XMT-HW2: Breadth-First Search

Course:	ENEE459P/ENEE699, Fall 2010
Title:	Breadth-First Search
Date Assigned:	October 20th, 2010
Date Due:	November 2nd, 2010 **11:59pm**
Contact:	Fuat Keceli - keceli@umd.edu

1 Problem Statement

Breadth first search in parallel: Given a connected undirected graph $G(V,E)$ and a vertex $s \in V$, the breadth-first search (BFS) method visits vertices in the following order: First, visit s, then visit (in some order) all the vertices $w \in V$, where the edge $(s,w) \in E$; denote the set of these vertices by V_1, and the singleton set consisting of s by V_0; in general, V_i is the subset of vertices of V, which are adjacent on a vertex in V_{i-1} and have not been visited before (i.e., they are not in any of the sets $V_0, V_1, \ldots, V_{i-1}$). Each set V_i is called a layer of G and let h denote the number of layers in G.

The input graph will be stored using **incidence lists**. Let $V = 1,\ldots,n$ and $|E| = m$. In this representation, the edges are stored in an array of length $2m$. This vector will contain first all the edges incident on vertex 1, then all the edges incident on vertex 2, and so on. Note that each edge will appear twice in this vector (for an undirected graph).

This data structure will be provided using the following arrays:

- `edges[2m][2]`: the start and end vertex for each edge, grouped by the starting vertex.
- `vertices[n]`: the index in the `edges[]` array where the adjacent edges for each vertex begin.
- `degrees[n]`: the degree of each vertex

An example is shown in figure 1.

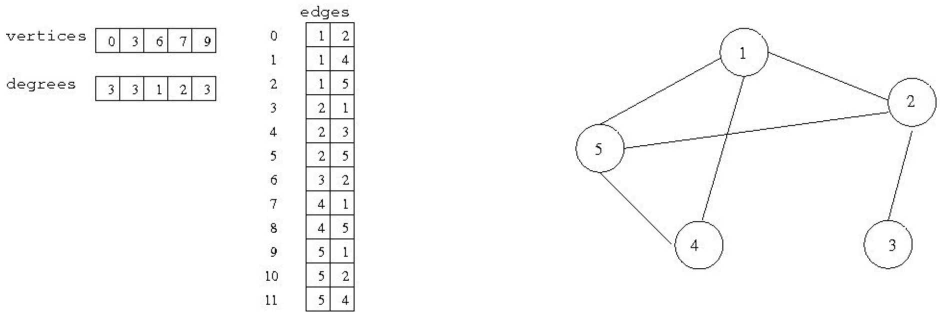

Figure 1: Incidence lists representation

2 Assignment

You will be required to submit two implementations with different solutions for the BFS problem:

1. Serial Implementation:
   - Describe a serial algorithm in the file algorithm.s.txt
   - Provide a brief work and time complexity analysis of this algorithm. Append this analysis to the file algorithm.s.txt
   - Write an XMTC program (XMTSER) that executes this algorithm. Name your code file bfs.s.c **Important:** The program should have as a result the lengths of the shortest path from the start vertex to all the other vertices stored in the array called `level`.
   - Run this program using the data sets given in the Input section.
   - Collect the number of clock cycles for each run into file table.txt (see Output section).

2. Nested Parallel Implementation:
   - Describe a parallel algorithm using nested spawns in the file algorithm.np.txt.
   - Provide a brief work and time complexity analysis of this algorithm. Do this analysis in two different ways: first taking into account that the inner spawn is serialized, and second assuming that the nested spawn is truly supported (all threads of the nested spawns run in parallel and are created in O(1) time). Append this analysis to the file algorithm.np.txt
   - Write an XMTC program (XMTPAR) that executes this algorithm. Name your code file bfs.np.c **Important:** The program should have as a result the lengths of the shortest path from the start vertex to all the other vertices stored in the array called `level`. Your program should use nested spawn sections and should not contain `sspawn` statements.
   - Run this program using the data sets given in the Input section.
   - Collect the number of clock cycles for each run into file table.txt (see Output section).

3 Input

3.1 Setting up the environment

The template files and the Makefile can be downloaded from */opt/xmt/class/xmtdata/*. To get these files, log in to your account in the class server and copy the *bfs.tgz* file from the directory above using the following commands:

```
$ cp /opt/xmt/class/xmtdata/bfs.tgz ~/
$ tar xzvf bfs.tgz
```

This will create the directory *bfs* with following folders: *src*, and *doc*. Put your *c* files to *src*, and *txt* files to *doc*.

Data files are located at a common location in the server (*/opt/xmt/class/xmtdata/bfs*). If you use the Makefile system explained in Section 5, you will not need to explicitly refer to this location. The provided Makefile utilizes command line options to pass the paths to the headera and data files to the compiler.

3.2 Input Format

The type and size of the data structures provided is given in the following table.

#define N	The number of vertices in the graph
#define M	The number of edges in the graph (each edge counts twice)
edges[M][2]	The start and end vertex of each edge
vertices[N]	The index in the `edges` array, at which point the edges incident to vertex begin
degrees[N]	the degree of each vertex
gatekeeper[N]	gatekeeper per vertex
level[N]	result array: stores the distance from the source for each vertex

Note that each edge is listed twice in this input format, i.e. `M` is equal to $2m$, where m was given as the size of the set E in the problem definition (Section 1).

In order to use the same code and dataset with multiple starting nodes, you will assume that a C pre-procesor variable `START` is available. During compilation you will be able to modify this preprocesor variable using `START=...` command line option of the Makefile. This has the same effect as if the first line of your code has the compiler directive: `#define START ...` (substitute an integer in place of `...`). for grading purposes, your program will be compiled and run with different starting nodes using this method. If you do not adhere to this convention, your assignment may not be graded fully.

You can declare any number of global arrays and variables in your program as needed. For example, this is valid XMTC code:

```
#define N 16384

int temp1[16384];
int temp2[2*N];
int pointer;

int main() {
 //...
}
```

3.3 Data Sets

The following two datasets are provided:

Data set	N = # Vertices	M = # Edges	Start node	Header file	Binary File
Hexagon	20	86	0	hexagon/bfs.h	hexagon/bfs.xbo
Large	1000	20000	142	large/bfs.h	large/bfs.xbo
Huge	10000	200000	101	huge/bfs.h	huge/bfs.xbo
Long	10000	200000	0	long/bfs.h	long/bfs.xbo
Web	50000	278934	0	web/bfs.h	web/bfs.xbo

The paths are given with respect to */opt/xmt/class/xmtdata/bfs*, however you will not need these paths unless you do not use the Makefile system. The below list provides details for each data set.

1. **Hexagon graph:** The dataset `hexagon` corresponds to the graph in figure 2.
2. **Large:** This is a rather large graph, generated with an automated tool. In order to test for correctness of your algorithm, you can check that the following (node:level) pairings hold: (199:2), (300:3), (900:3), (401:3).

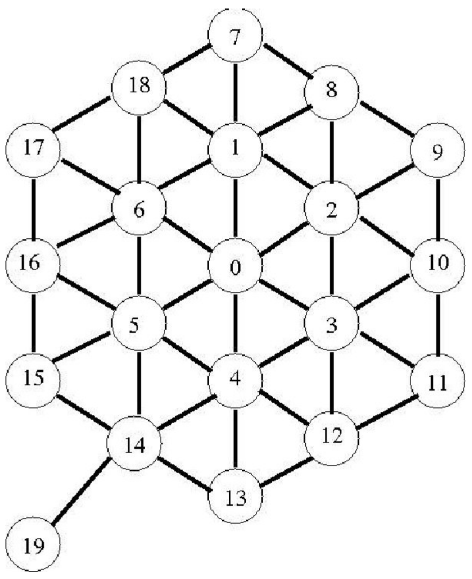

Figure 2: The `hexagon` dataset

3. **Huge:** This is an even larger graph, generated with an automated tool.
4. **Long:** This is a layered graph, with a high diameter. Starting from node 0, a bfs traversal requires a large number of iterations (500), with about 20 nodes visited per iteration on average.
5. **Web:** This is a graph representing a small subsection of the web with edges representing hypelinks between web pages. The degrees of the nodes follow a power law distribution: a few nodes represent very popular pages and have very high degrees (5000+), while most of the nodes have fairly low degrees.

As mentioned earlier, each edge is listed twice in the input files. For example, the undirected graph Huge has 100,000 undirected edges, but it contains 200,000 directed edges in the adjacency list, listed in the array `edges`.

4 Output

Prepare and fill the following table: Create a text file named table.txt in doc. **Do not print anything (including results) while taking these measurements.** Printf statements increase the clock count. Therefore the measurements with printf statements may not reflect the actual time and work done.

Dataset:	Hexagon	Large	Huge	Long	Web
XMTPAR Clock cycles					
XMTSER Clock cycles					

5 Compiling and executing via the Makefile system

For your convenience, a Makefile is provided with the homework distribution. You can use the provided makefile system to compile and run your programs. To run the parallel BFS on the web data set, use the following command in the src directory:

```
> make run INPUT=bfs.np.c DATA=web START=0
```

This command will compile and run the bfs.np.c program with the web data set and a starting node 0. For other programs and data sets, change the start node and the name of the input file and the data set.

If you need to just compile the input file (no run):

```
> make compile INPUT=bfs.np.c DATA=web START=0
```

Above commands will omit the printf statement that is surrounded with the PRINT_RESULT macro. In order to print the results add `PRINT_RESULT=1` to the command line:

```
> make run INPUT=bfs.np.c DATA=web START=0 PRINT_RESULT=1
```

With this addition, the makefile will define the PRINT_RESULT macro during compilation. **Do not print the results while obtaining cycle counts.**

You can get help on available commands with

```
> make help
```

Note that, you can still use the `xmtcc` and `xmtfpga` commands as in the earlier assignments. You can run with the makefile system first to see the commands and copy them to command line to run manually.

6 Submission

The use of the make utility for submission *make submit* is required. Make sure that you have the correct files at correct locations (*src* and *doc* directories) using the make submitcheck command. Run following commands to submit the assignment:

```
$ make submitcheck
$ make submit
```

7 Reference Performance Results

The cycle counts for our reference implementation for some of the input datasets are provided in the following table. The speedup row is calculated by dividing the serial cycle count by the parallel cycle count.

Dataset:	Large	Huge	Long	Web
XMTPAR Clock cycles	46.4K	608.8K	1.5M	4.1M
XMTSER Clock cycles	1M	16.6M	13.9M	44.6M
Speedup	21.6	27.3	9.3	10.9

8 Hints and remarks

In implementing the parallel solution, you might come across a case where two or more threads try to write to the same memory location simultaneously. The XMT platform forbids arbitrary concurent writes, the only mechanism that can be used in this case being prefix-sum to memory `psm()`.