문제 3 (학부/대학원 공통): 넓이 우선 탐색(Breadth-First Search) 알고리즘 병렬화 (배점: 학부 40 점, 대학원 30 점)

문제 개요

SNS와 스마트폰의 등장 이래 데이터가 천문학적인 숫자로 쌓이고 있다. 전문가들은 데이터를 분석하는 빅데이터가 인터넷 이후 기업에 가장 큰 영향을 미칠 것으로 보고 있다. 이를 분석하기 위한 방법으로 그래프라는 자료 구조를 많이 사용한다. 아래 표 1[1]은 현실 세계에서의 그래프 규모에 대한 참고자료이다.

그래프는 자료 요소에 해당하는 정점(vertex)과 정점 간의 관계를 표시하는 간선(edge)으로 구성되며, 그래프 G는 G=(V, E)로 표현한다. 이에 대해 구성하고 검색하는 것은 중요한 절차이다.

	정점(vertex) 수	간선(edge) 수
Facebook	2,937,612	41,919,708
Wikipedia	3,566,908	84,751,827
LiveJournal	4,847,571	85,702,475
Twitter	61,578,415	1,468,365,182
Graph500	536,870,912	8,589,926,431

표 1. 현실 세계에서의 그래프 규모

본 문제는 그래프를 구성하는 모든 정점(vertex)들을 체계적으로 방문하는 방법으로 많이 쓰이는 넓이 우선 검색(Breadth-First Search) 알고리즘을 병렬화하는 문제이다. 넓이 우선 검색은 시작 정점(Starting Vertex)을 방문한 후 시작 정점에 인접한 모든 정점들을 우선 방문하는 방법이다. 더 이상 방문하지 않은 정점이 없을 때까지 넓이 우선 검색을 적용한다. 아래 그림 1의 그래프에서 이 검색 방법을 사용하면 ㉠→㉡→㉢→㉣→㉤→㉥→㉦→㉧ 순으로 방문하게 된다.

[1] Nadathur Satish, Navigating the Maze of Graph Analytics Frameworks using Massive Graph Datasets, SIGMOD 2014

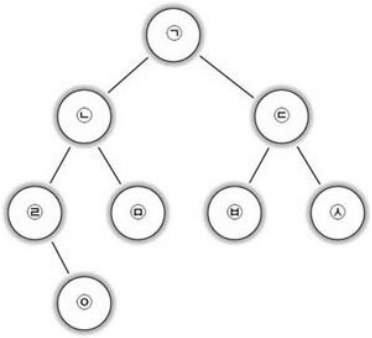

그림 1. 그래프 예제

최근에 가장 빠른 슈퍼컴퓨터를 측정하는 방법으로 GRAPH500 벤치마크(www.graph.org)가 도입되고 있는데, 그래프 넓이 우선 검색의 성능을 측정한다. 본 문제에서는 GRAPH500 벤치마크에서 사용하는 R-MAT(Recursive MATrix) Random 그래프 생성기에서 생성된 그래프를 입력자료로 받아 병렬화된 넓이 우선 검색 기능을 구현한다.

문제 설명

Serial 코드

주어진 Serial 코드는, 시작 정점(startvtx)과 그래프의 데이터 구조(아래 설명)를 포함하는 바이너리 파일을 입력으로 받으며, 넓이 우선 검색을 이용해 시작 정점에서 연결 가능한 그래프 데이터 구조의 모든 정점들을 탐색한다.

탐색 과정에서 각 정점에 대해서 level 번호를 할당하게 되는데, 이의 예가 그림 2 (sample 입력파일) 에 자세히 제시되어 있다. 시작 정점인 1은 level 번호 0을 할당한다. 정점 1에서 이웃(연결)하고 있는 정점들은 2,5,3이다. 정점 2,5,3은 level 1로 할당한다. 계속해서 각 정점들이 방문하지 않은 정점가운데 이웃하고 있는 정점들인 4,6은 level 2로 할당하고, 더 이상 다른 정점으로 연결되어 있지 않는 level 3까지 찾게 되면 모든 프로세스가 완료된다. 프로세스가 끝나면, 코드는 총 level의 수 (4개) 와 각 level별로 탐색한 정점의 개수를 화면에 출력한다.

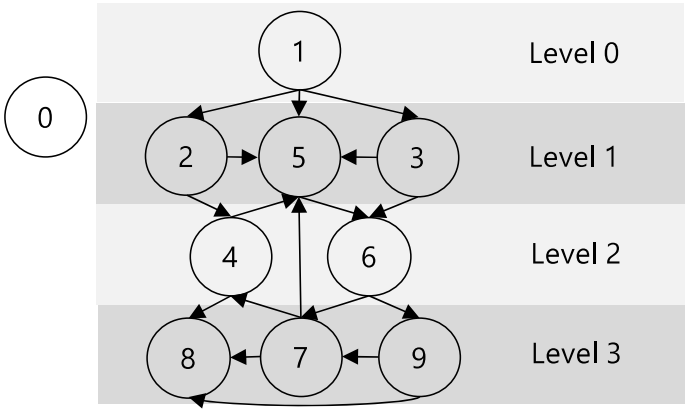

그림 2. sample 그래프 예제

그래프 데이터 구조

자료구조는 "compressed sparse adjacency lists"라는 그래프 구조를 사용하는데, 이는 희소 행렬(sparse matrix) 자료 구조에서 많이 사용하는 CSR(Compressed sparse row) 형태와 유사하다. 본 그래프 구조에서는 두 가지 배열 nbr과 firstnbr을 사용하는데, 배열 nbr에는 정점 0[2] 부터 정점 n까지의 모든 이웃들이 차례로 저장 된다. 배열 firstnbr에는 각 정점의 첫 번째 이웃 정점이 저장된 nbr 배열 인덱스가 저장된다. 그림 2를 이용해 firstnbr 배열을 할당하는 방식에 대한 설명과 할당된 nbr/firstnbr 배열의 값이 아래에 제시되어 있다.

- 정점의 개수는 0부터 n까지 총 n+1개
- 정점 a와 연결된 이웃 정점들은 nbr 배열의 “firstnbr[a]번”째에서 “firstnbr[a+1]-1번” 째 사이에 저장된다. (아래에 예를 참고)
    - 예1) 정점 2와 연결된 이웃 정점들은 nbr배열의 firstnbr[2]번째부터 차례대로 저장된다. 정점 2의 첫번째 첫번째 이웃 (정점 5)가 nbr 배열의 4번째에 저장되어 있으므로, firstnbr[2]의 값은 3이 되며 (배열의 인덱스는 0부터 시작), 정점 2의 이웃 정점은 총 2개 이므로 nbr 배열의 4번째와 5번째에 저장된다. 따라서, firstnbr[3]은 5가 된다 (여섯번째).
    - 예2) 정점 8은 연결된 이웃 정점들이 없으므로, firstnbr[8]과 firstnbr[9]의 값은 동일하다.
- nbr배열의 요소개수는 정점들을 연결하는 간선(edge)들의 개수와 같다.

[2] 본 문제에서는 정점번호로 32BIT 정수만 사용한다. (GRAPH500 은 64BIT 정수 사용)

- firstnbr 배열의 요소 개수는 n+2가 된다. 마지막 정점 n의 이웃 정점이 nbr 배열의 몇 번째 인덱스에 저장되었는지 알기 위해서는 firstnbr[n+1]이 필요하기 때문이다.

정점(vertex)들을 연결하는 간선(edge)들:
1->2, 1->5, 1->3, 2->5, 2->4, 3->5, 3->6, 4->5, 4->8, 5->6, 6->7, 6->9, 7->5, 7->4, 7->8, 9->7, 9->8

firstnbr = 0 0 3 5 7 9 10 12 15 15 17

nbr = 2 5 3 5 4 5 6 5 8 6 7 9 5 4 8 7 8

그림 3. 그림 2의 예를 이용해 구성한 firstnbr/nbr배열 및 정점들을 연결하는 간선의 리스트

병렬 넓이 우선 검색

병렬 넓이 우선 탐색의 기본 원리는 각 level별로 병렬로 정점들을 동시에 탐색하는 것이다. 이때 각 정점들을 분할하는 알고리즘에 의해 성능 차이가 날 것이다. 디버깅을 위해서 주어진 샘플 입력파일(sample.g)은 병렬 코드가 오히려 느릴 수 있다. 하지만, 최종적으로 결과 검증에서 사용할 파일인 rmat27.g[3]는 병렬 수행이 serial 코드보다 나아야 한다.

[3] GRAPH500 에서 가장 작은 class 인 Toy 보다 scale 이 하나 더 큰 규모인 27 이고 134,217,719 개의 정점과 2,147,483,648 개의 간선을 가지고 있음

참고 및 유의 사항

1. 결과 검증은 rmat27.g 입력 파일(해당 시스템의 /tmp/rmat27.g 파일을 복사)을 사용하여 level 총 수와 각 level 별 탐색한 정점 개수가 다음과 같이 화면에 표출해야 한다.

```
Starting vertex for BFS is 12

Breadth-first search from vertex 12 reached 11 levels and 15519839 vertices.
level 0 vertices: 1
level 1 vertices: 2
level 2 vertices: 189
level 3 vertices: 49780
level 4 vertices: 4545171
level 5 vertices: 9533846
level 6 vertices: 1315040
level 7 vertices: 71899
level 8 vertices: 3684
level 9 vertices: 217
level 10 vertices: 10
```

그림 4. 출력 예시

2. 코드는 주어진 pbfs.c(혹은 pbfs.f90) 파일을 수정해서 작성해야 하고, 주어진 makefile (수정 불가)로 컴파일해야 한다.
3. 코드내의 graph 자료구조(정점수:nv는 32Bit 정수타입, 간선수:ne는 64Bit 정수타입)는 수정이 불가능하다.

평가 방법

1. 병렬코드를 32 core 에서 실행, Graph500 에서 사용되는 GTEPs (giga- traversed edges per second)의 단위로 성능을 측정한다. GTEPs 숫자가 높을수록 성능이 좋다.
2. 파일을 읽는 부분은 성능에 포함되지 않는다.