





Graphlet transform of Sparse Graphs Using CUDA

Overview

Given a sparse graph with dense neighbor matrix A , the goal of the assignment is to find specific information for each Node of the graph. The information we need to find is:

	σ_1	1-path, at an end	$\hat{d}_1 = p_1$
	σ_2	2-path, at an end	$\hat{d}_2 = p_2$
	σ_3	bi-fork, at the root	$\hat{d}_3 = p_1 \odot (p_1 - 1)/2$
	σ_4	3-clique, at any node	$\hat{d}_4 = c_3$

Where:

- $p_1 = A * e$ (e is the unit vector)
- $p_2 = A * p_1 - 2 * c_3$
- $d_3 = p_1 \odot (p_1 - 1)/2 - c_3$ (The formula in the table is incorrect)
- $c_3 = A \odot A^2 - c_3$

Sequential Implementation

The graph is going to be stored in csc sparse format.

The calculation of all the above excluding c_3 is straightforward that requires one or two loops to go through the data.

The calculation of c_3 is the most troublesome.

Calculating C_3 :

- We know that matrix A is sparse, meaning most values are 0. So, in order to calculate c_3 we will not calculate the squared A first because it will require a lot more time. Instead, we will calculate only the values of A^2 that A doesn't have 0 on.

- First, we will go through all Nodes, we will call current node i .
- Second, we will check all the neighbors of i , say j .
- Now we have one row and one column of the matrix, so we need to calculate $c3$. We know that A only contains ones, that means A^2 will only contain ones as well. In the end A^2_{ij} is going to be the sum of common neighbors between i and j .

And that's the sequential program.

Parallelizing with CUDA

The first thing we need to do is turn all our functions into kernels.

Then we need to allocate memory on our device and copy all the data from the host to the device

Then all that is left is to parallelize the outer loop of every function. By parallelizing the outer loop, we know that there are not going to be any race conditions and each loop writes to only one object, and that is the current Node.

We set the `<<<grids,cores>>>` of each kernel to an appropriate number according to our device and we run.

Result

Running some sparse matrices from <https://suitesparse-collection-website.herokuapp.com/> I got the following results.

Auto DIMACS10	V = 448K , E = 6M	1,9s	0,7s
great-britain_osm DIMACS10	V = 7.7M, E = 16M	2,3s	1,6s
Delaunay_n22 DIMACS10	V = 4.1M, E = 25M	4,2s	2,3s
coPapersDBLP DIMACS10	V = 500K, E = 40M	18,9s	10,8s

I transformed the matrix to csc formatted by using a python script that I included in the repository. The script takes the matrix market formatted graph and creates two text files named indices and data which the `.c` and `.cu` programs read and produce the result. The above times take are (sequential-time|parallel-time)

As we can the parallel version takes half the time to run the same matrix as the sequential problem in the current graphs. I did some experiments with some larger graphs such as com-Orkut and the results were changing from better to same or even slightly worse. I suspected that my hardware is to blame as the GPU utilization while running the program was on 100%

Improvements that could be made

As I mentioned above, most of the processing happens during the calculation of $c3$. That could be improved by taking of the symmetrical properties of matrix A , as the graph is undirected.

Unfortunately, I couldn't find a way to take advantage of this property and I was recalculating half of the matrix A^2 . If I were to find a way, it would have drastically reduced the runtime of both my sequential and parallel implementations.

Notes

The parallel test above were run using a GTX 1050Ti GPU. Results may vary using a different GPU.

In order to run the python scrip to test a new graph, you need to set up an environment first and install SciPy.

I have included two extra examples to verify the correctness of the code, which are named: `example_cude.cu` and `example_sequential.c` . They use the graph in the included picture to calculate the requested.

I also have included the indices and data of the auto matrix, so the program can run without running the python script, but you have to download them through my drive as they were too big to upload to github. Download them [here](#). (They should be in the same directory as the programs in order to work)

All my source code can be found [here](#).

Arvanitis Themistoklis,
10118