A Framework to Transform In-Core GPU Algorithms to Out-of-Core Algorithms Supplemental material

Takahiro Harada*
Advanced Micro Devices, Inc.

1 Kernel Examples

Here we show a few kernel transformation examples. There are in core and out of core impementations. The modification made for the out of core implementation is highlighted in **bold**. These kernels are slightly simplified from the one used for the benchmark of the paper for illustration purpose.

1.1 Kernel computing geometric normal and shading normal

1. In core implementation

```
__kernel
void ComputeNormalKernel( __global Ray* gRays, __global Hit* gHits, __global HitNormal* gHitNormalOut,
  __global Triangle* gTriStorage )
{
  const int gIdx = GET_GLOBAL_IDX;
  Triangle t;

  if( hasHit( gHits[gIdx] ) )
  {
    t = gTriStorage[ gHits[gIdx].m_idx ];

    const float4 ng = normalize3( cross3( t.v1-t.v0, t.v2-t.v0 ) );
    const float4 hp = gRays[gIdx].getHitPoint();
    const float4 bCrd = calcBaryCrd( hp, t.v0, t.v1, t.v2 );
    const float4 ns = normalize3( bCrd.x * t.n0 + bCrd.y * t.n1 + bCrd.z * t.n2 );
    gHitNormalOut[gIdx].m_ng = ng;
    gHitNormalOut[gIdx].m_ns = ns;
  }
}
```

2. Out of core implementation

```
__kernel
void ComputeNormalKernel( __global Ray* gRays, __global Hit* gHits, __global HitNormal* gHitNormalOut,
  VM_KERNEL_ARGS ) //ooc
{
  const int gIdx = GET_GLOBAL_IDX;
  Triangle t;

  VMInitialize;

  if( hasHit( gHits[gIdx] ) )
  {
    VMLoad( Triangle, gHits[gIdx].m_idx * sizeof(Triangle), &t, 0 );

    const float4 ng = normalize3( cross3( t.v1-t.v0, t.v2-t.v0 ) );
    const float4 hp = gRays[gIdx].getHitPoint();
    const float4 bCrd = calcBaryCrd( hp, t.v0, t.v1, t.v2 );
    const float4 ns = normalize3( bCrd.x * t.n0 + bCrd.y * t.n1 + bCrd.z * t.n2 );

    gHitNormalOut[gIdx].m_ng = ng;
    gHitNormalOut[gIdx].m_ns = ns;
  }

  VMFinalize;
}
```

*e-mail:takahiro.harada@amd.com

1.2 Kernel casting rays to BVH to find the closest intersection

We used stackless BVH traversal to minimize the per-WI context. If stacked traversal is used, full stack needs to be loaded and stored.

1. In core implementation

```
__kernel
void RayCastKernel(__global Ray* gRays, __global Hit* gHits,
  __global BvhNode* gBvh, __global Triangle* gTriStorage )
{
  const int gIdx = GET_GLOBAL_IDX;

  Triangle t;
  float minDist = gHits[gIdx].m_fraction;
  u32 minIdx = 0;
  u32 nodeIdx = 0;
  BvhNode node;
  Ray ray = gRays[gIdx];

  while( nodeIdx != BVH_TERMINATION )
  {
    node = gBvh[ nodeIdx ];

    nodeIdx = NodeNext( &node );
    Aabb aabb = NodeGetAabb( &node );
    float ff = AabbIntersect( aabb, ray );
    if( ff < minDist )
    {
      if( NodeIsLeaf( &node ) )
      {
        int faceIdx = NodeGetLeafData( &node );

        t = gTriStorage[faceIdx];

        float f = castRay( t.v0, t.v1, t.v2, ray );
        if( f < minDist )
        {
          minDist = f;
          minIdx = faceIdx;
        }
      }
      else
      {
        nodeIdx = NodeGetChild0( &node );
      }
    }
  }

  if( minDist < gHits[gIdx].m_fraction )
  {
    gHits[gIdx].m_fraction = minDist;
    gHits[gIdx].m_idx = minIdx;
  }
}
```

2. Out of core implementation

```
__kernel
void RayCastKernel( __global Ray* gRays, __global Hit* gHits,
  __global u64* gShapeOffsets, VM_KERNEL_ARGS ) //ooc
{
  const int gIdx = GET_GLOBAL_IDX;

  Triangle t;
  float minDist = gHits[gIdx].m_fraction;
  u32 minIdx = 0;
  u32 nodeIdx = 0;
  BvhNode node;
  Ray ray = gRays[gIdx];

  VMInitialize;

  while( nodeIdx != BVH_TERMINATION )
  {
    VMLoad( BvhNode, gShapeOffsets[0] + nodeIdx * sizeof(BvhNode), &node, 0 );

    nodeIdx = NodeNext( &node );
    Aabb aabb = NodeGetAabb( &node );
    float ff = AabbIntersect( aabb, ray );
    if( ff < minDist )
    {
      if( NodeIsLeaf( &node ) )
      {
        int faceIdx = NodeGetLeafData( &node );

        VMLoad( Triangle, gShapeOffsets[1] + faceIdx * sizeof(Triangle), &t, 1 );

        float f = castRay( t.v0, t.v1, t.v2, ray );
        if( f < minDist )
        {
          minDist = f;
          minIdx = faceIdx;
        }
      }
      else
      {
        nodeIdx = NodeGetChild0( &node );
      }
    }
    pc = 0;// reset program counter
  }

  if( minDist < gHits[gIdx].m_fraction )
  {
    gHits[gIdx].m_fraction = minDist;
    gHits[gIdx].m_idx = minIdx;
  }

  VMFinalize;
}
```

2 The Macro

Here we show the implementation of the macro used in the kernels shown above.

```
typedef struct
{
  u64 m_baseAddr;
  u64 m_dataSize;

  u32 m_pageSize;
  u32 m_nPages;
  u32 m_padd;
  u32 m_padd1;
} VMHeader;

typedef struct
{
  u64 m_offset;
  u32 m_isAvailable;
  u32 m_time;
} VMPageTable;

u64 VMAddr2Offset( u64 addr, VMHeader h ) { return (addr-h.m_baseAddr); }
u64 VMAddr2PageIdx( u64 addr, VMHeader h ) { return VMAddr2Offset( addr, h )/h.m_pageSize; }

#define VM_ARGS __global VMPageTable* vmPt, __global char* vmStorage, const VMHeader vmHeader, const int vmFirst
#define VM_ARG_LIST vmPt, vmStorage, vmHeader, vmFirst
#define VM_KERNEL_ARGS  __global u32* vmReqs, __global u32* vmNReqs, \
  __global char* vmCtxt,\
  VM_ARGS,\
  const int vmNReqsMax

// Load the WI state and go where the WI was suspended
#define VMInitialize \
  u64 vmAddr = 0;\
  u32 pc = 0;\
  if( !vmFirst )\
  {\
    LOAD_CTXT_MACRO;\
    if( pc == VMPcFinished ) return;\
    switch( pc )\
    {\
    case 0: goto VMRESTART0;\
    case 1: goto VMRESTART1;\
    case 2: goto VMRESTART2;\
    default: break;\
    }\
  }

// Save the WI state and append the page request if any
#define VMFinalize \
  pc = VMPcFinished;\
  SAVE_CTXT_MACRO;\
  return;\
SAVE_CTXT:\
  {\
    u32 o = AtomInc( vmNReqs[0] );\
    AtomInc( vmReqs[(vmAddr-vmHeader.m_baseAddr)/vmHeader.m_pageSize] );\
  }\
  SAVE_CTXT_MACRO;

// Request memory at addr
#define VMRequest(TYPE, addr, dst, VM_ARGS) \
  bool request##TYPE( u64 addr, TYPE* dst, VM_ARGS )\
  {\
    u64 o = VMAddr2Offset( addr, vmHeader );\
    u64 pidx = VMAddr2PageIdx( addr, vmHeader );\
    VMPageTable p = vmPt[pidx];\
    if( !p.m_isAvailable )\
      return false;\
    u64 d = o - vmHeader.m_pageSize * pidx;\
    (*dst) = *(__global TYPE*)&vmStorage[ p.m_offset + d ];\
    return true;\
  }

VMRequest(float, addr, dst, VM_ARGS)

// Load data through the VM
#define VMLoad( TYPE, address, dst, LOADCOUNTER )     \
VMRESTART##LOADCOUNTER: \
  {vmAddr = address;\
  if( !request##TYPE( vmAddr, dst, VM_ARG_LIST ) ) \
  { \
    pc = LOADCOUNTER; \
    goto SAVE_CTXT; \
  }}
```