Maximum Independent Set in a Graph
(Team MAXIMUS)

By-
Chinmay Ladage
Deepak Ravi Shankar
I. Overview

Our project is to find out the Maximum Independent Set in a graph. Here are the definitions of the important terms in this project:

**Independent Set** is a set of vertices in a such that there are no adjacent vertices in the set.
**Maximal Independent Set** is an independent set which is not a subset of any other independent set.
**Maximum Independent Set** is an independent set with largest number of vertices.

Example: For the given graph below, following are the different sets,

![Graph Image]

**Independent Sets:**
(A, E), (E, D), (C, D), (B, C), (B, E), (A, B, E)
Here for set (A, E), A and E are not adjacent, hence they form an independent set.
All the other sets consist of nonadjacent vertices.

**Maximal Independent Sets(MIS):**
(E, D), (C, D), (B, C), (A, B, E)
Set (A, E) is also a subset of independent set (A, B, E). Hence it is not a Maximal Independent Set.

**Maximum Independent Sets(MaxIS):**
(A, B, E)
Out of all the possible Maximal Independent Sets, (A, B, E) is the largest, hence it is the Maximum Independent Set for the above graph.
II. Computational Problem

Maximum Independent Set is a NP hard problem. In short, if we are given a set of vertices, we can find out in polynomial time whether it is an independent set.

To find out the Maximum Independent Set, we need to find out all the possible Maximal Independent Sets in the graph, and the largest of them will be the Maximum Independent Set. But no such algorithm exists that can find out all the Maximal Independent Set, hence finding Maximum Independent Set in a graph is NP hard problem.

In our approach, we are going to find out large number of Maximal Independent Set, so that we can find out an approximate (near to actual) Maximum Independent Set.

For this, we will be using a greedy approach,
  - Choose a vertex and add it to the independent set and remove all its neighbors from V(list of all vertices).
  - When the V becomes empty, we have a Maximal Independent Set.

We repeat this for ‘n’ times, where ‘n’ is a very larger number. Higher the value of ‘n’, better the approximation towards Maximum Independent Set.
III. Research Paper 1

Title: Parallel algorithms for the Maximal Independent Set problem in graphs
Authors: Luis Barba
Journal: School of Computer Science Carleton University, Ottawa, Canada
Date: December 2012
Pages: 11

Problem Addressed:
There already exists a sequential algorithm for finding Maximal Independent Set using priority ordering based on degree of vertices. Parallelizing that sequential algorithm in optimal way to get the Maximum Independent Set in lesser runs.

Approach:
Heuristic approach is used in this paper. The approach is as follows:
- In the parallel greedy algorithm approach, we choose vertex v which have no higher priority neighbors in the graph. If such a v is found, then we add it to the MIS regardless of what happens with other higher priority vertices than v.
- This is the key factor to obtain the parallelism. There is heuristic function that will check the priority before selecting each v.

What are we using:
Using heuristics increases the chances of finding the Maximum Independent Set is lesser number of o
Title: Towards Maximum Independent Set on Massive Graph
Authors: Yu Liu, Jiaheng Lu, Hua Yang, Xiaokui Xiao, Zhewei Wei.
Journal: Proceedings of the 41st International Conference on Very Large Data Bases
Date: September 2015
Pages: 2122-2133

Problem Addressed:
The existing methods to find Maximum Independent Set require at least memory space of linear size of the input graph. Also, the cost of memory access increases with increasing size of graph. As the size of graphs is ever increasing, this is a serious concern.

Approach:
Greedy Algorithm

- Input: A sorted adjacent-list file for graph G
- Output: A Maximal Independent Set of G

\[
\begin{align*}
\text{for } v \in V \text{ of } G \text{ do } \\
\text{State}[v] \leftarrow \text{INITIAL}; \\
\text{for } v \in V \text{ of } G \text{ do } \\
\text{if State}[v]=\text{INITIAL} \text{ then } \\
\text{State}[v] \leftarrow \text{IS}; \\
\text{for each } u \text{ in } N(v) \text{ do } \\
\text{if State}[u]=\text{INITIAL} \text{ then State}[u] \leftarrow \text{~IS}; \\
\text{Return all vertices whose states are IS;}
\end{align*}
\]
Example to illustrate that figure (a) represents Maximal Independent Set generated by greedy algorithm, whereas figure (b) is required Maximum Independent Set [2].

**One-k swap algorithm:**
This algorithm exchanges MIS vertices with more non MIS vertices. In the example, this algorithm exchanges vertex 1 by vertices 3, 4 and 5 to the size of the independent set.
- Input: A sorted adjacent-list file for graph G and initial independent set.
- Output: A larger independent set of G

To solve the problem of random memory access while swapping, 6 states of each vertices are defined.
- IS in the Maximal Independent Set
- Non IS not in the Maximal Independent Set
- Adjacent adjacent to only one Maximal Independent Set node
- Conflict being in non-independent set in next iteration
- Protected being in independent set in next iteration
- Retrograde being in non-independent set in the next iteration

**What are we using:**
Idea of swapping an existing MIS vertex by non MIS vertices. This should give a larger MIS.
V. Research Paper 3

Title: Finding Near-Optimal Independent Set at Scale
Authors: Lamm, S. and Sanders, P. and Schulz, C. and Strash, D. and Werneck, R. F
Journal: Proceedings of the 16th Meeting on Algorithm Engineering and Experimentation
Date: 2016

Problem Addressed:

To enhance the given independent set by computing new offsprings in combination to a local search algorithm to find a better Maximum Independent Set

Approach:

In this paper Genetic algorithm is used.
- Input: An independent set obtained from one-k sw
- Output: A Maximum Independent Set of G

The algorithm is as follows:
- Create a population of individuals – a set of independent sets and combine them to get a better offspring. Initially, we select a population by doing k-swap technique.
- The offspring is improvised using mutation and combine operators.
- The algorithm ends when the population converges. We keep the mutation going on so that we get a very high population

What are we using:

Since this uses genetic approach, we are ensured to Maximum Independent Set of all the Maximal Independent Sets.
Sequential Design flow:

1. Randomly choose a vertex $v$ from the set of all vertices
2. Add the vertex $v$ to the MIS
3. Remove from $V$ the vertex $v$ and its neighbors given by $N(v)$

- If $MIS > MaxIS$, $MaxIS = MIS$
Sequential implementation:

```java
for (int i = 1; i < Integer.parseInt(args[1]); i++) {
    Random rnd = new Random();

    // mis stores the maximal independent set for the current iteration
    IntList mis = new IntList();

    // temporary list of vertices
    IntList VList = new IntList();

    // populating the VList
    for (int iv = 0; iv < V; iv++) {
        VList.addLast(iv);
    }

    // while the temporary list is not null, keep repeating
    while (!VList.isEmpty()) {
        // generate a random vertex index
        int n = rnd.nextInt(VList.size());

        // get the vertex at the generated random index
        int v = VList.remove(n);

        // add the vertex to mis
        mis.addLast(v);

        // remove all the neighbors of v from the temporary list
        for (int j = 0; j < VList.size(); j++) {
            if (dM.adjMatrix[v][VList.get(j)] == 1) {
                VList.remove(j);
            }
        }
    }

    // replace maxIS with current mis if mis is larger
    maxIS.update(new MAXIS(mis));
}
```
VII. Parallel Program

Parallel Design flow:

A - Randomly choose a vertex v from the set of all vertices
B - Add the vertex v to the MIS
C - Remove from V the vertex v and its neighbors given by N(v)
Parallel implementation:

```java
parallelFor(1, Long.parseLong(args[1])).exec(new LongLoop() {
    // thread local copy of the reduction variable
    MAXISVbl thrMaxISVbl;

    public void start() {
        thrMaxISVbl = threadLocal(maxISVbl);
    }

    public void run(long i) throws Exception {
        Random rnd = new Random();
        // local MIS stores the maximal independent set for the
        // current iteration
        IntList MIS = new IntList();

        // thread local list of vertices
        IntList Vlist = new IntList();

        // populating the Vlist
        for (int iv = 0; iv < V; iv++) {
            Vlist.addLast(iv);
        }

        // while the temporary list is not null, keep repeating
        while (!Vlist.isEmpty()) {
            // generate a random vertex index
            int n = rnd.nextInt(Vlist.size());
            // get the vertex at the generated random index
            int v = Vlist.remove(n);

            // add the vertex to MIS
            MIS.addLast(v);

            // remove all the neighbors of v from the temporary list
            for (int j = 0; j < Vlist.size(); j++) {
                if (G.getAdjMatrix[v][Vlist.get()] == 1) {
                    Vlist.remove();
                }
            }

            // parallel reduction
            thrMaxISVbl.m.update(new MAXIS(MIS));
        }
```
The project consists of 5 main class files:

- **SeqMaxIS.java**: the sequential implementation
- **ParallelMaxIS.java**: the parallel implementation
- **AdjMatrix.java**: holds adjacency matrix for a graph
- **MAXIS.java**: storage variable used for reduction
- **MAXISVbl.java**: reduction class

To test the software, we are making use of Graph.java, RandomGraph.java and Edge.java provided for programming project 2 by Prof. Alan Kaminsky.

To compile the software on RIT CS machines, place all the above java files (except for RandomGraph.java which is used only for testing) together and run these commands:

1. export CLASSPATH=.:./parajava/pj2/pj2.jar
2. javac *.java
IX. User’s Manual

Our project runs on terminal and command prompt. To run the project, compile using the instructions from Developer’s Manual and then use the following commands:

Both the classes SeqMaxIS and ParallelMaxIS take 2 arguments:
1. The graph (we are using RandomGraph class for demo)
2. Number of iteration ‘n’ as mentioned in sequential and parallel design description.

For sequential program:
java pj2 debug=makespan cores=1 SeqMaxIS "RandomGraph(20,40,142863)"
10

Only for this demo purpose, we are printing all the Maximal Independent Sets.
But since $n$ and the graph can be very large, we for final implementation we are only printing Maximum Independent Set.

For parallel program:
```
java pj2 debug=makespan cores=1 SeqMaxIS "RandomGraph(20,40,142863)"
```

10

As we are using Parallel Java 2 Library by Prof. Alan Kaminsky, we can use the configurations specified in the documentation to modify the flow or performance of the parallel program.

We are using all default configuration, because it is giving us ideal performance.
In this demo, we are using a large graph with $n=1000$ and 4 cores for parallel computing of the Maximum Independent Set.
X. Strong Scaling

Running time vs Cores:

As the size of graph increases, the time taken increases. For each case, as the cores are increased, the time taken decreases.

Speedup vs Cores:

It is clearly visible from this graph that for all test cases, the lines coincide with almost same slope.
Efficiency vs Cores:

As you can see the lines for different cases almost coincide. There is a small deviation in efficiency of V=750 from others, but this is caused due to random approach used, which may sometimes deviate a little.
XI. Strong Scaling results explained

**Running Time vs Cores:** If we compare the lines of different cases in running time vs cores result, almost all of them show the same trend with change in number of cores. This shows that our parallel program behaves the same with any number of vertices.

**Speedup vs Cores:** The speedup vs cores results indicate that our parallel program gives ideal speedup for any number of vertices.

**Efficiency vs Cores:** The efficiency vs cores results indicate that the efficiency remains the same in all test cases.

All the above points indicate that our parallel program gives ideal results.
XII. Weak Scaling

Runtime vs Cores:

There is small deviation in V=500 because of the random algorithm.

Size-up vs Cores:

This graph shows that all the lines coincide.
This graph shows that all the lines follow the same trend.
XIII. Weak Scaling results explained

**Running Time vs Cores:** If we compare the lines of different cases in running time vs cores result, we can see lines parallel to each other indicating ideal results.

**Size-up vs Cores:** The coinciding lines in size-up vs cores graph indicate that our parallel program gives ideal size-up in all cases.

**Efficiency vs Cores:** The efficiency vs cores results indicate that the efficiency remains the same in all test cases.

All the above points indicate that our parallel program gives ideal results.
XIV. Future work

We are looking forward to implement a neural network by creating decision trees that will decide the best set of nodes to select for instead of doing a random selection of nodes. This would significantly lessen the running time of the algorithm and the efficiency also increases. The reason behind increase in efficiency is because, the decision tree uses the hidden mark on models to decide the probability for each node to select, and as the probability for a node increases, the chance of arriving at a Maximum Independent Set is significantly greater than a node with less probability.
XV. Lessons Learned

While doing this project, we learnt how to design an algorithm that would solve the problem using a parallel approach. We also learned how the clusters interact and how they communicate with each other while processing large input.

In trying to solve the problem of Maximum Independent Set, we learned that how a NP hard problem which does not have a solution in polynomial time can be solved in a significantly low amount of time using a parallel approach.
XVI. Member Contributions

Both the members have worked together at every stage in the project. Both members provided valuable inputs and equal contribution towards achieving the goal of project completion.

The unique contributions of each team members are as follows:

**Chinmay Ladage:**
- Detailed analysis of research paper 2
- Performed Strong Scaling
- Re-verified weak scaling

**Deepak Ravi Shankar:**
- Detailed analysis of research paper 1
- Performed Weak Scaling
- Re-verified strong scaling
XVII. REFERENCES

