HPC - MPI - work sharing and broadcast

Table of Contents

1 Introduction

The Floyd-Warshall algorithm calculates the shortest paths between all N^2 pairs of nodes in a weighted graph with N nodes. Assume that the matrix M is initialized with non-negative weights (distances) between neighboring nodes: $M[i,j]$ is the distance from node i to node j. The Floyd-Warshall algorithm is defined as follows:

```
for (k = 0; k < N; k++)
   for (i = 0; i < N; i++)
      for (j = 0; j < N; j++)
         if (M[i,j] > M[i,k] + M[k, j])
            M[i,j] = M[i,k] + M[k, j];
```

We will implement the Floyd-Warshall algorithm on a distributed memory machine.

2 Files

Files for today: mpi-lab-02.zip

- `hello-world-bcast-par.cpp` a hello world using broadcast.
- `graph-base.h` definition of a graph and some basic functions. We won't modify this file.
- `graph-base.cpp` implementation of the basic functions. We won't modify this file.
- `graph-utils.h` additional functions for the graph. We shouldn't need to modify this file.
- `graph-utils-seq.cpp` this implementation assumes a sequential execution. We won't modify this file.
- `graph-utils-par.cpp` this implementation assumes a parallel execution. **To implement (exercise 1)**
- `generator-seq.cpp` generates and shows the graph, sequential execution. We won't modify this file.
- `generator-par.cpp` generates the graph, distributes it among processes and gathers the results. We won't modify this file.
- `floyd-warshall-seq.cpp` sequential implementation of Floyd-Warshall.
- `floyd-warshall-par.cpp` parallel implementation of Floyd-Warshall. **To implement (exercise 2)**

3 Distributing work among processes

We will use a natural distribution of data and work among processes corresponding to the parallelization of the second loop of Floyd-Warshall (Why not the first? Why not the third?). This corresponds to a so-called row-block data distribution: each process stores roughly the same number of consecutive rows.

Exercise 1:

Implement functions from `graph-utils-par.c` that (1) partition the matrix; and (2) write the matrix to std-out. Implement a partitioning in which the number of rows stored by each pair of processes differs by at most one. Use `graph-utils-seq.c`, a sequential version, as a template. `generator-par.c` tests your implementation: this should give exactly the same result as `generator-seq.c`.

Assume that processes are memory-bound, i.e., no process can store more than O(1) blocks. Note that the matrix is generated by a stateful random number generator, thus only a single process should generate the matrix. Similarly, a single process should write the matrix to the std out. (problems to consider at home: How limiting these assumptions are for a large-scale system? Which methods can we use to gain more parallelism here?)

4 Broadcast

As discussed during the lecture, if a communication pattern involves more than a pair of processes, MPI collective communication functions http://mpi-forum.org/docs/mpi-3.1/mpi31-report/node95.htm#Node95 should be more efficient.

The definition of broadcast is as follows:

```
int MPI_Bcast(void* buffer,   /* the message will be written here */
                              /* if my_rank==root, the message will be read from here */
              int count,      /* number of items in the message */
              MPI_Datatype datatype, /* type of data in the message */
              int root,       /* if my_rank==root, I'm sending, otherwise I'm receiving */
              MPI_Comm comm   /* communicator to use */
             );
```

See `hello-world-bcast-par.c` for an example application: process 0 broadcasts a message to other processes.

5 Implementation of the distributed Floyd-Warshall algorithm

Implement the distributed Floyd-Warshall algorithm by completing fixmes in `floyd-warshall-par.c`. You can use `floyd-warshall-seq.c`, a sequential implementation, as a template.

Compute speed-ups (strong scaling) for different sizes of the graph and 1-16 nodes.

Date: 2024/04/19

Author: Krzysztof Rządca (based on Konrad Iwanicki's materials)

Created: 2024-04-19 Fri 11:20
Emacs 25.3.50.1 (Org mode 8.2.10)
Validate