APPENDIX

In this section, we show an example about graph algorithm implementation using our APIs. In particular, we compare the implementation difference between conventional distributed systems and our proposed middlewares, denoted by CPU- and GPU-based implementation, respectively[3].

The codes for CPU- and GPU-based implementation are shown in Sample Codes 1 and 2. The differs of both side of codes are highlighted in different colors, red for CPU-based code and green for GPU-based code. From the two samples, we can see the implementations are quite similar, and only lines 17-28 are different. For CPU-based implementation, this part is for invoking CPU computation. For GPU-implementation, this part is for transferring data to GPU and executing corresponding kernel functions. This way, the distributed graph algorithms for conventional distributed systems can be replanted to our middleware with small changes. More, algorithm engineers can access kernel function implementation for further optimization.

[3]For ease of presentation, we show MSGApply() functions for SSSP-BF. The implementation of other functions, such as MSGGen() and MSGMerge(), and other algorithms, such as PageRank, are similar, and are omitted due to page limits. Please refer to our open source project for more details on the implementation.

```cpp
template <typename VertexValueType, typename MessageValueType
    >
int BellmanFord<VertexValueType, MessageValueType>::MSGApply(
    Graph<VertexValueType> &g, const std::vector<int> &
    initVSet, std::set<int> &activeVertice, const MessageSet
    <MessageValueType> &mSet)
{
    //******************Init******************
    activeVertice.clear();
    if(g.vCount <= 0) return 0;
    MessageValueType *mValues = new MessageValueType [g.
        vCount * this->numOfInitV];
    for(int i = 0; i < g.vCount * this->numOfInitV; i++)
        mValues[i] = (MessageValueType)INVALID_MASSAGE;
    for(int i = 0; i < mSet.mSet.size(); i++){
        auto &mv = mValues[mSet.mSet.at(i).dst * this->
            numOfInitV + g.vList.at(mSet.mSet.at(i).src).
            initVIndex];
        if(mv > mSet.mSet.at(i).value)
            mv = mSet.mSet.at(i).value;
    }
    //****************Init End*****************
    //******************CPU******************
    for(int i = 0; i < g.vCount; i++)
        vSet[i].isActive = false;
    for(int i = 0; i < g.vCount *
        numOfInitV; i++){
        if(g.verticesValue[i] > (VertexValueType)mValues[i]){
            g.verticesValue[i] = (VertexValueType)mValues[i];
            if(!g.vList[i / numOfInitV].
                isActive)
                g.vList[i / numOfInitV].
                    isActive = true;
        }
    }
    //*****************CPU End*****************
    for(int i = 0; i < g.vCount; i++){
        if(g.vList.at(i).isActive)
            activeVertice.insert(i);
    }
    free(mValues);
    return activeVertice.size();
}
```

Sample Code 1: CPU Implementation

```cpp
template <typename VertexValueType, typename MessageValueType
    >
int BellmanFord<VertexValueType, MessageValueType>::MSGApply(
    Graph<VertexValueType> &g, const std::vector<int> &
    initVSet, std::set<int> &activeVertice, const MessageSet
    <MessageValueType> &mSet)
{
    //******************Init******************
    activeVertice.clear();
    if(g.vCount <= 0) return 0;
    MessageValueType *mValues = new MessageValueType [g.
        vCount * this->numOfInitV];
    for(int i = 0; i < g.vCount * this->numOfInitV; i++)
        mValues[i] = (MessageValueType)INVALID_MASSAGE;
    for(int i = 0; i < mSet.mSet.size(); i++){
        auto &mv = mValues[mSet.mSet.at(i).dst * this->
            numOfInitV + g.vList.at(mSet.mSet.at(i).src).
            initVIndex];
        if(mv > mSet.mSet.at(i).value)
            mv = mSet.mSet.at(i).value;
    }
    //****************Init End*****************
    //******************GPU******************
    GPU_Mem_Copy_In(g.vList, vSet_d);
    for(auto offset = 0; mSet.mSet.length() >
        offset; offset += threadNum){
        int localMNum = mSet.mSet.length() -
            offset > threadNum ? threadNum :
            mSet.mSet.length() - offset;
        GPU_Memory_Copy_In(mSet.mSet[offset],
            msgSet_d, localMNum);
        MSGApply_kernel<<<1, threadNum>>>(mSet.
            mSet.length() - offset);
    }
    GPU_Mem_Copy_Out(vSet_d, g.vList);
    //*****************GPU End*****************
    for(int i = 0; i < g.vCount; i++){
        if(g.vList.at(i).isActive)
            activeVertice.insert(i);
    }
    free(mValues);
    return activeVertice.size();
}
```

Sample Code 2: GPU Implementation