Assignment I:

Multi-threaded implementation of PageRank

PageRank

- PageRank measures the importance of a node ***i*** in a graph ***G*** as the ***weighted sum of the importance of its neighbours***.

- Let ***v*** be a vector storing the importance of each node
  - ***v[i]*** = importance of the ***i***-th node, and elements of ***v sum up to 1***
  - The contribution of each neighbor ***j*** is normalized by its out-degree ***o(j)***

$$v^{t+1}[i] = \sum_{j \to i} \frac{v^t[j]}{o(j)}$$

PageRank

$$v^{t+1}[i] = \sum_{j \to i} \frac{v^t[j]}{o(j)}$$

- Note that the above update rule can be rewritten as a matrix-vector multiplication:

$$v^{t+1} = Mv^t \qquad \text{with} \qquad M[i,j] = \begin{cases} \frac{1}{o(j)} & \text{if } j \to i \\ 0 & \text{otherwise} \end{cases}$$

- The above update rule, defines an ***iterative process***:
  - **we start from a random vector *v^0***, (v^0 sums up to 1, it is a probability distribution)
  - **the update rule is applied for several iterations** (<=50) **until convergence**
  - a.k.a. random surfer model

PageRank

$$v^{t+1} = Mv^t \qquad \text{with} \qquad M[i,j] = \begin{cases} \frac{1}{o(j)} & \text{if } j \to i \\ 0 & \text{otherwise} \end{cases}$$

- Does it converge?
- Only if the original graph ***G*** is ***irreducible*** (all states are reachable from any other state) and ***aperiodic*** (no "cycles" of fixed length)
- The Web Graph does not satisfy this criteria (it has ***dead ends*** and ***cycles***)
- ***Solution*** (*leading to the so called* ***Google Matrix***):
  Teleportation *and* ***dead ends removal***

$$v^{t+1} = \beta M v^t + (1-\beta)1/n \qquad \text{with} \qquad M[i,j] = \begin{cases} \frac{1}{o(j)} & \text{if } j \to i \\ 1/n & \text{if } o(j) = 0 \\ 0 & \text{otherwise} \end{cases}$$

  - with probability β we follow M, ***with probability (1-β) we jump to a random node***
  - ***dead end nodes link to all nodes of the graph***

To Be Delivered

- Sequential implementation (C/C++)
- Parallel implementation
  - Multi-threaded std::threads or OpenMP
- Report discussing performance figures of the proposed parallel implementation
  - varying graphs (small, large, sparse, dense)
  - varying number of threads
  - pdf file, max 2 pages

در ریپورت باید آنالیز کنید . لازم نیست که page rank را توضیح دهید.

Deadline and Evaluation

- **Final Score** = 70% written exam +
  10% 1st assignment + 10% 2nd assignment + 10% 3rd assignment
- Delivery before March 29:
  - speed bonus +3/30
- Or, delivery at written exam
  - no speed bonus
- Evaluation is based on:
  - report quality
  - code quality
  - depth of analysis

یک فایل زیپ: سورس کد (با گوشیتون عکس نگیرید)و پی دی اف

- Oral presentation/discussion, first weeks of April…

Implementation

- Datasets:
  - https://snap.stanford.edu/data/index.html
  - You may generate synthetic data by implementing the preferential attachment algorithm https://en.wikipedia.org/wiki/Barab%C3%A1si%E2%80%93Albert_model

- Dataset Representation
  - you may represent *M* as a ***full square matrix***
  - ***sparse representation I***: for each ***row*** of *M*, store non-zero elements by using an ***array of node ids***, and an ***array of weights***
  - ***sparse representation II***: for each ***column*** of *M*, store non-zero elements by using an ***array of node ids***, and ***a single value for o(j)***
  - ***remove dead ends*** and process them separately

بهتر است که انواع مختلف
گراف را توضیح دهید

Implementation

- Parallelize by partitioning $\boldsymbol{v}^{t+1}$ into chunks (and corresponding rows of *M*)
    - each thread computes a chunk of $\boldsymbol{v}^{t+1}$
- Is the load balanced?
    - depends on the graph, can we measure this?
- Does the order of the rows impact on the load balancing?
    - do I need to assign to a thread consecutive rows of $\boldsymbol{v}^{t+1}$?
    - can I further split rows with a large number of neighbors?
- *[optional]* Parallelize by partitioning ***M*** *into chunks*
    - *by rows or columns? depends on the representation…*
    - *by both rows and columns?*

References

- Mining of Massive Datasets
  - Sections 5.1 and 5.2

- Short
  - http://www.plutospin.com/files/OpenMP_reference.pdf
  - https://computing.llnl.gov/tutorials/openMP/
- Long:
  - http://lib.mdp.ac.id/ebook/Karya%20Umum/Portable_Shared_Memory_Parallel_Programming.pdf
  - http://www.openmp.org/wp-content/uploads/OpenMP4.0.0.Examples.pdf

- Compilation
  - Use the –fopenmp flag when compiling !

Comments

Recommendations:

- Algorithm
  - provide a ***description*** of the ***algorithm*** you implemented and the ***data structures*** you used
    - in some reports, the algorithm is clear only after looking at the code
- Analysis
  - try to ***explain*** where you algorithm performs nicely and when it does not
  - possibly with experiments
    - e.g. “i think it is because of the cache”, then measure the cache performance

Recommendations:

- Implementation and source code
  - think about implementing a ***library*** that someone else may easily use
    - clear definition of the “core” functions, e.g. load_graph, compute_something
  - ***separate the “core”*** code from the ***experiments***
    - e.g., avoid testing different thread numbers inside the implementation of your solution
  - avoid hardcoding input files, rather use shell arguments

Recommendations:

- Experimental settings and results
  - ***clarify the number of cores***/processors of your test environment
  - ***consider using the cluster***, as speedup up to 4 cores does not make a lot of sense
  - it is ok to focus on the speedup, ***please report at least the running time of the sequential algorithm*** as the reader might like to know if we are talking about seconds or hours of computation
  - make sure you compile with -O3 and make this clear in the report

Recommendations:

- Baselines and careful implementations
  - scalability is the focus, but performance is important
  - make sure your implementation cannot be easily improved or it has some major drawbacks
    - e.g., you touch every cell of a large and sparse square matrix

Recommendations:

فقط بعضی از قسمت ها که مهم
تر هستند را optimize کنید
لازم نیست همه را optimize
و paralleize کنید

- Analysis
  - ***clarify what you are optimizing***
    - e.g., "i'm optimizing step X which takes 90% of the total computation", and support with experiments
  - Consider and evaluate different strategies and different data structures
- Experiments
  - ***clarify what you are measuring*** and motivate
    - e.g., "when computing the speedup, i'm not including the loading time because..."

Recommendations:

- Analysis
  - most of the times poor scalability is due to ***load imbalance***
  - ***measure the load*** (size of input data) or the running time ***of each task/thread***
  - evaluate strategies to ***reduce load imbalance***
    - e.g., different order of the tasks, different granularity, split larger tasks, merge smaller tasks

Recommendations:

- Report
    - think at the report as a book
    - ***be careful with everything***
    - make sure you use the right terminology when relevant and avoid ambiguities
        - e.g., if you use a std::vector do not say you are using a list