Powerset reduction with MPI

--Vivek kuchibhotla
Application being used

• Solving Bayesian inference
• Runtime of $O(2^n \times n^2)$
Problem with layer wise traversal

- Varying sizes of layers
- Job scheduling required
- Movement of data is not consistent
Why is powerset traversal useful

- Power sets can be represented as hypercubes
- Hypercubes scale very well with increasing processors
- Data dependency will always be with processors that are connected.
- Needs an initial warm-up phase and can go through with all the processors.
Approaches

1. Use MPI_Reduce, MPI_Scatter calls recursively
2. Add a new MPI call, MPI_Hypercube_Reduce.
Organizing processors and setup

• Need to initially organize the processors in Hyper cube formation with **MPI_Cart_create**
• Load the data into corresponding processors.
• Create the warm-up loop
• Finished work until here.
Continuing the work

• Work on the scatter, reduce calls to move data around until we reach the end of the powerset tree.
• If time permits implement a MPI call that does the same.
• Finished creating the setup and compute with communication built and compute performed.
Create hypercube struct

```c
MPI_Comm nthCube;
int nDim = log2(size);
int processPerDim[nDim];
//std::cout << "ndim is " << nDim << std::endl;
int period[nDim];
for (int i = 0; i < nDim; ++i) {
    processPerDim[i] = 2;
    period[i] = 1;
}

MPI_Cart_create(MPI_COMM_WORLD, nDim, processPerDim, period, true, &nthCube);

int rankInDim;
MPI_Comm_rank(nthCube, &rankInDim);

int rank_source, rank_desta, rank_destb, rank_destc, rank_destd;
int rank_adjacent[nDim];
for(int i=0;i<nDim;++i){
    MPI_Cart_shift(nthcube, i,1,&rank_source, &rank_adjacent[i]);
}  
```
The Warmup

```c++
for(int j=0; j<pow(2, power_set_size)/size; j++){
    for (int i = 0; i < nDim; i++) {
        vector<int> processors_to_process = generate_binary_strings(nDim, i);
        for(int j=0;j<processors_to_process.size(); j++){
            if(processors_to_process[j] == rankInDim){
                /*
                * Wait on receiving data from previous processors
                */
                vector<int> to_recv = get_numbers_with_flipped_one(rankInDim, nDim);
                if(to_recv.size() > 0){
                    int recv_data[to_recv.size()];
                    int recv_data[to_recv.size()];
                    std::vector<MPI_Request> requests(to_recv.size());
                    for(int k = 0; k < to_recv.size(); k++){
                        //std::cout << "recv from " << to_recv[k] << std::endl;
                        MPI_Recv(recv_data[k], 1, MPI_INT, /*source*/ to_recv[k], /*tag*/ 0, nthCube, &requests[k]);
                    }
                    completed_index = MPI_Waitany(to_recv.size(), requests.data(), &completed_index, MPI_STATUS_IGNORE);
                }
                Process();
                vector<int> to_send = get_numbers_with_flipped_zero(rankInDim, nDim);
                for(int k=0;k<to_send.size();k++){
                    int aj = 10;
                    //std::cout << "sending to " << to_send[k] << std::endl;
                    MPI_Send(&aj, 1, MPI_INT, to_send[k], 0, nthCube);
                }
            }
        }
    }
}
```
Testing and performance

• Ran a simple test with a single processor on a test data set Alarm.
• Created synthetic data to simulate complex computation on top of a power set traversal problem.
• Will also primarily investigate strong and weak scaling.
Scaling for now
Future Work

• Get real world data
• Understand weak vs strong scaling
• Dynamic data movement
References

• https://www.bnlearn.com/bnrepository/discrete-small.html
• https://www.sciencedirect.com/science/article/pii/S0743731513000622