Obsidian: GPGPU Programming in Haskell

Joel Svensson

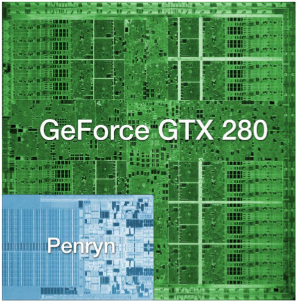
Chalmers University of Technology

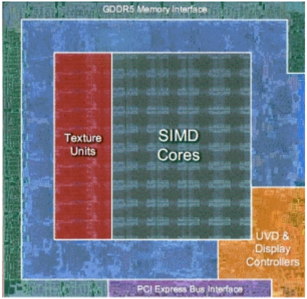
May 15, 2009

General-purpose computations of GPUs

Why are GPUs interesting for non-graphics programming:

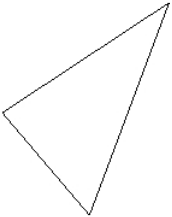
- Cost efficient highly parallel computers
- ~500 processing cores (NVIDIA 295GTX)
- Taste of the future, today!
  - Programming a 500 core machine is very different from a 2,4,8,16 core machine.





GPUs and Graphics

GPUs are made to draw Triangles. As many as possible as quickly as possible.



Screenshot

Figure: Fallout 3, Bethesda Softworks.

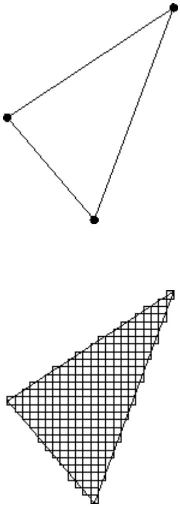
Vertex and Fragment programs

Vertex programs:

- Executed per vertex.
- 3d to 2d transformations.

Fragment programs:

- Executed per fragment (potential pixel).
- Computes a color value.



Vertex and Fragment programs

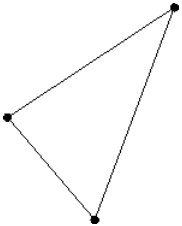
Vertex programs:

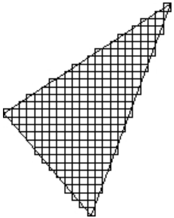
- Executed per vertex.
- 3d to 2d transformations.

Geometry programs:

Fragment programs:

- Executed per fragment (potential pixel).
- Computes a color value.





Unified Architecture

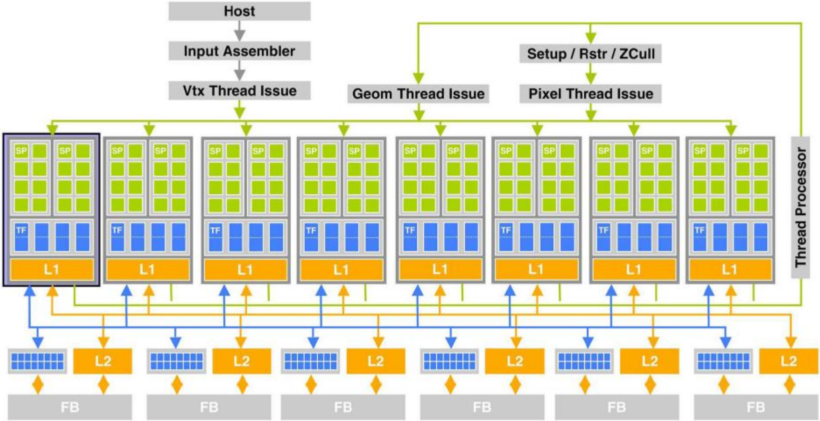


Figure: The NVIDIA 8800GTX GPU architecture, with 8 pairs of multiprocessors. Diagram courtesy of NVIDIA.

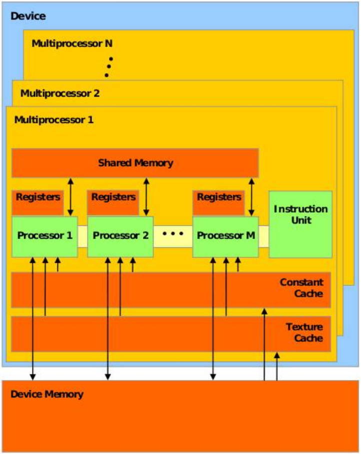
Unified Architecture

In each multiprocessor:

- 16Kb Shared memory.
- 8192 32-bit registers.

Memory:

- Uncached device memory.
- Read-only constant memory.
- Read-only texture memory.



CUDA

NVIDIA Compute Unified Device Architecture:

- NVIDIA's Hardware architecture + programming model.
- Provides compiler and libraries.
    - Extension to C.
- Enables/Simplifies implementing general purpose algorithms on the GPU

Simple CUDA Example

```
__global__ static void sum(int * values,int n)
{
  extern __shared__ int shared[];
  const int tid = threadIdx.x;
  shared[tid] = values[tid];
  for (int j = 1; j < n; j *= 2) {
  __syncthreads();
    if ((tid + 1) % (2*j) == 0)
      shared[tid] += shared[tid - j];
  }
  values[tid] = shared[tid];
}
```

More CUDA

The code listing on the previous slide defines a *Kernel*:

- A kernel is executed by a block of threads:
  - Up to 512 threads per block.
  - Run on one multiprocessor.
- The threads are further divided into *Warps*
  - The scheduled unit.
  - Group of 32 threads.
- Many blocks can be executed concurrently:
  - Referred to as a *Grid* of Blocks.

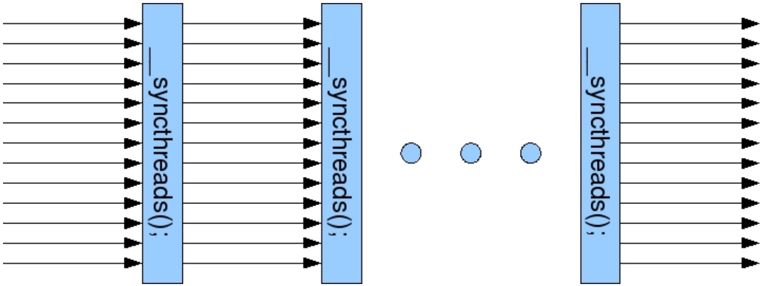
__synchtreads()

Barrier synchronisation primitive:

- Barrier across all the threads of a block.
- Used to coordinate communication
  - Within a block.

```
if (even(tid)) {
   /* do something */
   __syncthreads();
}
```

More __synchthreads()



Obsidian Outline:

- Embedded in Haskell.
- Tries to stay in the spirit of Lava.
  - Combinator library.
- Higher level of abstraction compared to CUDA.
  - While still assuming knowledge of architecture characteristics in the programmer.

Our aims with Obsidian

- Generate efficient code for GPUs from short and clean high level descriptions.
- Make design decisions easy.
  - Where to place data in the memory hierarchy.
  - What to compute where, and when.

Our aims with Obsidian

- Generate efficient code for GPUs from short and clean high level descriptions.
- Make design decisions easy.
  - Where to place data in the memory hierarchy.
  - What to compute where, and when.
- We are not there yet.

Simple Obsidian Example

```
sumUp :: Int -> Arr IntE :-> Arr IntE
sumUp 0 = Pure id
sumUp n = Pure (pairwise (uncurry (+))) ->- sync
                   ->- sumUp (n-1)
```

Running an Obsidian program on the GPU

```
Obsidian> execute GPU (sumUp 4) [0..15]
[120]
```

The generated code

```
__global__ void generated(unsigned int* input,unsigned int* result){
unsigned int tid = (unsigned int)threadIdx.x;
extern __shared__ unsigned int s_data[];
unsigned int __attribute__((unused)) *sm1 = &s_data[0];
unsigned int __attribute__((unused)) *sm2 = &s_data[8];
sm2[tid] = ((unsigned int)((input[(tid << 1)] + input[((tid << 1) + 1)])));
__syncthreads();
if (tid < 4){
sm1[tid] = ((unsigned int)((((int)(sm2[(tid << 1)])) + ((int)(sm2[((tid << 1) + 1)])))));
}
__syncthreads();
if (tid < 2){
sm2[tid] = ((unsigned int)((((int)(sm1[(tid << 1)])) + ((int)(sm1[((tid << 1) + 1)])))));
}
__syncthreads();
if (tid < 1){
sm1[tid] = ((unsigned int)((((int)(sm2[(tid << 1)])) + ((int)(sm2[((tid << 1) + 1)])))));
}
__syncthreads();
if (tid < 1){
result[tid] = ((unsigned int)(sm1[tid]));
}
}
```

Inside Obsidian

Key parts of Obsidian:

- Arrays

```
data Arr a = Arr (IxExp -> a) Int
```

- Obsidian programs

```
data a :-> b
     = Pure (a -> b)
     | Sync (a -> Arr FData) (Arr FData :-> b)
```

- Collection of combinators and functions.

`two`, `->-`, `sync`, `pair`, `unpair`, `zipp`, `unzipp`, etc

About `sync`

The Obsidian `sync` has many roles :

- Values are stored in shared memory.
  - Enables sharing of computed results between threads.
- Introduces parallelism.
- Assigns work to threads.
  - The length of the input array specifies the number of threads.

```
sync :: (Flatten a) => Arr a :-> Arr a
```

instances of Flatten have functions `toFData` and `fromFData` defined on them.

Implementation of sync

```
sync :: Flatten a => Arr a :-> Arr a
sync = Sync (fmap toFData) (Pure (fmap fromFData))
```

Example: `sync` Introduces Parallelism

```
sumUp :: Int -> Arr IntE :-> Arr IntE
sumUp 0 = Pure id
sumUp n = Pure (pairwise (uncurry (+))) ->- sync
                 ->- sumUp (n-1)

sumUp2 :: Int -> Arr IntE :-> Arr IntE
sumUp2 0 = Pure id
sumUp2 n = Pure (pairwise (uncurry (+)))
                 ->- sumUp2 (n-1)
```

Example: `sync` Assigns work to threads

```
addOne :: Arr IntE :-> Arr IntE
addOne = Pure (fmap (+1)) ->- sync

addOne' :: Arr IntE :-> Arr (IntE,IntE)
addOne' = Pure (fmap (+1)) ->-
               Pure pair ->- sync

Obsidian> execute GPU addOne [0..7]
[1,2,3,4,5,6,7,8]

Obsidian> execute GPU addOne' [0..7]
[(1,2),(3,4),(5,6),(7,8)]
```

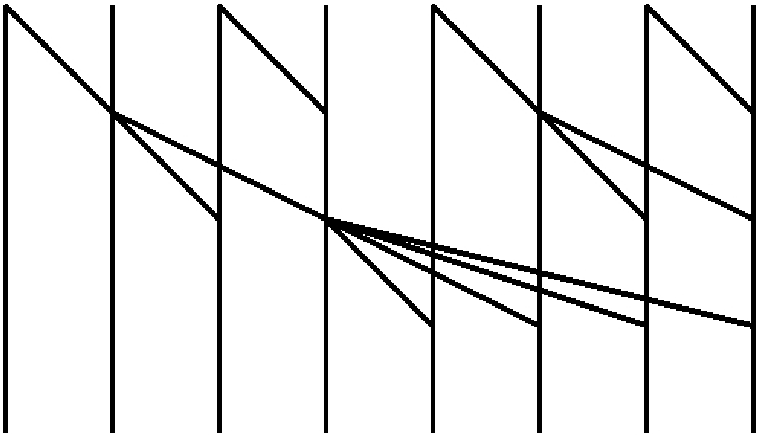
Parallel prefix

```
sklansky :: (Flatten a, Choice a) =>
            (a -> a -> a) -> Int -> (Arr a :-> Arr a)
sklansky op 0 = Pure id
sklansky op n = two (sklansky op (n-1)) ->- Pure (fan op) ->- sync

fan op arr = conc (a1, (mapArray (op c) a2))
    where (a1,a2) = halve arr
          c       = a1 ! (fromIntegral (len a1 - 1))


Obsidian> execute GPU (sklansky (+) 3) ([0..7] :: [IntE])
[0,1,3,6,10,15,21,28]
```

Drawing a Sklansky



Generated code

```
__global__ void generated(unsigned int* input,unsigned int* result){
unsigned int tid = (unsigned int)threadIdx.x;
extern __shared__ unsigned int s_data[];
unsigned int __attribute__((unused)) *sm1 = &s_data[0];
unsigned int __attribute__((unused)) *sm2 = &s_data[8];
sm2[tid] = ((unsigned int)((((tid & 0xfffffff9) < 1) ?
((int)(input[tid])) :
(((int)(input[(tid & 0x6)])) + ((int)(input[tid])))))));
__syncthreads();
sm1[tid] = ((unsigned int)((((tid & 0xfffffffb) < 2) ?
((int)(sm2[tid])) :
(((int)(sm2[((tid & 0x4) | 0x1)])) + ((int)(sm2[tid])))))));
__syncthreads();
sm2[tid] = ((unsigned int)(((tid < 4) ?
((int)(sm1[tid])) :
(((int)(sm1[3])) + ((int)(sm1[tid])))))));
__syncthreads();
result[tid] = ((unsigned int)(sm2[tid]));
}
```

Conclusions

- Obsidian is work in progress.
  - Changing rapidly.
- Promising: for some applications we are generating quite efficient code.
  - More needs to be done.
  - Generalise.

Conclusions cont.

Applications:

- Sorting.
- Parallel prefix.
- Reduction.

End.

More Questions?