CHALMERS
UNIVERSITY OF TECHNOLOGY

DAT400 Lab 5
Parallelizing a serial code using MPI

Contacts: Sonia Rani Gupta, Hari Abram, Miquel Pericas

September 28, 2023

After a good exercise on parallelizing a code through shared memory parallel programming model, you will now explore the potential of a distributed memory parallelization using MPI. Since you now have a good grip on the code and you know well about the most computationally expensive part of the application, this lab exercise is all about implementing and profiling an MPI version of the code.

1 Parallel neural network training

For distributed computing, every process should have its respective data buffer to work on. Rather than parallelizing the GEMM kernel, you are now expected to make the training itself parallel. Divide the work among the processes (BATCH_SIZE / number of processes). Every process computes the same function and updates the data, with the process with rank 0 acting as the main process which recollects and displays results. Have a look at figure 1 to see how this could be done.

To compile and run the mpi program, change the value of *CXX* variable to *mpicxx*. To run the program use the *make run_parallel* rule. You should only need to modify the nnetwork.cxx file.

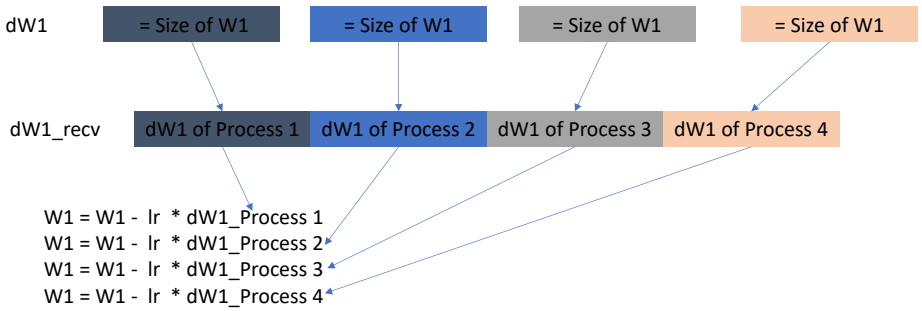

Figure 1: An example of dividing work among MPI processes

1.1 Hints

- Use *MPI_Comm_size() and MPI_Comm_rank* to get the total number of processes and rank of each process respectively.
- The strategy described above can be implemented in various ways using MPI collectives or MPI point-to-point calls. Figure 1 shows an example. It's up to you what you want to use, provided that the results are accurate.
- Change the weight update routine by accumulating updates from each MPI process.

1.2 Your task

- Parallelize the training of the neural network with MPI using the instructions above.
- Run the program with 1, 2 and 4 processes and report the speedup.
- Describe your implementation approach in the report. How you divide the work among processes? What changes you made in order to update weights? Which MPI collectives you used and why?

2 Task 2 - Parallel GEMM per process (Optional)

For a hybrid parallel program, run GEMM for loop with *omp parallel for*, which means every process does a parallel GEMM on its own data.

3 Task 3 - Profiling/tracing with Extrae and Paraver

We can use Extrae and Paraver to trace the execution of MPI applications (or hybrid, MPI+OpenMP, applications) and gain insights on the communication costs, the timing of the processes and the overall parallel efficiency of the application. This is particularly useful for large-scale runs, where the communication of thousands of processes significantly affects parallel performance. In this part, you will collect a trace of your application using Extrae and visualize it using Paraver. Before you start, you need to perform the following setup steps:

- Download and unpack `Lab5_profiling.tar` in your `$HOME` directory. Inside, you will find a release of Extrae, Paraver, and other tools, in a directory named `SW_packages`, a script named `SW_setup.sh`, and a configuration file for Extrae, named `extrae.xml`.
- Copy the directory `SW_packages` in the `/scratch` directory (*cp -r SW_packages /scratch*).
- Run `source SW_setup.sh` in your terminal to set up execution of Extrae and Paraver.

Please note that all items in the `/scratch` directory are temporary and deleted once you log out of the computers. Therefore, make sure you keep a copy of the `SW_packages` directory in your home directory.

To profile your code with Extrae, you will need to:

- Set the configuration file for Extrae:
  `export EXTRAE_CONFIG_FILE=$HOME/Lab5_profiling/extrae.xml`
- Preload the tracing library for MPI:
  `export LD_PRELOAD=/scratch/SW_packages/lib/libompitrace.so`

- Run your code without any modification. (Note: The process may end with a segmentation fault but .prv should be generated).
- Clear the `LD_PRELOAD` variable:
  `export LD_PRELOAD=`

After successful execution, Extrae will create several files. You can then run Paraver to visualize the trace, by executing `wxparaver &`. This command will open a graphical environment. To visualize the trace in Paraver:

- Load the trace (a file with a `.prv` extension).
- Visualize the trace (option "Single Timeline Window").
- Explore the different views (timelines and histogram) produced by Paraver by using the options under the menus "Hints/Useful", "Hints /MPI", "Hints/PAPI counters", and "Hints/OpenMP".

3.1 Your task

- Profile your code with Extrae and explore the capabilities of Paraver.
- Comment on the computation/communication ratio in your code, the communication functions and volume, the load balance between processes, and any bottlenecks you are able to identify through Paraver. How do you assess the overall efficiency of your code? Do you think it would seamlessly scale on high numbers of cores? Support your arguments with results from your analysis. Feel free to add interesting screenshots in your report.

Lab 5 Report Submission:

- Submit the report as a group of two. Write down the group number and CIDs of each partner (i.e., who participated in this lab).
- You will get a PASS only if all the tasks are properly addressed in the report, including your code implementation.