Finding the Strongly Connected Components of a directed graph in a shared memory multiprocessor

A strongly connected component (SCC) is a maximal subgraph of a directed graph in which there is a path from each vertex to any other vertex in the subgraph. In this homework, we will be implementing a parallel strongly connected components algorithm to identify the number of SCCs in large directed graphs and compare its performance and scalability to the state-of-the-art sequential algorithms by Tarjan and Kosaraju. These algorithms are based in depth first traversal of the graph nodes, so they are not a good choice to parallelize, but do understand how they work and why.

You may also use this as a reference implementation for the sequential algorithm.

The algorithm you are implementing is the SCC Coloring Algorithm:

function ColoringSCCAlgorithm(G)

while G ≠ ∅ do

initialize color(vid) = vid

while at least one vertex has changed colors do

for all v ∈ G do

for all u ∈ Nin(v) do

if colors(v) > colors(u) then

color(v) ← color(u)

for all unique c ∈ color do

SCC(cv) ← PRED(G(cv),c)

G ← (G \ SCC(cv))

return SCC

Where

- Nin(v) denotes the neighboring vertices that point to vertex v

- PRED(G,cv,c) denotes the predecessor nodes of node cv in graph G that are of color c that have a path to node cv

0. Implement the SCC in C/C++ (2 points)

Implement the SCC Coloring Algorithm, (Alg. 2 as described in pg 551 of the article BFS_and_ColoringBased_Parallel_Algorithms_for_Strongly_Connected_Components_and_Related_Problems.pdf

A graph will be represented by its adjacency matrix. Represent a graph with a memory-efficient data structure. We will be processing all the directed graphs from the SuiteSparse Matrix Collection repository that can fit the memory of our computers.

Use any available library to read these dataset matrix files, starting from the top.

Graph	n	m	ncc
Newman/celegansneural	297	2345	57
Pajek/foldoc	13356	120238	71
Tromble/language	399130	1216334	2456
LAW/eu-2005	862664	19235140	90768
SNAP/wiki-topcats	1791489	28511807	1
SNAP/sx-stackoverflow	2601977	36233450	953658
Gleich/wikipedia-20060925	2983494	37269096	975731
Gleich/wikipedia-20061104	3148440	39383235	1040035
Gleich/wikipedia-20070206	3566907	45030389	1203340
Gleich/wb-edu	9845725	57156537	4269022
LAW/indochina-2004	7414866	194109311	1749052

Graph	n	m	ncc
LAW/uk-2002	18520486	298113762	3887634
LAW/arabic-2005	22744080	639999458	4000414
LAW/uk-2005	39459925	936364282	5811041
SNAP/twitter7	41652230	1468365182	8044728

Use *trimming* to simplify the graph. The trimming procedure identifies and removes all trivial SCCs. All vertices that have an in-degree or out-degree of zero (excluding self-loops) form their own single-node SCC and are masked out from subsequent processing. Trimming may be performed recursively, as removing/masking out a vertex changes the effective degrees of its neighbors.

The common and time-consuming algorithmic component is the Breadth First Search (BFS) traversal of the graph from all nodes (following the edges forward initially and then backward to establish the predecessors). A way to express the BFS is to think or implement it as a sparse matrix-vector multiplication. However, instead of using scalar addition and multiplication, you substitute with your functions to copy the vector value and take the minimum.

The BFS will be successive matrix-vector "multiplications". As vertex values will converge, and fewer and fewer vertices will participate, you may want to restrict the computations where needed.

Test the correctness and performance of your implementation.

1. Parallelize your implementation on multicore CPUs (2+2+2 points)

Use OpenCilk (2pt), OpenMP (2pt) and PThreads (2pt) to parallelize your sequential code.

2. Test the correctness and performance of your implementation (2 points)

Design and display a couple of figures that demonstrate the performance of your implementation. Identify the different cases in terms of the number of threads and graphs of various sizes (in terms of the number of nodes, number of edges, and number of SCCs.

What to submit

- Submit a 4-page report in PDF format (any pages after the 4th one will not be taken into account). Make sure you describe your data structure and your

parallelization strategies and choices. Pay special attention to the presentation and analysis of the correctness and performance test results. Do not expect the reviewers to dig out the diamonds (or the coal) in your results. Be upfront in explaining the success or failure of your approach!

- Upload the source code on any cloud like GitHub, BitBucket, Dropbox, Google Drive, etc., and add a link to your report. Leave instructions with dependencies and a Makefile for the reviewers to run your code and replicate your results.

General advice

Start the assignment early and discuss your questions or observations in public at our course Zulip. Extra credit to the people that provide insightful comments in Zulip (questions as well as answers).

It is always better to work on a concept/model implementation in a dynamic language like python, Julia or MATLAB to make sure you have understood what needs to be done before you dive into the C/C++ quagmire. If you use Julia, you may also include your parallel code for extra credit.

Please use the Internet and utilize available codes and libraries but always cite your sources accurately.

By all means, discuss and help each other, but the collaborations must be limited to two persons who submit the code and report. If in doubt, please discuss any extended collaborations with others beyond the team of two that files the report.

https://suitesparse-collection-website.herokuapp.com/