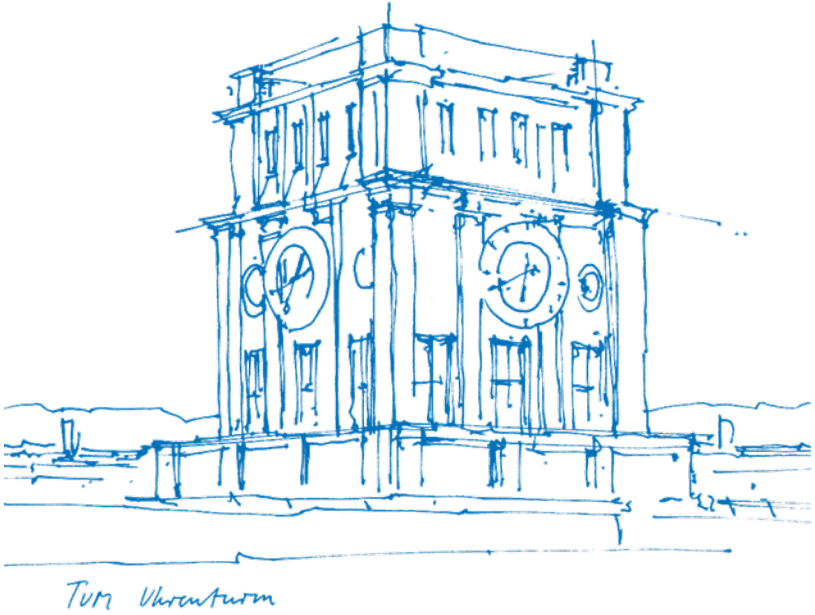
Tutorial 02 – C++ Threaded Programming

Dennis-Florian Herr, M.Sc.
Chair for Computer Architecture and Parallel Systems (CAPS)
Technical University Munich

May 07, 2024



Homework Solution

Assignment 1: VV-AES

General Ideas to optimize sequential code

- Improve algorithmic performance (think about asymptotic complexity / Big O)
- Think about the right data structure for the problem
- Optimize cache usage (cache access is orders of magnitude faster than main memory access)
- Reuse previously computed values (cf. dynamic programming)
- Consider precomputing an often recomputed small amount of data
- Do not reinvent the wheel (check *at least* the standard library for existing solutions)

Improve substitute_bytes

Slow approach by repeatedly searching through the originalCharacter array

Before

```
void substitute_bytes() {
  // For each byte in the message
  for (int column = 0; column < BLOCK_SIZE; column++) {
    for (int row = 0; row < BLOCK_SIZE; row++) {
      int index = -1;
      for (int i = 0; i < UNIQUE_CHARACTERS; i++) {
        if (originalCharacter[i] == message[row][column]) {
          index = i;
        }
      }
      message[row][column] = substitutedCharacter[index];
    }
  }
}
```

Improve substitute_bytes

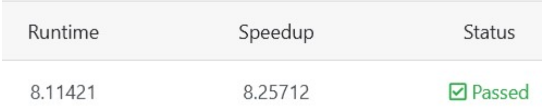
Create a lookup map reducing the search from up to 256 iterations to 1

Optimized

```
uint8_t substituteMap[UNIQUE_CHARACTERS];
void create_substitute_map(){
    for(int i = 0; i < UNIQUE_CHARACTERS; i++){
        substituteMap[originalCharacter[i]] = substitutedCharacter[i];
    }
}
void substitute_bytes() {
    // For each byte in the message
    for (int column = 0; column < BLOCK_SIZE; column++) {
        for (int row = 0; row < BLOCK_SIZE; row++) {
            message[row][column] = substituteMap[message[row][column] ];
        }
    }
}
```

Improve substitute_bytes

And we’re done.... (oops)



Runtime	Speedup	Status
8.11421	8.25712	☑ Passed

Improve shift_rows

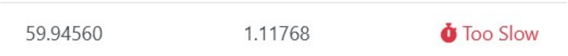
Don’t reinvent the wheel. Use the std library rotate function

optimized

```
#include <algorithm>
void shift_rows() {
    // Shift each row, where the row index corresponds to how many columns the data is shifted.
    for (int row = 0; row < BLOCK_SIZE; ++row) {
        std::rotate(std::begin(message[row]), std::begin(message[row]) + row, std::end(message[row]));
    }
}
```

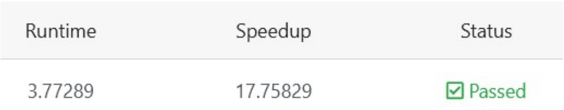
Improve shift_rows

Alone it's dwarfed by substitute_bytes



59.94560	1.11768	⏱ Too Slow

But together it's a significant boost



Runtime	Speedup	Status
3.77289	17.75829	☑ Passed

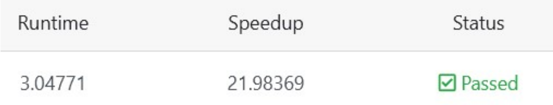
Improve mix_columns

Precompute all possible powers of 0 to 255

optimized

```c
int powers[256][BLOCK_SIZE + 1];
// precomputed powers of all possible message values (256)
for (int i = 0; i < 256; i++) {
    for (int j = 1; j <= BLOCK_SIZE; j++) {
        powers[i][j] = power(i, j);
    }
}
void mix_columns() {
    for (int column = 0; column < BLOCK_SIZE; ++column) {
        for (int row = 0; row < BLOCK_SIZE; ++row) {
            int result = 0;
            for (int degree = 0; degree < BLOCK_SIZE; degree++) {
                result += polynomialCoefficients[row][degree] * powers[message[degree][column]][degree + 1];
            }
            message[row][column] = result;
        }
    }
}
```

Improve substitute_bytes



Runtime	Speedup	Status
3.04771	21.98369	☑ Passed

Pushing further

Use CISC PSHUFB: Packed Shuffle Bytes

```c
uint8_t shift_row_mask[64] = {
    0, 1, 2, 3, 4, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 8,
    2, 3, 4, 5, 6, 7, 0, 1, 11, 12, 13, 14, 15, 8, 9, 10,
    4, 5, 6, 7, 0, 1, 2, 3, 13, 14, 15, 8, 9, 10, 11, 12,
    6, 7, 0, 1, 2, 3, 4, 5, 15, 8, 9, 10, 11, 12, 13, 14
};

inline void shift_rows() {
    __asm__ (
    "leaq    shift_row_mask(%rip), %rax\n"
    "vmovdqa message(%rip), %xmm0\n"
    "vpshufb (%rax), %xmm0, %xmm0\n"
    "vmovaps %xmm0, message(%rip)\n"
    "vmovdqa 16+message(%rip), %xmm1\n"
    "vpshufb 16(%rax), %xmm1, %xmm1\n"
    "vmovaps %xmm1, 16+message(%rip)\n"
    "vmovdqa 32+message(%rip), %xmm2\n"
    "vpshufb 32(%rax), %xmm2, %xmm2\n"
    "vmovaps %xmm2, 32+message(%rip)\n"
    "vmovdqa 48+message(%rip), %xmm3\n"
    "vpshufb 48(%rax), %xmm3, %xmm3\n"
    "vmovaps %xmm3, 48+message(%rip)\n"
    );
}
```

Leaderboard

A word on the leaderboard

- Friendly competition. Completely optional!
- Code/algorithm can be changed freely (as long as the result is accepted)
- ...but please don't try to hack the system!
- ...and don't write scripts to submit hundreds of times (I will delete and disqualify these)

How to compete?

- Find optimal parallelization
- Optimize sequential performance
- Skip unnecessary work
- Approximate solutions might be good enough!

Need for Speed

Need for Speed

Share your cool solutions (after the deadline)

`https://zulip.in.tum.de/#narrow/stream/2330-ParProg-24/topic/Need.20for.20Speed`

Theory

C++ Threads : Passing Data

Creating Threads

```
#include <thread>
std::thread thread_object( callable )
std::thread thread_object( callable , arguments)
```

C++ Threads : Passing Data

Creating Threads

```
#include <thread>
std::thread thread_object( callable )
std::thread thread_object( callable , arguments)
```

Pass by Pointer

Main

```
...
int argument = 1;
std::thread thread( kernel, &argument )
...
```

Kernel

```
void* kernel( int* args) {
   ++(*args);
     ...
}
```

C++ Threads : Passing Data

Creating Threads

```
#include <thread>
std::thread thread_object( callable )
std::thread thread_object( callable , arguments)
```

Pass by Pointer

Main

```
...
int argument = 1;
std::thread thread( kernel, &argument )
...
```

Kernel

```
void* kernel( int* args) {
    ++(*args);
      ...
}
```

Pass by Reference

Main

```
      ...
int argument = 1;
std::thread thread( kernel, std::ref(argument) )
...
```

Kernel

```
void* kernel( int &args) {
     ++args;
      ...
}
```

C++ Threads: Join

Joining Threads

```
#include <thread>
std::thread::join()
```

C++ Threads: Join

Joining Threads

```
#include <thread>
std::thread::join()
```

Example

Main

```
...
// Fork a thread
std::thread t(my_kernel);

// Join the thread
t.join();
...
```

Kernel

```
void my_kernel () {
    sleep(1);
    ...
}
```

C++ Threads: Pitfalls

Example 1

Main

```
...
// Start numThreads threads
std::thread threads[numThreads];
for (int i = 0 ; i< numThreads ; ++i){
    threads[i] = std::thread(oops_kernel, &i);
}
for (int i = 0 ; i< numThreads ; ++i){
    threads[i].join();
}
...
```

Kernel

```
void oops_kernel (int *args) {
    int argument = *args;
    ...
}
```

C++ Threads: Pitfalls

Example 1

Main

```
...
// Start numThreads threads
std::thread threads[numThreads];
for (int i = 0 ; i< numThreads ; ++i){
    threads[i] = std::thread(oops_kernel , &i);
}
for (int i = 0 ; i< numThreads ; ++i){
    threads[i].join();
}
...
```

Kernel

```
void oops_kernel (int *args) {
    int argument = *args;
    ...
}
```

✗ Use of freed memory / Race Condition

C++ Threads: Pitfalls

Example 1

Main

```
...
// Start numThreads threads
std::thread threads[numThreads];
for (int i = 0 ; i< numThreads ; ++i){
    threads[i] = std::thread(oops_kernel, &i);
}
for (int i = 0 ; i< numThreads ; ++i){
    threads[i].join();
}
...
```

Kernel

```
void oops_kernel (int *args) {
    int argument = *args;
    ...
}
```

✗ Use of freed memory / Race Condition

Kernel

```
void wicked_kernel (int *args) {
    (*args) = 0;
    ...
}
```

C++ Threads: Pitfalls

Example 2

Main

```
...
// Start numThreads threads
int ids[numThreads];
std::thread threads[numThreads];
for (int i = 0 ; i< numThreads ; ++i){
    ids[i] = i;
    threads[i] = std::thread(a_kernel, &ids[i]);
}
for (int i = 0 ; i< numThreads ; ++i){
    threads[i].join();
}
...
```

Kernel

```
void a_kernel (int* args) {
    int argument = *args;
    ...
}
```

C++ Threads: Pitfalls

Example 2

Main

```cpp
...
// Start numThreads threads
int ids[numThreads];
std::thread threads[numThreads];
for (int i = 0 ; i< numThreads ; ++i){
    ids[i] = i;
    threads[i] = std::thread(a_kernel, &ids[i]);
}
for (int i = 0 ; i< numThreads ; ++i){
    threads[i].join();
}
...
```

Kernel

```cpp
void a_kernel (int* args) {
    int argument = *args;
    ...
}
```

✓ No Problems

C++ Threads: Pitfalls

Example 3

Main

```
std::thread threads[numThreads];
void lets_spawn_threads(){

  int ids[numThreads];
  for (int i = 0 ; i< numThreads ; ++i){
     ids[i] = i;
     threads[i] = std::thread(noooo_kernel, &ids[i]);
  }
}

int main(){
  lets_spawn_threads();
  for (int i = 0 ; i< numThreads ; ++i){
     threads[i].join();
  }

}
```

Kernel

```
void noooo_kernel (int* args) {
    int argument = *args;
    ...
}
```

C++ Threads: Pitfalls

Example 3

Main

```
std::thread threads[numThreads];
void lets_spawn_threads(){

  int ids[numThreads];
  for (int i = 0 ; i< numThreads ; ++i){
     ids[i] = i;
     threads[i] = std::thread(noooo_kernel, &ids[i]);
  }
}

int main(){
  lets_spawn_threads();
  for (int i = 0 ; i< numThreads ; ++i){
     threads[i].join();
  }

}
```

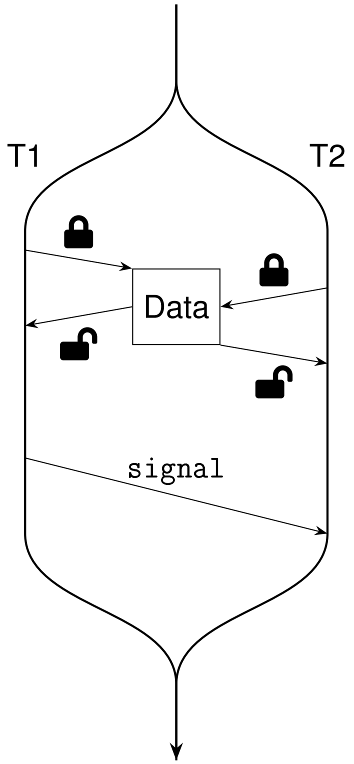
Kernel

```
void noooo_kernel (int* args) {
    int argument = *args;
    ...
}
```

✘ Use of freed memory

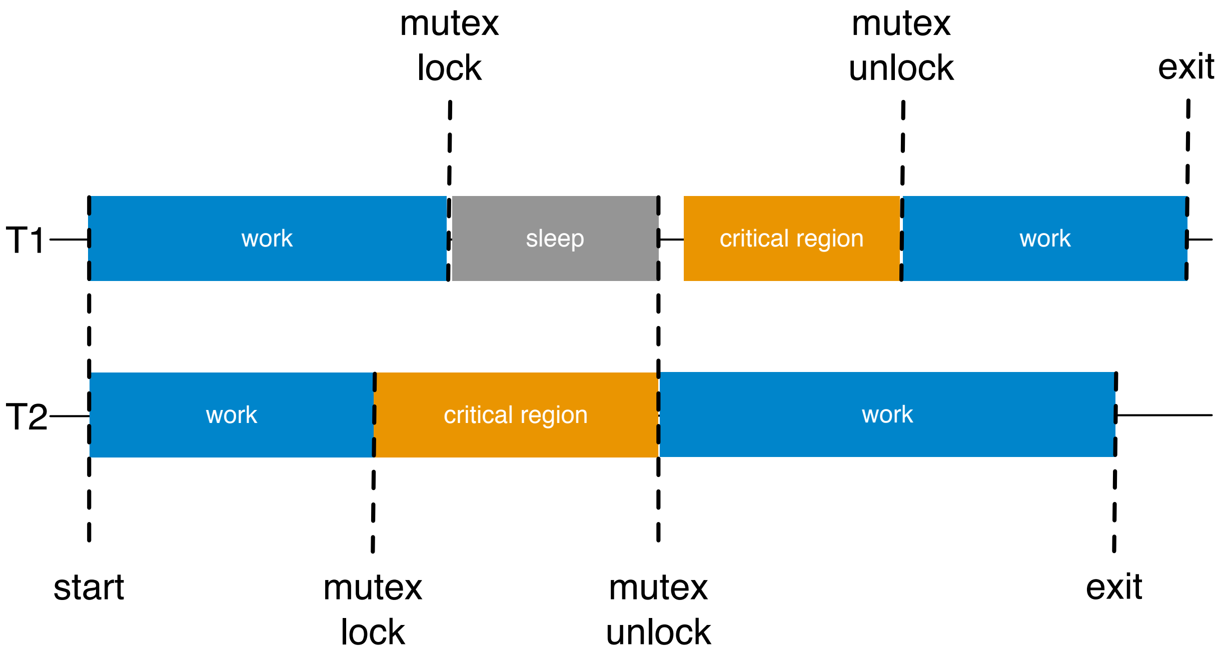
Synchronization

- Needed for accesses to shared resources
- Drawback: Overhead
- Frequent reasons for synchronizing:
  - Prevention of concurrent access
  - Signal passing
- C++ standard library provides following mechanisms (not exhaustive):
  - Mutexes
  - Condition Variables
  - Barriers (not covered)
  - Semaphores (not covered)



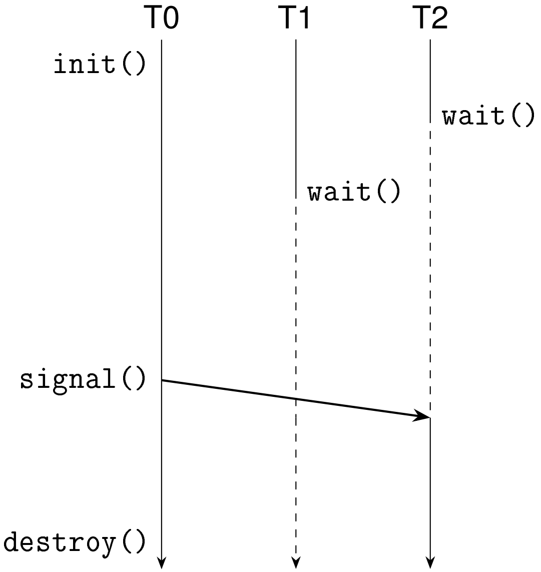
Mutex (Mutual Exclusion Lock)

- The simplest and most primitive synchronization method
- Uses atomic (hardware) operations
- Ensures absolute owner of (critical) code section
- Threads can lock and unlock mutexes

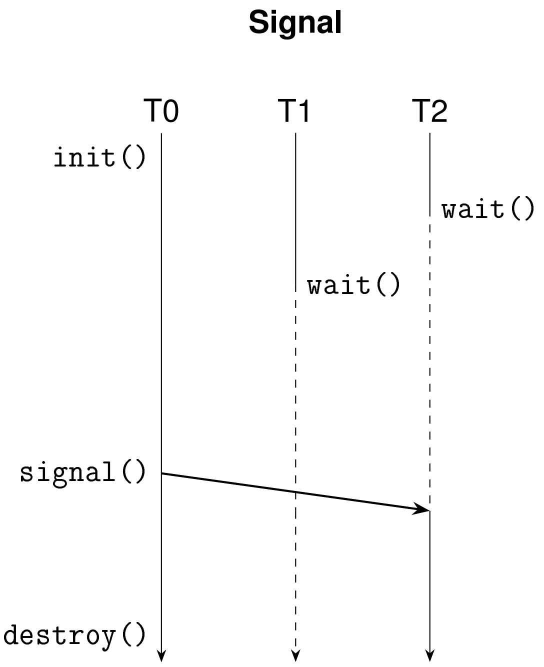


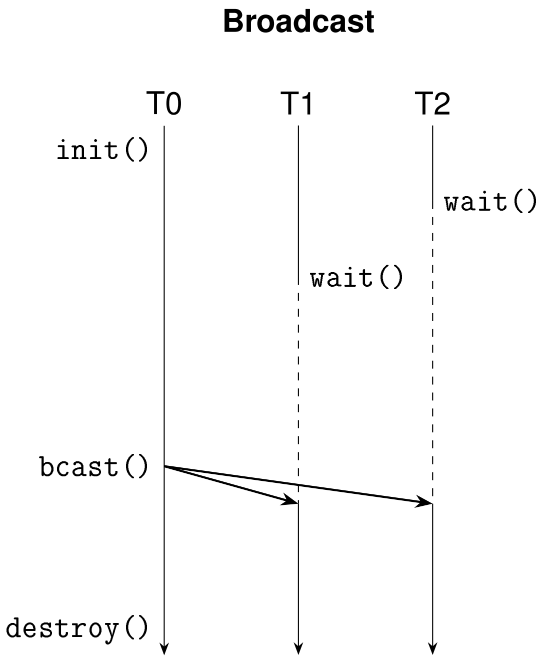
Condition Variables – Signaling

Signal



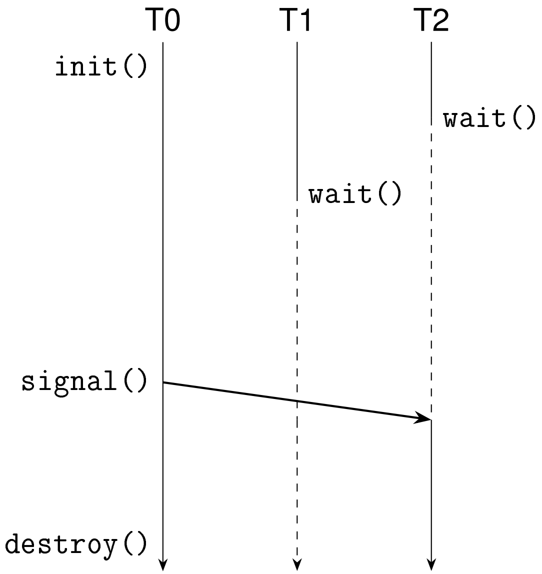
Condition Variables – Signaling



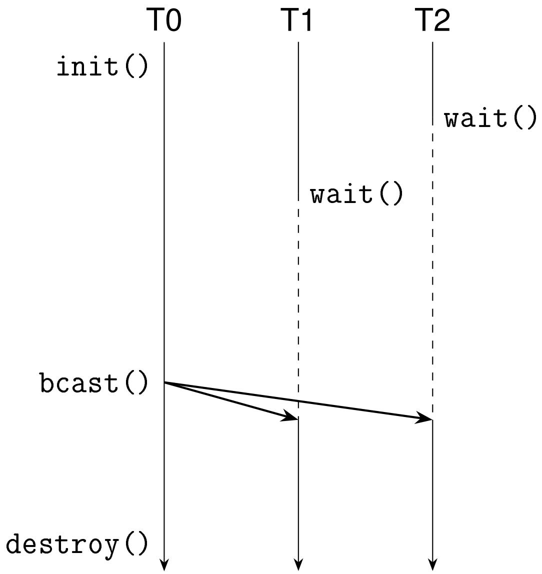


Condition Variables – Signaling

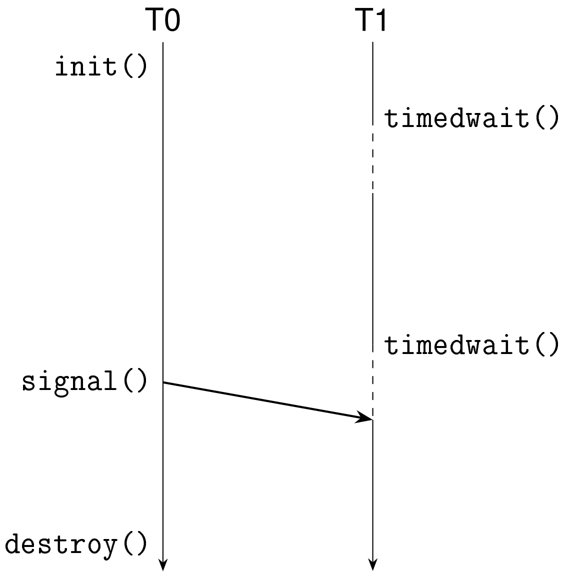
Signal



Broadcast



Timed Wait



Work Distribution Pitfalls

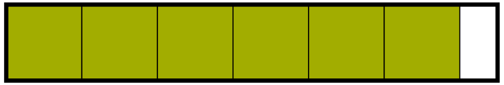
Undercounting

```
int blockSize = SIZE / NUM_THREADS;




for(int i = 0; i < blockSize; i++) {
    int dataIndex = threadId * blockSize + i;
    ...
}
...
```

Data

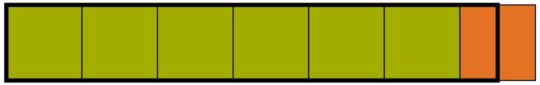


Overcounting

```
int blockSize = 8;
int myBlock = 0;

while(myBlock * blockSize < SIZE) {
    myBlock = getNextBlock();
    for(int i = 0; i < blockSize; i++) {
        int dataIndex = myBlock * blockSize + i;
        ...
    }
}
```

Data



Quiz

Moodle Quiz

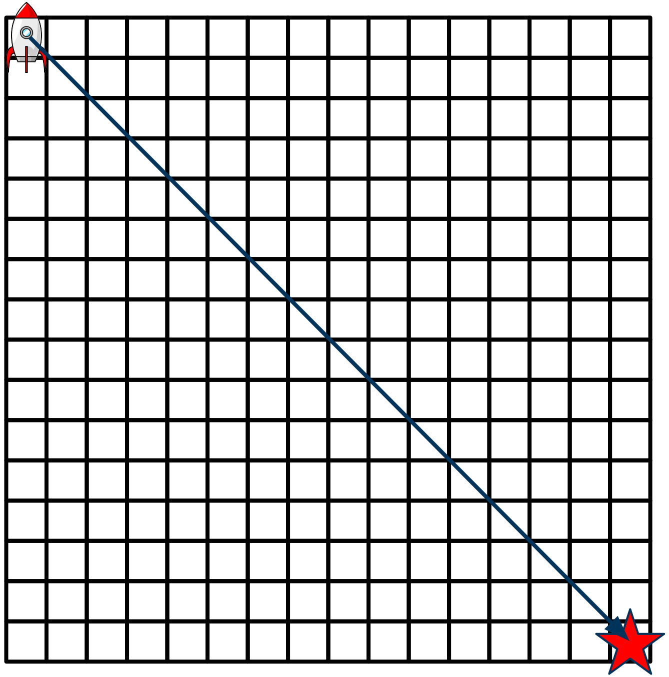
`https://www.moodle.tum.de/mod/quiz/view.php?id=2976059`

In-Class Exercise

Help to calculate the starship’s damage

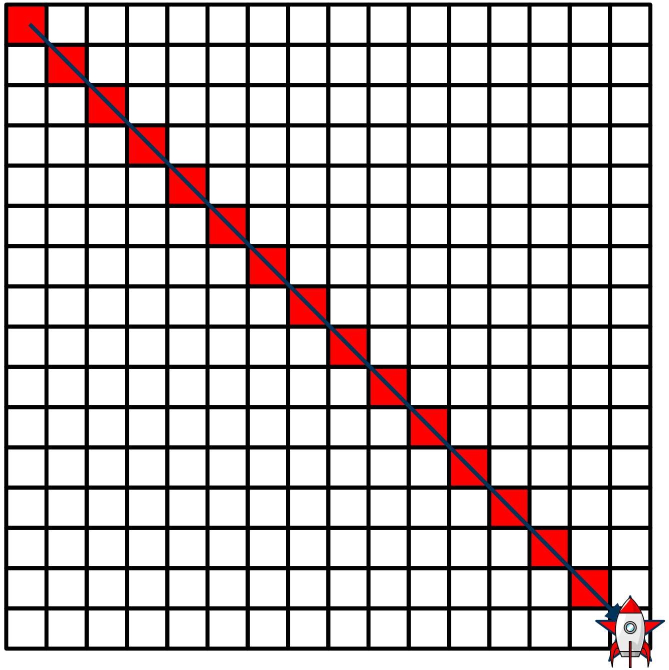
The classic asteroids game

Help to calculate the starship's damage



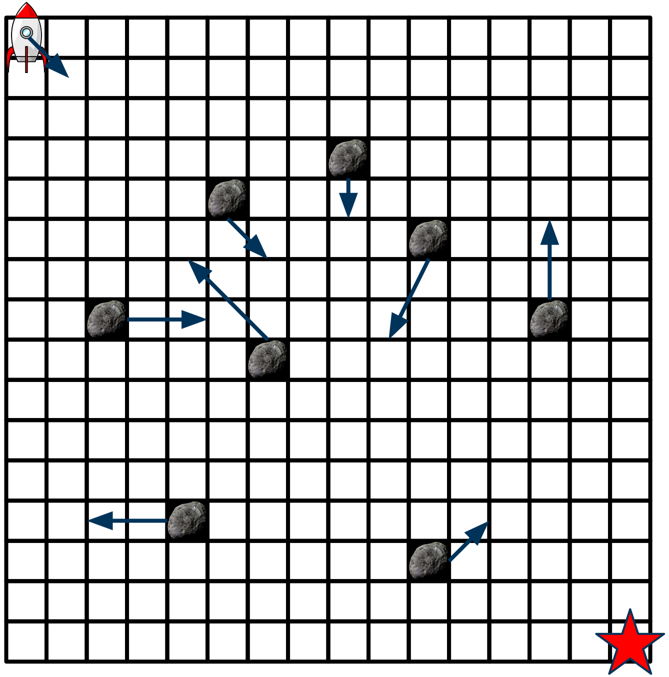
Our starship wants to go to the goal along the diagonal of the grid map

Help to calculate the starship’s damage



The red blocks are the intended trajectory of the starship

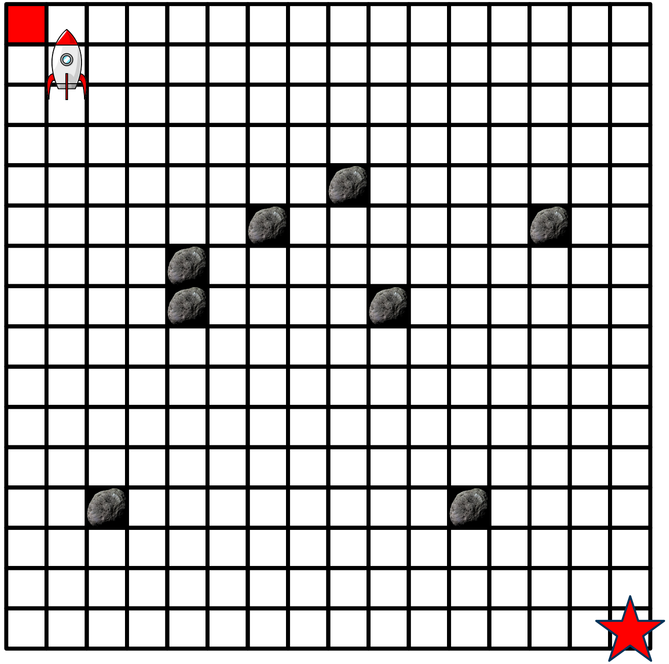
Help to calculate the starship's damage



But there are many rocks floating in the space, having very complicated velocity function

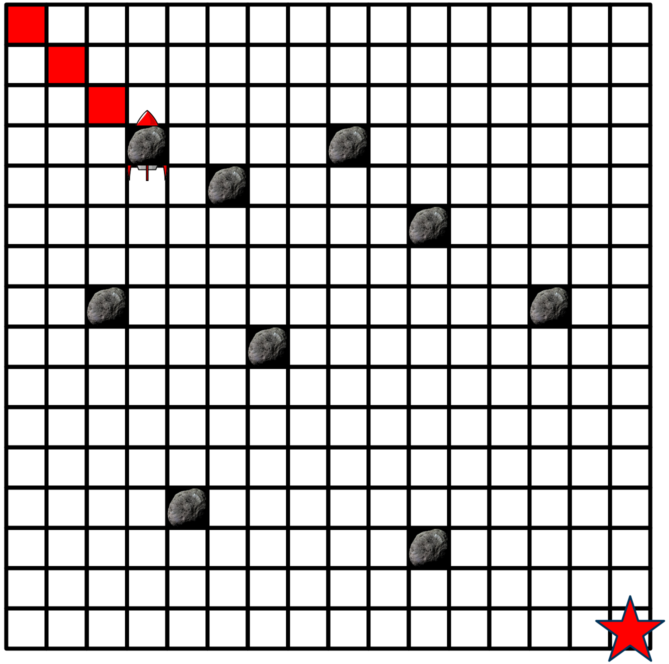
We can call *compute_vel( )* function to calculate the velocity of a rock

Help to calculate the starship’s damage



At every time step, all the rocks in the map and the starship will update their location
We can call *update_rock( )* to update a rock’s locations

Help to calculate the starship's damage

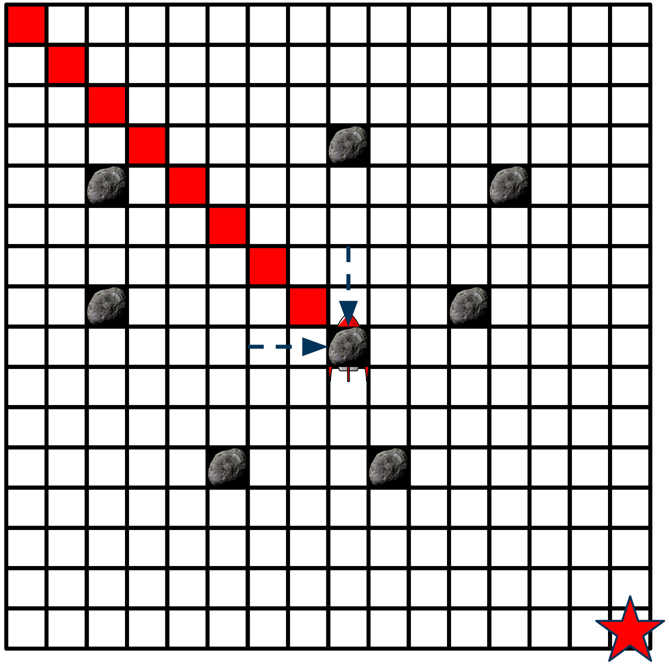


hits+1!

When our starship moves towards the goal, it could crash rocks

We can call *calc_hits( )* to calculate how many rocks hit the starship at this time step

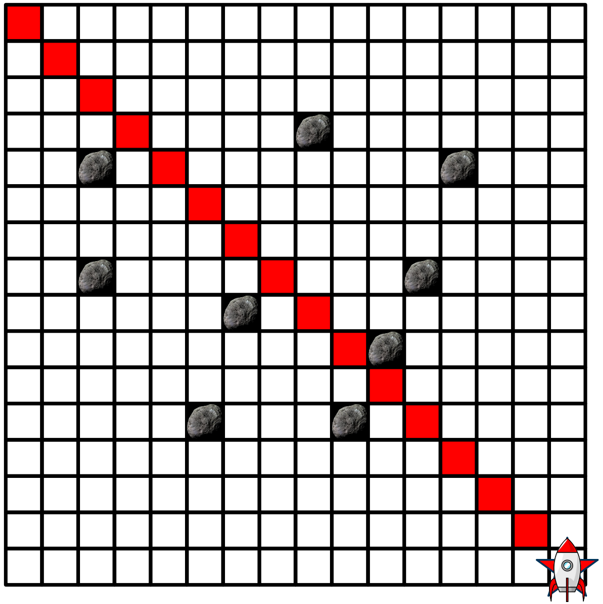
Help to calculate the starship's damage



hits+2!

Several rocks can hits our starship at the same time

Help to calculate the starship’s damage



Help us to calculate the total hits number faster!
Although we implemented the sequential way to calculate this, but it is too slow!

Use C++ standard library threads to help us calculate this faster !

Where to find the exercise ?

- Go to the following repository to get the exercise:
  `https://gitlab.lrz.de/lrr-tum/teaching/parprog/ss2024/published-assignments`
- Use git to clone the exercise to your local machine:
  cd your_folder
  git clone `https://gitlab.lrz.de/lrr-tum/teaching/parprog/ss2024/published-assignments.git`
- You can pull from this repository every time a new exercise is published
- Go to folder in-class-2 for this week's task, you can also find a README.md there

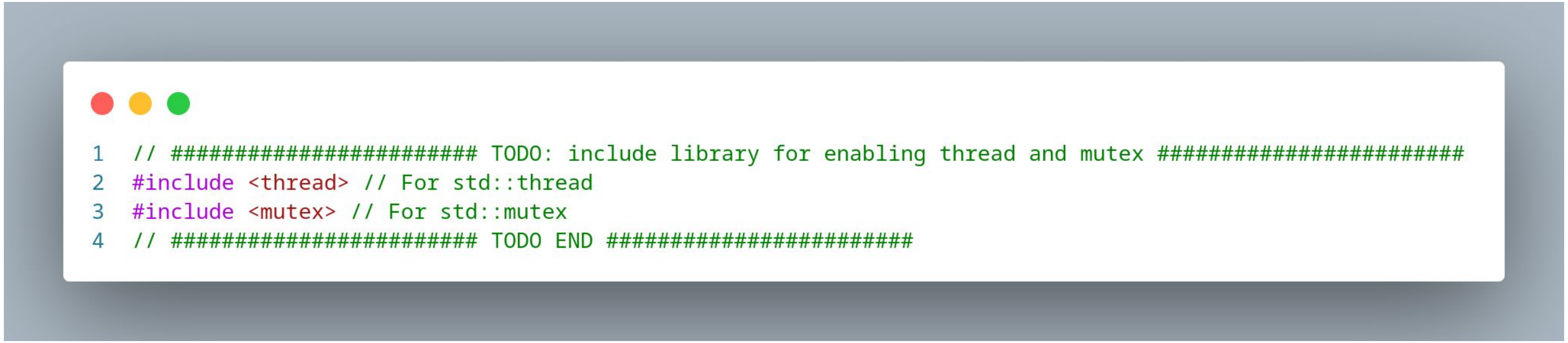
Week 2 exercise introduction:

Use C++ standard library threads

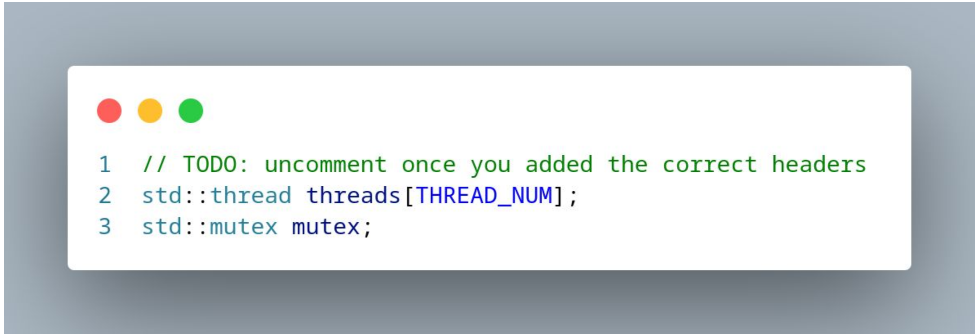
- You will find `student_submission.cpp` the partially implemented parallel code
- Complete the //TODO section to
  - include necessary library
  - create threads for parallelization (pass called function, arguments etc.)
  - join threads to terminate parallelization
  - use mutex locks to avoid data racing
- achieve a speed up of 12 on the sever.
- Our server has 16 cores and 2 way hyperthreading (i.e. 32 threads)

In-Class Exercise: Solution

TODO#1: Include thread and mutex

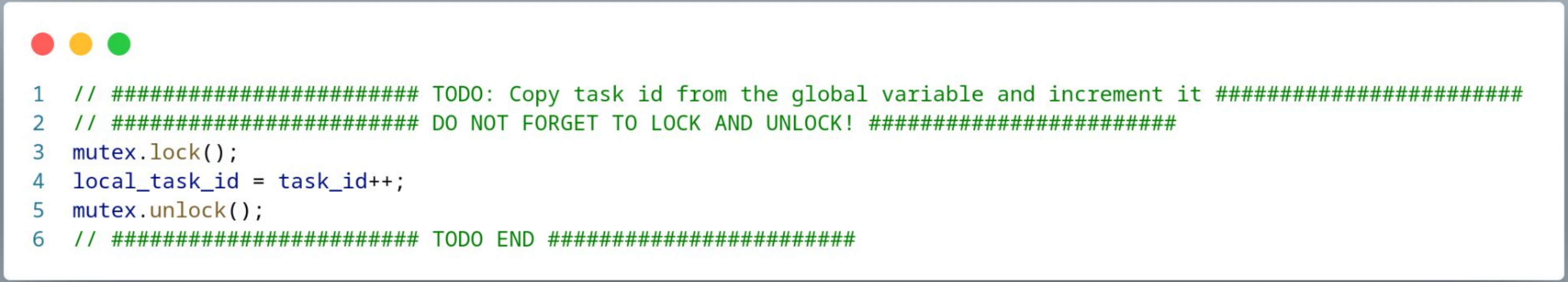


```
// ######################## TODO: include library for enabling thread and mutex ########################
#include <thread> // For std::thread
#include <mutex> // For std::mutex
// ######################## TODO END ########################
```



```
// TODO: uncomment once you added the correct headers
std::thread threads[THREAD_NUM];
std::mutex mutex;
```

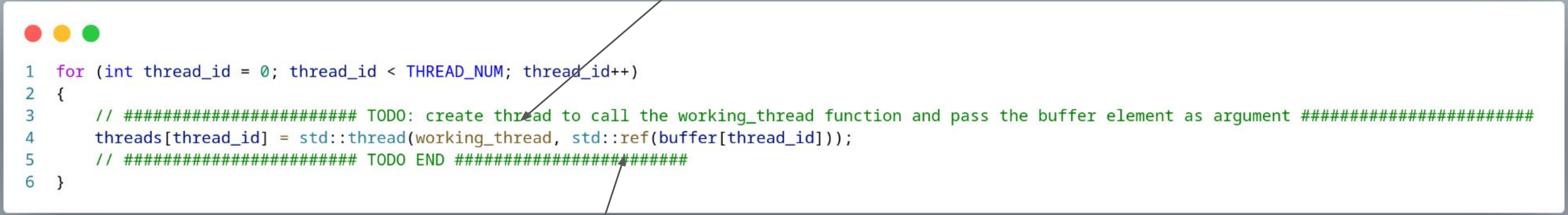
TODO#2: Implement synchronized access to global variable



```
// ######################## TODO: Copy task id from the global variable and increment it ########################
// ######################## DO NOT FORGET TO LOCK AND UNLOCK! ########################
mutex.lock();
local_task_id = task_id++;
mutex.unlock();
// ######################## TODO END ########################
```

TODO#3: Create the threads

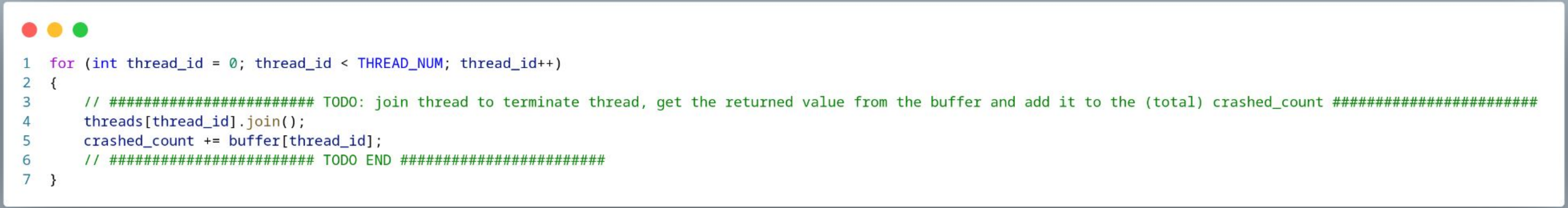
Thread kernels need to have return type void!



```cpp
for (int thread_id = 0; thread_id < THREAD_NUM; thread_id++)
{
    // ######################## TODO: create thread to call the working_thread function and pass the buffer element as argument ########################
    threads[thread_id] = std::thread(working_thread, std::ref(buffer[thread_id]));
    // ######################## TODO END ########################
}
```

Ensures we pass the reference and avoids (rare) compile errors

TODO#4: Wait for threads to complete and accumulate result



```
for (int thread_id = 0; thread_id < THREAD_NUM; thread_id++)
{
    // ######################## TODO: join thread to terminate thread, get the returned value from the buffer and add it to the (total) crashed_count ########################
    threads[thread_id].join();
    crashed_count += buffer[thread_id];
    // ######################## TODO END ########################
}
```

Homework

Assignment: Raytracer

- Task: Render a 3D scene consisting of metallic spheres using raytracing
  - The program creates a random scene based on the seed read from `stdin`.
- The renderer sends rays for each pixel.
- When a ray hits a surface it gets reflected. A ray can only be reflected a certain number of times.
- Color of the pixel is determined by the materials of the surfaces a ray hits.
- Each pixel is sampled multiple times to reduce noise.

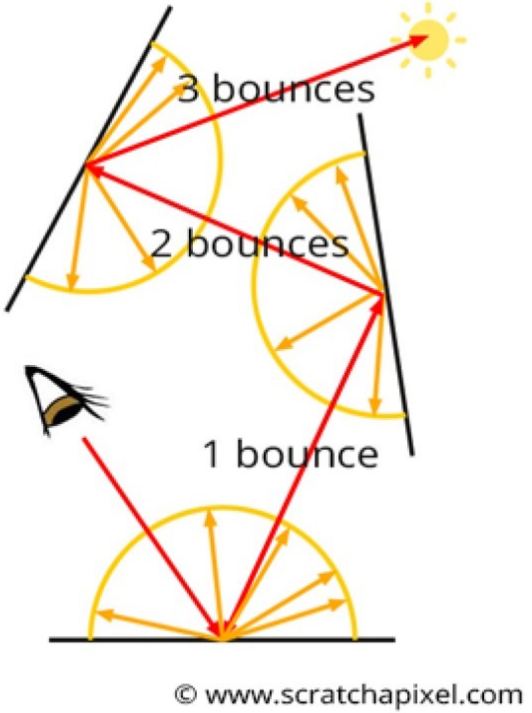


Figure 1: Rendering with a raytracer.

- Parallelize the sequential implementation with C++ threads.
- Your speedup should be ≥ 10.
- You don't have to read `maths.h` or `raytracer.h` to work on your solution.
- Evaluation command: `./student_submission -n «< <seed>`
- The output is a PPM image file.
  - Try xdg-open or use `http://paulcuth.me.uk/netpbm-viewer` to view PPM files.

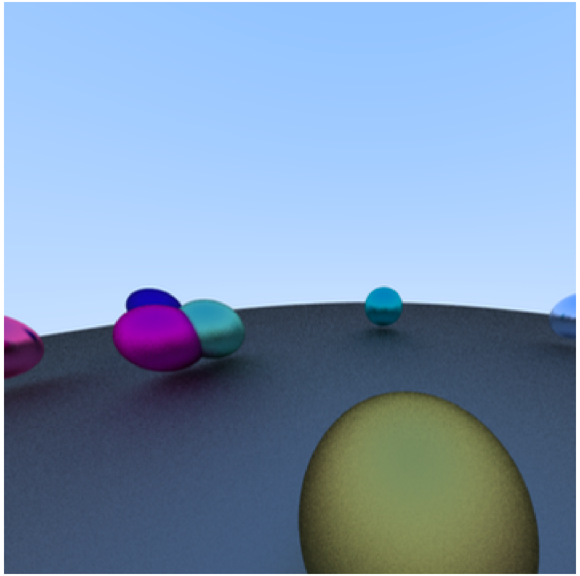


Figure 2: Raytraced scene.

Recap & Questions

Covered today:

- Launching & Joining Threads
- Synchronization

Questions

This slide is intentionally left blank.