Assignment 2 : Performance Analysis using Caliper

CSCE-435 Fall 2025
Texas A&M University

September 27, 2025

1 Compile and execute the project

- Upload the starter code to your home directory after logging into the grace portal.
- Navigate to the directory that the files were uploaded to after logging into the grace portal.
- Allocate an interactive node using

  ```
      $ salloc --nodes=1 --ntasks=1 --cpus-per-task=4 --mem=8G --time=02:00:00 --
  partition=short
  ```

  Running `salloc` will allocate a node for you and you will receive the output

  ```
      salloc: Granted job allocation [job_id]
      salloc: Nodes [node_id] are ready for job
  ```

  where `node_id` is the node allocated to you. Also keep note of the `job_id` associated with this request as you will need it to free up the node later.
- `ssh` into the allocated node as

  ```
      $ ssh [node_id]
  ```

  Make sure your prompt is now `user_id@node_id`
- Initialize the CMake build using the command:

  ```
      $ . build.sh
  ```

- After your build is done, deallocate the node as

  ```
      $ scancel [job_id]
  ```

  where `job_id` refer to the job of allocating a node you received when running `salloc` in step 3.
- Run the batch file by running the following command, making sure to specify the matrix size (**m**) and the number of processors(**p**):

  ```
      $ sbatch mpi.grace_job <m> <p>
  ```

- Once the job is completed, you will be able to see the output file named according to the job id. The caliper performance measurements will be written to a `.cali` file(you will be able to find out if whether a recent job has been completed by going to grace dashboard -> jobs -> active jobs).

- **Whenever** a change is made to the code, allocate a node using `salloc` as described in step 3 through 6 and run make to re-build:

  ```
          $ make
  ```

  Make sure to free up the node after the build is done.

- On the grace dashboard, when starting up the jupyter notebook, for the module selection, choose **Python 3.10.8** and for the type of environment choose **Module load + Python virtualenv**. You can change the number of hours but leave everything else as default.

2 Assignment

Overview

The structure of this assignment is the same as the previous assignment, except this time we annotate the regions with **Caliper**.

NOTE: Do not copy the entire code from lab-1 to lab-2 as there are some extra dependencies.

Caliper

The caliper functions that we will use are

- `CALI_CXX_MARK_FUNCTION` : C++ macro to mark a function.
- `CALI_MARK_BEGIN` : Mark begin of a user defined code region
- `CALI_MARK_END` : Mark end of a user defined code region

The `CALI_CXX_MARK_FUNCTION` has already been implemented for you. Your assignment is to open and close the regions using `CALI_MARK_BEGIN` and `CALI_MARK_END`. The region names are already defined for you.If correctly implemented, the caliper config will automatically collect and aggregate the min, max, and average times into the caliper file.

Worker Processes

Use **Caliper** and `MPI_Reduce` to calculate the *minimum*, *maximum* and *average* runtime taken by the **receiving**, **calculation** and **sending** part of the worker processes. These regions are marked in the code using comments.

Master Process

Use **Caliper** to calculate the *minimum*, *maximum* and *average* runtime taken by the **whole computation**, **initialization** and **sending & receiving** part of the master process. These regions are marked in the code using comments.

Implementation

1. **Implement timers:** Use **Caliper** to create timers to measure over the regions marked in the code.(Read the comments in provided code)

2. **MPI_Reduce:** Use `MPI_Reduce` to calculate the min, max and average times for each of the worker timers.
   NOTE: Be careful if you use `MPI_COMM_WORLD` as your communicator, your values will be incorrect. This is because your master process will be implicitly included in your worker calculations. You need to either

   - Create a new MPI communicator
   - Initialize the variables you are reducing to in a way that avoids this problem

Inputs(For the 500 section only!!)

For both the **master** and **worker** process, vary the process and the matrix size as follows

- **Number of processes:** [2,4,8,16,32,64].
- **Matrix Size** [128 ×128, 1024 ×1024, 8192 ×8192].

Plot the runtimes observed for the regions marked previously using **Thicket**. You may plot all three times: max, min, avg in a single graph with different colors.

NOTE: Each node has 48 cores so you will have to change the number of nodes in order to get 64 processors.

NOTE: Only request the minimum number of nodes required to get the required number of processes. For example do if you want 32 processes, do not request 8 nodes with 4 tasks per node. Instead request 1 node with 32 tasks pernode.

Inputs(For the honors section(200) only!!!!)

For both the **master** and **worker** process, vary the process and the matrix size as follows

- **Number of processes:** [2,4,8,16,32,64,128].
- **Matrix Size** [128 ×128, 1024 ×1024, 8192 ×8192].

Plot the runtimes observed for the regions marked previously using **Thicket**. You may plot all three times: max, min, avg in a single graph with different colors.

NOTE: Each node has 48 cores so you will have to change the number of nodes in order to get 64 processors.

NOTE: Only request the minimum number of nodes required to get the required number of processes. For example do if you want 32 processes, do not request 8 nodes with 4 tasks per node. Instead request 1 node with 32 tasks per node.

Analysis

Analyze the plots and explain the observations. Write down the explanations for the trends observed for each plot.

Grading

Exercise	**Points**
Worker process time plots(10 points for each matrix size)	30
Master process time plots(10 points for each matrix size)	30
Observations	15
Correctness of `MPI_Reduce` code changes	10
Correctness of the annotated Caliper regions	5
Used thicket to generate plots	10

3 Submission

Submit a `.zip`file on Canvas containing

- A **pdf** with the plots and your observations
- `mpi_mm.cpp` file with your code changes
- The `.cali` files generated