HPC - MPI - asynchronous communication

Table of Contents

1 Files

Files for lab 3: mpi-lab-03.zip

- `scatter-gather-max.cpp`: a distributed computation of maximum using scatter-gather group communication.
- `ring-nonblocking.cpp`: each process exchanges its rank with its left and right neighbor; processes use non-blocking communication.
- `laplace-seq.cpp`: a sequential implementation of the Laplace method.
- `laplace-par.cpp`: a template for your distributed implementation **to implement**.
- `lamplace-common.h`: declarations of some helper functions.
- `lamplace-common.cpp`: definitions of some helper functions.

2 Collective communication

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 declarations of these functions are similar to each other, see `MPI_Scatter` below:

```
int MPI_Scatter(const void* sendbuf, /* if my_rank==root, the message will be read from here */
                int sendcount, /* number of elements to send to each process */
                MPI_Datatype sendtype,
                void* recvbuf, /* where to put the message */
                int recvcount, /* number of elements a receiver is expecting */
                MPI_Datatype recvtype,
                int root, /* if my_rank==root, I'm sending from my sendbuf */
                MPI_Comm comm)
```

```
      data                                 data
P0: D1 D2 D3 D4      scatter      P0: D1 -- -- --
```

```
P1: -- -- -- --      ──────→     P1: D2 -- -- --
P2: -- -- -- --      ←──────     P2: D3 -- -- --
P3: -- -- -- --       gather     P3: D4 -- -- --
```

Useful functions:

- `MPI_Allgather`: same as gather, but each processor receives the result.
- `MPI_Reduce`: aggregates the input data (e.g.: sum, max, etc.) and places the aggregation at the root process.
- `MPI_Allreduce`: same as reduce, but all processes receive the result.

`scatter-gather-max.cpp` illustrates how to compute `max` in a distributed way.

3 Non-blocking communication

Standard MPI functions may block (as when a function returns, the communication buffers are ready to be re-used, so the operation has to be completed). Thus, to overlap communication with computation, MPI defines a series of asynchronous operations. Below, we focus on point-to-point operations; but the standard defines non-blocking collectives as well.

- `MPI_Isend`: orders a non-blocking (asynchronous) send. The buffer cannot be reused until the send operation completes.
- `MPI_Irecv`: orders a non-blocking receive.
- `MPI_Wait`: waits until a request is completed: `int MPI_Wait(MPI_Request *request, MPI_Status *status)`
- `MPI_Test`: tests whether a request is completed: `int MPI_Test(MPI_Request *request, int *flag, MPI_Status *status)` (returns `flag==true` if a request is completed).
- `MPI_Waitany`, `MPI_Waitall`: waits until any/all of the requests are completed. `int MPI_Waitall(int count, MPI_Request array_of_requests[], MPI_Status array_of_statuses[])`.

`ring-nonblocking.cpp` shows how to use non-blocking communication with `MPI_Waitall`. Each process exchanges its rank with its left and right neighbor.

4 NSEW Stencil with a Checkerboard Algorithm

The Laplace's equation https://en.wikipedia.org/wiki/Laplace's_equation describes equilibrium states in many natural processes. One of the simplest analysis methods is to discretize the equation and the boundary conditions and then solve the resulting equations numerically using the successive over-relaxation (SOR) method https://en.wikipedia.org/wiki/Successive_over-relaxation.

We will implement a distributed version of SOR on a two-dimensional system of $n \times n$ points for a simple Laplace equation $\delta^2(x,y)/\delta x^2 + \delta^2(x,y)/\delta y^2 = 0$. The numerical algorithm is a NSEW stencil executed for all grid points. It works as follows:

1. Compute the average w of the 4 (NSEW) neighbors.
2. Update the point value v as a weighted average of its previous value and the neighbors' average: $vnew = (1-\omega)v + \omega w$ (ω is a parameter of the method).

3. If the difference $|vnew - v|$ is greater than δ (a parameter), execute another iteration of the algorithm.

We will use a checkerboard algorithm to update the stencil. Each iteration of the algorithm is composed of two stages. First, the algorithm first updates the "white" fields of the checkerboard using the values of the "black" fields from the previous iteration. Then, the algorithm updates the "black" fields using the newly-updated "white" fields. Note that this algorithm differs from the standard NSEW stencil discussed during the lectures: in the standard NSEW stencil, all fields are updated in the same simulated time moment, so they use the neighbor's values from the previous time moment.

Input data (the number in the field corresponds to the iteration number):

0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

The first step: update the black fields:

0	1	0	1	0
1	0	1	0	1
0	1	0	1	0

The second step: update the white fields:

2	1	2	1	2
1	2	1	2	1
2	1	2	1	2

We implement this method sequentially in `laplace-seq.cpp`.

Your goals are as follows:

1. Implement the first, working, distributed version by extending `laplace-par.cpp`. You can either use synchronous, point-to-point communication or asynchronous communication. Compute speed-ups.
2. Use `MPI_Allreduce` to test the termination condition of the main loop. Is the program significantly faster?
3. Use asynchronous, point-to-point communication to overlap the computation of the "black" fields with the communication of the "white" borders (analogously for the second phase). (alternatively, if your implementation already uses asynchronous communication, reimplement it with synchronous communication). Does this communication-computation overlap make your program faster?
4. (optional) As our algorithm has low computational intensity, one possible optimization is to communicate less frequently by computing redundant boundaries (communication-avoiding algorithm).

Date: 2024/04/26

Author: Krzysztof Rządca (based on Konrad Iwanicki's materials)
Created: 2024-04-26 Fri 10:10
Emacs 25.3.50.1 (Org mode 8.2.10)
Validate