A0 Report

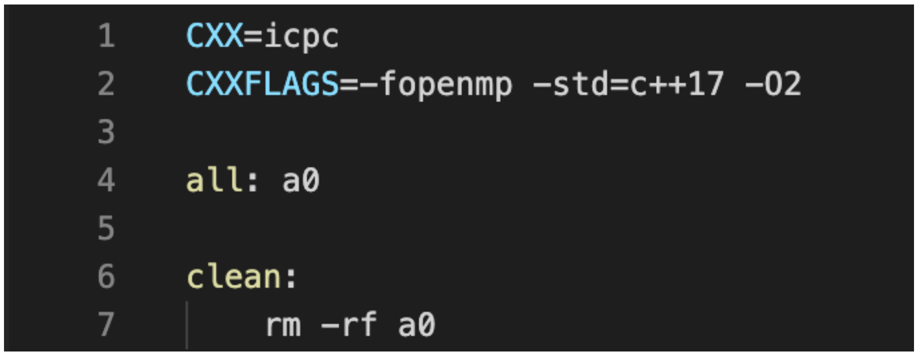
Name: Viswa Nihar Nukala
UB Name: viswanih
UB Number: 50414392

The code to be implemented is filter_2d which takes in 4 parameters, n: number of rows, m: number of columns, K: a vector of size 3x3 and A: a vector of size nxm.

I have implemented one extra function to determine the index of an element in the vector using the relative position of row and column. But since the function is inline, the function doesn't take much time to execute.

Coming to the parallel part of the program, as I have used 2 nested for loops, I have decided to go with taskloops. As there are 2 for loops, I have used 2 taskloops. One of the taskloop is to create tasks for iterating rows and the other taskloop is to iterate the columns of each row. The first task has (A, K and B) as shared because these vectors need to be accessed by the tasks parallelly. The second task loop has variable as firstprivate as the (i) value needs to retain and incremented properly between tasks.

The following is a snapshot of my Makefile:



```
CXX=icpc
CXXFLAGS=-fopenmp -std=c++17 -O2

all: a0

clean:
	rm -rf a0
```

I have profiled with g++ and intel compilers and I have chosen intel compiler to compile my program as the optimization and compile time is slightly better over g++.

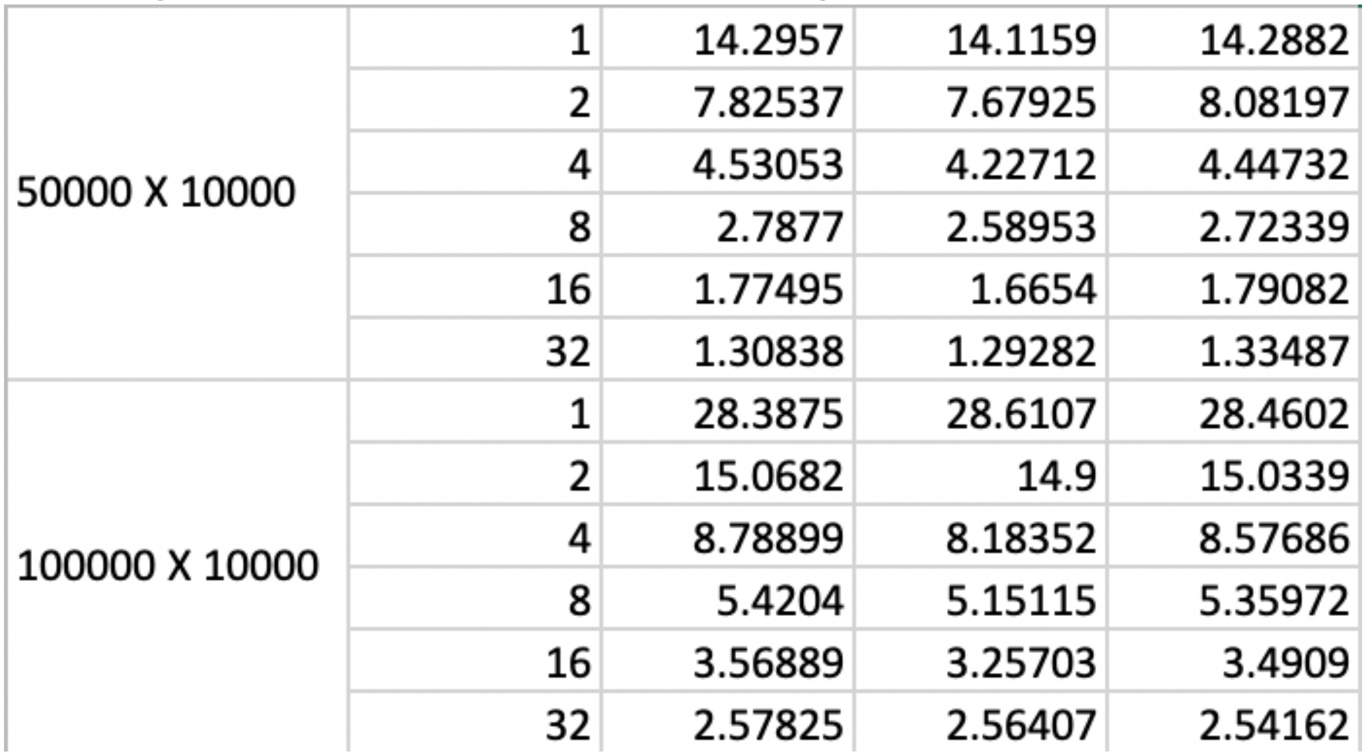
The following is a snapshot of my slurm.sh:

```
#!/bin/bash

####### PLANEX SPECIFIC - DO NOT EDIT THIS SECTION
#SBATCH --clusters=faculty
#SBATCH --partition=planex
#SBATCH --qos=planex
#SBATCH --account=cse570f21
#SBATCH --exclusive
#SBATCH --mem=64000
#SBATCH --output=%j.stdout
#SBATCH --error=%j.stderr

####### CUSTOMIZE THIS SECTION FOR YOUR JOB
#SBATCH --job-name="changeme"
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=32
#SBATCH --time=00:35:00

# your app invocation should follow
# ...

module load intel/20.2

make clean

make

OMP_NUM_THREADS=1 ./a0 50000 10000
OMP_NUM_THREADS=2 ./a0 50000 10000
OMP_NUM_THREADS=4 ./a0 50000 10000
OMP_NUM_THREADS=8 ./a0 50000 10000
OMP_NUM_THREADS=16 ./a0 50000 10000
OMP_NUM_THREADS=32 ./a0 50000 10000

echo

OMP_NUM_THREADS=1 ./a0 100000 10000
OMP_NUM_THREADS=2 ./a0 100000 10000
OMP_NUM_THREADS=4 ./a0 100000 10000
OMP_NUM_THREADS=8 ./a0 100000 10000
OMP_NUM_THREADS=16 ./a0 100000 10000
OMP_NUM_THREADS=32 ./a0 100000 10000

make clean
```

I have given ntasks-per-node as 32 as there is one of the outputs which needs 32 threads. I have taken 2 different sizes of matrices, 50000 X 10000 and 100000 x 10000. The first line after SBATCH flags loads the intel module. Then I call make clean to remove any binaries. Then make is called to compile the code. The next 12 lines are to run the output with different number of processors with different data size.

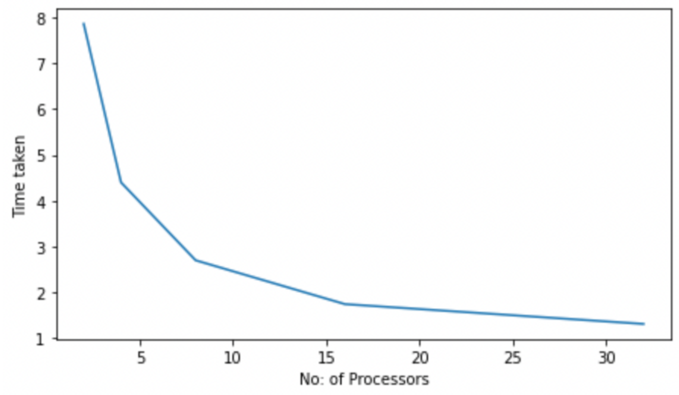
Why Strong Scaling:
I have chosen strong scaling as this gives a clearer idea of how the processors are impacting the speed on how the program is improving for a fixed data size.

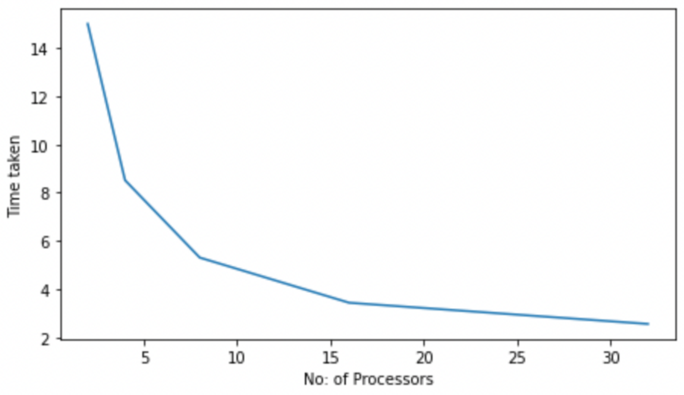
Following are the times I have obtained running different matrix sizes:



50000 X 10000	1	14.2957	14.1159	14.2882
	2	7.82537	7.67925	8.08197
	4	4.53053	4.22712	4.44732
	8	2.7877	2.58953	2.72339
	16	1.77495	1.6654	1.79082
	32	1.30838	1.29282	1.33487
100000 X 10000	1	28.3875	28.6107	28.4602
	2	15.0682	14.9	15.0339
	4	8.78899	8.18352	8.57686
	8	5.4204	5.15115	5.35972
	16	3.56889	3.25703	3.4909
	32	2.57825	2.56407	2.54162

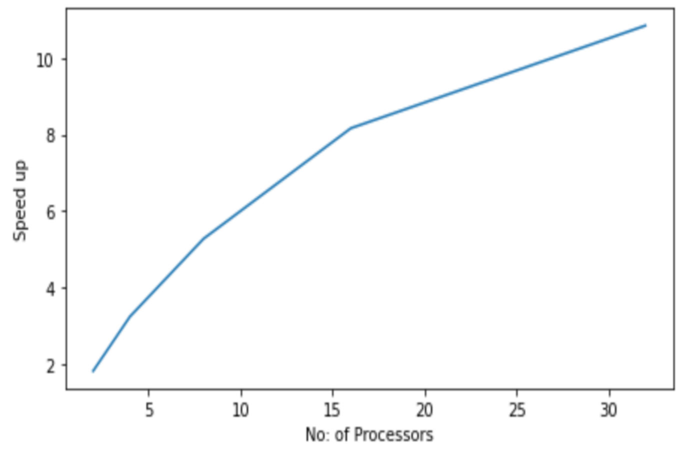
The following is the time graph obtained by running the above slurm.sh:

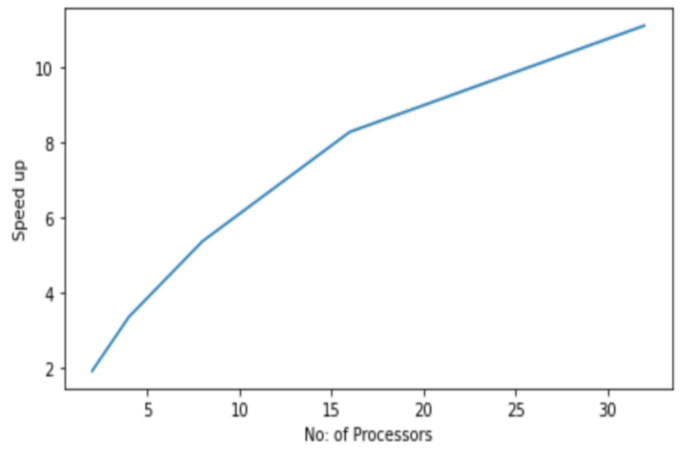




We can infer that as the number of processors increase, the time taken for the program to complete decreases but after a number of processors it is tending to become linear.

The following is the speed up graph obtained by running the above slurm.sh





The graph on the left is the speed up for matrix of size 50000 X 10000 and the graph on the right is for 100000 x 10000.
We can infer from the graph that as number of processors increase, the speed up decreases and after some number of processors the speed up remains constant.