Design of Simulator

2020/1/13

# System Overview:

DAG parser

Strategy Config

Raw DAG

Extended DAG 2

Device Assign

Module

Device Assign Config

Simulation Execution module

Adapter

Profiling DB

Execution Time Estimator

Parallel Strategy

Module

Execution DAG: The DAG output by Device Assign Module, contains all execution information, including node and device info.

Execution Time Estimator module: This module will estimate the execution time of each node in the Execution DAG according to the offline profiling database, then write the execution time into Execution DAG.

Adapter module: This module parse Execution DAG file to Simulation DAG

Simulation DAG: The node in execution DAG is saved in extended Node class. This class only contains attributes used for simulator and is independent from framework.

Simulator Execution module: The simulator will read the Simulation DAG and simulate the execution. It will only use attributes belong to basic Node class. This will make it independent from specific framework.

# Simulation DAG Data Structure Define:

## Class NodeMetadata():

This class stores all static metadata of a node, which will not change during execution.

Attributes:

|  |  |  |
| --- | --- | --- |
| Name | Data Type | Description |
| Index | Int | ID of node |
| Op | String | Operation |
| Name | String | Name |
| Device | String | Name of device to run this node |
| Execution time | Float | Estimated execution time, in microsecond |
| Input\_ids | List(int) | ID of all input data nodes |
| Dependency\_ids | List(int) | ID of all nodes that only have control dependency |
| Successor\_ids | List(int) | ID of all nodes depends on this node |

## Class Node():

This class represent a running node, store dynamic status of a node, which will change during execution.

Attributes:

|  |  |  |
| --- | --- | --- |
| Name | Data Type | Description |
| Metadata | NodeMetadata ref | The metadata object of this node |
| Status | Enum | Status of the node ‘wait’, ‘pending’, ‘done’ |
| Dependency\_cnt | Int | Num of dependency that is not done now |
| Device | Device ref | The device object that this node run on. |
| Input\_nodes | List(Node ref) | The Node object of all input data nodes |
| Dependency\_nodes | List(Node ref) | The Node object of all dependency nodes |
| Successor\_nodes | List(Node ref) | The Node object that depends on this node |

## Class Device():

This class represent a running device, store dynamic status of a device.

Attributes:

|  |  |  |
| --- | --- | --- |
| Name | Data Type | Description |
| name | String | Name of the device |
| Node\_queue | List(Node) | All nodes that have been submitted to this device |
| Queue\_head | Int | The head of node queue, indicate the current running node |
| Head\_end\_time | float | The end time of the current running node |

Methods:

|  |  |
| --- | --- |
| Name | Description |
| Is\_idle | Return True/False, whether the device is idle |
| Head\_node | Return the ref of current running node. |
| Enqueue\_node | Enqueue a node into the device. |
| Dequeue\_node | Dequeue the first node in the device. Reset head\_end\_time |

## Class GraphDefine():

This class stores all static metadata of a graph, which will not change during execution.

|  |  |  |
| --- | --- | --- |
| Name | Data Type | Description |
| Nodes\_metadata | List(NodeMetadata ref) | All node’s metadata in the graph |
| Devices\_name | List(string) | All devices name |

## Class Graph():

This class represent a running graph, store dynamic status of a graph.

Attributes:

|  |  |  |
| --- | --- | --- |
| Name | Data Type | Description |
| Graph\_define | GraphDefine  ref | The static metadata of the graph |
| Nodes | List(Node) | All dynamic nodes object |
| Devices | List(Device) | All dynamic devices object |
| Ready\_list | List(Node) | Store nodes that all dependency nodes are done, which is ready to start |
| Ready\_list\_head | Int | The head of ready\_list |
| All\_nodes\_done | Boolean | Flag that whether all nodes are done. |
| Enqueue\_sequence | List(int, float) | The enqueue sequence of all nodes. Each element is a pair of <node\_id, enqueue\_time> |
| Time\_now | float | Current timestamp in simulation |

Methods:

|  |  |
| --- | --- |
| Name | Description |
| \_\_init\_\_ | Initialize the graph. Create all node and device objects according to the graph\_define. |
| Start\_node | Start to execute a node. Enqueue it to device and mark status as ‘pending’. |
| Find\_earliest\_complete\_node | Find the node that will complete earliest. |
| Synchronize\_a\_pending\_node | Wait until the earliest node is completed. Modify the node status to ‘done’ and dequeue it from device.  Update all successor nodes’ dependency\_cnt and add ready nodes into ready\_list. |

# Adapter Module:

Defined in simulator.py, simulator\_module(). This function is the entrance of simulator module and act as the adapter. It will read required attribute form input execution DAG and build all NodeMetadata objects and GraphDefine object.

If the data format of input execution DAG is changed, we only need to modify adapter module.

# Simulation Execution Module:

Defined in simulator.py, simulator\_execution(). Run the simulation.

The result of simulation is stored in Graph object.

## Algorithm overview:

Regard all computation devices (both CPU and GPU) as **asynchronized** devices. Each device maintains its job queue, and the finish time of its current job.

Init Simulator

Sync a pending node

Start ready nodes

Return

The simulator keeps a ready list. All nodes whose dependency are fulfilled will be added to the ready list.

Simulator run in two phases:

Phase 1: still has nodes not started. In this phase, the simulator will run a loop, includes 2 steps:

1. Wait until one enqueued op is finished. Dequeue it from device and update all successor nodes’ dependency counter. If the counter is zero, add the successor node into ready list.

This step will handle only one node at a time.

1. Start all nodes in ready list. Enqueue them into device and mark status as ‘pending’

Phase 2: wait until all nodes finished. Record the time of last finished node.

## Flow chart:

### Flow chart for the algorithm





### Flow char for the experiment



## Module system architecture

The UML class diagram for AI Simulator is shown as below.

图片包含 屏幕截图

描述已自动生成