A3: Question 1

- The `matmult_c/f.nvc++` driver is provided

```
matmult_f.nvc++ 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)
mkn_omp       - the OpenMP mkn version
mkn_offload   - the OpenMP offload mkn version
mnk_offload   - the OpenMP offload mnk version
blk_offload   - the OpenMP offload blocked version
asy_offload   - the OpenMP offload asynchronous version
lib_offload   - the OpenMP offload CUBLAS library version
exp_offload   - the OpenMP offload experimental version (if needed)

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

- New `README` and `Makefile` provided

A3: Question 1

- 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(...)
   {
      ... // Same code as in week 1
   }
...etc.
}
```

A3: Question 2

- The driver provides host arrays only – you must map them to / from the device
- It is not efficient to do the summation in the innermost loop directly in device memory as `C[i][j] += ....` Use a variable.

```
   void matmult_mnk_offload(int m, int n, int k, double
**A, double **B, double **C)
   {
      ...
         double sum = 0;
         for (int kk = 0; kk < k; ++kk)
             sum += A[i][kk] * B[kk][j];
         C[i][j] = sum;
      ...
   }
```

A3: Question 3

- Block the outermost m loop with blocksize BLK

```
void matmult_blk_offload(int m, int n, int k, double **A,
double **B, double **C)
{
    #define BLK ...
    ...
    for (int i = 0; i < m; i += BLK) {
        for (int j = 0; j < n; ++j) {
            if (i + BLK - 1 < m) {
                double sum[BLK] = {0};
                // Do BLK elements of C here
            } else {
                // Do the remainder part here
            }
        }
    }
}
```

(completely unrolled) !

A3: Question 4

- Difficult – Ask!,.. don't get stuck

```
void matmult_asy_offload(int m, int n, int k, double **A,
double **B, double **C)
{
    // #pragma target data enter ...
   #define SLAPS 4
    ...
   #pragma omp parallel for
   for (int s = 0; s < SLAPS; ++s) {
       // #pragma target data update to(A... )

       // Compute C for slap s

       // #pragma target data update from(C... )
   }
   ...
}
```

A3: Question 5

- 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!!

```
cublasHandle_t handle;
cublasCreate(&handle);
int cublas_error = cublasDgemm(handle, ...);
```

- The `matmult_c.nvc++` driver creates a handle (and destroys it again) to ”wake up” cuBlas

General notes

- For development (until it works) please use
  - `MFLOPS_MAX_IT=1 ./matmult_f.nvc++ ...`
- For benchmarks please use
  - `MATMULT_COMPARE=0`
- Overflow for large matrices (Please note!!)
  - You should have the same overflow for all methods!

```
n-62-12-19(hhbs) $ ./matmult_f.nvc++ lib_offload 2048 2048 2048
   98304.000 1178532.905 300 # matmult_lib_offload
```

Overflow – not an actual error!

A3: Question 6

- Use the OpenMP version from last week as a starting point and reference for comparison
  - Remove the stopping criteria calculation for simplicity
- Setup and initialize everything on the host
- Make device versions of routines in `alloc_3d.h`
- Transfer u, u_old, and f with `omp_target_memcpy`
- Do the outer loop over iterations on the host
  - Offload only the inner iterations (Jacobi update)
  - Swap pointers on the host

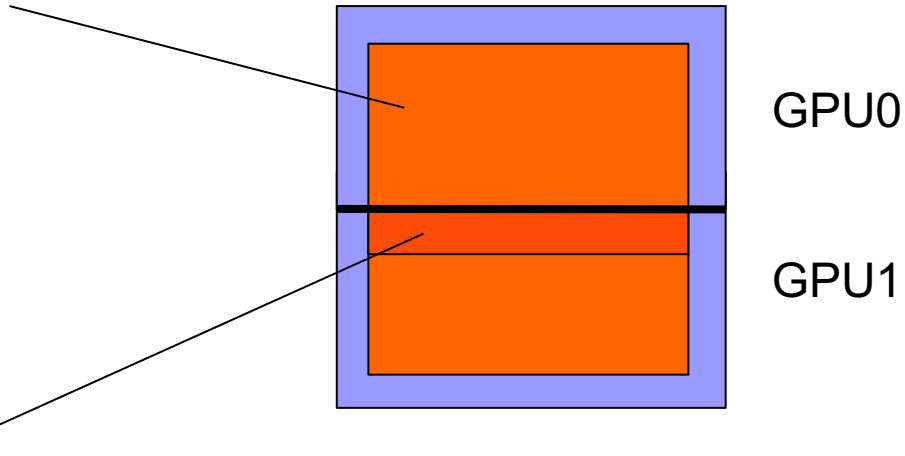
A3: Question 7

Remember!!

```
export CUDA_VISIBLE_DEVICES=0,1
```

- Multi-GPU version using Peer-to-peer access
  - Split task into two – top and bottom
  - Most interior points can be updated from local array
  - Border points must read ”peer values” from other GPU

Read from local array `u0[.][.][.]`



Available from other GPU `u1[.][.][.]`

A3: Question 8

- Use the reduction clause on the target construct
- Look at how much additional time this clause adds to the runtime
- Compare with standard OpenMP on CPU

A3: Profiler GUI

- X-window from login node doesn't work!
- Use:
  - ThinLink -> Open xterm -> `hpcintrogpush -X`