

Developing Soft and Parallel Programming Skills Using Project-Based Learning
Assignment 5
Fall 2019

Group Name: The Snakes
Group Members: Austin Yuille, Nabeeha Ashfaq, Jose Diaz, Matthew Hayes, Micah Robins

Planning and Scheduling

Name	Email	Task	Duration (hours)	Dependency
Jose Diaz	jdiaz28@student.gsu.edu	Managed the Github project and repository	6	Github and Raspberry Pi
Austin Yuille	ayuille1@student.gsu.edu	Created and directed video presentation	6	Youtube and video editor
Micah Robins	mrobins1@student.gsu.edu	Parallel Programming Discussion Questions	6	Google Docs and Raspberry Pi
Matt Hayes	mhayes37@student.gsu.edu	Team Coordinator and Task 3 Questions	6	Google Docs and Raspberry Pi
Nabeeha Ashfaq	nashfaq1@student.gsu.edu	Parallel Programming Report	6	Google Docs and Raspberry Pi

Parallel Programming Skills

a) Foundation:

(1) What are the basic steps (show all steps) in building a parallel program? Show at least one example.

Step one is to figure out a set of tasks that can be processed simultaneously. Once that has been figured out, there are a few questions that need to be answered. Can the data be broken down into equal sized chunks? Is the processing for each task going to be the same? Does there need to be communication between the tasks as they work? The answers to these questions will establish which parallel technique should be used. If the data can be split into equal size chunks then static load balancing can be used. If not, then dynamic load balancing will be needed. If the processing for each task is the same and there does not need to be any communication between the tasks as they work, then each task can be easily split into parallel without a need to synchronize the tasks.

Once all of this information is known, then the program can be set to follow these four processes: One, decomposition is the process of breaking down the problem into the individual tasks, Two, assigning the tasks to their particular processes if the processing for each task must be different. Three, orchestrating how each thread can access data, communicate with other threads, and synchronize their processes. And four, mapping each thread to a particular processor to handle it.

In the provided reading, there was an example that used the master/worker technique to compute an approximation of π . The program does this by placing a circle inside of a square whose dimensions are equal to the circle's diameter. And then it calculates the difference in the area between the square and the circle. In this case, the data was easily broken down into equal sized chunks. Each task picks a random point inside the square and checks if it is also inside the circle. The final answer is found by dividing the number of points in the circle by the number of points in the square, and then multiplying that answer by 4. So each thread can be assigned the same number of points to measure. The processing for each task is the same, and the threads do not need to communicate or synchronize with each other.

Decomposition of the problem breaks down to each task checking if a random point is inside of the circle or not. The second process is simple: every task is assigned the same process. The only thing that needs to be orchestrated between the threads is keeping count of how many points lie within the circle, which is passed to the master thread by each worker and added together.

(2) What is MapReduce?

MapReduce is a system developed by Google to process very large amounts of data. It is a parallel process that can scan large amounts of data by spreading it across many machines. It uses a function to process the input data, and then combines all of the elements produced by the function. For instance, MapReduce could be used to count how many times every word in the dictionary is used.

(3) What is map and what is reduce?

Map takes in a data set and a function as its input. It scans through all of the data and applies the function to it. Reduce combines elements of a sequence. It merges together those elements to create a smaller set of elements.

(4) Why MapReduce?

MapReduce allows the programmer to apply simple calculations across huge data sets, without having to worry about the details of building a parallel program to do it. All of the parallelization, load balancing, mapping, synchronization, etc is handled by the MapReduce library.

(5) Show an example for MapReduce.

As mentioned above, MapReduce could be used to count how many times every word in the dictionary is used. By passing the document (dictionary) to the map function it will split the document in to smaller chunks that can be processed by threads in parallel. Each thread will keep a count for the words of it's particular piece of the dictionary, and then pass that information to the reduce function. The reduce function will collect these lists of word counts and then sum them together to create a final result for how many instances of each word occurred.

(6) Explain in your own words how MapReduce model is executed?

There is a master system that controls the flow of the MapReduce process. The master distributes the tasks to other systems performing the Map function. As these other systems complete their tasks and return the results from their Map work, the master will pass the data to systems running the Reduce function. Once the Reduce task is finished the master will return the final results.

(7) List and describe three examples that are expressed as MapReduce computations.

- 1) A count of URL access frequency. The map function processes logs of web page requests and outputs. The reduce function adds together all values for the URL and emits the pair.
- 2) Term vector per host. A term vector summarizes the most important words that occur in a document as a list of word and frequency pairs. The map function emits a hostname and term vector pair for each input. The reduce function adds the term vectors together, throwing away any infrequent pairs, and then emits a final hostname and term vector pair.
- 3) A reverse web link graph. The map function outputs target and source pairs for each link to a target URL found in a source page. The reduce function concatenates the list of all source URLs associated with a target URL and emits the pair of target and list(source).

(8) When do we use OpenMP, MPI, and MapReduce (Hadoop), and why?

OpenMP:

OpenMP is a tool for adding shared memory parallelism to code. It is an efficient directive based library. OpenMP is useful for running a for loop in parallel. Using OpenMP, put the directive `#pragma parallel` before the for loop and it'll split the loop into multiple threads, so each of them would handle a chunk of the loop's iterations.

MPI:

Message Passing Interface is a distributed memory parallel model implementation, typically used to develop parallel scientific applications. It usually uses tightly synchronized code. This is important because having slow threads makes an MPI program inefficient. MPI works well for very specific types of applications.

Hadoop MapReduce:

MapReduce uses two processes that can be applied over large amounts of data. The first is Map and the second process is Reduce. The data is distributed over multiple machines, and you can apply some operation for each data element. Then some kind of reduction is applied to the results. MapReduce is a simple process, but can be applied to complex problems. Hadoop MapReduce works well when there is a very large amount of data that needs to be combed through.

(9) In your own words, explain what a drug design and DNA problem is in no more than 150 words.

The drug uses ligands, which attach to protein strands and change the shape of the protein. By changing the protein's shape we have effectively changed what the protein accomplishes within the body. The problem is that there are many different variations of ligands that can be produced. Some of them will fit in with the protein and create the resulting shape that we desire. However, some will fit the protein but do not create the shape we want. And most ligands will not even fit the protein. As a result, there are massive amounts of ligands that must be tested. Ligands of different lengths, and various permutations of each length until we find one that creates the result we desire. Each one must be constructed and tested to see if it will fit and produce the desired shape of the protein.

b) Parallel Programming Basics:

The programming part of Task 3 was about solving drug design problem, by finding the amount ligands in a protein. We were given three programs that did so using the `Generate_task()`, `Map()`, and `Reduce()` functions. Each program called these functions and ran them in different ways. We then found out how long each program took to run according to how many threads of ligands were found by using the `time p` instruction.

The first program, `dd_serial`, solves the problem sequentially. It first called on the `Generate_task()` function, which produces pairs of ligands and proteins. Next, the `Map()` function, which is applied to ligands and protein pairs. The binding score is computed for each pair and the function will return a score, ligand pair. The `Reduce()` function is then applied to these pairs in order to identify the ligands with the highest scores. When implemented the time for the program with a maximum of 7 ligands to run was 123.57 seconds. We found this result by running the program three times and then finding the average of our results. You can see our result in Appendix B, 1.

The next program, `dd_omp`, uses OpenMP to solve the problem. Here, the program turns the `Map()` function into a parallel function by turning it to a for loop and then adding OpenMP threads. When solving the problem with one thread, the average time after running the program three times was 122.73 seconds. The results can be seen in Appendix B, 2. When running the program with two threads, the average time was 104.97 seconds. The time for three threads was 90.29 seconds and the time for 4 threads was 25.19 seconds. These results can be seen in Appendix B, 4.

The third program, `threads`, used C++ 11 standard threads instead of OpenMP threads. Here the threads have to be created and managed by the users by using the master-worker parallel programming we used in previous assignments. Here, the `do_Map()` portion pops a new ligand string from the queue and calls `Map()` until it arrives at the end marker. When working with one thread, the average time for the program to run was 122.73 seconds, this can be seen in Appendix B, 3. For two threads the time was 77.37 and for three threads the time was 54.69. These results can be found in Appendix B, 4. From these results we can confirm that parallel programming is more efficient as the run time decreases when the thread count increases. We have also found that the C++ 11 threads work faster than the OpenMP threads.

Parallel Programming Discussion Questions:

Measure Run-Time (See Appendix C,6 and C,7):

Implem entation	Avg Time (s)	Time(s) 2 Threads	Time(s) 3 Threads	Time(s) 4 Threads		2nd attempt	2 threads	3 threads	4 threads
dd serial	0.01								
dd omp	0.013	0.02	0.053	0.21			104.97	90.26	75.19
dd threads	0.016	0.02	0.05				77.37	54.69	41.74

(1) Which approach is fastest?

Out of the three option dd_threads (C++) is fastest . (See Appendix C,6)

(2) Determine the number of lines in each file (use wc -l). How does the C++11 implementation compare to the OpenMP implementations?

From least amount of lines to greatest: dd_serial contains the least amount of lines at 63 (See Appendix C,3). dd_threads has the second least at 137 (See Appendix C,2). dd_omp has the most at 152 (See Appendix, C,3).

(3) Increase the number of threads to 5 threads. What is the run time for each?

The time for dd_threads is 42.32 (See Appendix C,4). The time for dd_omp is 65.73 (See Appendix C,5).

(4) Increase the maximum ligand length to 7, and rerun each program. What is the run time for each?

Time for dd_threads is 77.37, 54.69, and 41.74 for 2, 3, and 4 threads respectively (See Appendix C,7). Time for dd_omp is 104.97, 90.26, and 75.19 for 2, 3, and 4 threads respectively (See Appendix C,6) Time for serial is 123.57 see Appendix (C,2). See (Appendix C,1) just for values.

Appendix A: Links

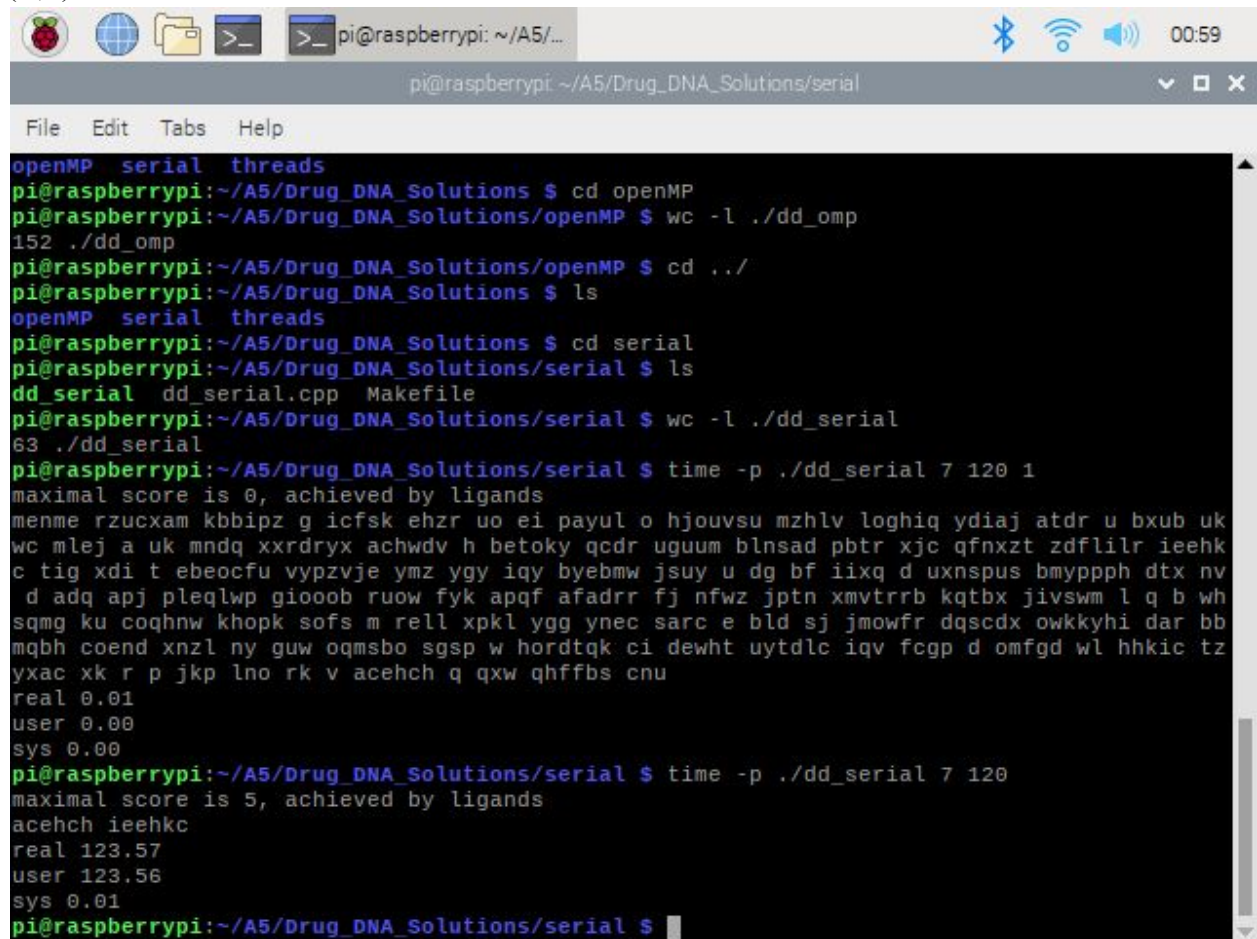
Github Project: <https://github.com/orgs/thesnakes-csc3210/projects/1>

Github Repository: <https://github.com/thesnakes-csc3210/projectA5>

Video Presentation: <https://youtu.be/iQPE292oKrQ>

Appendix B: Parallel Programming Task

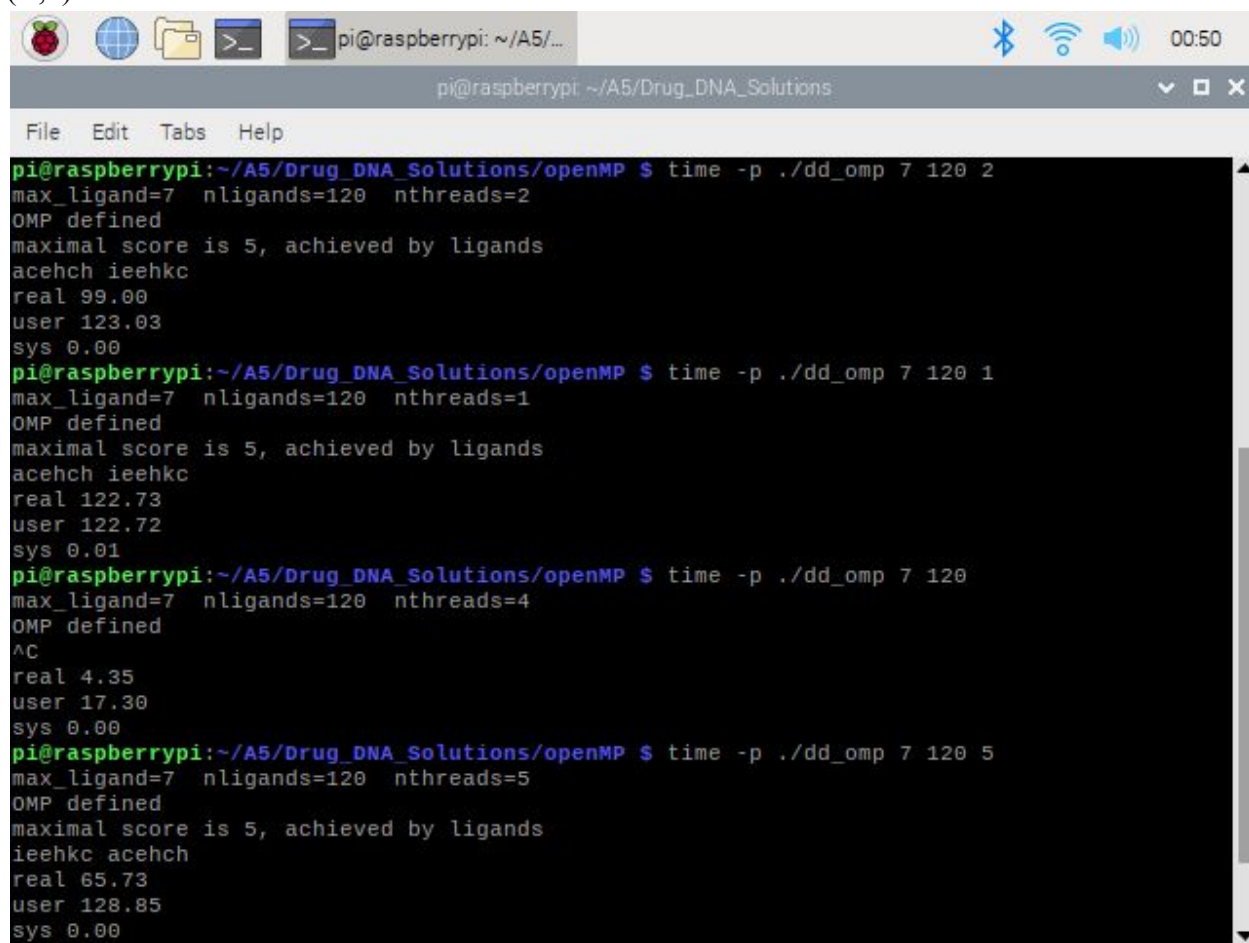
(B,1)



```
pi@raspberrypi: ~/A5/...
pi@raspberrypi: ~/A5/Drug_DNA_Solutions/serial
File Edit Tabs Help
openMP serial threads
pi@raspberrypi:~/A5/Drug_DNA_Solutions $ cd openMP
pi@raspberrypi:~/A5/Drug_DNA_Solutions/openMP $ wc -l ./dd_omp
152 ./dd_omp
pi@raspberrypi:~/A5/Drug_DNA_Solutions/openMP $ cd ../
pi@raspberrypi:~/A5/Drug_DNA_Solutions $ ls
openMP serial threads
pi@raspberrypi:~/A5/Drug_DNA_Solutions $ cd serial
pi@raspberrypi:~/A5/Drug_DNA_Solutions/serial $ ls
dd_serial dd_serial.cpp Makefile
pi@raspberrypi:~/A5/Drug_DNA_Solutions/serial $ wc -l ./dd_serial
63 ./dd_serial
pi@raspberrypi:~/A5/Drug_DNA_Solutions/serial $ time -p ./dd_serial 7 120 1
maximal score is 0, achieved by ligands
menme rzucxam kbbipz g icfsk ehxr uo ei payul o hjouvsu mzhlv loghiq ydiaj atdr u bxub uk
wc mlej a uk mndq xxrdryx achwdv h betoky qcdr uguum blnsad pbtr xjc qfnxzt zdflilr ieehk
c tig xdi t ebeocfu vypzvje ymz ygy iqy byebmw jsuy u dg bf iixq d uxnsps bmyppph dtx nv
d adq apj pleglwp giooob ruow fyk apqf afadrr fj nfwz jptn xmvtrrb kqtbx jivswm l q b wh
sqmg ku coqhnw khopk sofs m rell xpkf ygg ynec sarc e bld sj jmowfr dqscdx owkkyhi dar bb
mqbh coend xnzl ny guw oqmsbo sgsp w hordtqk ci dewht uytdlc iqv fcgp d omfgd wl hkhic tz
yxac xk r p jkp lno rk v acehch q qwx qhffbs cnu
real 0.01
user 0.00
sys 0.00
pi@raspberrypi:~/A5/Drug_DNA_Solutions/serial $ time -p ./dd_serial 7 120
maximal score is 5, achieved by ligands
acehch ieehkc
real 123.57
user 123.56
sys 0.01
pi@raspberrypi:~/A5/Drug_DNA_Solutions/serial $
```

The run time for dd_serial program.

(B,2)



The screenshot shows a terminal window titled "pi@raspberrypi: ~/A5/Drug_DNA_Solutions". The terminal displays the execution of the `dd_omp` program with various parameters. The output for each run includes the maximal score, the ligands achieving it, and the execution time (real, user, sys).

```
pi@raspberrypi:~/A5/Drug_DNA_Solutions/openMP $ time -p ./dd_omp 7 120 2
max_ligand=7 nligands=120 nthreads=2
OMP defined
maximal score is 5, achieved by ligands
acehch ieehkc
real 99.00
user 123.03
sys 0.00
pi@raspberrypi:~/A5/Drug_DNA_Solutions/openMP $ time -p ./dd_omp 7 120 1
max_ligand=7 nligands=120 nthreads=1
OMP defined
maximal score is 5, achieved by ligands
acehch ieehkc
real 122.73
user 122.72
sys 0.01
pi@raspberrypi:~/A5/Drug_DNA_Solutions/openMP $ time -p ./dd_omp 7 120
max_ligand=7 nligands=120 nthreads=4
OMP defined
^C
real 4.35
user 17.30
sys 0.00
pi@raspberrypi:~/A5/Drug_DNA_Solutions/openMP $ time -p ./dd_omp 7 120 5
max_ligand=7 nligands=120 nthreads=5
OMP defined
maximal score is 5, achieved by ligands
ieehkc acehch
real 65.73
user 128.85
sys 0.00
```

Run time for `dd_omp` program.

(B, 3)

```

pi@raspberrypi: ~/A5/...
pi@raspberrypi: ~/A5/Drug_DNA_Solutions/threads
File Edit Tabs Help
real 65.73
user 128.85
sys 0.00
pi@raspberrypi:~/A5/Drug_DNA_Solutions/openMP $ ../
bash: ../: Is a directory
pi@raspberrypi:~/A5/Drug_DNA_Solutions/openMP $ cd ../
pi@raspberrypi:~/A5/Drug_DNA_Solutions $ ls
openMP  serial  threads
pi@raspberrypi:~/A5/Drug_DNA_Solutions $ cd threads
pi@raspberrypi:~/A5/Drug_DNA_Solutions/threads $ time -p ./dd_omp 7 120 1
bash: ./dd_omp: No such file or directory
real 0.00
user 0.00
sys 0.00
pi@raspberrypi:~/A5/Drug_DNA_Solutions/threads $ time -p ./dd_threads 7 120 1
max_ligand=7 nligands=120 nthreads=1
maximal score is 5, achieved by ligands
acehch ieehkc
real 122.78
user 122.75
sys 0.02
pi@raspberrypi:~/A5/Drug_DNA_Solutions/threads $ time -p ./dd_threads 7 120 5
max_ligand=7 nligands=120 nthreads=5
maximal score is 5, achieved by ligands
acehch ieehkc
real 42.32
user 139.63
sys 0.00
pi@raspberrypi:~/A5/Drug_DNA_Solutions/threads $ wc -l ./dd_threads
137 ./dd_threads
pi@raspberrypi:~/A5/Drug_DNA_Solutions/threads $

```

Run time for dd_threads program.

(B, 4)

	A	B	C	D	E	F	G	H
1	Implementation	Time						
2	dd_serial	123.57						
3	dd_omp	122.73						
4	dd_threads	122.73						
5								
6								
7	Implementation	Time 2 Threads	Time 3 Threads	Time 4 Threads	Time 5 Threads			
8	dd_omp	104.97	90.29	75.19	65.73			
9	dd_threads	77.37	54.69	41.74	42.32			
10								
11	Implementation	Line count(wc -l)						
12	dd_serial	63						
13	dd_omp	152						
14	dd_threads	137						
15								
16								

Spreadsheet of run times and line count for the three programs.

Appendix C: Parallel Programming Discussion Questions

(C,1)

	A	B	C	D	E	F	G	H	I	J	K
1	Implementation	Time									
2	dd_serial	123.57									
3	dd_omp	122.73									
4	dd_threads	122.73									
5											
6											
7	Implementation	Time 2 Threads	Time 3 Threads	Time 4 Threads	Time 5 Threads						
8	dd_omp	104.97	90.29	75.19	65.73						
9	dd_threads	77.37	54.69	41.74	42.32						
10											
11	Implementation	Line count(wc -l)									
12	dd_serial	63									
13	dd_omp	152									
14	dd_threads	137									

Spreadsheet of the times and line count for dd_serial, dd_omp, and dd_threads.

(C,2)

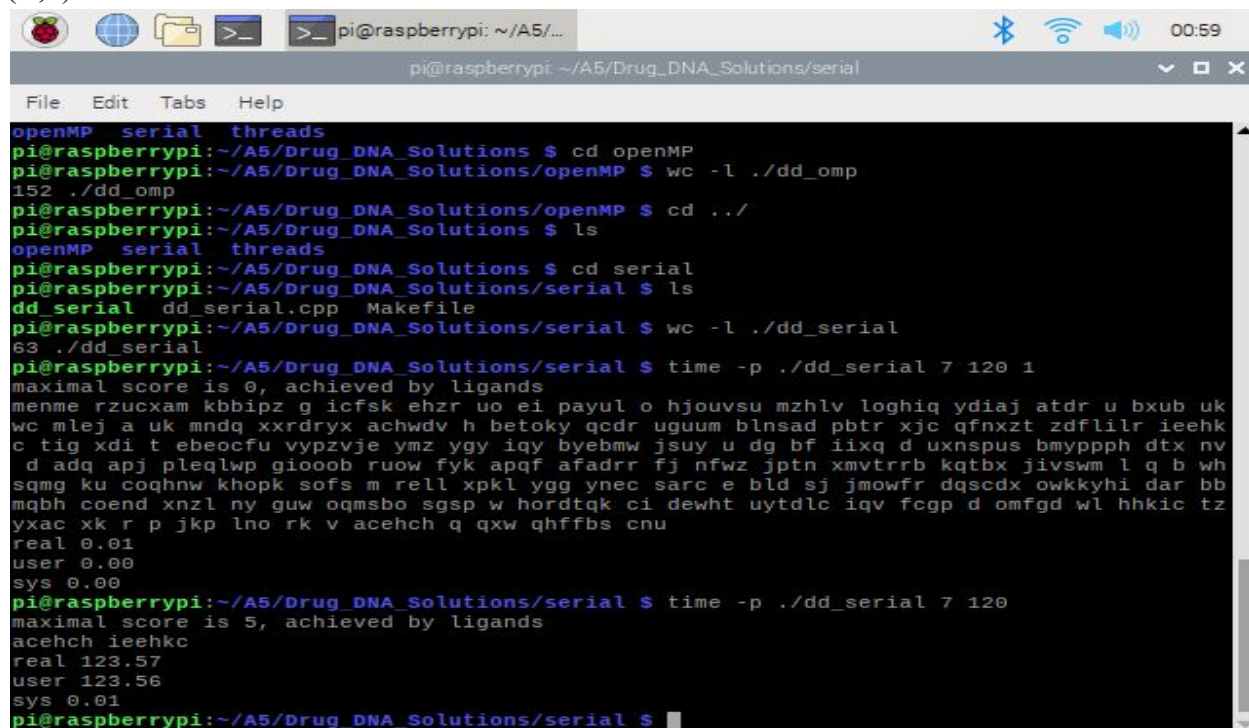
```

pi@raspberrypi: ~/A5/...
pi@raspberrypi: ~/A5/Drug_DNA_Solutions/openMP
File Edit Tabs Help
pi@raspberrypi:~/A5/Drug_DNA_Solutions $ ls
openMP serial threads
pi@raspberrypi:~/A5/Drug_DNA_Solutions $ cd threads
pi@raspberrypi:~/A5/Drug_DNA_Solutions/threads $ time -p ./dd_omp 7 120 1
bash: ./dd_omp: No such file or directory
real 0.00
user 0.00
sys 0.00
pi@raspberrypi:~/A5/Drug_DNA_Solutions/threads $ time -p ./dd_threads 7 120 1
max_ligand=7 nligands=120 nthreads=1
maximal score is 5, achieved by ligands
acehch ieehkc
real 122.78
user 122.75
sys 0.02
pi@raspberrypi:~/A5/Drug_DNA_Solutions/threads $ time -p ./dd_threads 7 120 5
max_ligand=7 nligands=120 nthreads=5
maximal score is 5, achieved by ligands
acehch ieehkc
real 42.32
user 139.63
sys 0.00
pi@raspberrypi:~/A5/Drug_DNA_Solutions/threads $ wc -l ./dd_threads
137 ./dd_threads
pi@raspberrypi:~/A5/Drug_DNA_Solutions/threads $ cd ../
pi@raspberrypi:~/A5/Drug_DNA_Solutions $ ls
openMP serial threads
pi@raspberrypi:~/A5/Drug_DNA_Solutions $ cd openMP
pi@raspberrypi:~/A5/Drug_DNA_Solutions/openMP $ wc -l ./dd_omp
152 ./dd_omp
pi@raspberrypi:~/A5/Drug_DNA_Solutions/openMP $

```

Line count for dd_threads and dd_omp.

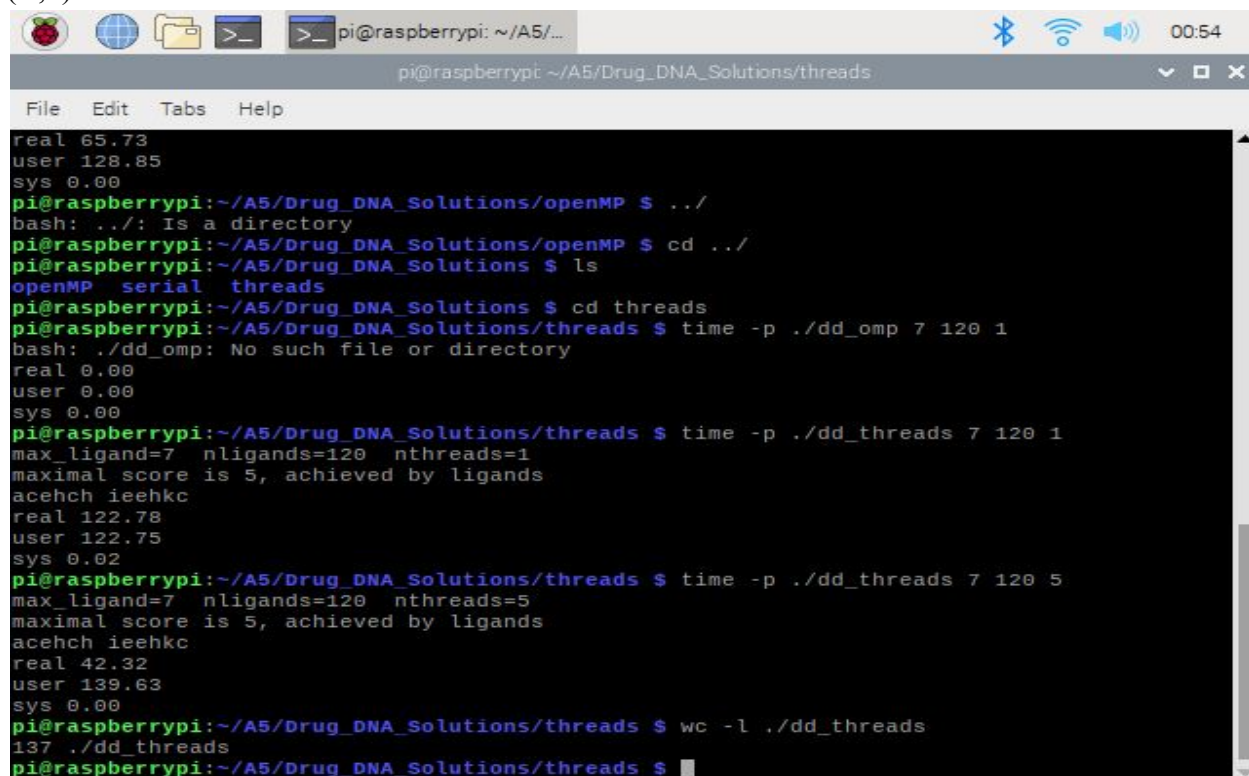
(C,3)



```
pi@raspberrypi: ~/A5/...
pi@raspberrypi: ~/A5/Drug_DNA_Solutions/serial
openMP serial threads
pi@raspberrypi:~/A5/Drug_DNA_Solutions $ cd openMP
pi@raspberrypi:~/A5/Drug_DNA_Solutions/openMP $ wc -l ./dd_omp
152 ./dd_omp
pi@raspberrypi:~/A5/Drug_DNA_Solutions/openMP $ cd ../
pi@raspberrypi:~/A5/Drug_DNA_Solutions $ ls
openMP serial threads
pi@raspberrypi:~/A5/Drug_DNA_Solutions $ cd serial
pi@raspberrypi:~/A5/Drug_DNA_Solutions/serial $ ls
dd_serial dd_serial.cpp Makefile
pi@raspberrypi:~/A5/Drug_DNA_Solutions/serial $ wc -l ./dd_serial
63 ./dd_serial
pi@raspberrypi:~/A5/Drug_DNA_Solutions/serial $ time -p ./dd_serial 7 120 1
maximal score is 0, achieved by ligands
menme rzucxam kbbipz g icfsk ehzr uo ei payul o hjouvsu mzhlv loghiq ydiaj atdr u bxub uk
wc mlej a uk mndq xxrdrx achwdv h betoky qcdr uguum blnsad pbtr xjc qfnxzt zdfllir ieehk
c tig xdi t ebeocfu vypzve ymz ygy iqy byebmw jsuy u dg bf iixq d uxnsps bmyppph dtx nv
d adq apj pleqlwp giooob ruow fyk apqf afadrr fj nfwz jptn xmvtrrb kqtbx jivswm l q b wh
sqmg ku coqhnw khopk sofs m rell xpkf ygg ynec sarc e bld sj jmowfr dqscdx owkkyhi dar bb
mqbh coend xnzl ny quw oqmsbo sgsp w hordtqk ci dewht uytdlc iqv fcgp d omfgd wl hkkic tz
yxac xk r p jkp lno rk v acehch q qwx qhffbs cnu
real 0.01
user 0.00
sys 0.00
pi@raspberrypi:~/A5/Drug_DNA_Solutions/serial $ time -p ./dd_serial 7 120
maximal score is 5, achieved by ligands
acehch ieehkc
real 123.57
user 123.56
sys 0.01
pi@raspberrypi:~/A5/Drug_DNA_Solutions/serial $
```

Line count for dd_serial.

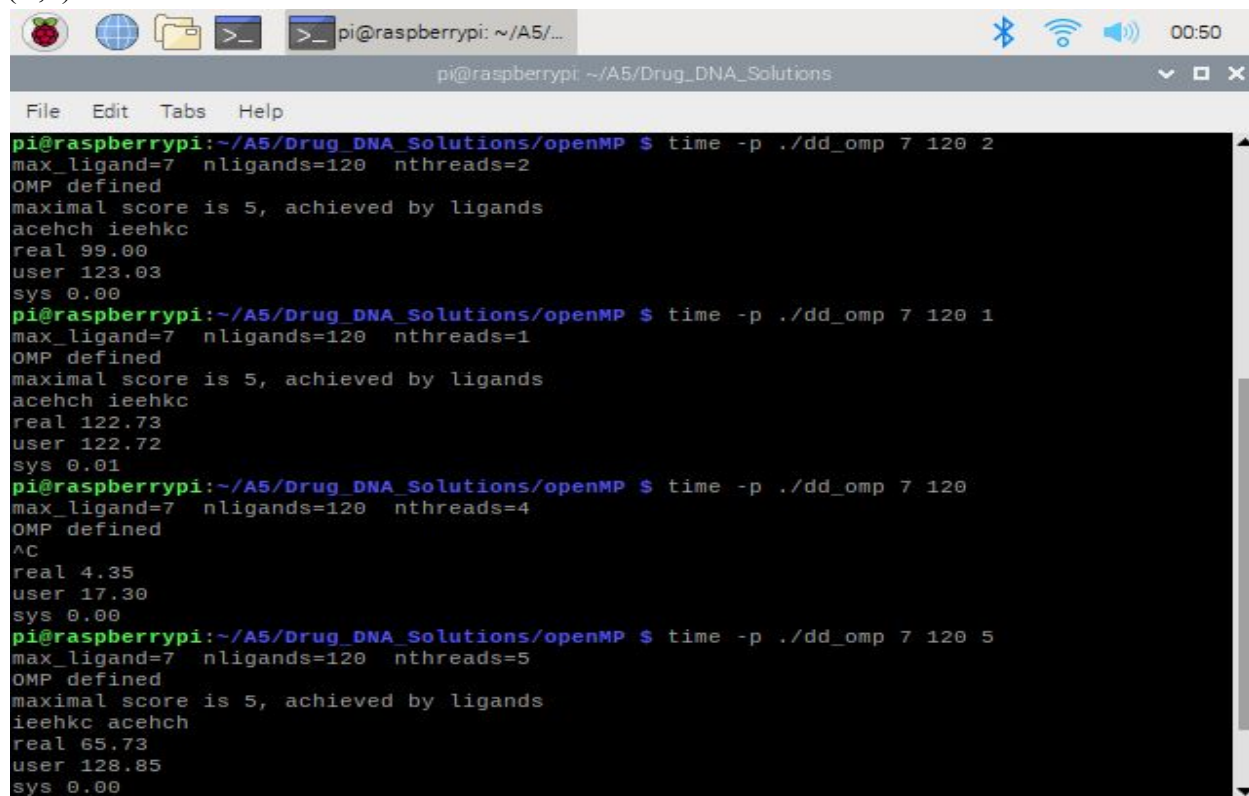
(C,4)



```
pi@raspberrypi: ~/A5/...
pi@raspberrypi: ~/A5/Drug_DNA_Solutions/threads
real 65.73
user 128.85
sys 0.00
pi@raspberrypi:~/A5/Drug_DNA_Solutions/openMP $ ../
bash: ../: Is a directory
pi@raspberrypi:~/A5/Drug_DNA_Solutions/openMP $ cd ../
pi@raspberrypi:~/A5/Drug_DNA_Solutions $ ls
openMP serial threads
pi@raspberrypi:~/A5/Drug_DNA_Solutions $ cd threads
pi@raspberrypi:~/A5/Drug_DNA_Solutions/threads $ time -p ./dd_omp 7 120 1
bash: ./dd_omp: No such file or directory
real 0.00
user 0.00
sys 0.00
pi@raspberrypi:~/A5/Drug_DNA_Solutions/threads $ time -p ./dd_threads 7 120 1
max_ligand=7 nligands=120 nthreads=1
maximal score is 5, achieved by ligands
acehch ieehkc
real 122.78
user 122.75
sys 0.02
pi@raspberrypi:~/A5/Drug_DNA_Solutions/threads $ time -p ./dd_threads 7 120 5
max_ligand=7 nligands=120 nthreads=5
maximal score is 5, achieved by ligands
acehch ieehkc
real 42.32
user 139.63
sys 0.00
pi@raspberrypi:~/A5/Drug_DNA_Solutions/threads $ wc -l ./dd_threads
137 ./dd_threads
pi@raspberrypi:~/A5/Drug_DNA_Solutions/threads $
```

Time for dd_threads at 5 threads. Time for dd_threads at 7 max_ligand.

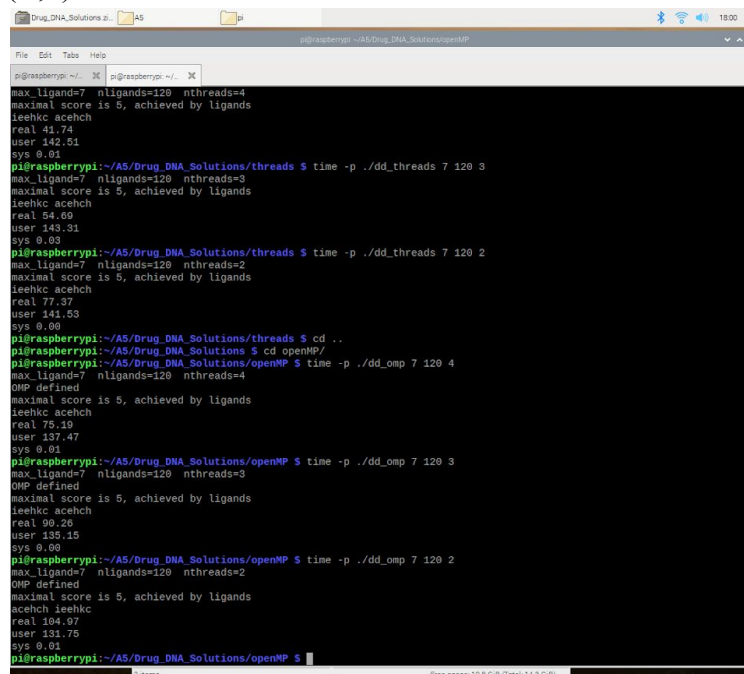
(C,5)



```
pi@raspberrypi: ~/A5/...
pi@raspberrypi: ~/A5/Drug_DNA_Solutions
File Edit Tabs Help
pi@raspberrypi:~/A5/Drug_DNA_Solutions/openMP $ time -p ./dd_omp 7 120 2
max_ligand=7 nligands=120 nthreads=2
OMP defined
maximal score is 5, achieved by ligands
acehch ieehkc
real 99.00
user 123.03
sys 0.00
pi@raspberrypi:~/A5/Drug_DNA_Solutions/openMP $ time -p ./dd_omp 7 120 1
max_ligand=7 nligands=120 nthreads=1
OMP defined
maximal score is 5, achieved by ligands
acehch ieehkc
real 122.73
user 122.72
sys 0.01
pi@raspberrypi:~/A5/Drug_DNA_Solutions/openMP $ time -p ./dd_omp 7 120
max_ligand=7 nligands=120 nthreads=4
OMP defined
AC
real 4.35
user 17.30
sys 0.00
pi@raspberrypi:~/A5/Drug_DNA_Solutions/openMP $ time -p ./dd_omp 7 120 5
max_ligand=7 nligands=120 nthreads=5
OMP defined
maximal score is 5, achieved by ligands
ieehkc acehch
real 65.73
user 128.85
sys 0.00
```

Time for dd_omp at 5 threads. Time for dd_omp at 7 max_ligand.

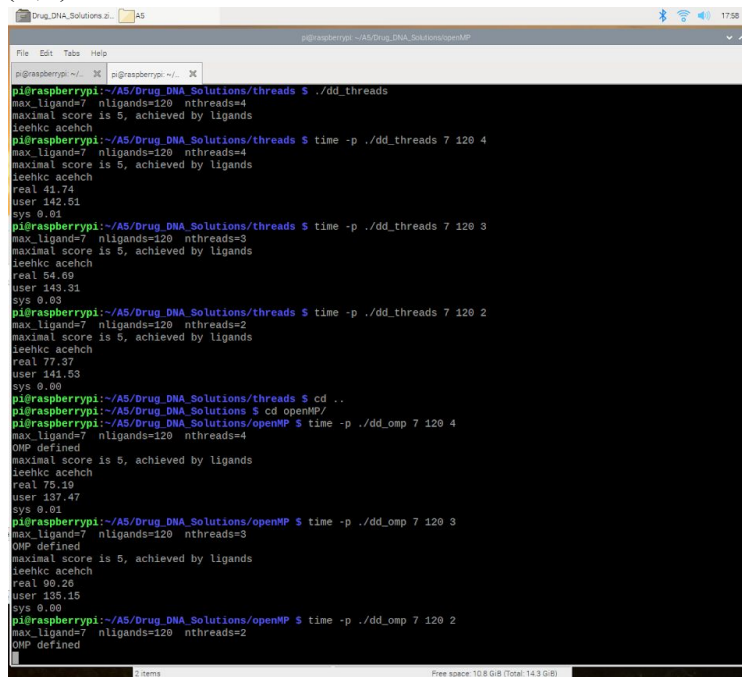
(C,6)



```
Drug_DNA_Solutions.zi A5 pi
pi@raspberrypi: ~/A5/Drug_DNA_Solutions/openMP
File Edit Tabs Help
pi@raspberrypi:~/A5/Drug_DNA_Solutions/openMP $ time -p ./dd_threads 7 120 4
max_ligand=7 nligands=120 nthreads=4
OMP defined
maximal score is 5, achieved by ligands
ieehkc acehch
real 41.74
user 142.51
sys 0.01
pi@raspberrypi:~/A5/Drug_DNA_Solutions/threads $ time -p ./dd_threads 7 120 3
max_ligand=7 nligands=120 nthreads=3
OMP defined
maximal score is 5, achieved by ligands
ieehkc acehch
real 54.69
user 143.31
sys 0.00
pi@raspberrypi:~/A5/Drug_DNA_Solutions/threads $ time -p ./dd_threads 7 120 2
max_ligand=7 nligands=120 nthreads=2
OMP defined
maximal score is 5, achieved by ligands
ieehkc acehch
real 77.37
user 141.53
sys 0.00
pi@raspberrypi:~/A5/Drug_DNA_Solutions/threads $ cd ..
pi@raspberrypi:~/A5/Drug_DNA_Solutions $ cd openMP/
pi@raspberrypi:~/A5/Drug_DNA_Solutions/openMP $ time -p ./dd_omp 7 120 4
max_ligand=7 nligands=120 nthreads=4
OMP defined
maximal score is 5, achieved by ligands
ieehkc acehch
real 75.19
user 137.47
sys 0.01
pi@raspberrypi:~/A5/Drug_DNA_Solutions/openMP $ time -p ./dd_omp 7 120 3
max_ligand=7 nligands=120 nthreads=3
OMP defined
maximal score is 5, achieved by ligands
ieehkc acehch
real 90.26
user 135.15
sys 0.00
pi@raspberrypi:~/A5/Drug_DNA_Solutions/openMP $ time -p ./dd_omp 7 120 2
max_ligand=7 nligands=120 nthreads=2
OMP defined
maximal score is 5, achieved by ligands
acehch ieehkc
real 104.97
user 131.75
sys 0.01
pi@raspberrypi:~/A5/Drug_DNA_Solutions/openMP $
```

dd_omp time for 7 max_ligand.

(C,7)



```
pi@raspberrypi:~/A5/Drug_DNA_Solutions $ ./dd_threads
max_ligand=7 nligands=120 nthreads=4
maximal score is 5, achieved by ligands
ieehkc acehch
pi@raspberrypi:~/A5/Drug_DNA_Solutions/threads $ time -p ./dd_threads 7 120 4
max_ligand=7 nligands=120 nthreads=4
maximal score is 5, achieved by ligands
ieehkc acehch
real 41.74
user 142.51
sys 0.01
pi@raspberrypi:~/A5/Drug_DNA_Solutions/threads $ time -p ./dd_threads 7 120 3
max_ligand=7 nligands=120 nthreads=3
maximal score is 5, achieved by ligands
ieehkc acehch
real 54.69
user 143.31
sys 0.03
pi@raspberrypi:~/A5/Drug_DNA_Solutions/threads $ time -p ./dd_threads 7 120 2
max_ligand=7 nligands=120 nthreads=2
maximal score is 5, achieved by ligands
ieehkc acehch
real 77.37
user 141.53
sys 0.00
pi@raspberrypi:~/A5/Drug_DNA_Solutions/threads $ cd ..
pi@raspberrypi:~/A5/Drug_DNA_Solutions $ cd openMP/
pi@raspberrypi:~/A5/Drug_DNA_Solutions/openMP $ time -p ./dd_omp 7 120 4
OMP defined
maximal score is 5, achieved by ligands
ieehkc acehch
real 75.19
user 137.47
sys 0.01
pi@raspberrypi:~/A5/Drug_DNA_Solutions/openMP $ time -p ./dd_omp 7 120 3
max_ligand=7 nligands=120 nthreads=3
OMP defined
maximal score is 5, achieved by ligands
ieehkc acehch
real 90.26
user 135.15
sys 0.00
pi@raspberrypi:~/A5/Drug_DNA_Solutions/openMP $ time -p ./dd_omp 7 120 2
max_ligand=7 nligands=120 nthreads=2
OMP defined
```

dd_threads times for 7 max_ligand.