GPU Work Graphs HLSL Cheat Sheet

By Bastian Kuth, Max Oberberger, Quirin Meyer

Launch Modes

```
// launch one thread with one record
[Shader("node")]
[NodeLaunch("thread")]
void ThreadNode(
  ThreadNodeInputRecord<Type> record
){
    Type r = record.Get();...
```

```
// launch grid of groups with one record
[Shader("node")]
[NodeLaunch("broadcasting")]
[NodeDispatchGrid(x, y, z)]
// or use [NodeMaxDispatchGrid(x, y, z)]
// with SV_DispatchGrid in record
[NumThreads(x, y, z)]
void BroadcastingNode(
  DispatchNodeInputRecord<Type> record,
  uint3 gid : SV_GroupId,
  uint3 gtid : SV_GroupThreadId
){
    Type r = record.Get();...
```

```
// launch one group with up to n records
[Shader("node")]
[NodeLaunch("coalescing")]
[NumThreads(x, y, z)]
void CoalescingNode(
  [MaxRecords(n)]
  GroupNodeInputRecords<Type> records,
  uint3 gtid : SV_GroupThreadId
){
    if(gtid.x < records.Count()){
        Type r = records.Get(gtid.x);...
```

Output Attributes

`[MaxRecords(n)]` maximum outputs
`[MaxRecordsSharedWith(name)]` share out mem
`[NodeID("name")]` use alias name
`[NodeID("name", idx)]` use idx in node array
`[NodeArraySize(count)]` set array size
`[UnboundedSparseNodes]` no array limit
`[AllowSparseNodes]` allow node to not exist

Limits

maximum graph depth: 32
maximum (non empty) outputs per group: 256
per thread-launched thread: $8 = 256/32$
maximum total size of outputs per group: 32KB
per thread-launched thread: $128\text{B} = 32\text{KB}/32/8$

Function Attributes

`[NodeLaunch("mode")]` declare desired launch mode
`[NodeIsProgramEntry]` to enter graph via this node
`[NodeID("name")]` override node id
`[NodeID("name", idx)]` or set array idx
`[NodeShareInputOf("name")]` launch with other
`[NodeDispatchGrid(x,y,z)]` static broadcast grid
`[NodeMaxDispatchGrid(x,y,z)]` limit dynamic grid
`[NodeMaxRecursionDepth(d)]` max self-recursion

Node Output

```
// allocate outputs per _thread_
[NumThreads(8, 1, 1)]
  ...
  [MaxRecords(16)] NodeOutput<Type> targetId
){
  // allocate 2 per thread and 2*8=16 total
  ThreadNodeOutputRecords<Type> records =
    targetId.GetThreadNodeOutputRecords(2);
  // 2 outputs visible only to the thread
  records.Get(0)...
  records.Get(1)...
  records.OutputComplete();...
```

```
// allocate outputs per _group_
[NumThreads(8, 1, 1)]
  ...
  [MaxRecords(8)] NodeOutput<Type> targetId
){
  GroupNodeOutputRecords<Type> records =
    targetId.GetGroupNodeOutputRecords(8);
  // 8 outputs visible to the whole group
  records.Get(gtid)...
  records.OutputComplete();...
```

Recursion

```
[NodeMaxRecursionDepth(4)]
void Node(
  ThreadNodeInputRecord<Type> inputRecord,
  [MaxRecords(4)] NodeOutput<Type> Node
){
  bool hasOutput =
    GetRemainingRecursionLevels() > 0;
  // always do thread uniform allocation!
  ThreadNodeOutputRecords<Type> o =
    Node.GetThreadNodeOutputRecords(
      hasOutput * 4
    );
  // but only write to valid memory!
  if (hasOutput) {
    o.Get(0-4)...
  }
  o.OutputComplete();
}
```

Synchronization

```
struct [NodeTrackRWInputSharing] Type {...
[Shader("node")]
[NodeLaunch("broadcasting")]
  ...
  globallycoherent
  RWDispatchNodeInputRecord<Type> input
){
  ...
  Barrier(NODE_INPUT_MEMORY,
    DEVICE_SCOPE | GROUP_SYNC);
  if(!input.FinishedCrossGroupSharing())
    return;
  // do smth as the last group to finish
```

COBURG UNIVERSITY of applied sciences and arts

AMD together we advance_