CUDA Graphs / HiHAT Graphs
Hierarchical Heterogeneous Asynchronous Tasking

Wojtek Wasko, CJ Newburn

Version 190916

HiHAT Graphs

motivation

- use graphs but allow interoperability and retargetability
- other targets may not (yet) support graphs
  - an agent may sequence the graph into a series of primitives (hhActions) that the targets support
- prototyping with CUDA Graphs as the underlying implementation

HiHAT Graphs API (1)

helper functions to fill out the descriptor structs

- helper functions to fill out descriptors
- HiHAT actions translated into node types

```
hhRet hhhFillGraphNodeMemsetDescr(hheGraphNodeMemsetDescr *descr,
        hhResrcHndlSet exec_resrc_set,
        hhDataView mem_hndl, unsigned char value, size_t num_bytes);
```

- one function to add nodes

```
hhRet hheGraphNodeAdd(
    hheGraphNodeDescr *descr,
    hheGraphNodeExecDescr *exec,
    hheGraphTemplate graph,
    hheGraphTemplateNode *out_node
);
```

- another to add edges

```
hhRet hheGraphTemplateEdgeAdd(
    hheGraphTemplateNode       source,
    hheGraphTemplateNode       target
);
```

HiHAT Graphs API (2)

instantiate, invoke

- preserved HiHAT semantics of execution resources, configuration and policy

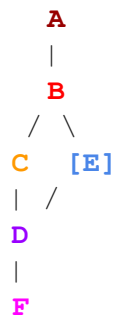
```
hhRet hheGraphInstantiate(
    hheGraphTemplate graph,
    hhResrcHndlSet exec_resrc_set,
    hhExecPol exec_pol,
    hhExecCfg exec_cfg,
    hheGraphInstance *out_graph
);
```

- output action handle - composes with rest of HiHAT

```
hhRet hheGraphInvoke(
    hheGraphInstance graph,
    hhResrcHndlSet exec_resrc_set,
    hhExecPol exec_pol,
    hhExecCfg exec_cfg,
    hheGraphInvocation *out_graph_invocation,
    hhActionHndl *out_action_hndl
);
```

Sample graphs application

```
This is an example of how graphs can be used in HiHAT.
The graph in this sample shows manipulation of two integer values by different resources.
It is constructed as follows:
```




```
Where the nodes are:
    *  A  - invocation on the host, modifies the original integers
    *  B  - memcpy of the modified integers in host memory to GPU0
    *  C  - invocation on GPU0, which modifies the integers again
    *  D  - a memcpy of one of the integers from GPU back into CPU memory
    * [E] - subgraph which consists of 1 node only, printing the values in CPU memory
    *  F  - again printing the values in CPU memory
Integers start out with the value 1 and 4. CPU modifies the integers by multiplying them
by 26 and 17, respectively. GPU modifies the values by dividing them by the same factors

[At beginning of time] Integer 1:  1. Integer 2:  4
[E] Integer 1: 26. Integer 2: 68
[F] Integer 1:  1. Integer 2: 68
hhGraphs exiting.
```

Next steps

- generalize to targets which do not support graphs: serialize graph into actions.
- execution of graphs on remote targets