HW4: PageRank & Smith-Waterman

MGP 2022
Due - 4/18 Mon 11:59pm

0 Introduction

In this project, you will optimize the application of choice - PageRank, and Smith-Waterman Algorithm with Parallel Programming. It's up to you to choose which to optimize. Also, note that you are free to do both for maximum credit! :)

The specification has the following contents.

- 1 Environment Setup: About our environment
- 2 PageRank: Brief description of PageRank
- 3 Smith-Waterman: Brief description of Smith-Waterman
- 4 Criteria: What you have to do
- 5 Grading: How we grade the results
- 6 Server Info: Extra informations about server
- 7 Reference

!!!Caution: Plagiarism!!!

You need to be careful to comply with plagiarism regulations. There is an ambiguity between discussion, idea, and solution. Therefore, be careful not to take away the opportunity of your colleague.

- No web code-searching
- No code sharing
- **No idea sharing**
- Ok discussion about the project

1 Environment Setup

The same manner as in previous homeworks.

You don't need to set up the environment. We prepared everything you need to do an experiment about multi-threading. Check GNU GCC/G++ 9.2.1, GNU make, and HTCondor documentation for more information.

2. PageRank

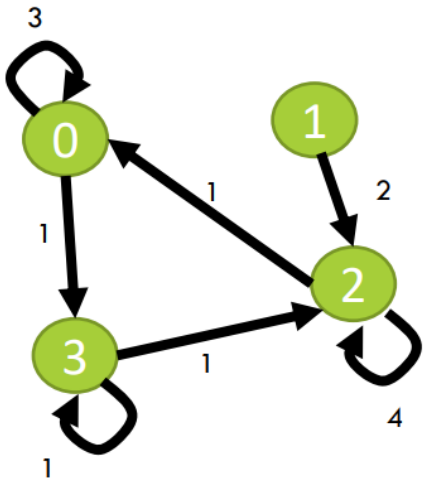
2-1 PageRank Algorithm

PageRank is an algorithm to rank vertices in a graph. Each of nodes have their own pagerank, and it calculates according to the following expression.

PR(A) = (1-d)/N + d (PR(T1)/DEG(T1) + … + PR(Tn)/DEG(Tn))

Let's say A is the node that we want to calculate page rank for. N is the total number of nodes. T_x denotes vertices that have incoming edges to A. From each T_x, A receives score: ***PR(T_x) / DEG(T_x)*** which is their scores divided by their out-degrees(the number of out-edges from T_x). d is damping factor. In PageRank paper, d is set as 0.85. By calculating above equation, we obtain page rank for vertex A. We run this algorithm for all vertices in each iteration, and after a sufficient number of iterations the values will converge.

In the provided reference, the data is given in a CSR form, which is one of the graph formats that can reduce a lot of space. It consists of value, row index, column index. Row index is accumulation of the number of edges. For instance, The graph is given like that:



You can transform csr for in-edge like this:

row index : 0 2 2 5 7
data : 3 1 2 4 1 1
col index : 0 2 1 2 3 0 3

Or, you can transform for out-edge:

row index : 0 2 3 5 7
data : 3 1 2 1 4 1 1
col index : 0 3 2 0 2 2 3

if you don't like it, you can transform it into whatever you want.
the time for transformation will not be taken into account (preprocessing)

2-2 Structure of template code

```
/HW4/PageRank$ tree .
.
├── bin                    // make file generate run file to here
├── Makefile               // Compile, Clean, Format, Submit
├── pr_facebook.cmd        // Command for condor
├── pr_livejournal.cmd     // Command for condor
├── result                 // Result from condor
└── src                    // Source Code
    ├── benchmark.h
    ├── builder.h
    ├── command_line.h
    ├── graph.h
    ├── platform_atomics.h
    ├── print_util.h
    ├── reader.h
    ├── timer.h
    └── pr.cc

3 directories, 13 files
```

2-3 Code Explanation

There are lots of defined classes and functions in source code but you don't have to understand most of them. So, we're going to briefly explain some of them that you must understand.

2-3-1 graph.h

graph.h has ' CSRGraph' class. This class has graph data in CSR and some graph processing functions

graph.h

variables:
- index_t* in_vertex_table_;
  => rowindex(source) is stored for in-edges.
- DestID_* in_edge_table_;
  => colindex(destination) is stored for out_edges

functions:
- int64_t num_nodes();
  => return the number of vertices
- int64_t num_edges();
  => return the number of edges
- out_degree(NodeID_ v);
  => return out degree of node v

2-3-2 pvector.h

pvector.h has 'pvector' class. This class is an improved class of std::vector. Methods are almost equal to vector class.

pvector.h

How to Initialize:
- pvector<T>();
  => make empty pvector instance
- pvector<T>(n);
  => make pvector has length n. The initial values are 0.
- pvector<T>(n, val);
  => make pvector has length n and initialized to val.

functions:
- void fill(T init_val);
  => initialize/change all of values to init_val
- size_t capacity();
  => return remain space

2-3-3 benchmark.h

benchmark.h has type definition.

benchmark.h

type definition:
- typedef int32_t NodeID;
  => NodeID is an alias of int32_t type.
- typedef CSRGrap<NodeID> Graph;
  => Graph is an alias of CSRGraph<int32_t> class.

If you need more information that is excluded from the specification, read the source code. We strongly recommend you to read the code in detail.

2-4 Run Program/Submission

To run the program, you must complete 'pr.cc'. You. **In case you wish to make modifications outside of pr.cc for your implementation but if you are unsure, you can ask the TAs.** You just need to fill in the 'PageRank()' function in 'pr.cc'. (You can add other files if you want) 'pr.cc' has 2 sections briefly explaining what to do. The sections are as follows: Preprocessing, PageRank run.

Preprocessing:
- If you need any preprocessing for pagerank, you can fill this section. (ex. change data stg functions/frames that are specified "prohibited", even in 'pr.cc'. For instance, you cannot modify parameters of 'PageRank()'.ruction from csr to csc, adj matrix, etc..)
- **Preprocessing time won't count.**
- If not, you can remain this empty.
- **Only preprocessing operations are allowed!**
- Ask TAs if you are unsure what can go here.

PageRank run:
- Your main code will be filled in this section.
- You should do pagerank here and fill the result to 'ScoreT results[]' array that is predefined in the function.
- This section's running time is fully added to total running time.

If you need any other functions, You can write your own functions in 'pr.cc'. Moreover, if you want your own header files, you can write it and include it in 'pr.cc'. However, you cannot modify existing functions/frames that are specified "prohibited", even in 'pr.cc'. For instance, you cannot modify parameters of '**PageRank()**'.

```
vector<pair<ScoreT, NodeID>> PageRank(const Graph &g, int num_iterations, int
mode) {
  /* Editing is Prohibited*/
  Timer alloc_timer;
```

If there are any signs of modification in the "prohibited" sections, we give you 0 pts. If you want other structures, you can use the 'Preprocessing section'.

After completion, you can submit the code using condor the same as before. We give you two condor commands, 'pr_facebook.cmd' and 'pr_livejournal.cmd'. Facebook and Livejournal are graph data for grading pagerank. You can submit your program to condor using **'make remote_facebook'** or **'make remote_livejournal'**. Otherwise, you can local run using this command: './bin/pr -f graph_file.el -c graph_answer.txt -k 10'
**Note that Graph data and answer exist in '/nfs/home/mgp2022_data/pagerank' folder!*

```
/HW4/PageRank$ ./bin/pr -f facebook.el -c facebook_answer.txt -k 10

Read Time:           0.34685
EL size : 88234
alloc edgelist 88234
alloc edgelist 88234
Build Time:          0.15106
directed constructor
preprocessing Time : 0.00000
trial Time:          0.00389
Printing Top5 Ranks
1911:0.00452479
3434:0.00450683
2655:0.00435287
1902:0.00431468
1888:0.00330874
PASS!! your total pass: 5
preprocessing Time : 0.00000
trial Time:          0.00387
Printing Top5 Ranks
1911:0.00452479
3434:0.00450683
2655:0.00435287
1011:0.00431468
1213:0.00330874
PASS!! your total pass: 5
….

PASS!! your total pass: 5
PageRank End. Your Pass Score: 10, Mininum Runtime: $@&%#
```

*This is just an example and the numbers here are arbitrary.

PageRank function will run 10 times. After 10 times running, the program will print your total correct count compared to answer and minimum runtime. Minimum(=best) runtime is your score.
For helping other experiment (e.g. other graph not given, pagerank iteration..) if you need. We explain some options. These are defined in 'command_line.h'.

```
Run program : ./bin/pr -f graph_file.el (-opt opt_param | -opt )*
```

option

- **f : Read graph file for pagerank. opt_param is graph file path. graph file must be '.el' format.**
- **c : Read the answer file of pagerank. opt_param is answer file path. If you run other graphs neither facebook or livejournal, you can skip it.**
- **k : Set the number of pagerank iterations. opt_param is new iteration value. We grade program to set the value 10.**
- **n : Set the number of trials. opt_param is new trial value.**
- **h : Print help message. It do not require opt_param.**

Graph data and answer files exist in '/nfs/home/mgp2022_data/pagerank' folder. If you want other graphs, you can download in here, but you must convert data to '.el' format. '.el' format is just a series of edge indices. If you need details, read 'facebook.el'.

3 Smith-Waterman

We will use the affine gap model of the Smith-Waterman model [1] (Maybe an easier description in [2]).

3-1-0 Structure of template code

```
/HW4/Smith-Waterman$ tree .
.
├── build
│   ├── main.o
│   ├── similarity_algorithm_sequential.o
│   └── smith_waterman_sequential.o
├── HW4_base.cmd
├── LICENSE
├── Makefile
├── README.md
└── src
    ├── backtrace.h
    ├── back_up_struct.h
    ├── similarity_algorithm_parallel.cc
    ├── similarity_algorithm_parallel.h
    ├── smith_waterman_parallel.cc
    ├── smith_waterman_parallel.h
    └── utils.h

2 directories, 14 files
```

Your job is to parallelize **Run()** defined in similarity_algorithm_parallel.cc, which calls **FillMatrices()** and **BackwardMoving()** defined in smith_waterman_parallel.cc. You can change other parts of the files as long as it has the same functionality, but make sure

you don't do any calculations outside of Run(), because that is the only function that gets timed. For your reference, here's a snippet of the main()

```
   SimilarityAlgorithmParallel *parallel= new SmithWatermanParallel(s1Len, s2Len, s1, s2, 5, 2);

   auto start = std::chrono::steady_clock::now();
   parallel->Run();
   auto end = std::chrono::steady_clock::now();
   diff = end-start;
   std::cout<<"parallel took "<<diff.count()<<" sec"<<std::endl;
   parallel->PrintResults(outprefix+std::string("_prl"));
```

3-1-1 Compile and Test

```
/HW4/Smith-Waterman$ make exec
g++ -O3 -g build/main.o build/similarity_algorithm_sequential.o
build/smith_waterman_sequential.o src/similarity_algorithm_parallel.cc
src/smith_waterman_parallel.cc -o sw -fopenmp -march=znver2

/HW4/Smith-Waterman$ make run
./sw 5 5
s1l 5 s1l 5
S1 5 S2 5

== sequential start ==
Matrix Filled: maxY 3 maxX 3 maxVal 10
Backward Moving: Y 1 X 1 Val 0
== sequential took 1.3004e-05 sec ==

== parallel start ==
Matrix Filled: maxY 3 maxX 3 maxVal 10
Backward Moving: Y 1 X 1 Val 0
== parallel took  $@&%#sec ==

/HW4/Smith-Waterman$ make remote
s1l 10000 s1l 10000
S1 10000 S2 10000

== sequential start ==
Matrix Filled: maxY 9943 maxX 9995 maxVal 8770
Backward Moving: Y 0 X 12 Val 0
== sequential took 1.81211 sec ==

== parallel start ==
Matrix Filled: maxY 9943 maxX 9995 maxVal 8770
```

```
Backward Moving: Y 0 X 12 Val 0
== parallel took $@&%# sec ==
```

You can compile the program with **make exec**. Also, you can test this program locally with **make run**. This 'Smith-Waterman' program gets 2 arguments. Both are the length of input strings. In the local test case, we use a size of 5 for both reference and target strings. In the result message, **sequential** is a reference result. Initially, **parallel** is almost the same as **sequential** and you should modify files to make **parallel** be executed parallely. After all, the **parallelized implementation must find the same destination(destX, destY) with the sequential version**, even if there are multiple possible destinations. Because **sequential** and **parallel** get the same input and execute the same algorithm, outputs should also be the same.

3-1-2 Implement your optimized solution

There is a **smith_waterman_parallel.cc** file in the **src** directory. A sequential version of Smith-Waterman algorithm is already implemented in the file. You need to modify **FillMatrices()** and **BackwardMoving()** functions to implement a parallel version of Smith-Waterman algorithm. But, if you want, you also can modify another part of **smith_waterman_parallel.cc** and **smith_waterman_parallel.h** files.

4 Criteria

4-1 Requirements

4-1-0 General

- It has to be parallel (no serial implementation)
- No open source, or parallel library - you can only use STL.
  You can use boost::barrier, but if you want to use anything else, contact us before doing so.
- You can use pthread, std::thread, openMP, or MPI as your parallelization framework. If you want to use something else (e.g., Apache Spark), contact us first.
- Use of SIMD is fine: AVX, SSE, etc
- Do not override driver.o

4-1-1 PageRank

- Problem graph: Livejournal (4,847,571 nodes, 68,993,773 edges)
- Performance requirements: **1.4 sec (Livejournal, 10 iterations)**
- Correctness: Should work correctly for any graphs smaller than Livejournal. To prevent expediency, we test correctness using some graphs that are not given.
- Do not modify prohibited region in 'pr.cc'

4-1-2 Smith-Waterman

- Problem size: 10000x10000
- Performance requirements: **0.5 sec**
- Correctness: Should work correctly for any string pairs of input between size of 2x2 ~ 10000x10000.

4-2 Measurements

We are measuring performance from a real machine, and it involves a certain amount of luck. To reduce the luck effect as much as possible, we do the following:

- Turn off DVFS of the machine
- Warm up the server for a few minutes before measuring. This means that the **program will run slower than your own tests** for all of you during our grading.
- We will run your program at least five times and **use the minimum exec.time**.

4-3 Report

Your report should include

- What techniques you have implemented
- How to run your code
- How each technique affected your performance (+ comparison)
- Why you think your technique, or your combination of techniques produced the best result
- max 4 pages (firm)
- **PDF only**
- If 4 pages is too short to contain everything you want to say, use a double-column format (e.g., https://ieeecs-media.computer.org/assets/zip/Trans_final_submission.zip , https://ieeecs-media.computer.org/assets/zip/ieeetran-final_sub.zip)

5 Grading

Correct parallel implementation (30) - finish faster than the reference code, produce correct result with a *PARALLEL* implementation
Report (20) - Refer to 4-3.Report
Performance bar (30) - Based on last year's students and many criteria, we have set the bar for each project, which is not too challenging. If your execution time is x seconds where x > bar, here's your score:

$$score = (30*bar/x)$$

We might choose to increase the time bar in case of too few students working on a specific application, but we will not lower it (we will not ask you to make a faster program in the middle)

Ranking (20) - The ranking is decided by the execution time of interest defined below for each application:

$$rank\_score = 20 * (80 - rank)/79$$

This means that if students are equally split into two, the lowest score will be 10. If you're the only one working on an application, you will automatically get 20 points.

- You need to pass the performance bar in order to get the ranking point
- We might choose to tie some rankings if the difference is small.

We will occasionally announce the current scoreboard.

Extra point (10) - You can choose to implement both applications, where you will get some extra points. It only counts when you pass the performance bar. Your extra points will be:

$$extra = [lower\_of\_the\_two] * 0.10$$

making it a maximum of 10 points.

6 Server Info

tell us if you need anything else!

```
$ cat /proc/cpuinfo
processor       : 0
vendor_id       : AuthenticAMD
cpu family      : 23
model           : 113
model name      : AMD Ryzen 7 3700X 8-Core Processor
stepping        : 0
microcode       : 0x8701012
cpu MHz         : 2196.672
cache size      : 512 KB
physical id     : 0
siblings        : 16
core id         : 0
cpu cores       : 8
apicid          : 0
initial apicid  : 0
fpu             : yes
fpu_exception   : yes
cpuid level     : 16
wp              : yes
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat
pse36 clflush mmx fxsr sse sse2 ht syscall nx mmxext fxsr_opt pdpe1gb rdtscp lm
```

constant_tsc rep_good nopl nonstop_tsc cpuid extd_apicid aperfmperf pni pclmulqdq monitor ssse3 fma cx16 sse4_1 sse4_2 movbe popcnt aes xsave avx f16c rdrand lahf_lm cmp_legacy svm extapic cr8_legacy abm sse4a misalignsse 3dnowprefetch osvw ibs skinit wdt tce topoext perfctr_core perfctr_nb bpext perfctr_llc mwaitx cpb cat_l3 cdp_l3 hw_pstate sme ssbd mba sev ibpb stibp vmmcall fsgsbase bmi1 avx2 smep bmi2 cqm rdt_a rdseed adx smap clflushopt clwb sha_ni xsaveopt xsavec xgetbv1 xsaves cqm_llc cqm_occup_llc cqm_mbm_total cqm_mbm_local clzero irperf xsaveerptr wbnoinvd arat npt lbrv svm_lock nrip_save tsc_scale vmcb_clean flushbyasid decodeassists pausefilter pfthreshold avic v_vmsave_vmload vgif umip rdpid overflow_recov succor smca
bugs : sysret_ss_attrs spectre_v1 spectre_v2 spec_store_bypass
bogomips : 7186.75
TLB size : 3072 4K pages
clflush size : 64
cache_alignment : 64
address sizes : 43 bits physical, 48 bits virtual
power management: ts ttp tm hwpstate cpb eff_freq_ro [13] [14]

… (continues until processor 16)

7 Reference

- GNU GCC/G++ 9.2.1
- GNU make
- HTCondor
- SNAP Dataset
- PageRank
- [1] Osamu Gotoh, An improved algorithm for matching biological sequences,
- Journal of Molecular Biology, 1982
- [2] Peiheng Zhang, Implementation of the Smith-Waterman algorithm on a reconfigurable supercomputing platform, HPRCTA, 2007
- Our Lecture Slides
- Ask Professor, TA(Of course not about the code)