P. Hadjidoukas, E. Dermatas, E. Gallopoulos

Set 1 - MPI and OpenMP

Issued: November 21, 2024

Question 1: Hybrid Programming Model and Parallel I/O

a) Implement the MPI_Exscan call of the MPI programming model using point-to-point communications (send/recv). Assume that you have *P* processes and that the implementation of the MPI_Exscan will work for integer values, i.e. variables of type int. You can name the new function MPI_Exscan _pt2pt.

b) Extend the previous implementation to be suitable for the hybrid model (MPI + OpenMP). An application runs with *P* processes and *T* threads in each process. Each thread in the application executes the same code, applying the SPMD (Single Program Multiple Data) programming model. All threads call the new function (MPI_Exscan_omp) at the same time, with each thread using its own integer value. The implementation of the function exploits both programming models (MPI, OpenMP).

c) By extending and utilizing the code of the previous question, implement the following scenario: Each thread of the hybrid application binds and initializes, with a specific value of a private randon seed, a real number matrix of size NxNxN. Then, using MPI I/O the application will write the matrices to a binary file. The writing phase will be triggered simultaneously by all threads in the app. To calculate the correct write location of each matrix, use the function that you implemented in the previous question, even if the size of the data is the same for each thread. Also, you will have to programmatically confirm, in the same application or in a different one (parallel or serial), that the data in the binary file is correct. This assumes that the application reads the data from the file and compares it with the initials (see initialization of each matrix with specific random values).

d) Repeat the previous question using a data compressor. Before the writing process, each thread will compress its three-dimensional matrix, and the result of the compression, which is likely to be a different size for each matrix, will be written to the binary file.

   Matrix compression can be performed with a standard data compression library, such as ZLIB and ZSTD, or with a scientific data compression library, such as ZFP and SZ.

Question 2: Parallel Parametric Search in Machine Learning

Import

Parametric search in machine learning is a process that aims to improve the performance of a model by optimizing its parameters. These parameters are commonly known as hyperparameters and directly affect the model's performance. There are several methods for implementing parametric search, such as grid search, random search, and more advanced techniques such as Bayesian optimization.

Grid search is an exhaustive process that tests all potential combinations of hyperparameters in a predefined grid. For example, if we have two hyperparameters with three and four possible values each, the grid search will try all 3 x 4 = 12 combinations.

Random search selects random combinations of hyperparameters within a predefined space. Unlike grid search, random search does not search for all possible combinations, but for a predetermined amount of them. This can be more efficient in large parameter space. Bayesian optimization is a more sophisticated technique that uses a statistical model to predict which hyperparameter combinations are most likely to perform best. It builds on past results and adapts dynamically as the process progresses.

```
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import ParameterGrid
from sklearn.datasets import make_classification
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split

X, y = make_classification(n_samples =10000 , random_state =42 , n_features =2 ,
     n_informative =2 , n_redundant =0 , class_sep =0.8 )

X_train , X_test , y_train , y_test = train_test_split(X, y , test_size =0.33 ,
     random_state =42)


params = [{'mlp_layer 1': [16 , 32],
           'mlp_layer 2': [16 , 32],
           'mlp_layer 3': [16 , 32]}]

pg = ParameterGrid(params)

results = []
for i, p in enumerate(pg):
     print(p)
     l1 = p['mlp_layer 1']
     l2 = p['mlp_layer 2']
     l3 = p['mlp_layer 3']
     m = MLPClassifier(hidden_layer_sizes=(l1,l2, l3))
     m.fit(X_train, y_train)
     y_pred = m.predict(X_test)
     ac = accuracy_score(y_pred, y_test)
     print(i, ac)
     results.append((i,p, ac))

for r in results:
     print(r)
```

Application

In order to perform a parametric search, the hyperparameters to be optimized must first be determined. For example, in a Random Forest Classier, the hyperparameters may include the number of trees (n_estimators), the maximum depth of trees (max_depth), and the fraction of characteristics used to form each tree (max_features). Similarly, in an MLP Classifier, i.e. in a neural network, typical hyperparameters are the number of levels, the number of neurons per level, the weight optimization algorithm and the learning rate.

Parallel Parametric Search

The main objective of the exercise is to parallelize parametric search to machine learning problems, such as the one shown in the Python serial code above. The processing of each parameter group can be performed independently, so the search time can be reduced by utilizing the available task-level parallelism. The above exploitation in Python, in a computational node, must be done with the following methods

- multiprocessing pool
- mpi futures
- by implementing the master-worker model using MPI, similar to the one we saw in the lecture of the course.

Measurements

Once you have completed the parallel version of the application and made sure that it is working properly, you will measure its performance to show the correct utilization of the available multiprocessor hardware. You should measure the execution time of the application, for various numbers of processing elements (processes) in order to calculate the speedup of the application. You can customize the problem data in the original serial code (e.g. data size, number of layers and neurons in the network) according to your needs.

Question 3: OpenMP Tasking

In the following code, the initialize function sets values in a vector A of dimension N using the thread-safe and time-consuming work function, implemented somewhere else.

```
extern double work(int i);
void initialize(double *A, int N)
{
    #pragma omp parallel for schedule(dynamic, 2)
    for (int i = 0; i < N; i++){
        A[i] = work(i);
    }
}
```

a) Explain how loop repeats are distributed in the threads.

b) Write an equivalent implementation of the parallel code based on the OpenMP task model.

Deliverables

Code (in zip file) and report (in pdf file) with analysis of the problem, explanation of the solution, and presentation of the results.