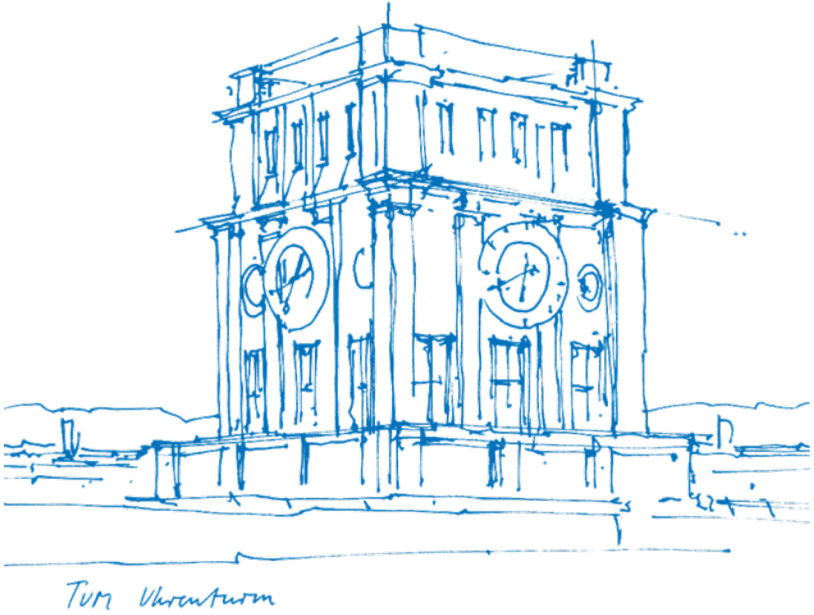
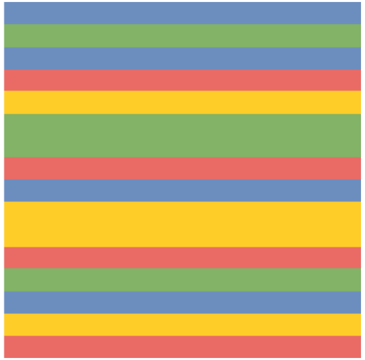
Parallel Programming Tutorial - OpenMP Basics

Amir Raoofy, M.Sc.
Chair for Computer Architecture and Parallel Systems (Prof. Schulz)
Technichal University Munich
2. Mai 2018



Solution for Assignment 2

Solution for Assignment 2



```c
typedef struct {
    void *image;
    int chunk_size, max_iter;
    int x_resolution, y_resolution;
    double view_x0, view_x1, view_y0, view_y1;
    double x_stepsize, y_stepsize;
    int palette_shift;
} compute_args;
```

```c
static int global_start;
#define CHUNK_SIZE 8
pthread_mutex_t mtx = PTHREAD_MUTEX_INITIALIZER;

void mandelbrot_draw( args ...) {
    global_start = 0;
    int i, t;

    pthread_t *thread = (pthread_t*)malloc\
                      (num_threads * sizeof(*thread));
    compute_args *args = (compute_args*)\
                       malloc(num_threads * sizeof(*args));

    for (t=0; t < num_threads; t++) {
        args[t].image= (void*) image;
        args[t].chunk_size = CHUNK_SIZE;
        // ... similar for other parameters ...
        pthread_create(&thread[t], NULL, kernel, &args[t]);
    }
    for (t = 0; t < num_threads; t++)
        pthread_join(thread[t], NULL);

    free(thread); free(args);
}
```

Solution for Assignment 2 (Cont.)

```
void* kernel1(void* arguments){
    compute_args *args = (compute_args*) arguments;
    int chunk_size = args->chunk_size;
    unsigned char (*image)[x_resolution1][3]= (unsigned char (*)[x_resolution1][3])args->image;
    // ... same for the rest of arguments ...
    int start; //local variable
    for (;;) { //infinite loop
        pthread_mutex_lock(&mtx);
        if ( y_resolution - global_start < 1 ) { // if every row is processed unlock and come out
            pthread_mutex_unlock(&mtx); break;
        }
        start = global_start; global_start += chunk_size;  // set the start and increase global variable
        pthread_mutex_unlock(&mtx);
        if ( y_resolution - start < chunk_size ) // for the thread that works on the last chunk
            chunk_size = y_resolution - start;
        for (int i = start; i < start + chunk_size; i++)
        {
            for (int j = 0; j < x_resolution1; j++)
            {
                // ... calculation of pixels ...
            }
        }
    }
}
```

Hints for Assignment 3

Hints for Assignment 3

- Use a profiler! (see last session)
- Try to reduce the critical region. **That is the bottleneck!**
- Use `std::ref()` to pass arguments by reference to a task function
- Use the launch policy `std::launch::async` when using `std::async` to explicitly spawn new threads

OpenMP

Introduction to OpenMP

- OpenMP is an API for explicit shared-memory parallelism
- Supported by most compilers (gcc, icc, msvc, clang)
- Utilizes OS threading capabilities (e.g. Pthreads)
- Fully documented in the specification (see `http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf`)
- Comprised of three programming layer components

1. Compiler Directives
   - Spawning parallel regions
   - Distributing loop iterations across threads
   - Synchronization
   - ...
2. Runtime Library Routines
   - Setting/Querying the number of current threads
   - Querying thread-id's and wall-clock time
   - ...
3. Environment Variables
   - Setting number of threads
   - Binding threads to processors
   - ...

Directives

Format

```
#pragma omp <directive name> <{clause, ...}>
```

- `#pragma omp`
  Required for all OpenMP C/C++ directives
- `directive name`
  A valid OpenMP directive
- `{clause, ...}`
  Optional. Clauses can be in any order
- Most OpenMP constructs applay to a structured block

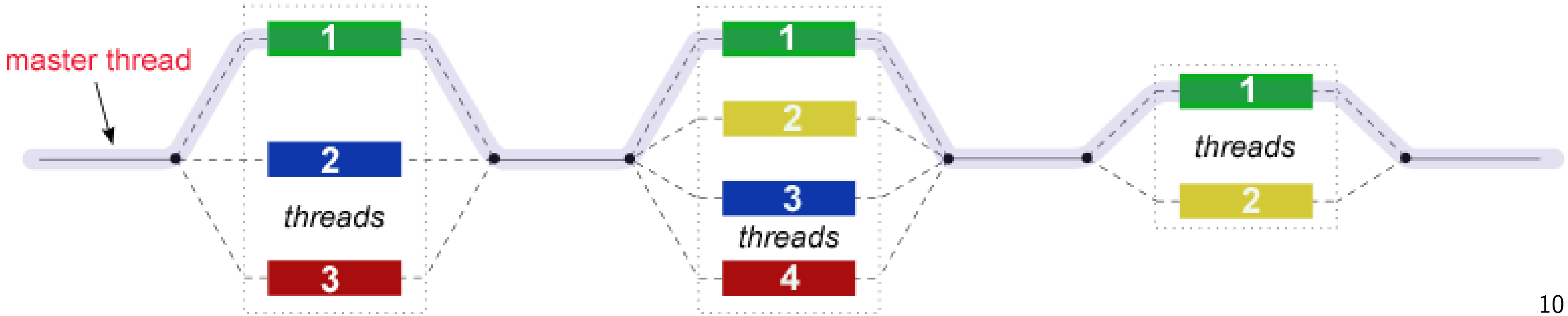
Example

```
#pragma omp parallel default(shared) private(i)
```

Parallel Region

```
#pragma omp parallel <{clause, ...}>
```

- A block of code that will be executed by multiple threads
- Number of threads defined by #cpu, clauses, or env. variables
- The reaching thread (0) creates a team of N threads (1, ..., N-1)
- At the end of a block there is an implicit join (barrier)
- There may be nested parallel regions



Parallel Region - Example

- omp_get_thread_num() returns the current thread number
- omp_get_num_threads() returns the number of threads

```
int main(int argc, char** argv) {

  #pragma omp parallel
  {
    printf("Hello World from thread %d\n", omp_get_thread_num());

    // only executed by main thread
    if (omp_get_thread_num() == 0)
      printf("Number of threads is %d\n", omp_get_num_threads());
  }
  return 0;
}
```

```
./hello_world
Hello World from thread 1
Hello World from thread 0
Number of threads is 3
Hello World from thread 2
```

Parallel Region - Clauses

- `if ( <scalar expression> )`
  only executed multithreaded if scalar expr. evaluates to non-zero
- `private ( <list> )`
  each thread gets a copy of variables in a comma separated list (variables might be uninitialized)
- `firstprivate/lastprivate ( <list> )`
  same as `private`, but value is copied at the entry/exit
- `shared ( <list> )`
  variables in `list` are shared (no elements of structs or arrays)
- `default ( shared | none )`
  sets the default behaviour (`none` means that data sharing needs to be explicit)
- `reduction ( <operator: list> )`
  reduction operation and associated operand
- `num_threads ( <integer expression> )`
  sets the number of threads for the parallel region
- ...

`for` Directive

```
#pragma omp for <{clause, ...}>
```

- Worksharing construct to execute the immediately following loop by a team of threads
- Assumes that a parallel region has already been initiated
- There is an implicit barrier at the end of the loop
- Clauses:
  - `schedule ( static|dynamic|guided|runtime|auto )`
    sets the scheduling behaviour (see next slide)
  - `nowait`
    threads do not synchronize after the parallel loop
  - `ordered`
    iterations must have the same order as in a serial program
  - `collapse`
    specifies the number of (nested) loops that shall be collapsed into a larger iteration space
  - `private, firstprivate...`

`schedule` clause

- `schedule (static, `*`chunk_size`*`)`
  The iterations are divided into chunks of size *chunk_size* and assigned to the threads in round-robin fashion. When no chunk size is specified, the iterations are equally divided (at most one iteration per thread).
- `schedule (dynamic, `*`chunk_size`*`)`
  The iterations are distributed to threads in chunks as the executing threads request them. Each thread executes a chunk of iterations, then requests another chunk, until no chunks remain. Each chunk contains *chunk_size* iterations, except for the last chunk. If no chunk size is specified, it defaults to 1.
- `schdule (guided, `*`chunk_size`*`)`
  Similar to `dynamic`, but...
  At the beginning the size of each chunk is proportional to the number of unassigned iterations divided by the number of threads, decreasing to 1 (or `chunk_size`, if specified).
- `schedule (auto)`
  The scheduling decision is given to the compiler/runtime system.
- `schedule (runtime)`
  The scheduling decision is deferred until run time, the schedule and chunk size are taken from internal control variables.

for Directive - Example

```c
#include <omp.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>

const char *colored_digit[] = {
 "\e[1;30;1m0", "\e[1;31;1m1", "\e[1;32;1m2", "\e[1;33;1m3", "\e[1;34;1m4", "\e[1;35;1m5", "\e[1;36;1m6", "\e[1;
};

int main(int argc, char** argv) {
  unsigned int x_size = 80;
  unsigned int y_size = 40;
  unsigned long str_len = strlen (colored_digit [0]);
  char *string_2D = (char*)malloc(x_size * y_size * str_len + y_size);

  #pragma omp parallel for schedule(runtime)
  for (unsigned long i = 0; i < y_size; i++) {
    for (unsigned int j = 0; j < x_size; j++) {
      memcpy(string_2D + ( i * x_size * str_len + i ) + (j * str_len), colored_digit[omp_get_thread_num()], str_
    }
  }

```

for Directive - Example (Cont.)

```
OMP_NUM_THREADS=4 OMP_SCHEDULE="STATIC" ./scheduling
```

```
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
11111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111
22222222222222222222222222222222222222222222222222222222222222222222222222222222
22222222222222222222222222222222222222222222222222222222222222222222222222222222
22222222222222222222222222222222222222222222222222222222222222222222222222222222
22222222222222222222222222222222222222222222222222222222222222222222222222222222
22222222222222222222222222222222222222222222222222222222222222222222222222222222
22222222222222222222222222222222222222222222222222222222222222222222222222222222
22222222222222222222222222222222222222222222222222222222222222222222222222222222
22222222222222222222222222222222222222222222222222222222222222222222222222222222
22222222222222222222222222222222222222222222222222222222222222222222222222222222
22222222222222222222222222222222222222222222222222222222222222222222222222222222
33333333333333333333333333333333333333333333333333333333333333333333333333333333
33333333333333333333333333333333333333333333333333333333333333333333333333333333
33333333333333333333333333333333333333333333333333333333333333333333333333333333
33333333333333333333333333333333333333333333333333333333333333333333333333333333
33333333333333333333333333333333333333333333333333333333333333333333333333333333
33333333333333333333333333333333333333333333333333333333333333333333333333333333
33333333333333333333333333333333333333333333333333333333333333333333333333333333
33333333333333333333333333333333333333333333333333333333333333333333333333333333
```

for Directive - Example (Cont.)

```
OMP_NUM_THREADS=4 OMP_SCHEDULE="STATIC,2" ./scheduling
```

```
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
11111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111
22222222222222222222222222222222222222222222222222222222222222222222222222222222
22222222222222222222222222222222222222222222222222222222222222222222222222222222
33333333333333333333333333333333333333333333333333333333333333333333333333333333
33333333333333333333333333333333333333333333333333333333333333333333333333333333
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
11111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111
22222222222222222222222222222222222222222222222222222222222222222222222222222222
22222222222222222222222222222222222222222222222222222222222222222222222222222222
33333333333333333333333333333333333333333333333333333333333333333333333333333333
33333333333333333333333333333333333333333333333333333333333333333333333333333333
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
11111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111
22222222222222222222222222222222222222222222222222222222222222222222222222222222
22222222222222222222222222222222222222222222222222222222222222222222222222222222
33333333333333333333333333333333333333333333333333333333333333333333333333333333
33333333333333333333333333333333333333333333333333333333333333333333333333333333
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
11111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111
22222222222222222222222222222222222222222222222222222222222222222222222222222222
22222222222222222222222222222222222222222222222222222222222222222222222222222222
33333333333333333333333333333333333333333333333333333333333333333333333333333333
33333333333333333333333333333333333333333333333333333333333333333333333333333333
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
11111111111111111111111111111111111111111111111111111111111111111111111111111111
11111111111111111111111111111111111111111111111111111111111111111111111111111111
22222222222222222222222222222222222222222222222222222222222222222222222222222222
22222222222222222222222222222222222222222222222222222222222222222222222222222222
```

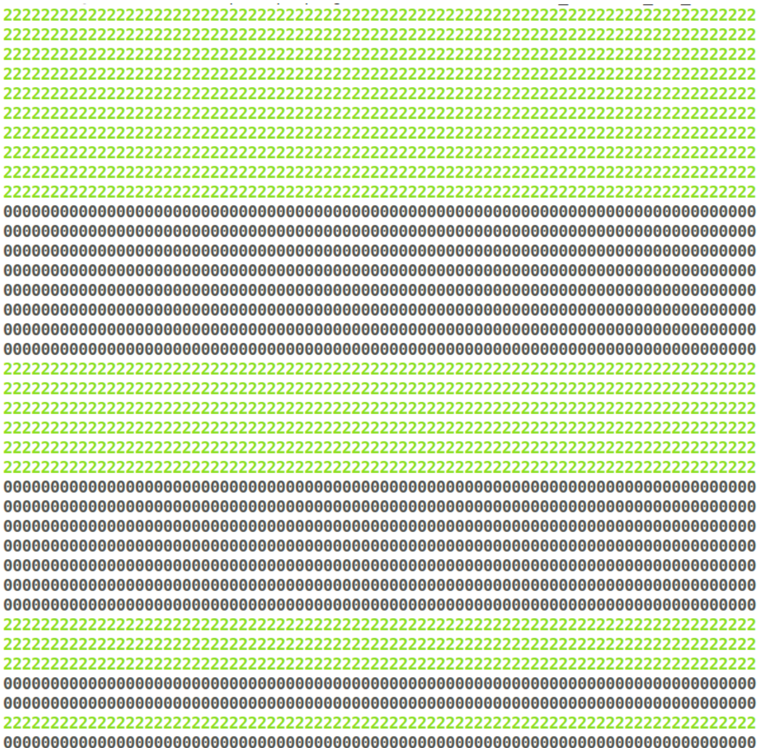
for Directive - Example (Cont.)

OMP_NUM_THREADS=4 OMP_SCHEDULE="DYNAMIC" ./scheduling

```
00000000000000000000000000000000000000000000000000000000000000000000000000000000
11111111111111111111111111111111111111111111111111111111111111111111111111111111
00000000000000000000000000000000000000000000000000000000000000000000000000000000
11111111111111111111111111111111111111111111111111111111111111111111111111111111
00000000000000000000000000000000000000000000000000000000000000000000000000000000
11111111111111111111111111111111111111111111111111111111111111111111111111111111
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
11111111111111111111111111111111111111111111111111111111111111111111111111111111
00000000000000000000000000000000000000000000000000000000000000000000000000000000
11111111111111111111111111111111111111111111111111111111111111111111111111111111
00000000000000000000000000000000000000000000000000000000000000000000000000000000
11111111111111111111111111111111111111111111111111111111111111111111111111111111
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
11111111111111111111111111111111111111111111111111111111111111111111111111111111
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
11111111111111111111111111111111111111111111111111111111111111111111111111111111
00000000000000000000000000000000000000000000000000000000000000000000000000000000
11111111111111111111111111111111111111111111111111111111111111111111111111111111
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
11111111111111111111111111111111111111111111111111111111111111111111111111111111
00000000000000000000000000000000000000000000000000000000000000000000000000000000
11111111111111111111111111111111111111111111111111111111111111111111111111111111
00000000000000000000000000000000000000000000000000000000000000000000000000000000
11111111111111111111111111111111111111111111111111111111111111111111111111111111
00000000000000000000000000000000000000000000000000000000000000000000000000000000
11111111111111111111111111111111111111111111111111111111111111111111111111111111
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
11111111111111111111111111111111111111111111111111111111111111111111111111111111
00000000000000000000000000000000000000000000000000000000000000000000000000000000
11111111111111111111111111111111111111111111111111111111111111111111111111111111
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
```

for Directive - Example (Cont.)

```
OMP_NUM_THREADS=4 OMP_SCHEDULE="GUIDED" ./scheduling
```



How do you do it for Mandelbrot?

```
void mandelbrot_draw(int x_resolution, int y_resolution, int max_iter,
                        double view_x0, double view_x1, double view_y0, double view_y1,
                        double x_stepsize, double y_stepsize,
                        int palette_shift, unsigned char (*image)[x_resolution][3],
                                                 int num_threads) {
        // ....

        #pragma omp parallel num_threads(num_threads)
        {
                #pragma omp for schedule(dynamic)
                for (int i = 0; i < y_resolution; i++)
                {
                        for (int j = 0; j < x_resolution; j++)
                        {
                                //pixel calculation ...
                        }
                }
        }
{
```

Assignment 4 - Edge detection (OpenMP)

Assignment 3 - Edge detection (OpenMP)

- You have two weeks time for this assignment
- Use OpenMP
- no valgrind, helgrind, #threads...
- The speedup with 32 cores must be at least 16
- Consider:
  - Previous strategies may apply here

Assignment 4 - Edge detection (OpenMP) - x_gradient()

```
template <typename SrcView, typename DstView>
void x_gradient(const SrcView &src, const DstView &dst, int num_threads)
{
    typedef typename channel_type<DstView>::type dst_channel_t;

    for (int y = 0; y < src.height(); ++y)
    {
        typename SrcView::x_iterator src_it = src.row_begin(y);
        typename DstView::x_iterator dst_it = dst.row_begin(y);

        for (int x = 1; x < src.width() - 1; ++x)
        {
            static_transform(src_it[x - 1], src_it[x + 1], dst_it[x],
                             halfdiff_cast_channels<dst_channel_t>());
        }
    }
}
```

Assignment 4 - Edge detection (OpenMP) - Provided Files

- Makefile
  - contains rules to build executables
  - available targets: parallel, sequential, all (default), clean
  - 'mode=debug make [target]' to build debug version, use 'make clean' before
- main.c
  - main function - argument handling + file handling + call `x_luminosity_gradient()`
  - `x_luminosity_gradient()` calls `x_gradient()`
  - you implement the parallel version of `x_gradient()`
- x_gradient.h
  - Header file for `x_luminosity_gradient()`
- x_gradient_seq.h
  - Sequential version of `x_gradient()`
- student/x_gradient_par.h
  - Implement the parallel version in this file
- unit_test.c
  - The unit tests that execute both the serial and parallel version to compare results.

Assignment 4 - Edge detection (OpenMP) - Compilation and execution

- Compilation
  - You need to install libjpeg, boost and boost/gil
  - `make [all] [sequential] [parallel] [unit_test]`
  - You implement your solution in a header file
  - You have to `make clean` every time and `make` again
- Execution
  - `./student/x_gradient_seq`
  - `./student/x_gradient_par -t 4 -f tum.jpg`
  - `./student/unit_test`