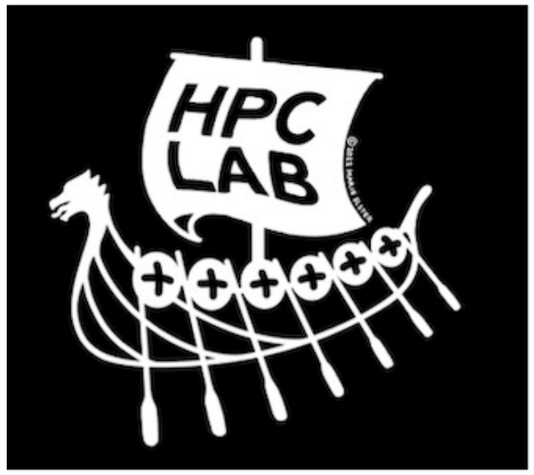


TDT 4200 Fall 2024
Parallel Computing

CUDA Intro
(+ materials CalTech – spring 2024– see BB)

Prof. Anne C. Elster, PhD

Dept. of Computer Science (IDI)

Norwegian Univ. of Science & Technology

(NTNU Trondheim, Norway)

&

Univ. of Texas at Austin (Senior Visiting Scientist)

TDT4200 F2024 Course Information

– See also https://www.ntnu.edu/studies/courses/TDT4200/

- **PS 4 Threads/OpenMP out**
- **PS 5 on CUDA intro out later this week**

NOTE: Compulsory assignments:

– **You need to do and pass ALL Problem Sets/Exercises in order to take the final**

… also those that are Pass/Fail!!

TDT4200 Fall 2024 Student Reference Group

Please contact for praise and suggestions on how we can make this course better:

- Ingar Asheim – ingara@stud.ntnu.no

- Elin Skålund Berntsen – elinsbe@stud.ntnu.no
- Hanne Marie Haakaas – hannamhaa@stud.ntnu.no
- Suzanne Maduga – suzannm@stud.ntnu.no

- Simone Deidier – simondei@stud.ntnu.no
- Davide Esposito – davidees@stud.ntnu.no
- Diego Velasco – diegove@stud.ntnu.no
- ???
- ???

+ Need someone that are from NTNU, but not CS….

Outline

Last time: PS4 OpenMP/Pthead intro by JCM

Today:

- **Course info**
- **CUDA Intro**

CalTech Slides: http://courses.cms.caltech.edu/cs179/
(taught Spring 2024)

GPU Programming

- CUDA (targets Nvidia GPUs) – this course
- OpenCL – more verbose, but works on AMD GPUs etc.
  works also on other heterogenouse platforms
- HIP (CUDA envelope for OpenCL on AMD (e.g. LUMI)
- …

Some of these can also be combined with MPI,
E.g.

- **MPI + Pthreads or MPI + Pthread**
- **MPI + CUDA (across many GPU systems)**

CUDA program – step by step
(based on CalTech slides)

- Setup inputs on the host (CPU-accessible memory)
- Allocate memory for outputs on the host CPU
- Allocate memory for inputs on the GPU
- Allocate memory for outputs on the GPU
- Copy inputs from host to GPU (slow)
- Start GPU kernel (function that executes on gpu – fast!)
- Copy output from GPU to host (slow)

NOTE: Copying can be asynchronous,
and unified memory management is available

CUDA Kernel (”function”)
(based on CalTech slides)

- Like OpenMP “parallel” function
- Excecuted by each thread
- Simple vector add example, implementation:

```
__global__ void
cudaAddVectorsKernel(float * a, float * b, float * c) {
    //Decide an index somehow
    c[index] = a[index] + b[index];
}
```

CUDA Kernel - indexing
(based on CalTech slides)

```
__global__ void
cudaAddVectorsKernel(float * a, float * b, float * c) {
    unsigned int index = blockIdx.x * blockDim.x + threadIdx.x;
    c[index] = a[index] + b[index];
}
```

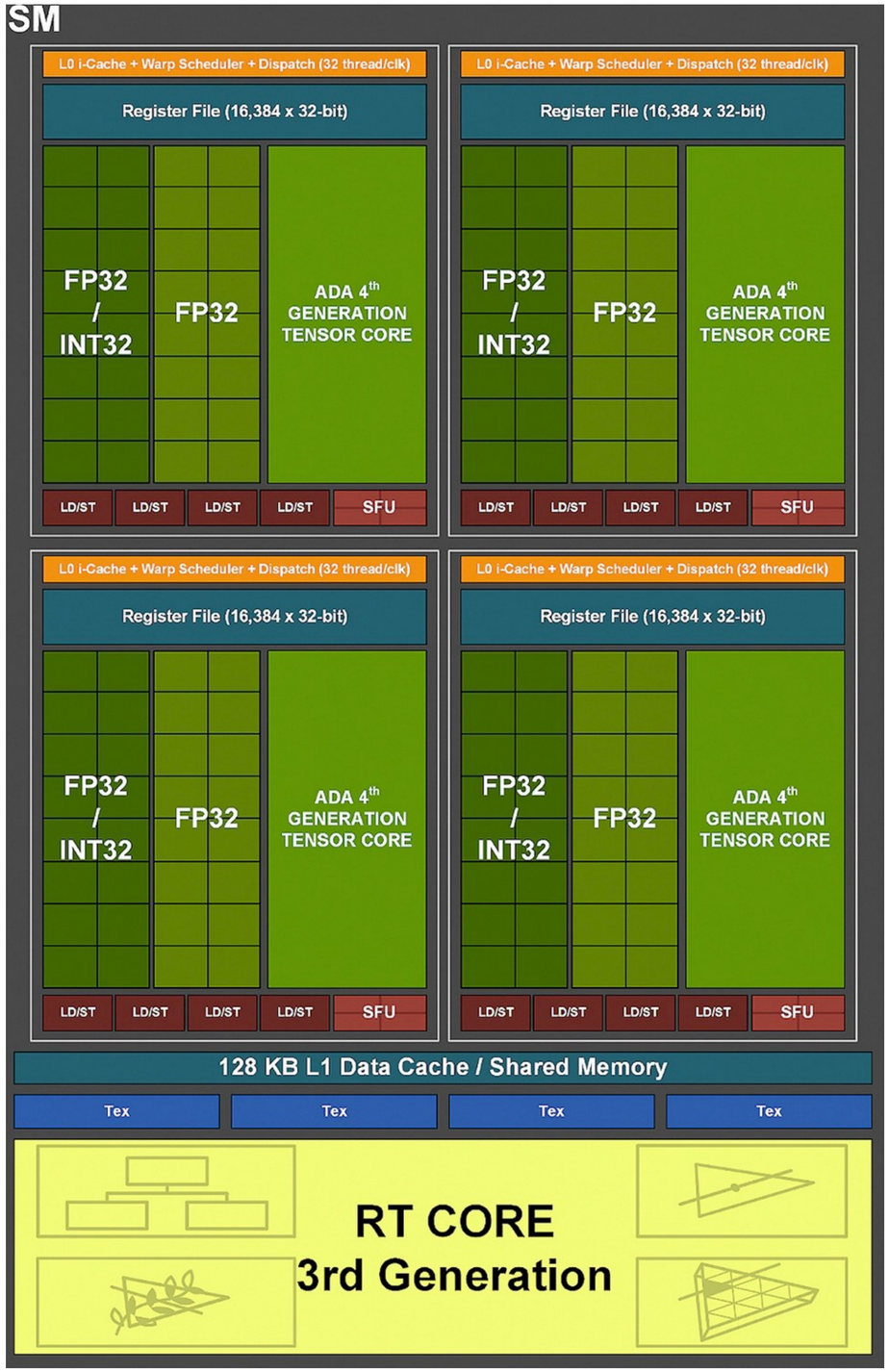
- https://cs.calvin.edu/courses/cs/374/CUDA/CUDA-Thread-Indexing-Cheatsheet.pdf
- https://en.wikipedia.org/wiki/Thread_block

CUDA Kernel – indexing Thread Hierarchy
(based on Bart VB slides)

An SM can only have a certain number of threads active at the same time (usually 2048)

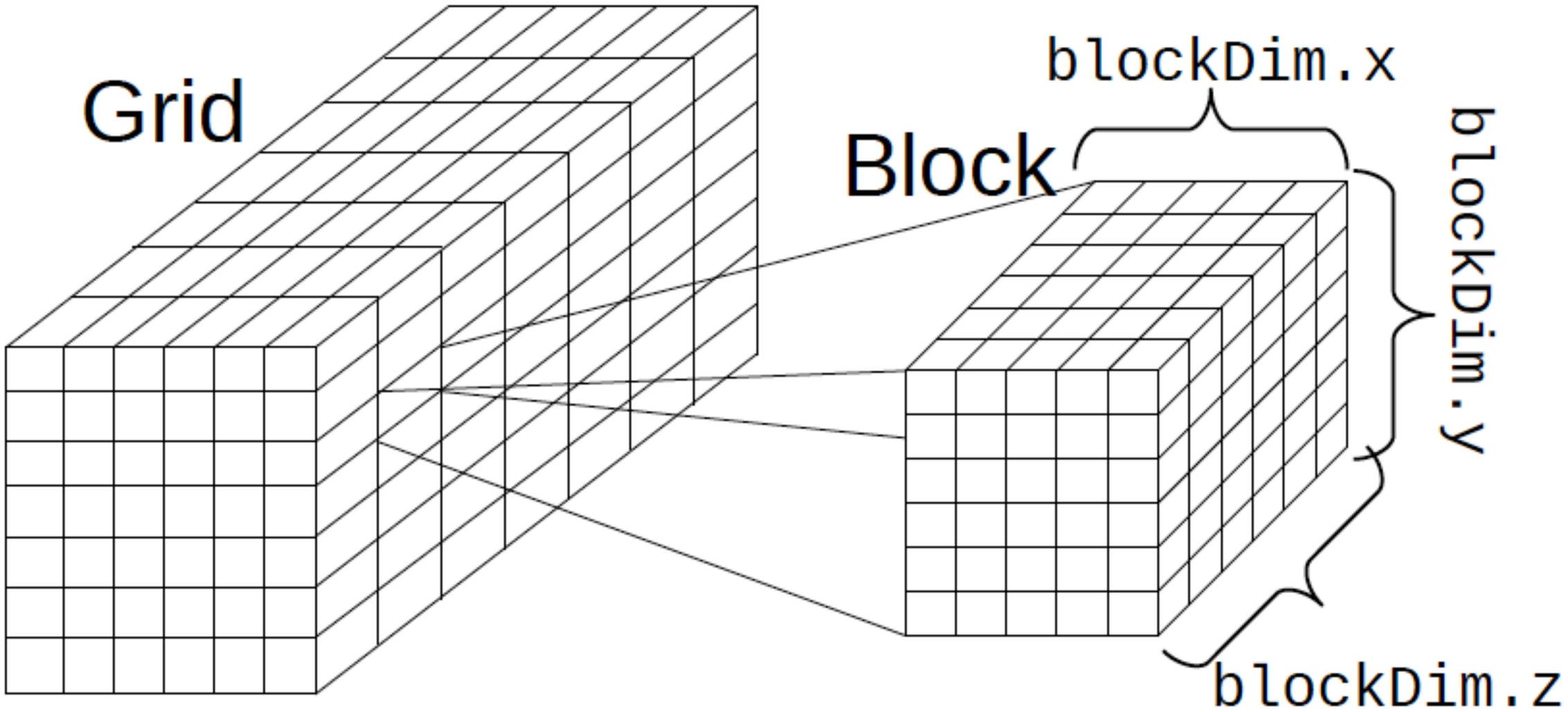
- Our problems are usually much larger, and we tend to use many threads

→ Solution: thread hierarchy



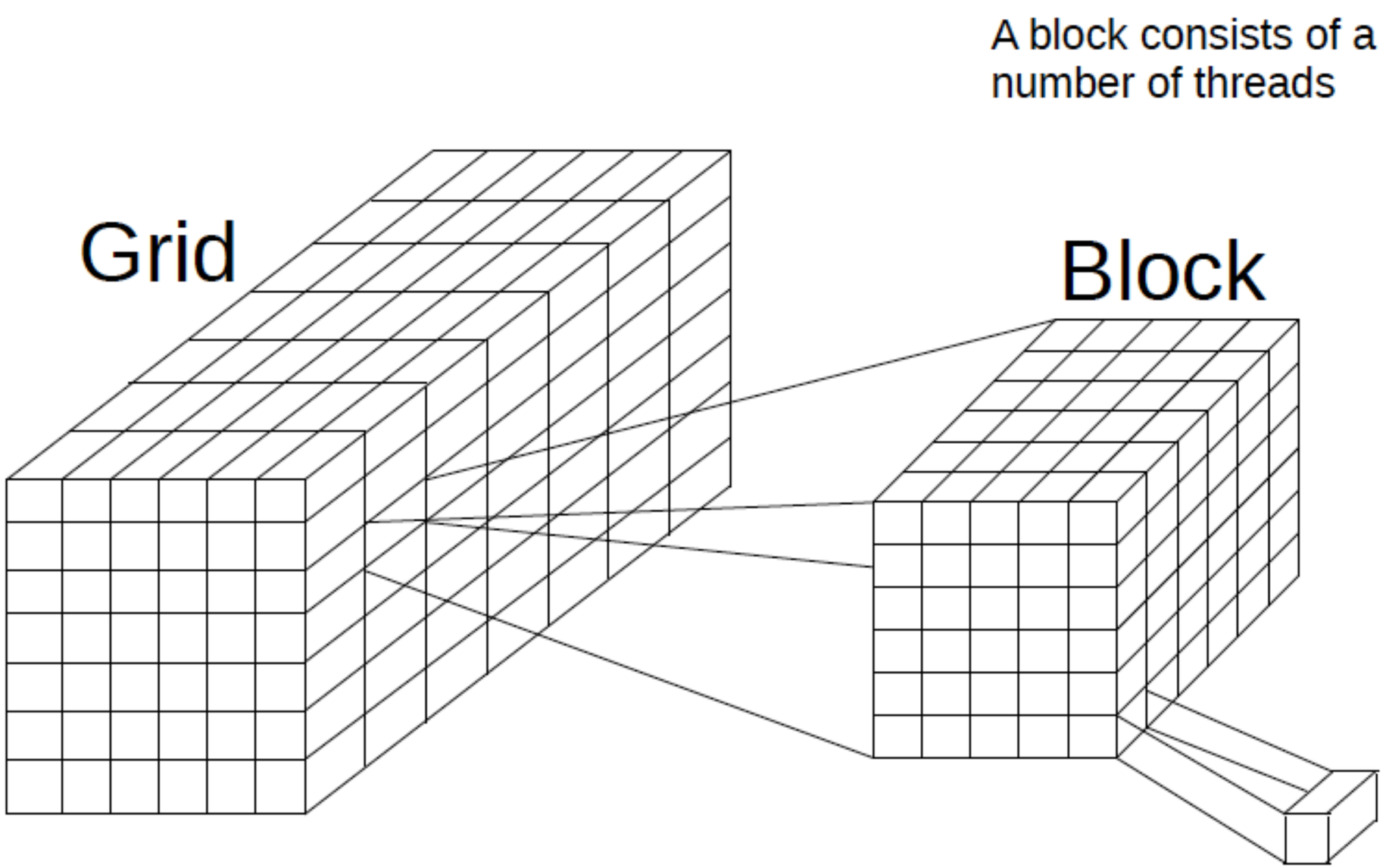
CUDA Kernel – indexing
Thread Hierarchy
(based on Bart VB slides)

A block is a chunk of our grid that is small enough to fit within an SM



CUDA Kernel – indexing
Thread Hierarchy

(based on Bart VB slides)



CUDA Kernel - indexing
(based on CalTech slides)

```
__global__ void
cudaAddVectorsKernel(float * a, float * b, float * c) {
    unsigned int index = blockIdx.x * blockDim.x + threadIdx.x;
    c[index] = a[index] + b[index];
}
```

- https://cs.calvin.edu/courses/cs/374/CUDA/CUDA-Thread-Indexing-Cheatsheet.pdf
- https://en.wikipedia.org/wiki/Thread_block

CUDA Kernel – invocation (calling the kernel)

(based on CalTech slides)

```
void cudaAddVectors(const float* a, const float* b, float* c, size){
    //For now, suppose a and b were created before calling this function

    // dev_a, dev_b (for inputs) and dev_c (for outputs) will be
    // arrays on the GPU.

    float * dev_a;
    float * dev_b;

    float * dev_c;

    // Allocate memory on the GPU for our inputs:
    cudaMalloc((void **) &dev_a, size*sizeof(float));
    cudaMemcpy(dev_a, a, size*sizeof(float), cudaMemcpyHostToDevice);

    cudaMalloc((void **) &dev_b, size*sizeof(float)); // and dev_b
    cudaMemcpy(dev_b, b, size*sizeof(float), cudaMemcpyHostToDevice);


    // Allocate memory on the GPu for our outputs:
    cudaMalloc((void **) &dev_c, size*sizeof(float));
```

CUDA Kernel – invocation (calling the kernel)

(based on CalTech slides)

```
    //At lowest, should be 32
    //Limit of 512 (Tesla), 1024 (newer)
    const unsigned int threadsPerBlock = 512;

    //How many blocks we'll end up needing
    const unsigned int blocks = ceil(size/float(threadsPerBlock));

    //Call the kernel!
    cudaAddVectorsKernel<<<blocks, threadsPerBlock>>>
        (dev_a, dev_b, dev_c);

    //Copy output from device to host (assume here that host memory
    //for the output has been calculated)

    cudaMemcpy(c, dev_c, size*sizeof(float), cudaMemcpyDeviceToHost);

    //Free GPU memory
    cudaFree(dev_a);
    cudaFree(dev_b);
    cudaFree(dev_c);
}
```

NTNU

CUDA applications
(covered in CalTech course)

- Solving PDEs on GPUs
- GPU vs CPU fluid mechanics
- Ray Traced Quaternion fractals and Julia Sets
- Deep Learning and GPUs
- Real-Time Signal Processing with GPUs

TDT4200 Fall 2024 Student Reference Group

Please contact for praise and suggestions on how we can make this course better:

Ingar Asheim – ingara@stud.ntnu.no

Elin Skålund Berntsen – elinsbe@stud.ntnu.no
Hanne Marie Haakaas – hannamhaa@stud.ntnu.no
Suzanne Maduga – suzannm@stud.ntnu.no

Simone Deidier – simondei@stud.ntnu.no
Davide Esposito – davidees@stud.ntnu.no
Diego Velasco – diegove@stud.ntnu.no
???
???

Parallel Computing is Fun!