GPU Matrix multiplication

- The `matmult_f.nvcc` driver is provided

```
matmult_f.nvcc type m n k [bs]

where m, n, k are the parameters defining the matrix sizes, bs is the
optional blocksize for the block version, and type can be one of:

nat     - the native/naive version
lib     - the library version (note that this now calls a multithreaded
library)
gpu1    - the first gpu version
gpu2    - the second gpu version
gpu3    - the third gpu version
gpu4    - the fourth gpu version
gpu5    - the fifth gpu version
gpu6    - the sixth gpu version
gpulib  - the CUBLAS library version

as well as blk, mnk, nmk, ... (the permutations).
```

- See README for more (also week 1 README)

GPU Matrix multiplication

- Driver uses `dlsym` for dynamic linking (C99 library) – therefore all functions in your shared library must have C naming convention
- Use `extern "C" {}`

```
extern "C" {
   #include <cblas.h> // The "C” headers also inside
...
   void matmult_lib(...)
   {
      ...
   }
...etc.
}
```

GPU Matrix multiplication

- New `cuBlas` API used in the assignment
  - `#include <cublas_v2.h>`
- Creating a handle (to `cuBlas` context)
  - Important when using multiple host threads and multiple GPUs and makes it reentrant (non-blocking)
  - First creation has a large overhead!!

```
cublasStatus_t stat;
cublasHandle_t handle;
stat = cublasCreate(&handle);
```

- The `matmult_f.nvcc` driver creates a handle (and destroys it again) to ”wake up” cuBlas

GPU Matrix multiplication

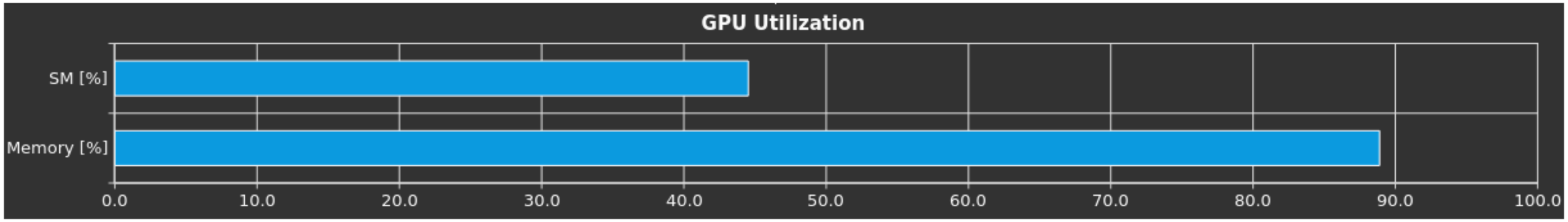
- For large matrices please use
  - `MFLOPS_MAX_IT=1 ./matmult_f.nvcc ...`
- For benchmarks please use
  - `MATMULT_COMPARE=0`
- For running on 1 core on CPU
  - `MKL_NUM_THREADS=1`
- Overflow for large matrices (Please note!!)
  - You should have the same overflow for all methods!

```
n-62-12-19(hhbs) $ ./matmult_f.nvcc gpulib 2048 2048 2048
   98304.000 1178532.905 300 # matmult_gpulib
```

Overflow – not an actual error!

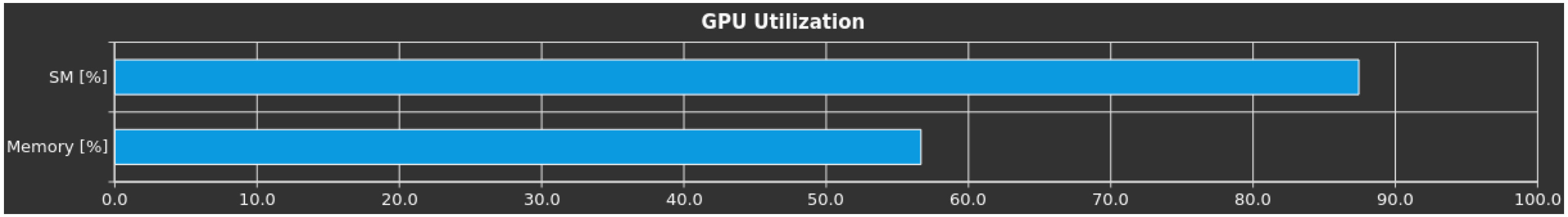
GPU Matrix multiplication

- Profiler (if available) may help to explain results
  - ❑ `matmult_f.nvcc gpu2 1024 1024 1024`



Memory bound!

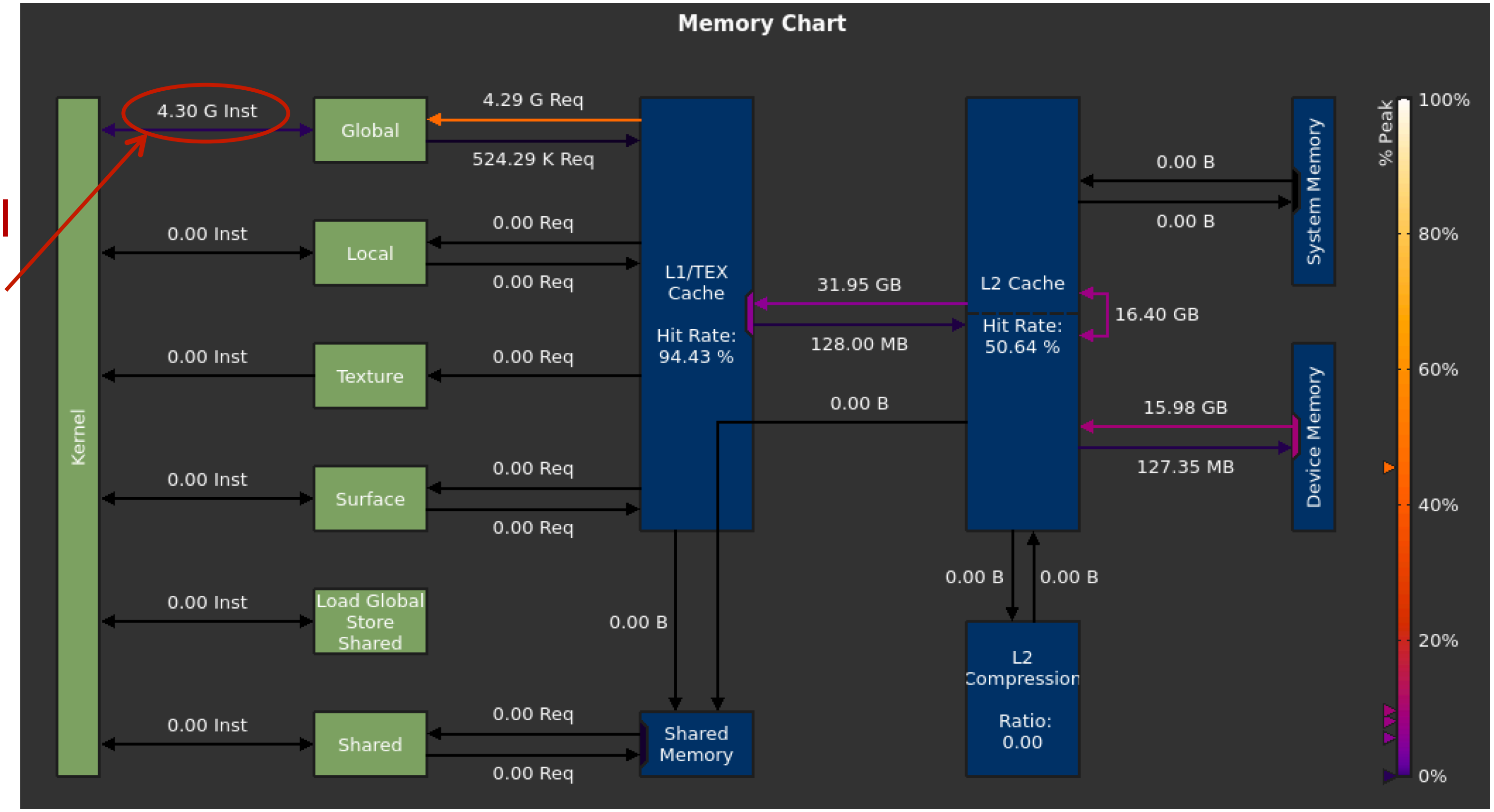
  - ❑ `matmult_f.nvcc gpulib 1024 1024 1024`



Compute bound!

GPU Matrix multiplication

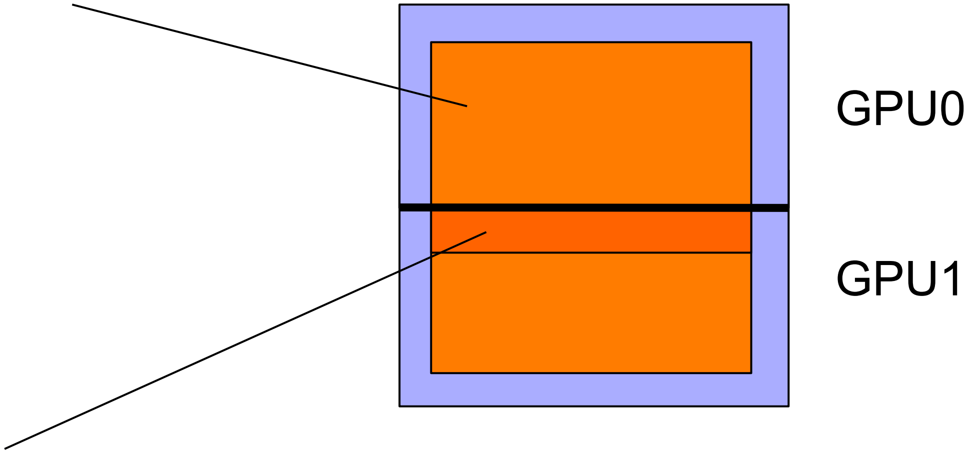
- Profiler (if available) may help to explain results
  - ❑ `matmult_f.nvcc gpu2 4096 4096 4096`



We will try to reduce this

GPU Poisson problem

- Multi-GPU version using Peer-to-peer access
  - ❑ Split task into two – top and bottom
  - ❑ Interior points can be updated from global memory
  - ❑ Border points must read ”peer values” from other GPU



Read from global memory

Available from other GPU

Submitting GPU batch jobs

- Benchmark runs should always be submitted to queue `hpcintrogpu`
- See https://www.hpc.dtu.dk/?page_id=2759
- Maximum wall-clock time on jobs is 1 hour!
- For jobs using two GPUs use `num=2`
- For CPU-only jobs please do not request GPUs
- The `-G` flag sets debug lines into your code
  - This reduces the performance drastically
  - Remove it for performance tuning!!!

Reports / Analysis

- Assess your performance (Gflops / Bandwidth)
  - Profiler gives these numbers (“Throughput”)
- Speed-up calculations (fair)
  - If using 1 GPU, compare with 1 CPU!
  - `numactl --cpunodebind=0` to bind threads to cpu 0
- Tuning considerations
- Profiler analysis (if available)
- Relevant comments and observations
- Please keep the report format as close to the assignment questions as possible
- Please try to have one section in the report for each question in the assignment (helps us read it)