CIS 6930 Project 3 Report

Tae Seung Kang

In the project report you need to describe the implementation and performance results of the programs in both tasks.

A. Task 1 (OpenMP)

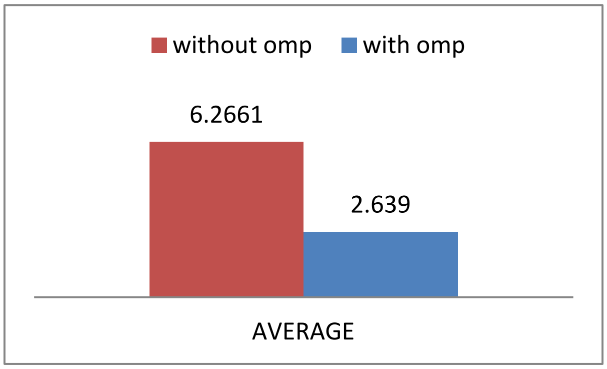
1. Parameters

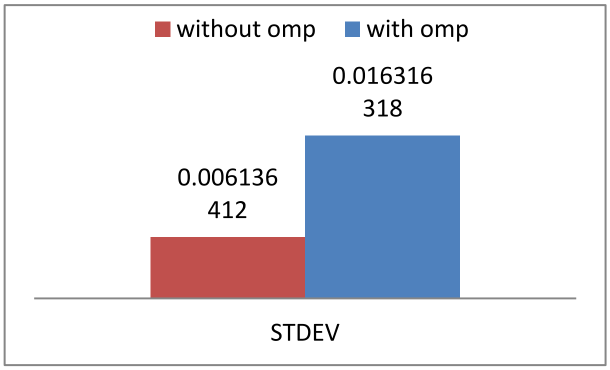
The following is the illustration of data processing pipelining with table dependencies.

Damping factor = 0.85

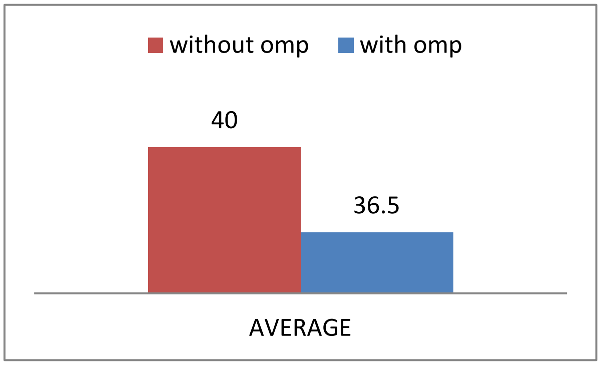
10 runs are executed and averaged.

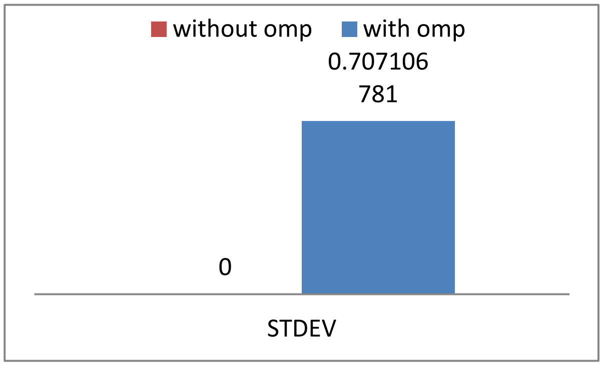
Running time comparison: average and standard deviation





comparison of number of iterations: average and standard deviation





With omp directives:

	running time (sec)	iterations
run1	2.647	37
run2	2.649	37
run3	2.65	37
run4	2.642	37
run5	2.67	37
run6	2.614	35
run7	2.623	36
run8	2.639	37
run9	2.625	36
run10	2.631	36
average	2.639	36.5
stdev	0.016316	0.707107

Without omp directives:

	running time (sec)	iterations
run1	6.265	40
run2	6.261	40
run3	6.274	40
run4	6.265	40
run5	6.267	40
run6	6.258	40
run7	6.264	40

run8	6.259	40
run9	6.273	40
run10	6.275	40
average	6.2661	40
stdev	0.006136	0

2. What kind of analytics do you apply on the dataset? What are the Hive queries?

We analyze the data based on the following questions. For Hive queries, refer to the attachments (/src/enron folder). The HiveQLs for base tables are in /src/enron/ken.sql and /src/enron/steve.sql.

6. What difficulties you faced and what you learned from this project?

- The data given to us didn't fit into Hive format as the enron data file contains ^M character which is considered as tab. Hence, we had to remove the character before loading to Hive table.

B. Task 2 (MPI)

1. How to compile and run

Directory: Task2/src

Program files: reducer.cpp, makefile, 100000_key-value_pairs.csv

I use Linux 'make' utility to compile and run. To compile and run the program, go to the directory Task2/src.

- To compile: make compile

- To run: make run

This will run the following command to execute the binary code reducer.o with a single process.

```
time ./reducer.o
```

- You can edit makefile to change the input graph file. The default input file is set to 100000_key-value_pairs.csv under the current working directory.

- mpirun [--nolocal] [-np 4] [-machinefile machinefile] out

  where -np is the number of processors

implementation methods, experimental settings, outputs and corresponding

conclusions

implementation methods

- c++ standard library (stl) is used.
- At first, unordered_map is used for hash tables. Later, I realized that the

  order of the output is wrong. So, I used map under stl.
- too hard to send vector type: when sending key-value pair list to each processor, I had to convert

  the vector of key-value pairs to 2-dimensional int array.
- maximum number of message (int array): 16375*2 = 32760 (integers)

experimental settings

- IBM Blade HS22 8cores with 16threads (16 logical processors) with 24GB ram
- output order messy without synchronization: needed barrier
- compare single processor and multiprocessors

outputs and corresponding conclusions

How to compile and run

- compile: make compile
- run: make run
- you can edit makefile to change the input file. The default input file is 100000_key-value_pairs.csv under the current working directory.
- mpicc -foepnmp -std=c99 -o out file
- mpirun [--nolocal] [-np 4] [-machinefile machinefile] out

  where -np is the number of processors

implementation methods, experimental settings, outputs and corresponding conclusions

implementation methods

- c++ standard library (stl) is used.
- At first, unordered_map is used for hash tables. Later, I realized that the order of the output is wrong. So, I used map under stl.
- too hard to send vector type: when sending key-value pair list to each processor, I had to convert the vector of key-value pairs to 2-dimensional int array.
- maximum number of message (int array): 16375*2 = 32760 (integers)

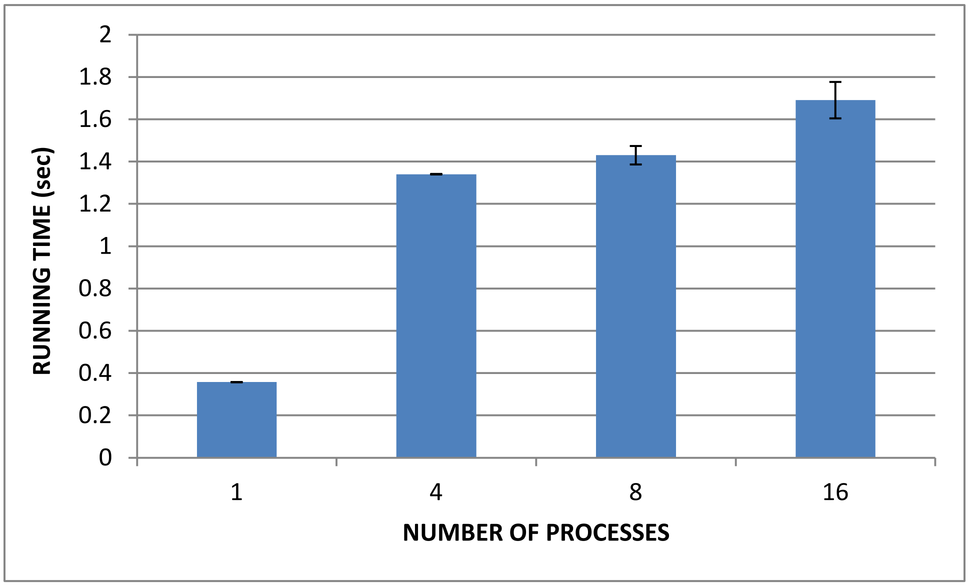
experimental settings

- IBM Blade HS22 8cores with 16threads (16 logical processors) with 24GB ram
- output order messy without synchronization: needed barrier
- compare single processor and multiprocessors

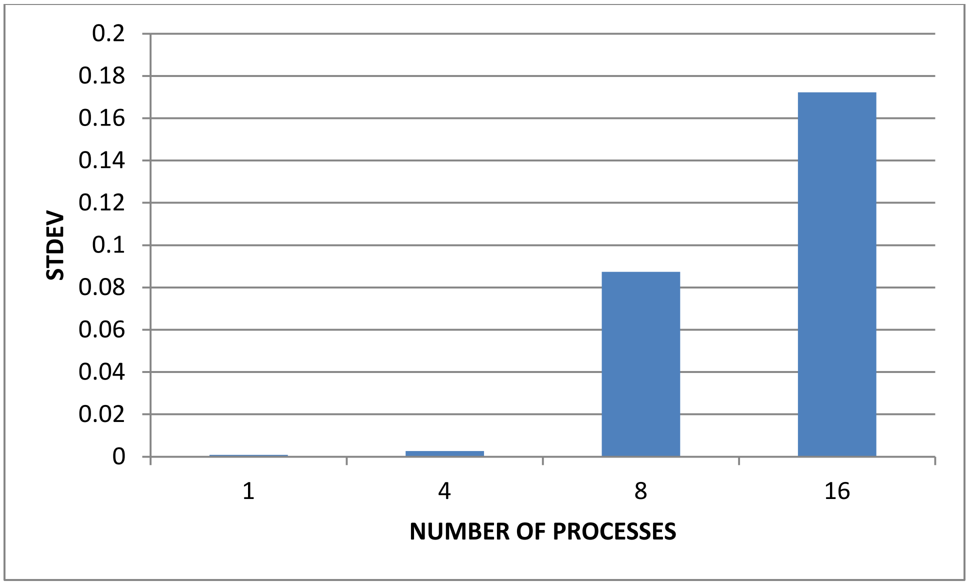
outputs and corresponding conclusions

- running time vs. number of processes. 5 runs are executed and averaged for each number of processes.

Running time according to the number of processes with error bars:



Standard deviations of running times:



Data:

Number of processes	1	4	8	16
run1	0.358	1.34	1.36	1.582
run2	0.358	1.336	1.531	1.584
run3	0.356	1.342	1.382	1.691
run4	0.357	1.343	1.519	1.988
run5	0.357	1.341	1.358	1.607
average	0.3572	1.3404	1.43	1.6904
stdev	0.000837	0.002702	0.087336	0.172187

- output file: Output_Task2.txt

- output format: key and sum of values are separated by a tab

- single processor is faster than multiprocessors. This is because the dataset size is too small to take advantages of parallel processing. Rather, communication overhead (MPI_Sendrecv) would've dominated the running time. Also, synchronization control like MPI_Barrier should be the bottleneck.

- What are you looking for regarding performance results in the project report?

Mainly refer to the output results, and based on this you need to give some analysis of advantages using corresponding parallel computing method (not need too much). Additionally for the implementation, you need to give more details.

6. What difficulties you faced and what you learned from this project?

- join on large dataset like movie_ratings (100M x 100M records) is challenging because of runtime and intermediate disk space. We needed to come up with a smart way to deal with the issue. When we ran the query on the original datasets, we got the checksum error or were unable to rename the output file due to lack of disk space.