Cancelling a taskflow

Main idea- place a flag with each topology indicating if it is cancelled or not. The cancel method will be called by a modified return type of the executor.run() calls.

Details:

Create 2 atomic Boolean flags – is_cancel and is_torn. is_cancel will indicate if a topology has been cancelled. is_torn will indicate if a topology has been torn down.

Return type of executor run: Initially, executor runs returned std::future object. Now they will return tf::Future object. The new class will have a std::future member which will store the original future object. There will also be a topology in this class corresponding to the executor.run() call. The class will have overloading with the functions of std::future wherein the class will call the corresponding function of std::future.

e.g. auto t1 = exec.run(tf1);

t1 will contain a std::future<T> future_obj and a Topology.

t1.wait() -> will call future_obj.wait().

t1.get() -> will return future_obj.get().

T1.cancel() -> will set is_cancel of the topology to 1.

For async tasks, there is no associated topology. So we create a async_cancel flag member in the Node class. Corresponding to async tasks, we have t1.async_cancel() to cancel async tasks.

Cancelling a node:

Running of a node's task in done in _invoke() method of the exec. Corresponding to each task type, the corresponding invoke is done e.g. _invoke_static_task, _invoke_dynamic_task, etc. The checking of the flag is done in these functions. If the flag is 1, the task is not run.

e.g.

```
inline void Executor::_invoke_static_task(Worker& worker, Node* node) {
  _observer_prologue(worker, node);
  if (!node->_topology->is_cancel){
    std::get<Node::StaticTask>(node->_handle).work();
  }
  _observer_epilogue(worker, node);
}
```

Also, if a tpg is cancelled, the successors of the nodes will not be scheduled, the join_counters will not be changed. Instead, the topology will be sent for tearing down directly.

Tearing a cancelled topology:

In conventional method, a topology will be rerun if the predicate is false, and the callback will be run after the topology run is complete. For a cancelled topology, there will be no checking of predicate and no callback function will be run. Instead, the topology will be torn down directly. We create a new function _tear_cancelled_topology. It is based on _tear_down_topology method. In this function, we first check if is_torn of the tpg is true or not. This flag indicates if that tpg has been torn down. If it hasn't been torn down, we acquire the lock and recheck if is_torn is false. Then we tear down the tpg just as _tear_down_topology does it. It makes is_torn true after tearing down to ensure that another thread does not tear down the same topology in parallel.

Overall, the following changes have to be done:

1. Executor.hpp-
   Change return types of run, run_n and run_until;
   add flag checks to invoke_static_task, invoke_dynamic_task, etc;
   add a function _tear_cancelled_topology;
   add flag checks to call_tear_cancelled_topology in _invoke method;
   change return type of async method.
2. Taskflow.hpp
   Add tf::Future class which has std::future object and topology. Add corresponding methods of std::future like get(), wait(), etc. Add cancel() method to set flag of topology.
3. Graph.hpp
   Add aync_cancelled flag to node class to check cancellation of async tasks.
4. Topology.hpp
   Add is_cancel and is_torn flags to check if tpg is cancelled and torn down.