

Parallel Dijkstra Algorithm

Yesheng Ma
Ke Chang

October 31, 2016

Overview

① Introduction

② Environment

③ Implementation

Parallel Dijkstra Algorithm

- Similar to serial version
- Exploit parallelism in find the vertex with least distance
- The key idea is just **map** and **reduce**

Build and Test

- Use GNU Makefile to relief you from repeating
make, make clean etc
- Write Bash scripts with command line arguments to test:
 - in batch
 - with specified argument

MPI API

- `MPI_Init`: initialize MPI context

MPI API

- `MPI_Init`: initialize MPI context
- `MPI_Comm_size`: get size of MPI communication space

MPI API

- `MPI_Init`: initialize MPI context
- `MPI_Comm_size`: get size of MPI communication space
- `MPI_Comm_rank`: get the rank of current thread, i.e. `my_rank`

MPI API

- `MPI_Init`: initialize MPI context
- `MPI_Comm_size`: get size of MPI communication space
- `MPI_Comm_rank`: get the rank of current thread, i.e. `my_rank`
- `MPI_Allreduce`: similar to reduce in FP(list→reduced value)

MPI API

- `MPI_Init`: initialize MPI context
- `MPI_Comm_size`: get size of MPI communication space
- `MPI_Comm_rank`: get the rank of current thread, i.e. `my_rank`
- `MPI_Allreduce`: similar to reduce in FP(list→reduced value)
- `MPI_Gather`: gather small arrays to form a large one

Core Implementation

- Encapsulate parallel Dijkstra algorithm
 1. Pass more arguments to Dijkstra function
 2. But reusability and modularity gained

```
void Dijkstra(int loc_mat[], int loc_dist[],  
              int loc_pred[], int loc_n, int my_rank, int n);
```

Impl Cont'd

The implementation itself is quite easy:

- ① Initialization: `loc_dist`, `loc_known`, `loc_pred`
- ② Do $n-1$ times iteration:
 - ① `Find_min_loc_dist` and store to `my_min`
 - ② do Allreduce to get `glbl_min`
 - ③ for all unknown vertices, update if possible
- ③ Algorithm finished, print necessary message.

Deployment on PI

There are generally 3 steps to deploy the algorithm on PI:

- ➊ Add log information in code.
- ➋ Write the slurm script.
- ➌ Process the output file.

Log information

There are mainly two kinds of information we care about.

- ① Serial time: `sBegin`, `sEnd`
- ② Parallel time: `pBegin`, `pEnd`

Slurm Script

- ① Grammar just like common shell script.
- ② The script reads the input file in order and run each file with 1, 2, 4, 8, 16 processes respectively.
- ③ Save the result in *A_B.out* file where A is number of progresses and B is input file number.

Process output

Write a Python script to process output.

Potential Risks

Actually, the graph is far from real-world complex networks.

- May have wired topological structures \Rightarrow other algorithms
- May have large edge weights \Rightarrow introduce BigInt like Java

The End