona de la siguiente manera:**

1 #!/bin/bash

2

3 # Execution Parameters

--Se carga el archive parametrosMPI.txt con las variables que utilizará la ejecución:

4 source parametrosMPI.txt

5

6 # Iteration executions

--Se inicializa un for para contar las iteraciones del experimento. Debido a que los jos en el server eran terminados sin razón se decidió llevar a cado el experimento solo con 10 iteraciones:

7 for ((i=0; i<10; i++));

8 do

9 echo "Main FOR - Execute i = ${i}"

--Se realiza lectura del arreglo de hilos a utilizar

10 for thread in "${threads[@]}";

11 do

12 echo "Thread - Execute Thread = ${thread}"

--Se realiza lectura del arreglo de tamaños de matrices a utilizar

13 for matrix in "${matrixes[@]}";

14 do

15 echo "Matrix - Execute matrix = ${matrix} thread = ${thread}"

--Se realiza lectura del arreglo de programas a utilizar

16 for program in "${programs[@]}";

17 do

18 # Commands

19 filename="MPI-${program}-${matrix}-TH-${thread}.txt"

20 echo "Execute MPI-${program}-${matrix}-TH-${thread}"

--Se ejecuta el commando mpi:

21 mpirun --mca btl\_tcp\_if\_include eno1 --host worker4:${thread},worker5:${thread} mpi${program} ${matrix} >> ${program}/${filename}

22 #Comando de referencia:

23 #Se debe incluir el parametro --mca btl\_tcp\_if\_include eno1 para

24 #restringir el socket de red que se puede usar, pues el proceso

25 #estaba evaluando todos, incluido el de docker instalado

26 #recientemente.

27 #mpirun --mca btl\_tcp\_if\_include eno1 --host worker4:2,worker5:2 ./mpiMM1c 300

28

29 #echo "Ejecutando $cmd"

30 #$cmd > "salida\_${program}\_${thread}\_${matrix}.txt"

31 done

32 done

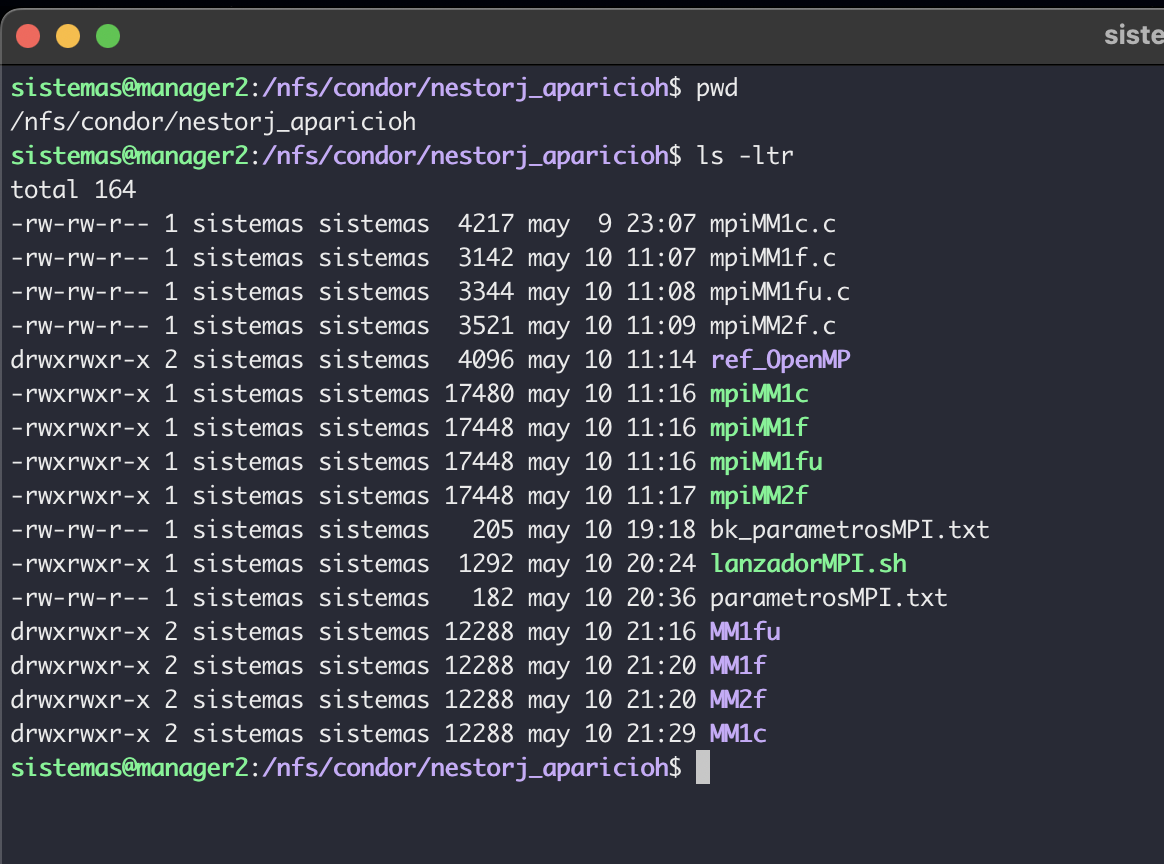
33 done

34 done

--Finalizado el ciclo de ejecución del comando mpi por programa \* tamaño de matrices \* cantidad de hilos \* iteraciones del experimento, se imprime este mensaje al log

35 echo "Todas las ejecuciones han finalizado."

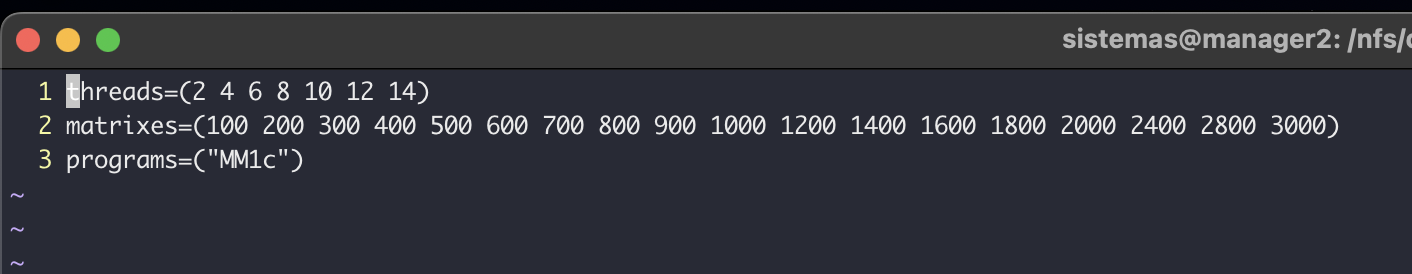
Con el fin de optimizar el uso de los recursos, se segmenta el ejercicio para ejecutar un lanzador por cada algoritmo, así se crea una carpeta para cada algoritmo:



Dentro de cada carpeta se copia, el algotirmo correspondiente, el archivo de parámetros y el archivo del lanzador:



Para cada algoritmo, el archivo de parámetros se define de la siguiente manera:



En cada carpeta de algoritmos, se ejecuta el lanzador con el siguiente comando:

nohup ./lanzadorMPI.sh &

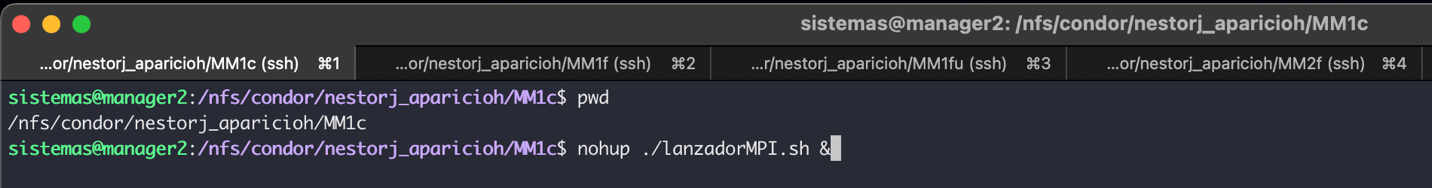
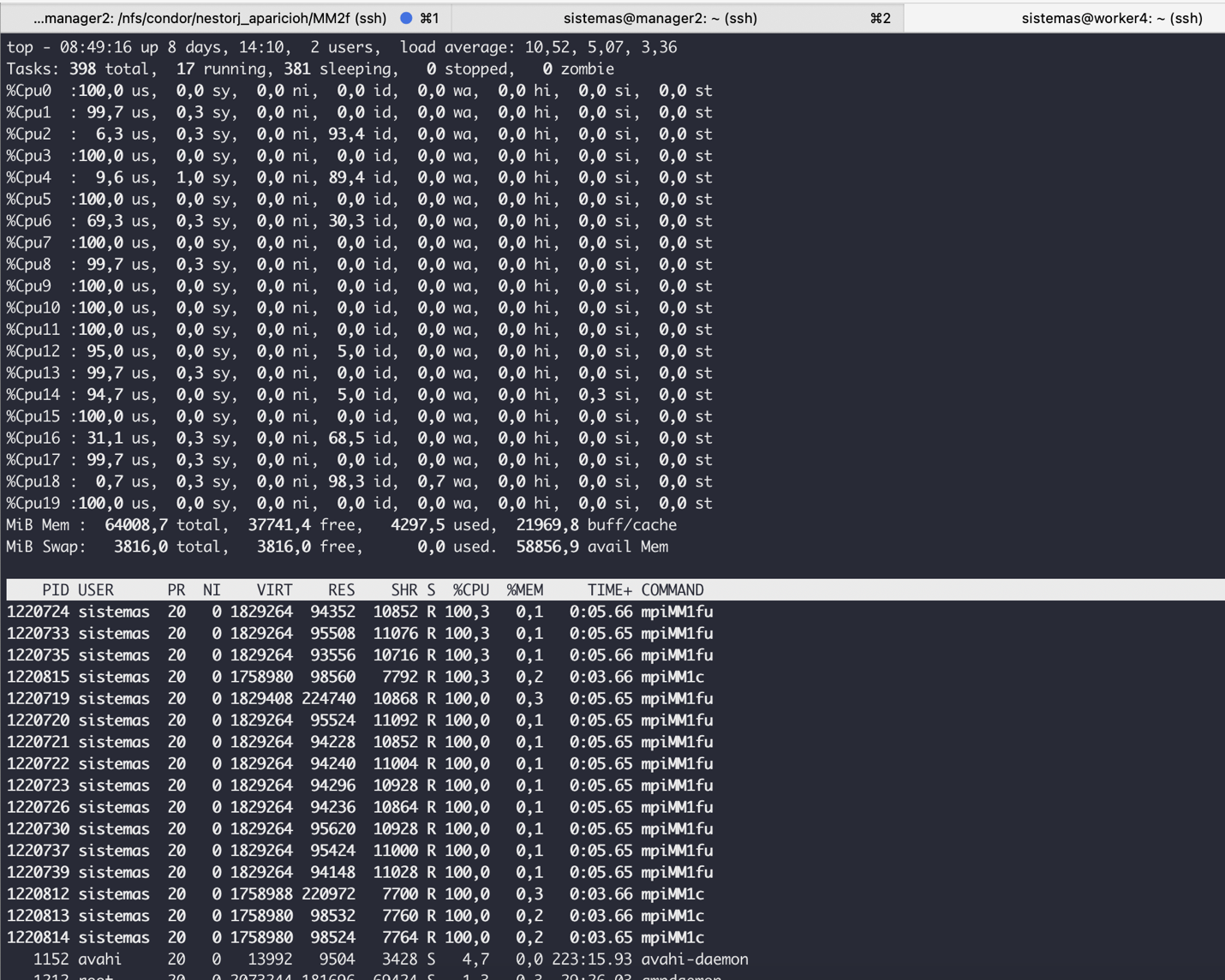
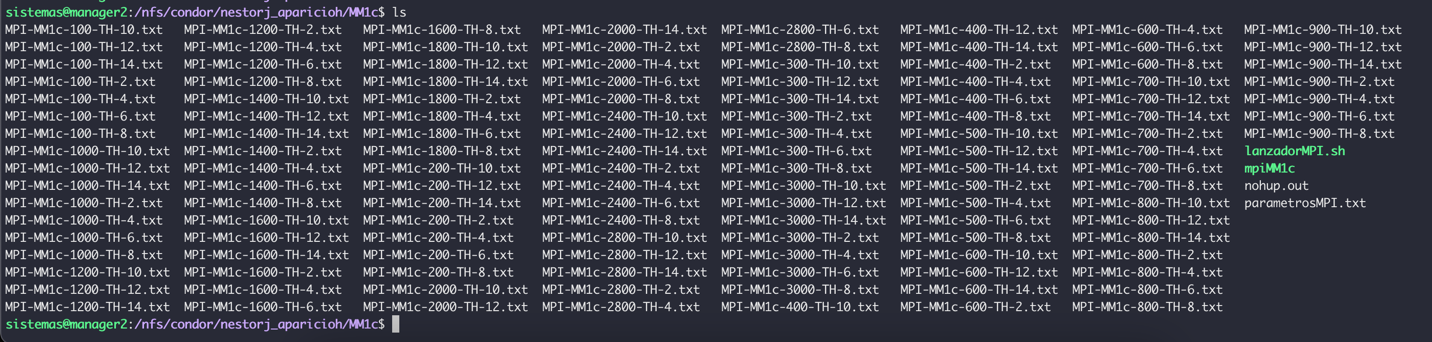


Imagen de referencia de como se ve el nodo trabajando:



Con la ejecución del lanzador, en cada carpeta, obtendremos un listado de archivos como el siguiente:



Al finalizar la ejecución de todos los algoritmos, se extrajo la información y se tabuló así:

Ver adjunto.