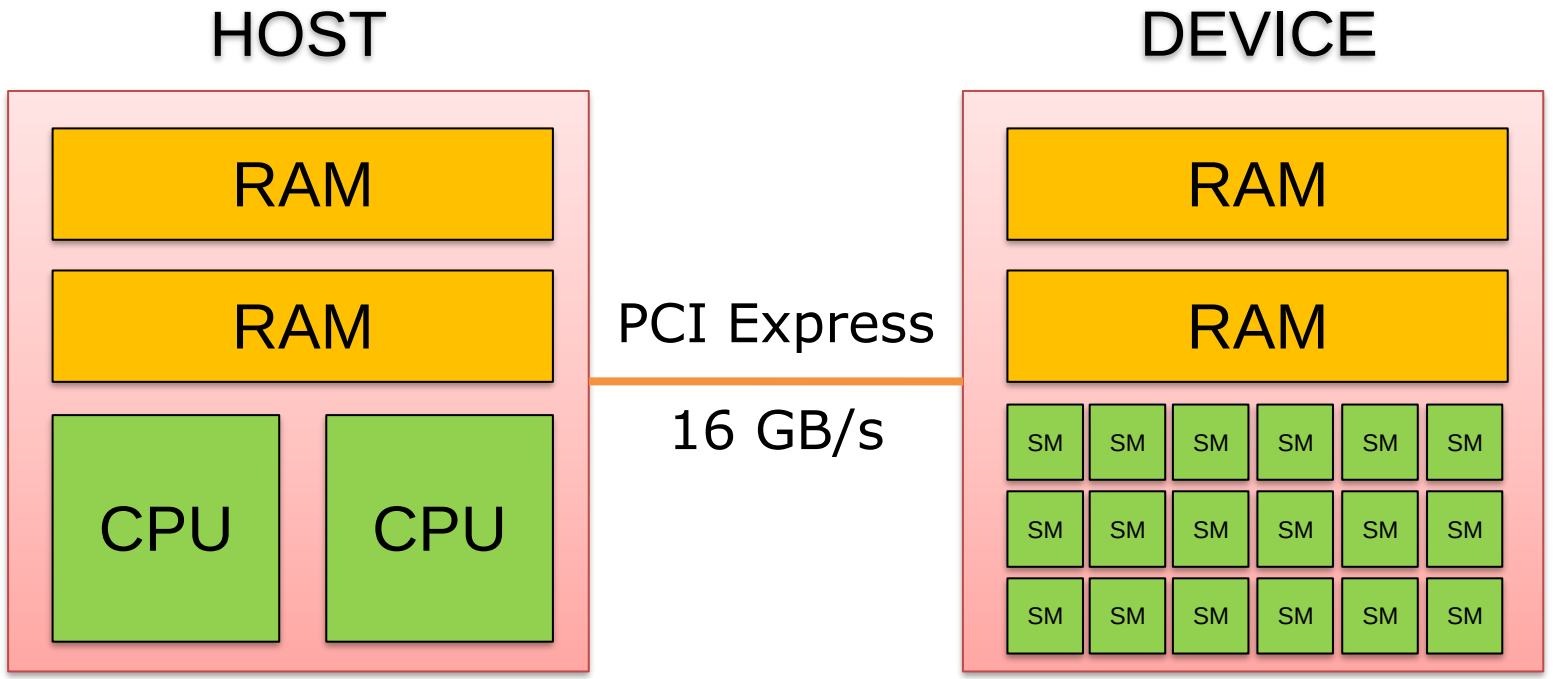
Parallel Computing for Mobile Robots (MA-INF 4226)

Sicong Pan, Benno Wingender

Prof. Dr. Maren Bennewitz

Humanoid Robots Lab, University of Bonn

Parallel Computing with a GPU



CPU

12 cores, 3.4 GHz

64GB RAM

GPU

2560 cores, 1.6 GHz

8GB RAM

Lab Overview

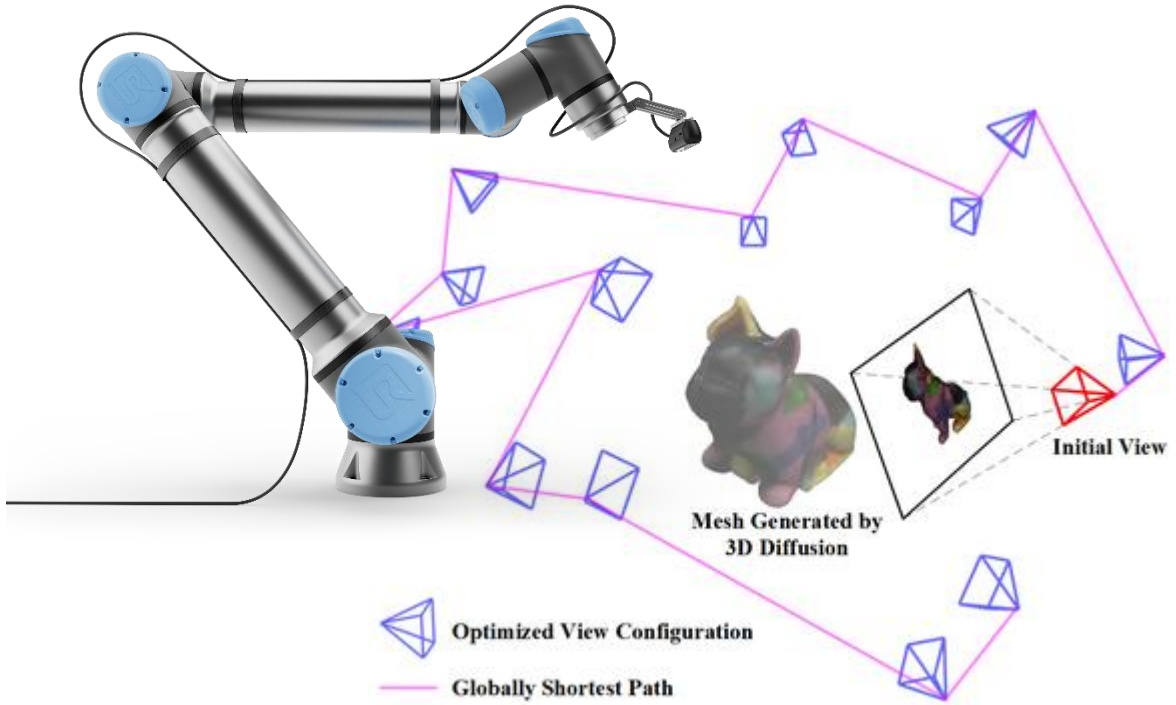
- Learn parallel programming with CUDA framework
- Implement vision related algorithms on the GPU
- Compare Cuda implementations against CPU versions
- Apply the vision algorithms on real robots

Agenda

- CUDA Tutorial
- Profiling: Measure and understand speed of data transfers and memory access
- Parallelization of basic algorithms (vector addition, max finding)
- Parallelization of Chamfer Distance computation for point clouds
- Parallelization of Ray Casting in voxel grids for model-based next-best-view selection in mobile robotics

Agenda Real World

- Apply the vision algorithms on real robots
- Graded Lab presentation about the project



Workflow

- Small groups of 2 or 3 people
- Assignments communicated by email
- Communication by email, zoom and in person meetings for your questions possible
- Test code on our lab computers through remote access
- Submission of assignments via git before deadlines
  - https://gitlab.igg.uni-bonn.de
- Presentation of the project and algorithm
- Final oral exam with questions related to the tasks

CUDA-Basics

- CUDA extends C++
- Supports kernels with __global__ prefix
- Kernels called from host, executed on device
- Number of blocks and threads per block specified on call
- Can be three-dimensional

```
__global__ void processData(double *x, uint64_t n) {
    int startIdx = blockIdx.x * blockDim.x + threadIdx.x;
    for (uint64_t i = startIdx; i < n; i += blockDim.x * gridDim.x)
        x[i] *= 2;
}

processData<<<NUM_BLOCKS, THREADS_PER_BLOCK>>>(data, N);
```

Memory Management

- CUDA provides its own memory allocation functions
- cudaMalloc → allocate memory on device
- cudaMallocHost → allocate memory on host
- cudaMallocManaged → automatically managed memory
- cudaMemcpy to copy between device and host

- More details: https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__MEMORY.html

Compilation

- nvcc compiler: nvcc -std=c++11 main.cu other.cpp -o main
- For your tasks: Provide compilation instructions / makefile
- Use of CMake possible

```
cmake_minimum_required(VERSION 3.24)
set(CMAKE_CXX_STANDARD 14)
set(CMAKE_CUDA_ARCHITECTURES native)
set(CMAKE_CUDA_SEPARABLE_COMPILATION ON)
project(CudaLab VERSION 0.1 LANGUAGES CXX CUDA)
find_package(CUDAToolkit)
include_directories(${CUDAToolkit_INCLUDE_DIRS})
add_executable(main main.cu)
```

Next steps

- Registration in BASIS until 26nd April
- Send email to pan@cs.uni-bonn.de
  - gsg account, gitlab account, group preferences…
- Notification and instructions from our side
- First assignment after registration
- Check out CUDA C++-Guide: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html