Assignment 4

DSE 512

Assigned 2022-4-19 — Due 2022-4-30 11:59pm

1. Basics

1. (20%) Briefly explain in your own words the difference between *task parallelism* and *data parallelism*.
2. (20%) Briefly explain in your own words the advantages and disadvantages of using

   - Fork
   - MPI
   - Mapreduce

   for parallelism. Think about what might cause one to be a better choice than another for a particular task. You don't have to agree with my personal positions, but what you say must be *defensible*. Examples of non-defensible positions are "MPI has no value whatsoever" and "you should only ever use MPI."

2. Task Parallelism

In R or Python (your choice), you are given a function `fit_model()` which takes as inputs a data matrix `X` and a response `y`. The model is likelihood-based and emits an AIC score. You are tasked with finding the best possible model by AIC (a lower number is better). You have no idea a priori which predictor variables (columns) are useful in modeling the response, so you plan to try all possible models. In addition to the predictor columns, the model has a constant term which should never be omitted.

1. (5%) If you are given a full input matrix `X` with

   a. 1
   b. 5
   c. 10
   d. 20
   e. 100

   columns, how many possible models do you need to fit in each case? You can use scientific notation rounding to a few significant digits if you wish.
2. (5%) Assuming it takes 1 second to fit each model on average, how long would you expect the run time to be for each of the cases above?
3. (15%) Give the code to fit all of these models in parallel using fork. For simplicity, you may assume the number of cores `p` is fixed at 8 throughout. (Hint: I recommend the rows-as-inputs strategy mentioned in class using R's `expand.grid()` or similar).
4. (5%) Assuming you get perfect linear scaling on 8 cores, what are the run times reduced to in calculation 2.2 above?

3. Data Parallelism

1. (15%) Poisson Regression is a generalized linear model (GLM) used to model dependent variables which are counts. It was originally used to model the number of expected mule kicks to the head for Prussian

soldiers. Its cost function is:

$$J(X, y) = \min_{\theta \in \mathbb{R}^n} \sum_{i=1}^{m} \left(y_i \eta_i - L^{-1}(\eta_i) \right)$$

where $\eta = X\theta$.

Implement a distributed Poisson regression fitter in R using pbdMPI. Assume the inputs are a data matrix X distributed by rows with global dimension $m \times n$, and a response vector y distributed identically to X with global dimension $m \times 1$. Each SPMD process has local pieces `X_local` and `y_local` which the same number of rows on each process, and n and 1 columns (respectively) on all processes. In R, you can access the inverse link function L^{-1} via `linkinv_poisson = poisson(log)$linkinv`.

2. (15%) Assume the regression fitter above takes 100 optimization steps to complete. If you have p total MPI processes, describe the total amount of communication (in terms of MPI operations).

OPTIONAL Challenge Problem

1. (0 points) Typically in parallel computing, you start with a working serial implementation and fail (at first) to get a working parallel implementation. The C++ code snippet below computes the Pearson correlation of 2 vectors correctly when computing in parallel on a **small** test case, but *incorrectly* in serial. Can you figure out why?

```cpp
template <typename REAL>
REAL cor(const cpuvec<REAL> &x, const cpuvec<REAL> &y)
{
  const len_t n = x.size();
  if (n != y.size())
    throw std::runtime_error("non-conformal arguments");

  const REAL *x_d = x.data_ptr();
  const REAL *y_d = y.data_ptr();

  const REAL meanx = x.sum() / n;
  const REAL meany = y.sum() / n;
  REAL cp = 0, normx = 0, normy = 0;

  #pragma omp parallel for reduction(+: cp, normx, normy)
  for (len_t i=0; i<n; i++)
  {
    const REAL xi_mm = x_d[i] - meanx;
    const REAL yi_mm = y_d[i] - meany;

    cp = xi_mm * yi_mm;
    normx += xi_mm * xi_mm;
    normy += yi_mm * yi_mm;
  }

  return cp / sqrt(normx * normy);
}
```