CSE 291: Differentiable Programming
Project Proposal

Jash Gautam Makhija (A59025270)
Animesh Kumar (A59023817)

June 13, 2024

Distributed AD: Add OpenMPI support to loma so that it can be distributed to multiple machines.

Abstract

The primary goal of our project is to enhance loma, by integrating OpenMPI (Open Message Passing Interface)[1] support. This integration will enable loma to distribute and parallelize the compilation process across multiple machines, significantly improving compilation speed and efficiency for executing multiple test cases on different machines simultaneously and also to help split a very large data-set.

Implementation

Open MPI (Open Message Passing Interface) is an open-source implementation of the Message Passing Interface standard, which is widely used for programming parallel computers. Open MPI provides the essential communication framework to enable efficient data exchange and synchronization between compute nodes during distributed training.

The integration of OpenMPI in Loma will involve the following enhancements:

1. Add backward compatibility to run existing test cases with OpenMPI framework. This involves changing the loma compiler codebase to support *mpicc* compiler and *mpirun/mpiexec* to execute serial and parallel jobs in Open MPI.
2. Support and add new test cases to run loma code snippet on multiple nodes. This will involve creating custom decorator and generate OpenMPI code(e.g MPI_Init, MPI_Comm_rank, MPI_Comm_size, $MPI\_Finalize$) to support loma compilation to generate OpenMPI code. This will allow loma to run independently across machines/nodes.
3. Add communication between nodes to support distributed computations. This will involve adding OpenMPI's message passing interface(MPI_Send, MPI_Recv) to the loma compiler. This will allow us to perform task like distributed gradient descent calculation where local gradient are calculated on different nodes and finally gathered on the master node for every batch.

Listing 1: Open MPI generate code example

```c
// Additional import needed for MPI
#include <mpi.h>
#include <stdio.h>
#include <math.h>


// Generated C code from loma for identity function
typedef struct {
    float val;
    float dval;
} _dfloat;

float identity(float x);

_dfloat d_identity(_dfloat x);

_dfloat make__dfloat(float val, float dval);

float identity(float x) {
    return x;
}

_dfloat d_identity(_dfloat x) {
    return make__dfloat((x).val,(x).dval);
}

_dfloat make__dfloat(float val, float dval) {
    _dfloat ret;
    ret.val = 0;
    ret.dval = 0;
    (ret).val = val;
    (ret).dval = dval;
    return ret;
}

// Using a main function for now. Ideally this should be called in python by the lib file generated by loma.
int main(int argc, char** argv) {

    // Initialize MPI interface
    MPI_Init(&argc, &argv);

    // Rank corresponds to number of nodes
    int rank;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    //  General C code
    float x = rank;
    float iden = identity(rank);
    printf("Rank %d. identity Output - %f\n", rank, iden);

    // Loma Code
    _dfloat d_x = {rank, 0.0};
    _dfloat d_iden = d_identity(d_x);
```

```
    printf("Rank %d. d_identity Output - val: %f dval: %f\n", rank, d_iden.val,
    d_iden.dval);

    // Terminates MPI execution environment
    MPI_Finalize();
    return 0;
}
```

Listing 2: Open MPI generate code output

```
Rank 3. identity Output - 3.000000
Rank 3. d_identity Output - val: 3.000000 dval: 0.000000
Rank 1. identity Output - 1.000000
Rank 1. d_identity Output - val: 1.000000 dval: 0.000000
Rank 2. identity Output - 2.000000
Rank 2. d_identity Output - val: 2.000000 dval: 0.000000
Rank 0. identity Output - 0.000000
Rank 0. d_identity Output - val: 0.000000 dval: 0.000000
```

Scope and Timeline

Checkpoint 1: Interface and Compilation Setup

- **Objective**: Establish the foundational infrastructure for distributed compilation and testing.
- This feature will enable the parallel execution of multiple test cases across different machines. We plan to implement functionality that allows distinct test cases of a single program to be distributed and executed concurrently on separate machines. For instance, for a forward differentiation module with 32 test cases, we will partition these test cases and allocate subsets to different nodes, thereby parallelizing the testing process across the distributed system.

Final Submission: Synchronization and Communication

- **Objective**: Implement robust synchronization and communication mechanisms to handle large-scale distributed execution and result aggregation.
- The capability to execute a program with large input data concurrently across multiple machines. For example, running computationally intensive algorithms such as gradient descent in parallel, followed by an all-gather operation that consolidates and aggregates outputs from all nodes at the end, thereby significantly reducing overall execution time.
- Compile and Execute differentiable Ray Tracing algorithm in parallel with a set of inputs.

References

[1] Open MPI. https://github.com/open-mpi/ompi. 2004.