Technische Universität Braunschweig

Dynamic parallelism using CUDA

The shallow water equation on an adaptive grid

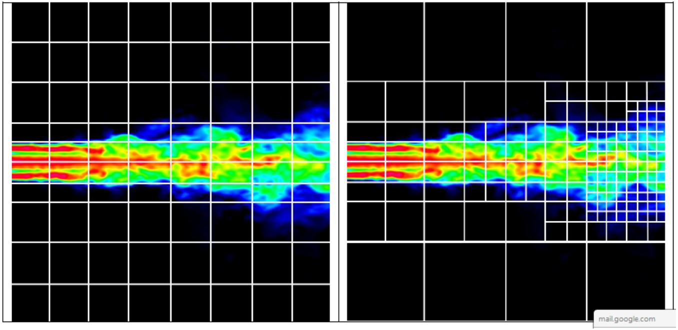
Marc Kassubeck, Torsten Thoben, 18. März 2015

Inhalt

- **The task**
- **Dynamic parallelism using CUDA**
  - Implementation of DP with CUDA
  - DP and Synchronisation
  - Recursion Depth and Device Limits
- **The Problem**
  - First try
  - Back to Arrays
  - Results and discussion

The Task

- Extension of an existing shallow water sovler
- With dynamic prallelism (DP)



Implementation of DP with CUDA

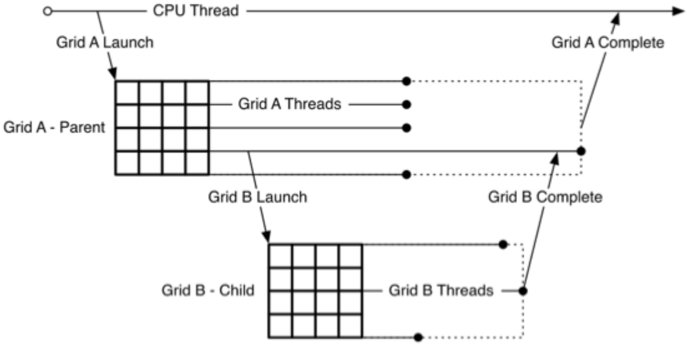
- DP in CUDA means calling a kernel inside a kernel
- So every thread calls a new grid

```
main(){
    ...
    func <<< g,b >>> (depth);
}

__global__ void func(int depth){
    ...
    if(depth<2)
        func  <<< g, b >>> (depth+1);
}
```

DP and Syncronisation

- Father-kernel finished if his subgrid is finished but can continue working after the subkernel call
- To synchronize between one thread and his subgrids use `cudaDeviceSynchronize()` ($\neq$ `__syncthreads()`)

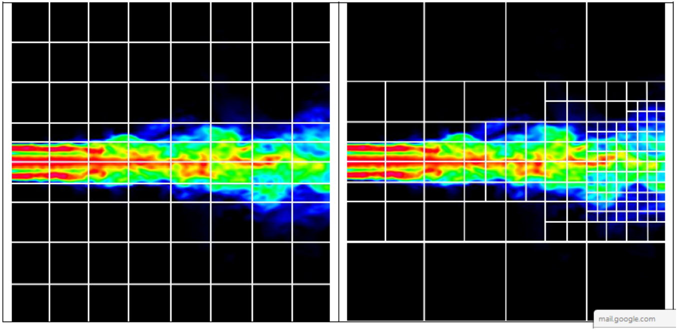


Recursion Depth and Device Limits

- Nesting depth (hardware-limit is 24)
  - Need to reserve a buffer for running/suspended or launching kernels
  - `cudaDeviceSetLimit( cudaLimitDevRuntimePendingLaunchCount, x)`
  - Default is set to 2048
- Synchronization depth
  - `cudaDeviceLimit(cudaLimitDevRuntimeSyncDepth, x)`
  - Default is set to 2

Problem: Implementation of the datastructure

- Datastructure: Forest (many Trees)
- Limit to coarseness is the `forestsize` (number of trees)
- Limit to getting fine end up to the maximum recursions
- Trees: eg Quadtrees, Octrees...



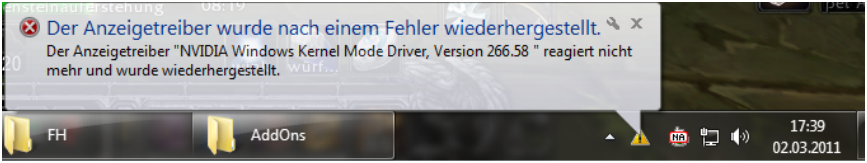
GPU Datastructure using pointer

```
class LeafElem : public TreeElem
{
public:
   float value;
   __device__ __host__ LeafElem();
   virtual __device__ __host__ ~LeafElem(){};
   virtual __device__ __host__ bool isLeaf()
   {
     return true;
   }
};
```

GPU Datastructure using pointer

```
class BranchElem : public TreeElem
{
 public:
    int nx;
    int ny;
    int depth;
    TreeElem** children;
    __device__ __host__ BranchElem(int nx, int ny, int depth)
    virtual __device__ __host__ ~BranchElem();
    virtual __device__ __host__ bool isLeaf()
    {
        return false;
    }
 };
```

But this fails



Der Anzeigetreiber wurde nach einem Fehler wiederhergestellt.

Der Anzeigetreiber "NVIDIA Windows Kernel Mode Driver, Version 266.58 " reagiert nicht mehr und wurde wiederhergestellt.

- Kernel calls take too long to execute (> 7 seconds)
- Possible causes:
  - Exponentially growing trees
  - Overhead to launch subgrids
  - No coalesced memory access with pointers of this data structure

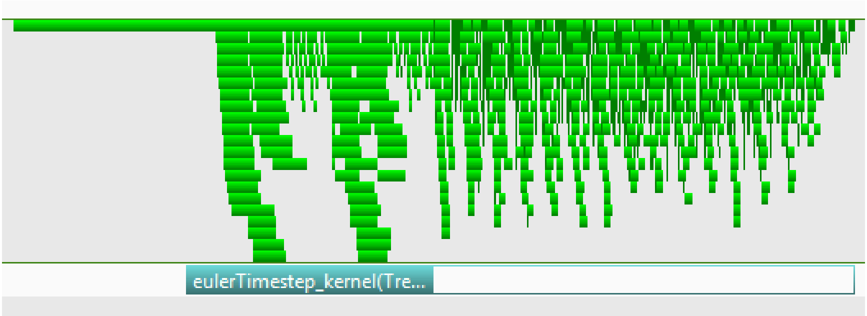
Storing the tree-structure in an Integer array

0	0	0	0	1	1	2	2
0	0	0	0	1	1	2	2
0	0	0	0	1	1	1	1
0	0	0	0	1	1	1	1

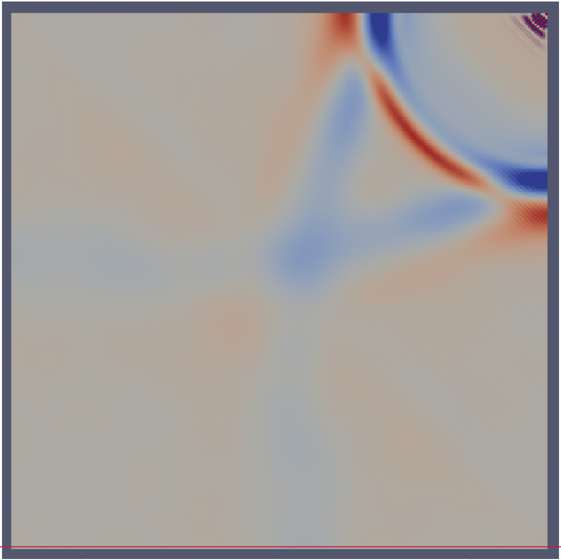
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

- Integer values denote the level in the tree
- The higher the value, the finer the grid will be in this area
- Solution values are averaged and stored in all corresponding 'supercells'
- Allows for better coalesced memory access

Profiling



Results



Problems

- Unfortunately this is still the solution using the original solver
- Some problems surfaced, when implementing the dynamic version
- Implementing the memory using CUDA Unified memory:
  - On the one hand no `cudaMemcpy`
  - On the other hand many calls to `cudaDeviceSynchronize`
  - Too much data transfer CPU ↔ GPU
  - Too slow using our current implementation
- Problems in the solver code
  - Solutions may contain ∞ or `#INDEFINITE`
  - Maybe stability issue
  - Maybe bug in solver code