NNPred OpenFOAM Module Document

Weishuo Liu[a]

[a]*School of Energy and Power Engineering, Beihang University, 37 Xueyuan Road, Haidian District, Beijing, 100191, China*

A+B Model and Code Example

A simple **A + B = C** model is presented to show the deployment process. This model computes the sum of two 2D arrays (i.e., **A** and **B**) and holds the result in the output node, **C**. The I/O nodes' dimension sizes are $n \times 2$, with n being the number of data instances, and their data type is specified as the single-precision floating point. Figure. 1 shows a minimal application case of the model deployment and interaction with external arrays (i.e., `Array_A`, `Array_B` and `Array_C`). In this case, `Array_B` has different data type from the node's definition (`int` vs `float`), and `Array_C` differs to node, `output_c`, in both data type (`double` vs. `float`) and memory layout (3×2 vs. 3×2).

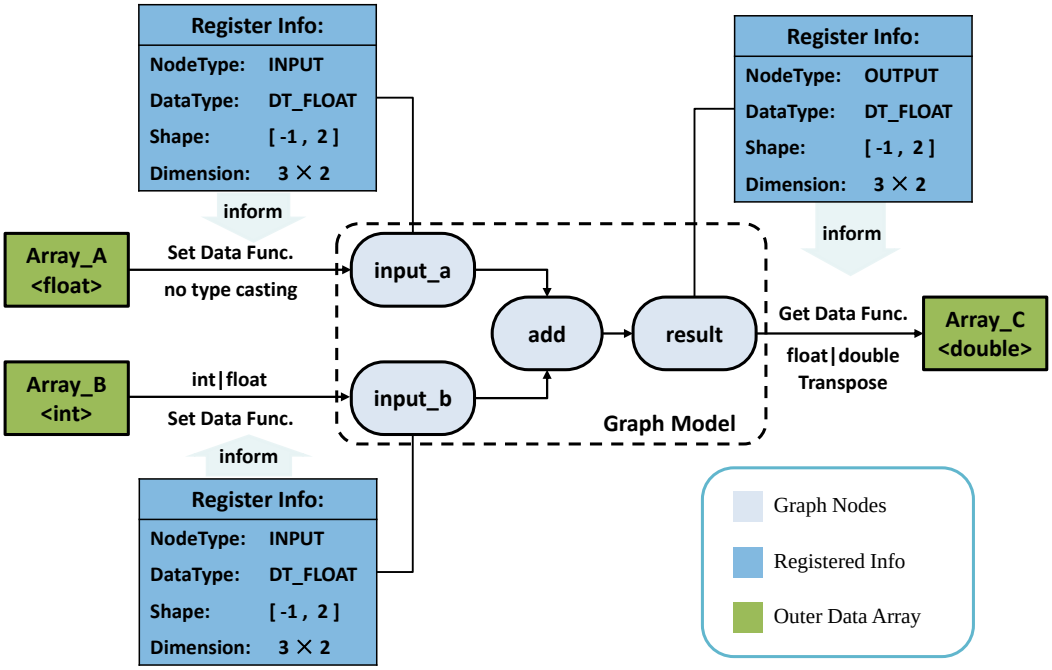

Figure 1: Interaction relation within NNPred elements and with outer memory spaces during a prediction process.

The whole process can be divided into two major parts: initialization and prediction. The initialization consists of loading the model, registering the input/output nodes, and setting the number of data instances, whereas the prediction part includes setting the input data, running predictions, and extracting the result. The OpenFOAM module further encapsulated the steps with just only the constructor function and prediction function. All the available setting options are specified in an OpenFOAM dictionary:

Email address: `liuweishuo@buaa.edu.cn` (Weishuo Liu)

```
model  // Dictionary name, referenced by the constructor function
{
    readFromPB                yes;               // yes/no for PB/SavedModel
    modelDirectory            "models/simple_graph_tf2.pb";
//  tags                      "serve";           // only needed for SavedModel
    copyMethod                Eigen;             // Eigen/Safe for SIMD/element-wise copy
    layout                    ColMajor;          // RowMajor or ColMajor
    inputs                    ("input_a" "input_b");    // use space as separator
    outputs                   ("result");               // use space as separator
}
```

Listing 1: OpenFOAM dictionary to initialize predictor library

The constructor function and prediction function are shown at line 31 and 34 in the following codes (Listing. 2). But calling the prediction is slightly different since OpenFOAM defines its own fundamental data structures. Two-level container (`Foam::List`) of the `Foam::scalarField` references need to be assembled. The outer-level `Foam::List` corresponds to the number of nodes, whereas the inner-level capacity corresponds to the known dimension of the node shape. For example in the **A + B** model, two independent nodes are registered as input nodes, so that the outer list capacity of the `multi_inputs` is two (line 36). The dimensions of the inner list (e.g., `multi_inputs[0]` in line 40) is two because the tensor shape of the input/output node is `[-1, 2]`. The exact code example can be found below:

```
#include "scalarField.H"      // Header from OpenFOAM
#include "TF_OF_Predictor.H"  // Header for TF_Predictor

int main() {

    // External OF fields ...

    // create input field A (columns):
    scalarField input_field_a_col1(3, scalar(0.0));
    input_field_a_col1[0] = 0.0;
    input_field_a_col1[1] = 2.2;
    input_field_a_col1[2] = 4.4;
    scalarField input_field_a_col2(3, scalar(0.0));
    input_field_a_col2[0] = 1.1;
    input_field_a_col2[1] = 3.3;
    input_field_a_col2[2] = 5.5;


    // create input field B (columns):
    scalarField input_field_b_col1(3, scalar(0.0));
    input_field_b_col1[0] = 5.0;
    input_field_b_col1[1] = 3.0;
    input_field_b_col1[2] = 1.0;
    scalarField input_field_b_col2(3, scalar(0.0));
    input_field_b_col2[0] = 4.0;
    input_field_b_col2[1] = 2.0;
    input_field_b_col2[2] = 0.0;

    // create output field C (columns)
    scalarField output_field_c_col1(3, scalar(0.0)); // output field for pb model column1
    scalarField output_field_c_col2(3, scalar(0.0)); // output field for pb model column2


    // Create the entry list for OF fields:
    // PB format model:
    Foam::List<Foam::List<scalarField*>> multi_inputs(2);
    Foam::List<Foam::List<scalarField*>> multi_outputs(1);

    // For input_a:
    multi_inputs[0].append(&input_field_a_col1);
    multi_inputs[0].append(&input_field_a_col2);

    // For input_b:
    multi_inputs[1].append(&input_field_b_col1);
```

```
    multi_inputs[1].append(&input_field_b_col2);

    // For output_c:
    multi_outputs[0].append(&output_field_c_col1);
    multi_outputs[0].append(&output_field_c_col2);


    Info << "Creating TF_OF_Predictor\n" << endl;
    TF_OF_Predictor pd = TF_OF_Predictor("constant/TF_Predictor_Dict", "model_pb_2D");


    Info << "Running the models \n" << endl;
    pd.predict(multi_inputs, multi_outputs);

    Info << "Print the running results\n" << endl;
    Info << "Results of column 1: \n"
         << input_field_a_col1 << " + " << input_field_b_col1 << " = " <<
    output_field_c_col1 << endl;

    Info << "Results of column 2: \n"
         << input_field_a_col2 << " + " << input_field_b_col2 << " = " <<
    output_field_c_col2 << endl;

    return 0;
}
```

Listing 2: Minimal example to run A + B model in OpenFOAM

Entire API list for OpenFOAM Module

- **OF01: Model Initialization:**
  - `TF_OF_Predictor()`

    To create the `TF_OF_Predictor` object from the OpenFOAM dictionary stored in default directory. The default file path of the dictionary is `constant/TF_Predictor_Dict`, and the settings are under the default dictionary name, `model`.
  - `TF_OF_Predictor(std::string Model_name)`
    - `Model_name` – the name of the dictionary the model will be launched under its setting options.

    To create the `TF_OF_Predictor` object from the OpenFOAM dictionary stored in default directory. The default file path of the dictionary is `constant/TF_Predictor_Dict`, and the settings are under the dictionary name, `Model_name`.
  - `TF_OF_Predictor(std::string Dict_dir, std::string Model_name)`
    - `Dict_dir` – the OpenFOAM dictionary file holding the dictionaries of the settings of the `TF_OF_Predictor`.
    - `Model_name` – the name of the dictionary the model will be launched under its setting options.

    To create the `TF_OF_Predictor` object from the OpenFOAM dictionary stored in `Dict_dir`, and the settings are under the dictionary name, `Model_name`.
- **OF02: Model Prediction:**
  - `predict(Foam::List<Foam::scalarField*>& inputs, Foam::List<Foam::scalarField*>& outputs)`
    - `inputs` – the reference container holding the references of input `scalarField` for a single-input-output model.
    - `outputs` – the reference container holding the references of output `scalarField` for a single-input-output model.

    To perform prediction of a single-input-output model, which means the number of the input nodes and output nodes are both **one**. In this case, the capacity of the reference container corresponds to the known dimension of the node shape. For example, if the node shape is `[-1, 2, 3]`, then the reference container needs to have $2 \times 3 = 6$ references of `scalarField`.
  - `predict(Foam::List<Foam::List<Foam::scalarField*>> & multi_inputs, Foam::List<Foam::List<Foam::scalarField*>> & multi_outputs)`

* `inputs` – the two-level reference container holding the references of input `scalarField` for a multiple-input-output model.
* `outputs` – the two-level reference container holding the references of output `scalarField` for a multiple-input-output model.

To perform prediction of a multiple-input-output model, which means model has more than one input nodes or output nodes. In this case, the outer-level capacity of the two-level reference container equals to the number of nodes (e.g., the `multi_inputs` in the example), and the inner-level capacity corresponds to the known dimension of the node shape. For example, if the node shape is `[-1, 2, 3]`, then the reference container needs to have $2 \times 3 = 6$ references of `scalarField`.

- `predict(Foam::List<Foam::volScalarField*>& inputs, Foam::List<Foam::volScalarField*>& outputs)`
  * `inputs` – the reference container holding the references of input `volScalarField` for a single-input-output model.
  * `outputs` – the reference container holding the references of output `volScalarField` for a single-input-output model.

  To perform prediction of a single-input-output model, which means the number of the input nodes and output nodes are both **one**. In this case, the capacity of the reference container corresponds to the known dimension of the node shape. For example, if the node shape is `[-1, 2, 3]`, then the reference container needs to have $2 \times 3 = 6$ references of `volScalarField`.
- `predict(Foam::List<Foam::List<Foam::volScalarField*>> & multi_inputs, Foam::List<Foam::List<Foam::volScalarField*>> & multi_outputs)`
  * `inputs` – the two-level reference container holding the references of input `volScalarField` for a multiple-input-output model.
  * `outputs` – the two-level reference container holding the references of output `volScalarField` for a multiple-input-output model.

  To perform prediction of a multiple-input-output model, which means model has more than one input nodes or output nodes. In this case, the outer-level capacity of the two-level reference container equals to the number of nodes (e.g., the `multi_inputs` in the example), and the inner-level capacity corresponds to the known dimension of the node shape. For example, if the node shape is `[-1, 2, 3]`, then the reference container needs to have $2 \times 3 = 6$ references of `volScalarField`.